Deliverable D6.2
Technical Report on the Integration of MonB5G Technologies in the Network Architecture

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	D6.2 – Technical Report on the Integration of MonB5G Technologies in the Network Architecture		
Work Package	WP6		
Contractual due date	M42	**Actual submission date**	31 May 2023
Nature	Technical Report	**Dissemination Level**	Public
Lead Beneficiary	IQU		
Responsible Author	Engin Zeydan (CTTC), 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), Sihem Cherrared (OR-FR), Robert Kołakowski (ORA-PL), Rafał Tępiński (ORA-PL), Sławomir Kukliński (ORA-PL), Georgia Pantelide (eBOS), Karim Boutiba (EUR), Sofiane Messaoudi (EUR), Adlen Ksentini (EUR), Ashima Chawla (LMI), Anne Marie Cristina Bosneag (LMI), Vasiliki Vlahodimitropoulou (OTE), Cédric Morin (BCOM), Eric Gatel (BCOM), Cao-Thanh Phan (BCOM)

Disclaimer

Copyright message

TABLE OF CONTENTS

List of Figures 5
List of Tables 8
List of Acronyms 9
1 Executive Summary 14
2 Introduction 17
2.1 Objectives of the Performance Evaluation of the MonB5G Distributed Architecture and Enablers 17
2.2 Deliverable Overview and Structure 17
3 Infrastructure Setup, Testing and Deployment of Software Components in MonB5G Architecture 19
3.1 PoCs Testbed and Summary of Experimental Scenarios 19
3.1.1 Scenario Description and Mapping of PoC#1 Solutions 19
3.1.2 Scenario Descriptions and Mapping of PoC#2 Solutions 21
3.1.3 MonB5G Architecture in PoC Testbed 24
3.2 Infrastructure Details of the Testbed and Interconnection 25
3.2.1 Testbed for PoC-1 25
3.2.2 Testbed for PoC-2 29
3.3 Application for PoC-1 Setup 32
3.3.1 Video Streaming Overview 32
3.3.2 Example Video 32
3.3.3 The RTSP Protocol 33
3.3.4 Emulating High Demanding VR Video Streaming with Transcoding 33
3.4 KPIs for the POCs and Summary of Integrated Solutions 34
3.4.1 KPI Mapping to Experimental Scenarios in PoC-1 34
3.4.2 KPI Mapping to Experimental Scenarios in PoC-2 41
4 Evaluation of KPIs of MonB5G Components in the Experimental Framework for PoC-1 47
4.1 Recap on the Deployment of MonB5G Monitoring System in PoC-1 47
4.1.1 Multi-gNodeB RAN setup 47
4.1.2 Amarisoft Remote UE 50
4.1.3 Monitoring Sampling Functions 52
4.1.4 Grafana Visualization 53
4.2 Datasets from Experimental Trials 54

4.2.1 Federated Learning Dataset 54
4.2.2 5G RTSP Video Streaming Dataset for Anomaly Detection 55
4.3 Evaluating KPIs of MonB5G Solutions at the PoC-1 Scenario 1 Testbed 57
4.3.1 Traffic Prediction with Context Awareness 57
4.3.2 FL Predictor 60
4.3.3 Slice KPI Prediction with Interpretable Multivariate Anomaly Detection 77
4.3.4 LSTM-Based Anomaly Detection 82
4.4 Evaluating KPIs of MonB5G Solutions at the PoC-1 Scenario 2 Testbed 86
4.4.1 Slice Admission control (DE) Based on traffic Prediction 86
4.4.2 A multi-agent learning for distribution resource allocation in the RAN domain 89
4.4.3 RL-based slice admission control 100
5 Evaluating KPIs of the MonB5G Components over the Experimental Framework for PoC-2 102
5.1 Recap on the deployment scenarios in PoC-2 102
5.1.1 MonB5G PoC-2 Scenario 1: mMTC attack 102
5.1.2 MonB5G PoC-2 Scenario 2: Federated Learning attacks 102
5.1.3 MonB5G PoC-2 Scenario 3: aLTEr attack 103
5.2 Datasets from experimental trials 109
5.2.1 MonB5G PoC-2 Scenario 2: FL attack 109
5.2.2 MonB5G PoC-2 Scenario 3: aLTEr attack 112
5.3 Evaluating KPIs of MonB5G Solutions at the PoC-2 Scenario 1 Testbed: mMTC *ATTACK* 113
5.4 Evaluating KPIs of MonB5G Solutions at the PoC-2 Scenario 2 Testbed: FL Attack 119
5.5 Evaluating KPIs of MonB5G Solutions at the PoC-2 Scenario 3 Testbed: aLTEr attack 124
6 Lessons Learnt from Experiments 127
7 Conclusions 129
8 Appendixes 130
9 References 146

List of Figures

Figure 3-1: MonB5G architecture mapped to PoC. 24
Figure 3-2: 5G hardware components. 26
Figure 3-3: PoC1 Architecture components 27
Figure 3-4: Outdoor and Indoor Radio Units 29
Figure 3-5: Global architecture of the 5G Facility 31
Figure 3-6: VR Video Streaming infrastructure and PoC #1 setup. 32
Figure 3-7: Integration of AEs and DEs with MS and ACTs for various solutions in PoC#1 35
Figure 3-8: Integration of AEs and DEs with MS and ACTs for various solutions 38
Figure 3-9: Integration Solutions for DDOS Attack 42
Figure 3-10: Integrating solution for Poisoning attack 44
Figure 3-11: MonB5G security orchestrator integration for the aLTEr attack scenario 45
Figure 4-1: Deploying edge and cloud cluster in a multi-domain environment. 47
Figure 4-2: Setting up a 5G network with 2 gNBs in Amarisoft. 48
Figure 4-3: Schematic for the integration Callbox Ultimate – Simbox – Callbox Mini. 49
Figure 4-4: Components and interfaces for Callbox Ultimate – Simbox – Callbox Mini. 50
Figure 4-5: Remote UE setup with N UEs emulated by Simbox but run at remoteue VM. 50
Figure 4-6: Remote UE deployment for Poc1.1. 51
Figure 4-7: Displaying video clients for different UEs at remoteue. 51
Figure 4-8: Monitoring System Sampling Functions at edge sites. 52
Figure 4-9: Flowchart of the Amarisoft-gnb SF. 52
Figure 4-10. Flowchart of the pod-infra-metrics SF. 53
Figure 4-11: Grafana dashboard for PoC1.1. 54
Figure 4-12: Sample dataset 55
Figure 4-13: Traffic pattern throughout a day in terms of number of users per eNB sector area. 56
Figure 4-14: Metrics shown in Grafana for the dataset 56
Figure 4-15: Features and sample of the dataset 57
Figure 4-16: Quality of Prediction of ECATP compared with state-of-the-art predictors. 59
Figure 4-17: Probability of SLA violations in ECATP compared with state-of-the-art predictors. 60
Figure 4-18: Performance of ECATP compared with state-of-the-art predictors. 60
Figure 4-19: Congestion in the VR video streaming server due to overload 61
Figure 4-20: FL deployment sites, VR video streaming servers and clients 62
Figure 4-21: Different patterns of number of VR-streaming users for each sites used for experiments 64
Figure 4-22: FL training steps 69
Figure 4-23: Sidecar container on aggregation server to collect log and visualize 71
Figure 4-24: VR video streaming server outbound traffic versus number of VR video streaming clients 72
Figure 4-25: VR video streaming Server CPU load versus number of VR video streaming clients 73
Figure 4-26: Pair Plot. 74
Figure 4-27: Power Consumption at container at UP versus N 75
Figure 4-28: FL Training phase (a) average NMSE (b) average computation time 76
Figure 4-29: Analytics Engine reference Architecture 78
Figure 4-30: Prediction Comparison among Different Methods 79

Figure 4-31: Interpretable Anomaly Detection depicting contributing factors. 80
Figure 4-32: Cloud Implementation of the AE deployment 80
Figure 4-33: AE deployed and running in the testbed (CTTC) 81
Figure 4-34: Docker deployed and running. 81
Figure 4-35: Multivariate Anomalies been detected by the model (CPU, O, R at same time) 82
Figure 4-36: The metrics collected during non-anomalous network operation. 84
Figure 4-37: Non-anomalous signal traffic 85
Figure 4-38: Signal with anomalies 85
Figure 4-39: The evaluation approach of TASAC enabler 87
Figure 4-40: Comparison in terms of: (a) obtained reward (b) resources 88
Figure 4-41: Comparisons in terms of: (a) resources (b) violations ratio 89
Figure 4-42: Federated RAN slicing architecture 90
Figure 4-43: Testbed architecture 90
Figure 4-44: MonB5G MS and monitored dataset. 92
Figure 4-45: Decision Agents 92
Figure 4-46: Multi-agent setup 93
Figure 4-47: Evaluation Scenario 94
Figure 4-48: GUI: Graphical User Interface 95
Figure 4-49: Communication Overhead 96
Figure 4-50: The convergence performance of agents A and agent B in first gNB 97
Figure 4-51: The convergence performance of agents A and agent B in the second gNB. 97
Figure 4-52: Allocation gap performance for agents A and agent B in gNB1. 98
Figure 4-53: Allocation gap performance for agents A and agent B in gNB2. 98
Figure 4-54: The CDF of experienced allocation gap (in Mbps) within network slices. 99
Figure 4-55: Performance evaluation for different network loads. 100
Figure 4-56: KPI evaluation of PreBAC for the current use case 101
Figure 4-57: KPI evaluation of the baseline solution for the current use case 101
Figure 5-1: Overview of TQFL Framework. 103
Figure 5-2: Mapping between the implementation and the architecture. 104
Figure 5-3: The actual platform implementation for the scenario aLTEr 105
Figure 5-4: The security of data plane is continuously monitored and the aLTEr attack is ongoing. 106
Figure 5-5: Raw data at the N6 interface are collected and sent the MS via a VXLAN tunnel. 106
Figure 5-6: Meaningful logs of DNS transactions at the N6 interface received by the AE 107
Figure 5-7: The AE raises an event indicating the aLTEr attack is occurring for the UE. 107
Figure 5-8: The ACT is executing the action plan provided by the DE. 108
Figure 5-9: Effective protection of DNS transactions using TLS to eradicate the attack aLTEr 109
Figure 5-10: Test platform and technological components 113
Figure 5-11: AE's components. 114
Figure 5-12: Results of the detection algorithm over normal traffic. 115
Figure 5-13: Results of the detection algorithm over abnormal traffic. 115
Figure 5-14: The result of the statics method over abnormal traffic. 116
Figure 5-15: The result of the statics method over normal traffic 116
Figure 5-16: Flowchart of the DE's components. 118

Figure 5-17: Overview of TQDL framework 120
Figure 5-18: LDA + K-means for different number of malicious nodes (ADAM optimizer). 121
Figure 5-19: LDA + KNN for different number of malicious nodes (ADAM optimizer) 122
Figure 5-20: LDA + K-means for different number of malicious nodes (SGD optimizer) 122
Figure 5-21: LDA + KNN for different number of malicious nodes (SGD optimizer). 122
Figure 5-22: PCA + K-means for different number of malicious nodes (ADAM optimizer) 123
Figure 5-23: PCA + KNN for different number of malicious nodes (ADAM optimizer) 123
Figure 5-24: PCA + K-means for different number of malicious nodes (SGD optimizer). 123
Figure 5-25: PCA + KNN for different number of malicious nodes (SGD optimizer) 124
Figure 5-26: The information path defined to measure the attack detection time 125
Figure 5-27: The information path defined to measure the attack response time 125
Figure 5-28: MTTD and MTTR measured over 100 attacks and mitigations. 126

List of Tables

Table 1: Objective mapping to PoC #2 Experimental Scenarios 21
Table 2: Objective mapping to PoC #2 Experimental Scenarios 24
Table 3. Original video file information.....33
Table 4: Target KPIs versus experimental results for solutions in ES1.1.....36
Table 5: Experimental results versus target KPIs for solutions in ES1.2 39
Table 6: Experimental results versus target KPIs for solutions in ES2.1: DDoS attack 43
Table 7: Experimental results versus target KPIs for solutions in ES2.2 Poisoning attack.....44
Table 8; Experimental results versus target KPIs for solutions in aLTEr attack scenario.....46
Table 9: Dataset features and output used in FL.....69
Table 10: Parameters of regression analysis model.....72
Table 11: Monitoring overhead comparison between centralized solution and FL-based algorithms 76
Table 12: Energy consumption comparisons.....77
Table 13: The accuracy of the statistical model considering different Duration values.....117
Table 14: Impact of the AE Detection threshold.....117
Table 15: Impact of the DE Detection threshold 119

List of Acronyms

Acronym	*Description*
3GPP	*Third Generation Partnership Project*
5GC	*5G Core*
ACT	*Actuator*
AD	*Anomaly Detection*
AE	Analytic Engine
AI	Artificial Intelligence
AMF	Access and Mobility Management Function
API	Application Programming Interface
AR	Augmented Reality
B5G	Beyond-5G
CCI	Cloud Continuum Infrastructure
CN	Core Network
CNF	Cloud Native function
CPU	Central Processing Unit
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
DNS	Domain Name System
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
ECA	Event Condition Action
ENI	Experiential Networked Intelligence
ES	Experimental Scenario
ETSI	European Telecommunications Standards Institute
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
IoT	Internet of Things
ITU	International Telecommunication Union
K8	Kubernetes
KNN	k-nearest neighbours
KPI	Key Performance Indicator
LCM	Lifecycle Management
LDA	Linear Discriminant Analysis
LSTM	Long-Short Term Memory
LXC	Linux Containers
ML	Machine Learning
MANO	Management and Orchestration
MaaS	Management as a Service
MAN-F	Management Function
MCC	Mobile Country Code
MEC	Multi-access Edge Computing

MECapp	MEC applications
MEO	MEC Orchestrator
MEP	MEC Edge Platform
MGEN	Multi-Generator
ML	Machine Learning
MNC	Mobile Network Code
MNO	Mobile Network Operator
mMTC	Massive Machine Type Communications
MS	Monitoring System
MSE	Mean Squared Error
MVAD	Multi-variate time series anomaly detection
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

OPEX	Operational Expenditure
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
RAM	Random Access Memory
RAN	Radio Access Network
RBAC	Role-Based Access Control
RNN	Recurrent Neural Networks
RTSP	Real Time Streaming Protocol
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
ToD	Time of the Day
TQFL	Trust deep Q-learning Federated Learning
UC	Use Case

UDM	Unified Data Management
UE	User Equipment
UV	Univariate
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
VR	Virtual Reality
XAI	interpretable Artificial Intelligence
ZSM	Zero-touch network and Service Management

1 Executive Summary

MonB5G aims to provide zero-touch management and orchestration to support network slicing at scale for 5G LTE and beyond. The vision is to build a decentralized zero-touch management and orchestration system that supports network slicing for Beyond-5G (B5G) networks. This deliverable presents and quantifies the main outcomes of the MonB5G project towards achieving the vision. The scope of this deliverable focuses on the performance validation of the components of the MonB5G concept developed in previous deliverables (WP2, WP3, WP4, and WP5) (architecture, proposed algorithms, and the state-of-the-art Artificial Intelligence (AI) algorithms) and how they are integrated into the testbed to perform the envisioned control-loop capabilities. The methodology for performing this validation requires the following:

1. The mechanisms that serve as the basis for performance comparison.
2. Representative use-cases of Beyond-5G networks with their representative workloads for the Proof-of-Concepts (PoCs) and their respective Experimental Scenarios (ES).
3. The Key Performance Indicators (KPIs) for each PoC and ES, previously described in WP2.

To provide context to the performance evaluation methodology, this deliverable also details PoC testbeds, namely the specifics of the hardware, the deployment process of use cases, the baseline mechanisms used for reference, and the KPIs used for comparison, as well as how they are measured and monitored.

After a review of MonB5G architecture and components, that is, the Monitoring Systems (MS), Analytics Engines (AE), Decision Engines (DE) and Actuators (ACT) in PoC testbed, performance validations using these MonB5G components are shown. The contributions in this deliverable consist of presentation of the KPIs associated with the previously studied MonB5G enablers, as well as details of the deployment on the testbed, that go beyond D6.1. Appropriate baseline scenarios are used as a reference for performance comparison.

The main objective of PoC-1 scenario 1 is to assess the data-driven management systems in a multi-domain scenario with respect to their ability to guarantee the stringent end-to-end service level agreement (SLA) of the B5G applications. The automated zero-touch service management and multiple redundancy mechanisms aims practically zero downtime due to the critical, high-availability. The PoC-1 scenario 2 demonstration shows how MonB5G mechanisms respond to local performance issues in different technological domains as well as to changes in traffic patterns in different timescales. Finally, there is a special focus on data-driven mechanisms for radio resource management to optimize the RAN sub-slice.

PoC-2 scenario 1 proposes a zero-touch security management solution that addresses the challenges of in-slice DDoS attack detection and mitigation, using mMTC (massive machine type communication) network slices as an example. The critical challenge addressed in this work is the detection of a DDoS attack originated from a compromised set of MTC devices within a network slice. PoC-2 scenario 2 the ZSM concept in B5G aims to automate the management and orchestration of running network slices. This scenario addresses the robustness of FL algorithms against attacks, such as the poisoning attack. PoC-2 scenario 3 proposes a zero-touch security management solution with a local security orchestrator, that addresses the challenges of aLTEr attack detection and mitigation. To detect the attacks, AI/ML (Machine Learning) algorithms have been studied to detect the aLTEr attack executed on AE. The mitigation step executed by DE is to update the security policy on the firewall to block the private DNS address of the UE (User Equipment).

In PoC-1, D6.2 evaluates the KPIs of MonB5G solutions in the PoC-1 testbed, including various scenarios such as *FL predictor, Slice KPI prediction with Interpretable and multivariate anomaly detection and LSTM-Based Anomaly Detection and ECATP (Enhanced Context-Aware Traffic Predictor)*, *Slice Admission Control based on traffic prediction* and *resource allocation in the RAN domain using a multi-agent deep reinforcement learning (DRL) approach*. In PoC-2, D6.2 evaluates the *DDoS attack mitigation, malicious users blacklisting, secure FL, trust deep Q learning, incident detection and incident response of MonB5G security orchestrator* techniques. Performance validation of MonB5G enablers and their compliance with KPIs defined in WP2. The analysed KPIs are of paramount importance as they demonstrate the benefits of the MonB5G architecture, components and algorithms compared to the baseline mechanisms and validate the contributions of MonB5G for what they are worth. More specifically, the key achievements covered by this deliverable are the following:

- *In PoC-1 scenario 1,* the solutions aimed to enhance network performance, reduce reaction time to malfunctions, optimize resource allocation, and improve the efficiency of service management and orchestration. Together with data-driven management systems, the monitoring overhead is reduced **by more than 11 times,** signalling and monitoring overhead **by a factor of 10** compared to the centralized SLA-constrained algorithm**.** This resulted in **a 10-fold improvement** in the energy efficiency. By distributing MonB5G architecture components across multiple sites, the l**ocal elements handled a greater portion of service management** and **slice lifecycle tasks**. This reduced dependency on centralized systems and improved performance and response times. **OPEX reduction due to service management automation** is achieved by automating the network service management through distributed AI and reducing monitoring overhead and energy consumption. Automated ZSM techniques and proactive anomaly detection across a network slice show approximately **20.42% faster convergence** on training and **reduced reaction time to malfunctions**.
- *In PoC-1 scenario 2,* solutions aimed to reduce SLA violations, optimize resource management, improve performance isolation, and reduce monitoring and management overhead. These advancements contributed to enhancing the cost-effectiveness of network slice management and service provisioning. Slice admission control techniques reduced the number of **SLA performance violations by more than 30%** by reducing the number of slices deployed, enabling better use of network resources through automatic slice admission. A **three-fold increase** in **user service request admission** during resource shortages and **potential OPEX reductions** were also achieved. The data-driven radio resource management mechanisms achieved **an average 3.5% improvement in SLA violations,** while **minimizing radio resource utilization**. In addition, **the time required to manage RAN resources** for network slices was reduced, ensuring **fair resource access** and **slice performance isolation**. Compared to centralized management system, **the management overhead and the amount of monitored data were reduced by up to 25%.**
- In PoC-2 scenario 1, solutions focused on detecting and mitigating DDoS attacks, ensuring high availability of network slices, and achieving fast attack identification and remediation while maintaining a low false positive rate. Zero-touch Security Management chieved anomaly detection time for the **first occurrence of an attack to be equal to the attack duration plus 10**

milliseconds and for repetitive attacks (UEs that have previously participated in an attack) to be **less than 10 milliseconds while achieving 10 times faster attack/anomaly identification**. The entire procedure of malicious user blacklisting took **less than 10 milliseconds** and the false positive rate for attack classification (incorrectly classifying events as attacks) was **below 4%.**

- *In PoC-2 scenario 2,* solutions focused on securing federated learning against poisoning attacks, detecting and blocking malicious FL clients, and ensuring the robustness and accuracy of the learning process. Secure Federated Learning was able to detect all malicious FL clients, **resulting in a false positive rate of 0%**. Trust Deep Q-Learning Federated Learning **kept the mean squared error (MSE) below 0.4** even with an increasing number of malicious FL clients.
- *In PoC-2 scenario 3,* solutions focused on achieving faster identification of security attacks/anomalies and accelerate attack remediation and reconfiguration in the order of 10s. In the Incident Detection of the MonB5G Security Orchestrator solution, the measured **Mean Time to Detect (MTTD) was approximately 500 milliseconds (ms)**. In the Incident Response of the MonB5G Security Orchestrator solution, an action plan for the deployment of countermeasures is created based on rules and facts from DE, while the ACT executed the plan. The time **to make a decision (mean time) was around 200 ms**, and the measured **Mean Time to Resolve (MTTR) was around 3 seconds (<10 seconds for reconfiguration).**
- **Lessons learned** from the experimental demonstrations are also summarized at the end of the deliverable.

In summary, the MonB5G vision of the creating an intelligent, decentralized, and secure zero-touch management and orchestration framework in B5G networks has been successfully demonstrated through the tangible results obtained of experimental testbed platforms.

2 Introduction

2.1 Objectives of the Performance Evaluation of the MonB5G Distributed Architecture and Enablers

This deliverable presents the evaluation and validation of the KPIs defined as the main scope of contribution for the components of the MonB5G architecture. The functional validation initially achieved in D6.1 is now extended with a performance validation that materializes autonomic control-loops within the framework of B5G communication networks, for which matching experimental testbeds were built and designed.

The vision of MonB5G project is to provide autonomic management with the features envisaged for B5G communication networks. The project aims to achieve this by integrating and validating the MS/AE/DE/ACT components to materialize autonomic control-loops within the framework of B5G 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. The aim is to demonstrate the ability of the integrated MonB5G components to fulfil the KPIs presented in WP2, as part of the MonB5G architectural vision for distributed management. The achievement of this is dependent on the interoperability and integration of the MonB5G components, presented in this deliverable. In this deliverable, functional aspects refer to the features and capabilities of a system and the components that actually provide the service, while performance and deployment aspects refer to how well the system performs and how it is implemented and deployed, as well as the reliability of the network service elements and the deployment frameworks of the MonB5G components.

The goal of this deliverable is to go deeper into the details of the infrastructure used to implement the testbeds, how the PoCs/ES are deployed on the infrastructure, the KPIs that are used for evaluation, the introduction of MonB5G enablers and applications (including VR video streaming) in each of the considered scenarios and then to present the actual evaluation with comparisons.

This document addresses the following project objectives:

- Obj. #1: Distributed management plane to support a massive deployment of network slices.
- Obj. #2: Definition of novel e2e slice KPIs and development of AI-based mechanisms for their accurate prediction from multi-level metrics
- Obj. #3: Data-driven management system based on federated learning.
- Obj. #4: Zero touch network configuration
- Obj. #5: DE decisions tailored to the RAN.
- Obj. #7: AI-driven energy efficient network management

2.2 Deliverable Overview and Structure

This deliverable provides a comprehensive overview of the infrastructure setup, testing, deployment, and evaluation of MonB5G components in both PoC-1 and PoC-2 scenarios, covering various aspects and performance metrics. The main part of the information includes the following sections:

Section 3 focuses on the Infrastructure Setup, Testing, and Deployment of Software Components in the MonB5G Architecture. It covers the PoCs Testbed and Experimental Scenarios recap, hardware details of the testbed and interconnection, as well as the application setup for PoC #1, including video streaming overview, the RTSP protocol, and emulating high-demanding VR video streaming with transcoding. It also includes KPIs for the PoCs and a summary of integrated solutions.

Section 4 discusses the Evaluation of KPIs of MonB5G Components in the Experimental Framework for PoC-1. It provides a recap of the deployment of MonB5G MS in PoC-1, including the multi-gNB RAN setup, Amarisoft Remote UE, monitoring sampling functions, and Grafana visualization. Additionally, it presents the datasets from experimental trials, such as the federated learning dataset and the 5G RTSP video streaming use case for anomaly detection. It then evaluates the KPIs of MonB5G solutions at the PoC-1 Testbed, including various scenarios such as ECATP, FL predictor, Slice KPI prediction with Interpretable Anomaly Detection, Multivariate Anomaly Detection, and LSTM-Based Anomaly Detection. It also evaluates MonB5G DEs in PoC-1, specifically Slice Admission Control based on traffic prediction and resource allocation in the RAN domain using a multi-agent learning approach.

Section 5 focuses on the Evaluation of KPIs of MonB5G Components in the Experimental Framework for PoC-2. It begins with a recap of the deployment of MonB5G MS in PoC-2. Then, it focuses on the Evaluation of KPIs of the MonB5G Components over the Experimental Framework for PoC-2. It begins with a recap of the deployment of MS in PoC-2 and evaluates the KPIs of MonB5G MS in PoC-2. It then evaluates the MonB5G AEs at the PoC-2, including gradient boosting interval regression, dimensionality reduction, and anomaly detection of attacks. Finally, it evaluates the MonB5G DEs in PoC-2, such as in-slice attacks, robustness of learning algorithms in the face of attacks, AE for mMTC attack and aLTEr attack, and response and mitigation of both attacks.

3 Infrastructure Setup, Testing and Deployment of Software Components in MonB5G Architecture

3.1 PoCs Testbed and Summary of Experimental Scenarios

MonB5G has two Proof of Concepts (PoCs) that are being considered to address certain objectives. The first PoC, known as PoC #1, aims to achieve zero-touch network and service management with end-to-end service level agreements (SLAs). The second PoC, known as PoC #2, aims to implement AI-assisted policy-driven security monitoring and enforcement. To achieve these objectives, PoC #1 is considering two experimental scenarios. The first scenario, ES 1.1, focuses on achieving zero-touch multi-domain service management with end-to-end SLAs. The second scenario, ES 1.2, focuses on achieving elastic end-to-end slice management. The purpose of these experimental scenarios is to test the feasibility and effectiveness of the proposed solutions in real-world settings.

Similarly, PoC #2 is also considering three experimental scenarios to achieve its objectives. The first scenario, ES 2.1, aims to detect and mitigate attacks using AI-assisted policy-driven security monitoring. The second scenario, ES 2.2, aims to test the robustness of learning algorithms to attacks and ES2-3 studies aLTEr Attack detection and mitigation. These experimental scenarios are designed to test the effectiveness of the proposed solutions in detecting and mitigating potential security threats in real-world settings.

These experimental scenarios help in identifying and testing the feasibility of proposed MonB5G solutions, which ultimately lead to the development of more robust and effective systems for network and service management and security monitoring. Descriptions of these Experimental Scenarios as well as their mapping to project objectives described in Description of Work, along with used MonB5G Solutions, are described below.

3.1.1 Scenario Description and Mapping of PoC#1 Solutions

Experimental Scenario 1.1: Zero-Touch multi-domain service management with end-to-end SLAs

Objectives: The main objective of this scenario is to assess the data-driven management systems in a multi-domain scenario w.r.t. their ability to guarantee the stringent end-to-end SLA of the Tactile Internet application. Automated zero-touch service management and multiple redundancy mechanisms must ensure practically zero downtime.

Description: In this experimental scenario, multiple NFVIs, hosted in both project testbeds as well as on AWS infrastructure will be combined to demonstrate Zero-Touch service management in complex multi-domain services. In this scenario the Storage, Compute, and RAN functions of the Tactile Internet application will be hosted in different regions under the control of local NFVOs and Decision Engines, while end-to-end SLA must be honoured. Continuous monitoring and closed-loop autonomic control mechanisms, which will be common across regions and testbeds, will ensure self-healing, self-configuring and self-scaling of services, to address faults and performance issues in any of the service technological domains. In this scenario, the following will be demonstrated:

1. **Continuous monitoring** of QoS (e.g., bitrate) and QoE (e.g., video quality) metrics across all technological domains and service functions, by the respective local monitoring engines. **MonB5G**

solutions presented in this deliverable are addressing this description where **continuous monitoring of QoS metrics and local monitoring engines** are tackled by *Monitoring System and design solution*.

2. Model updates are forwarded by the **local monitoring engines** to the centralized Decision Engine where it is assessed **w.r.t. the probability of impacting the SLA**. The s**olutions presented in this deliverable from MonB5G** address the requirement of incorporating **local plans and policies with federated updates** while considering **SLA restrictions** using *Statistical Federated Learning resource predictor and scaling solution***.**
3. At the Decision Engine model updates are **federated** and correlated with predicted events (e.g., Peak Planned Occurrences) and service degradation indicators. **MonB5G solutions presented in this deliverable** address the described scenario by incorporating **federated model updates**. This is accomplished through the utilization of a Statistical Federated Learning resource predictor and scaling solution.
4. **Local plans and policies** are forwarded to local decision engines, who remain responsible for implementing them (e.g., scaling the respective VNFs). The previously mentioned issue is answered by **MonB5G solutions** covered in this deliverable where Fault Detector, LSTM, and graph-based anomaly algorithms **foresee service degradation events**.
5. The "**energy slicing**" subsystem, which is part of the Decision Engines, is responsible for implementing the energy optimization policies, while ensuring that end-to-end SLAs are not affected. **MonB5G solutions** presented in this deliverable propose an **energy efficiency in a slice** with *Statistical Federated Learning based resource predictor and scaling* solution.

Experimental Scenario 1.2: Elastic end-to-end slice management

Objectives: As part of our experiments, we will demonstrate how MonB5G mechanisms react to address local performance issues in multiple technological domains as well as at changes to traffic patterns in various timescales. The ability of these mechanisms to guarantee almost zero latency for Tactile Internet applications by proactively acting (and predicting) spikes in user demand will be assessed. Finally, special emphasis will be on data-driven Radio Resource Management mechanisms to optimize the RAN sub-slice.

Description: As the number of Network Slice Instances (NSIs) increases, the scale and complexity of lifecycle management and slice reconfiguration making 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. In this scenario, in addition to the Tactile Internet application NSI, a massive number of slices will also be emulated in order to demonstrate the following:

1. **Continuous monitoring of each NSSI** by the respective **Monitoring Engine** at appropriate time-scales, to identify performance issues (e.g., deteriorating signal reception). **MonB5G solutions** presented in this deliverable are addressing this description where **continuous monitoring of each NSSI metrics by the Monitoring Engine** are addressed by *Monitoring System and design solution*.
2. Decision Engines at each domain are able to recover local faults, but also **forward model updates** with respect to sub-slice performance to the central Decision Engine. **MonB5G solutions** presented in this deliverable are addressing this description where **models updated with data Federation** for **proactive actions at the Decision Engine** is addressed by *multi-agent DRL-based resource allocation* solution.
3. Sub-slice performance data will be **federated** with traffic pattern predictions **at the Decision Engine**, and **proactive actions** will be taken to prevent missing end-to-end service SLAs. **MonB5G solutions**

presented in this deliverable are addressing this description where **federated solutions at the decision engine** are addressed by multi-agent DRL-based resource allocation solution.

4. **Proactive Actions** are implemented by the **respective domain controller** (e.g., forcing UE handovers at the RAN, or traffic steering to avoid bottleneck points). **MonB5G solutions** presented in this deliverable are addressing this description where **proactive actions by the domain controller** are addressed by *PreBAC solution*.

Mapping of PoC #1 Experimental Scenarios to project objectives is presented in Table 1.

Table 1: Objective mapping to PoC #2 Experimental Scenarios

Objectives	ES 1.1	ES 1.2
Obj. #1: Distributed management plane to support massive deployment of network slices	X	
Obj. #2: Definition of novel e2e slice KPIs and development of AI-based mechanisms for their accurate prediction from multi-level metrics	X	X
Obj. #3: Data-driven management system based on federated learning		X
Obj. #4: Zero touch network configuration	X	X
Obj. #5: DE decisions tailored to the RAN		X
Obj. #7: AI-driven energy efficient network management	X	

3.1.2 Scenario Descriptions and Mapping of PoC#2 Solutions

Experimental Scenario 2.1: Attack detection and mitigation

Objectives: In this scenario, we propose a Zero-touch Security Management solution that addresses the challenges of in-slice DDoS attack detection and mitigation, considering the case of mMTC network slices. Generally, this type of attack targets the 5G CN elements shared among the network slices. The proposed ZSM solution relies on a closed-control loop composed of a triplet (Monitoring System - MS, Analytical Engine - AE, and Decision Engine – DE) that interacts with the 5G CN in order to detect attacks and automatically react by mitigating the attacks. The critical challenge addressed in this work is how to detect a DDoS attack initiated by a compromised set of MTC devices inside a network slice. Indeed, there is no available traces or dataset that reproduce abnormal traffic in 5G, unlike other types of networks where many datasets are available. Besides, it is very challenging to detect during an event if there is an attack and what are the involved devices in the attack.

Description: In this scenario, an attack is generated that does not follow the distribution specified by 3GPP (I.e., beta 3,4 distribution). The MonB5G solution should detect and mitigate the attack.

MonB5G solutions: The scenario leverages the closed-control loop components (MS, AE, and DE) that interact and protect the shared sub-slice components (5G CN and gNBs) against DDoS attacks. Here, the focus is on protecting AMF as it is the entry point of the 5G CN and treats all the Attach Requests coming from the different gNB under its control. The Monitoring System (MS) collects information from the AMF, while the

Analytical Engine (AE) uses ML to predict attacks, and Decision Engine (DE) reacts to the alert sent by AE by acting on AMF (block and blacklist UE). The control-control loop runs as software and can be co-located with the orchestrator managing the life cycle of the shared sub-slice. It should be noted that AMF, via an Element Manager (EM), exposes API for an orchestrator or a manager to extract and monitor information on the AMF's functioning or to change the configuration of the latter. In the proposed framework, MS monitors the Attach Requests received by the AMF, while DE requests AMF to send Registration Reject to suspected devices. 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 Unified Data Management (UDM) if a device is blacklisted or not. As a reminder, 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. Meanwhile, MS, via the EM/AMF API, monitors the Attach Requests received by AMF. MS filters the data to extract 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) 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.

Experimental Scenario 2.2: robustness of learning algorithms to attacks

Objectives: The Zero Touch Management (ZSM) concept in 5G and Beyond networks (B5G) aims to automate the management and orchestration of running network slices. This requires heavy usage of advanced deep learning techniques in a closed-loop way to auto-build the suitable decisions, enabling to meet network slices' requirements. In this context, Federated Learning (FL) is playing a vital role in training deep learning models in a collaborative way among thousands of network slice participants while ensuring their privacy and hence network slice isolation. Specifically, running network slices may share only their model parameters with a central entity, e.g., Inter Domain Slice Manager, to aggregate them and build a global model. Thus, the central entity does not directly access the training data. However, FL is vulnerable to poisoning attacks, where an inside participant may upload poisoning updates to the central entity so that it can cause a construction failure of the global model and thus affect its global performance. Therefore, it is crucial to design security means to detect and mitigate such threats. This scenario addresses the robustness of FL algorithms to attacks, such as the poisoning attack.

Description: In this scenario, some malicious clients are injected to perturb the training of FL clients. The MonB5G solution should detect and mitigate the attack.

MoB5G solutions: We consider that running network slices are interconnected to an Inter Domain Slice Manager (IDSM), which is in charge of the management and orchestration of network slices. To enable Zero Touch Management (ZSM), the IDSM side includes an Analytic Engine (AE) for building learning models and a Decision Engine (DE) to make suitable decisions based on AE's outputs. On the other side, each running network slice is managed locally by a Domain Slice Manager (DSM), which also includes a Monitoring System (MS) for monitoring data and an Analytic Engine (AE) for building learning models. For privacy, running network slices may share only their model parameters with the central IDSM. The latter will aggregate them

and build a global model. Thus, the central entity does not directly access the training data. However, FL can be exposed to poisoning attack. To mitigate this attack, we design a novel framework to automatically detect malicious participants in the FL process. In particular, our framework first uses a deep reinforcement algorithm to dynamically select a network slice as a trusted participant, based mainly on its reputation. The selected participant will then be in charge of identifying poisoning model updates by leveraging unsupervised machine learning.

Experimental Scenario 2.3: aLTEr Attack detection and mitigation

Objectives: In this scenario, we propose a Zero-touch Security Management solution, with a local security orchestrator, that addresses the challenges of aLTEr attack detection and mitigation. The aLTEr attack is an MITM attack type and is carried out between the user equipment (UE) and the gNodeB (gNB). It involves a breaking layer two of the user radio bearer, exploiting the fact that the user data integrity protection can be missing as a vulnerability to carry out the attack. To detect the attacks, AI/ML algorithms were explored to detect the aLTEr attack, executed at AE. The mitigation step executed by DE consists of a security policy update on the firewall to deny the private DNS address from the UE.

Description: In this scenario, attacker exploits the lack of mandatory integrity protection of the PDU session and modifies DNS messages to redirect user traffics towards malicious servers. Scenario will show the implementation of the MonB5G Autonomic Security Orchestrator architecture to handle cybersecurity incidents and to deploy on demand counter-measures via the CNF Orchestrator.

MonB5G solutions: MonB5G system integrates autonomous security orchestrator architecture to automate the processing of security incidents, enabling quicker and more effective reactions and minimizing impact of security threats. Counter measures are deployed on demand via CNF Orchestrator, being able to reconfigure and change functions. To achieve it, the environment that has been set up for the testing is based either on simulation or virtualized infrastructure. When the virtualized infrastructure is used, the hosts are virtualized machines (VM) managed by OpenStack, on which we deploy applications or worker nodes of Kubernetes (K8S) cluster. The network service of the control plane of the 5G system and the MonB5G components MS, AE, DE, and ACT are K8s services and pods, while the remaining parts of the 5GS such as the data plane and the simulator of UE and RAN are applications running on the VMs. Moreover, we also keep all host clocks synchronized by the Network Time Protocol (NTP) to have consistent timestamps in logs and to correlate events.

The focus is on protecting the user and to avoid that traffic generated by this user is used with malicious servers. The Monitoring system collects information from the N6 interface with a N6 tap which forward a copy of the manipulated DNS request to the Monitoring System: this MS will create a log file with Zeek tool. Then data is integrated by the Analytics Engine (AE). The AE leveraging various ML algorithms detects anomalies in the manipulated DNS message and raises a cybersecurity incident. This incident is then analysed by the Decision Engine for eradication: the DE infers to use DNS over TLS to enhance the integrity protection. Eventually, the Actuator (ACT) component executes the remediation order received by the DE, which requests the installation and setup of a DoT client on the UE (UERANSIM). The HTTP client (Chrome) can now safely reach the web site and the attacker can no longer intercept DNS messages.

Mapping of PoC #2 Experimental Scenarios to project objectives is presented in Table 2.

Table 2: Objective mapping to PoC #2 Experimental Scenarios

Objectives	ES 2.1	ES 2.2	ES 2.3
Obj. #1: Distributed management plane to support massive deployment of network slices		X	
Obj. #2: Definition of novel e2e slice KPIs and development of AI-based mechanisms for their accurate prediction from multi-level metrics	X		
Obj. #3: Data-driven management system based on federated learning		X	X
Obj. #4: Zero touch network configuration	X		X
Obj. #5: DE decisions tailored to the CN	X		X

3.1.3 MonB5G Architecture in PoC Testbed

The design philosophy of MonB5G, as it was defined in WP2 (specifically in [MonB5GD24]), is to provide hierarchical, feedback-loop-based control for fault, configuration, accounting, performance, and security (FCAPS) management, and slice orchestration, featuring different control loops with different scopes, goals, and timescales. This control strategy with all of its aforementioned 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 3-1 shows the MonB5G architecture mapped to PoC testbed components.

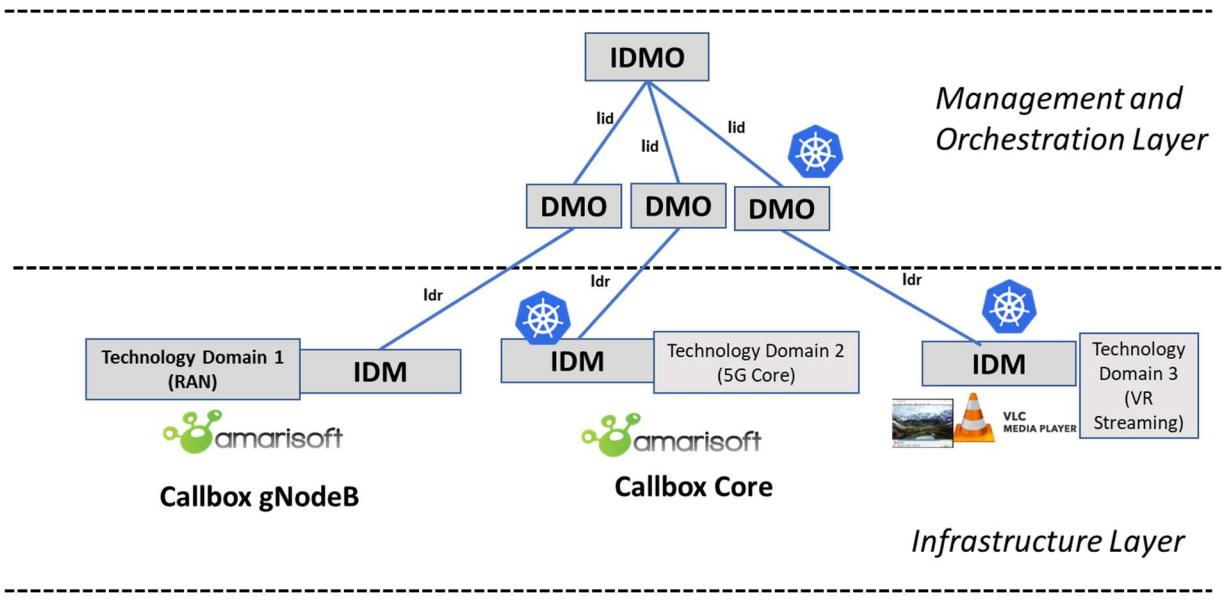


Figure 3-1: MonB5G architecture mapped to PoC.

The MonB5G architecture is composed of static and dynamically deployed components. The former is presented in Figure 3-1, namely the MonB5G Portal, the IDMO (Inter-Domain Manager Orchestrator), the DMO (Domain Manager Orchestrator) and the Infrastructure Domain Manager (IDM). The dynamically deployed components, on the other hand, are associated to the autonomic management capabilities within the slice, referred to as Inter-Slice Management (or ISM).

Whether static or dynamically deployed, all of the components of the MonB5G architecture consist of the following components:

1. Monitoring System (MS) Sublayer: responsible for collecting, aggregation and processing of the monitored data
2. Analytics Engine (AE) Sublayer: responsible of performing different types of data analytics on the data stored from the system being monitored
3. Decision Engine (DE) Sublayer: responsible of generating orchestration decisions of different types
4. Actuators (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 previously presented in [MONB5GD24] where the Functional Layer refers to the layer corresponding to the components that provide the server, while the MonB5G layer holds the aforementioned MS, AE, DE and ACT sublayers. In the design philosophy of MonB5G, the DMO (a static component of the MonB5G architecture) consists of 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. To realize DMO and IDM in corresponding testbeds, we have utilized separate Kubernetes clusters for both MonB5G components and infrastructure related components.

Starting from Section 4, 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 to validate these contributions. These high-level descriptions given for each experimental scenario are given in Sections 3.2 and Sections 3.3 for the First and Second Proof-of-Concepts, respectively.

3.2 Infrastructure Details of the Testbed and Interconnection

3.2.1 Testbed for PoC-1

Details on providing access for remote users for deployment are as follows. The MonB5G project partners are granted access to a CTTC Testbed Instance (TI), which serves as a virtual experimental environment that comprises an independent and self-contained Network Functions Virtualization ecosystem. Prior to utilizing

the resources, the partners have requested their needs and the resources that CTTC can provide in terms of time and effort. To facilitate this work, a Testbed Instance Requirements template has been provided to each involved partner. Partners were required to fill out the template, which includes the description of the solution or experiment for which the TI will be used, the technical and administrative personnel involved from both sides, the type of Testbed Access needed (remote or physical), the estimated duration of the TI, and most importantly, the resources required.

Hardware details: In this section, we describe hardware details for PoC#1 testbed. The typical resources available in a Testbed Instance include:

- Computing: physical servers, virtual machines, and containers.
- Platform as a Service/Management and Orchestration: Kubernetes
- Monitoring Systems: MonB5G MS, Prometheus, and Kube-Prometheus.
- 5G (see Figure 3-2):
    - 5G Core: Amarisoft Callbox 5GC, Free5GC, OAI CN, and Open5GS.
    - 5G RAN: Amarisoft Callbox gNBs and UERANSIM gNB.

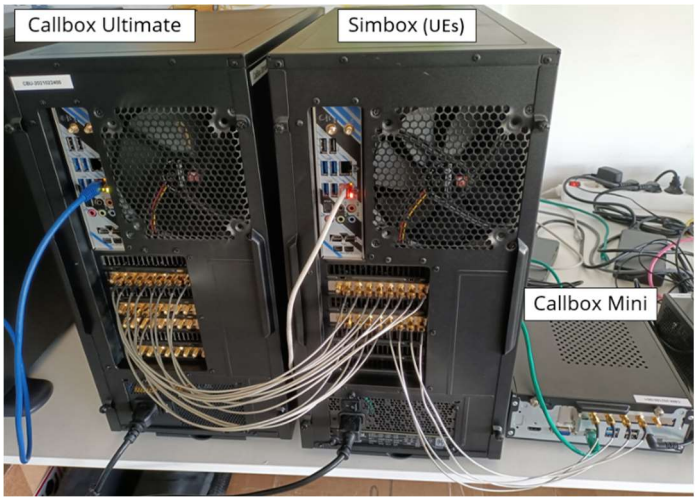


Figure 3-2: 5G hardware components.

Regarding remote access to the Testbed Instance, CTTC provides partners with a VPN connection. Specifically, a physical server located at CTTC's premises hosts a Linux Container (LXC) with an OpenVPN (OVPN) server service running inside. The OVPN server has been configured to allow access to the specific virtual machines (VMs) involved in the TI, using routing rules.

CTTC's infrastructure exposes a public IP address to the partners, which can be accessed using OVPN client files created and given specifically to them. These client files enable partners to connect to the OVPN server and access the restricted environment, a private network, through Secure Shell (SSH) protocol. Access is granted only after the partner's public IP address or DNS domain has been whitelisted in CTTC's infrastructure firewall. Finally, the traffic between the partner and the OVPN server is established and encrypted using the OVPN client file.

The objective of PoC1 infrastructure setup is to be able to show contributions by the partners and achieve **integration**. PoC#1 setup is given in Figure 3-3. Both ES 1.1 solution 1 on Federated Learning and ES 1.1 solution 2 on Anomaly detection have three sites, each with a Kubernetes cluster running a monitoring system instance. Site 1 (or A in this figure) and Site 2 (or B in this figure) run the main demo workloads and sampling functions for the monitoring. In site-1 we have univariate and multivariate anomaly detection modules. In Site-1, 2 and 0, we have federated learning modules. Site 0 (or C in this figure) is used for federated learning aggregation server.

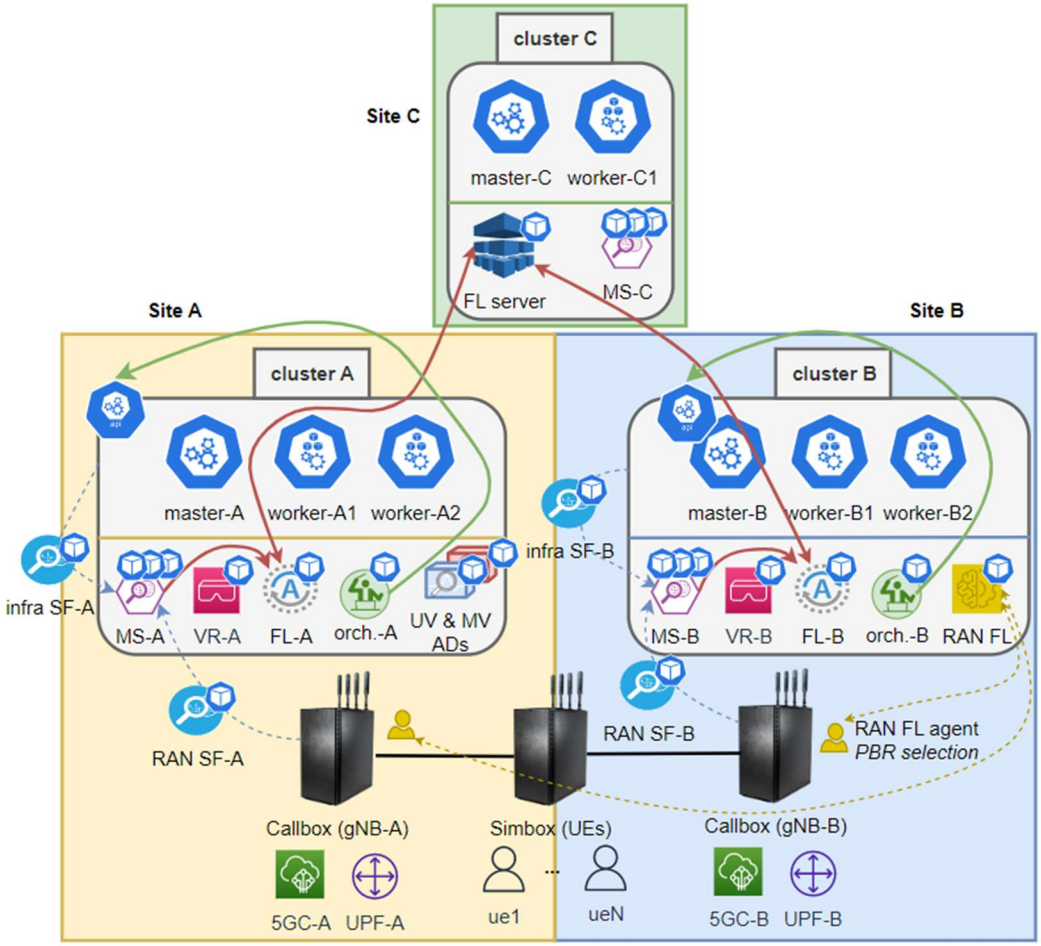


Figure 3-3: PoC1 Architecture components

Another testbed instance for PoC#1 has been deployed and consists of the 5G CORE (with Open5GS and OpenAirInterface deployed, however Open5GS is used for the current functional validation), the 5G RAN (consisting of an Amarisoft Callbox gNB) with an Edge Cloud Domain attached, and 5G UEs. Enhanced Context-Aware Traffic Prediction (ECATP) was deployed in this testbed, and it was tailored also for the deployed of Prediction-Based Admission Control, which relies on ECATP as part of its input variables.

Testbed for ES 1.1 and ES1.2 Infrastructure Components available: The testbed instance we will be using in this scenario is designed to evaluate MonB5G solutions in a 5G network environment. It is comprised of a

variety of different components that work together to enable the testing and evaluation of different federated learning approaches. The components can be categorized as follows:

- **Cloud-native infrastructure:** the cloud-native computing infrastructure of the testbed is built using containers and virtual machines. In essence, VMs are used as (worker/master) nodes to deploy Kubernetes clusters, while containers run different applications and microservices. This allows for flexibility and scalability in the testbed, as well as the ability to easily replicate and test different configurations. In particular, we consider three Kubernetes clusters: two for the edge/RAN domains and one in the cloud for showcasing the hierarchical features of the monitoring system.
- **Monitoring System**: To monitor and troubleshoot the testbed, we deploy three instances of the MonB5G MS. The FL-related MS instances running at the edge sites will run customized sampling functions (for infrastructure monitoring and for RAN monitoring). These monitoring systems will be able to monitor and collect metrics from the testbed infrastructure, as well as from the RAN. We also use Prometheus to expose some metrics of interest, which can then be visually displayed in Grafana for easing their analysis. The cloud MS will run a generic SF to showcase how aggregated/processed metrics can be taken from the MS instances at the edge sites.
- **Orchestration**: The testbed uses a Python script based on the Kubernetes scheduler to act as the intermediary between the federated learning agents and the Kubernetes scheduler. The script triggers actions related to pod scale-up/scale-out, to ensure pre-emptive and efficient resource allocation via the federated learning process.
- **Federated learning (FL)**: The testbed includes a FL client that builds local models in phase I and makes inferences in phase II. The FL client connects to the monitoring system to collect and analyse metrics during the learning process. The FL aggregation server is responsible for exchanging weights between FL clients to aggregate the learned models.
- **5G network**: The testbed includes a UE emulator realized with Amarisoft Simbox and 2x gNB realized with Amarisoft Callbox. These components are plugged together to create a rich multi-domain setup for testing federated learning algorithms. There are different alternatives for the 5G core, including Amarisoft's proprietary core or open-source projects like Open5GS, Free5G, or OpenAirInterface Core Network. Each of these open-source cores have been integrated with Amarisoft Callbox gNB, so we have the flexibility to use any of them.
- **Application**: The testbed includes video on demand (VoD) and streaming servers/clients, both of which are cloud native. High bandwidth video is used to stress the CPU of the server and trigger actions, such as scaling out pods, to test the performance of the federated learning algorithms under different conditions.
- **UV and MV AD**: The testbed includes univariate (UV) and multivariate (MV) anomaly detectors. These systems connect to the MS to collect and analyse metrics for both learning and inference (to detect anomalies). UV AD uses single input and MV AD uses multiple inputs.

In the case of the testbed instance deployed at IQU, similar components are also present:

- 5G Network: two Amarisoft gNBs materialized with the Amarisoft Callbox models
- Orchestration capabilities: materialized through OpenStack and Opensource MANO (OSM) deployed in a virtualized environment.

- Monitoring System: deployment of multiple instances of the Monitoring System (MS) for data gathering and for deployment of the different enablers.
- Cloud-native Support: this is necessary to allow the flexible deployment of the enablers, and to provide federation of resources across the 5G Core Cloud and the 5G Edge Cloud Domain. A Kubernetes cluster is deployed to provide for this functionality.

3.2.2 Testbed for PoC-2

Providing access for remote users for deployment

a. Site description

The site is located at EURECOM premises in Sophia-Antipolis (South East of France). The site offers a 5G facility composed of Radio, Core Network, and an Edge platform. The 5G radio and Core Network are based on OpenAirInterface (OAI), the edge platform is running a Kubernetes (K8) cluster. The facility supports full 5G StandAlone.

b. Radio infrastructure:

The radio infrastructure includes indoor and high-power outdoor radio-units operating in several 4G and 5G bands in the immediate vicinity of the test site, specifically Band n38 (2.6 GHz TDD), Band n78 (3.5 GHz TDD) and Band n258 (25 GHz TDD). The outdoor units are interconnected with the switching fabric using 300m fiber (10/25 Gbit/s). The units are a combination of in-house designs and commercial remote radio-units. The outdoor installation is shown below:

Figure 3-4: Outdoor and Indoor Radio Units

The RUs used at the site include: AW2S eCPRI split-8 2x2n38, 2x2 n78, 4x4 n78 (upgradeable to O-RAN 7.2 Q3 2022), Mavenir O-RAN 7.2 n78 (indoor), InterDigital MHU n258.

The sites RAN elements are based on the OAI CU/DU deployed as a cloud-service using K8S. The RAN is fully open and configurable to within the limitations of the OAI implementation. The implemented RAN interfaces are: fronthaul (eCPRI, ORAN 7.2, UHD), split-6 NFAPI, F1-C, F1-U, E1, E2, N1, N2, N3.

c. 5G Core Network

The core network part of the testbed is a partial 3GPP 5GC service-based architecture including the following elements: NRF, AMF, SMF, multiple UPF, UDN, UDR, AUSF, NWDAF and NSSF. All these elements are cloud native and can be deployed using Kubernetes.

d. Edge computing

The site's cluster computing resource makes use of RedHat's OpenShift 4.9 Kubernetes container platform and benefits from technical support from REDHAT. The main cluster is used for radio-access, core network and mobile-edge functions. The main cluster is comprised of 3 2x18-core Xeon Gold 6154 and 3 2x18-core Xeon Gold 6254 x86 servers (Dell R640). All nodes are connected 2x25Gbit/s to the switching fabric. One computing node has extra fronthaul networking interfaces for direct interconnections with O-RAN radio equipment. A second single-node OpenShift cluster built on a 2x64-core AMD Epyq server (Dell R7525) is available for sandbox testing and as a dedicated 3GPP-DU cluster. It is equipped with 4x25Gbit/s Ethernet connected to both the main switching fabric and for direct interconnection with O-RAN radio equipment.

e. Management and orchestration of vertical trials

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 3-5 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.

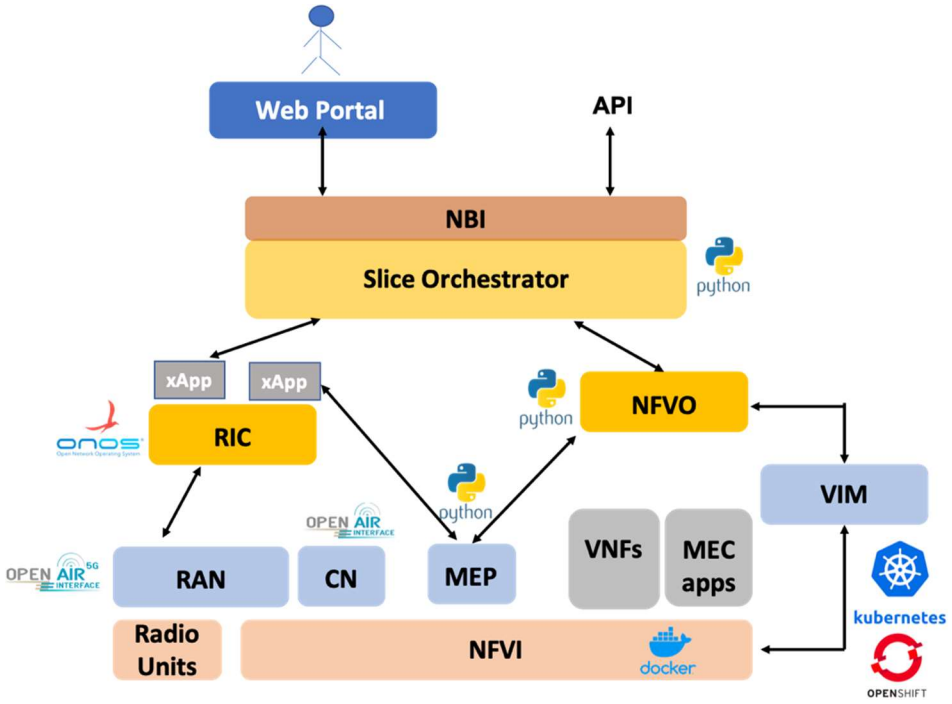


Figure 3-5: Global architecture of the 5G Facility

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. 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 sees 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 deploy 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.

3.3 Application for PoC-1 Setup

In this section, we discuss about video streaming application details and deployment.

3.3.1 Video Streaming Overview

In Figure 3-6 we show a scenario where UEs attach to a gNB that provides access to a streaming server, which is stressed with high demanding transcoding workloads to emulate VR/AR processes. The UEs are realized through the *Remote UE* mode enabled by Amarisoft Simbox emulator, and the gNodeB Amarisoft Callbox, which also provides the user plane connectivity. The emulated VR video streaming server is deployed as a Nginx pod in a Kubernetes cluster. We rely on MonB5G sampling functions to feed monitoring data to the monitoring system, and the monitoring system exposes this data to Grafana for the sake of representation and alarm triggering. The monitoring system also feeds the analytic engine, which is trained to forecast pod's CPU. Then, the decision engine and actuator deployed also as pods in the cluster, pre-emptively trigger actions through the Kubernetes application programming interface to scale up/down the virtual reality streaming server pod's CPU.

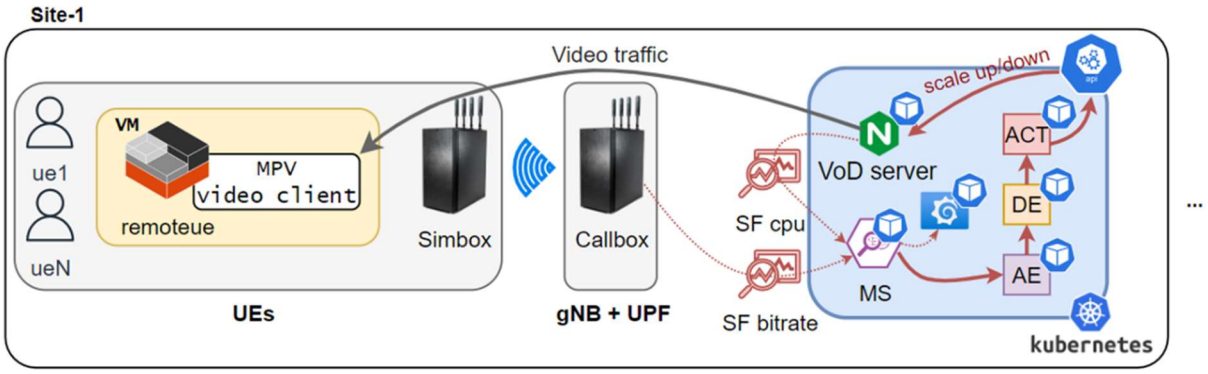


Figure 3-6: VR Video Streaming infrastructure and PoC #1 setup.

3.3.2 Example Video

To run the experiments, a streaming video server has been deployed with the help of a NGINX server. It provides video-on-demand and video streaming, which can be accessed by any user (or UE) for real-time reproduction. This VR video streaming emulation aids to assess the performance of the network and therefore the benefits that each solution has brought. For testing reasons, we adopted the well-known "Big Buck Bunny" video, using h.264 encoding protocol and a resolution of 1920x1080p. All the characteristics of the streamed video are shown in Table 3

Table 3. Original video file information.

Video codec	Advanced Video Codec (AVC)
Width	1920 pixels
Height	1080 pixels
Display aspect radio	16:9
Duration	10 min 34 s
Max Bitrate	16.7 Mb/s
Frame rate	30 FPS

Note that the video bitrate may not be challenging enough for 5G. On the other hand, it should be noted that video transcoding can impose significant stress to the system, especially considering the absence of acceleration hardware [Zhu2019]. This is the rationale behind selecting this video file.

3.3.3 The RTSP Protocol

At the server side, we use **cvlc** to invoke the VLC media player with streaming protocol set to Real-Time Messaging Protocol (RTSP) with the server address and port left blank (rtsp://:8090/stream). The --sout-keep option allows the output stream to be kept open after the file ends.

Notice that RTSP is a popular choice for testing video and virtual reality solutions for several reasons. Firstly, RTSP is a streaming protocol that is specifically designed for delivering real-time multimedia content, making it suitable for applications that require low-latency streaming, such as video and virtual reality. RTSP also allows for the efficient delivery of audio and video data over the network, enabling smooth playback and interactive experiences. Second, RTSP supports a client-server architecture, where the server hosts the multimedia content and the client requests and receives the content. This makes it easy to test and simulate realistic scenarios, such as multiple clients connecting to a server, in a controlled environment. RTSP also provides features such as content synchronization and time-based seeking, which are beneficial for testing video and virtual reality solutions that require precise timing and synchronization of multimedia content. Furthermore, RTSP is a widely supported protocol, with many media players and streaming servers supporting its use. This makes it a versatile choice for testing video and virtual reality solutions across different platforms and devices.

3.3.4 Emulating High Demanding VR Video Streaming with Transcoding

Overall, applying transcoding to a regular video stream solution can be a good approach to emulate the high CPU consumption that may be generated by VR/AR solutions, providing a realistic testing environment for evaluating the performance and scalability of video streaming solutions in scenarios that resemble the resource-intensive demands of VR/AR applications. For those experiments related with the FL use case, we enable transcoding in VLC by generating MPEG-4 Video code (mp4v) in real time to highly impact the CPU consumption pod so as to emulate VR/AR high computation and stress the server.

Notice that applying transcoding to a regular video stream solution can be a good approach to emulate high CPU consumption, similar to what may be experienced in virtual reality (VR) or augmented reality (AR) solutions, for several reasons. Firstly, VR/AR applications often involve rendering and processing complex 3D graphics and interactive content in real-time, which can put a significant load on the CPU. Transcoding, which involves converting video from one format to another, can be computationally intensive and requires significant CPU resources. By applying transcoding to a regular video stream, the CPU workload can be artificially increased, simulating the high CPU consumption that may be generated by VR/AR applications.

Secondly, transcoding can introduce additional processing overhead due to the need for video compression and decompression. This can mimic the resource-intensive nature of VR/AR solutions, which typically involve encoding and decoding of multimedia content to achieve real-time streaming or rendering. Furthermore, transcoding can also impact other system resources such as memory and network bandwidth, which can affect the overall performance of the video stream solution. This can help emulate the holistic impact of VR/AR applications on system resources, beyond just CPU usage.

In fact, with federated learning solution we were able to set up scenarios where CPU forecasting and proactive management was critical to ensure good quality of service and comply with high demanding SLAs.

3.4 KPIs for the POCs and Summary of Integrated Solutions

The definition of the KPIs for MonB5G required an extensive and well-documented research effort by the contributors during the initial stages of the project [MonB5GD22]. This effort on our part was required due to the gap in the state-of-the-art regarding the definition and KPIs of 5G networks for the evaluation of security and energy efficiency, and due to the general inadequacies with regards to the distributed zero-touch management and orchestration objectives that MonB5G aimed for. The conclusion of these efforts resulted in a series of KPIs.

3.4.1 KPI Mapping to Experimental Scenarios in PoC-1

Those experimental scenarios studied in PoC#1 have different KPIs according to project objectives, and four of them are common with ES 1.1 and ES 1.2. Complete KPI relevant to PoC scenarios are as follows:

ES 1.1 for PoC#1:

- Reduce the reaction time (time from identification to resolution via appropriate reconfigurations) to an NS malfunction.
- Improve NS performance prediction.
- x10 reduction in signalling/monitoring overhead with the use of federation techniques.
- OPEX reduction due to the automation of service management.
- Increase the ratio of service management and slice LCM tasks resolved by local AE/DE components.
- Improve the accuracy of the AE/DE mechanisms for detection of slice performance degradation.
- Improve network energy efficiency by a factor of 10.

ES 1.2 for PoC#1:

- Reduce the number of SLA performance violations by 20%.
- OPEX reduction due to the automation of service management.

- Significantly reduce the amount of communicated data to the centralized management system.
- Support for orders of magnitude more Network Slice Instances (NSIs) with dynamic data-driven reconfiguration.
- Optimize the convergence time for the distributed/federated AI algorithms so that it does not exceed that of the centralized solutions.
- Generate accurate and reusable data on network slice performance.
- Reduce time to manage RAN resources dedicated to network slices, particularly for uRLLC (AE and DE are located at the edge).
- Improve on slice performance isolation by ensuring the latency and reliability (uRLLC), as well as bandwidth (eMBB) requirements of coexisting slices (measured in terms of related SLA violations and other lower-level metrics).
- Reduce the management overhead of the RAN by reducing the monitoring overhead for RAN-level slice resource (and other) reconfigurations.

3.4.1.1 INTEGRATING SOLUTIONS FOR DEPLOYMENT IN ES 1.1

Figure 3-7 below shows the ES 1.1 integration diagram with all the solutions in the CTTC testbed. It is based on the MonB5G architecture above.

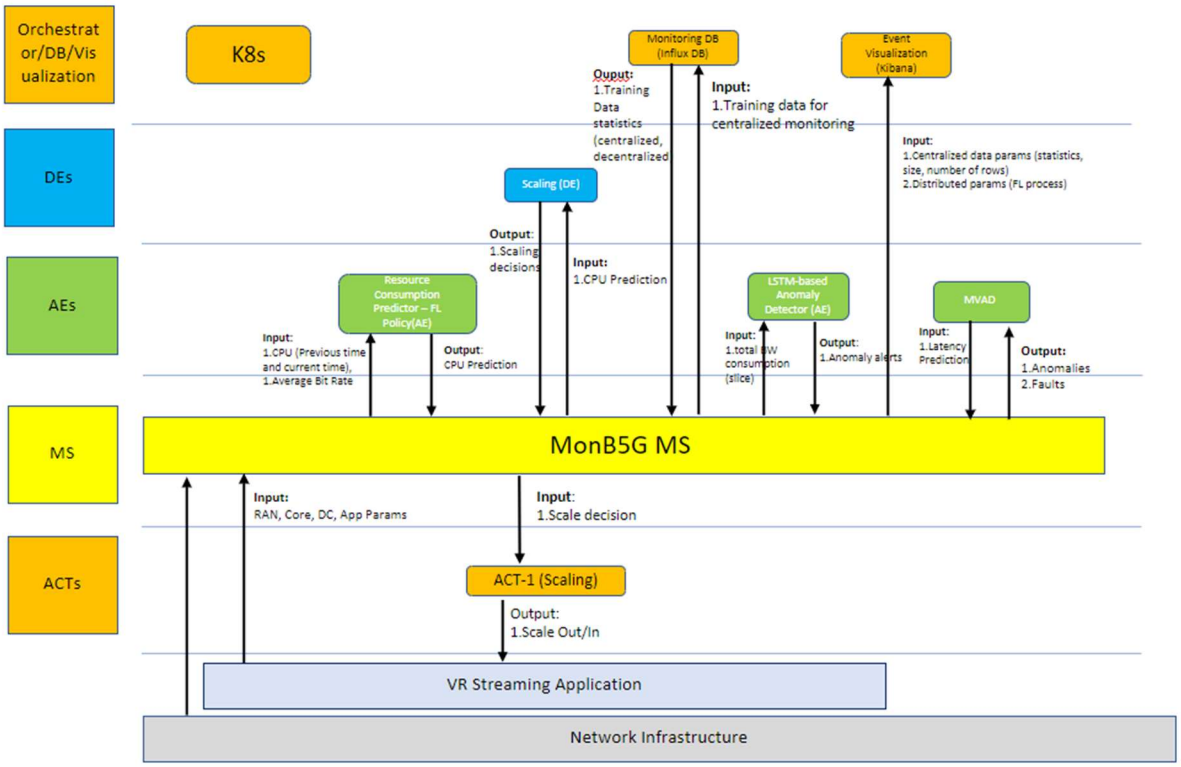


Figure 3-7: Integration of AEs and DEs with MS and ACTs for various solutions in PoC#1

Table 4 below summarizes how each solution has shown their experimental gains and how they are compared to initial target KPIs were envisioned from the start of the project.

Table 4: Target KPIs versus experimental results for solutions in ES1.1

Target KPIs	Experimental Results
x10 reduction in signaling / monitoring overhead	For the computation of the overhead, we have considered that both the datasets and update models are coded in 32 bits. In the uplink between the clients and the aggregation server, the approximate overhead can be calculated as in [MONB5GD34]. Starting from the convergence point of Federated Learning based resource predictor and scaling at round 8, reduction of communication overhead is achieved by more than x11 in the experimental setup considered in comparison with the centralized SLA-constrained algorithm.
Improve network energy efficiency by a factor of 10	Using the same approach as in [MONB5GD54] to calculate the energy and using the modeling formulas, energy reduction gain is calculated as proportional to the transmitted communication overhead gain. In our experimental case, convergence was reached after only 8 iterations. Consequently, energy reduction in the management and orchestration plane was achieved by a factor of more than 10 at the convergence point.
Increase the ratio of service management and slice LCM tasks resolved by local AE/DE components	By design of MonB5G architecture. distributed MS, AE, DE and ACT elements are deployed in 2 sites. This has increased the amount of local processing at each site. By distributing MonB5G components across multiple sites, the MonB5G architecture enables the local AE/DE components to handle a greater portion of service management and slice lifecycle management tasks. This approach reduces the dependency on centralized systems and minimizes the need for long-distance communication between different components, leading to improved performance, enabling quicker decision-making and response times.
Improve NS performance prediction	To improve network slice performance prediction, an FL-based approach has been implemented that focuses on VR video streaming server CPU load prediction. VR video

	streaming server CPU load prediction performance is similar to centralized approach (Mean Squared Error (MSE) of prediction below 0.1), while minimizing the need for extensive communication between different components or systems compared to a centralized solution. The decentralized nature of FL-based approach allows for faster and more accurate prediction results. With the prediction model running directly on each VR video streaming server, it can quickly respond to changes in CPU load and make real-time adjustments accordingly. This agility ensures that network slice can efficiently allocate resources to meet the demands of VR video streaming, leading to improved performance and user satisfaction.
OPEX reduction due to the automation of service management	The automation of network service management through distributed AI contributes to OPEX reduction in addition to the benefits brought by the FL approach, particularly in terms of energy consumption. The decentralized nature of FL allows the training and inference to be performed locally on edge devices or nodes, reducing the need for centralized processing in data centers. Since the FL approach reduces overhead and consequently energy consumption, OPEX is also improved.
Reduce the reaction time (time from identification to resolution via appropriate reconfigurations) to an NS malfunction	The time taken for GRU based model is approximately **20.42%** faster than LSTM due to faster convergence in training [Graph2023]. The proactive detection of anomalies across the multitude of NS metrics reduces the overall reaction time. The numerical analysis shown in the experiments varies, depending on the network slice resource utilization metrics. Proactive detection of anomalous behavior enables reacting with reconfiguration before fault occurs. Preliminary results of AD have been previously presented in D3.2. Results of AD using testbed data are presented in section 4.3. Numerical results would vary greatly depending on system configuration (i.e., sampling rate on different management levels), and the improvements in reaction time have implicative character.

Improve NS performance prediction	Improved quality of predictions are performed with awareness of system model and objectives of infrastructure provider oriented towards providing better insight for orchestration tool.
Reduction of SLA Violations	The FL approach considered in ES1.1 reduces the SLA violations as shown in [Chergui2021TWC] [Chergui2021]. Simulations results demonstrated in [MONB5GD31], CPU load SLA violation rates of the different slices are dramatically reduced in the constrained case and reach the target threshold, i.e., 1%, which is an acceptable value for operators and slices tenants.

3.4.1.2 INTEGRATING SOLUTIONS FOR DEPLOYMENT IN ES 1.2

Figure 3-8 below shows the ES 1.2 integration diagram of solutions integrated in CTTC testbed.

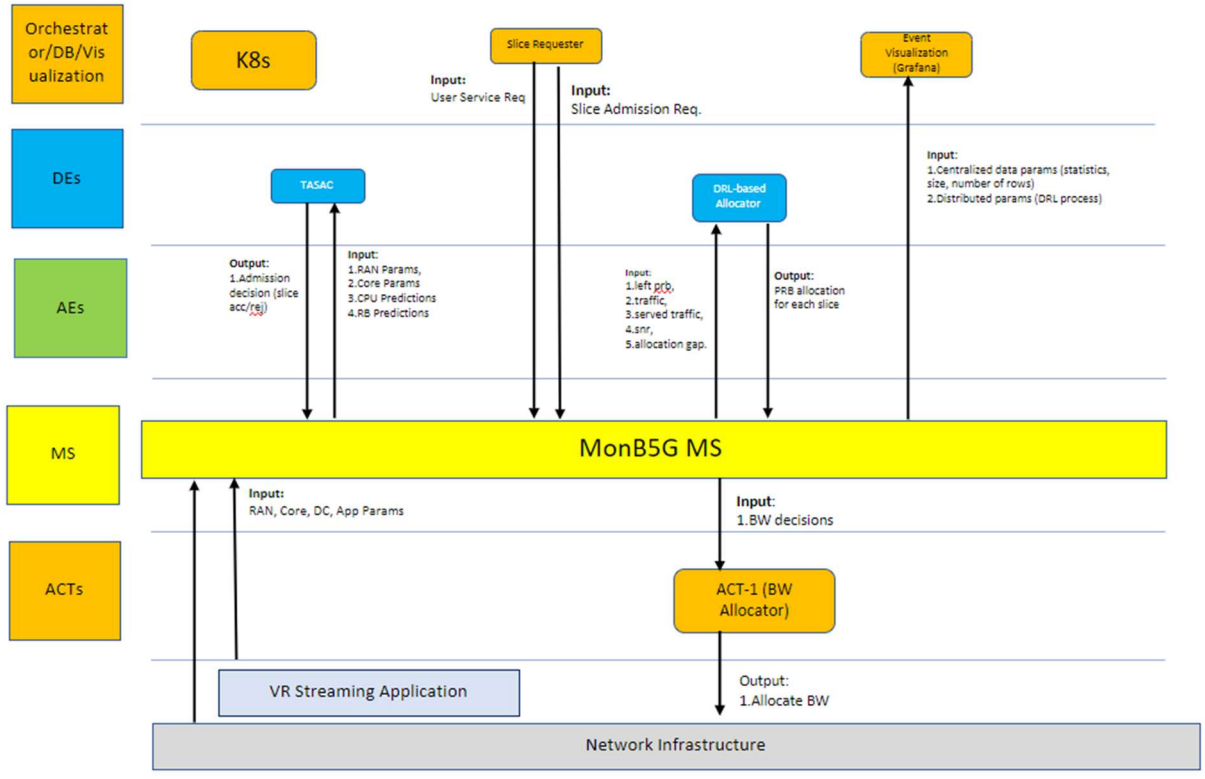


Figure 3-8: Integration of AEs and DEs with MS and ACTs for various solutions

Table 4 below shows experimental results versus target KPIs for solutions studied in ES1.2.

Table 5: Experimental results versus target KPIs for solutions in ES1.2

Target KPIs	Experimental Results
Reduce the number of SLA performance violations by 20%.	We contributed to the achievement of this KPI in several ways. Firstly, we perform a thorough analysis of the monitoring and reporting of performance metrics according to traffic and usage patterns, collecting such information from local DE deployed alongside the RAN nodes. Then, by means of AI/ML models trained on such real-time data, accurate decision schemes are derived reducing the likelihood of violations while improving communication efficiency and overhead towards the core of the network. On average 3.5% improvement in SLA violation in terms of allocation gap considering both under-provisioning and overprovisioning cases with respect to non-federated approach. Note that the difference is slight since this was done experimentally and the scenarios, we could build were small. To achieve higher performance, we needed more slices to generalize the model better and evaluate the federated approach. Results with higher number of users is shown in simulations in Figure 4-55. The number of SLA violations for TASAC DE and DQN-based DE are similar under the given resource conditions. However, decreasing the number of deployed slices for TASAC will enable to significantly diminish the number of SLA violations. (over 30% lower number of violations, cf. Figure 4-41). The User Service Request admission is increased by 3x in some instances under resource pressure conditions.
OPEX reduction due to the automation of service management.	The TASAC DE enables much better utilization of network resources via slice automated admission process (often considered as the initial step in the slice LCM). The ability to deploy more slices over the same infrastructural resources result in increased profit in comparison to conventional techniques, which can be alternatively traded for the deployment of similar amount of slices, i.e., consuming the same amount of resources, over the diminished infrastructure (allowing significant OPEX reduction). Dynamic RAN resource allocation allows to meet instantaneous traffic demand minimizing the usage of radio resources. At the same time,

	smart processing of monitoring data directly at the RAN makes the overall decision process more efficient and less cost demanding.
Optimize the convergence time for the distributed/federated AI algorithms so that it does not exceed that of the centralized solutions.	While it is difficult to characterize and control a-priori the convergence time of distributed AI/ML algorithms, we investigate such scalability aspects in simulation environment. More details are available in related publication [TVT2022].
Generate accurate and reusable data on network slice performance	The developed testbed easily allows to collect monitoring and operational data from the different involved entities, which are stored in time series database and can be retrieved by means of ad-hoc APIs.
Significantly reduce the amount of communicated data to the centralized management system.	Up to 25% improvement has been measured in the testbed.
Reduce time to manage RAN resources dedicated to network slices, particularly for uRLLC (AE and DE are located at the edge).	RAN decisions are performed by local DE collecting and processing raw monitoring information derived from the RAN nodes. This data is used to train RL-Based algorithms which automatize the radio resource management task in network slicing scenarios.
Reduce the management overhead of the RAN by reducing the monitoring overhead for RAN-level slice resource (and other) reconfigurations.	The communication overhead is reduced by exchanging only updated local models rather than raw and fine-grained monitoring information. Up to 25% improvement has been measured in the testbed.

3.4.1.3 OTHER KPIS IN ES1.1 AND ES1.2

Due to limitations in the testbed setup and demonstration constraints, some KPIs were evaluated based on emulation/simulation environments or by the definition of MonB5G architecture itself. Those KPIs are listed below:

- **Reducing Static Slicing overhead will result in 30% higher utilization (will be achieved with dynamic reconfiguration techniques):** The goal is achieved by distribution of the management operations and local processing of management information. It especially concerns the reduction of the exchange of the monitoring data and the ability to take fast, local actions during slice lifetime. The use of management loops at the level of VNFs (i.e., EEMs) is the first mechanism that significantly contributes to monitoring overhead reduction and faster reconfigurations, the embedded, in-slice management platform is the second one. Due to such mechanisms the external, management-related data exchange is significantly reduced. Please note that in the original ETSI NFV MANO approach there

is only a single OSS/BSS which has to interact with orchestrators and all network slices. The approach is highly centralised. The MonB5G architecture allows for distributed AI what also contributes to faster and better focused operations**.** The AI can be used for DEs of any level, as the architecture allows for hierarchical, control-loops based management. The AI can be also used for monitoring and analytic engines optimising behaviour of these entities by implementing their adaptive behaviour (adaptive monitoring, etc.).

- **Compared to Static Slicing, demonstrate the same or better SLA tolerances (or risk of missing SLAs) when dynamic slicing techniques**: This is achieved by using data-driven (proactive) orchestration that is native in MonB5G, as the orchestration requests concerning resource scaling, functions placement or migration can be driven by analysis of slice KPIs at the slice level not by the consumption of resources that is common in other approaches. The use of the time-of-day curve for AI-driven, proactive resource allocation also contributes to the reduction of SLA violations (the TASAC case). The FL approach considered in ES1.1 reduces the SLA violations as shown in [Chergui2021TWC] [Chergui2021]. Simulations results demonstrated in [MONB5GD31], CPU load SLA violation rates of the different slices are dramatically reduced in the constrained case and reach the target threshold, i.e., 1%, which is an acceptable value for operators and slices tenants.
- **Support for orders of magnitude more Network Slice Instances (NSIs) with dynamic data-driven reconfiguration:** This feature is mainly related to the ability of the MonB5G approach for making fast and accurate predictions related to resource consumption by running slices and on that basis taking the decisions about new slice requests acceptance. Please note that data-driven slice admission provides is more robust than the classical approach and therefore the admission reduces SLA violations. The use of the time-of-day prediction curve (the TASAC case) enables more robust and efficient allocation of available resources. The TASAC approach can be extended by slice calendaring mechanism which allows the deployment of short-lived slices at night, i.e., during the time when the consumption of resources by the 'human activity related' slices is very low.
- **Improve the accuracy of the AE/DE mechanisms for detection of slice performance degradation:** The anomalies detected accurately were helpful for the DE to execute VNF placement in the slice. The explainable framework provided more insight into root cause of anomalies and accordingly the action taken by DE.
- **Improve on slice performance isolation by ensuring the latency and reliability (uRLLC), as well as bandwidth (eMBB) requirements of coexisting slices (measured in terms of related SLA violations and other lower-level metrics)**: Multiple decision agents can be deployed and must co-exist on the same base station platform, each one dealing with the resource allocation task of a specific slice, according to its traffic demand and requirements. A safe coexistence of multiple agents can be achieved in several ways: For example, by limiting the action space of each agent to a specific portion of the radio spectrum by means of external control, or by providing the agent information about the resource leftover from other deployed agents. In this last case, a prioritization scheme should be adopted to ensure fair access to the available resources.

3.4.2 KPI Mapping to Experimental Scenarios in PoC-2

Those experimental scenarios studied in PoC#2 have different KPIs according to project objectives. Whole KPI lists are as follows:

- 10x faster attack/anomaly identification
- 10x faster attack remediation and reconfiguration in the order of 10s.
- E2e(End-to-end) slice availability > 99%
- Per slice component availability (probability that the service is available) > 99%
- False positive rate in attack classification (false classification of events as attacks) below 1%.
- Learning robustness: Precision, recall (true positive rate), fall-out (false positive rate), Area Under Curve values above/below specific thresholds vs. specific ratios of misreporting slice components.
- The accuracy loss should remain neglected. Adversarial training protects the model against adversarial attacks however this additional training may negatively affect the original model accuracy due to this a balance should be established between the adversarial training and the model accuracy.

3.4.2.1 INTEGRATING SOLUTIONS FOR DEPLOYMENT IN THE SCENARIO OF DDOS ATTACK

Figure 3-9 below shows the DDOS Attack integration diagram of solutions integrated in EUR testbed.

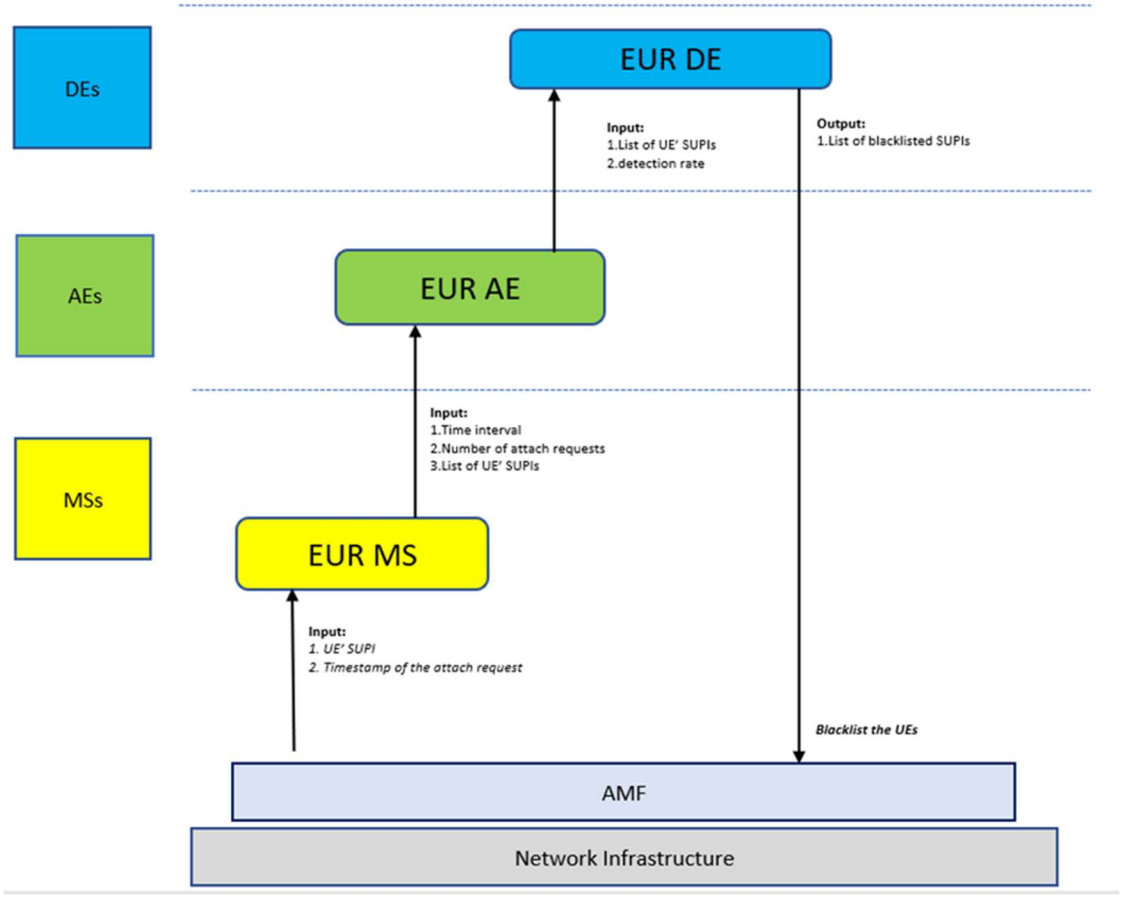


Figure 3-9: Integration Solutions for DDOS Attack

Table 6 illustrates the solutions in DDOS Attack and how they are targeting corresponding KPIs.

Table 6: Experimental results versus target KPIs for solutions in ES2.1: DDoS attack

Target KPIs	Experimental Results
10x faster attack/anomaly identification	No benchmark was available since DDoS attack detection work is novel and no prior work has been done on detecting mMTC DDoS attacks. Time of anomaly detection for the first time (i.e., UEs participating in the attack for the first time) = attack duration + 10ms Time of anomaly detection for repetitive attacks (i.e., UEs already participated in a previous attack) < 10 ms
Per slice component availability (probability that the service is available) > 99%	The slice is always available since the AMF will still receive attach requests from non-malicious UEs
10x faster attack remediation and reconfiguration in the order of 10s.	The DE blacklist UEs and send the list to AMF. The latter will save the blacklisted UEs in a database. All the procedure takes less than 10ms.
E2E slice availability > 99%	The slice is always available since the AMF will still receive attach requests from non-malicious UEs.
False positive rate in attack classification (false classification of events as attacks) below 1%	The false positive rate < 4%

3.4.2.2 INTEGRATING SOLUTIONS FOR DEPLOYMENT IN THE SCENARIO OF POISONING ATTACKS

Figure 3-10 below shows the Poisoning attack integration diagram of solutions integrated in EUR testbed.

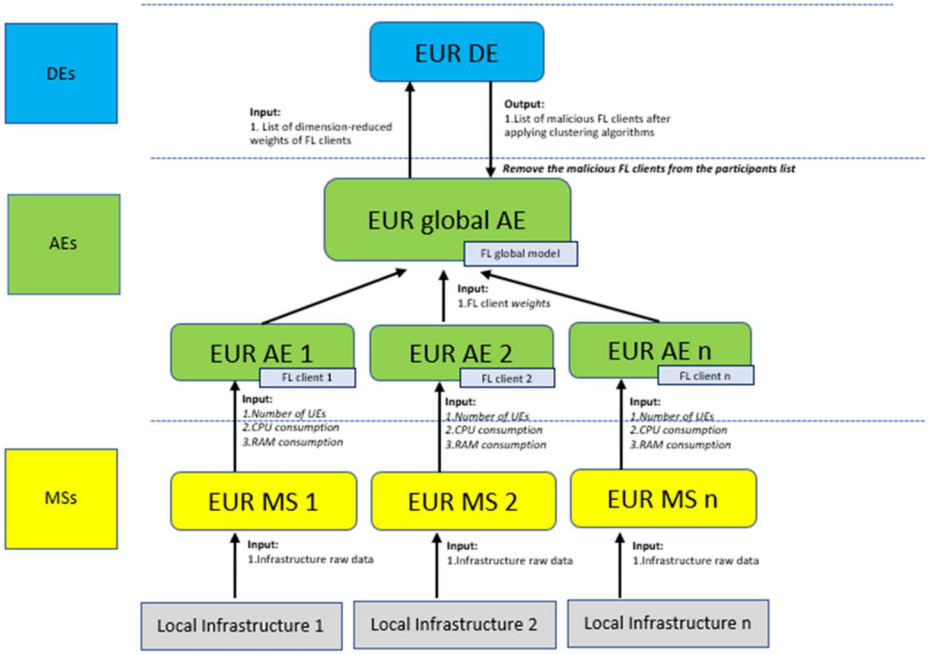


Figure 3-10: Integrating solution for Poisoning attack

Table 7 illustrates the solutions in ES2.2 and how they are targeting corresponding KPIs.

Table 7: Experimental results versus target KPIs for solutions in ES2.2 Poisoning attack

Target KPIs	Experimental Results
False positive rate in attack classification (percentage of false classification of events as attacks) below 1%.	The secure federated learning AE detects all the malicious FL clients, so the false positive rate is 0%
Learning robustness: Precision, recall (true positive rate), fall-out (false positive rate), Area Under Curve values above/below specific thresholds vs. specific ratios of misreporting slice components.	We computed the MSE by varying the number of malicious FL clients using Trust deep Q-learning Federated Learning (DE). The MSE is kept less than 0.4 while increasing the number of malicious FL clients.

The accuracy loss should remain neglected. Adversarial training protects the model against adversarial attacks however this additional training may negatively affect the original model accuracy due to this a balance should be established between the adversarial training and the model accuracy.	The untrusted FL clients are blocked, and the learning continues without their participation.

3.4.2.3 INTEGRATING SOLUTIONS FOR DEPLOYMENT IN THE SCENARIO OF ALTER ATTACK

Figure 3-11 depicts the logical interaction between the MonB5G Security Orchestrator components involved in the aLTEr attack scenario.

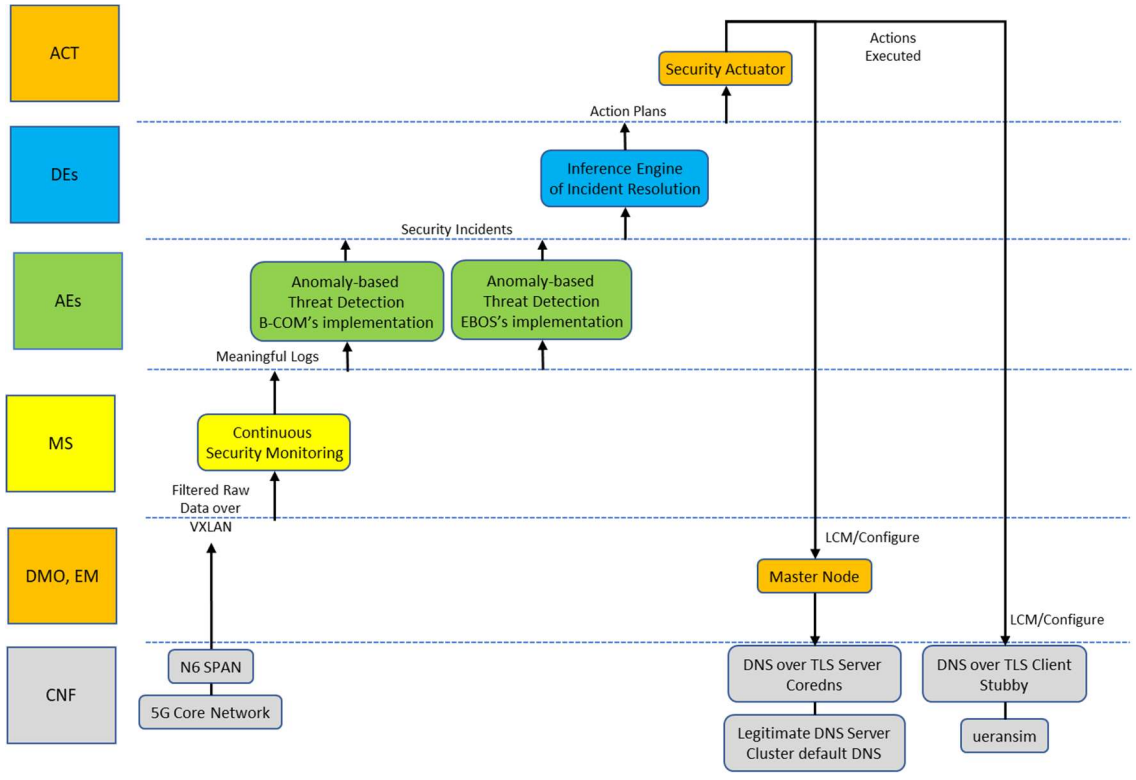


Figure 3-11: MonB5G security orchestrator integration for the aLTEr attack scenario

Table 8 below illustrates the solutions in aLTEr attack scenario and how they are targeting corresponding KPIs.

Table 8; Experimental results versus target KPIs for solutions in aLTEr attack scenario.

Target KPIs	Experimental Results
10x faster identification of security attack/anomaly	Threat detection includes security monitoring and threat identification. Together with Incident detection of the MonB5G security orchestrator (MS, AE), the overall measured MTTD is roughly 500 milliseconds (ms). The security monitoring time to make data available for analysis may vary with the observed sessions, as it requires a complete session before translating raw data to meaningful logs. The anomaly prediction time is at the order of 10 ms, therefore the main threat detection delay is related to the monitoring data extraction and data transformation. The report **[IBM2022]** on the average time to identify a data breach over several years shows that MTTD is still in order of days in 2022, however, the detection procedure might be more comprehensive and includes more procedures and various threats. Therefore, their comparison is less relevant.
10x faster attack remediation and reconfiguration in the order of 10s	Limiting the impact of the incident comprises several steps starting by choosing a remediation strategy based on criteria such as cost or effectiveness, then application of decided countermeasures. The Incident Response of the MonB5G Security Orchestrator (DE, ACT) derives the action plan to deploy the countermeasures from its rules and facts. Then, the ACT realizes the actions of the plan. Depending on the number of inference rules and facts, the inference time may vary. In the D5.3, the mean time to make decision is around 200 ms. The measured MTTR is around 3s (<10s of reconfiguration). Therefore, the ACT is the main consumer of the time budget. The ACT time depends on the execution time of invoked API. For the scenario aLTEr, there is the delay of Kubernetes to deploy a CNF as a NodePort service and the delay of the UERANSIM to enable the DoT client. However, it is difficult to compare with the benchmark MTTR even though the report **[IBM2022]** indicates days , as it depends on the complexity of the response activities and the threats.

4 Evaluation of KPIs of MonB5G Components in the Experimental Framework for PoC-1

4.1 Recap on the Deployment of MonB5G Monitoring System in PoC-1

In our PoC deployment, we will be utilizing three instances of the MonB5G 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, anomaly detectors, and an orchestrator. These edge sites are close to their respective 5G RAN infrastructure. Instead, site C is deployed at the cloud and hosts the federated learning server, which facilitates the exchange of weights between clients. Further, site C also runs a parent/root instance of the Monitoring System, which uses the monitoring systems at sites A and B as sampling functions. The overall deployment layout is illustrated in Figure 3-3.

In our implementation, we deploy and configure Kubernetes clusters using Linux Containers (LXC) to instantiate VMs that then 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 4-1.

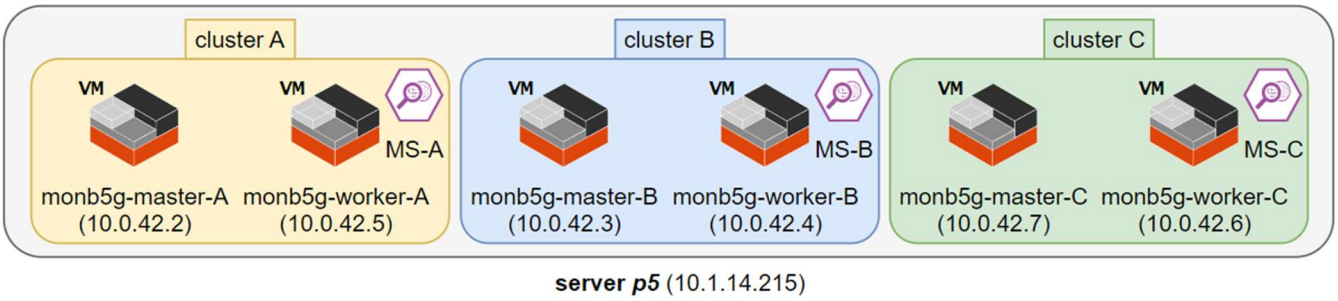


Figure 4-1: Deploying edge and cloud cluster in a multi-domain environment.

More details on the Monitoring System architecture and implementation can be found in [MONB5GD61].

4.1.1 Multi-gNodeB RAN setup

In order to enable FL, we need at least 2x FL clients, (at site A and site B), that share learning information through an aggregation server (at site C). This entails deploying a 5G network with at least two gNBs. In our case, we achieve that by relying on Amarisoft Callbox equipment. Notice that setting up an experimental scenario with two gNBs is a valid approach for evaluating federated learning solutions because it allows for realistic emulation of distributed elements. With two gNBs, altogether with a UE emulator (see Figure 4-2), we can emulate real-world cellular network conditions and assess how federated learning algorithms perform in different network environments.

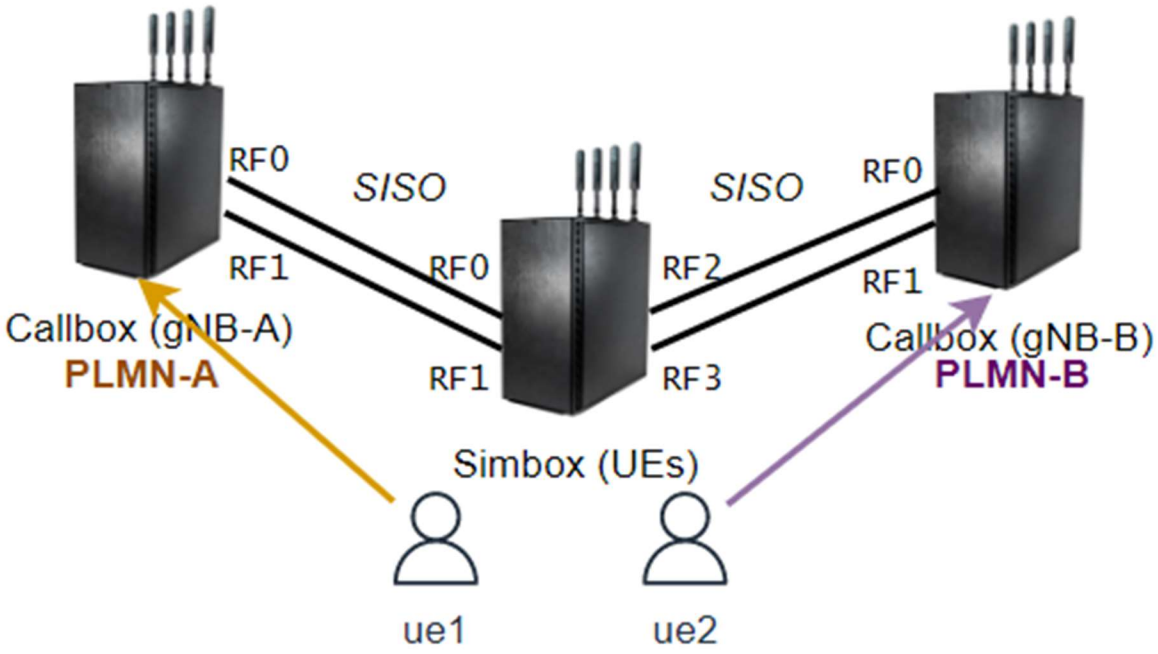


Figure 4-2: Setting up a 5G network with 2 gNBs in Amarisoft.

Setting up and configuring a 5G network with two gNBs and a UE emulator using Amarisoft equipment involves several steps. These include hardware and software setup, network configuration, radio resource management, protocol configuration, UE emulator configuration, testing and validation, and troubleshooting and fine-tuning. Each step requires careful attention to detail. In particular, as shown Figure 4-2, our approach for evaluating federated learning solutions involves connecting two gNBs, in this case, the Callbox Ultimate and Callbox Mini, to the same 5GC (5G Core Network) and enabling three cells at the Ultimate and one cell at the Mini. To ensure controlled experiments and minimize over-the-air radiation, we physically wire the Simbox UE emulator to each Callbox. This setup allows us to have all the UEs at the Simbox emulator within range of the four cells, and the system can then make decisions on which cell to attach each UE based on different conditions, particularly for PoC1.2. This approach provides a controlled environment for testing and evaluating federated learning solutions, enabling us to accurately assess their performance and capabilities.

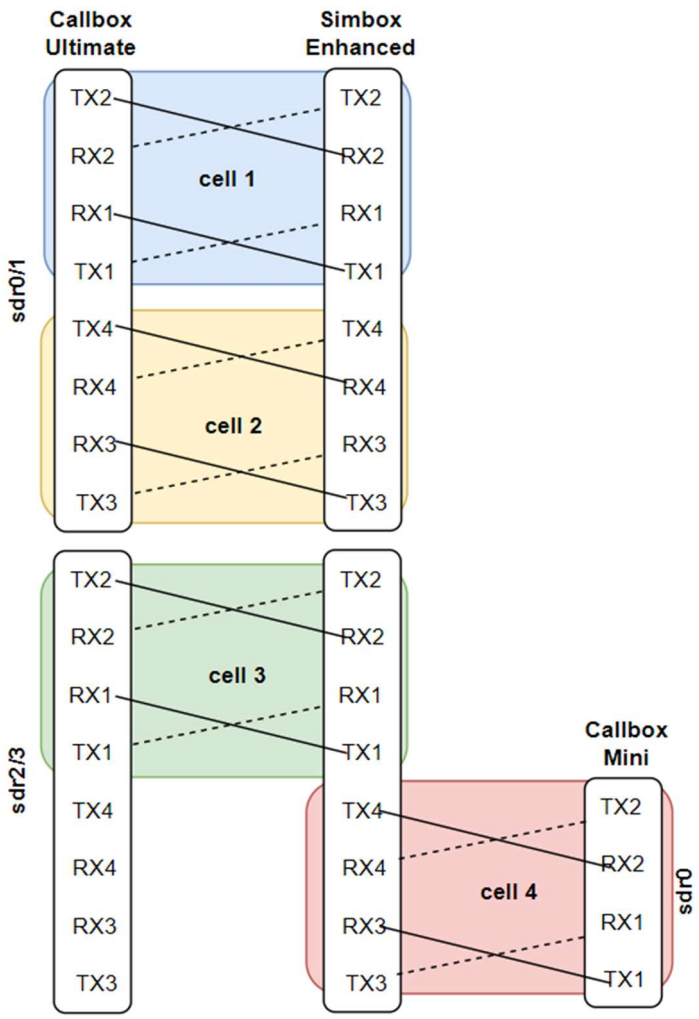


Figure 4-3: Schematic for the integration Callbox Ultimate – Simbox – Callbox Mini.

The schematic of the components and interfaces is shown in Figure 4-3. Remarkably, with such a setup UEs emulated at Simbox can attach and handover to both gNBs, including any of the 4 cells. As a minor remark, Amarisoft Simbox requires that each UE is configured to search by default for a particular cell frequency at a particular RF port. This means that UEs do not scan for possible cells at all frequencies, but simply try to find the one they have been configured, so we first configure each UE with a particular cell and then perform the handover at convenience.

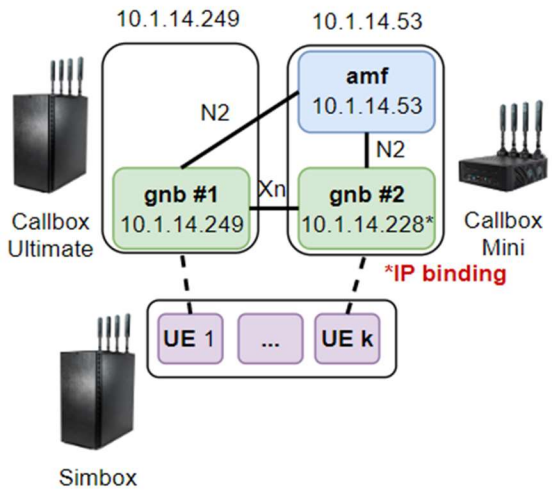


Figure 4-4: Components and interfaces for Callbox Ultimate – Simbox – Callbox Mini.

4.1.2 Amarisoft Remote UE

In order to run the video clients outside the Simbox UE emulator, we rely on Amarisoft's Remote UE tool, which forwards 5G NR traffic from/to Simbox to/from an external data network beyond the UPF. When employing a tunnel interface with a program outside Simbox (e.g., the video client), there is a need to execute that program on a separate computer with GUI (not in the Simbox itself). In such cases, the Remote UE tool can be utilized to transfer IP traffic from each emulated UE to a remote entity. To achieve this, the "lterue" program is executed on a different computer, which utilizes GTP (GPRS Tunnelling Protocol) over SCTP (Stream Control Transmission Protocol) for communication with the LTEUE at Simbox. This allows for the seamless transfer of IP traffic from the UE to a remote entity, enabling efficient testing and evaluation of the network performance in a distributed setup. An illustration depicting how video traffic is forwarded from Simbox to the *remoteue* VM running the UEs is shown in Figure 4-5. These UEs can communicate with the external VoD server through Simbox.

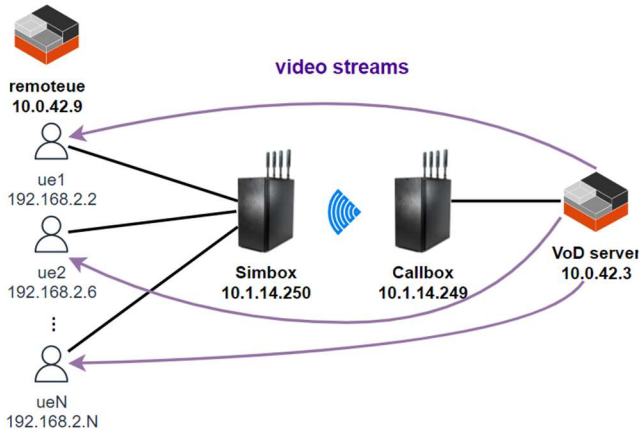


Figure 4-5: Remote UE setup with N UEs emulated by Simbox but run at remoteue VM.

A more detailed setup in the context of MonB5G PoC1.1 is shown in Figure 4-6. In there, we observe how UEs emulated by the UE are actually run at the Remote VM *remoteue*, sending and receiving traffic from to external services in the data network such as the VoD server.

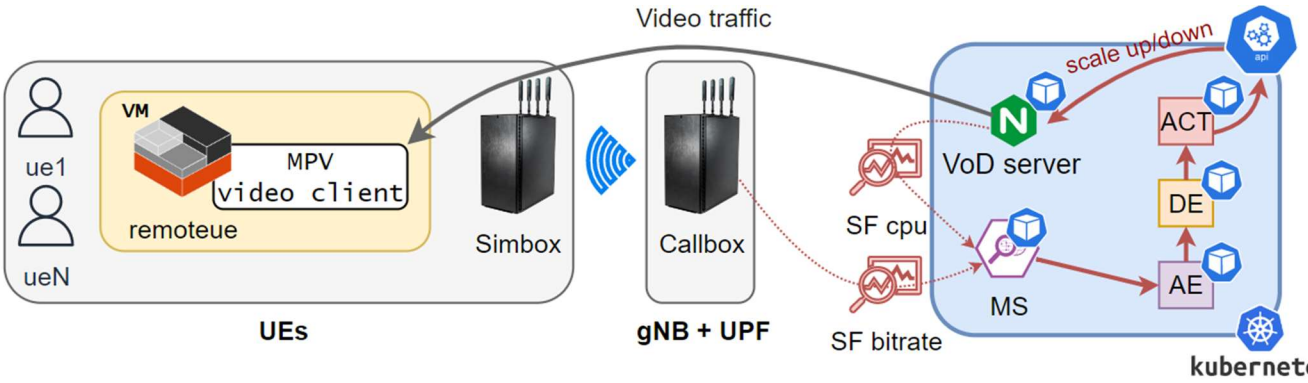


Figure 4-6: Remote UE deployment for Poc1.1.

With such a setup, we can run video clients for each emulated UE. As shown in Figure 4-7, *remoteue* GUI can display video clients for different UEs, where the video traffic of each client flows through the Simbox in a specific 5G PDU session per UE. In Figure 4-7, video traffic of each client flows through the Simbox in a particular PDU session provided by the Callbox RAN and core.

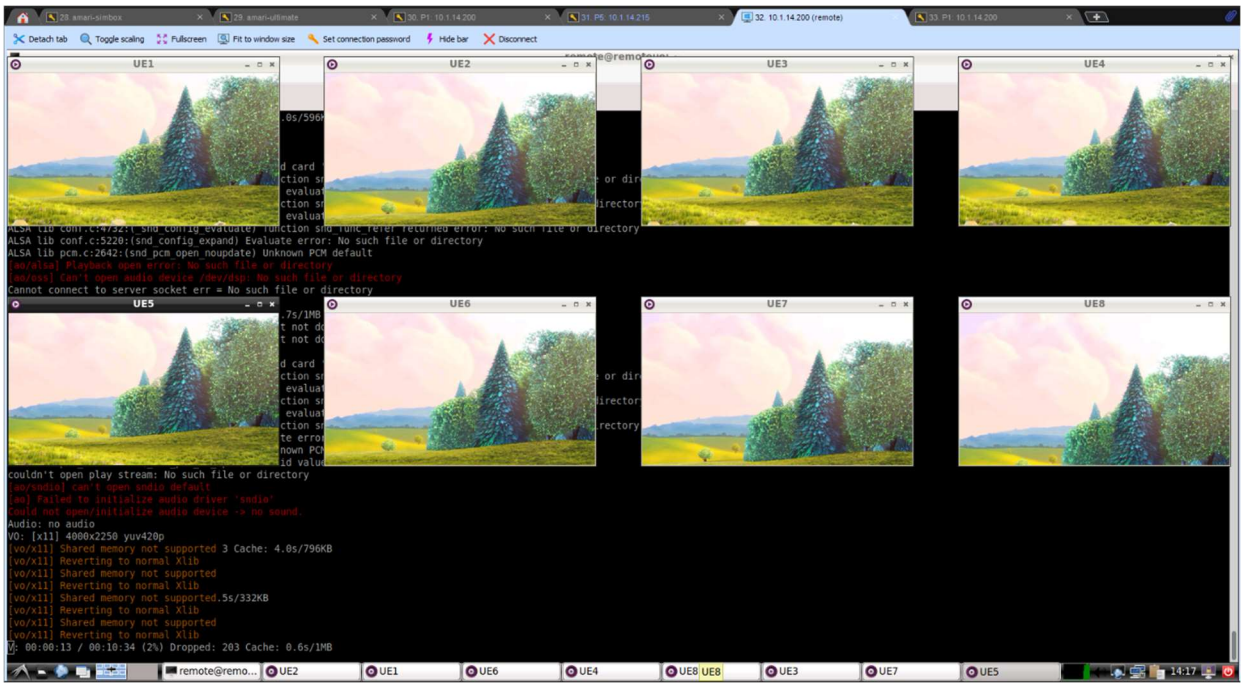


Figure 4-7: Displaying video clients for different UEs at remoteue.

4.1.3 Monitoring Sampling Functions

Once the Monitoring System is deployed at the edge sites (A and B), we 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. An overview of the MS components at edge site A is shown in Figure 4-8.

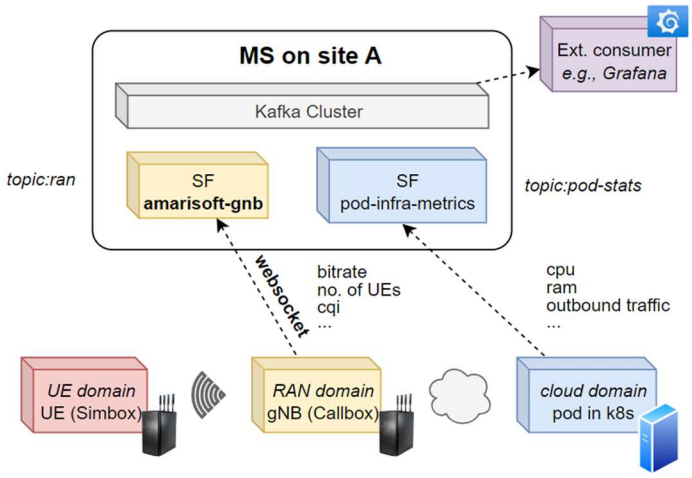


Figure 4-8: Monitoring System Sampling Functions at edge sites.

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 aggregated downlink bit rate, number of active UEs at RAN level, CQI, etc. These metrics can be used to, e.g., later estimate CPU usage at the APP server. The metrics are provided by a custom API that runs in Amarisoft Callbox. 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 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 and inference phases. A flowchart for instantiating the first sampling function is depicted in Figure 4-9.

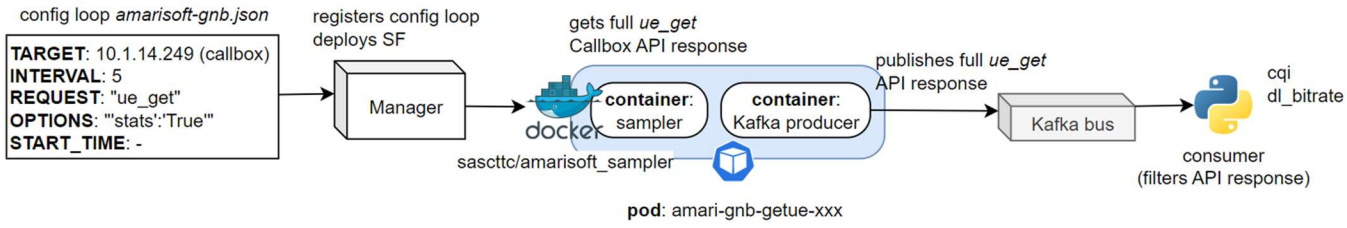


Figure 4-9: Flowchart of the Amarisoft-gnb SF.

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 VoD and video streaming 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. A flowchart for instantiating the second sampling function is depicted in Figure 4-10.

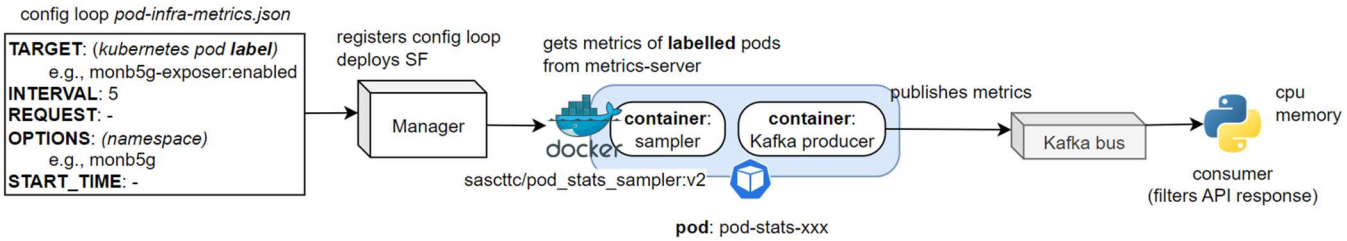


Figure 4-10. Flowchart of the pod-infra-metrics SF.

Notice that this second sampling function 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 create a ServiceAccount, ClusterRole, and ClusterRoleBinding. The ServiceAccount is used by the sampler pod to authenticate with the Kubernetes API and access the metrics. The ClusterRole and ClusterRoleBinding 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.

4.1.4 Grafana Visualization

In order to facilitate the visualization of the monitored metrics and other cluster metrics, we use 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 proceed to configure Grafana, a popular and versatile open-source data visualization tool that offers several advantages for displaying metrics of interest. One of its key advantages is the ability to create a centralized dashboard that displays metrics from diverse sources (and domains) in a unified manner. Grafana also offers customizable visualizations, allowing for the creation of intuitive and meaningful visualizations tailored to specific metrics and requirements. We have created a dashboard and charts with the queries to the metrics of interest exposed by the Kafka consumer(s). An example of the running dashboard for PoC1.1 is shown in Figure 4-11.

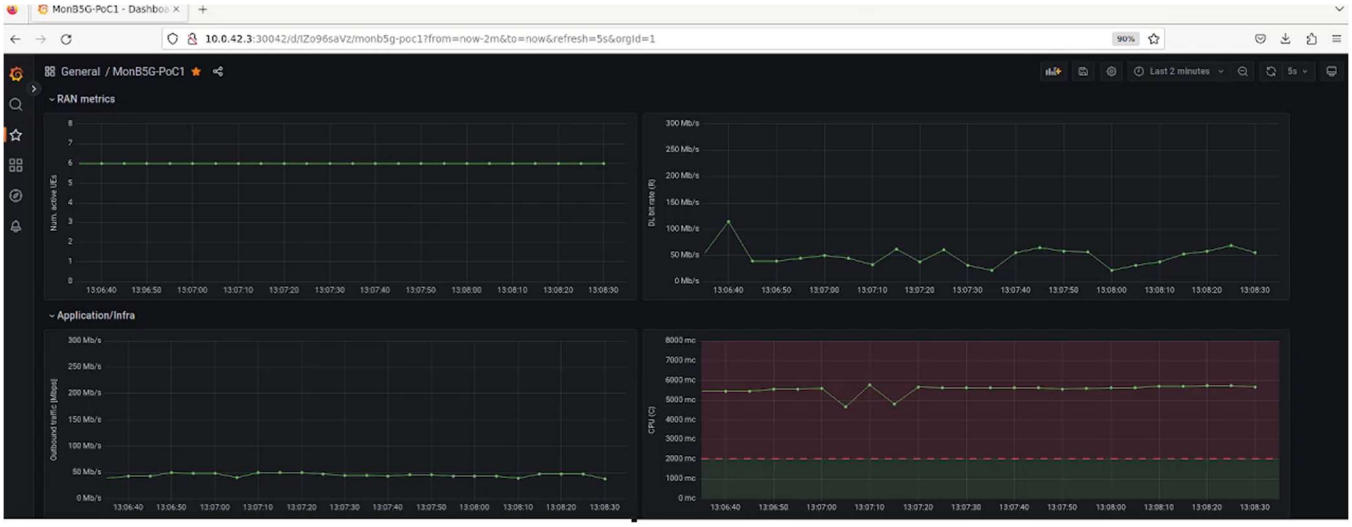


Figure 4-11: Grafana dashboard for PoC1.1.

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.

4.2 Datasets from Experimental Trials

4.2.1 Federated Learning Dataset

The dataset used for FL experiments in PoC1 scenario 1 can be found in [Sergio2023]. The dataset includes index number, time of the experiment (Time), number of VR video streaming clients (N), observed radio downlink bit rate (R), and VR video streaming server CPU (C) as shown below in Figure 4-12.

R (Bit rate): It is the number of bits that is transmitted per second on the radio network.

C (CPU utilization): It is the percentage of CPU time that is being used to process requests in the pod system. In this system, the CPU utilization also appears to increase as the number of users increases. This is likely due to the increased load on the system from more users.

O (Outbound Bit rate out of pod): It is the number of bits that can be transmitted per second out of a pod.

N (Number of VR video streaming users): It is the number of VR video streaming users in the system.

To create this dataset, a video streaming server based on NGINX was set up to provide video-on-demand and streaming services, which were accessible by any user or UE in real-time. Moreover, the number of VR video streaming clients are gradually increased from N=0 and N=8 and decreased again from N=8 to N=0. The time patterns in Figure 4-12 time is created using this dataset where each site is assigned a unique traffic pattern.

	Time	N	R	C	O
354	1.680199e+12	8	71.314170	6583.236763	61.540757
355	1.680199e+12	8	63.417927	6623.148324	59.281517
356	1.680199e+12	8	62.088018	6623.148324	59.281517
357	1.680199e+12	8	66.335833	6623.148324	61.951017
358	1.680199e+12	8	75.438206	5089.278437	61.951017

Figure 4-12: Sample dataset

4.2.2 5G RTSP Video Streaming Dataset for Anomaly Detection

This dataset[1] collects data from a 5G video streaming use case. A video is streamed by a CVLC server (realized as a Kubernetes pod) through RTSP to a variable number of 5G UE clients that activate according to a daily traffic pattern. The values of the 4 dataset features (number of active UEs, gNB's downlink bit rate, pod's outbound traffic, and pod's CPU usage) are collected by the MS.

The Kubernetes cluster, including the server pod, runs in COTS servers. The 5G core and gNB is realized through Amarisoft Callbox Ultimate. The UEs are emulated through Amarisoft Simbox. In order to display the video in ffplay clients, we use the Remote UE from Amarisoft, so traffic from Simbox is forwarded to an external VM with GUI.

In this use case, UEs trigger video streams from a video server, and the number of UEs triggering the video stream follow a typical daily network traffic pattern (see Figure 4-13) scaled and ranging from 0 to a maximum of 8 UEs. The dataset includes four variables:

- **Number of active UEs**: the number of UEs that are actually downloading data traffic (*N*). They are filtered by the condition of having corresponding downlink bit rate greater than 100 kbps.
- **Aggregate downlink bit rate**: the instant aggregate downlink bit rate at gNB (*R*).
- **Outbound traffic at the server**: average outbound traffic (O) flowing from the data interface of the video server.
- **CPU usage at the server:** average CPU usage at the server (*C*).

The system collects data at a sample frequency of 1 sample every 10 seconds, and the hour of the daily patterns have been scaled down to 2 min each for the sake of convenience, so we are able to generate 5 “days” of data in just some hours. Each iteration represents a whole day, composed of 24 "demand periods". Each demand period takes 2 minutes and is given by the number of active UEs consuming the video stream

[1] https://zenodo.org/record/7858064#.ZEY7kHbP3mg

(N). N is included for informative reasons. Sampling rate is 10 seconds, but some parameters are refreshed at a lower frequency given monitoring limitations. This means that some parameters repeat the same value in consecutive measurements.

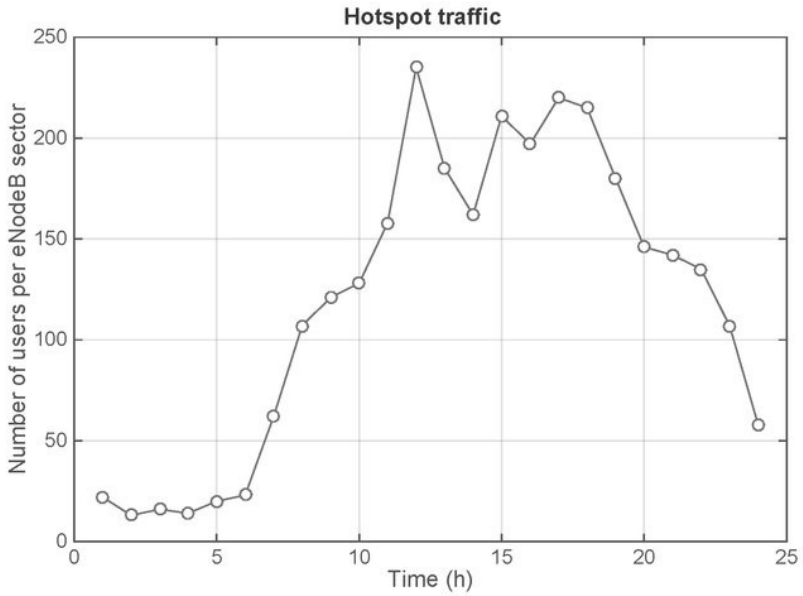


Figure 4-13: Traffic pattern throughout a day in terms of number of users per eNB sector area.[2]

Figure 4-14 below shows metrics shown in Grafana for the dataset and Figure 4-15 shows the dataset sample.

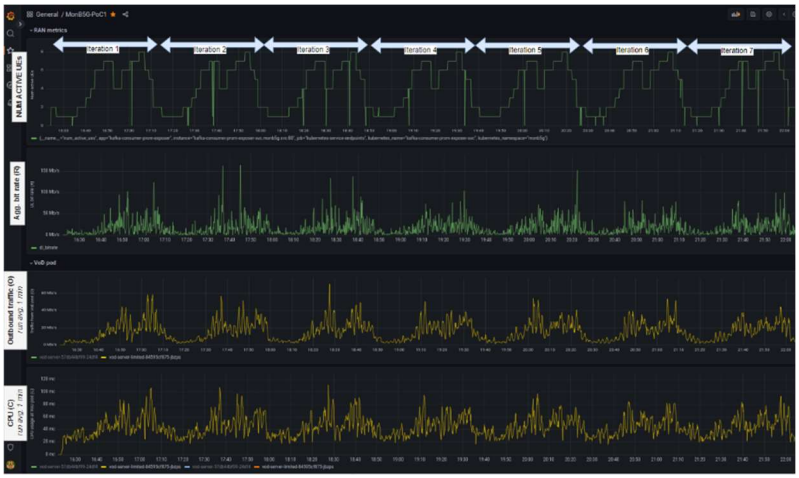


Figure 4-14: Metrics shown in Grafana for the dataset

[2] Mesodiakaki, A., Zola, E., Santos, R., & Kassler, A. (2018). Optimal user association, backhaul routing and switching off in 5G heterogeneous networks with mesh millimeter wave backhaul links. Ad hoc networks, 78, 99-114.

```
Time,N,R [Mbps],O [Mbps],CPU [mc]
1679588650000,2,2.820442,1.759633138,15.45377772
1679588660000,2,12.824529,3.184627451,27.27656836
1679588670000,2,3.393401,4.932337856,29.06567461
1679588680000,2,3.778014,5.47596754,27.64382753
1679588690000,2,2.706229,3.8329076,29.75041296
1679588700000,2,10.279406,4.920414532,31.19531485
```

Figure 4-15: Features and sample of the dataset

4.3 Evaluating KPIs of MonB5G Solutions at the PoC-1 Scenario 1 Testbed

4.3.1 Traffic Prediction with Context Awareness

In 5G networks, the traffic through each network slice varies on a temporal basis, and the ability to make predictions of its behaviour has proven useful to drive resource allocation and orchestration mechanisms throughout the network fabric. In previous deliverables, the context-aware traffic predictor (CATP) has been explained in detail, and intermediate deployments have been developed throughout the MonB5G project. For the deployment of PoC-1, an extended version of this predictor, referred to as Extended CATP (ECATP) was developed in order to exploit better statistical features of the time-series resource demand profile for context-aware prediction.

4.3.1.1 USE CASE DESCRIPTION AND BASELINE

ECATP was deployed at the testbed instance in a cloud-native way, for which its interfaces were modified in order to enable its containerization and communication with the instance of the Monitoring System (MS) deployed in the 5G Edge Cloud Domain. It sends its prediction results back to the MS for further analysis and to one or multiple instances of DEs to provide status information for orchestration and management decisions. ECATP can generate predictors with different loss functions and resulting training procedures. For this experimental scenario, ECATP was trained with two loss function of the following forms:

$$L_{ecatp01}(y_i, y_i^p) = \frac{1}{B} \cdot \sum_{i=1}^{B} C_{01}(y_i, y_i^p)$$

$$L_{ecatp02}(y_i, y_i^p) = \frac{1}{B} \cdot \sum_{i=1}^{B} C_{02}(y_i, y_i^p)$$

The cost functions C_{01} and C_{02} in the above equations are expressed as:

$$C_{01} = \begin{cases} (C_d + C_r) * (y_i - y_i^p) & y_i^p - y_i < 0 \\ 0 & 0 \leq y_i^p - y_s \leq y_s \\ (C_w + C_r + P_r * C_d) * (y_i^p - y_i - y_s) & y_s \leq y_i^p - y_i \end{cases}$$

and

$$C_{02} = \begin{cases} (C_d + C_r) * \left(y_i - y_i^p + \frac{y_s}{2}\right) & y_i^p - y_i < -\frac{y_s}{2} \\ 0 & -\frac{y_s}{2} \leq y_i^p - y_i \leq \frac{y_s}{2} \\ (C_w + C_r + P_r * C_d) * \left(y_i^p - y_i - \frac{y_s}{2}\right) & \frac{y_s}{2} < y_i^p - y_i \end{cases}$$

In these equations, the parameters have the following meaning:

- C_d, C_r, C_w: these are the costs associated to resource under-provisioning, resource reconfiguration and resource overallocation for a given network slice, respectively.
- P_r: the probability of resource under-provisioning for the network slices as related to an increase of over-provisioning for a given slice
- y_i, y_i^p, $\frac{y_s}{2}$: these are the current resource demand, the predicted resource demand, and the provisioning deviation of a network slice for a given resource, respectively.

Two other variations of loss functions where also introduced, namely $L_{ecatp03}$ and $L_{ecatp04}$, which are constructed with cost functions denoted by C_{03} and C_{04} that use a second-degree polynomial and a square root function for service degradation due to under-provisioning, respectively. Associating different cost profiles for the under-provisioning case allows to make the predictor more aware on the impact of an SLA violation, and tune it better to the needs of the human operator.

A parameter sweep was done on the parameters listed above to do a thorough comparison of different ECATP instances when providing predictions for different experimental scenarios. This parameter exploration was done using $L_{ecatp01}$ and $L_{ecatp02}$, and compared with the following baseline loss functions: Median Average Error (MAE), Minimum Square Error (MSE) and the loss function in [DEEPCOG] referred to as $L_{deepcog}$. For the latter, a parameter space was also undertaken in order to provide a fair comparison.

The experiments consisted of having three types of services generating traffic for a Base Station/gNB. Enough samples of traffic were stored, running for a period of almost a day, during which the predictors were trained periodically with their respective loss functions. Once the initial training period was done, the frequency of training was reduced, and the outcome of the different predictors were compared with each other.

The outcomes of these predictions were then sent back to the MS and logged into its local storage component. From here, Grafana was granted access to this store, from where the traffic associated to the User Service Requests (USRs) was obtained together with the associated traffic prediction, all on real time.

4.3.1.2 EVALUATION OF KPIS OVER EXPERIMENTAL PLATFORM

The quality of the prediction was assessed using the following function:

$$P_i(y_i, y_i^p) = \begin{cases} CR_{\frac{OP}{UP}} * (y_i^p - y_i) & if \quad y_i^p \geq y_i \\ 1.0 * (y_i - y_i^p) & if \quad y_i > y_i^p \end{cases}$$

In which $CR_{\frac{OP}{UP}}$ denotes the ratio of the impact of over-provisioning over under-provisioning. This ratio is usually a value smaller than 1.0, since the impact of under-provisioning is larger throughout the network, since it increases the probability of SLA violations, generating revenue losses for the Infrastructure Provider and reduction of Quality of Service for the end users.

The following plots show how ECATP outperforms other baseline predictors significantly for the experiments that were executed, with $CR_{\frac{OP}{UP}} = 0.9$. This value of $CR_{\frac{OP}{UP}}$ means that the impact of over-provisioning is 10% smaller than that of under-provisioning. Fig. 4-16 shows that training a Long-Short Term Memory (LSTM) Deep Neural Network (DNN) architecture with $L_{ecatp04}$ results in a quality of prediction at least 20% better when compared against a DeepCog loss function.

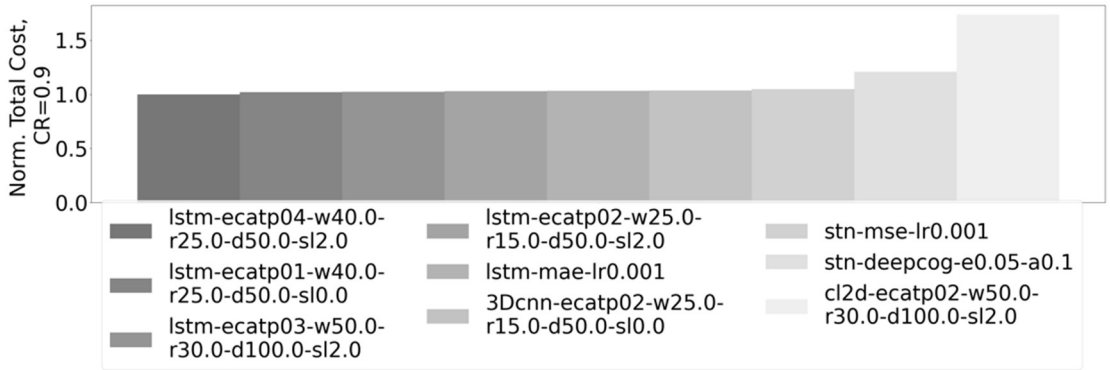


Figure 4-16: Quality of Prediction of ECATP compared with state-of-the-art predictors.

Figure 4-17 shows the probability of SLA violations, as understood as the probability of resource under-provisioning for a given slice, associated to each of the DNN architectures and Loss functions observed in Figure 4-16. Here we see that the LSTM DNN with $L_{ecatp04}$ presents the smallest probability of SLA violation, except for

the case of the Spatio-Temporal Network (STN) with Deepcog loss function. Regardless of this, the STN DDN still presents a lower quality of prediction, mostly because it yields a high probability of over-allocation as shown in Figure 4-18. In conclusion, our approach for context-aware forecasting provided by ECATP successfully improves network slice performance prediction, improving the KPIs for ES1.1 of PoC#1.

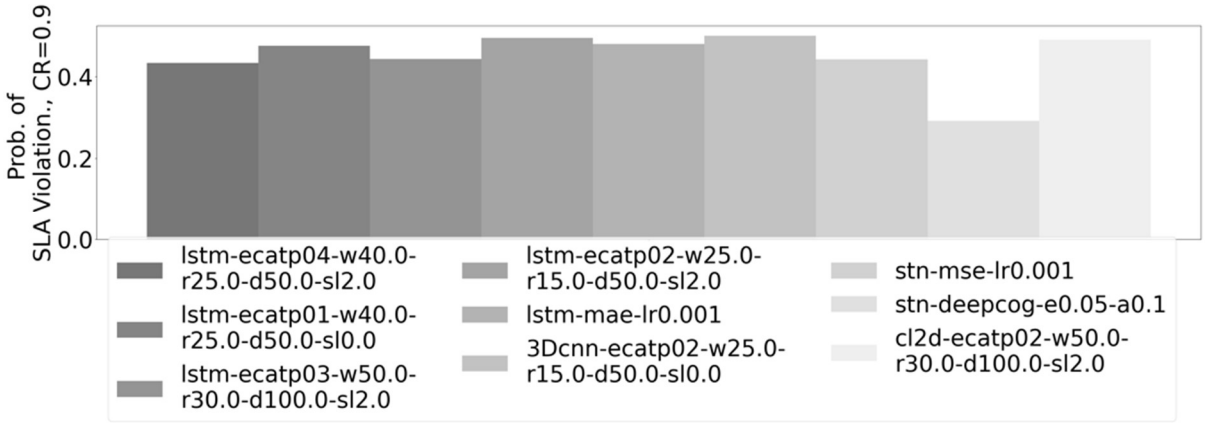


Figure 4-17: Probability of SLA violations in ECATP compared with state-of-the-art predictors.

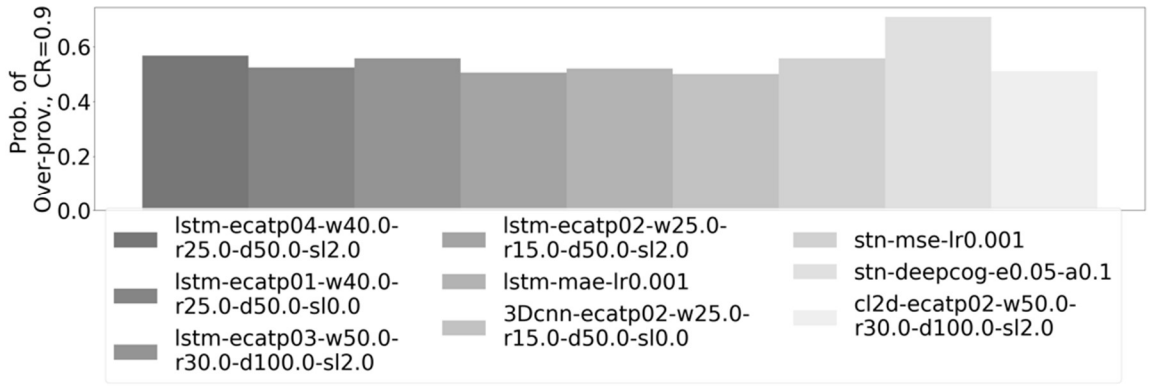


Figure 4-18: Performance of ECATP compared with state-of-the-art predictors.

4.3.2 FL Predictor

4.3.2.1 USE-CASE DESCRIPTION AND BASELINE

Current resource prediction mechanisms in edge cloud networks primarily ignore cell radio conditions to determine when to scale resources (such as CPU) in the application domain and provide very limited scaling options, mostly relying on observed parameters in the cloud. This leads to problems on the server due to congestion when there is high traffic demand as given in Figure 4-19. However, rather than taking reactive actions when traffic demand is high, proactive measures such as predicting resource utilization in advance,

considering both radio and cloud parameters, can improve the global user experience and specifically for virtual reality streaming applications. In this demonstration, we consider such proactive measures to improve the performance of virtual reality streaming services by considering radio network conditions and traffic as well.

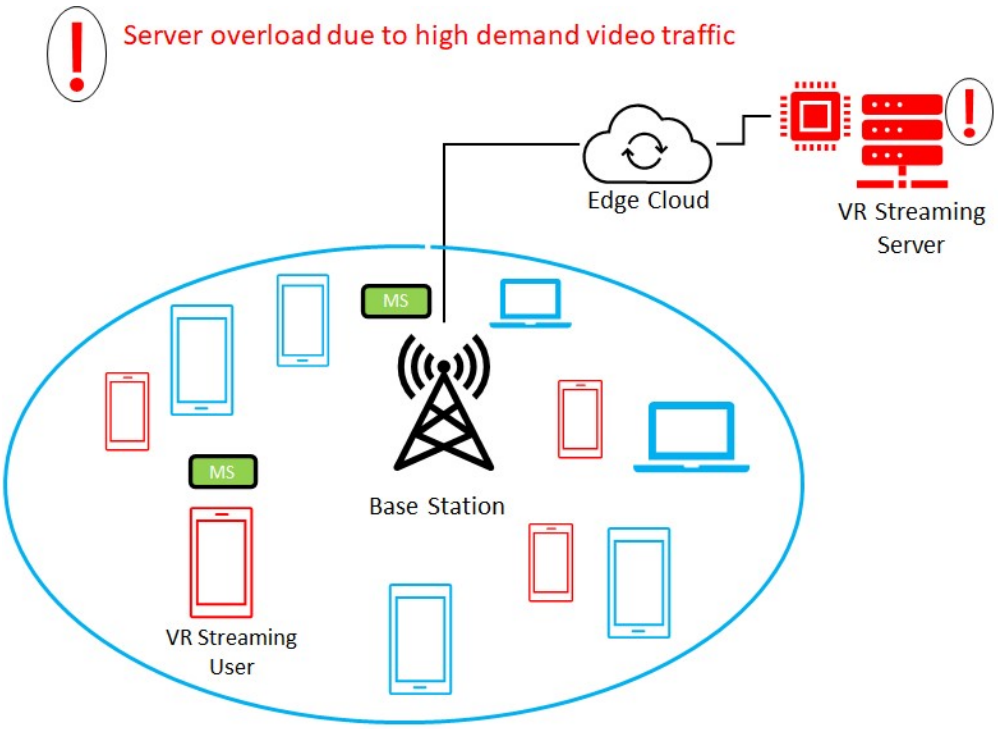


Figure 4-19: Congestion in the VR video streaming server due to overload

The proposed intelligent VR video streaming approach described below is multi-domain, intelligent, dynamic, distributed and ensures proactive resources allocation.

EXPERIMENTS DETAILS FOR POC ES 1.1

This section explains the experiments carried out to test the network, check its operation and provide evaluation results. The VR video streaming experiments are divided into two parts and will be detailed in in the following subsections.

ES 1.1 FL experiments:

Scaling of the VR video streaming: The first experiment carried out consisted of scaling of the VR video streaming server, where FL technique is used to improve the quality of experience of VR video streaming clients connected to BS at each site. To do this, the network below has been deployed with 5G Amarisoft BS, the VR video streaming clients, VR video streaming server and the MonB5G MS, AE, DE and ACT at each FL site. FL facilitates distributed collaborative learning without disclosing original training data where the idea behind FL is to train the ML model collaboratively among distributed clients without sharing their data

and violating the privacy accord. Therefore, FL locates ML services and operations closer to the clients, facilitating leveraging available resources on the network's edge.

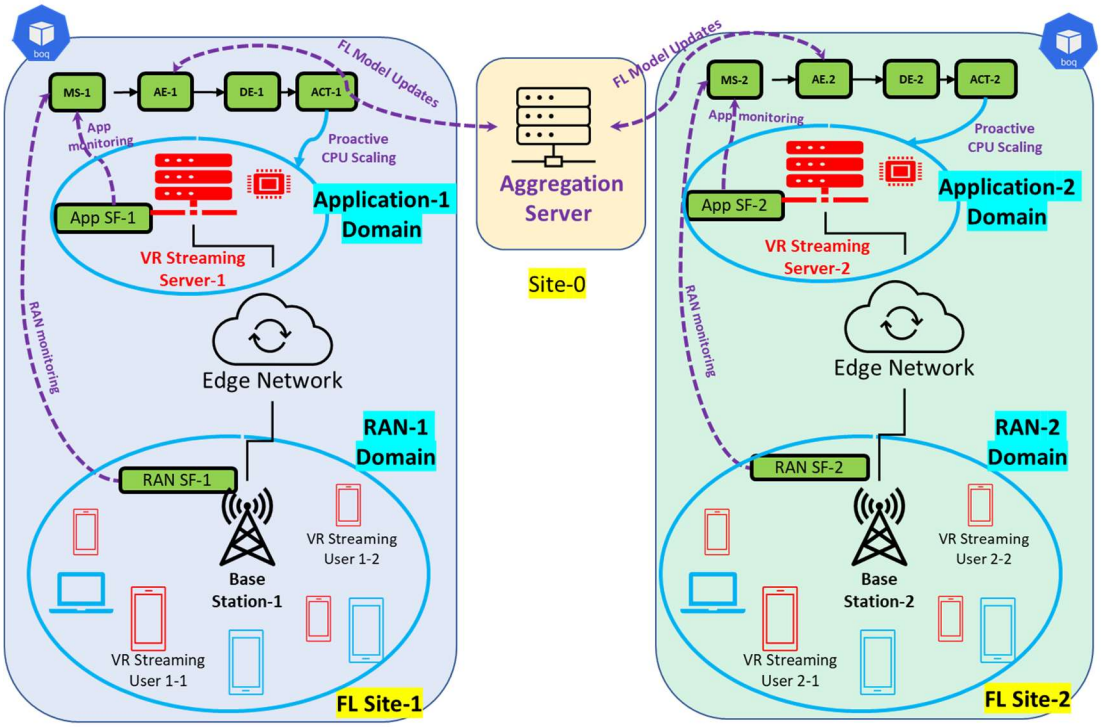


Figure 4-20: FL deployment sites, VR video streaming servers and clients

Figure 4-20 shows the experimental topology created with FL architecture at CTTC premises. All MonB5G components of the FL are implemented as pods in K8s environment in separate K8s clusters. In our demonstration setup, we have three sites to perform experiments. 5G BS and the VR-streaming servers are located in FL site-1 and FL site-2. Finally, the aggregation server is located at site 0. All these sites are connected to each other via service mesh deployment model in K8s. Each VR video streaming users are connected to corresponding BSs at each site for VR video streaming traffic from VR-streaming servers. The number of VR video streaming users are increased in different patterns.

Each site has a radio access network domain that includes a base station, core network and VR video streaming server and corresponding VR-streaming users. The user equipment are realized through the remoteue mode enabled by Amarisoft Simbox emulator, and the gNodeB is realized through the Amarisoft Callbox, which also provides the user plane function. The Virtual reality streaming server is deployed as a Nginx pod in a Kubernetes cluster.

Each radio access network domain is connected to the application domain through core network where the VR video streaming server is running and the MonB5G components, monitoring system, analytics engine, decision engine and Actuator are deployed near the VR Stream server. We rely on MonB5G sampling functions that feed monitoring data to the monitoring system, and the monitoring system exposes this data to Grafana for the sake of representation and alarm triggering. The sampling functions

for the application and radio access network are each performed at the location of the VR-streaming server pod application and the base station respectively. So, sampling functions collect data to monitor radio access network and application related data at site-1. The monitoring system also feeds the analytic engine, which is trained to forecast pod's CPU. Then, the decision engine and actuator deployed also as pods in the cluster, preemptively trigger actions through the Kubernetes application programming interface in order to scale up/down the virtual reality streaming server pod's CPU. The same is true for site-2 where we have separate radio access network and a connection to another VR-streaming and a separate monitoring system, analytics engine, decision engine and actuator. The application sampling function and radio access network sampling function also collect the corresponding metrics at radio access network and application level. In addition, we have an aggregation server located in site-0 that is used for federated learning purposes. As can be seen here, federated learning updates between site-1 and site-2 are performed through this aggregation server, aggregating the models in AE-1 and AE-2 to learn different number of VR video streaming user patterns for each site. Finally, after analytics engine and the decision engine generate a decision over the online monitoring system data based on the common model built, the actuator performs proactive CPU scaling on VR-streaming servers to improve the quality of experience of VR-streaming clients.

Measurements of the BS and VR video streaming server have been done via MS to be able to feed data into FL training process as well as for online inference.

Experiments with multiple VR video streaming users: In the first part, an increasing number of VR video streaming users has been used as a video receiver. All experiments have been carried out with the 5G network. VR-streaming users are connected to VR-streaming NGINX server in order to receive the video using the 5G network. The VR video streaming users made the query to the VR video streaming server simultaneously, so they received the video at the same time. This procedure has been performed in both site-1 and site-2. During the video playback, the MS has been running on the edge to capture the transmitted and received packets that go through the network. These experiments have been carried out to demonstrate that the network created can support a greater number of VR video streaming users connected at the same time. It has been also wanted to prove that the video can be played smoothly on all connected VR video streaming users (until 8).

To make this experiment, first, the number of VR-streaming clients using VR-streaming server over the 5G network is increased until VR-streaming clients observe streaming problems due to high CPU consumption at VR-streaming server. Later, the tests have been carried out again with second FL site but this time the number of connection VR-streaming users to the second VR-streaming server follows a different pattern. Figure 4-21 shows the patterns followed in each FL sites, namely FL site-1 and site-2.

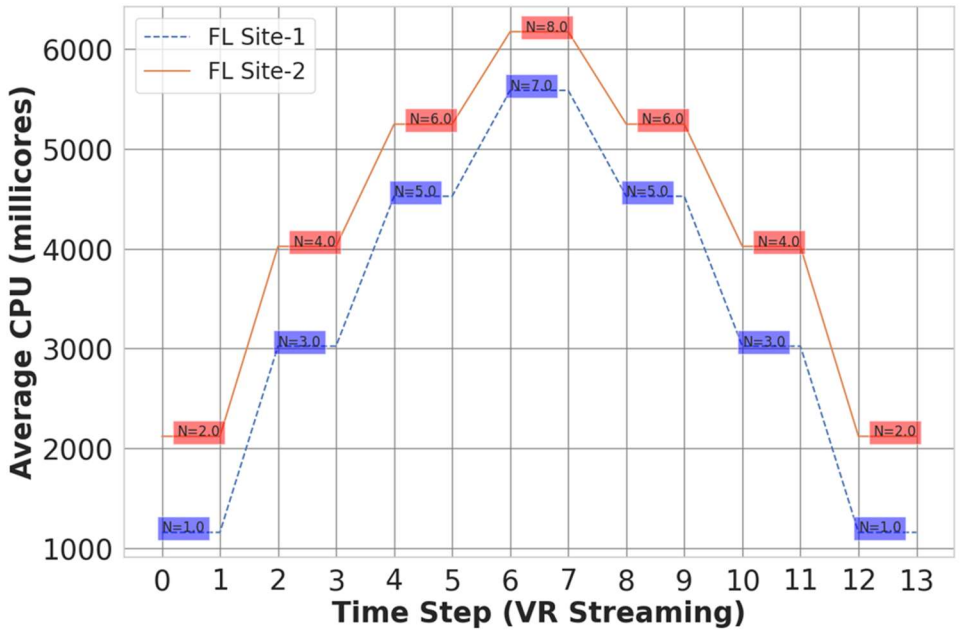


Figure 4-21: Different patterns of number of VR-streaming users for each sites used for experiments

By doing these tests for training, it has been possible to know the CPU consumption that can be reached using these two different patterns and training the FL model. Once the CPU consumptions that the VR-streaming server can reach has been verified, video playback experiments have been possible to start in order to test the network.

Metrics: The metrics that has been studied to analyse network and application behaviour are bit rate at BS and CPU usage at application server pod. These metrics have been chosen because of the use case being studied. When multiple VR video streaming users want to watch a video in real-time, the important parameters usually are the amount of exchanged data rate and the response time of the server (related to CPU availability). The VR video streaming users expect the video to be displayed smoothly. For this experiment, not only application-level pod measurements are taken into account but also the radio level parameters (bitrate observed at BS). This parameter is in direct proportion with the number of VR-streaming users connecting to VR-streaming client using the mobile 5G communication. With the MS software deployment, it is possible to take measurements of these values on the application pod and BS where the RAN is hosted to know how the many bits are sent by the VR video streaming clients when they arrive or leave sequentially.

4.3.2.2 EVALUATION OF KPIS OVER EXPERIMENTAL PLATFORM

As described in [MONB5GD33], 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., [MONB5GD33], and [MONB5GD5.4]). The target KPIs of this solution are also described in Table 4.

Infrastructure and MonB5G Components

In our demonstration setup, we have three sites. Each site has a radio access network domain that includes a base station, core network and VR video streaming server and corresponding VR video streaming users. Each radio access network domain is connected to the application domain through core network where the VR video streaming server is running and the MonB5G components, monitoring system, analytics engine, decision engine and Actuator are deployed near the VR Stream server. The sampling functions for the application and radio access network are each performed at the location of the VR-streaming server pod application and the base station respectively. So, sampling functions collect data to monitor radio access network and application related data at site-1.

The same is true for site-2 where we have separate radio access network and a connection to another VR-streaming and a separate monitoring system, analytics engine, decision engine and actuator. The application sampling function and radio access network sampling function also collect the corresponding metrics at radio access network and application level.

In addition, we have an aggregation server located in site-0 that is used for federated learning purposes. As can be seen here, federated learning updates between site-1 and site-2 are performed through this aggregation server, aggregating the models in AE-1 and AE-2 to learn different number of VR video streaming user patterns for each site.

Finally, after analytics engine and the decision engine generate a decision over the online monitoring system data based on the common model built, the actuator performs proactive CPU scaling on VR-streaming servers to improve the quality of experience of VR-streaming clients. All of these are implemented in separate Kubernetes clusters and each MonB5G component runs as separate container orchestrated by Kubernetes.

All MonB5G and infrastructure related components are below and will be running in K8s environment in final demo. In the infrastructure, a VR video 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 and identically distributed (i.i.d) in distribution.

VR video streaming client is the application component that enables users to enjoy virtual reality experiences. This component is designed to emulate the user environment (UE) in a simulated environment known as Simbox. The VR video streaming client is implemented as a streaming client within Simbox, which allows users to stream VR content seamlessly. It can also run on a regular laptop.

VR video streaming server is an infrastructure-related component that is designed to be deployed in a Kubernetes (k8s) cluster. It supports scalability by being able to scale up or down based on commands received from the FL AE client and the top orchestrator. Deploying the VR video streaming Server in a Kubernetes cluster enables efficient scalability, fault tolerance, resource utilization, and integration with the FL AE client and top orchestrator, facilitating the management and operation of the VR video streaming infrastructure for federated learning applications.

Monitoring System is an infrastructure-related component that is responsible for collecting RAN (Radio Access Network) and server-related parameters in a distributed environment. The purpose of the MS is to monitor the performance and health of the system by gathering data from different FL clients (two in our case). It plays a critical role in collecting RAN and server-related parameters from multiple FL clients in a distributed environment. By leveraging Kafka for data streaming and enabling real-time analysis, the MS provides valuable insights into the performance and health of the infrastructure, aiding in the efficient management of the FL system.

Core Network is an infrastructure-related component that represents the control elements in the system. Specifically, it refers to the Amarisoft Core, which is used to manage the control plane of the network. The Core Network component is responsible for controlling and coordinating the data plane elements,

FL aggregation server: This element is used to exchange weights between FL clients. This is implemented in Docker container and runs in K8s. The deployment details are as follows: deployment is specified in the corresponding yaml file with a "kind" of "Deployment". It has metadata that defines the name of the Deployment as "fl-server-pod". The desired states of the Deployment are as follow: there should be one replica of the pod, and it should use a container named "fl-server-pod" that is based on the container image “server-demo-with-policy”. The container exposes port 8081 and mounts a volume named "log". The container also defines a number of environment variables, including "NUMBER_ROUNDS" and "NUMBER_SELECTED_CLIENTS" and a second container called "filebeat", which is based on the image "elastic/filebeat:7.16.3" and mounts the same volume "log". Inside the same yaml file, the Service used is specified with the "kind" of "Service" and metadata with the name of the Service as "fl-server-pod". The "spec" section specifies that the Service should be of type "NodePort", which exposes the Service on a static port on each node in the cluster. The Service exposes port 8081 and maps it to the target port 8081 of the Deployment. The Service has a nodePort of 30000 and an external IP of 10.0.42.7. The Service also has a selector that matches the labels of the Deployment, which include "run: fl-server-pod", "app: fl-client-1-pod", and "app: fl-client-2-pod".

The ISTIO_META_POD_NAMESPACE environment variable is used in the context of the Istio service mesh, which provides a way to manage traffic between services in a Kubernetes cluster and specifies the namespace in which the pod is running, which can be useful for Istio to identify the location of the pod and route traffic accordingly. When a pod is created in Kubernetes, it is assigned to a namespace, which is a logical partition within a Kubernetes cluster. The namespace provides a way to organize and manage resources within the cluster. The metadata.namespace field of the pod contains the name of the namespace that the pod is assigned to.

Overall, the above configuration provides a way to run and expose an aggregation server application that can be accessed by other applications in the cluster. The use of environment variables and the second container for logging allow for additional configuration and monitoring capabilities.

FL AEs: 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. In the Kubernetes YAML files, a deployment and a service for a client application named "fl-client-1-pod" and a separate file "fl-client-2-pod" are described. Here's a breakdown of its functionalities:

Deployment Section:

- apiVersion: This field specifies the version of the Kubernetes API being used.
- kind: This field specifies the type of resource being defined, in this case, a deployment.
- metadata: This field contains metadata for the deployment, such as its name.
- spec: This field contains the deployment's specifications, including the number of replicas and the template for the pods.

Selector Section:

- matchLabels: This field specifies a label selector for the pods, ensuring that only pods with a matching label are managed by the deployment. The selector selects which pods are targeted by this deployment.

Replicas Section:

- replicas: This field specifies the number of pod replicas to be created by this deployment.

Template Section:

- metadata: This field specifies the labels for the pod.
- labels: This field specifies the labels for the pod.
- spec: This field specifies the pod's specifications, including its containers and volumes.

Container Section:

- name: This field specifies the name of the container.
- image: This field specifies the image to use for the container.
- ports: This field specifies the ports to expose from the container.
- env: This field specifies environment variables for the container.
- volumeMounts: This field specifies the volumes to mount in the container.

Volume Section:

- name: This field specifies the name of the volume.
- configMap: This field specifies the configMap to use for the volume.

Service Section:

- apiVersion: This field specifies the version of the Kubernetes API being used.
- kind: This field specifies the kind of resource being defined, in this case, a service.
- metadata: This field contains metadata for the service, such as its name.
- spec: This field defines the desired state for the service, including:
    - The type of the service (NodePort in this case).
    - The ports that will be exposed by the service.
    - The external IP address(es) that will be assigned to the service.
    - The selector that selects which pods are targeted by this service.

Service Selector Section:

- selector: This field specifies the label selector for the service, ensuring that the service only targets pods with the matching labels.

In general, the common yaml files start with the definition of a Deployment, which is a Kubernetes resource that defines a set of replicas for a pod. In this case, the Deployment is named **fl-client-1-pod (or fl-client-2-pod)** and specifies that there should be one replica of the pod. The **selector** field in the Deployment specifies the labels used to match the pod. In this case, the **run** label is set to **fl-client-1-pod (or fl-client-2-pod)**, which means that the pod's **metadata.labels.run** field will match this label. The **template** field in the Deployment specifies the configuration for the pod. It includes metadata for the pod's labels, which are set to **run: fl-client-1-pod (or fl-client-2-pod)** and **app: fl-server-pod**. The **containers** field in the pod specifies the container that should be run in the pod. In this case, there is one container named **fl-client-1-pod (or fl-client-2-pod)**, which is defined using the **"fl-client-1" (or "fl-client-2")** image. This container is configured to listen on port **2051** and has a number of environment variables that configure its behaviour, including the URL for the server, the location of a CSV file, and the size of the batch of data to send to the server for each round of training. The container also mounts a volume called **data** at the path **/var/data**, which is used to store the CSV file. The **volumes** field in the pod specifies the volumes that should be mounted in the container. In this case, there is one volume named **data**, which is a ConfigMap that contains the CSV file. The file also defines a Service, which is a Kubernetes resource that exposes the pod to the network. The Service is named **fl-client-1-pod** and is of type **NodePort**, which means that it exposes the pod on a port that is accessible from any node in the cluster. The Service listens on port **2051** and forwards traffic to port **2051** on the pod. The **externalIPs** field in the Service specifies the external IP address of the pod. In this case, it is set to **10.0.42.2 (or 10.0.42.3)**. The **selector** field in the Service specifies the labels used to match the pod. In this case, it is set to **run: fl-client-1-pod (or fl-client-2-pod)**, which means that the Service will forward traffic to pods with the **metadata.labels.run** field set to **fl-client-1-pod (or fl-client-2-pod)**.

K8s is used as the main orchestrator to manage the cluster and its containerized applications, including components related to MonB5G and Infrastructure. Kubernetes provides a robust and flexible platform for managing containerized applications, and its features make it well-suited for orchestrating complex systems like MonB5G and Infrastructure components.

Top Orchestrator (DE) interacts with the Kubernetes orchestrator to manage the scaling of the VR video streaming server based on certain rules. It is responsible for making decisions regarding the scaling of the VR video streaming server. It likely monitors specific metrics, such as CPU utilization, and applies simple rules to determine when to scale the server. The DE communicates with the Kubernetes orchestrator to issue scaling commands based on its decisions. FL AE clients are connected to the top orchestrator via Kafka to enable the top orchestrator to receive data updates from the FL AE clients in real-time. The data published by the clients include relevant metrics or information used by the DE to make scaling decisions for the VR video streaming server. Overall, by allowing the FL AE clients to publish data into Kafka, the top orchestrator (DE) can make scaling decisions based on predefined rules. The DE then communicates with the Kubernetes orchestrator to scale the VR video streaming server as needed, ensuring optimal resource allocation and performance.

Dataset: Considered features from MS used in FL are as given in table below:

Table 9: Dataset features and output used in FL

	Feature	Description
Features	**Current Bitrate**	Includes the instantaneous bit rate observed at gNodeB
	Previous CPU Load	Previous Time Window CPU resource consumption
	Current CPU Load	Current Time Window CPU resource consumption
Output	**Next CPU Load**	Next Time Window CPU resource consumption

We use RAN and application related parameters as an input and CPU as output to build a ML model using federated learning approach. Our training dataset consists of 2400 samples obtained from the virtual reality streaming server and the gNodeB. The input features are the current bitrate, which is the instantaneous bit rate observed at the gNodeB when VR video streaming is running, the previous CPU load, which is CPU resource consumption of the previous time window for VR video streaming server and the current CPU load, which is the CPU resource consumption of the current time window for VR video streaming server. Finally, our training dataset output is the next CPU load, which is CPU resource consumption of the virtual reality streaming server in the next time window.

Working principles of the overall system:

The experiments will run in two phases: (1) Offline training and (2) Online Inference. **Offline training phase:** Figure 4-22 shows the offline training phase of the experiment setup for policy-based FL approach.

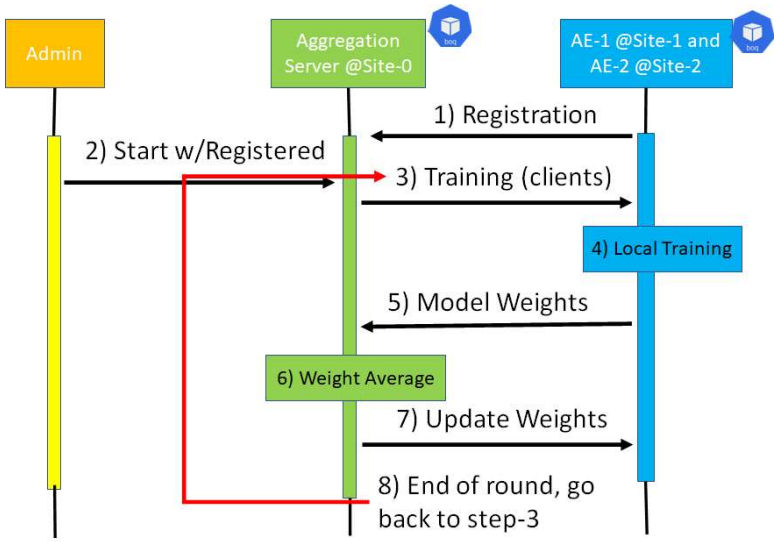


Figure 4-22: FL training steps

STEP 1: REGISTRATION: All clients register with their IP address in the server node as shown in Figure 4-22 below. 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 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. Each local MS collects the relevant measured KPIs from RAN average bitrate) and VR video streaming server ((previous and current time window CPU). Each local FL client grasps those data.

STEPS 3: TRAINING: Server sends request to registered clients and start FL training. This request is made asynchronously. 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.).

STEPS 7-8-9: COMPUTE & UPDATE WEIGHTS: Model weights of each FL client are sent to the server through when each FL AE finishes the training, and then Server averages the weighs and update the weights of the clients & repeat same procedure next FL rounds (GO TO **STEP 3**).

Finally, all those rounds are visualized via Kibana dashboard (which is detailed below) and final models are stored in both local nodes inside docker containers.

During **online inference phase**, the process is initiated by the admin and inference is done periodically**.** Note that in this phase, MS constantly streams relevant data (related to VR video 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 video 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 video streaming server based on the predicted CPU. K8s orchestrator as an actuator scales up/down the VR video streaming server.

Data visualization: To visualize data during Federated Learning (FL) training, we utilized the Sidecar Container pattern on Kubernetes to install Logstash for log aggregation. Log aggregation is crucial for monitoring system failures. When running FL on Kubernetes, the logs belong to a single Pod, and if the Pod is deleted, the log is lost. Hence, we require a log aggregation system to track system failures. We use the ELK stack (Elasticsearch, Logstash, and Kibana), and to collect logs on each Pod, we use a Sidecar Container.

SideCar Container: The Sidecar Container is a separate container that performs the log collection process instead of implementing it on the application containers. This approach avoids affecting the performance of the application containers. Sidecar containers run alongside the main container in the Pod and extend and enhance the application containers in many ways.

Logging with Logstash: Logstash's original task is to monitor logs and transform them into a meaningful set of fields, and then stream the output to a defined destination. However, it can have performance issues. Elastic has launched Filebeat, which monitors logs and streams the output to a defined destination. We are using Filebeat (e.g., instead of FluentD or FluentBit) because it is an extremely lightweight utility and has a

support for Kubernetes and also more suitable for production level setups. Logstash acts as an aggregator that ingests data from various sources, transforms it, and sends it to Elasticsearch.

To deploy our system, we first deploy a Pod with an fl-aggregation-server container that writes logs to the file /var/log/access.log. Then, we deploy a Sidecar Container on the same Pod that runs Filebeat to collect logs and output them to Logstash. This process is illustrated in Figure 4-23 below showcasing the components where sidecar container on aggregation server collects log and sends to Logstash to be visualized via Kibana. The steps of Figure 4-23 are given in Appendix B of the document.

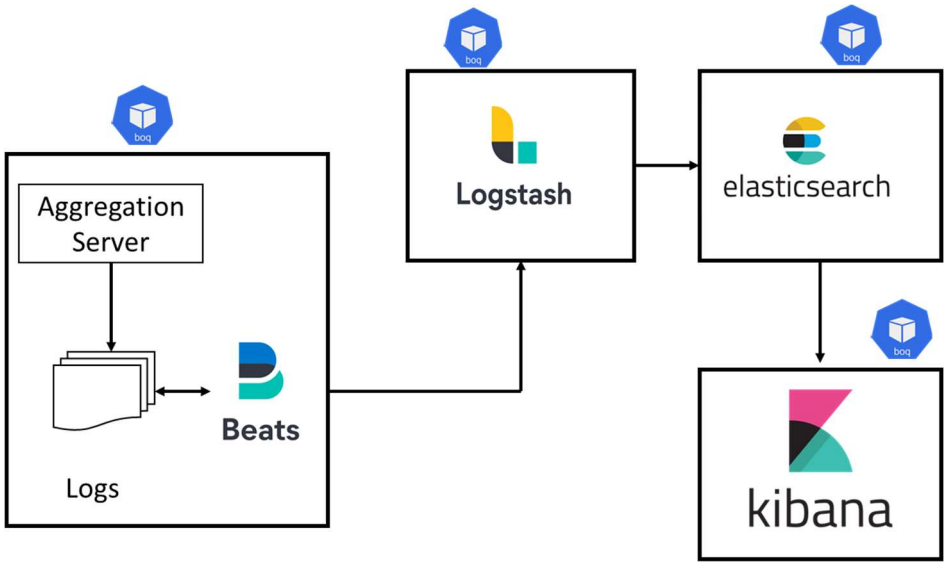


Figure 4-23: Sidecar container on aggregation server to collect log and visualize

To sum up, it has been used the Sidecar Container pattern to set up log collection for Pod, using ELK log stack. Finally, all yaml files described above can be found in Appendix A of this document.

Experimental Results Analysis

In our analysis, we use the dataset provided in Section 4.2. Figure 4-24 below shows the outbound bit rate of container, O in Mbps as the number of VR video streaming users increases. As the number of users increases, the bit rate out of pod also tends to increase, but not necessarily in a linear fashion. When N = 1, the median value of bit rate out of the pod is 8.69 Mbps, when N = 8, the median value bit rate out of the pod is 58.03 Mbps. The largest increase in bit rate out of the pod occurs between N = 1 and N = 2, where it jumps from 8.69 to 16.42 Mbps. This represents an increase of approximately 90%. After N = 1, the bit rate out of the pod still increases as N increases, but at a slower rate. For example, between N = 6 and N = 7, the bit rate out of the pod increases by only about 8%, whereas between N = 1 and N = 2, it is approximately 90%. From these observations, we can conclude that there is a positive correlation between bit rate out of the pod and the number of users. As more users access the system, the bit rate out of the pod increases, likely due to the increased traffic on the system. We can also see that the rate of increase in O appears to slow down as N increases. However, there is no clear pattern or trend in the relationship between N and bit rate out of pod after N=6. The dispersion of value for each number of VR video streaming users is not the same. There are also some points where the outbound bit rate out of the pod (O) appears to decrease or stay the same even

as the number of users (N) increases, which could be due to a variety of factors such as RTSP protocol behavior, system capacity limitations, or other device level factors.

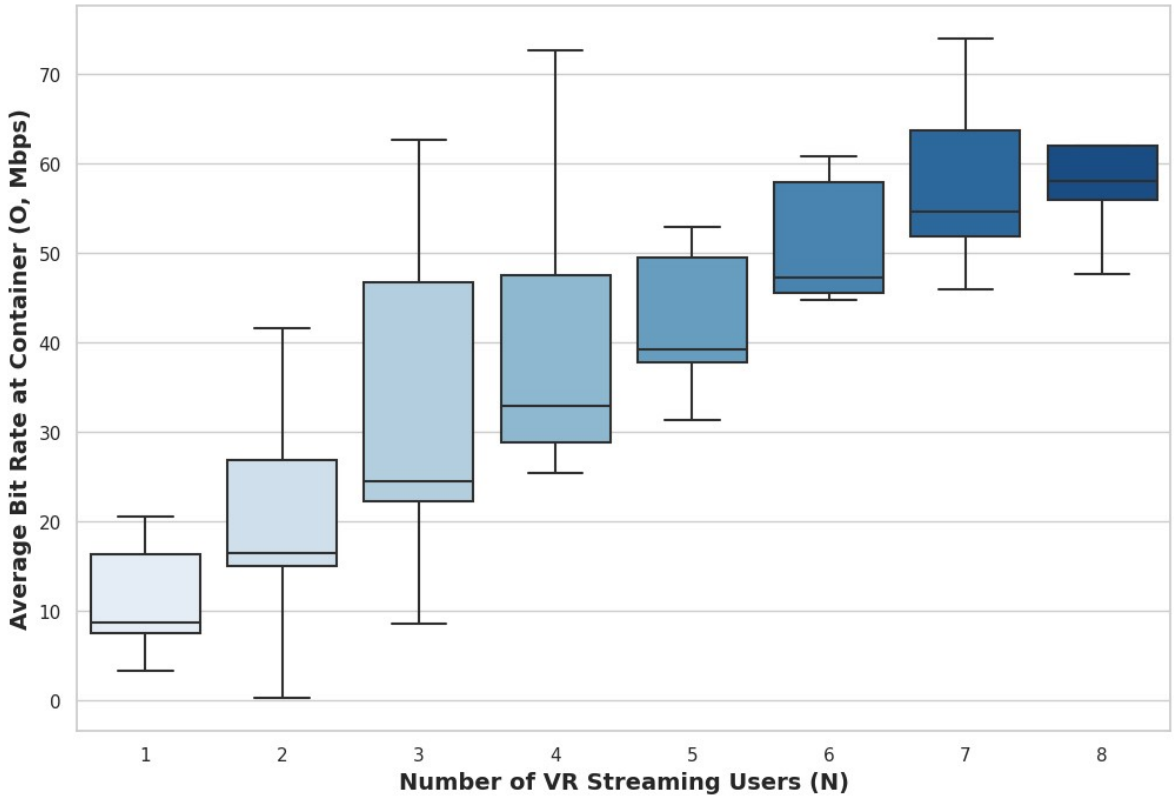


Figure 4-24: VR video streaming server outbound traffic versus number of VR video streaming clients

To analyze the relationship between O and N, using the median values we can use regression analysis to find a model that best fits the data. Here, we can use a simple linear regression model:

$$O = m*N + b$$

where m is the slope of the line (which represents the rate of change of O with respect to N), and b is the y-intercept (which represents the starting value of O when N = 0).

To fit the model, we can use the least squares method to find the values of m and b that minimize the sum of the squared differences between the actual values of O and the predicted values from the model. Here are the results of the linear regression analysis for the data in Table 10.

Table 10: Parameters of regression analysis model

Slope (m): 4.2354
Y-intercept (b): 1.4221
R-squared: 0.9643

The slope of the line (m) is positive, indicating that as the number of users increases, the bit rate out of pod also increases. Specifically, the model predicts that for every additional user, the bit rate out of pod increases by approximately 4.24 Kbps. The y-intercept (b) is also positive, indicating that even with zero users, there is still some bit rate out of the pod (about 1.42 Kbps in this case). The R-squared value of 0.9643 indicates that the model explains 96.43% of the variation in the data, which suggests that the linear relationship between O and N is a good fit for the data. However, it's worth noting that there may be other factors that influence the bit rate out of pod that are not captured by this model. Overall, our experimental analysis suggests that the bit rate out of pod increases as the number of users increases, and that this relationship can be approximated by a linear model. However, it's important to keep in mind that the actual bit rate out of pod may be influenced by other factors not captured by this model.

Figure 4-25 below shows the CPU load of container, C in percentage as the number of VR video streaming users increases. As the number of users increases, the CPU utilization also tends to increase. When N = 1, the median CPU utilization is 5.49% and when N = 8, the median CPU utilization is 48.01%. The largest increase in CPU utilization occurs between N = 1 and N = 2, where it jumps from 5.49% to 5.49%. After N = 1, the CPU utilization still increases as N increases, but at a slower rate. Moreover, we observe no clear pattern or trend in the relationship between N and CPU utilization after N=6.

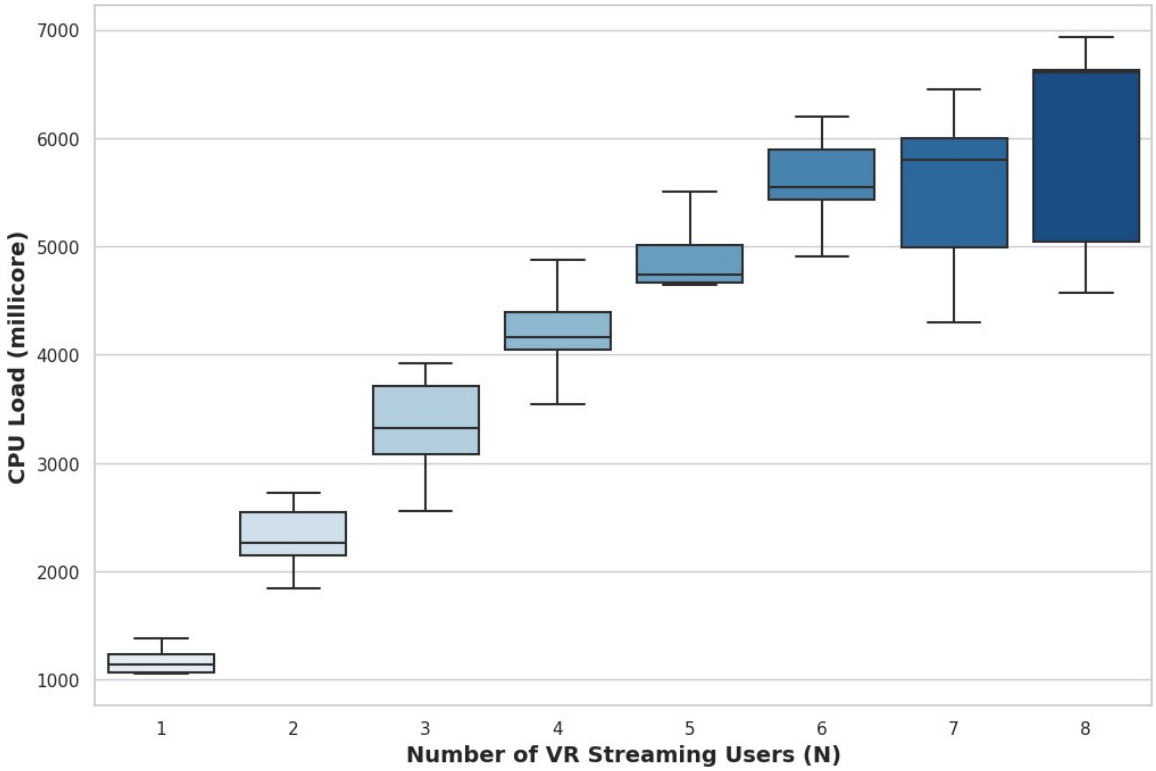


Figure 4-25: VR video streaming Server CPU load versus number of VR video streaming clients

Finally, Figure 4-26 shows pair plot between VR video streaming Server CPU load (C) and outbound bit rate of pod (O) with respect to number of VR video streaming clients (N). There seems to be a positive relationship between CPU utilization (C) and outbound bit rate out of pod (O). However, this relationship is not perfectly

linear, and there are several points that deviate from the general trend. At low levels of CPU utilization, the bit rate out of pod appears to be relatively low and stable. As the CPU utilization increases, the bit rate out of pod also tends to increase. However, at very high levels of CPU utilization, the bit rate out of pod seems to plateau and not increase much further, despite the increase in CPU utilization. It is also worth noting that there are some points where the CPU utilization is relatively high, but the outbound bit rate out of pod is low. These points could be due to other factors that affect the outbound bit rate, such as network latency, RTSP protocol or other external factors.

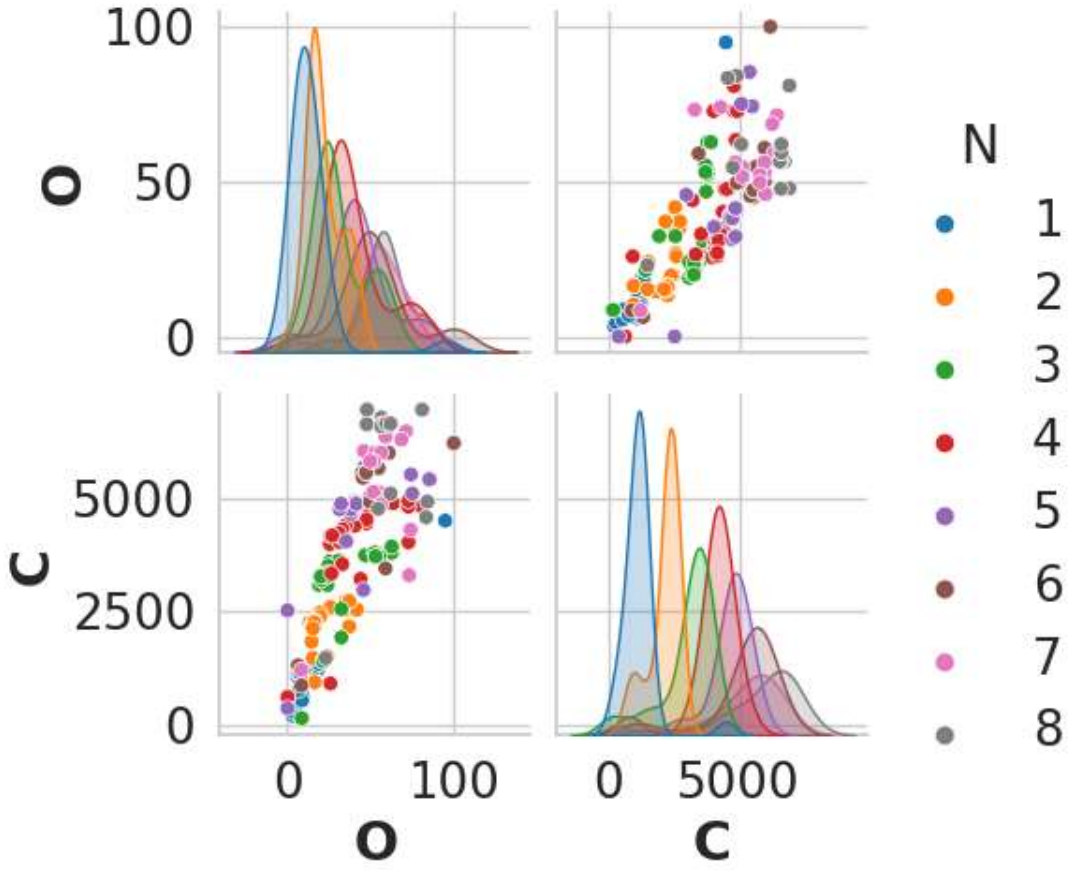


Figure 4-26: Pair Plot.

Energy consumption modelling in user plane

The authors in [Tadesse2017], have used RCE PM600 power meter to read real-time power statistics. The aim is to measure the power consumption associated with Docker containers when they run several basic test applications, from CPU-intensive computations to high-rate network transfers. The power consumption is characterized as the load varies, to build a model that can be used to estimate the power footprint of more complex applications.

We use the linear fitting formula in [Tadesse2017], and the linear regression law is as follows:

P = 0.1065c + 12.4411,

where P is the power (in watt) and c is the percentage of used CPU (e.g., 200 for two cores).

Figure 4-27 shows the power consumption at container at the user plane (UP) versus number of VR video streaming clients. From the above equation, as C increases, the power consumption (P) also increases. This is expected, as the CPU utilization requires more power to process data. Additionally, we can see that there are diminishing returns in terms of performance gains as the CPU utilization increases. This is indicated by the fact that power consumption continues to increase, but at a decreasing rate as we move from N=1 to N=8. It is also worth noting that the power consumption increases at a non-linear rate, as indicated by the fact that the change in power consumption is not proportional to the change in CPU utilization.

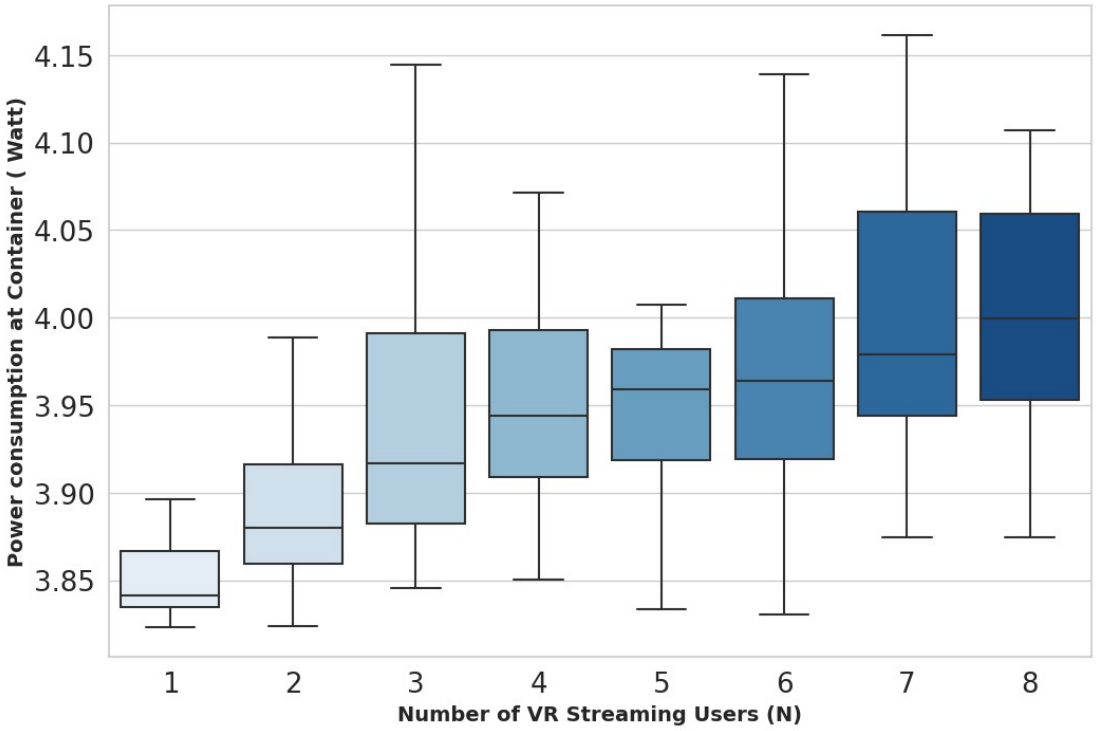


Figure 4-27: Power Consumption at container at UP versus N

Federated Learning Training Results

Figure 4-28 shows FL Training phase, graph (a) presents the average NMSE versus number of FL rounds, while graph (b) presents the average computation time versus number of FL rounds. According to NMSE and average computation time versus rounds, the convergence occurs at round number 8. Since the dataset in experimental setup was not big (only 2400 rows) and there were only two FL clients due to limitations of testbed, the convergence is observed to be fast.

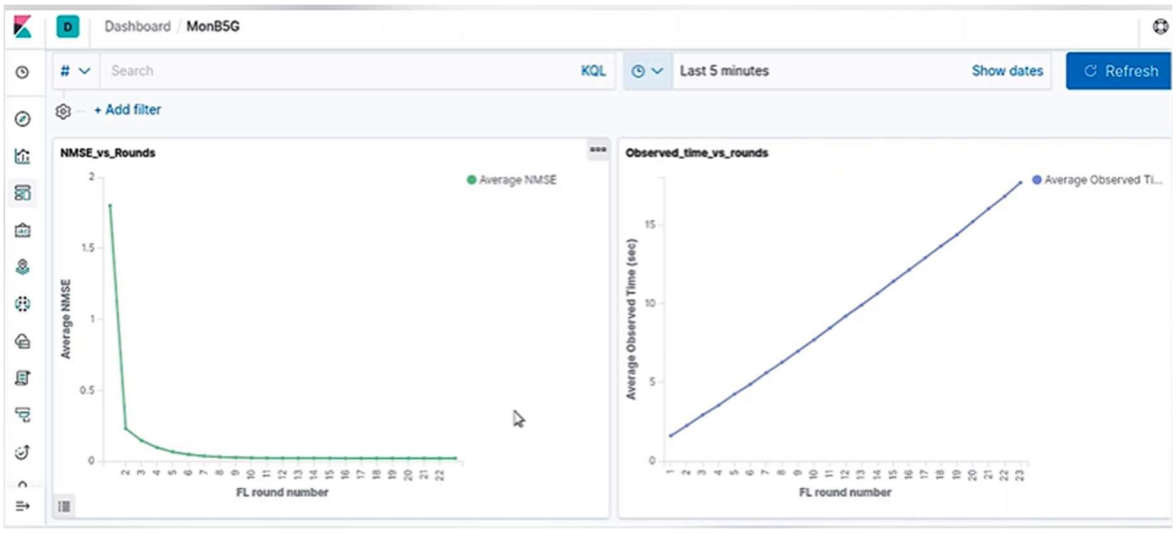


Figure 4-28: FL Training phase (a) average NMSE (b) average computation time

Overhead Gain Calculation Management and Orchestration

Table 11 shows the overhead induced by the baseline *fully centralized MANO deep learning (CCL)* [Chergui2020] and the *Statistical Federated Learning based resource predictor and scaling*. For the computation of the overhead, we have considered that both the datasets and update models are coded in 32 bits. In the uplink between the clients and the aggregation server, the approximate monitoring overhead can be calculated as in [MONB5GD3.3]. Starting from the convergence point of *Federated Learning based resource predictor and scaling* at round 8, more than 10 times overhead reduction (approximately 11.11 times) is obtained in comparison with the centralized SLA-constrained algorithm.

Table 11: Monitoring overhead comparison between centralized solution and FL-based algorithms

Rounds	2	5	8	20	30
Monitoring overhead CCL (KB)	52				
Monitoring overhead FL-based resource predictor and scaling (KB)	1.152	2.880	4.608	11.152	17.28
Monitoring Overhead Gain	**x45**	**x18.02**	**x11.2**	**x4.66**	**x3.0**

Energy Consumption Computation in Management and Orchestration Plane

Using the same approach as in [MONB5GD54] to calculate the energy and using the modelling formulas and considering that convergence has been achieved at 8 iterations in our experimental case, x11.2 energy gain is achieved in comparison to CCL as shown in Table 12.

Table 12: Energy consumption comparisons

Rounds	2	5	8	20	30
Energy CCL (mJ)	125.6				
Energy FL-based resource predictor and scaling (mJ)	2.7	6.9	11.1	26.9	41.75
Energy Gain	**x45**	**x18.02**	**x11.2**	**x4.66**	**x3.0**

Note that observed energy consumption in AI/ML tasks is negligible compared to other operations such as VR video streaming server (simultaneous transcoding and streaming). The reason is that VR video streaming processes more data than the one to be transmitted in the network by CCL and FL based resource predictor and scaling. Note that MonB5G MS samples features of the dataset (e.g., CPU consumption, bit rate at BS, etc), whereas VR video streaming server processes user plane data containing real-time video with very high data volume. Consider additionally that in case MS observes higher dataset, energy consumption of both CCL and FL-based resource predictor and scaling will increase. But in our analysis, the dataset size collected to achieve low NMSE was not big, so the consumption of energy was very low.

MonB5G solutions target management and orchestration energy and monitoring overhead reduction. On the other hand, the energy consumption reduction in the user plane is not in the scope of MonB5G. Therefore, any improvements on VR video streaming server to reduce its energy consumption is not considered in this deliverable.

Discussions on experimental analysis

According to modelling approach performed in [MONB5GD54], the main consumption of energy was originating from utilization of computational resources rather than the link energy transmission itself. As a matter of fact, the main factor limiting the energy consumption becomes the computing process in the container, and the impact of the link transmission energy and overhead is small or can be considered almost negligible. In the results presented above, there is a linear relationship between the CPU load and energy consumption. The results are in consistent with those presented in [MONB5GD54].

Note that during experiments, there may have been some anomalies or outliers in the data, possibly due to synchronization problems or the presence of an additional link in the setup (UE emulator to remote VM). These factors could introduce randomness or variations in the observed results. In addition, they can also be caused by RTSP protocol. However, we can still observe clusters in the dataset which shows identifiable patterns or groupings in the data, that could provide insights into the behaviour of the system.

4.3.3 Slice KPI Prediction with Interpretable Multivariate Anomaly Detection

4.3.3.1 USE-CASE DESCRIPTION AND BASELINE

We propose a Graph-based Interpretable Anomaly Detection (G5IAD) reference architecture as shown in Figure 4-29 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.

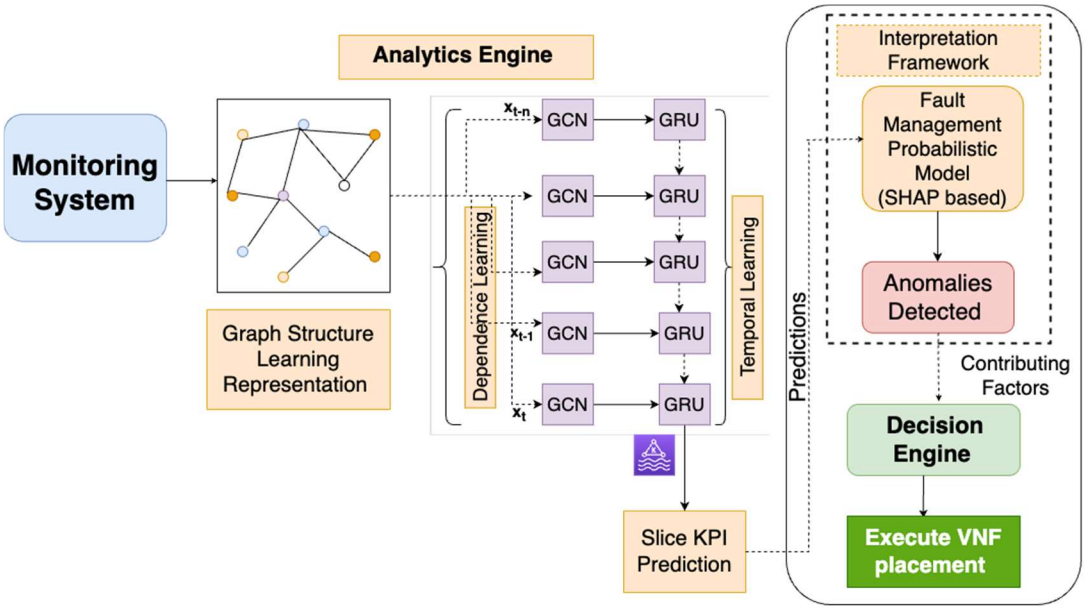


Figure 4-29: Analytics Engine reference Architecture

Figure 4-30 represents the performance of the compared algorithms trained using multivariate KPIs to predict slice latency KPI (normalized values) in a multivariate network data environment [D3.3], [Graph2023]. It was observed that compared to different models, the model with Graph GCN generalizes well over the training data as compared to the other models. The key idea behind GNNs is to aggregate feature information of nodes' local neighbors via neural networks.

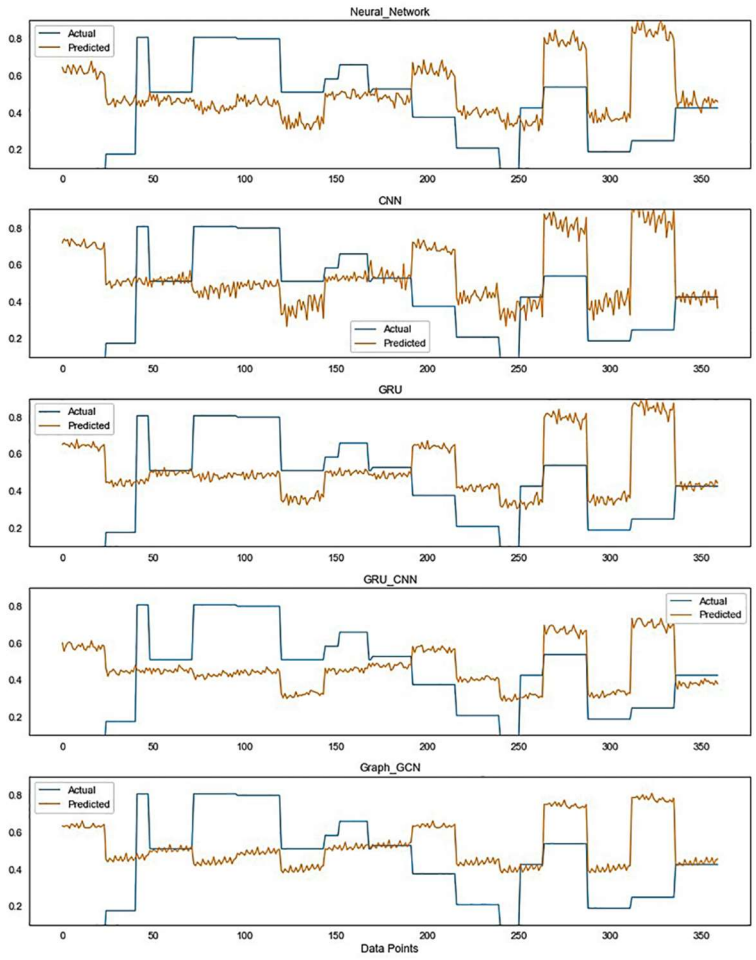


Figure 4-30: Prediction Comparison among Different Methods

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. Our results show that our AE architecture (joint learning framework based on GCN combined with GRU based architecture trained using network slice KPI data) helps to optimize the modelling of KPI data. The trained probabilistic model detects anomalies in slice latency KPI. The time taken for GRU based model (best tuned model with regularization technique, batch size = 32, Dropout 30% and callback mechanism technique) is approximately **20.42%** faster than LSTM due to faster convergence in training. When given an anomalous KPI sequence, the model detects the anomalies with lower probability scores.

4.3.3.2 EVALUATION OVER KPI EXPERIMENTAL PLATFORM

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 as shown in Figure 4-31). This information is taken as input by the operators and / or the DE.

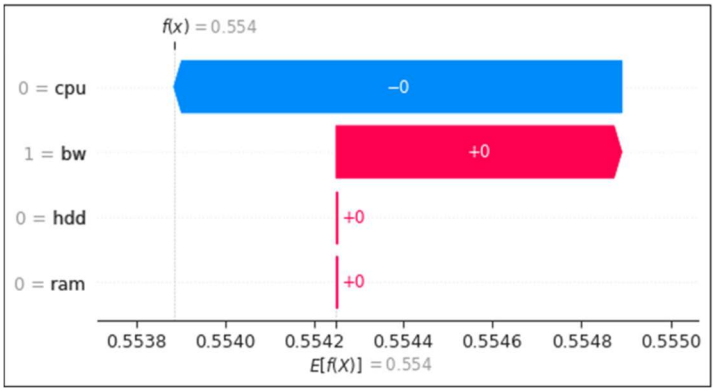


Figure 4-31: Interpretable Anomaly Detection depicting contributing factors.

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 4-32. 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. This enabler addresses the "Improve the accuracy of the AE/DE mechanisms for detection of slice performance degradation" KPI.

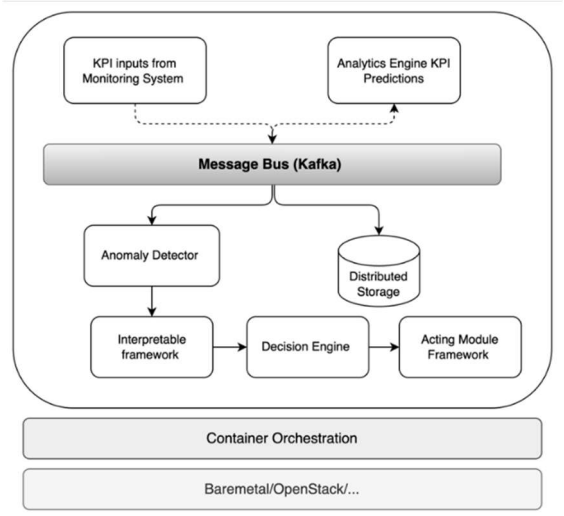


Figure 4-32: Cloud Implementation of the AE deployment

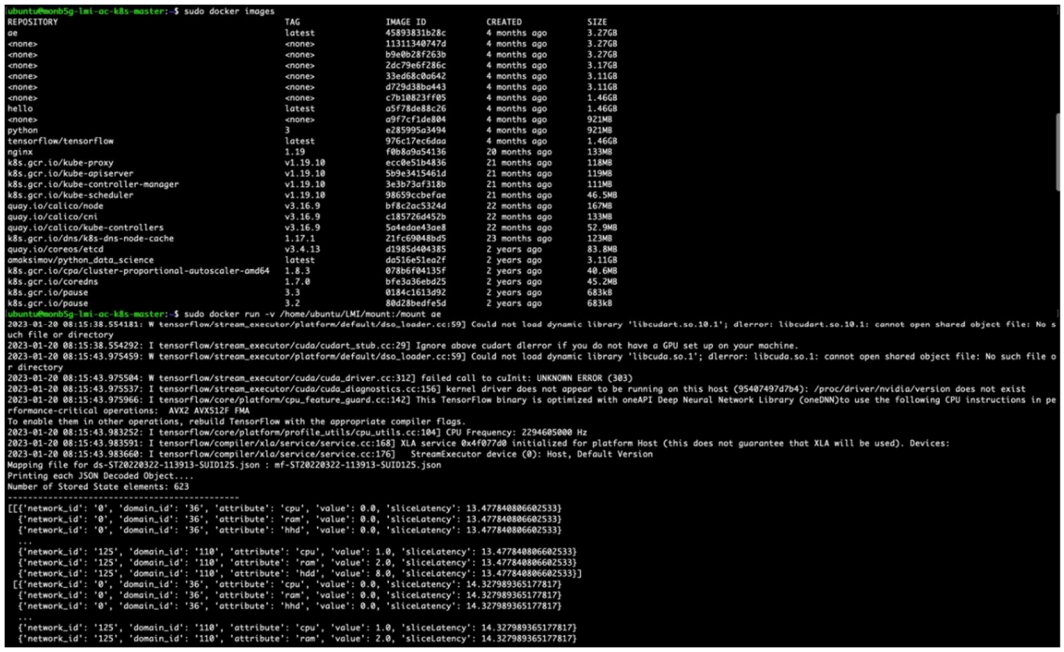


Figure 4-33: AE deployed and running in the testbed (CTTC)

In PoC1 scenario 1, we showcase the implementation of multivariate anomaly detection using an intelligent neural network-based algorithm powered by MonB5G distributed framework through Docker deployment as shown in Figure 4-34. After training the model with the dataset as mentioned in Section 7.2, using normal instances across multiple variables, Prometheus and Grafana are utilized to monitor and visualize the model's performance. Finally, the trained model will be employed to detect anomalies in unseen test samples.

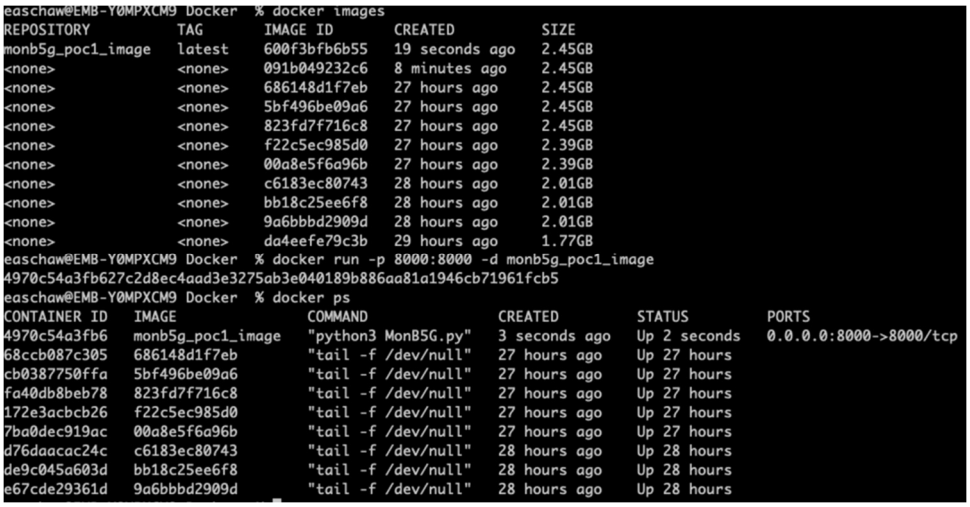


Figure 4-34: Docker deployed and running.

When Grafana dashboard is investigated, the normalized unseen test sample is depicted, showing CPU usage at the server, average CPU usage at the server, Aggregate downlink bit rate, the instant aggregate downlink bit rate at gNB (R) and Outbound traffic at the server, average outbound traffic (O) flowing from the data interface of the video server in Figure 4-35.

It can be observed that the first anomaly was detected at the same time step across all the variables, which was caused by the activation of 8 UEs at a period when only 1 UE was expected in the normal pattern. This resulted in the generation of anomalies in the three variables, with higher R, O, and CPU than expected depicting multivariable anomalies. For the rest of the time, there is no anomaly detected as the data is very similar to what has been trained. By proactively detecting the anomalies across the various NS measures, this enabler addresses the overall reaction time in face of anomalous behavior, and therefore addresses the "reduction of time between NS malfunction and anomaly detection" KPI. Multivariate-based Anomaly Detection detecting anomalies among the multitude of network resources (CPU usage at the server, Outbound traffic at the server, Aggregate downlink bit rate) at the same time.

Figure 4-35: Multivariate Anomalies been detected by the model (CPU, O, R at same time)

4.3.4 LSTM-Based Anomaly Detection

4.3.4.1 USE-CASE DESCRIPTION AND BASELINE

Anomaly detection mechanisms are used to identify abnormal states in the network. While the abnormal state does not necessarily indicate the network fault, its early detection is essential to maintain the stable network operation e.g., via taking suitable corrective action (e.g., slice/VNF reconfiguration, scaling, etc.). It must be noted, that in the Anomaly Detection process, time of reaction is of prime concern as it allows to mitigate the aggravation of potential network slice malfunctions and performance degradation. In a centralized network architecture, management system can detect and solve anomalies only after the specific metrics (PIs, KPIs) reach the central component, i.e., the time needed to repair the network is increased by

the RTT between the component and management system. Moreover, typically, due to large volume of transferred data, the metric collection frequency is significantly reduced to conserve the bandwidth further increasing the delay. On the other hand, the MonB5G approach, leverages decentralized architecture with relevant management components (MS, AEs, DEs, ACTs) deployed at the slice-level, or at the domain level (RAN, MEC, cloud) to enable more swift reactions to changes.

In this test, LSTM-based Anomaly Detection AE (further referred to as AD, originally described in [MONB5GD32]) is deployed in RAN domain and used to detect anomalous changes in traffic originating from a streaming server in the Cloud and consumed by several UEs. As previously described in [MONB5GD61], due to containerization and standardised APIs, the same AD component can be easily configured to operate in other domains e.g., in Cloud to analyse server-side metrics. In this test, it is assumed, that there is a typical time-of-day dependent traffic pattern, which represents the variable user activity (cf. Figure 4-36). In next generation networks, it is not unlikely to assume, that based on this time-of-day traffic pattern, adequate number of resources will be allocated to the slice for the purpose of efficient resource management. In an event of sudden traffic increase, it is possible that the allocated number of resources will not be enough, and relevant scaling operations will be needed to provide the access to the service for the increased number of users. Analysis of traffic can also help to identify other issues, such as incorrect load balancing, or a complete service failure. Detected anomalous information can be further sent to relevant MonB5G layer components (DEs, AEs) for further processing. One of the possible actions to tackle sudden, unexpected increase of user traffic (i.e., caused by an online event) would be server upscaling, to temporarily increase the service capacity.

4.3.4.2 EVALUATION OF KPIS OVER EXPERIMENTAL PLATFORM

Figure 4-36 depicts emulated time-of-day traffic pattern, by varying the amount of UEs between 1 and 8 (first chart), as well as user traffic (second chart), total user traffic (third chart) and CPU usage (fourth chart).

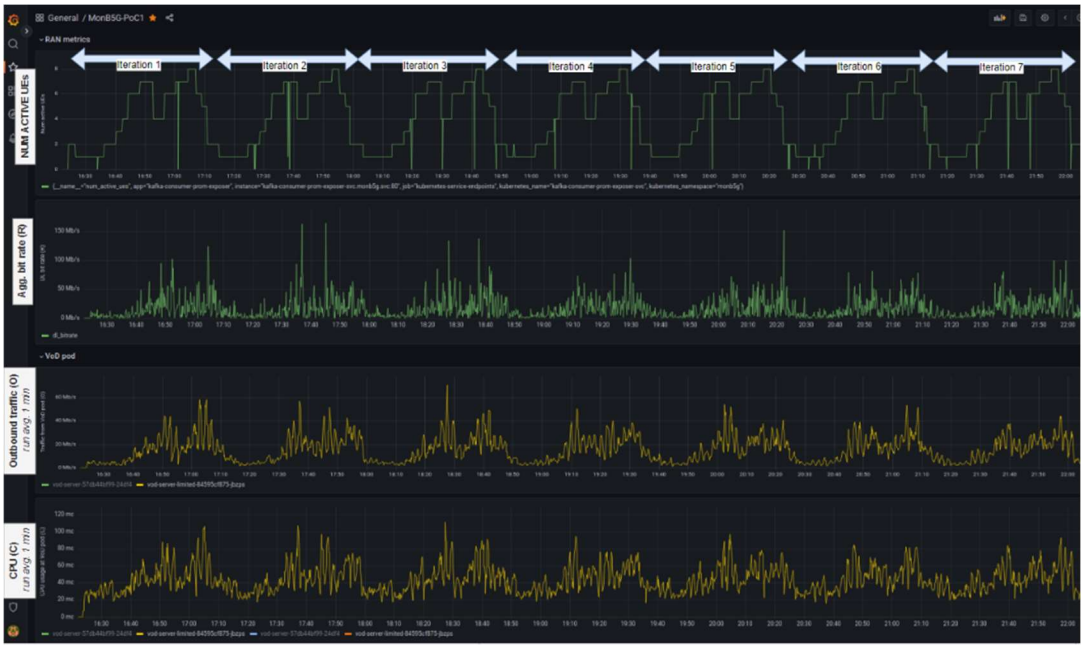


Figure 4-36: The metrics collected during non-anomalous network operation.

The data was further aggregated and used to train the AD's in-built LSTM model in the offline manner. Data from RAN is collected by the MS via SF and transferred to AD via message bus. The output from AD working on non-anomalous signal is depicted in Figure 4-37 Orange and yellow lines show real and predicted value respectively. Blue line depicts the error between prediction and actual value. Anomalies are determined based on the error value and a threshold algorithm. In this scenario, a static threshold (red line) has been used. Sign of the error (positive or negative) can also signal what kind of anomaly (traffic increase/traffic decrease) is detected.

Figure 4-38 depicts data with injected anomalous traffic changes:

- Sudden increase in traffic during night-time (1st, 3rd and 5th green peaks), matched with large, negative error,
- Earlier than usual decrease in traffic (2nd green peak), matched with large positive error,
- Sudden traffic drop (4th green peak), matched with large, positive error.

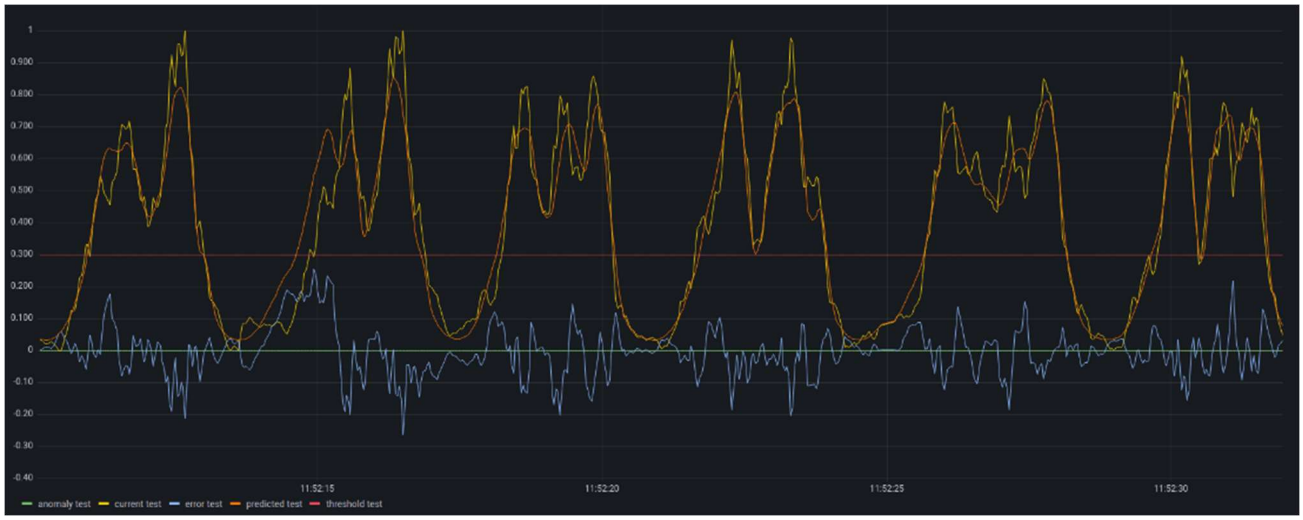


Figure 4-37: Non-anomalous signal traffic

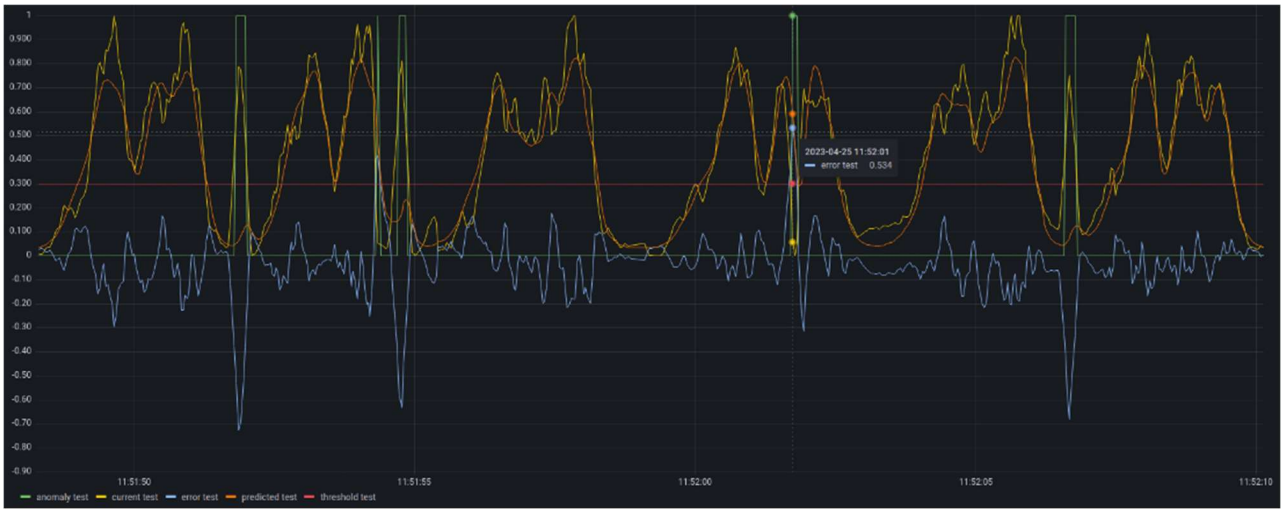


Figure 4-38: Signal with anomalies

It can be observed that the proposed AD trained on a typical traffic pattern (Figure 4-37), enables quick detection of traffic deviations (cf. Figure 4-38). As depicted in Figure 4-38, in all cases, the anomaly was detected at the beginning of an abnormal traffic change and lasted until after traffic returned to normal. Considering the local deployment, the detection delay is solely dependent on the traffic volume sampling rate and reporting period. This gives the system an early warning and time to take a proper action (i.e. run diagnostics, make a reconfiguration, etc.).

This enabler addresses the "Reduce the reaction time (time from identification to resolution via appropriate reconfigurations) to an NS malfunction" KPI. LSTM-based Anomaly Detection detects anomalies at the beginning of abnormal traffic change, giving management system extra time to react to change (as opposed to detecting the change after it's at maximum). MonB5G architecture being decentralized, allows for all necessary reconfigurations to be taken without informing higher management instances, which additionally

reduces reaction time to a malfunction, since reporting period of higher architectural layers can be considerably less frequent than local deployment.

4.4 Evaluating KPIs of MonB5G Solutions at the PoC-1 Scenario 2 Testbed

4.4.1 Slice Admission control (DE) Based on traffic Prediction

In this section, the description of Time-of-Day-Aware Slice Admission Control (TASAC) DE, which allows taking Slice Admission Control (SAC) decisions leveraging the information on the predicted future network traffic intensity to select the most profitable slices for the operator. First, the short description of TASAC principles is described together with the experimental use case aiming to prove the TASAC benefits over conventional SAC approaches (Section 4.3.1.1). The second part of this section is devoted to the evaluation of TASAC contribution to the key MonB5G targets and relevant KPIs (Section 4.3.1.2).

4.4.1.1 USE-CASE DESCRIPTION AND BASELINE

One of the key targets for the MNOs is to efficiently use the available infrastructural resources in time in order to maximize the profit. One of the key optimization mechanisms is the adoption of relevant SAC mechanisms which enable the selection of the most profitable slices and maximizing the slice admission rate under specific constraints. Conventional SAC algorithms treat the resources allocated to the slices as fixed, which can lead to significant resource underutilization since the actual resource consumption is dependent not only on the slice footprint but also the activity of slice users. To this end, the TASAC algorithm combines two approaches to derive the slice admission policy that enables maximization of slice admission rate and maximization of resources utilization. First, two slice types are distinguished according to the user traffic specifics: dependent on the Time of Day (ToD), referred to eMBB (traffic generated by human-operated terminal) and ToD-independent, referred to mMTC (traffic generated by machine-like terminals). For the ToD-dependent slices, the resource consumption scales with the progressing ToD (behaviour well-known and exploited by the MNOs during network design), while for the ToD-independent the consumption is relatively constant in the deployment period. TASAC leverages this information to increase admission rate in the less busy periods. Second, the TASAC DE implements the DRL model-free off-policy algorithm called Double Deep Q-Network (DDQN) [DDQN], which enables deriving the policy in complex environments in long-term perspective. The use case implemented to evaluate TASAC benefits is presented in Figure 4-39.

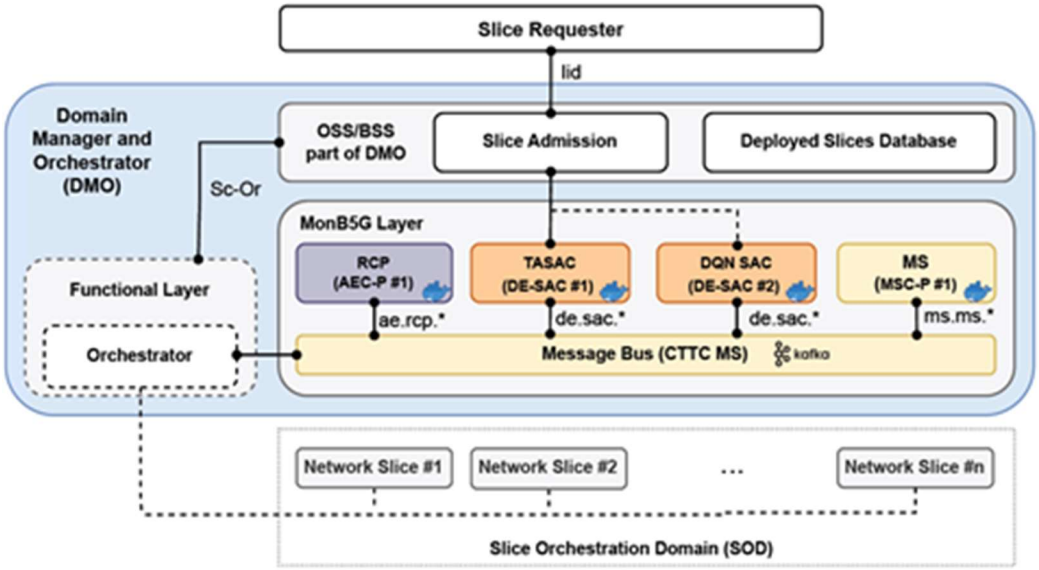


Figure 4-39: The evaluation approach of TASAC enabler

In the analysed UC, slice tenants (behaviour emulated by the Slice Requester block) issue the Slice Admission Requests (SAR) over the web-based I_{id} interface (REST-based queries) to deploy network slices. Each SAR contains the following information: slice type (eMBB/mMTC), slice deployment and termination time and maximum requested resources by the slice. The Slice Admission block sends a query to the active SAC DE to obtain the admission decision. The SAC DE interacts with the relevant deployed components (to obtain the network state (as described in [MonB5GD42] and [MonB5GD43]), make decision on its basis and calculate the reward for the action (slice acceptance or rejection). In the case of slice acceptance, the role of OSS/BSS part of DMO is to trigger relevant orchestration functions residing in the Functional Layer to orchestrate the slice instances specified in SAR. As described in [MonB5GD43], the large-scale testing of SAC is problematic due to the general limitations of the available resources, i.e., hundreds of slices require tremendous amounts of resources for their deployment, which is problematic from the lab-scale tests perspective. Hence, to collect the KPIs over the experimental platform, a hybrid approach has been taken which included using the fully functional SAC components integrated with the CTTC MS and a simulation part that handles the infrastructural limitations and simulates resource consumption of the individual slices. Therefore, when the SAR is accepted, the simulation environment is updated to reflect the resource consumption originating from the newly deployed slice.
As the baseline for the comparison, the same scenario is conducted for the DQN-based approach commonly used by the academia to optimize SAC process [SAC-DQN1] [SAC-DQN2].

4.4.1.2 EVALUATION OF KPIS OVER EXPERIMENTAL PLATFORM

To emulate the realistic conditions, first, the duration of a day is set to 1440 *tu* (time units) which correspond to the acquisition of monitoring data every 1 min. The influx of SARs is modelled as Poisson process with the rate *tu*. The requested slice types follow a discrete uniform distribution, the maximum resources requested by tenants equal to 10 *u* (generic resource units) for mMTC and 100 *u* for the eMBB slices, and their duration follow the discrete uniform distribution (equivalent to the 30 min to 24 hours deployment). The resources available for the

operator to orchestrate slices are *u*. The agent is trained over consecutive episodes which last 1440 *tu* each. During the experiments, the decisions are made based on available bandwidth and ToD activity information derived from the carrier-grade transport network. The comparison results of TASAC and the baseline DQN-based SAC in terms of: (a) obtained reward and (b) resources, requested (R) or consumed (C) by the deployed slices are presented in Figure 4-40 and in Table 14.

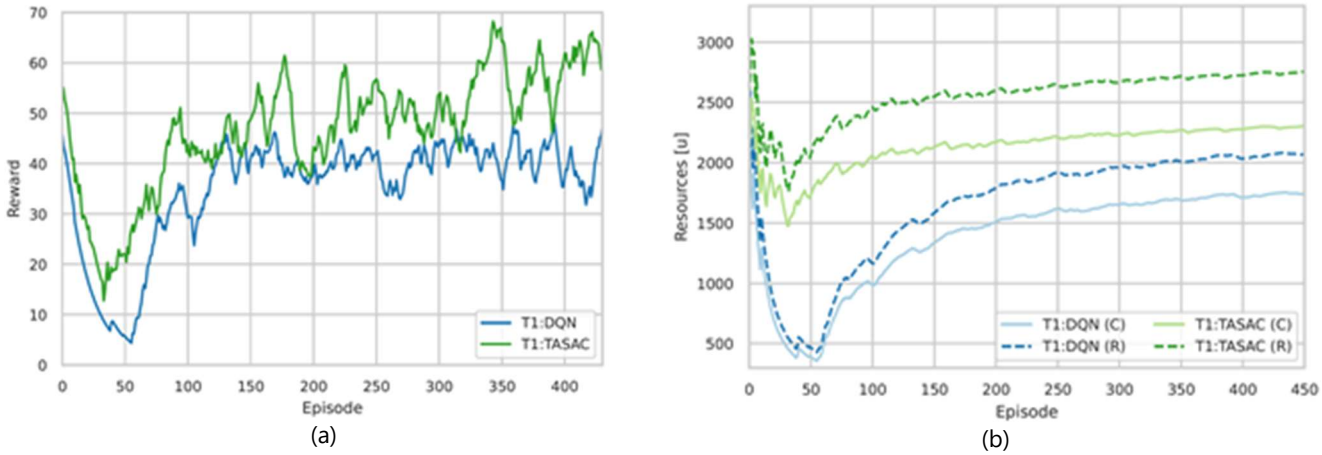


Figure 4-40: Comparison in terms of: (a) obtained reward (b) resources.

It can be observed that both the agent's accumulated reward as well as the resource utilization are much better for the TASAC DE than conventional DQN-based DE. In the late stages of operation i.e., when the near-optimal stable policies are reached by the agents the reward gain exceeds over 30%, while the resource utilization gain is around 60%.

Table 13: Comparison of cumulative gains of TASAC and DQN-based DE

Episode	Reward			Requested resources [u]			Consumed Resources [u]		
	DQN	TASAC	Gain [%]	DQN	TASAC	Gain [%]	DQN	TASAC	Gain [%]
50	148.6	755.9	**408.7**	47662.3	107797.1	**126.2**	39718.2	90213.9	**127.1**
150	3793.5	5510.6	**45.3**	167948.6	350174.5	**108.5**	141231.4	292807.7	**107.3**
250	7642.3	10296.2	**34.7**	348340.5	609580.5	**75.0**	293353.0	509734.0	**73.8**
350	11927.0	15664.1	**31.3**	545034.3	877934.7	**61.1**	458988.8	734095.2	**59.9**

The obtained utilization gain can be traded for a reduced number of SLA violations. Hence the evaluation of TASAC under more strict resource limits have been conducted to match the final resource utilization for both approaches and enable the comparison of SLA violations ratio. Figure 4-41 shows comparison of TASAC and the baseline DQN-based SAC in terms of: (a) resources, requested (R) or consumed (C) by the deployed slices, and (b) violations ratio.

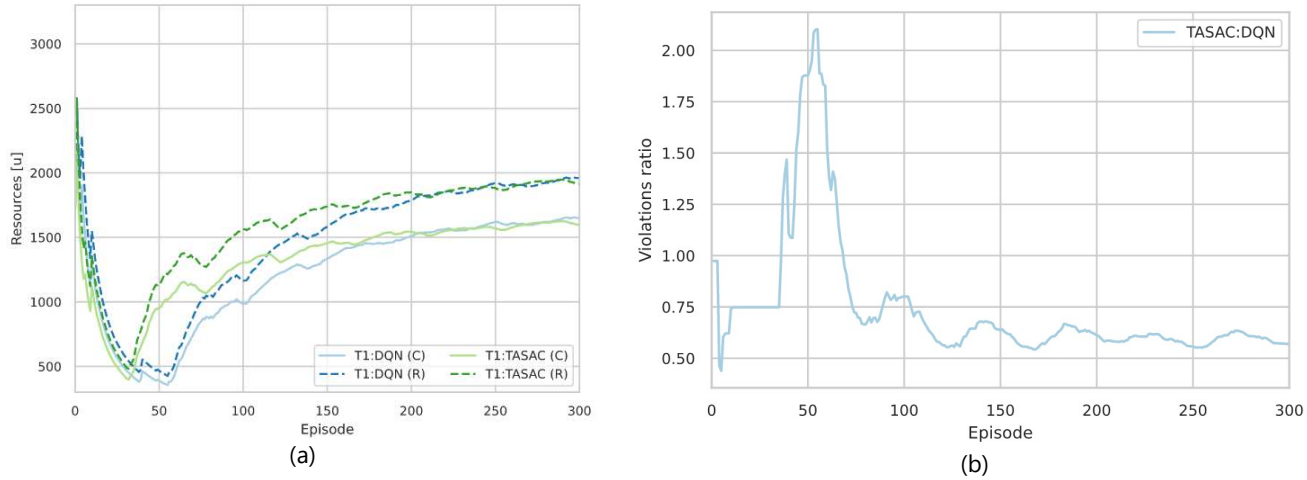


Figure 4-41: Comparisons in terms of: (a) resources (b) violations ratio

It can be observed that once the policy is satisfactory (around episode 200), TASAC approach enables over 30% reduction of number of SLA violations in comparison to conventional DQN-based approach.

Overall, the TASAC DE contributes to the MonB5G target KPIs as listed in Table 5 in the beginning of the deliverable.

4.4.2 A multi-agent learning for distribution resource allocation in the RAN domain

4.4.2.1 USE-CASE DESCRIPTION AND BASELINE

We cast the RAN resource allocation problem as an optimization problem in a network slicing setup, focusing on minimizing the traffic exceeding the service level agreement (SLA). The current approaches suffer from scalability issues in real deployments, where the amount of monitoring information to be exchanged, together with the large number of base stations (BSs), make it practically impossible to devise optimal resource allocation schemes in a timely and resource-efficient manner. Due to the distributed nature of the RANs domain, centralized approaches are doomed to provide sub-optimal performance and introduce significant communication overhead towards holistic resource controllers. The benefit coming from our approach are several: i) it enables resource allocation at the edge of the network, thus accounting for more timely and accurate information, ii) the amount of control information that needs to cross the network to reach the central controller dramatically decreases, thus reducing overhead towards the core network, iii) by allowing information exchange among local decision agents (DAs), we enable the provisioning of federated learning schemes to further enrich the capabilities of the DAs. In fact, DAs will not only learn from a local observation space, but also leverage information coming from other (statistically different) RAN nodes, thus improving the generalization of the learning procedure.

4.4.2.2 EVALUATION OF KPIS OVER EXPERIMENTAL PLATFORM

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. We design and implement the overall framework as shown in Figure 4-42 and Figure 4-43.

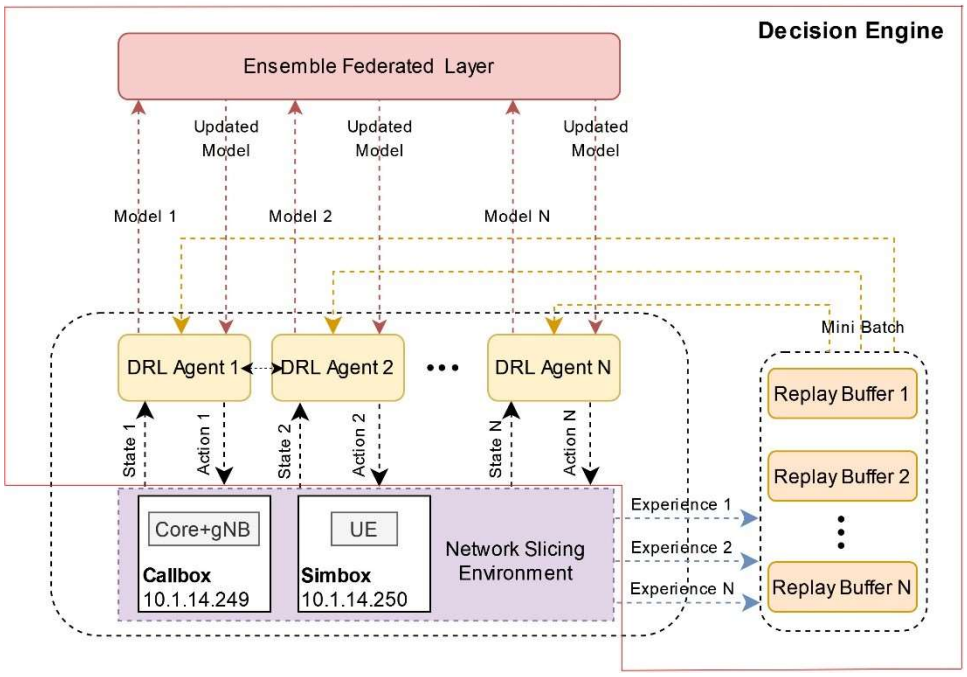


Figure 4-42: Federated RAN slicing architecture

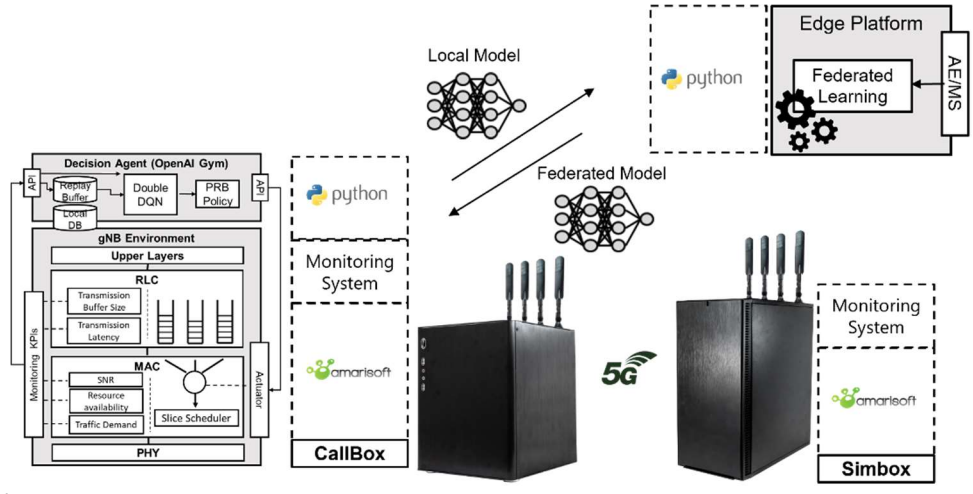


Figure 4-43: Testbed architecture

The testbed includes the following components:

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. We generate synthetic traffic traces with dynamic patterns by means of mgen traffic generation tool (https://github.com/USNavalResearchLaboratory/mgen). 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 is used to generate downlink traffic towards the UE with IP 192.168.2.2, over port 5000. More in details, a traffic burst of 5 seconds duration is generated regularly every 10 seconds. During the traffic burst, packets of 1500 bytes size are generated with a rate of 1000 packets/s, and directed to the Simbox.

AMARI Callbox: 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 [Amarisoft].

```
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
```

Monitoring System: Exploiting a dedicated API hosted by the gNB, it enables real-time monitoring of multiple KPIs, including the number of connected devices, bandwidth utilization, channel quality, etc., which are stored in a local database as shown in Figure 4-44.

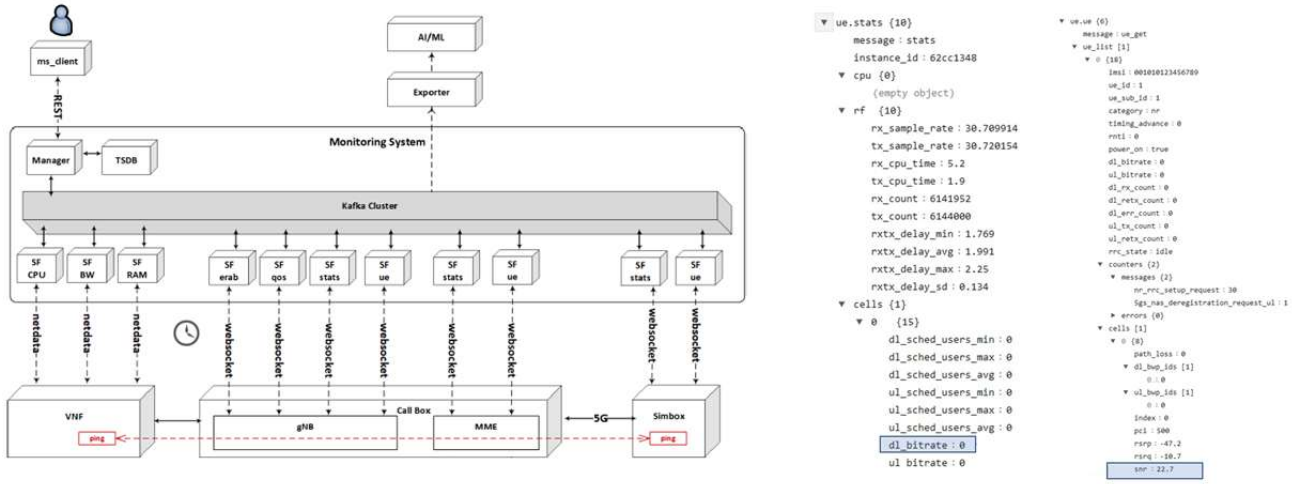


Figure 4-44: MonB5G MS and monitored dataset.

Local Decision Agents: they collect and consume local monitoring information from the monitoring system to adjust radio resource allocation policies according to real-time traffic variations as shown in Figure 4-45. Decision-making is supported by AI algorithms and DRL approaches.

```
[0.004] ### Ready: name=ENB, type=ENB, version=2022-06-18
[0.004] <== Send message config_set id#1
[0.016] ==> Message received
{
    "message": "config_set",
    "message_id": "id#1",
    "time": 401565.243
}
--->Connecting to Kafka bus via: 10.64.116.10:31308
------------------------------moniroring_dl_bitrate (Mbps) - kafka: --------- 999.713585
------------------------------moniroring_snr: ---------------------- 33.2
------------------------------PRB Allocation to the CELL: ------------------- 268
WebSocket remote API tool version 2021-02-23, Copyright (C) 2012-2021 Amarisoft
[0.003] ### Connected to 10.1.14.249:9001
[0.004] ### Ready: name=ENB, type=ENB, version=2022-06-18
[0.004] <== Send message config_set id#1
[0.016] ==> Message received
{
    "message": "config_set",
    "message_id": "id#1",
    "time": 401615.384
}
--->Connecting to Kafka bus via: 10.64.116.10:31308
------------------------------moniroring_dl_bitrate (Mbps) - kafka: --------- 998.138341
------------------------------moniroring_snr: ---------------------- 33.5
------------------------------PRB Allocation to the CELL: ------------------- 268
WebSocket remote API tool version 2021-02-23, Copyright (C) 2012-2021 Amarisoft
[0.005] ### Connected to 10.1.14.249:9001
[0.005] ### Ready: name=ENB, type=ENB, version=2022-06-18
[0.006] <== Send message config_set id#1
[0.017] ==> Message received
{
    "message": "config_set",
    "message_id": "id#1",
    "time": 401664.955
}
--->Connecting to Kafka bus via: 10.64.116.10:31308
```

Figure 4-45: Decision Agents

Figure 4-46 depicts a screenshot taken from the running multi-agent setup.

```
MultiAgent.INFO -  ---------------------------------- Internal Episode = 2/15
MultiAgent.INFO - Priority order: [0, 1, 2] - Initial actions: [5, 4, 3] -  Updated Actions [5, 0, 0] -  PRB Allocation [50, 1, 1]
{'message': 'config_set', 'time': 266181.372, 'utc': 1674124111.946}
{'message': 'config_set', 'time': 266182.385, 'utc': 1674124112.959}
{'message': 'config_set', 'time': 266183.398, 'utc': 1674124113.972}
Slice:  0 --- DL bitrate  12.87752 --- DL capacity:  0.0 --- difference:  -12.87752
GYM EnvBS1.INFO - Reward 0:  -12.87752
Slice:  1 --- DL bitrate  0.00472 --- DL capacity:  0.0 --- difference:  -0.00472
GYM EnvBS1.INFO - Reward 1:  -0.00472
Slice:  2 --- DL bitrate  6.238712 --- DL capacity:  26.195844 --- difference:  19.957132
GYM EnvBS1.INFO - Reward 2:  -19.957132
```

Figure 4-46: Multi-agent setup

The radio resource allocation decisions performed by the agents are translated into API calls like the one in 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 API call includes the IP address and port number as the target for the API call (corresponding to the gNB in this case), the target cell id, an allocation flag defining if the resource block allocation is fixed or not, and the boundaries of the desired radio resource block allocation. The latter is 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.

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.

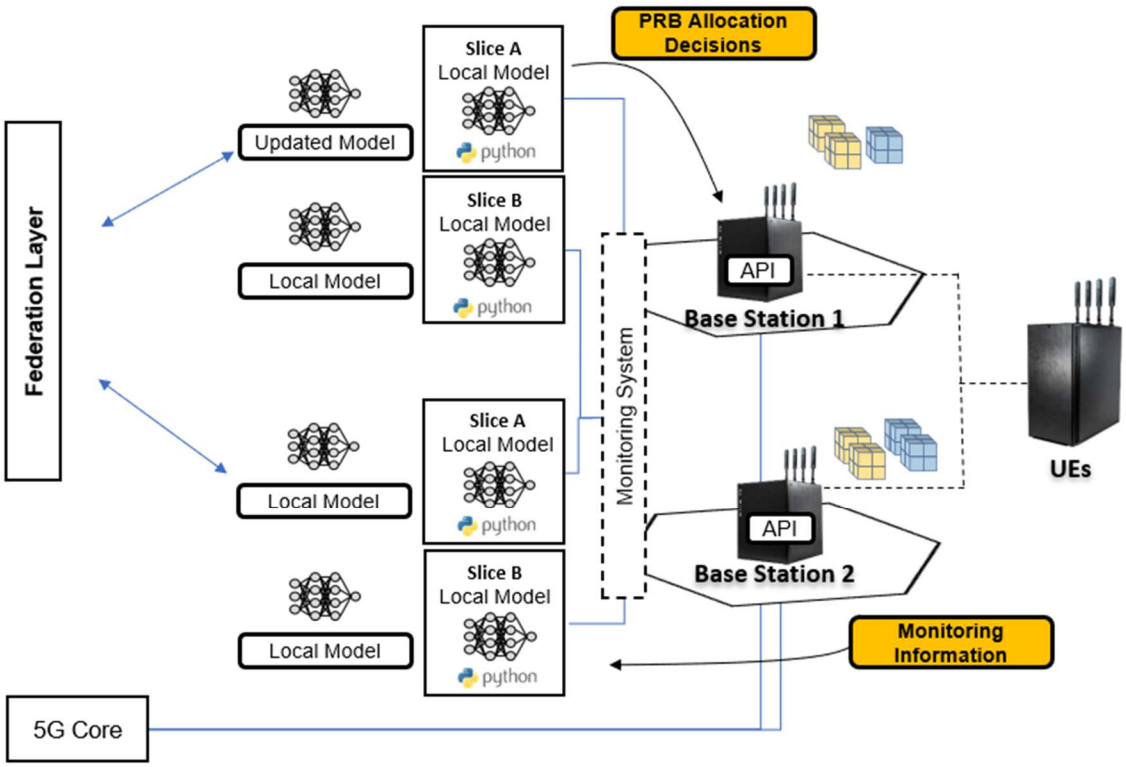


Figure 4-47: Evaluation Scenario

The preliminary integration of the software components developed started in the context of WP4 and has been finalized in WP6. Figure 4-47 depicts the evaluation scenario considered in our PoC. We leverage a distributed learning mechanism and multiple decision agents that collaboratively specialize their decision policies onto real-time traffic demands, aided by a coordinated exchange of information to avoid the occurrence of conflicting situations. In particular, we instantiate two gNBs and connect them through a local 5G core. Then, we deploy two eMBB slices, namely Slice A and Slice B, over each gNB, and spawn the setup of a local decision agent for each slice instance. As already detailed in D4.3 and D6.1, exploiting the monitoring system API, the RAN agents can retrieve real-time monitoring information from the Amarisoft RAN platform, which in turn are used to perform training activities pursuing RAN resource allocation. We generate different traffic for each running slice, considering it as the aggregated volume for each UE connected to the specific slice. Moreover, we allow only the agents belonging to Slice A to exchange their local models and perform federation.

Figure 4-48: GUI: Graphical User Interface

For system monitoring and performance analysis reasons, we also design an online dashboard that provides a real-time overview of the solution. Figure 4-48 illustrates the Grafana-based graphical user interface developed for the testbed. It provides an overview of the key performance metrics of test scenario, as well as the agents-specific metrics such as instantaneous reward, allocation gap, and allocated radio resources. After an initial exploration and training phase, achieves the right trade-off between optimal allocated radio resources and traffic demand accommodation, i.e., exploitation. To this aim, the agents receive partial observations from the running services or slices (e.g., channel quality, consumed resources, etc.) by interacting with each other and with the underlying physical environment. A reward is calculated as an incentive mechanism based on the actions of all agents, indicating how the agents ought to behave. The designed reward function guarantees adequate performances, impeding over-provisioning and leading to fair resource allocation among the running slices deployed within the same cell.

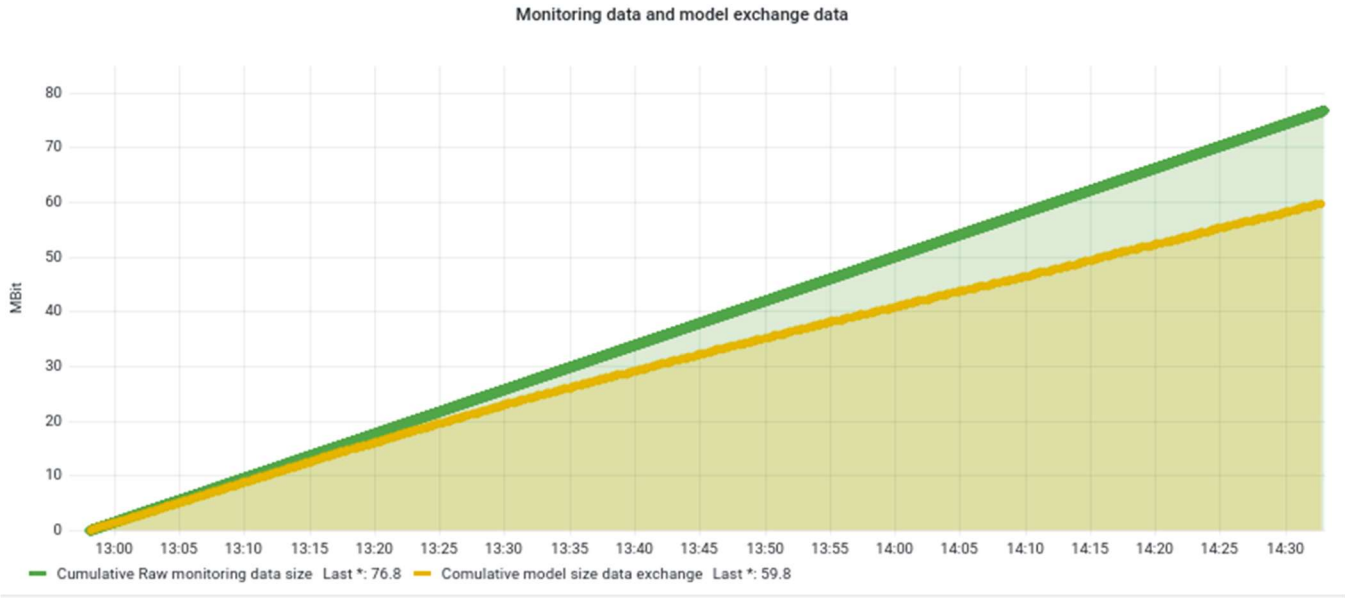


Figure 4-49: Communication Overhead

The communication overhead of a classical centralized radio resource management solution involves collecting monitoring data from underlying cells to gather an overall view of the underlying system and optimize resource allocation. In contrast, the proposed distributed approach consumes monitoring data locally and involves overhead data exchange with a centralized entity only for model weight exchange for federation purposes. Figure 4-49 compares the overhead of a centralized approach and our distributed one. Results demonstrate that the distributed approach significantly reduces communication overhead, with a performance gap that increases over time. At the end of our experiment, we saved ~20MB of data, which represents 25% of the overall monitoring data. It is worth mentioning that our experimental setup only considers a limited number of base stations, whereas in large RAN deployments, the monitoring and communication overhead reduction would be even more remarkable.

In Figure 4-50 and Figure 4-51, we present the performance comparison of two network slices based on reward. To conduct the comparison, both network slices in each gNB were trained using the same traffic and hyperparameters in the exploration phase. In the exploitation phase, traffic slices A and B were exchanged between the gNBs, and their performance was evaluated based on the number of iterations required for convergence. We exchanged traffic patterns of slice slice A and Slice B in both gNB at the end of the *exploration* phase. In this way, agents performing federation may benefit from the experience and knowledge of the others, and better adapt their decisions during the *exploitation* phase.

The study results indicate that network slices A outperformed network slices B in both gNBs. Specifically, the performance of slice A in gNB1 and slice A in gNB2, which were involved in the federation, was better than that of Slice B in gNB1 and Slice B in gNB2, which were not part of the federation. Based on these findings, network slices A engaged in the federation procedure are better suited for the given task and should be prioritized for further development and implementation. This is also supported by the performance comparison in Figure 4-52 and Figure 4-53 regarding the allocation gap.

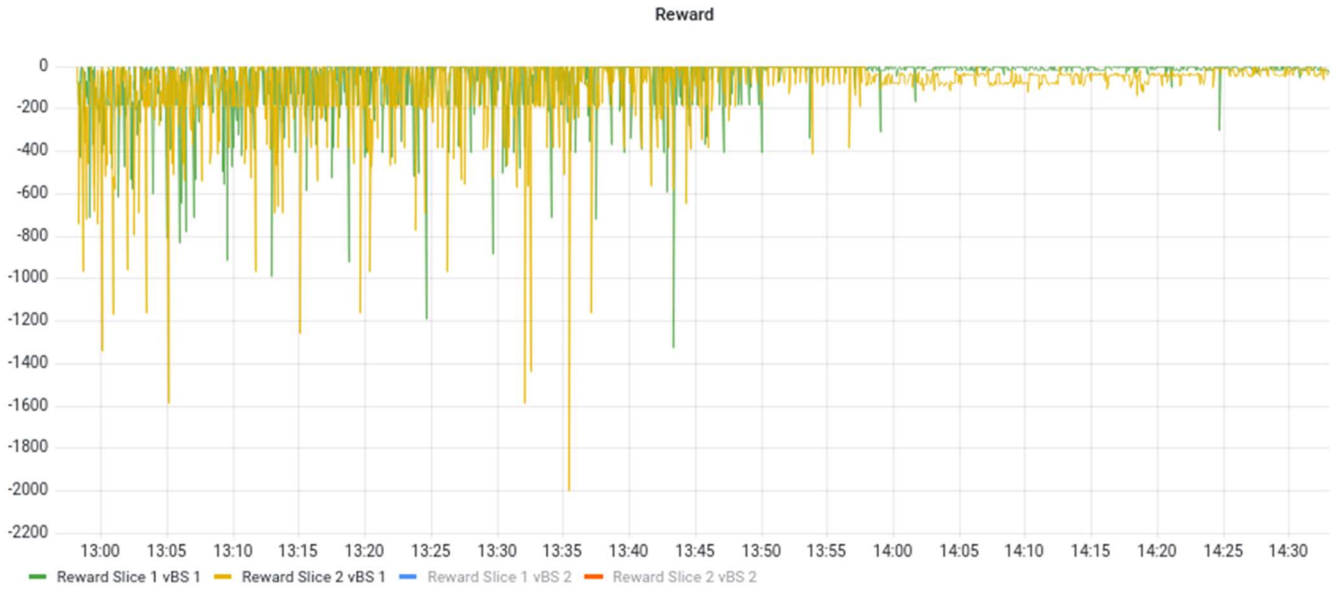


Figure 4-50: The convergence performance of agents A and agent B in first gNB

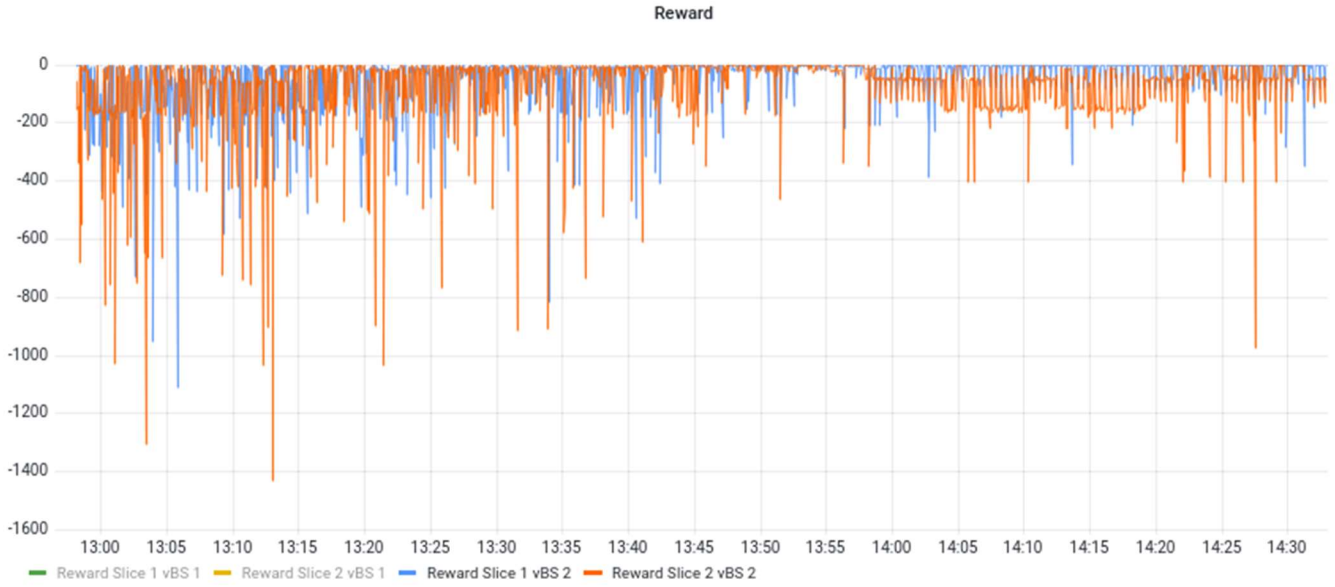


Figure 4-51: The convergence performance of agents A and agent B in the second gNB.

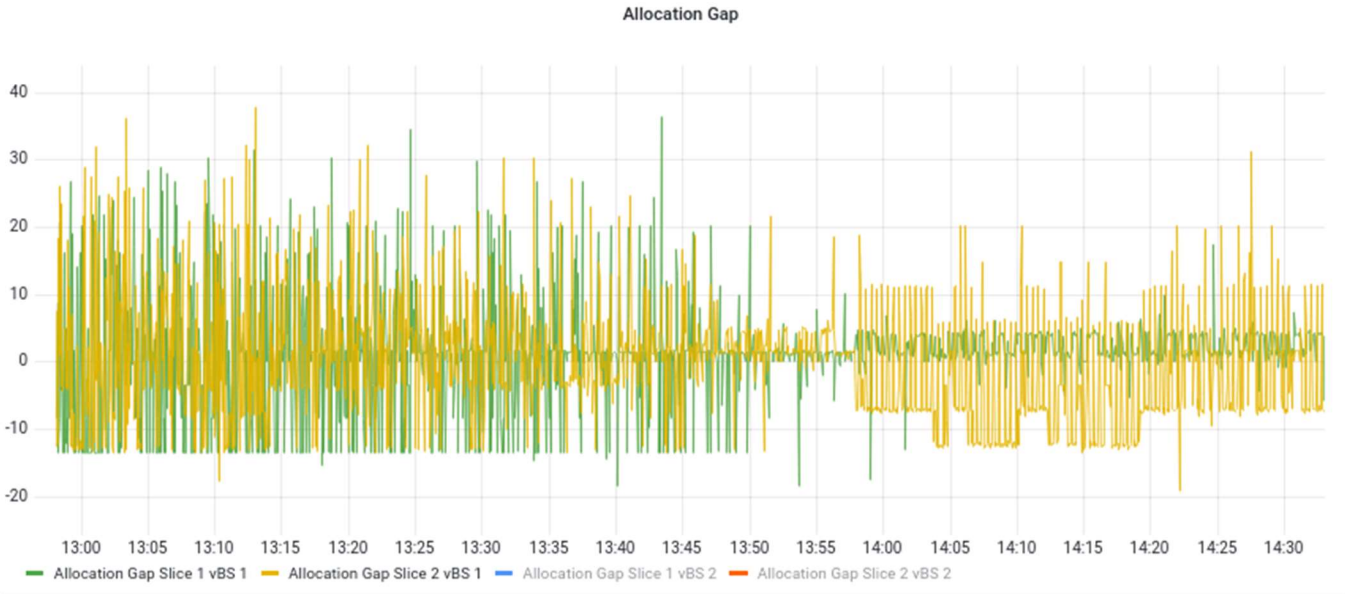


Figure 4-52: Allocation gap performance for agents A and agent B in gNB1.

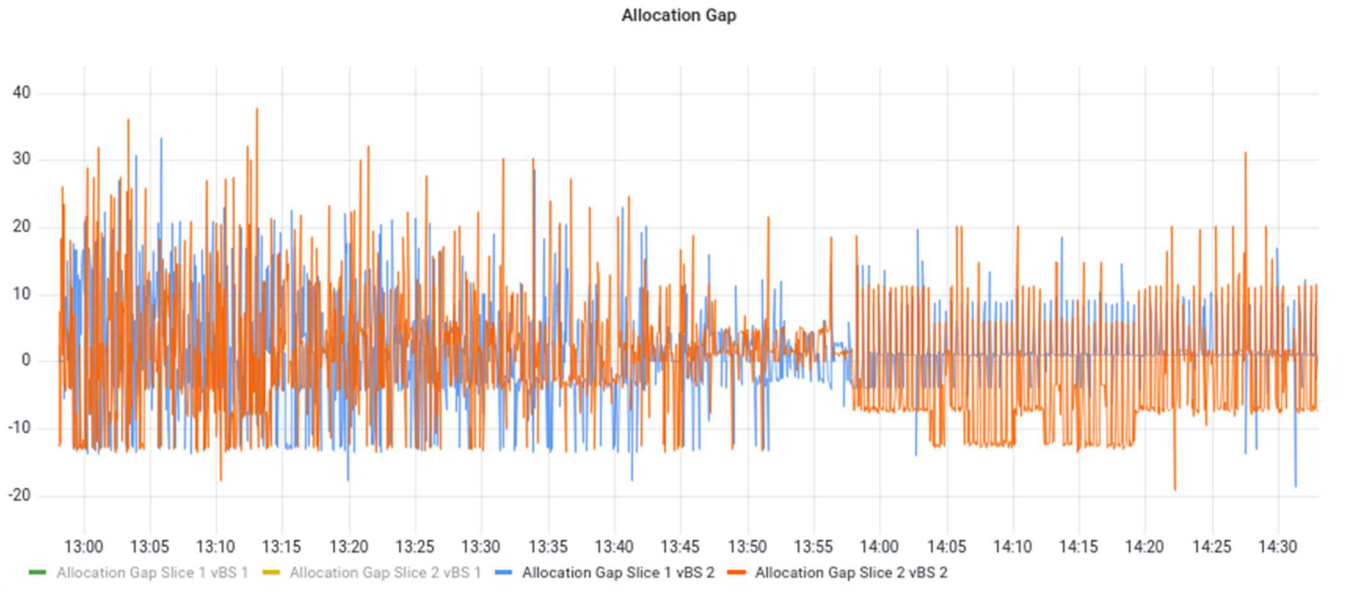


Figure 4-53: Allocation gap performance for agents A and agent B in gNB2.

Figure 4-54 illustrates the cumulative distribution function (CDF) of the allocation gap that slices experience within the network, resulting from the DRL agent's radio resource allocation policy. The term "allocation gap" commonly refers to the disparity between the requested or intended allocation and the actual allocation of a specific resource. In the realm of 5G network management, the allocation gap can signify the distinction between the desired allocation of PRB and the allocation that takes place. Indeed, the allocation gap can be

defined as the distinction between the requested allocation of PRBs and the achieved allocation of PRBs. PRBs are employed to allocate frequency and time resources in wireless networks, and the allocation gap in our scenario emerges due to the wrong decision of AI agents. In this regard, we pursue an experimental approach to deploy four eMBB slices with different traffic shapes, emulating the slice traffic generated by different gNB. The curves measure the probability of incurred allocation gap within slices during the agent training. The shape of the curves is based on the variability of allocation gap values, which is allocation gap distribution. The results indicate that federated slices, i.e., Slice 1 and Slice 3, have superiority compared to non-federated slices, i.e., Slice 1 and Slice 3, where 90% and 67% of the perceived allocation gap is between -2.5 and 2.5 Mbps, for Slice1 and Slice 3 respectively. In contrast, the values for Slice2 and Slice4 are 85% and 65%, respectively. It turns out that the proposed federated approach leads to optimal and more adequate PRBs allocation concerning dynamic traffic and resource contention among slices and then federated slices experience a lower allocation gap. On the other hand, the non-federated strategy incorrectly balances resource allocation for different network states (state-action pairs), leading to a higher allocation gap. In conclusion, 5% improvement in slice 1 and 3 percent improvement in slice 2 have been achieved in comparison to non-federated slices (namely slice 2 and slice 4). Therefore, on average 3.5% improvement in SLA violation in terms of allocation gap considering both under-provisioning and overprovisioning cases with respect to non-federated approach. Note that the difference is slight since we need more slices to generalize the model better and evaluate the federated approach and due to testbed limitations, the number of slices were limited and created up to 4 slices (2 federated and 2 non-federated).

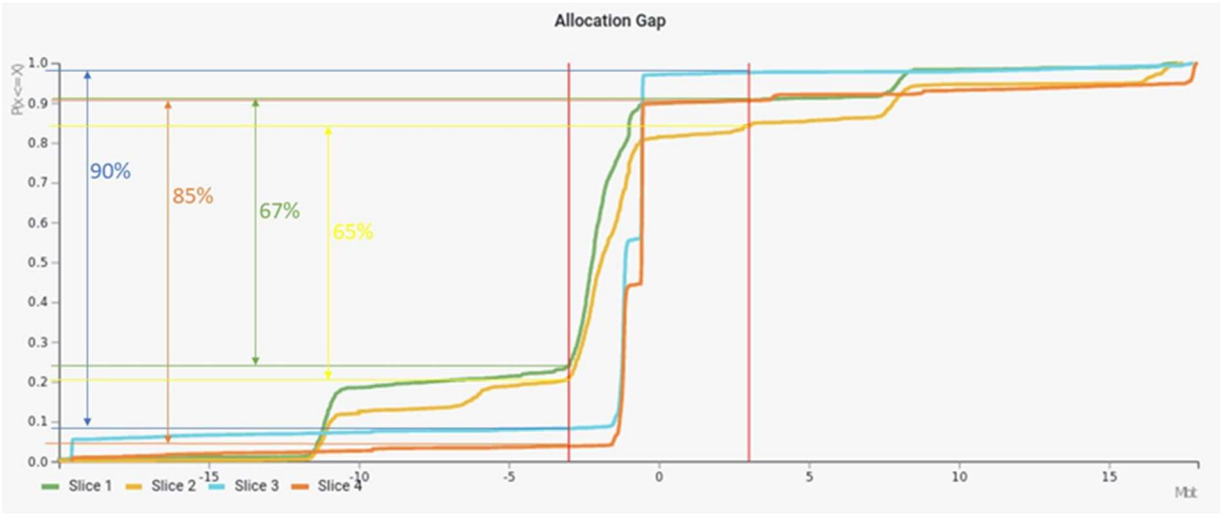


Figure 4-54: The CDF of experienced allocation gap (in Mbps) within network slices.

In Figure 4-55, we focus our analysis performance of 50 gNBs in simulation environment on the dropped traffic, i.e., the volume of traffic that did not meet the latency requirements due to wrong PRB allocation decisions (allocation gap due to under provisioning), measured in percentage of the offered traffic volume of each federation episode with an increasing number of end-users in federation strategy. From this figure, we can notice how during the initial exploration phase inexperienced PRB allocation decisions performed by the decision agents heavily affect the latency requirements of all network slices, with peaks of dropped traffic

that increase with the growing number of end-users. Nevertheless, this trend improves over time as the agents gain knowledge over the underlying scenario and get trained, finally converging after policy switch, i.e. after episode 400, towards values in the order of 2% for the eMBB slice, and 0,32% for the URLLC slice.

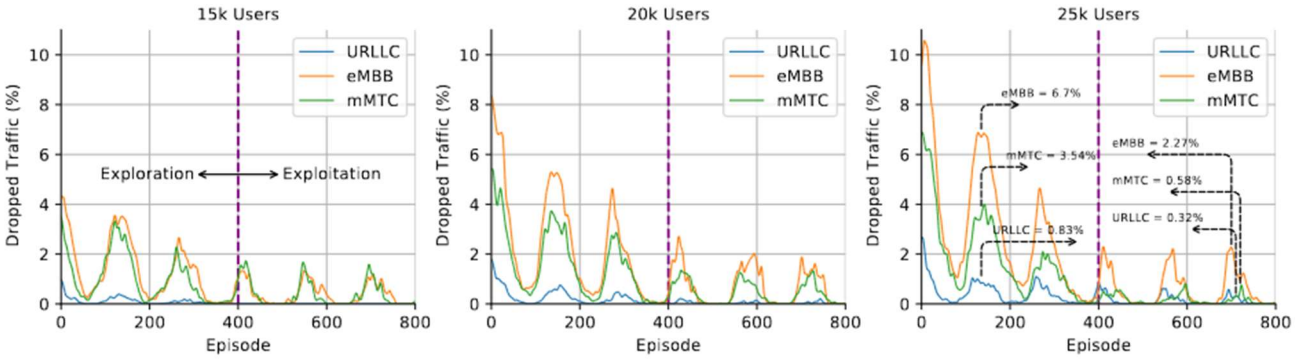


Figure 4-55: Performance evaluation for different network loads.

4.4.3 RL-based slice admission control

The mechanism proposed her for admission control is based on a Reinforcement Learning (RL) algorithm called Soft-Actor Critic, which was implemented in Python3.8 and Tensorflow 2.1.

4.4.3.1 USE-CASE DESCRIPTION AND BASELINE

The use-case consist of a series of UEs requesting service from end-to-end slices deployed in the 5G experimental testbed. These UEs request different type of services on top of a teleoperation use case developed in the platform. PreBAC leverages the information coming from CATP (Section 4.3.1) plus additional system state information coming from the User Service Requests (USRs) generated by the UEs in the process of the experiments. PreBAC is driven by a reward mechanism in which a human operator can establish the marginal utility associated to the admission of a USR of a specific service type, since PreBAC has the objective of maximizing the overall utility in an infinite horizon experiment while satisfying the following constraints:

1. The aggregated resource demand of admitted USRs cannot go beyond the BW capacity at the gNB
2. Minimize SLA violations for the deployed end-to-end network slices.
3. Establish a tolerance for excess resource overallocation with respect to the demand for each slices.

As the experiment progresses, and the UEs issue USRs for the slices, PreBAC will determine the admission of the USRs based on the bandwidth availability of the slice, which it can modify, and seek to increase the overall utility perceived by the operator. To assess the effectiveness of PreBAC, the same experiment is executed with a static USR admission mechanism, that seeks to admit USRs into each slice in an equal manner, without regards for minimization of SLA violations, or changes in the resource demand profiles.

4.4.3.2 EVALUATION OF KPIS OVER EXPERIMENTAL PLATFORM

The KPI evaluated in the case of RL-BAC consisted of “Reduction of SLA Violations”, and for the use-case described for RL-BAC, this KPI translates into a Reduction of Reduction Rate of USRs as given in Figure 4-56 and Figure 4-57.

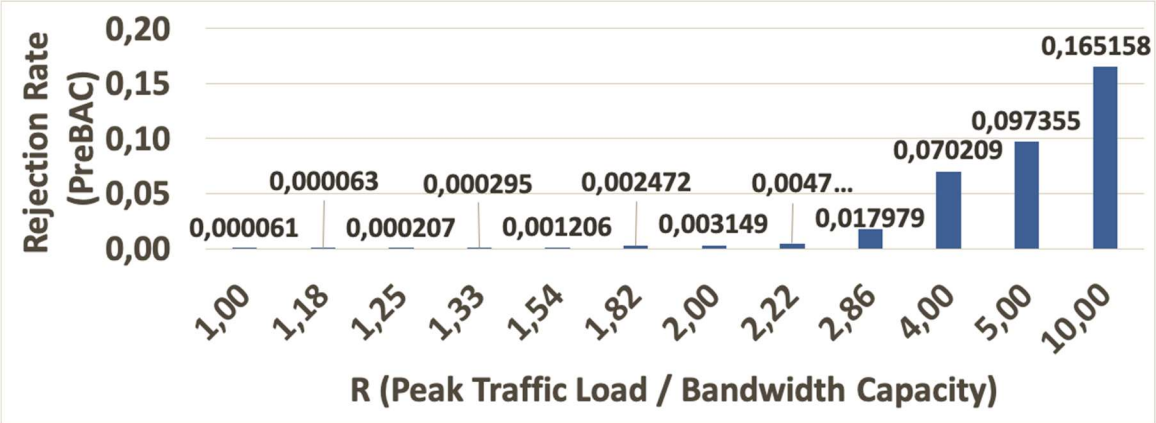


Figure 4-56: KPI evaluation of PreBAC for the current use case

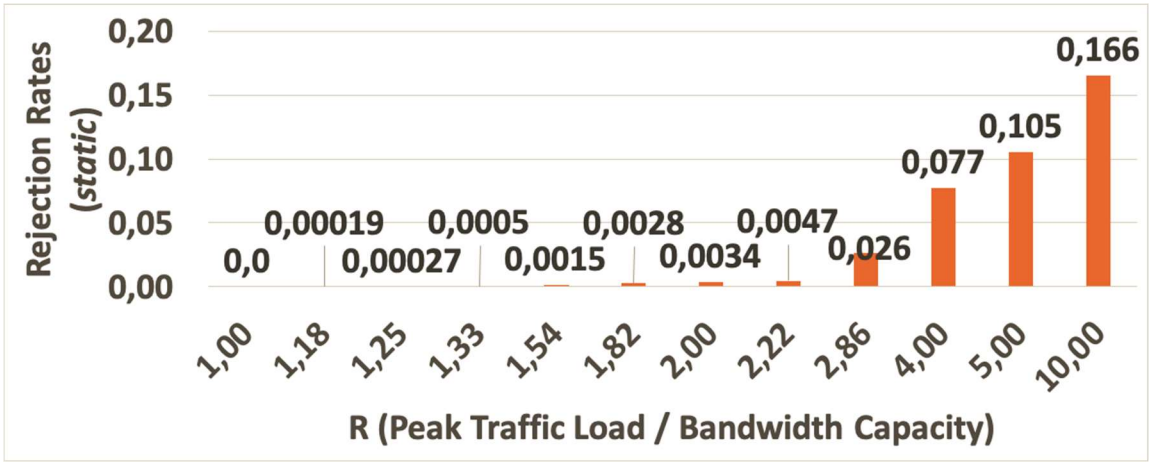


Figure 4-57: KPI evaluation of the baseline solution for the current use case

Notice that the USR Rejection Rate achieved by PreBAC is consistently smaller than that achieved by the static baseline. In some instances, PreBAC achieves a Rejection Rate that is 3 times smaller than the one of the static allocators for certain values of R.

5 Evaluating KPIs of the MonB5G Components over the Experimental Framework for PoC-2

5.1 Recap on the deployment scenarios in PoC-2

5.1.1 MonB5G PoC-2 Scenario 1: mMTC attack

For this scenario, the 5G testbed that is deployed at EURECOM premises was used. The testbed has been developed and used in many 5G European projects such as 5GEVE[3] and 5G!Drones[4]. 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.

MS collects data from AMF on every Attach Request received from the MTC devices. 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.

5.1.2 MonB5G PoC-2 Scenario 2: Federated Learning attacks

As depicted in Figure 5-1, 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 Inter-Domain Slice Manager (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 Domain Slice Manager (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 Deep Reinforcement Learning (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

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

[4] Online: https://5gdrones.eu/, Available: May-2023.

trusted participant applies a dimensionality reduction scheme and unsupervised machine/deep learning to detect the malicious participant (s) (see steps 4, 5, 6,7).

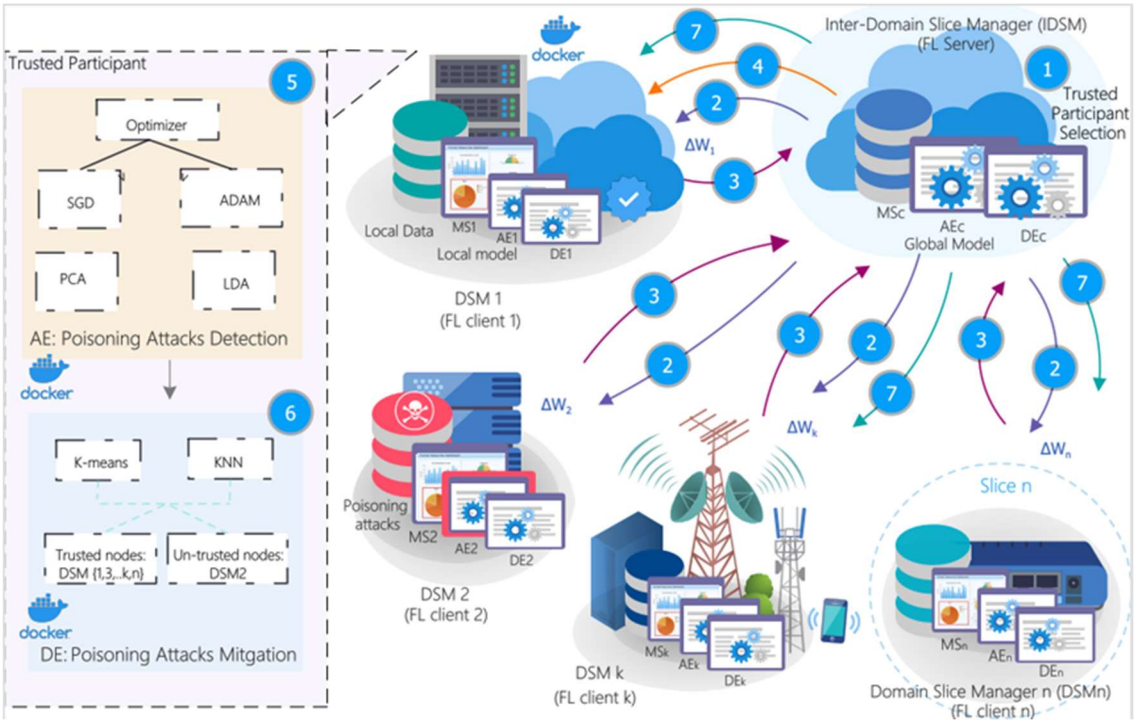


Figure 5-1: Overview of TQFL Framework.

5.1.3 MonB5G PoC-2 Scenario 3: aLTEr attack

As part of the PoC-2 "AI-assisted policy driven security monitoring & enforcement", we built a platform integrating MonB5G components and security tools to show the performance of such a system to detect and contain cyber security incidents. The support attack scenario is the aLTEr attack, and the asset targeted by the attacker is the web site being visited by the user on his equipment via the 5G network. The implementation of this scenario requires the integration of multiple components, starting with the context:

- The 5G core network in Stand-Alone mode with the control plane and user plane functions which provide data communication between the web client (Google Chrome) and the web server (monb5g.eu)
- The UE and the base station gNb devices are emulated by the "UERANSIM" tool.
- The manipulation of the DNS message by the attacker in the "UERANSIM" tool
- The malicious DNS server and the malicious HTTP server are implemented by CoreDNS and Apache HTTP Server respectively.

Reactive defence of 5G communications is ensured by the realisation of the MonN5G security orchestrator and security enforcement functions. As there is a single scenario and there is no need of coordination of

security activities between the local autonomic security orchestrator (L-ASO) and domain autonomic security orchestrator (D-ASO), the architecture defined in the [MONB5GD24] is simplified to ease the implementation. The implemented security orchestrator is an instance of L-ASO inheriting the interface Sc-Or of the D-ASO to manage the lifecycle of VNF/CNF. The DMO NFV-MANO is instantiated by the Kubernetes orchestrator and the Sc-Or interface is the Master API. Figure 5-2 below depicts the matching between the implementation and the architecture design of MonB5G Security Orchestrator.

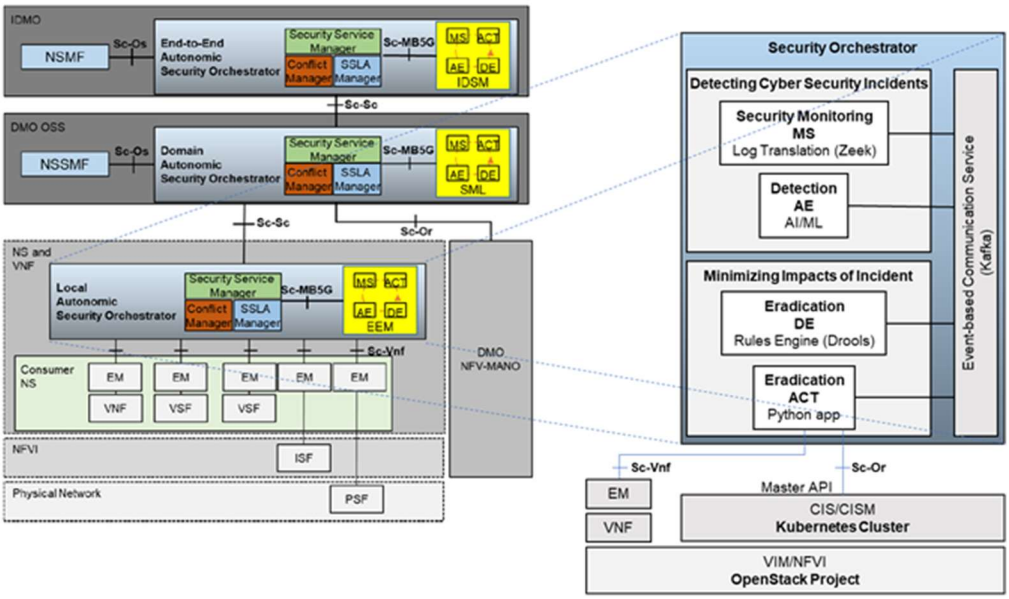


Figure 5-2: Mapping between the implementation and the architecture.

The security orchestrator automates the processing of security incidents by integrating the MonB5G components such as:

- The MonB5G MS consists of two components, the first one which is the port mirror of N6 interface, it extracts the selected raw data packets in the UPF at the N6 interface then sends via a VXLAN tunnel to the second component. The latter transforms the received raw data to meaningful logs and loads them onto the communication bus for analysis. These two MS components are implemented using the virtual switch VPP and the network security monitoring tool Zeek, and the communication bus uses Kafka topic.
- Threat detection leverage MonB5G AE components which implements several ML/AI algorithms as described in [MONB5GD5.4] to detect anomalies in meaningful logs prepared by the MS. This function raises an incident on detection of an anomaly.
- The security expert prepares the action plans and describes the inference rules in an expert system to decide on the conditions to execute them. The expert system is the Monb5G DE and it is implemented by the Business Rules Management System Drools. An action plan is inferred from the incident information and sent on the communication bus to be enforced.

- The ACT triggers the python script to configure of a security function or to trigger the network function orchestrator to manage the lifecycle of a function. For the scenario aLTEr, the action plan contains:
    - The deployment of the enforcement security function Coredns as a DNS over TLS (DoT) service on the Kubernetes cluster
    - The chaining of the DoT server with Kubernetes default DNS,
    - The deployment of the security function DoT client Stubby on the UE (UERANSIM) and the setting of DNS parameters

Figure 54 shows the mapping of the network functions involved the attack scenario (UE, commercial network, malicious relay, DNS servers and HTTP servers) on the actual components:

- 5G core network control plane Containerized Network Function (CNF) are deployed in the K8s cluster
- The user plane function (UPF) is running on a separate Virtual Machine (VM)
- The K8s cluster includes three VMs, one master node and two worker nodes
- UE, gNb, and the malicious relay are simulated by UERANSIM running in a VM
- The malicious DNS server and the malicious HTTP server are implemented on two VMs
- The playbook server is to automate the deployment of some components, Helm charts are used to deployed CNFs
- The development server for the implementation.
- The IP subnets are assigned to each interface

The security orchestrator components are simply CNF deployed in the K8s cluster, the port mirror N6 is in the UPF.

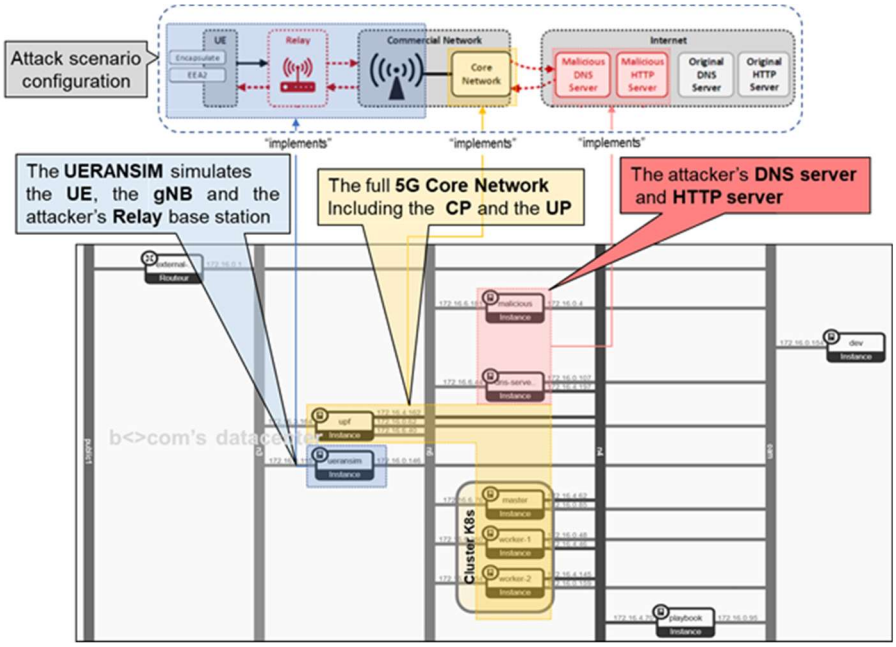


Figure 5-3: The actual platform implementation for the scenario aLTEr

From the simulator UERANSIM, a manipulated DNS packet is issued with the malicious DNS server IP as the bad destination to simulate the attack of MITM and it is sent over the GTP-U tunnel interface. From this moment, the attack is carried out and we will access the performance of the security orchestrator to detect and contain the attack.

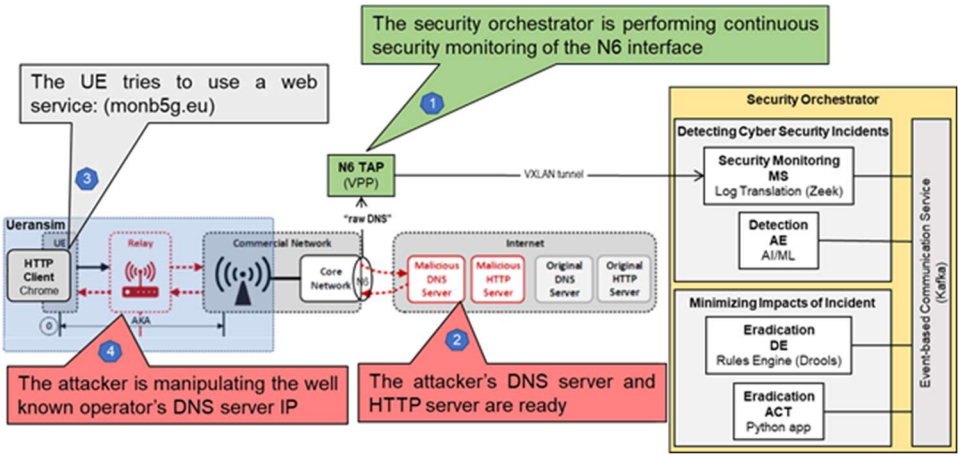


Figure 5-4: The security of data plane is continuously monitored and the aLTEr attack is ongoing.

Figure 5-4 shows the security orchestrator continuously monitoring the data plane by collecting selected data at the N6 interface of the UPF, and the scenario aLTEr is being played out when the user is trying to reach the web site "monb5g.eu". In the next section, we will measure the time to detect and the time to respond of the security orchestrator and see the impact of the attack from the perspective of the user.

The attack is being played out, and the port mirror of the N6 interface is configured to copy and forward data selected protocols such as DNS. The copied data is encapsulated in a VXLAN tunnel to reach the security monitoring MS as shown Figure 5-5.

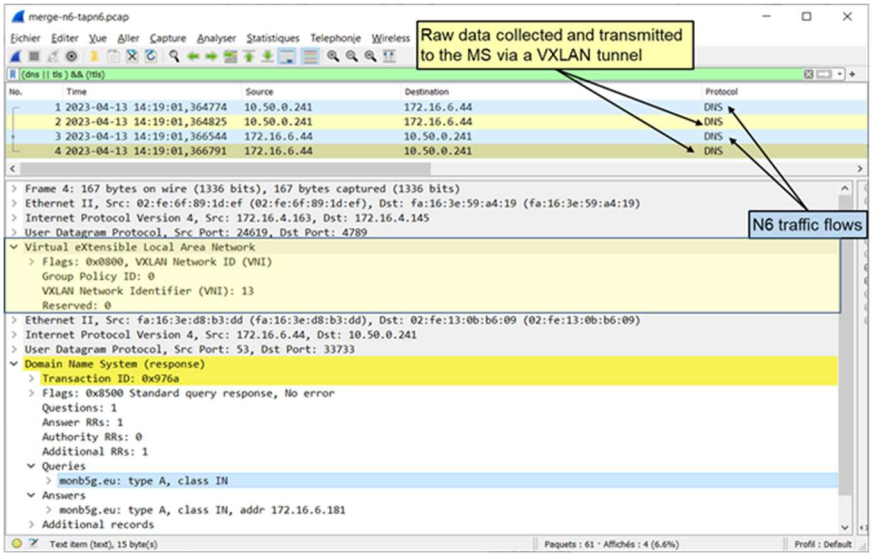


Figure 5-5: Raw data at the N6 interface are collected and sent the MS via a VXLAN tunnel.

The security monitoring MS reassemble packets of DNS transaction and translates them into meaningful logs for analysis. Figure 5-6 shows an example of DNS logs sent by the MS.

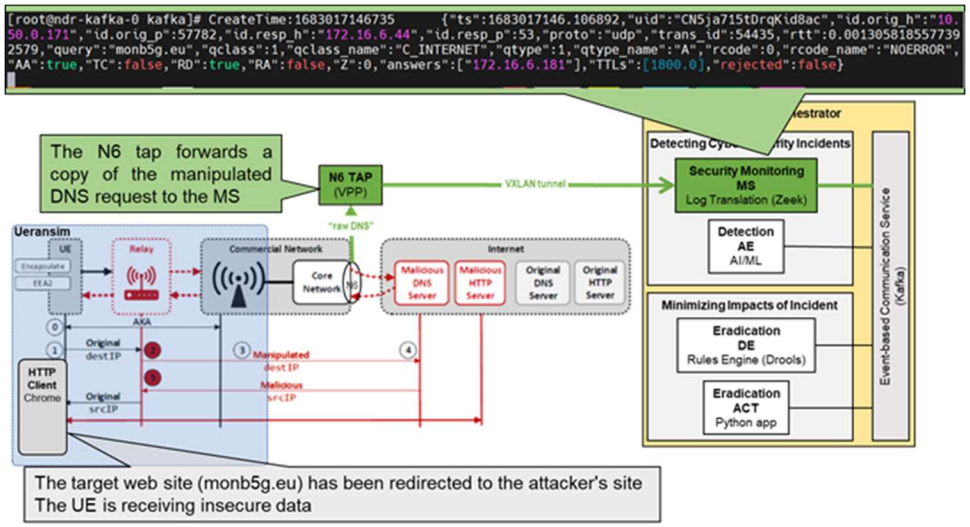


Figure 5-6: Meaningful logs of DNS transactions at the N6 interface received by the AE

Meaningful monitoring logs are sent to the Kafka topic where detection function is consuming data to extracts the features for the model to predict anomalies. Several algorithms have been implemented and their accuracy are described in the [MONB5GD5.4]. If an anomaly in the meaning logs is detected, the detection function creates an incident containing the contextual information and the attack identification and sends it as an event to the topic assigned to the incident response function as shown in Figure 5-7.

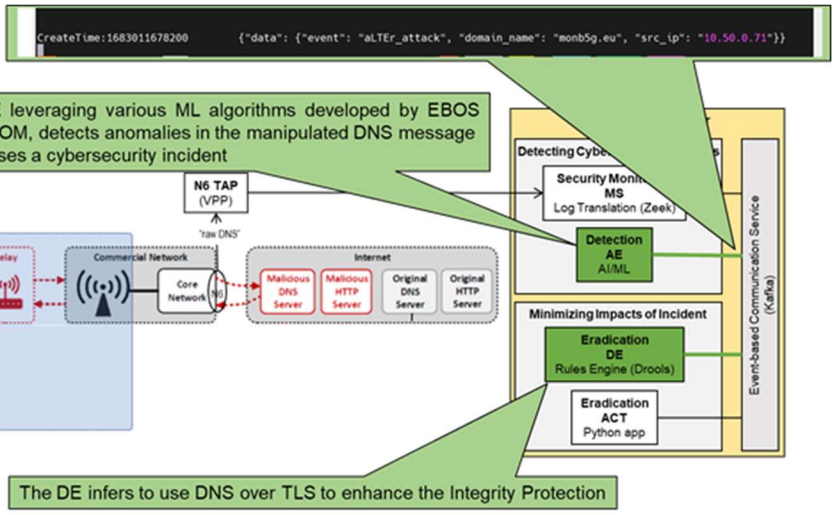


Figure 5-7: The AE raises an event indicating the aLTEr attack is occurring for the UE.

The security expert beforehand prepared the plan to eradicate the aLTEr attack and made up the conditions to trigger them by ingesting rules in the inference engine of the DE. Upon receipt of an incident, the DE will

derive the plan and instruct the ACT to execute the actions of the plan. To eradicate the aLTEr attack, the mitigation could be either:

- activating UP integrity protection in the UP security policy for the PDU session for protection of radio path between the UE and the base station,
- enabling the overlay protection TLS for DNS transactions

In this demo, we choose the second mitigation option to save the cost of protecting the integrity of the PDU session. The second option illustrates the capability of the security orchestrator:

- To deploy on demand security CNF which is the DoT service: the tool Coredns is set up to provide DoT server function, it is deployed as a K8s NodePort service pre-set with the TLS Certificate Authority (CA) information and the port exposed to the UE via the external network,
- To create a new chain of functions by associating the DoT server with the default DNS server, as the DoT server does not resolve domain names itself but forwards requests to the default DNS server,
- And to configure the network function on the UE to switch to DoT transparently for applications: the tool Stubby is installed and configured as the local name server listening the loopback interface and uses the CA certificate and port provided by the ACT to communicate with the DoT server.

To enforce security of the data plane, Figure 59 depicts the ACT that is executing three actions of the eradication plan: deploy the DoT server, chain it with the default DNS server and deploy the DoT client as the "nameserver" on the UE.

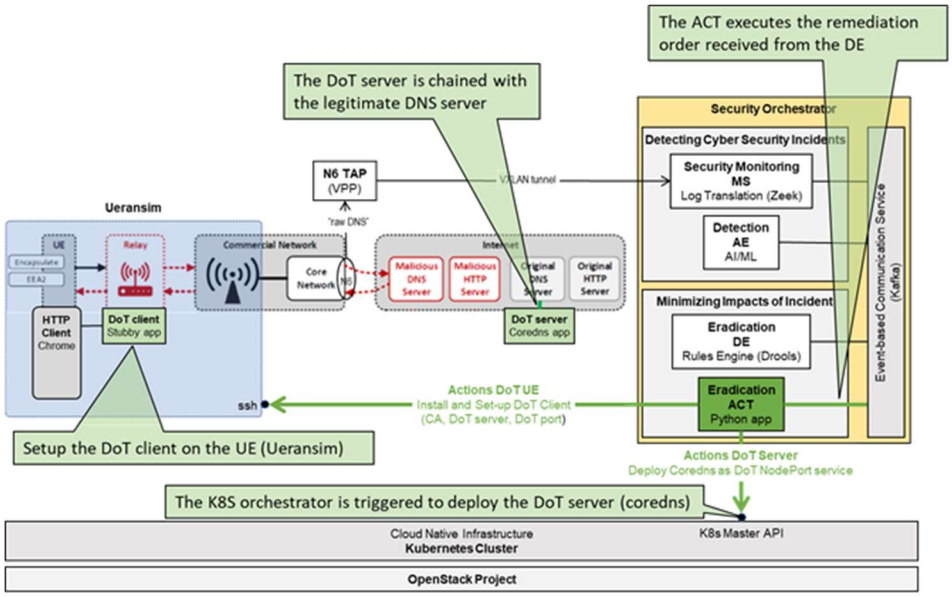


Figure 5-8: The ACT is executing the action plan provided by the DE.

From the user perspective, Google Chrome (GC) is used to view the content of the web site (monb5g.eu), as its DNS request has been manipulated, GC warns it receives insecure contents, as the contents come from the malicious HTTP server. After the first bad DNS transaction, the security orchestrator detects the attack and it applies in order of second DoT as the countermeasure, on the web page refresh, GC will get the secure

contents from the legitimate website monb5g. In Figure 5-9, the data capture at the N6 interface shows that plaintext DNS is replaced by TLS which prevents the attacker from redirecting user traffic.

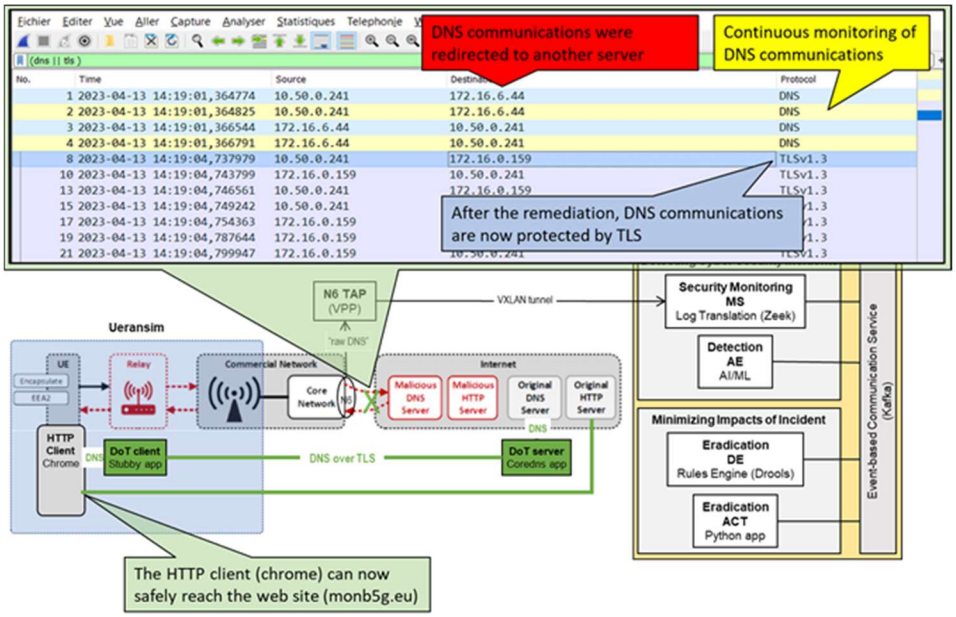


Figure 5-9: Effective protection of DNS transactions using TLS to eradicate the attack aLTEr

5.2 Datasets from experimental trials

5.2.1 MonB5G PoC-2 Scenario 2: FL attack

Cloud-native and containerization have changed the way to develop and deploy applications. Cloud-native rethinks the application architecture by embracing a microservice approach, where each microservice is packaged into containers to run in a centralized or an edge cloud. When deploying the container running the micro-service, the tenant must specify the needed computing resources to run their workload in terms of the amount of CPU and memory limit. However, it is not straightforward for a tenant to know in advance the computing amount that allows running the microservice optimally. This will have an impact not only on the service performances but also on the infrastructure provider, particularly if the resource overprovisioning approach is used. To overcome this issue, we conduct an experimental study aiming to detect if a tenant's configuration allows running its service optimally. We run several experiments on a cloud-native platform, using different types of applications under different resource configurations.

The datasets are collected for 3 types of applications: Web servers written in python and Golang, RabbitMQ data broker and the OpenAirInterface 5G Core network function AMF (Access and Mobility Management Function).

Web Servers:

We used Golang and Python-based web servers for the test. Each request to the web server returns a video of a size 43 MB. For testing we used ApacheBench, a command-line program used for benchmarking HTTP

web servers. ApacheBench allows parallel requests from multiple clients. For each web server instance we send a number of requests ranging from 100 to 1000 and a concurrency level between 1 and 100, representing the number of parallel clients performing the requests.

The information available in the dataset are as follows:

time: timestamp of collection of metrics.

ram_limit: the memory allocated to the container in megabytes.

cpu_limit: the CPU allocated to the container.

ram_usage: the amount of memory used by the container at the time of the metrics collection in byte.

cpu_usage: the amount of CPU used by the container at the time of the metrics collection.

n: the number of requests sent to the container.

c: the concurrency level in the requests.

lat50: the least response time for the best 50% requests in microseconds.

lat66: the least response time for the best 66% requests in microseconds.

lat75: the least response time for the best 75% requests in microseconds.

lat80: the least response time for the best 80% requests in microseconds.

lat90: the least response time for the best 90% requests in microseconds.

lat95: the least response time for the best 95% requests in microseconds.

lat98: the least response time for the best 98% requests in microseconds.

lat99: the least response time for the best 99% requests in microseconds.

lat100: the least response time in microseconds.

5G Core network's AMF:

For testing we use my5G-RANTester, a tool for emulating control and data planes of the UE and gNB (5G base station). The number of simultaneous registration requests that are sent to each instance of the AMF varies between 10 and 400.

The information available in the dataset are as follows:

time: timestamp of collection of metrics.

ram_limit: the memory allocated to the container in megabytes.

cpu_limit: the CPU allocated to the container.

ram_usage: the amount of memory used by the container at the time of the metrics collection in byte.

cpu_usage: the amount of CPU used by the container at the time of the metrics collection.

n: the number of parallel registration requests sent to the AMF.

mean: the mean registration time for all the registration requests in microseconds.

lat50: the median registration time for registration requests in microseconds.

lat75: the least registration time for the best 75% registration requests in microseconds.

lat80: the least registration time for the best 80% registration requests in microseconds.

lat90: the least registration time for the best 90% registration requests in microseconds.

lat95: the least registration time for the best 95% registration requests in microseconds.

lat98: the least registration time for the best 98% registration requests in microseconds.

lat99: the least registration time for the best 99% registration requests in microseconds.

lat100: the least registration time in microseconds.

RabbitMQ data broker:

For testing we used RabbitMQ PerfTest which is a throughput testing tool that simulates basic workloads and provides the throughput and the time that a message takes to be consumed by a consumer. For each deployed RabbitMQ server we used a number of producers and consumers that ranges from 50 to 500. Each producer sends messages to the broker with a rate of 100 messages per second for a period of time of 90 seconds.

The information available in the dataset are as follows:

time: timestamp of collection of metrics.

ram_limit: the memory allocated to the container in megabytes.

cpu_limit: the CPU allocated to the container.

ram_usage: the amount of memory used by the container at the time of the metrics collection in byte.

cpu_usage: the amount of CPU used by the container at the time of the metrics collection.

n: the number of producers sending messages to the RabbitMQ server.

Min: the minimum consumption time for the producer messages.

lat50: the median consumption time for the producer messages.

lat75: the least consumption time for the best 75% messages in microseconds.

lat95: the least consumption time for the best 95% messages in microseconds.

lat99: the least consumption time for the best 99% messages in microseconds.

5.2.2 MonB5G PoC-2 Scenario 3: aLTEr attack

Although the attacker intercepts and modifies the user's packets on the radio interface, the effect produced is in the application domain between the client and the application server. The attacker first seeks to gain control of domain name resolution so that he can then redirect application services to his server. To detect such attack, DNS packets coming from UE are inspected to search for anomalies. AI/ML techniques are used to learn normal behaviour, therefore, we capture data from the IT network, which is then anonymized before being used to train machine learning models.

The Zeek monitoring tool is used to translate the raw IP packets into meaningful records in the form of protocol transactions. Here below is an example of DNS transaction recorded in the dataset.


```
{
  "ts": 1683017146.10689,
  "uid": "CN5ja715tDrqKid8ac",
  "id.orig_h": "10.50.0.171",
  "id.orig_p": 57782,
  "id.resp_h": "172.16.6.44",
  "id.resp_p": 53,
  "proto": "udp",
  "trans_id": 54435,
  "rtt": 0.00130581855773925,
  "query": "monb5g.eu",
  "qclass": 1,
  "qclass_name": "C_INTERNET",
  "qtype": 1,
  "qtype_name": "A",
  "rcode": 0,
  "rcode_name": "NOERROR",
  "AA": false,
  "TC": false,
  "RD": true,
  "RA": true,
  "Z": 0,
  "answers": [
    "172.16.6.181"
  ],
  "TTLs": [
    1800.0
  ],
  "rejected": false
}
```

5.3 Evaluating KPIs of MonB5G Solutions at the PoC-2 Scenario 1 Testbed: mMTC *ATTACK*

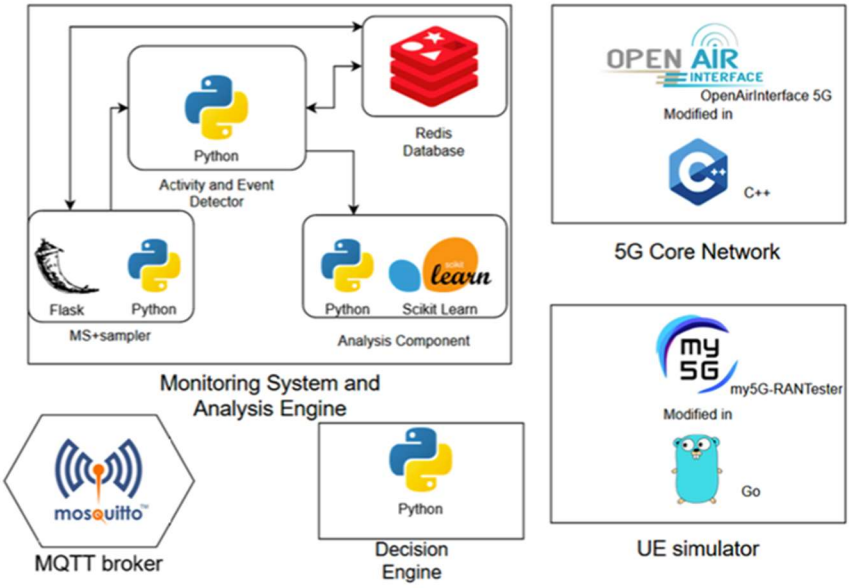


Figure 5-10: Test platform and technological components.

We have implemented the closed-loop control 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 5-10 illustrates the different technologies used to implement the above-mentioned components. The roles of the different components are:

1. MS and sampler: MS is the first component to receive traffic from the 5G CN. It performs basic filtering on it. While the sampler applies sampling of the input data, it receives information on Attach Requests as they are received (with no guaranteed periodicity) and emits periodic data, with the number of Attach Requests received in time intervals of a given length.
2. Activity and Event Detectors: These components receive the sampled data and should detect an event. For each event, these components only emit data at its end, with the number of requests on each time-slice and all the UEs that emitted traffic during the event.
3. Analysis Component: This component runs the ML algorithm on the given data, calculating a detection rate for each time-slice (for all devices in the time-slice).
4. DE: This component receives data from the Analysis Component and decides which devices should be disconnected from the network and then blacklisted.
5. MQTT Broker is used to implement the communication bus between the different components of the closed-loop control system, and between the closed-loop control system and the AMF.

Attack detection and mitigation AE:

Figure 5-11 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.

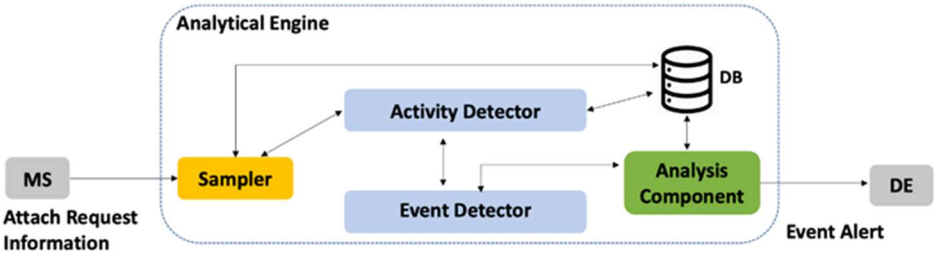


Figure 5-11: AE's components.

We measure the accuracy of the Gradient Boosting algorithm under both normal and abnormal traffic. On normal traffic, the accuracy denotes how often the system yields a detection rate of UEs that is greater than zero. This does not mean that these devices will get banned, but ideally, a value of 0 should be returned for all devices emitting normal traffic. To evaluate our model on normal traffic (True Positive (FP) = False Negative (FN) = 0), we generate data for 500 normal events and run our detection algorithm on each of them. We then counted the number of UEs for which we obtained a greater-than-zero detection rate versus the total number of UEs in all the events. We generate the event duration randomly (between 30 and 250 seconds). Regarding malicious traffic (True Negative (TN) = False Positive (FP) = 0), we also ran 500 tests, but this time, between 7 and 15 Attach Requests are received every 6 seconds, for a total duration that is random between 30 and 250 seconds.

Figure 5-12 allows visualizing the results for normal traffic. The points correspond to the event data, while the green line is the anomaly interval. If a point is outside the limit (in green), it will be assumed as an attack. For normal traffic, the accuracy is computed as 1 − FP/(TN+FP). Hence, the results show an Accuracy on normal traffic of 96.76%. We expected this result as the interval used in our training data includes around 95% of the data in the training dataset, as depicted in Figure 5-12.

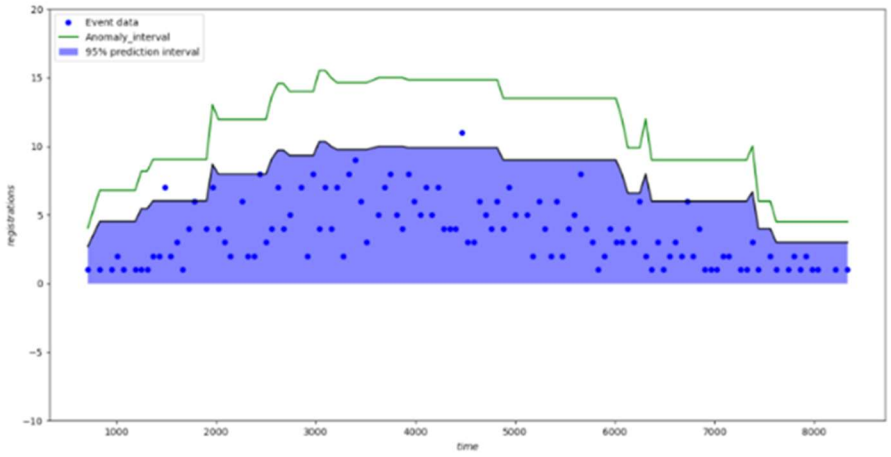


Figure 5-12: Results of the detection algorithm over normal traffic.

Figure 5-13 illustrates the results for malicious attacks. For this FN case, the accuracy is computed as 1−FN/(FN+TP). Hence, the results show an accuracy of 83.63%. This represents an excellent result as banning a relevant part of the devices taking part in a DDoS attack is enough to mitigate it. We recall that this is just the detection rate calculated by the AE component, the final decision regarding the devices that should be banned is taken by the DE component.

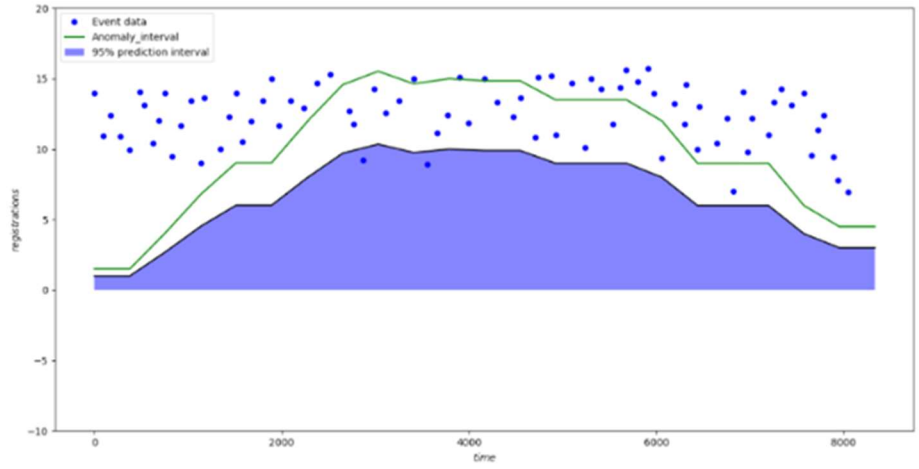


Figure 5-13: Results of the detection algorithm over abnormal traffic.

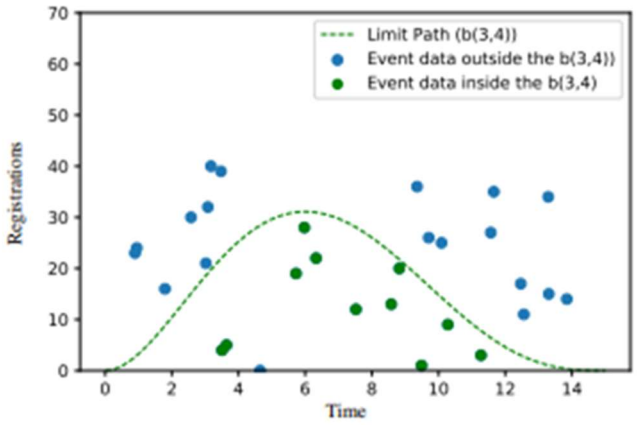


Figure 5-14: The result of the statics method over abnormal traffic

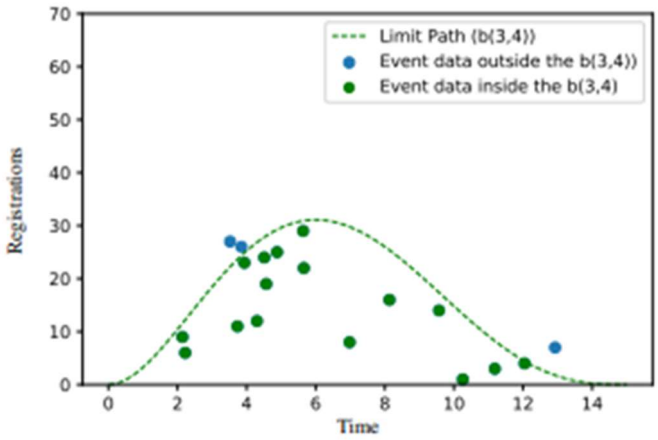


Figure 5-15: The result of the statics method over normal traffic

For the sake of comparison, we used the same scenarios as for Gradient Boosting to generate normal and abnormal traffic. Then, we applied the statistical method and verified its accuracy in detecting attacks. Figure 5-14 illustrates the usage of the statistical method in case of an abnormal event. The discontinue green line shows the β(3, 4) curve obtained according to the event duration. The β(3, 4) curve allows us to have a limited path from which all the outside points are considered anomalies, hence potential attacks. The statistical method's results show that 36.0% of the Attach requests are not following the β(3,4) distribution (they are out of the limit path). Therefore, they can be considered potential attacks. On the other hand, Figure 5-15 presents a test of a normal event. The results show that 90.0% of the Attach Requests follow the β(3,4) distribution. The accuracy of the statistical method to detect anomaly are : ((1 − FP/(TN+FP) = 84.21 % (normal traffic), 1 – FN/(FN+TP) = 57.14 % (abnormal traffic))). We remark that these values are weaker than the ones obtained with the Gradient Boosting algorithm. We argue these differences by the fact that the duration

estimation has a strong impact on the statistical solution. The shape of the β(3, 4) curve changes drastically according to the duration (noted D), which seriously impacts the accuracy. For instance, we change the duration by +/- ϵ = 2sec, and the obtained results are summarized in Table 14. We see clearly from this table the impact of the duration on the accuracy as a small error on the duration drastically yields a drop in the accuracy. Particularly, if the duration is less than the real one, many points will be out of the curve. In the gradient Boosting algorithm, we do not have this concern, as the latter normalizes the sample period duration and uses the trained model to detect the interval.

Table 14: The accuracy of the statistical model considering different Duration values

Method	GB	Static		
		D - Δt	D	D + Δt
Normal traffic	96.76859	45.18924	84.21052	80.24568
Abnormal traffic	83.633191	30.4156	57.142857	51.86854

Table 15 shows the performance of the Gradient Boosting-based solution when modifying the AE_DETECTION_THRESHOLD value. It is worth recalling that this value is used to derive the detection rate and corresponds to a protecting gap to reduce the impact of the ML prediction error and hence reduce the FP value. We remark that the value allowing to reduce both FP and FN is 2.0. Also, when the AE_DETECTION_THRESHOLD value increases, FP is reduced as the FN increases, whereas when it is reduced, both FP and FN increase.

Table 15: Impact of the AE Detection threshold.

AE_DETECTION_THR.	1.2	2.0	3.0
Normal event	82.98654	96.76859	97.64853
Abnormal event	92.92134	97.64643	75.7584

Attack detection and mitigation DE:

DE component is the decision-maker of the closed-loop control 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. 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 stricter, 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. 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 5-16). 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:

1. 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.
2. 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.
3. Third, DE ignores the alert sent by AE, and it will do nothing.

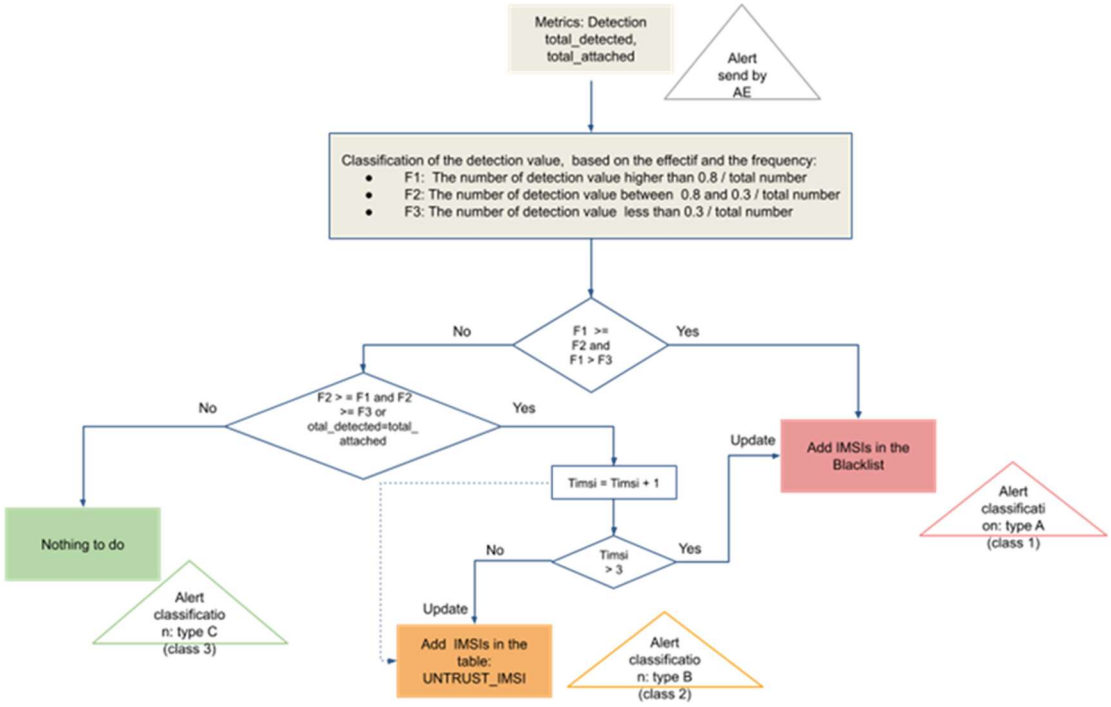


Figure 5-16: Flowchart of the DE's components.

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 Delta (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.

Regarding DE performances, we evaluated the first version that relies on a single threshold DE DETECTION THRESHOLD. Accordingly, in this section, we evaluate the DE DETECTION THRESHOLD impact on the number of blocked devices. Table 16 shows the number of banned UEs for three values of the DE DETECTION

THRESHOLD. As expected, we remark that lower threshold values (ex. 0.1) are very conservative, which allows blocking more UE. While a higher threshold value (ex. 0.8) may be less strict and reduces the number of banned UE. We advise two solutions to fix the DE DETECTION THRESHOLD value. The first one considers the performance limit of the element to protect against DDoS attacks. In our case, we computed the response time of the AMF to Attach Requests while increasing their number. After a certain number of Attach Requests, we noticed that the AMF started to be very slow, which can be caused by a DDoS attack. Therefore, after some tests, we found that the value of DE DETECTION THRESHOLD equal to 0.35, which avoids reaching the number of Attach Request that yields bad AMF performances. Another solution is to use a dynamic threshold value that decreases or increases over time according to the number of consecutive events classified as an attack.

Table 16: Impact of the DE Detection threshold

DE_DETECTION_THR.	0.1	0.35	0.8
Nb (Normal event)	3	1	0
Nb (Abnormal event)	21	16	10

5.4 Evaluating KPIs of MonB5G Solutions at the PoC-2 Scenario 2 Testbed: FL Attack

As depicted in Figure 5-17, 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 Inter-Domain Slice Manager (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 Domain Slice Manager (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 to prevent any latency-related SLA violation. (iii) Building an online Deep Reinforcement Learning (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).

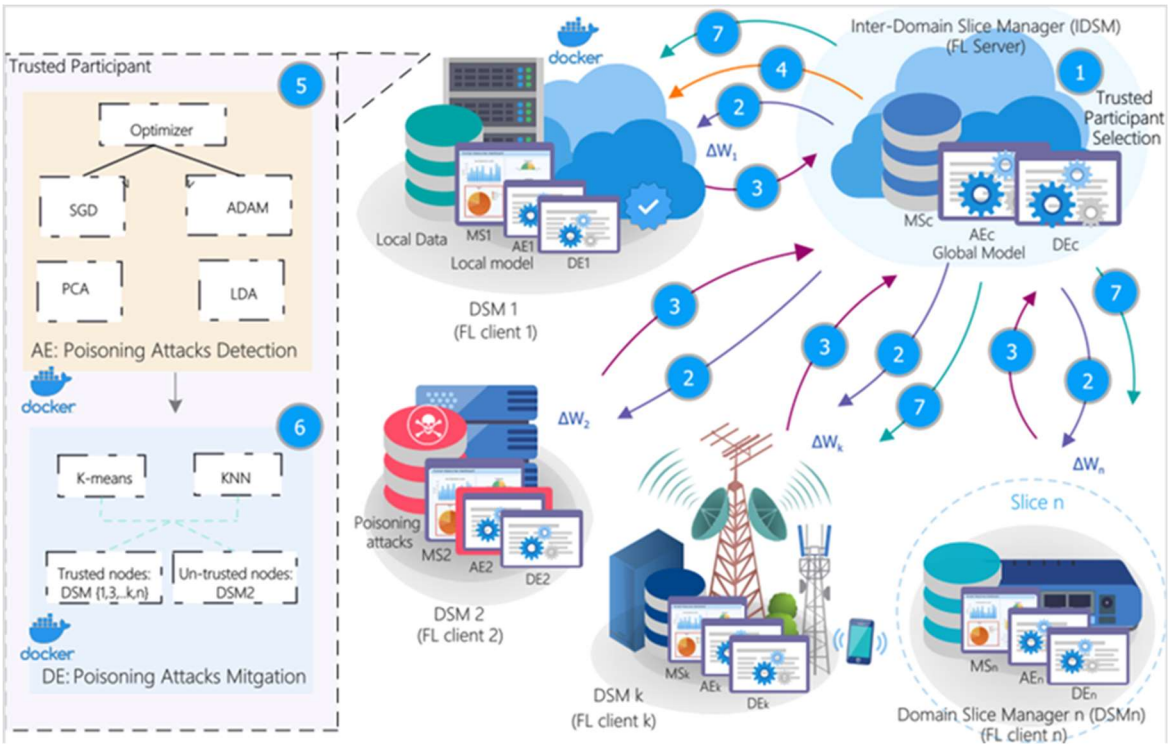


Figure 5-17: Overview of TQDL framework

Because of the lack of real dataset, a real testbed using Eurecom OAI platform is setup to collect and generate a realistic dataset, called EARCD for Eurecom AMF Resource Consumption Dataset. OAI implements 5G radio access and core networks, as open-source software. Ten instances of AMF are emulated, running as VNFs inside ten isolated network slices. The network slices differ from each other, in terms of their AMFs' configurations, for instance: the AMF of network slice 1 has 1GB of memory and 1 CPU, the AMF of network slice 2 has 2GB of memory and 2 CPUs, etc.

In addition, my5G-RANTester tool is used to emulate UEs, one gNB and to generate attach requests, that will be then handled by the different network slices' AMFs. By increasing the number of UEs, up to 560 attach request per second could be generated, covering different traffic density.

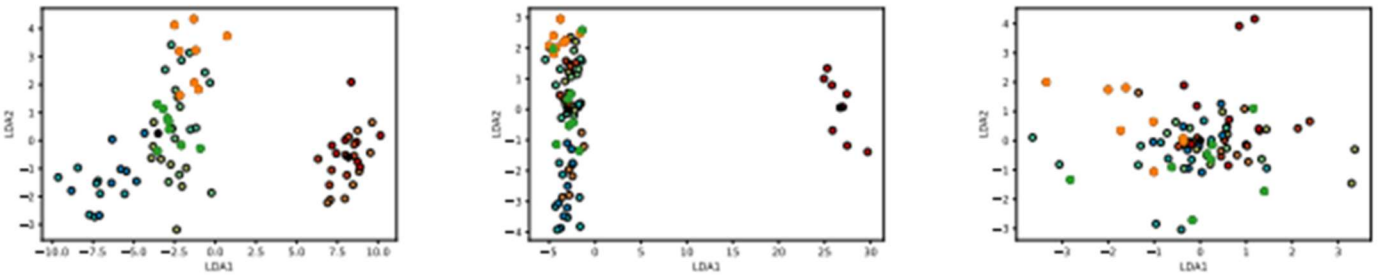
Therefore, ten local datasets (ten network slices) were generated, by varying the number of handled attach request/s, while each dataset contains 2813 samples (rows). In addition, each local dataset contains five features as input data, including RAM capacity, CPU capacity, RAM used, CPU used, and number of attach requests, and latency in terms of average duration of UEs attachment, as output data. The latter corresponds to the response time (latency in second), to handle UE attach requests by the network slices' AMFs.

This section presents the poisoning attack detection scheme, which consists first to elect a network slice participant as a trusted entity. The latter will then be in charge of detecting malicious clients, by leveraging unsupervised learning (Dimensionality Reduction algorithms).

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.

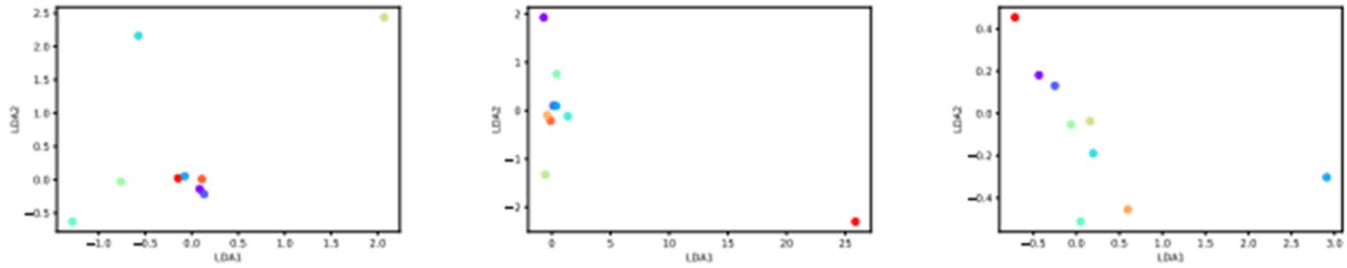
To ensure an effective detection of malicious updates, two different clustering algorithms (unsupervised and supervised) are selected: 1/ k-means which is an unsupervised learning algorithm, that considers no labelled update models' data, and 2/ k-nearest neighbours (KNN), as supervised learning algorithm 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.

Figure 5-18 and Figure 5-20 depict the detection results when combining LDA and K-means on top of ADAM and SGD optimizers, respectively. We also vary the percentage of malicious network slices' DSM and show the trusted nodes that are selected at each FL round (nodes in green color). As we see, our scheme can clearly detect the malicious nodes, even with only one malicious node. Specifically, the trusted node applies both LDA and Kmeans, and then all nodes that are in the same cluster with it are considered correct models, while the nodes (models) that are in the other (s) cluster (s) will be considered as malicious. Therefore, our trust participant selection algorithm helps us not only to select a trusted node but also to determine malicious nodes when performing dimensionality reduction and unsupervised clustering. Moreover, determining the trusted cluster of nodes will also help the FL server (IDSM) to select a trusted participant for the next FL round.



(a) Nb of malicious nodes = 3 (b) Nb of malicious nodes = 1 (c) Nb of malicious nodes = 0

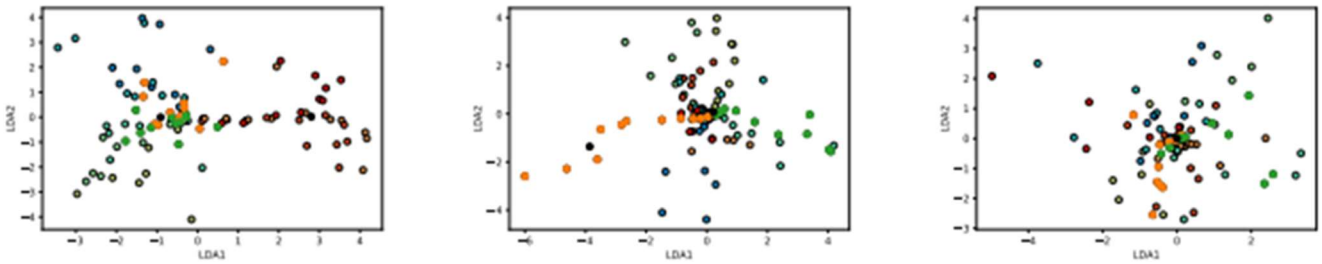
Figure 5-18: LDA + K-means for different number of malicious nodes (ADAM optimizer).



(a) Nb of malicious nodes = 3 (b) Nb of malicious nodes = 1 (c) Nb of malicious nodes = 0

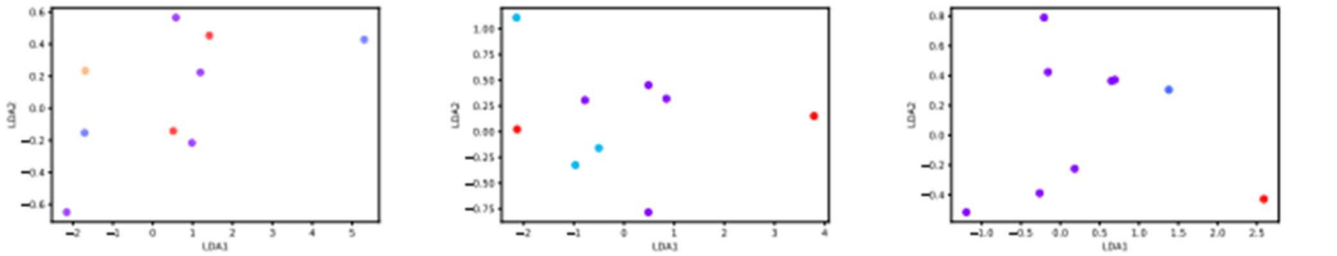
Figure 5-19: LDA + KNN for different number of malicious nodes (ADAM optimizer)

Figure 5-18 and Figure 5-20 also show the clustering of local models when applying both LDA and KNN on top of ADAM and SGD optimizers, respectively. Whatever the number of malicious nodes, we also observe that there are always some isolated points that represent the infected models sent by the malicious DSMs. However, for the LDA technique, we see that the isolated models (infected) are identified better with the ADAM optimizer (Figure 5-18 and Figure 5-19) than with the SGD optimizer (Figure 5-20 and Figure 5-21). Hence, the LDA (with KNN or k-means) technique gives better detection on top of the ADAM optimizer. In fact, these last combinations show the clearest clustering (two separate groups) compared to other algorithms.



(a) Nb of malicious nodes = 3 (b) Nb of malicious nodes = 1 (c) Nb of malicious nodes = 0

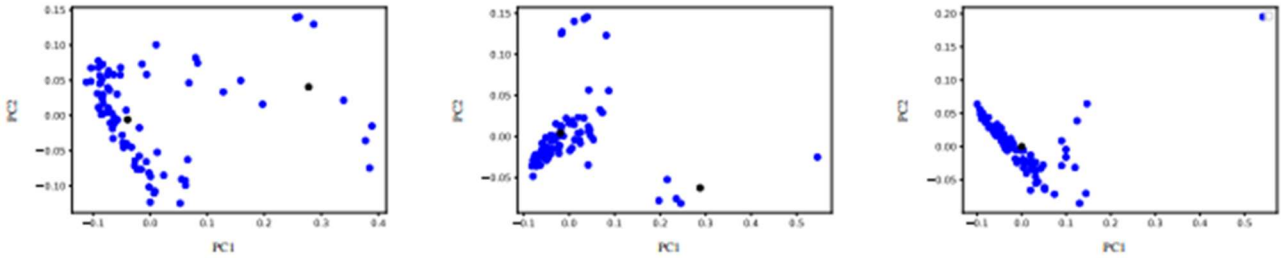
Figure 5-20: LDA + K-means for different number of malicious nodes (SGD optimizer)



(a) Nb of malicious nodes = 3 (b) Nb of malicious nodes = 1 (c) Nb of malicious nodes = 0

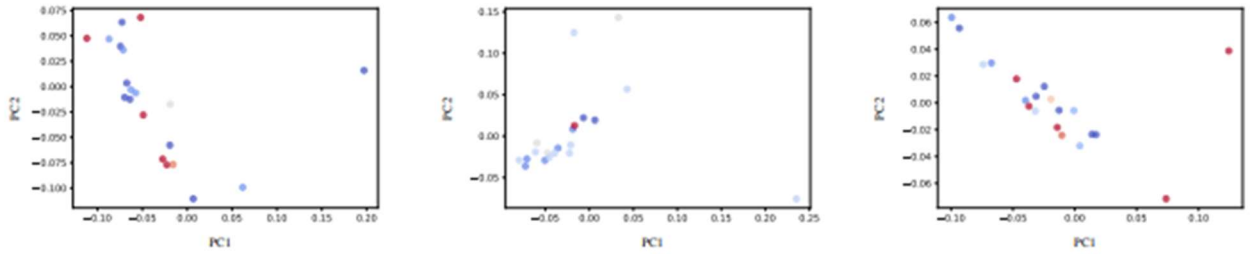
Figure 5-21: LDA + KNN for different number of malicious nodes (SGD optimizer).

As we did for LDA, we also evaluate the performance of PCA technique when combined with clustering unsupervised algorithms. Figure 5-22 and Figure 5-24 shows the clustering detection when combining PCA with K-means, on top of the ADAM and SGD optimizers, respectively. We remark that both optimizers succeed in separating and identifying infected models by incorrect data. However, infected models are better identified on top of the SGD optimizer as compared to the ADAM optimizer. Thus, PCA with K-means gives better performance in detecting infected models on top of the SGD optimizer. Indeed, this last combination exhibits the clearest clustering (two distinct groups) compared to other algorithms.



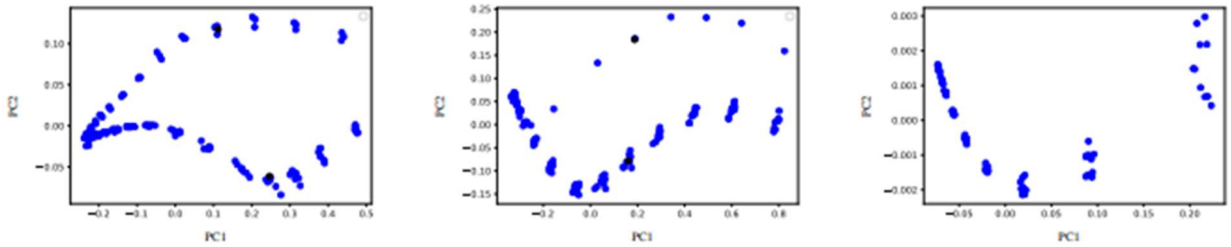
(a) Nb of malicious nodes = 3 (b) Nb of malicious nodes = 1 (c) Nb of malicious nodes = 0

Figure 5-22: PCA + K-means for different number of malicious nodes (ADAM optimizer)



(a) Nb of malicious nodes = 3 (b) Nb of malicious nodes = 1 (c) Nb of malicious nodes = 0

Figure 5-23: PCA + KNN for different number of malicious nodes (ADAM optimizer)

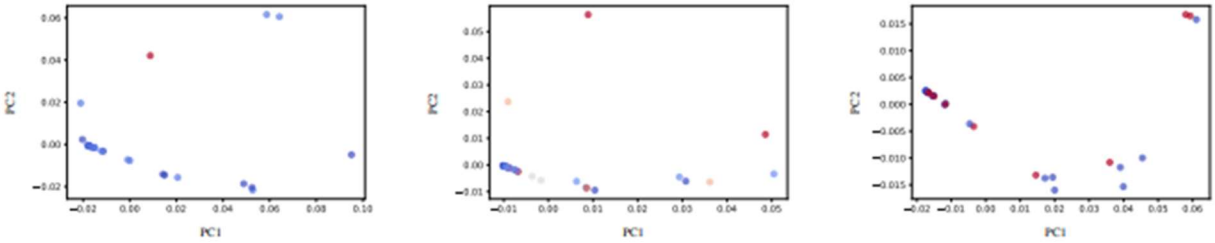


(a) Nb of malicious nodes = 3 (b) Nb of malicious nodes = 1 (c) Nb of malicious nodes = 0

Figure 5-24: PCA + K-means for different number of malicious nodes (SGD optimizer).

Similarly, Figure 5-23 and Figure 5-25 depict the detection when combining PCA with KNN on top of the ADAM and SGD optimizers, respectively. As in Figs. A1 and A3, PCA with KNN on top of both optimizers clearly separates correct local models from infected ones and thus enables detection/identification of malicious

DSMs. We also see that infected models are better identified when leveraging the SGD optimizer than the ADAM optimizer. Therefore, the PCA technique with either K-means or KNN gives better detection of malicious models on top of the SGD optimizer, which is confirmed in Figure 5-22, Figure 5-23, Figure 5-24 and Figure 5-25. In fact, when compared to other techniques, these last combinations show the clearest clustering (two separate groups).



(a) Nb of malicious nodes = 3 (b) Nb of malicious nodes = 1 (c) Nb of malicious nodes = 0

Figure 5-25: PCA + KNN for different number of malicious nodes (SGD optimizer)

5.5 Evaluating KPIs of MonB5G Solutions at the PoC-2 Scenario 3 Testbed: aLTEr attack

In order to measure the time to detect and the time to resolution, several scripts has been implemented and synchronized to:

- Initialize the test procedure by clearing all previous data and starting the consumers to listen events of Kafka topics
- Simulate the attack and generate manipulated DNS requests and replies
- Capture raw data at the GTP-U tunnel interface and translate them to logs with timestamps
- Remove all the deployed countermeasures for the next iteration of the test
- Repeat the previous N times
- Gather the collected logs, extract timestamps and calculate elapsed times
- Visualize the results via bow-plots or graphs

Using these scripts, we can measure the time to detect and the time to respond and calculate their mean values. The time to detect is the elapsed time from the input of the manipulated DNS request to the output of the event indicating a security incident. This duration covers the following operations:

- The raw data collection which includes the copy of DNS packets and their transmission to the MS via a VXLAN tunnel.
- The translation of the raw data to meaningful log, which delay depends on the network protocol specifications and the performance of end points. In fact, Zeek needs all packets of a transaction for the generation of logs, so if the end point is slow to reply or the session times out, the output log will be delayed by as much.
- The extraction of features from the meaning logs and the prediction of anomalies
- The production of the incident event and its posting in Kafka topic

Figure 5-26 below shows the path along which the transit time is measured.

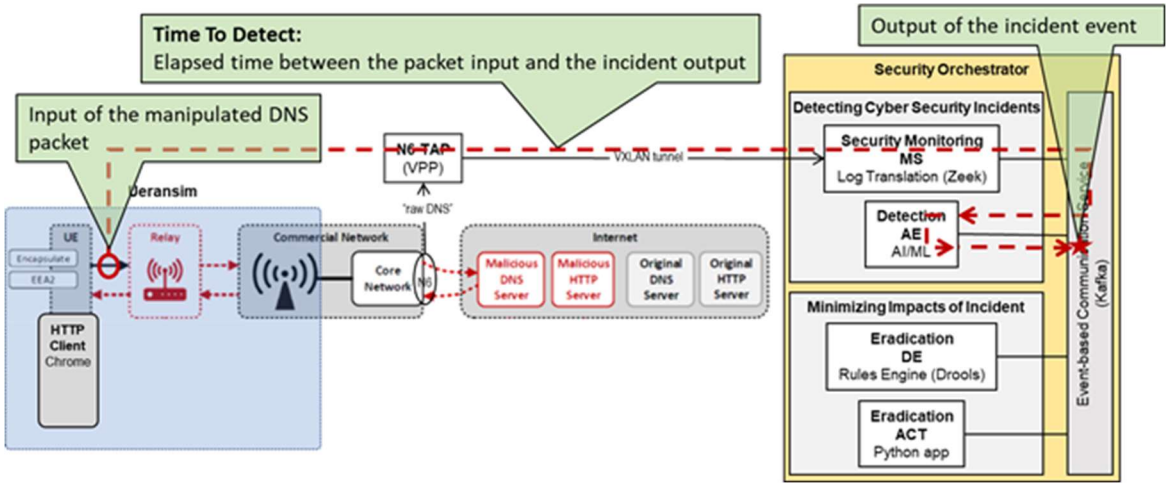


Figure 5-26: The information path defined to measure the attack detection time

As for the time to respond, the duration of following operations must be aggregated:

- The DE derives the action plan from the received event
- The produced action plan is posted in the dedicated topic
- The ACT execute the actions of the eradication plan, which includes:
  - the deployment of the NodePort service of Coredns as the DoT server, and its chaining with the default DNS server
  - The installation of Stubby on the UERANSIM and its configuration as the name server to communicate with the DoT server

The Figure 5-27 depicts the path to measure the time to respond.

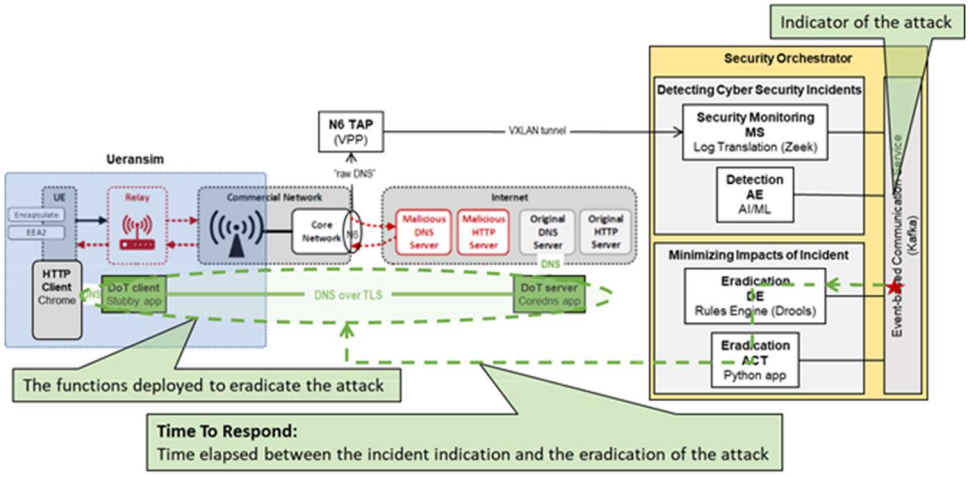


Figure 5-27: The information path defined to measure the attack response time

The attack scenario and the response of the security orchestrator are run automatically and repeatedly 100 times, and times and logs are collected at the end to calculate the Mean Time To Detect (MTTD) and the Mean Time to Respond (MTTR).

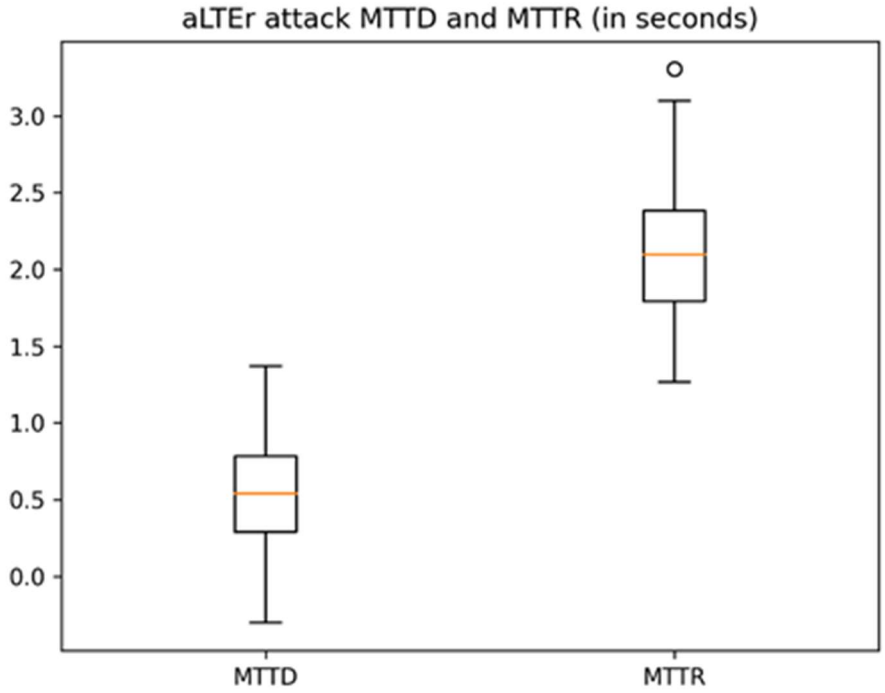


Figure 5-28: MTTD and MTTR measured over 100 attacks and mitigations.

The results in Figure 5-28 show the performance of the security orchestrator to detect and remediate aLTEr threat in order of seconds. Although the incident handling playbook suggested by the cybersecurity standards is partly implemented here, this use case demonstrates deploying automation with AI and security tools helps reduce the time to identify and eradicate threats. Moreover, the availability of exposed API to configure security and network functions, and to manage their lifecycle via the domain orchestrator such as Kubernetes master API widens the scope of possibilities to react to a threat, as a result, it enhances the dynamics of defense. All these time-saving advantages in responding to an attack come at a cost. Actually, automation involves the preparation of procedures and their integration into the protected systems and impact assessment on the services. All these steps require a significant investment in time, resources, and expertise.

6 Lessons Learnt from Experiments

Some lessons learnt from experimental evaluations are as follows:

VR streaming over 5G is not only related to 5G conditions. Our experiment results show that the resulting end-to-end performance of VR streaming application does depend on several factors which are not part of the 5G network performance. During delivering a real-time video feed, the processing delay introduced in video processing can create bottlenecks even in the case of high speed 5G communications.

RTSP versus HTTP: In our experiments focused on video streaming use cases, we compared the performance of two popular protocols, namely HTTP and RTSP. One notable observation was the contrasting behavior exhibited by these protocols. Firstly, RTSP enables real-time negotiation between the client and server, allowing dynamic adjustments to streaming parameters based on network conditions and client capabilities. This feature ensures better delivery of real-time content by adapting to varying bandwidth and latency. Additionally, RTSP provides fine-grained control over the streaming process, enabling clients to specify media tracks, set playback options, and perform interactive operations. While HTTP suffices for video streaming, RTSP has shown itself to be richer in terms of performance and complex dynamics (e.g., varying data rates).

Transcoding schemes: The sensitivity of VR streaming server to transcoding schemes is high. Application server may not use high CPU if only streaming traffic is controlled inside the container (e.g., on the order of 4 milicores for a 4k video traffic). However, after transcoding scheme on files are applied and they are streamed[5] simultaneously (which is called “transcode on the fly”, to define transcoding and streaming simultaneously.), the CPU consumption increases dramatically. Trade-off analysis between extra computation for transcoding versus larger data volume need to be evaluated carefully as well.

Learning different time patterns of number of streaming users: In our experiments, each site had different number of VR streaming clients appearing and exiting the cell site. Therefore, many different time patterns can emerge in case FL process needs to be trained. This creates a huge amount of training scenarios which should be handled to satisfy real-world deployments.

Logging with toolset: When tracking the logs of the aggregation server for example, relying only on one set of toolsets to create pipeline (e.g., logstash to collect logs) can create bottlenecks. For this reason, we have also used filebeat to improve the scalability of log collection process.

Remote UE: during the process of setting up Amarisoft's remote User Equipment (UE) for Simbox, we encountered several valuable lessons. Firstly, it is crucial to ensure compatibility and proper configuration between the remote UE (realized as a VM with GUI) and the Simbox platform. While this is a very convenient way of emulating UEs, not only at the radio level but also at the performance and user experience level, careful attention should be given to version compatibility and firmware updates to avoid any issues or conflicts. Secondly, network connectivity plays a significant role in establishing a reliable connection between the remote UE and the Simbox. To that aim, we ensured minimum link capacity between Simbox and the VM, so to avoid such a link becoming a bottleneck, rather than the 5G NR links. With that, we ensured almost

[5] Online: https://bit.ly/3GuLCen , Available: April 2023.

negligible overheads in the link between Simbox and Remote UE, enabling a transparent RAN such as if the Remote UE was directly connected to the gNB through the 5G NR.

Cybersecurity handling: The cybersecurity incident handling guideline defined by the security standards provide a template for the procedure to be followed to detect and resolve an incident. The incident handling template contains a number of functions and sub-procedures that can be executed using a closed-loop automation system. The aLTEr attack demo partially implements the main security incident handling loop, which is made possible by an in-depth analysis of the attack's modus operandi and appropriate countermeasures. The architecture of the MonB5G security orchestrator then enables such a model to be implemented to automate procedures using AI tools (ML, inference engine) and security tools. The advantage of such an approach is obviously a faster identification and removal of the threat compared with a process involving human intervention. Nonetheless, some steps should be left to the human expert control in the closed-loop system to validate the analysis results and decide on the actions to carry out if the attack is uncertain or complex.

Using the domain orchestrator to manage the lifecycle of security functions and modify network services opens up defense possibilities. The security expert then has more resources available to define his defense strategies, making them dynamic and adapted to the re-evaluated risks and therefore more difficult for the attacker to circumvent.

To use open interfaces to apply actions, it is necessary to configure their access control by creating a client identity and giving it the necessary authorizations. Lifecycle management APIs have a major impact on the proper operation of services, and the impact of their applications must be carefully studied and measured.

The benefits of automating and orchestrating security come at a cost. The preparation of procedures includes the activities such as:

- acquiring tools that make incident handling more efficient,
- understand the normal behavior of networks and services so that abnormal cases can be easily discovered,
- choosing the most effective remediation strategy,
- and integrating into the protected system.

All these steps require a significant investment in time, resources and expertise.

7 Conclusions

This deliverable presented the outcomes of the MonB5G project, which aims to achieve zero-touch management and orchestration for network slicing in 5G LTE and beyond. The performance validation of MonB5G components, including architecture, algorithms, and AI techniques, was conducted in the testbeds to assess their capabilities. The KPIs defined in previous work packages are used for comparison and evaluation. The deliverable showcased the achievements of two PoCs. In PoC-1 scenario 1, the data-driven management systems demonstrated their ability to guarantee stringent end-to-end service level agreements for the considered B5G application, in particular VR video streaming application. The automated zero-touch service management and redundancy mechanisms significantly reduced downtime and improved high availability. In PoC-1 scenario 2, the MonB5G mechanisms effectively responded to local performance issues and changes in traffic patterns. Special emphasis is placed on optimizing the RAN sub-slice through data-driven radio resource management mechanisms that rely on distribute AI techniques.

PoC-2 scenario 1 focused on addressing in-slice DDoS attacks in mMTC network slices. The proposed zero-touch security management solution successfully detected and mitigated DDoS attacks initiated by compromised MTC devices within a network slice. In PoC-2 scenario 2, the ZSM concept in B5G networks is explored. The use of FL techniques enabled the automated management and orchestration of running network slices while preserving privacy and isolation. The scenario specifically addressed the robustness of FL algorithms against attacks, including poisoning attacks. PoC-2 scenario 3 presented a zero-touch security management solution for detecting and mitigating aLTEr attacks, which exploited vulnerabilities between user equipment and gNodeBs. AI/ML algorithms were employed for attack detection, and the security orchestrator updates security policies to mitigate the attacks.

The performance validations of MonB5G enablers demonstrated their compliance with KPIs and the superiority of MonB5G architecture, components, and algorithms over baseline mechanisms. The achievements included enhanced network performance, reduced reaction time to malfunctions, improved resource allocation, energy efficiency, cost-effectiveness, and faster convergence on training. The solutions effectively addressed security challenges, such as DDoS attacks and poisoning attacks, with reduced detection time, low false positive rates, and robust learning processes. The lessons learned from the experimental demonstrations contributed to further refining and improving the MonB5G framework.

Overall, the validity of the MonB5G architecture is quantified through the experimental performance analysis of MonB5G enablers and their compliance with defined KPIs in WP2 are validated by experimental KPIs. The tangible results obtained from the experimental testbed platforms in this deliverable successfully demonstrated the realization of the intelligent, decentralized, and secure zero-touch management and orchestration vision in B5G networks.

8 Appendixes

Appendix A

Below are the yaml files that are used to instantiate FL training and inference process.

Site-C server yaml file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: fl-server-pod
spec:
  selector:
    matchLabels:
      run: fl-server-pod
  replicas: 1
  template:
    metadata:
      labels:
        run: fl-server-pod
        app: fl-client-1-pod
        app: fl-client-2-pod
    spec:
      containers:
      - name: fl-server-pod
        image: ezeydan/myrepo:server-demo-with-policy-with-updates-v14
        ports:
        - name: fl-server-pod
          containerPort: 8081
        imagePullPolicy: IfNotPresent
        stdin: true
        tty: true
        volumeMounts:
```

```
      - name: log
        mountPath: /var/log/
      env:
      - name: ISTIO_META_POD_NAME
        valueFrom:
          fieldRef:
            fieldPath: metadata.name
      - name: ISTIO_META_POD_NAMESPACE
        valueFrom:
          fieldRef:
            fieldPath: metadata.namespace
      - name: NUMBER_ROUNDS
        value: "30"
      - name: NUMBER_SELECTED_CLIENTS
        value: "2"
      - name: FL_SERVER_POD_SERVICE_PORT
        value: "8081"
      - name: PYTHONUNBUFFERED
        value: "1"
      - name: CSV_FILE_LOCATION
        value: "br_cpu_dataset_even.csv"
      - name: SERVER_BATCH_SIZE
        value: "240"
    - name: filebeat
      image: elastic/filebeat:7.16.3
      args:
        - -c
        - /etc/filebeat/conf.yaml
        - -e
      volumeMounts:
        - name: filebeat-config
```

```
            mountPath: /etc/filebeat
          - name: log
            mountPath: /var/log/
      volumes:
        - name: log
          emptyDir: {}
        - name: filebeat-config
          configMap:
            name: filebeat-config
---
apiVersion: v1
kind: Service
metadata:
  name: fl-server-pod
  labels:
    run: fl-server-pod
    app: fl-client-1-pod
    app: fl-client-2-pod
spec:
  type: NodePort
  ports:
  - port: 8081
    targetPort: 8081
    nodePort: 30000
  externalIPs:
    - 10.0.42.7
  selector:
    run: fl-server-pod
    app: fl-client-1-pod
    app: fl-client-2-pod
```

Site-B client yaml file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: fl-client-2-pod
spec:
  selector:
    matchLabels:
      run: fl-client-2-pod
  replicas: 1
  template:
    metadata:
      labels:
        run: fl-client-2-pod
        app: fl-server-pod
    spec:
      containers:
      - name: fl-client-2-pod
        image: ezeydan/myrepo:client-1-with-policy-v11
        ports:
         - name: fl-client-2-pod
           containerPort: 2052
        imagePullPolicy: IfNotPresent
        env:
        - name: ISTIO_META_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: ISTIO_META_POD_NAMESPACE
          valueFrom:
            fieldRef:
```

```
              fieldPath: metadata.namespace
        - name: SERVER_URL
          value: "http://10.0.42.7:8081"
        - name: CLIENT_URL
          value: "http://10.0.42.3:2052"
        - name: PORT_CLIENT
          value: "2052"
        - name: CSV_FILE_LOCATION
          value: "/var/data/br_cpu_dataset_odd.csv"
        - name: BATCH_SIZE
          value: "2400"
        - name: N_EPOCHS
          value: "60"
        - name: BOOTSTRAP_KAFKA_IP_PORT
          value: "10.109.107.159:9094"
        - name: TOPIC
          value: "ms.*"
        volumeMounts:
        - name: data
          mountPath: /var/data
      volumes:
      - name: data
        configMap:
          name: my-csv-configmap
---
apiVersion: v1
kind: Service
metadata:
  name: fl-client-2-pod
  labels:
    run: fl-client-2-pod
```

```
spec:
  type: NodePort
  ports:
   - port: 2052
     targetPort: 2052
     nodePort: 31516
  externalIPs:
   - 10.0.42.3
  selector:
    run: fl-client-2-pod
```

Site-A-client yaml file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: fl-client-1-pod
spec:
  selector:
    matchLabels:
      run: fl-client-1-pod
  replicas: 1
  template:
    metadata:
      labels:
        run: fl-client-1-pod
        app: fl-server-pod
    spec:
      containers:
      - name: fl-client-1-pod
        image: ezeydan/myrepo:client-1-with-policy-v11
```

```
      ports:
       - name: fl-client-1-pod
         containerPort: 2051
      imagePullPolicy: IfNotPresent
      env:
      - name: ISTIO_META_POD_NAME
        valueFrom:
          fieldRef:
            fieldPath: metadata.name
      - name: ISTIO_META_POD_NAMESPACE
        valueFrom:
          fieldRef:
            fieldPath: metadata.namespace
      - name: SERVER_URL
        value: "http://10.0.42.7:8081"
      - name: CLIENT_URL
        value: "http://10.0.42.2:2051"
      - name: PORT_CLIENT
        value: "2051"
      - name: CSV_FILE_LOCATION
        value: "/var/data/br_cpu_dataset_even.csv"
      - name: BATCH_SIZE
        value: "2400"
      - name: N_EPOCHS
        value: "60"
      - name: BOOTSTRAP_KAFKA_IP_PORT
        value: "10.109.107.159:9094"
      - name: TOPIC
        value: "ms.*"
      volumeMounts:
```

```
        - name: data
          mountPath: /var/data
      volumes:
      - name: data
        configMap:
          name: my-csv-configmap
---
apiVersion: v1
kind: Service
metadata:
  name: fl-client-1-pod
  labels:
    run: fl-client-1-pod
spec:
  type: NodePort
  ports:
    - port: 2051
      targetPort: 2051
      nodePort: 31515
  externalIPs:
    - 10.0.42.2
  selector:
    run: fl-client-1-pod
```

Logstash.cm

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: logstash
```

```
  labels:
    component: logstash
data:
  access-log.conf: |
    input {
      beats {
        port => "5044"
      }
    }
    filter {
      json {
        source => "message"
      }
    }
    output {
      elasticsearch {
        hosts => [ "elasticsearch:9200" ]
        index => "fl-server-pod"
      }
    }
```

Logstash.yaml

```
      apiVersion: apps/v1
      kind: Deployment
      metadata:
        name: logstash
        labels:
          component: logstash
```

```
  spec:
   strategy:
    type: Recreate
   selector:
    matchLabels:
     component: logstash
   template:
    metadata:
     labels:
      component: logstash
    spec:
     containers:
      - name: logstash
        image: logstash:7.16.3
        ports:
         - containerPort: 5601
        ports:
         - containerPort: 5044
        volumeMounts:
         - name: logstash-config
           mountPath: /usr/share/logstash/pipeline
     volumes:
      - name: logstash-config
        configMap:
         name: logstash
  ---
  apiVersion: v1
  kind: Service
```

```
metadata:
  name: logstash
  labels:
    component: logstash
spec:
  ports:
  - port: 5044
  selector:
    component: logstash
```

Elasticsearch.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: elasticsearch
  labels:
    component: elasticsearch
spec:
  strategy:
    type: Recreate
  selector:
    matchLabels:
      component: elasticsearch
  template:
    metadata:
      labels:
        component: elasticsearch
    spec:
      containers:
        - name: elasticsearch
          image: elasticsearch:7.17.8
          resources:
            requests:
              cpu: "1"
              memory: "2Gi"
            limits:
              cpu: "2"
              memory: "4Gi"
```

```
        ports:
          - containerPort: 9200
            name: client
          - containerPort: 9300
            name: nodes
        env:
          - name: JAVA_TOOL_OPTIONS
            value: -Xmx256m -Xms256m
          - name: discovery.type
            value: single-node
---
apiVersion: v1
kind: Service
metadata:
  name: elasticsearch
  labels:
    component: elasticsearch
spec:
  ports:
  - port: 9200
    name: client
  - port: 9300
    name: nodes
  selector:
    component: elasticsearch
```

Filebeat.cm

```
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: filebeat-config
  labels:
    component: filebeat
data:
  conf.yaml: |
    filebeat.inputs:
    - type: log
      paths:
        - '/var/log/*.log'
    output:
      logstash:
        hosts: [ "logstash:5044" ]
        bulk_max_size: 1024
        index: "fl-server-logs"
        pipeline: "fl-server-logs-pipeline"
```

Kibana.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kibana
  labels:
    component: kibana
spec:
  strategy:
    type: Recreate
  selector:
    matchLabels:
      component: kibana
```

```
  template:
    metadata:
      labels:
        component: kibana
    spec:
      containers:
        - name: kibana
          image: kibana:7.4.1
          resources:
            requests:
              cpu: "1"
              memory: "2Gi"
            limits:
              cpu: "2"
              memory: "4Gi"
          ports:
            - containerPort: 5601
---
apiVersion: v1
kind: Service
metadata:
  name: kibana
  labels:
    component: kibana
spec:
  ports:
  - port: 5601
  selector:
    component: kibana
```

Appendix B

The detailed steps of visualization platform are given below.

1. A file named filebeat.cm.yaml is used to store the Filebeat configuration file. The input of Filebeat is what is read from files /var/log/*.log, then these logs are output to Logstash.
1. In the fl-aggregation-server file, we mount the Filebeat configuration file into the /etc/filebeat/conf.yaml file and use the args to specify that configuration file for Filebeat. The fl-aggregation-server application container writes a log to the file /var/log/access.log. We use emptyDir volumes to share storage between two containers.
2. A file named logstash.cm.yaml is used to store the Logstash configuration file. A Logstash Deployment file named logstash.yaml is created. We mount the configuration file to the folder /usr/share/logstash/pipeline, Logstash will load the configuration files from this folder.
3. Elasticsearch and Kibana deployment files named elasticsearch.yaml and kibana.yaml are later created.
4. Run apply command to create resources.
5. Use port-forward to access Kibana Dashboard.
6. Now, in Kibana go to menu Stack Management > Index patterns and create an index pattern, then go to menu Discover and the logs collected from the fl-aggregation-server can be observed.

9 References

[Amarisoft] "AMARI Callbox Ultimate," https://www.amarisoft.com/app/uploads/ 2022/10/AMARI-Callbox-Ultimate.pdf, 2023, [Online; accessed 2023-01-05].

[Chergui2020] H. Chergui and C. Verikoukis, "Offline SLA-Constrained Deep Learning for 5G Networks Reliable and Dynamic End-to-End Slicing," IEEE Journal on Selected Areas in Communications, vol. 38, no. 2, pp. 350-360, Feb 2020.

[Chergui2021] H. Chergui, L. Blanco and C. Verikoukis, "CDF-Aware Federated Learning for Low SLA Violations in Beyond 5G Network Slicing," in IEEE ICC, 2021.

[Chergui2021TWC] Chergui, Hatim, Luis Blanco, and Christos Verikoukis. "Statistical federated learning for beyond 5G SLA-constrained RAN slicing." IEEE Transactions on Wireless Communications 21.3 (2021): 2066-2076.

[DDQN] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," 2015, doi: 10.48550/ARXIV.1509.06461. [Online]. Available: https://arxiv.org/abs/1509.06461

[DEEPCOG] D. Bega, M. Gramaglia, M. Fiore, A. Banchs and X. Costa-Perez, "DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning," *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, Paris, France, 2019, pp. 280-288, doi: 10.1109/INFOCOM.2019.8737488.

[Graph2023] A. Chawla, A. Bosneag, A. Dalgkitsis, "Graph-based Interpretable Anomaly Detection Framework for Network Slice Management in Beyond 5G Networks", IEEE NOMS 2023, Miami, USA, 8-12 May 2023.

[IBM2022] IBM. (2022). *Cost of a Data Breach - Report 2022.* IBM .

[MONB5GD22] Deliverable D2.2 - Techno-economic analysis of the beyond 5G environment use case requirements and KPIs, 2022.

[MONB5GD24] Deliverable D2.4 - Final release of the MonB5G architecture (including security), 2022.

[MONB5GD31] Deliverable D3.1 - Initial report on AI driven techniques for the MonB5G AE/MS, 2022.

[MONB5GD32] Deliverable D3.2 - Final Report on AI-driven Techniques for the MonB5G AE/MS, 2022.

[MONB5GD33] Deliverable D3.3 - Report on Integration and testing of the MonB5G AE and MS, 2023.

[MonB5GD42] MonB5G Deliverable D4.2 "Final Report on AI-driven Techniques for the MonB5G Decision Engine", 2022.

[MonB5GD43] MonB5G Deliverable D4.3 "Report on Integration and testing of the MonB5G DE", 2023.

[MonB5GD52] MonB5G Deliverable D5.2 “Final report on AI driven MonB5G security techniques”, 2023.

[MONB5GD61] MonB5G Deliverable D6.1 - Technical Report on System Integration and Operation, 2023.

[SAC-DQN1] W. F. Villota Jácome, O. M. Caicedo Rendon, and N. L. S. da Fonseca, “Admission control for 5G network slicing based on deep reinforcement learning,” Apr. 2021, doi: 10.36227/techrxiv.14498190

[SAC-DQN2] S. Bakri, B. Brik, and A. Ksentini, “On using reinforcement learning for network slice admission control in 5G: Offline vs. online,” International Journal of Communication Systems, vol. 34, no. 7, pp. 1–12, May 2021, doi: 10.1002/dac.475

[Sergio2023] Sergio Barrachina, Engin Zeydan, & Luis Blanco. (2023). Dataset from VR Streaming Server (Emulated) and Radio Access Network for Streaming Traffic [Data set]. Zenodo. https://doi.org/10.5281/zenodo.7944285

[Tadesse2017] Tadesse, Senay Semu, Francesco Malandrino, and Carla-Fabiana Chiasserini. "Energy consumption measurements in docker." 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC). Vol. 2. IEEE, 2017.

[TVT2022] F. Rezazadeh, L. Zanzi, F. Devoti, H. Chergui, X. Costa-Pérez and C. Verikoukis, "On the Specialization of FDRL Agents for Scalable and Distributed 6G RAN Slicing Orchestration," in IEEE Transactions on Vehicular Technology, vol. 72, no. 3, pp. 3473-3487, March 2023, doi: 10.1109/TVT.2022.3218158.

[Zhu2019] Zhu, Haishan, et al. "Kelp: Qos for accelerated machine learning systems." 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2019.