Grant Agreement No.: 101016509
Research and Innovation action
Call Topic: ICT-40-2020: Cloud Computing

CHARITY
Cloud for Holography and Augmented RealITY

Cloud for Holography and Cross Reality

D2.2: Edge and cloud infrastructure resource and computational continuum orchestration system

Version: v1.9

Deliverable type	R (Document, report)
Dissemination level	PU (Public)
Due date	30/04/2024
Submission date	30/04/2024
Lead editor	Tarik Taleb (ICT-FI)
Authors	Tarik Taleb (ICT-FI), Hao Yu (ICT-FI), Nora Taleb (ICT-FI), Tarik Zakaria Benmerar (ICT-FI), Yan Chen (ICT-FI), Qize Guo (ICT-FI), Abderrahmane Boudi (ICT-FI), Theodoros Theodoropoulos (HUA), Antonios Makris (HUA), Konstantinos Tserpes (HUA), Mike McElligott (Collins), Laura Sande (PLEXUS), Thomas Loven (PLEXUS), Yago González (PLEXUS), Peter Gray (CS), Paolo Barone (HPE), Giovanni Giuliani (HPE), Elena Spatafora (HPE), Alessandro Romussi (HPE), Luca Ferrucci (CNR), Massimo Coppola (CNR), Emanuele Carlini (CNR), Patrizio Dazzi (CNR), Luís Rosa (ONE), Luis Cordeiro (ONE), Luís Ferreira (ONE), Diogo Fevereiro (ONE), Ferran Diego (TID), Aravindh Raman (TID).
Reviewers	Philip Harris (Collins), Fermin Calvo (PLEXUS)
Work package, Task	WP2
Keywords	Extended Reality, Immersive Services, Orchestration, Cloud, Edge Cloud, Cloud Continuum, Adaptive Networking, Service Migration, Artificial Intelligence, Security and Privacy

Abstract

This report discusses the work being carried out in WP2, which is about offering a smooth and an efficient lifecycle management of both the resources composing the platform and the services running on top of it. In this work, an introduction to the CHARITY platform, consisting of the High Level Orchestration and the Low Level Orchestration, is provided, along with some intelligent algorithms for the lifecycle management of XR services such as service scheduling and service relocation. This report also introduces a monitoring framework and a security framework for XR services. Finally, an XR application management framework is also presented.

Document revision history

Version	Date	Description of change	List of contributor(s)
v0.1	05/05/22	Initial version taken from D2.1x with some editorial changes	All
v0.2	13/06/22	Updated sections with most recent works	All
V0.3	17/06/22	Complete review and adding Introduction and Conclusion	All
V0.4	29/06/22	Small editorial changes	All
V0.5	31/08/22	Addressed Reviewers comments	ICT-FI
v0.5.2	31/08/22	Overall correction	ICT-FI
v0.5.3	13/09/22	Overall correction of Sections 5-6	ICT-FI
v0.5.6	20/09/22	Overall correction (yet some comments pending)	ICT-FI
V0.6.3	27/09/22	Addressing pending comments	ICT-FI
V0.6.5	29/09/22	Revisions of Sections 2.3 and 7	ONE, CNR and ICT-FI
V1.0	30/09/22	Final editing and submission	EURES
V1.1	01/02/24	Initial draft of D2.2	ICT-FI
V1.2	09/03/23	Added Sections 6-8	ICT-FI
		Updated Section 1	
	12/03/23	Revised subsection 9.2.1 and added new subsections 9.2.2 and 9.2.3	
		Added content to Section 3	
		Added Section 7, 8	ICT-FI, ONE
V1.3	23/4/24	Added Section 2	CNR
V1.4	24/4/24 to 7/5/24	Editing and reorganizing the whole document	ICT-FI
V1.5	8/5/24	Version sent for review and to PC	ICT-FI
V1.9	22/5/24	Revised as per feedback from reviewers and final version submitted to GA approval	ICT-FI

Acknowledgment

The research conducted by CHARITY receives funding from the European Commission H2020 programme under Grant Agreement No 101016509. The European Commission has no responsibility for the content of this document.

[1] http://creativecommons.org/licenses/by-nc-nd/3.0/deed.en_US

Executive Summary

This deliverable presents key findings essential for implementing a platform designed to facilitate seamless and efficient life-cycle management of computational and networking resources, along with XR (Extended Reality) services hosted within the platform. These findings will inform the development of both the platform itself and the XR services offered by providers. The resulting platform is designed to harness and accommodate multiple network technologies and concepts, including Artificial Intelligence (AI) techniques for orchestration and XR microservices. Additionally, the report delves into the investigation of automation of processes for deployment and operation of advanced XR services, which is a crucial aspect of this initiative

We introduce a cognitive resources-aware orchestration framework designed for deploying XR services in a cloud-native environment. The orchestration framework is split into two parts, namely High Level Orchestration and Low Level Orchestration. It illustrates the management of resources to consistently meet the desired or defined Key Performance Indicators (KPIs) of running XR services within the CHARITY project. A variety of algorithms are introduced with a specific emphasis on XR service orchestration. The CHARITY framework incorporates service placement, scheduling and migration algorithms within its service orchestration by taking into account the network's and cloud's conditions. The operation and management methods of resources, e.g., dynamic bandwidth and cloud resource allocation solutions, that are necessary to meet the strict criteria of XR services are proposed. The utilization of simulation tools and the execution of experiments on resource management in a cloud-native environment are also demonstrated. Additionally, the framework can leverage various AI-based algorithms to perform diverse tasks, e.g., service orchestration, cloud/network resource management. For example, deep reinforcement learning algorithms are leveraged for the synchronized and asynchronized deterministic network flow routing and scheduling. Furthermore, continual deep reinforcement learning can be utilized to address the dynamic nature of multiple access in XR applications.

The protection and privacy of location-based user data in XR services are paramount due to its sensitive nature. Accordingly, a robust security and privacy framework is presented and discussed in this deliverable. Within the framework, several security concerns specific to cloud-native environments, including the essential requirement for security measures at the micro-service level and the necessity for security in orchestration and scheduling procedures, are addressed by presenting the various approaches and mechanisms that are being considered for incorporation within the CHARITY framework. Additionally, it explores the DevSecOps principles as they pertain to enhancing the pertinent security aspects of the upcoming generation of XR applications. Similar to orchestration, automation of CHARITY's security aspects is integral and forms part of the orchestrator. In this vein, this deliverable also explores how the security framework incorporates concepts such as Zero-Trust and SECaaS (Security as a Service) which allows application providers and developers to transfer the responsibility for security services to infrastructure providers.

Both the resource orchestration and security frameworks require an efficient monitoring service to continuously oversee the platform's components and XR services. To address this need, a monitoring and resource usage prediction platform is introduced by systematic gathering, analysis, and utilization of information to continuously see and understand the current state of an application, service, or infrastructure. The real-time monitoring of the environment enables the prompt detection and mitigation of issues, such as cyber-attacks, hence minimizing response time. This platform collects important metrics and utilizes intelligent algorithms, including prediction mechanisms, to extract insights from the data. The insights serve as the foundation for all the intelligent orchestration processes. The successful implementation of closed loop control and intelligent orchestration automation relies heavily on a thorough real-time monitoring strategy and the accuracy of the acquired metrics. The deliverable explains the monitoring framework implemented for CHARITY to reach a dynamic multi-cluster architecture. Also, it details the data gathered, used as a data source for forecasting algorithms and responsible for activating the custom alerting system that leverages real-time and predicted performance information.

The final aspect of this deliverable focuses on the interface between the CHARITY platform and XR providers and developers. An application management framework is introduced, allowing XR developers to define blueprints for their services. These blueprints are then utilized by both the high level and low level orchestration frameworks for deployment and life-cycle management of XR services.

Table of Contents

List of Figures

List of Tables

Abbreviations

5G-PPP	5G Infrastructure Public Private Partnership
AAA	Authentication, Authorization, Accounting
ACO	Ant Colony Optimization
ACNT	Abstraction and Control of Transport Networks
AE	Analytical Engine
AI	Artificial Intelligence
AICO	Analytics Intelligence Control and Orchestration
AIRO	Artificial Intelligence based Resource aware Orchestration
AMF	Application Management Framework
AMP	Application Management Portal
ANN	Artificial Neural Network
AR	Augmented Reality
ARMA	AutoRegressive Moving Average
ARIMA	AutoRegressive Integrated Moving Average
ASET	Adaptive Scheduling of Edge Tasks
ATS	Asynchronous Traffic Shaper
BN	Backhaul Network
CC	Central Cloud
CCROM	CHARITY Compute Resource Orchestrator Module
CD	Continuous Delivery
CI	Continuous Integration
CIS	Center for Internet Security
CNF	Containerized Network Function / Cloud-native Network Function
CNROM	CHARITY Network Resource Orchestrator Module
CP	Constraint Programming
CSP	Cloud Service Provider
DAST	Dynamic Application Security Testing
DCAE	Data Collection, Analytics and Events
DDPG	Deep Deterministic Policy Gradient
DE	Decision Engine
Det-SFCA	Det-SFC Adjustment
Det-SFCD	Det-SFC Deployment
DFS	Depth First Search
DL	Deep Learning
DRL	Deep Reinforcement Learning
ECMP	Equal-Cost MultiPath
EFK	ElasticSearch, Fluentd and Kibana
EM	Edge Minicloud

FG	Forwarding Graph
FL	Federated Learning
GA	Genetic Algorithm
GRU	Gated Recurrent Unit
GUI	Graphical UI
IPFT	Intelligent Proactive Fault Tolerance
ISP	Internet Service Provider
LFA	Link Flooding Attack
LE	Learning and Exploration
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MANO	MANagement and Orchestration
MEAO	Mobile Edge Application Orchestrator
MEC	Multi-access Edge Computing
MEP	Mobile Edge Platform
MILP	Mixed Integer Linear Programming
MINLP	Mixed Integer Non-Linear Programming model
ML	Machine Learning
MRMOGAP	Multi-Objective Generalized Assignment Problem
MS	Monitoring System
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
NFV	Network Function Virtualization
NFVI	NFV Infrastructure
NFVO	NFV Orchestrator
NN	Neural Network
NS	Network Services
NSD	Network Service Descriptor
ONAP	Open Network Automation Platform
OPA	Open Policy Agent
OSM	Open-Source MANO
OSPF	Open Shortest Path First
PNDA	Platform for Network Data Analytics
POLP	Principle Of Least Privilege
QoE	Quality of Experience
QoS	Quality of Service
RFT	Reactive Fault Tolerance
RL	Reinforcement Learning
RM	Request Manager

RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
ROIA	Real-Time Interactive Applications
SAIRMA	Seasonal ARIMA
SAP	Service Allocation Plan
SAST	Static Application Security Testing
SECaaS	Security as a Service
SDN	Software Defined Networking
SFC	Service Function Chain
SMDM	Slice Mobility Decision Maker
TOSCA	Topology and Orchestration Specification for Cloud Applications
TSN	Time Sensitive Networking
UBS	Urgency-Based Shaper
UC	Use Case
UI	User Interface
VIM	Virtualized Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VNFM	VNF Manager
VNF-FG	VNF-Forwarding Graph
VR	Virtual Reality
XR	eXtended Reality
XRSBTR	XR Service Blueprint Template Repository
XRSE	XR Service Enabler
ZSM	Zero-touch network and Service Management

1 Introduction

Having defined the architecture of the CHARITY platform in **D1.3**, we still need to actually define and devise the algorithms and mechanisms that shall run within the components of the architecture. Effectively, in the first phase of the project, the general architecture of the platform was introduced, including all the necessary components of the orchestration framework. Intuitively, these components consist of several algorithms that would bring the necessary intelligence to the envisioned platform. Such algorithms would perform actions such as Virtual Network Function (VNF) placement, cloud and network resource scheduling, service migration, path computation, etc.

This document represents the last version of the deliverable, based on the initial version **D2.1**, that introduces the orchestration framework of the CHARITY platform. It introduces and summarizes all the activities that took place in the ambit of WP2. Following the guidelines of the CHARITY architecture, this deliverable is meant to devise, implement, and experimentally evaluate some of the algorithms and the building blocks of the architecture.

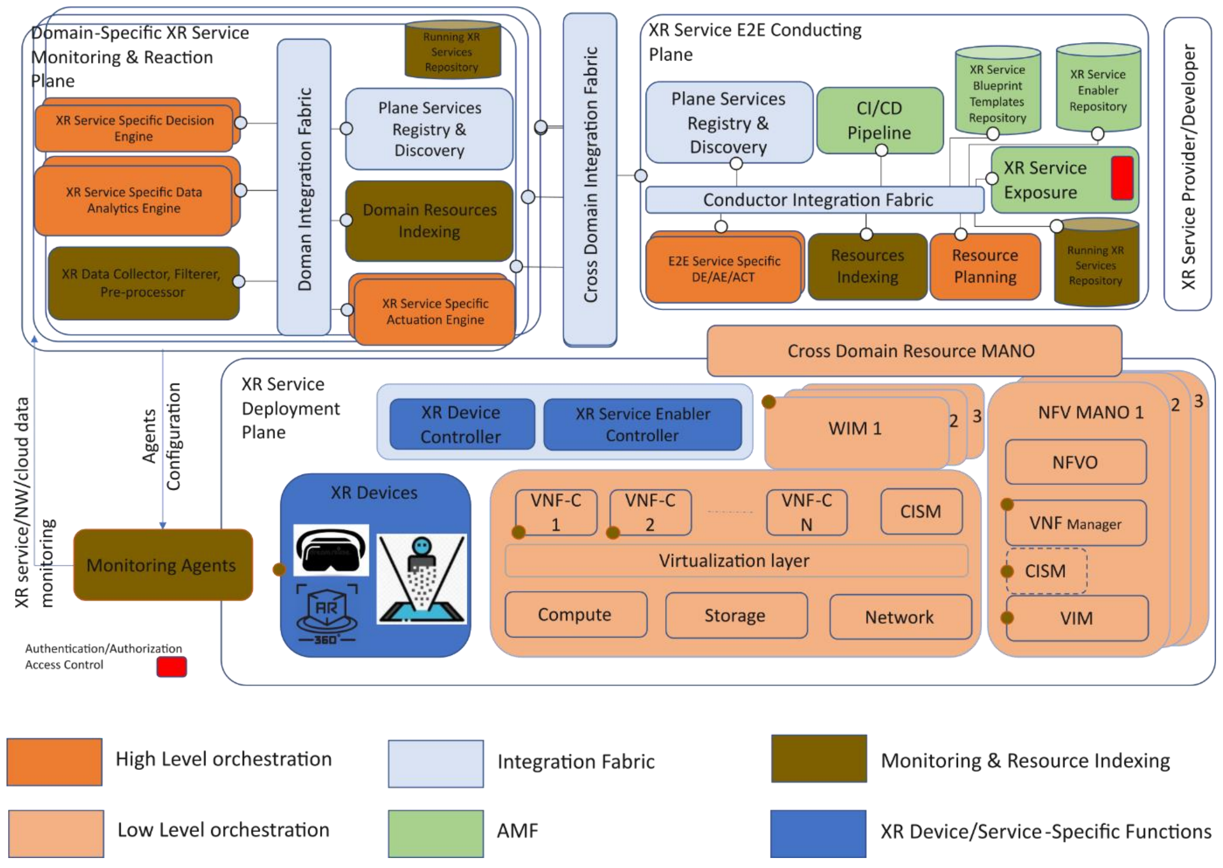

Domain-Specific XR Service Monitoring & Reaction Plane
Running XR Services Repository
XR Service Specific Decision Engine
XR Service Specific Data Analytics Engine
XR Data Collector, Filterer, Pre-processor
Doman Integration Fabric
Plane Services Registry & Discovery
Domain Resources Indexing
XR Service Specific Actuation Engine
Cross Domain Integration Fabric
XR Service E2E Conducting Plane
Plane Services Registry & Discovery
CI/CD Pipeline
XR Service Blueprint Templates Repository
XR Service Enabler Repository
XR Service Exposure
Conductor Integration Fabric
E2E Service Specific DE/AE/ACT
Resources Indexing
Resource Planning
Running XR Services Repository
XR Service Provider/Developer
Cross Domain Resource MANO
XR service/NW/cloud data monitoring
Agents Configuration
Monitoring Agents
Authentication/Authorization Access Control
XR Service Deployment Plane
XR Device Controller
XR Service Enabler Controller
WIM 1
2
3
NFV MANO 1
NFVO
VNF Manager
CISM
VIM
XR Devices
AR
360°
VNF-C 1
VNF-C 2
VNF-C N
CISM
Virtualization layer
Compute
Storage
Network
High Level orchestration
Integration Fabric
Monitoring & Resource Indexing
Low Level orchestration
AMF
XR Device/Service-Specific Functions

*Figure 1: Mapping CHARITY reference architecture with High-Level and Low-Level Orchestration **(D4.2)**.*

This deliverable is structured as follows. Hereunder, explanation will be provided on the evolution path that has been taken by the CHARITY architecture. In Section 2, the high-level orchestration (HLO) framework is introduced, along with the interplay that happens between the XR service and the orchestration framework. Section 2, together with Sections 6-8, also introduce some smart closed loops algorithms that composes the CHARITY orchestration framework. The low-level orchestration (LLO) framework, which intends to provide the means for proper resource instantiation and actual deployment in multi-domain infrastructures, is provided in Section 3, along with the enabling technologies and tools. LLO is responsible for infrastructure provisioning, networking, and application deployments. Monitoring and resource usage prediction define the focus of Section 4. Therein, the monitoring architecture is presented along with the tools that would be used for its implementation. Relevant networking and computation prediction mechanisms are also introduced. These are used for forecasting purposes that can be leveraged by the high-level orchestration framework to take pro-active decisions, as explained in Section 2. In Section 5, the application management framework is presented. It defines the main entry point for XR developers to CHARITY platform. Sections 6 introduces several algorithms to be incorporated into HLO, as discussed in Section 2, with focus on service orchestration. These algorithms also include several service migration algorithms that can be used in the CHARITY framework. One part is about the triggers of service migrations and the other part is about optimizing the actual service migration process while considering the status of the network. Section 7 details networking elements of the CHARITY architecture. It introduces algorithms for dynamic routing and deterministic networking solutions that may be needed to support and meet the stringent requirements of XR services. Section 8 shows some implemented simulation tools and present and discuss some conducted experiments on resource management in a cloud infrastructure. Section 9 is mainly targeted toward security & privacy aspects of the CHARITY framework. It details how XR are to be secured. It further proposes an architecture to ensure security and privacy. Finally, in Section 10, concluding remarks are drawn.

It shall be highlighted that the structure of this document **D2.2** differs from the original structure of **D2.1**, and that is in order to align with the evolved CHARITY architecture. Indeed, as already mentioned in **D4.2**, the CHARITY architecture has experienced a transition from the initial CHARITY architecture to the ongoing development of an integrated orchestration solution. This evolutionary phase in system design aims to capitalize on the existing architecture's strengths while incorporating current component developments. The goal is to create a more streamlined, efficient, and robust system without altering the fundamental principles of the CHARITY architecture. The mapping outlined in Figure 1 categorizes the original blocks into core components responsible for orchestrating XR services and the underlying infrastructure within the context of the CHARITY consolidated solution.

Figure 1 illustrates this mapping, comprising six distinct groups:

- High-Level Orchestration: Intelligent closed loops responsible for resource allocation and component migration decisions (i.e., detailed in Section 2, Sections 6-8).
- Low-Level Orchestration: Building blocks and components for provisioning and orchestrating a multi-domain infrastructure and XR services within each cluster (i.e., detailed in Section 3).
- Monitoring & Resource Indexing: Components related to monitoring and resource indexing for extracting and exposing metrics related to services and underlying infrastructure (i.e., detailed in Section 4).
- AMF (Application Management Framework): Front-end components for XR developers to visually compose services and trigger their provisioning into the CHARITY platform (i.e., detailed in Section 5).
- XR Device/Service-Specific Functions: Components providing XR-specific mechanisms, such as Mesh Merger (i.e., detailed in **D3.2**).
- Integration Fabric: The messaging and integration layer for inter-component communications.

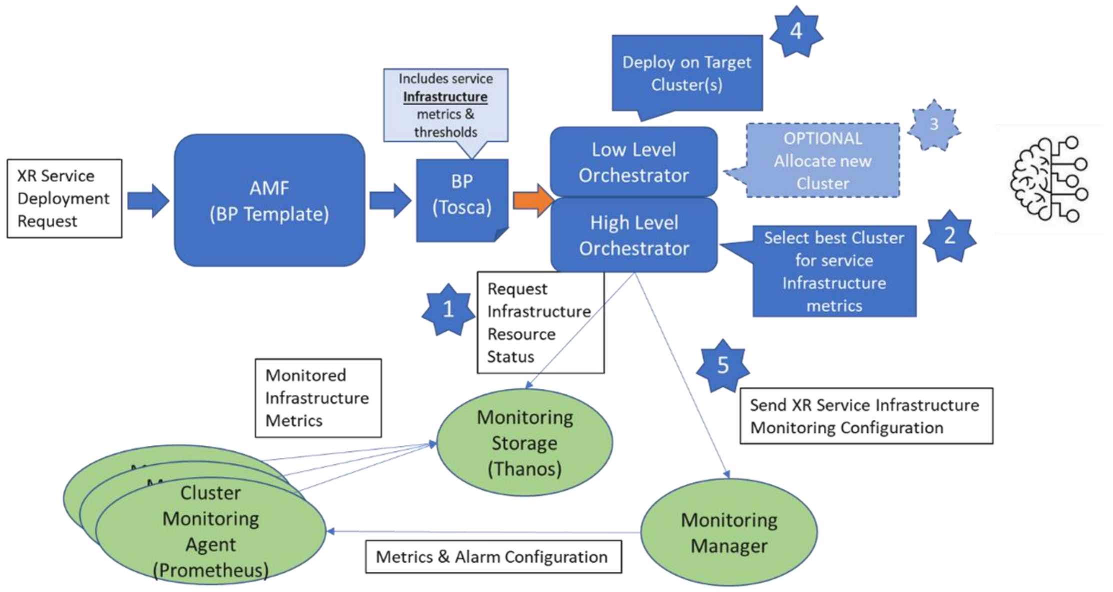

Deploy on Target Cluster(s)
4
Includes service Infrastructure metrics & thresholds
OPTIONAL Allocate new Cluster
3
XR Service Deployment Request
AMF (BP Template)
BP (Tosca)
Low Level Orchestrator
High Level Orchestrator
Select best Cluster for service Infrastructure metrics
2
1
Request Infrastructure Resource Status
5
Monitored Infrastructure Metrics
Monitoring Storage (Thanos)
Send XR Service Infrastructure Monitoring Configuration
Cluster Monitoring Agent (Prometheus)
Metrics & Alarm Configuration
Monitoring Manager

Figure 2: Main workflow and functionalities of the Integrated CHARITY Orchestration solution.

Moving to Figure 2, the main workflow and functionalities of the CHARITY integrated solution during XR application deployment are depicted. Five core features are highlighted:

- Monitoring of infrastructure, cluster, and application metrics.
- Selection of the best location for service deployment.
- Dynamic provisioning of new clusters.
- Actual deployment of services on the selected clusters.
- Reconfiguration of monitoring-related settings.

The AMF serves as the front end for immersive application developers, facilitating the creation and deployment of interactive services. Developers define application modules visually, and the AMF generates a TOSCA representation, triggering a new deployment. The orchestrator then intelligently decides where to deploy services based on real-time metrics and existing or newly created clusters.

Figure 3 and *Figure 4* delve into the steps involved in XR application deployment and re-deployment, respectively. The first orchestration stage analyses XR blueprint definitions, assesses resource infrastructure status, and forms a deployment plan. Two scenarios may arise: either required resources are available, or they need provisioning. The Low-level orchestrator translates decisions into deployments, including provisioning new clusters and forming connections between them. In re-deployment scenarios, AI-driven closed loops and predictive analysis determine when migration should occur, predicting metric threshold exceedance and triggering alerts for the orchestrator.

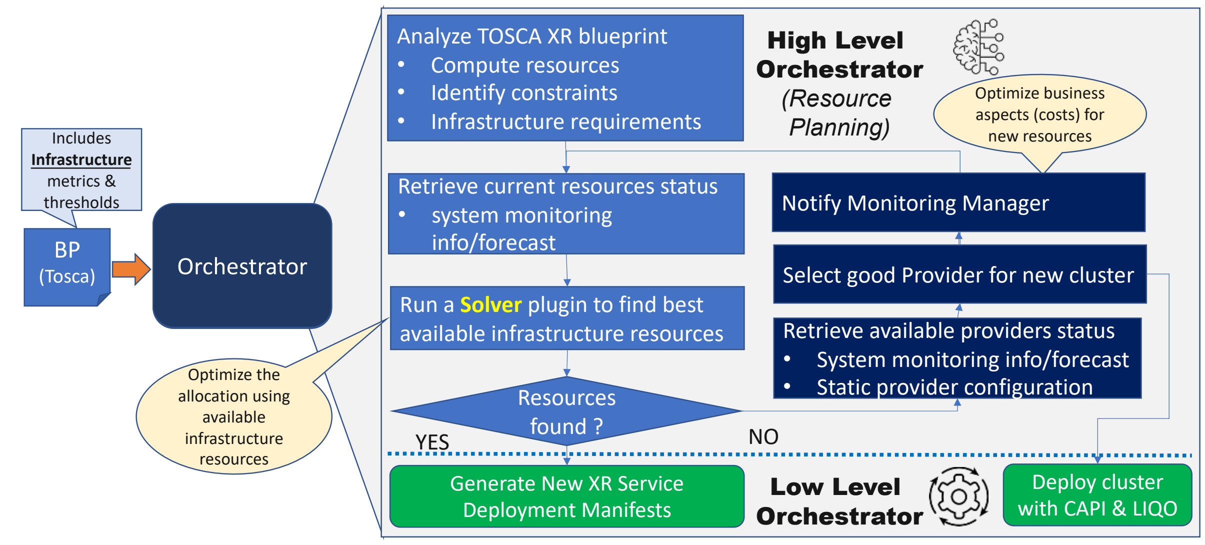

Includes Infrastructure metrics & thresholds
BP (Tosca)
Orchestrator
Optimize the allocation using available infrastructure resources
Analyze TOSCA XR blueprint
• Compute resources
• Identify constraints
• Infrastructure requirements
High Level Orchestrator (Resource Planning)
Optimize business aspects (costs) for new resources
Retrieve current resources status
• system monitoring info/forecast
Notify Monitoring Manager
Select good Provider for new cluster
Run a Solver plugin to find best available infrastructure resources
Retrieve available providers status
• System monitoring info/forecast
• Static provider configuration
Resources found ?
YES
NO
Generate New XR Service Deployment Manifests
Low Level Orchestrator
Deploy cluster with CAPI & LIQO

Figure 3: New XR service deployment request steps.

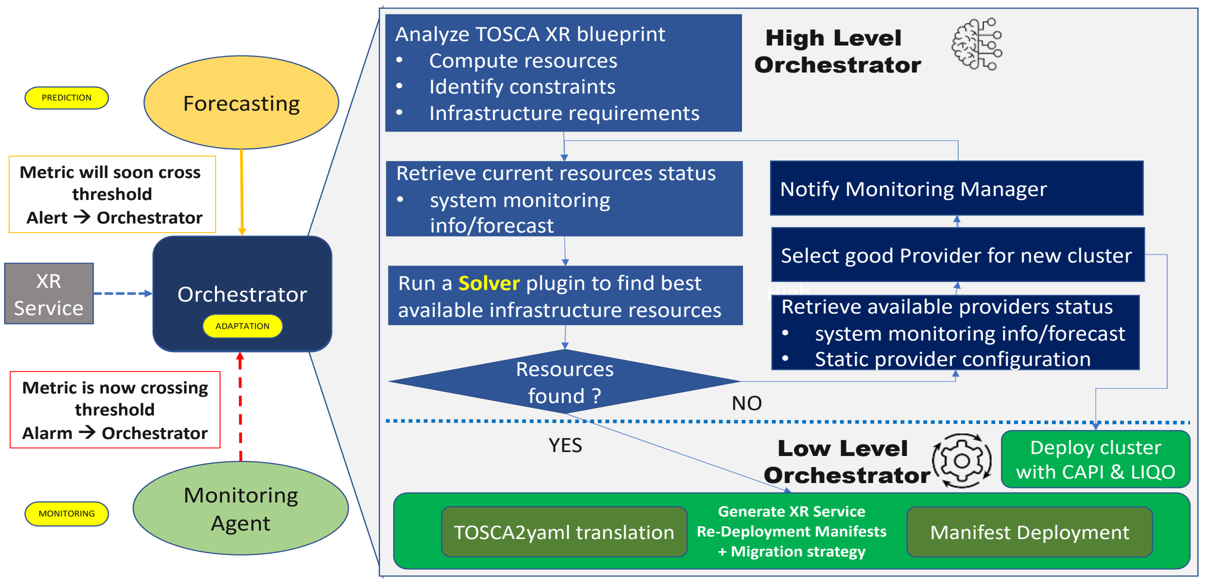

PREDICTION
Forecasting
Metric will soon cross threshold
Alert → Orchestrator
XR Service
Orchestrator
ADAPTATION
Metric is now crossing threshold
Alarm → Orchestrator
MONITORING
Monitoring Agent
Analyze TOSCA XR blueprint
• Compute resources
• Identify constraints
• Infrastructure requirements
High Level Orchestrator
Retrieve current resources status
• system monitoring info/forecast
Notify Monitoring Manager
Select good Provider for new cluster
Run a Solver plugin to find best available infrastructure resources
Retrieve available providers status
• system monitoring info/forecast
• Static provider configuration
Resources found ?
NO
YES
Low Level Orchestrator
Deploy cluster with CAPI & LIQO
TOSCA2yaml translation
Generate XR Service Re-Deployment Manifests + Migration strategy
Manifest Deployment

Figure 4: New XR service re-deployment request steps.

2 High Level Orchestration

This section is about the design and implementation of the High-Level Orchestrator module (HLO) which performs the high-level phase of resource allocation in the CHARITY platform. The HLO is a core component of the platform, interacting directly or indirectly with basically all the platform components. Hence, this section also provides a first peek into the overall orchestration solution as developed by the project. As we dwell in the design and features of the HLO, we will often be summarizing the role of various platform services, further detailed in the following sections of this deliverable, to clarify their respective interactions with the HLO.

The HLO is the component that takes decisions on the most suitable deployment topology for a specific XR application deployment request, considering

- the application requirements, in terms of the expected quality of services and quality of experience,
- and the resources available in the CHARITY deployment targets.

Besides being it a universal requirement for Continuum Computing platforms, dynamic resource management was included since the very first design of the platform, because CHARITY specifically manages XR applications with strict QoS. Dynamic orchestration enables the provisioning of a near-real-time, seamless experience to the application users. Dynamic orchestration also requires additional, continuously updated information about

- application component behaviour (the application components may change their resource usage for any business or operative reason, including the action and real-world locations of the collectivity of its users),
- deployed resource behaviour (as resource churn, overloading and interference can affect all aspects of orchestration: computation, networking, and storage), and
- availability and features of resources that are potentially needed for subsequent deployment actions.

We only briefly recall here that the archetypical methodology allowing dynamic management is that of the closed control loop, where the HLO loop beside the resources includes the LLO, the Monitoring, the Forecast, and the AMF user interface. Such a control loop is depicted in *Figure 3* (Service Deployment) and *Figure 4* (Service Adaptation), that are discussed in the following sections.

Over the course of the CHARITY project, and as explained earlier, the CHARITY architecture presented in **D1.3** and **D2.1** has evolved into the combination of a High-Level Orchestrator and a Low-Level Orchestrator (LLO, see Section 3). The combination of these two components allows to better separate the aspect of abstract resource allocation and management from the aspect of performing the concrete activities that deploy applications and steer the resource allocation on the real-world platform.

- The HLO is triggered by AMF for deployment requests arriving either from the XR developer or the external AMF REST API. This HLO interface is the main entry point for all orchestration decisions, and the HLO ultimately relies then on the LLO to enact high-level decisions on the target domains (e.g., new cluster allocation, deployment of XR application manifests).
- The HLO focuses on the problem of combined optimization of computational, networking and storage resources. As previously reported in **D2.1**, the whole allocation problem is cast as a Multi-Resource Multi-Objective Generalized Assignment Problem (MRMOGAP), which can be tackled with a variety of techniques sporting different trade-offs among algorithmic complexity, execution time, and solution quality.
- The HLO can support different techniques to solve the MRMOGAP problem by leveraging (1) the high-level application description based on TOSCA Blueprints, and (2) resource information that is parsed and transformed into the data structures needed by the specific method. The

LLO, by comparison, focuses more on interaction with the Cloud resources, connecting resources across multiple clusters and datacenters, and reacting directly to local effects.

- The layered HLO/LLO structure allows the HLO to remain more flexible in its implementation, leaving the burden of interacting with Cloud interfaces (e.g. ClusterAPI) to the LLO module (see Section 3).
- A set of alerts and warnings is propagated from the platform and the apps to the HLO, allowing the HLO to gather information about QoS violation, as well as forecasts of violations concerning a multiplicity of events, that range from flat resource unavailability to runtime behaviour variations. The set of events also optionally includes customized, application-defined metrics and application-mandated deployment reactions. The HLO employs the Monitoring Manager as the primary interface to all monitored metrics. The design and features of the monitoring system are described later, in Section 4.
- AI-informed decisions for self-* management are achieved by leveraging the CHARITY forecast service, employing various RNN and DNN techniques to foresee QoS incidents, as well as by developing AI-based plugins for the HLO. The AI hints at future behaviour are essential and have their own semantics, but in order to streamline the HLO design, the forecasting Manager is invoked through the Monitoring Manager, reusing the same interface and compatible abstractions for all kind of platform-initiated warning and alerts. This removes the need to develop within the HLO separate control loops for different types of events and warnings, allowing to employ common general abstractions and orchestration/adaptation techniques for the high level of orchestration. The set of AI techniques that are employed to forecast QoS violations, resource issues and application issues are further described in Sections 6-8.

2.1 HLO – Algorithmic View

So far, we summarized the overall interactions of the HLO with other components of the CHARITY platform. We now describe the HLO from the algorithmic viewpoint, detailing the different paths the HLO can follow in performing the orchestration process to fulfil all the CHARITY-mandated features of adaptivity, efficiency and intelligent resource management. The input sources for the CHARITY orchestrator include:

- The TOSCA Blueprint (BP) from the AMF describing the application structure according to the TOSCA format (Section 5).
- All types of resource information from the Monitoring System (i.e., resource availability and capabilities over the known virtual clusters – see Section 4).
- Alerts and warnings from the Monitoring system (i.e., these are asynchronous events that are relayed to the HLO by remapping the information onto the REST API – see Section 4).

The main Output expected from the CHARITY HLO consists of

- Deployment plans for the LLO in the form of augmented application BluePrints, where the deployment domains for each application module or subsets are specified via specific TOSCA fields.

Additionally, the HLO process can result in the allocation of new virtual clusters from known datacenters as a side effect, whenever the need for more resources is detected during the orchestration algorithm. We show in *Figure 3* (Service Deployment) the Initial Deployment phase of the overall orchestration process involving both the HLO and the LLO, as well as some of the interactions with other platform components.

The overall architecture of the CHARITY orchestration support, planned in deliverable **D2.1,** was targeting the AIRO (AI Resource-aware Orchestrator) as a composition of orchestration modules devoted to different orchestration aspects (computation resources and network resource) whose cooperation results in the ability to orchestrate services with the specific QoS required by the XR/AR reference applications of CHARITY. The initial design of these orchestration modules would also account for the option of having them decomposed hierarchically in order to better manage large and heterogeneous platforms. This initial architecture has evolved into a hierarchical decomposition of the orchestrator in two layers,

- a High-Level Orchestrator, which is actually a common module dealing with the orchestration problem as a whole, including all requirements and constraints related to any resource and application component but performing only abstract management. A more abstract viewpoint to the orchestration is chosen for the HLO, which reduces the dependencies from the low-level details of software deployment, allowing to focus on essential parameters of the deployment and the correlation between different resources. The higher abstraction level (e.g., ignoring low level details, grouping all resources in a cluster for the sake of problem-solving) makes amenable a mathematical formulation of the problem that can be either directly optimized, providing “best” solutions to each orchestration request, or approximated using different techniques to achieve a desired trade-off between optimality of the orchestration action and the complexity and cost of finding out the action.
- a Low-Level Orchestrator, performing concrete resource management, which is the topic of Section 3. To deploy applications, it directly interacts with the cluster resources via ClusterAPI and sets up the networking via LIQO, thus preserving the rough subdivision into specialized orchestration modules for different aspects. The LLO can also perform its own dedicated optimization, as documented in Section 3.

The abstract management performed by the HLO, focusing on high-level optimization, also allows for greater flexibility and reduced code complexity. One way the flexibility is used is in defining a plug-in interface for exploiting different orchestration policies as interchangeable modules, allowing for easier experimenting activities, and debugging activities. This is described in section 2.2, as well as in Sections 6-8, each addressing an optimization objective (i.e., a specific closed loop with a specific algorithm to achieve a specific optimization).

The improved flexibility is also exploited to more easily accomplish the second key task that AIRO was expected to tackle, that is, dealing with dynamic orchestration. Reacting to all kind of changes happening in the resources, in the application, and in the whole Continuum platform is based on sensing different events and alerts via the Monitoring system, as previously mentioned. This flux of events plus the ordinary submission of new applications from the users/XR developers create a stream of Orchestration requests and Adaptation (i.e., re-orchestration) requests.

We deal with this stream of requests by applying a control loop structure that receives two types of inputs (i.e. events alerts and submission requests) and has a main way of reacting to that, and a second loop to deal with the special case that additional resources need to be located to complete the orchestration. This leads to two main ways of working of the HLO, and two execution variants for each use case.

The two use cases, taken together, implement the control loop dynamically steering application orchestration in CHARITY.

- **Deployment** (or Initial Orchestration) – The Initial Orchestration is triggered by the user when application deployment is first required, with the user itself providing via the AMF all the information about the structure and requirements of the application and all its services, encoded in the application blueprint. The initial orchestration implements the beginning part of the control loop managing each application, when resource selection takes place according to application requirements, to resource availability and constraints, developing a high-level deployment plan for a newly launched app. Initial orchestration is depicted in *Figure 3* (Service Deployment).
- **Adaptation** (or re-orchestration) – The adaptation use case implements the need to revise the initial deployment as a reaction to a change in the environment (e.g., the app, the platform

resources) that is disrupting, or may soon disrupt, the stipulated QoS for some part of the application. The adaptation process is promoted by the monitoring system providing the HLO with a flow of detected metrics from the application, its components, and the resources, as well as foreseen values for some of these metrics predicting future QoS issues. We thus see that the Monitoring System closes the control loop toward the HLO, and the Adaptation use case in the HLO is nothing else than the implementation of the control steering for any application that was already deployed. The Adaptation part of the Orchestration process is depicted in *Figure 4* (Service Adaptation).

The Two variants of the execution are common to both use cases, and deal with the two possible meta-outcomes of attempting a high-level deployment plan

- **Feasibility** – enough resources are already available to generate a high-level plan. This default flow of actions leads directly to providing the LLO with the high-level plan for further action.
- **Unfeasibility** – the optimization phase of the high-level plan detects that this application deployment requires unavailable (amounts of) resources to succeed. In this execution variant, an inner loop of the orchestration process accounts for the need to locate additional resources, before attempting to satisfy the deploy request again. The loop seeking for additional resources can possibly retry several times the two meta-steps of high-level resource allocation (asking for new virtual clusters to be deployed) and attempting a high-level allocation.

Both execution variants (i.e., the straightforward and the inner resource location loop) are obviously present in both the Initial Deployment and in the Adaptation Orchestration cases.

2.2 The Solver Plugins

As already outlined in **D2.1 and further expanded in Sections 6-8**, there are several different approaches to solving the optimization problem underlying the (re)deployment of complex applications on Continuum platforms. These approaches can be generally organized in three categories:

- Algorithms for the service orchestration **(Section 6)**
- Algorithms for the network resource orchestration **(Section 7)**
- Algorithms for the cloud resource orchestration **(Section 8)**

Hereunder, we focus on a *solver plugin* interface that the CHARITY's HLO has been extended with. It can be configured to use different algorithms to solve the placement problem of the XR Application resources. This choice does not introduce changes either in the external interfaces of the HLO, as described in **D4.5 section 2**, or in the format of the data provided and returned by the HLO. Application requirements are described inside the TOSCA topology model.

The Solver Plug-in API allows asking for an optimal solution for the placement of application components into Kubernetes Pods on the CHARITY runtime resources available, providing a REST API with a POST method. All solver plugins are assumed to be stateless, so the HLO embeds all the needed information in the plugin call, namely

- the Tosca model for the XR Application coming from the AMF layer, where parameters of the application Blueprint have been instantiated by the AMF
- in case of a redeployment due to an application adaptation, the already deployed TOSCA blueprint is also included, as it has been augmented by the deployment process of the LLO to hold information about the current deployment status of the application, e.g., cluster, computing resources, IP addresses and existing virtual networks. When devising an update, the current deployment details
    - allow to avoid unnecessary application pod redeployment,

    - provide the essential information to allow newly deployed pods to join the existing application,
    - can help reduce the size of the optimization problem

- all static and dynamic information about the available datacenters and the clusters that are available for deployment, including their features and properties like e.g., the number and characteristics of their nodes (cores, memory, storage)

Based on this input, the solver returns a response which is again a TOSCA application description, further augmented with extra fields holding information needed by the LLO to enact deployment targets for compute and storage, as well as deployment of inter-cluster network resources by configuring peer connections between clusters. When not enough resources are available to allow the deployment, the solver plugin must identify the need for new clusters and provide the information needed for setting up new virtual clusters, e.g., type of resources and size of the cluster.

The basic need for a Solver plugin is to allow experimenting and mixing different optimization techniques on the CHARITY platform. Several optimization techniques were identified in D2.1. Among those the project developed two different "solvers":

– A complex scenario and large topology solver using an algorithm with significant complexity, capable of determining the "global best" solution, but requiring significant computing resources, hence sporting a non-negligible computation time to generate a deployment solution

– A simpler, fast algorithm using heuristics suitable for small scenarios and topologies. The simple solver can find a "local best" solution (a "good enough" one with no guarantees of optimality) with linear complexity and using minimal computing resources

We shall note that these two solutions can be seen as the two extremes of a range of techniques sporting different complexity and features. The MILP based complex scenario solver aims at finding optimal deployment for each request, at the cost of algorithmic "heavy lifting" and limited scalability. Less accurate, heuristic-based solutions or NN-based ones sit in the middle, as they can find comparatively good solutions in less time. While approximated methods do not guarantee the solution to be optimal, they can be preferred as they are typically more scalable and allow to save on time and resources spent to find mathematical optima. Finally, the greedy approach of the Simpler solver with its basic heuristics requires the least amount of resources and time to generate a feasible solution and let the deployment progress.

2.2.1 Simple Scenario Solver

The solver plug-in for the simple scenario (SSS) was already outlined in D4.5. This plugin is based on a simple heuristic algorithm that aims at quickly finding a solution using existing, available resources with no claim of optimality in the resource usage, sporting on the other hand low code complexity and requiring a limited amount of computation and memory resources to produce a deploy solution. The simple scenario solver is ideal when resources are not tight or costly and can simplify orchestration debugging, as it produces easily verifiable and usually highly repeatable solutions.

As all solver plugins, the SSS parses the input TOSCA model, extracting the relevant metadata of the CHARITYNode elements to be deployed, as well as the current and forecasted resource status and availability for the known datacenters and clusters.

A linear scan on all CHARITYNode and domains (datacenters) filters out the domains with not enough resources for each CHARITYNode, considering all resource aspects (e.g., CPU, memory, storage, GPUs). Compliant domains are then scored based (i) on network proximity to the connected geolocated ExternalSystem that the application need to interact with (priority is given to EDGE nodes and CLUSTER nodes with links to EDGE nodes) as well as (ii) on the availability of computing resources. Finally, the feasible domains are overall ranked.

Once the best domain (i.e., the datacenter with the highest score) has been selected for the

CHARITYNode, the existing clusters are examined to check their degree of "occupancy" (current and forecasted).

- If a "good" cluster is found – that is, the best one in the selected datacenter – the Node will be allocated in that cluster. To implement the choice, the CHARITYNode attributes inside the TOSCA model will be updated with the selected datacenter and cluster.
- If on the other hand no good cluster is found, this means that we are in the case when no free clusters are available, but the datacenter still has enough free resources. The CHARITYNode must be assigned to a new cluster, one that will be allocated prior to the deployment. We are now in the case variant where action is needed to gather more resources before the APP deployment can progress in the HLO. The name and size attributes of the new cluster to be created are then added to the list of clusters to be created returned by the solver to HLO, together with the updated TOSCA model.

To keep the placement algorithm complexity as low as possible, this simple solver plugin implements a redeployment request simply as a new independent request without considering the current state. While this guarantees that a new allocation can be found with minimal search overhead, this design choice does not attempt at minimizing the potential components migration costs.

2.2.2 Complex Scenario Optimal Solver (CSOS)

As previously reported in D4.5, the solver module targeting the complex scenario develops from the work done during the project in studying the Multi-Resource Multi-Objective Generalized Assignment Problem (MRMOGAP) that results from the matching of application structure, constraints, and parameters specified in the TOSCA blueprint, with the available computational resources belonging to the CHARITY federation. For the sake of completeness and readability this section and its subsections rework and integrate material from previous version of the deliverable with new results and information that was partly covered in deliverables from other WPs (deliverables D4.2 section 5.2.1, and D4.5 section 2.6). We instead leave out implementation details like the detailed REST API descriptions and the interaction diagrams that were presented there.

The MRMOGAP problem, besides being algorithmically hard, is also made more complex by the dynamicity of the Continuum environment, as platform resources status, application and user needs are dynamically changing over time. The characteristics and status of the resources, as well as those of the already deployed application nodes, are continuously monitored in CHARITY. Data gathered via the monitoring support contributes to the definition of each instance of the MRMOGAP problem (turned into coefficients for the optimization problem) and leads to dynamically changing solutions also in the "simple" case of repeating the application deployment for a given application.

Additionally, the Complex scenario Solver (CSOS) addresses the case variant when an application needs either total or partial redeployment, as a reaction to one of the different alerts that the platform can generate to signal verified QoS constraint violation, foreseen violations from the forecast module, or explicit scale-up/down of resources linked to application parameter changes.

The main steps of the CSOS plugin, which are partly common with those of the SSS plugin, are the following ones.

1. The plugin parses the TOSCA blueprint of the application to be (re)deployed, gathering all information about the desired structure and all related aspects of QoS (e.g., compute, networking, storage)
2. The plugin parses, if it is present, the TOSCA blueprint of the currently deployed application (*previous blueprint*). If a previous blueprint TOSCA is present in input, we are in the Redeployment process, as depicted in *Figure 4* (Service Adaptation). The previous blueprint is needed as solver plugins are stateless, hence we need to gather information about the previous deployment. The current blueprint is exploited to understand
    - what modules of the application are critical, or are soon foreseen to be, in terms of

QoS, due to changing application load or resource availability. These parts of the application will need reoptimization.

    - what modules of the application are not critical and can stay deployed as they are, and also how they are currently deployed in the first place.

3. The information about previous deployment non only provides the option to avoid redeploying the whole application because of a local change, but it also helps reducing the size of the optimization problem for the redeployment case, thus reducing the service time and workload of the CSOS module.
4. Information about the available resources in the platform is gathered and parsed like it is in the SSS plugin.
5. MRMOGAP problem generation. The plugin translates all the gathered information into a mathematical formulation of the MRMOGAP problem. In the CSOS plugin developed for CHARITY, we chose to exploit a Mixed-Integer Linear Programming (MILP) optimization technique, so the deployment solution is obtained by the solution of a specially crafted MILP problem. The design of the MILP problem generation and details on the approach to solving it are reported in subsection 2.2.2.1.
6. The plugin solves the MILP problem (see subsection 2.2.2.1).
7. The output of the optimization can lead to two possible cases (the two use-case variants).
    - The optimization problem is solvable, so a suitable allocation is found, and it is optimal with respect to the optimization criteria encoded in the MILP formulation:
        a. the MILP problem solution is translated into allocation parameters that are filled in the TOSCA blueprint by the main orchestrator module, specifying the domains and clusters for each CHARITYNode element like the SSS plugin does.
        b. The augmented blueprint is then returned to the main HLO module for the deployment process to proceed.
        c. As final step of the HLO to solver plugin interaction, the augmented blueprint is provided to the LLO, which will specify further details about the actual deployment and pass them back as TOSCA output variables to the main HLO module. Besides that, the augmented version of the blueprint is also cached within the HLO. It will be provided again to the CSOS plugin as app adaptation is triggered, to recover the current app deployment state and support dynamic reallocation and deployment.
    - The optimization problem is unsolvable. Then, no suitable allocation can be found with the available resources. In this case the additional resources needed to allow a successful deployment are inferred from the modelled MILP problem. The amount and type of additional resources required is returned to the main HLO module, which proceeds to recruit new resources e.g., by allocating new virtual clusters from an available datacenter or from a public cloud provider. This case variant is the same as the corresponding one of the SSS plugin, except for how the additional list of required resources is generated.

Finally, the output of the CSOS plug-in back to the main HLO is either

- a deployment allocation, whose details and topology are returned to the main orchestrator so that the deployment can proceed by interacting with the LLO, or
- the specification of a suitable set of additional resources to look for, since there is no feasible deploy plan with current resources.

2.2.2.1 The MRMOGAP problem as a Mixed Integer Linear Programming problem

The CSOS solver plugin approaches the MRMOGAP problem by encoding it as an instance of a Mixed Integer Linear Programming (MILP) problem. The optimization problem of deploying a set of components of a given microservice-based application onto the available resources of a set of datacenter / edge clusters is thus encoded as a MILP instance.

Linear Programming optimization (LP) tackles the problem of optimizing a linear function of a vector $x \in \mathbb{R}^n$ while obeying a set of linear constraints of the form $a \cdot x \leq b$.

MILP optimization is a variant of Linear Programming that requires a proper subset of the elements of the vector x to be restricted to integer values. While LP has exponential worst-case complexity but on average it requires a polynomial number of steps, MILP optimization is an NP-hard problem and can be efficiently solved only if we can guarantee specific preconditions on the matrix of coefficients, if we can apply search heuristics and search pruning algorithms. On the other hand, MILP provides the flexibility needed to encode a plethora of real-word optimization problems, and MRMOGAP problems specifically, so a many decades long research and development effort form the Operation Research community has been poured into crafting efficient techniques as well as proprietary and open-source tools to generate and solve MILP problems.

We translate the MRMOGAP problem into MILP by generating equations, (auxiliary) variables, and coefficients that encode all the constraints in the MRMOGAP, as well as defining the cost function based on the cost objectives of the orchestrator. These can be e.g., performance metrics, power metrics, economic costs, and in the general case a linear combination of those aspect-specific metrics that result in multicriteria optimization.

As reported in previous deliverables, the implementation of the CSOS plug-in relies on the Python-MIP library[2] version 5, for generating the actual MILP instance and interact with the solver library. Main and auxiliary variables of the instance are generated to encode the different allocation options, problem coefficients and equations for constraints are directly derived from the QoS constraints in the blueprint as well as built to ensure consistency of the MILP model with the original problem [3].

The MILP instance is then solved with the COIN-OR Branch-&-Cut CBC solver[4], a standard, open-source C-based optimization library. This choice allows CHARITY's HLO and its orchestration process to exploit the considerable expertise and the years of theoretical and technological contributions provided by the field of Operating Research and accumulated the tool. Countless optimization algorithms and heuristics are encoded in a flexible and robust software artifact that is developed and maintained over the years. The CSOS solver plug-in executes a Branch-&-Cut (BC) algorithm that provides the exact optimal solution in a finite time. The dev-supported integration of Python-MIP and the COIN-OR solver ensure that the solver plug-in exploiting them can be easily packaged and distributed as a docker container with standard REST interfaces with TOSCA and JSON encoded parameters, ensuring architectural modularity as well as easing the integration with the main HLO.

Going back to the topic of how the problem is modelled, for the sake of clarity we must underline that in the following we will need to distinguish between *graph edges,* that are mathematical abstractions, and *Edge resources*, that represent actual Edge computing systems (either to be located or known, available ones).

In our model an application *A* is represented by an undirected graph $G_A =< C, E >$

- *C* represents a set of *N* vertexes and *E* represents the set of *M* edges connecting the vertexes.

[2] https://www.python-mip.com

[3] See "SMARTORC: Smart Orchestration of Resources in the Compute Continuum", E. Carlini et al, Frontiers in High Performance Computing, Volume 1 – 2023. DOI: 10.3389/fhpcp.2023.1164915

[4] https://github.com/coin-or/Cbc

Each vertex $c_i \in C$ with $i \in \{1, \dots, N\}$ embodies a single component of the application, according to the microservice paradigm. Different applications can be deployed on distinct clusters, depending on each application's QoS requirements.

- E is a set of graph edges. Each edge $e_{i,j} \in E$ represents an undirected communication path connecting application components v_i and v_j, with $i, j \in \{1, \dots, N\} : i \neq j$.

Every vertex in C (an application component) and every edge in E (a communication link) can be labelled with a set of QoS attributes or requirements $Q = \{q_1, \dots, q_S\}$ that are associated with it. To express the semantics of the original MRMOGAP problem as a MILP, it is necessary to classify all attributes (requirements, constraints) into a specific taxonomy.

The classification outlines the different semantic aspects of the application requirements: whether they affect linearly the solution quality, or represent fixed constraints, whether they apply to a single cluster or machine, as opposed to the relationship among a set of distinct clusters/machines, and whether they relate with consuming shareable or non-shareable resources.

From the viewpoint of the MILP problem encoding, attributes in Q can be classified in two categories:

- *ascending/descending QoS attributes*, where higher (resp. lower) values of the attribute are better for the application QoS (e.g., provide higher performance to one of the application modules). Examples of ascending/descending QoS attributes are the minimum number of CPU cores, the quantity of available memory or disk space, or the current utilization of a resource.
- *equality QoS attributes*, where only equality or inequality constraints are meaningful from the viewpoint of the application deployment. Examples of equality QoS attributes are the presence/absence of a particular software feature, e.g., an Operating System (OS) or a software license, or the availability of a hardware feature, e.g., a Graphic Processing Unit (GPU).

QoS requirements can be also classified into

- *intra* requirements, that are associated with a single vertex or application component and are modelled as constraints over the resources of a single data center.
- *inter* QoS attributes model constraints over a set of resources belonging to different vertexes (also including constraints over actual network links). For instance, maximum latency or minimum bandwidth are examples of *inter* QoS attributes as they are associated with an edge of the graph, that is with two different graph vertexes.

Requirements may also imply the allocation of resources that cannot be shared.

- A *numerical* requirement *is* a metric requirement which refers to a resource that cannot be shared between different components or communication channels. The required value by the application reduces the available amount of the resource, as is the case e.g., for the number of (exclusive) cores, the amount of disk space or the consumed bandwidth of a channel over a link.
- A *non-numerical* QoS requirement represents a resource that can be shared under commonplace assumptions without being consumed. Examples of a QoS requirement that is non-numerical (or can be assumed to be, in ordinary working conditions) is the maximum latency induced by a communication channel.

As part of our modelization effort, we also need to model the available portion of the Compute Continuum. This is done likewise to the application model, by representing the Compute continuum as an undirected graph $G_{Cont} =< D, L >$, where D represents a set of vertexes and L represents the set of graph edges that connect those vertexes.

- Each vertex $d_i \in D$, with $i \in \{1, \dots, P\}$ represents a single cluster of the continuum. The resources associated with each data center are the sum of all the resources available among the devices, servers and other commodities that the datacenter supervises.

- Each edge $l_{i,j} \in L$, (with V being the size of L) represents an undirected communication link between two different data centers labelled as d_i and d_j, with $i, j \in P : i \neq j$. As it is for applications, each data center D_d and each communication link L_v is associated with a limited amount of *intra-datacenter (intra)* and *inter-datacenter (inter)* resources $R = \{r_1, \dots , r_T\}$, i.e. the maximum number of CPU nodes, memory size, or disk space, or the minimum latency and the maximum bandwidth over links.

2.2.2.2 Generation of the MILP instance

The MILP problem instance corresponding to a specific orchestration problem requires us to specify several set of mathematical constraints, each set encoding a specific semantics from the original MRMOGAP, and in general add both ordinary variables (representing the actual parameters of the deployment) and auxiliary ones (a standard OR tool to encode particular constraints and non-linear behaviours within a standard MILP formulation). We exemplify the process by focusing on a simple deploy instance of limited complexity.

A first set of constraints in the MILP model comes from the fact that the application components in execution at any point in time cannot exceed the currently available capacity of such resources, indicated as $C_{r_j,j}$, where $r_i \in R$ and $j \in \{1, \dots , M\}$ for *intra* resources, while $j \in \{1, \dots , V\}$ for *inter* resources.

- For every type of QoS requirement $q_i \in Q$, there is a corresponding resource $r_i \in R$ against which the requirement has to be satisfied. When attributes are used in specifying the application, they define constraint values to be respected by the corresponding resource allocated within the Compute continuum. For instance, the QoS requirement of a component about the number of cores needed to reach an expected performance level must be satisfied by the number of cores of the datacenter d_i on which it will be deployed.
- Requirements over edges are analogous, e.g., the maximum latency stipulated by a communication channel between two different components must be satisfied by the minimum latency associated to the communication link $l_i \in G_{Cont}$ connecting the two datacenters d_i and d_j where the two components are deployed.

$$
\begin{aligned}
\max_{\mathbf{x},\mathbf{y}} \quad & F_u(\mathbf{x},\mathbf{y}) \\
\text{s.t.} \quad & (1) \sum_{d=1}^{P} x_{i,d} = 1, \forall i \\
& (2)\, (1 - f_{inter}(q_j)) * f_{num}(q_j) * \left(\sum_{i=1}^{N} (x_{i,d} * q_j) - C_{r_j,d} \right) \leq 0, \forall d, j \\
& (3)\, (1 - f_{inter}(q_j)) * (1 - f_{eq}(q_j)) * \left((x_{i,n} * q_j) - C_{r_j,d} \right) \leq 0, \forall i, d, j \\
& (4)\, (1 - f_{inter}(q_j)) * f_{eq}(q_j) * \left((x_{i,n} * q_j) - C_{r_j,d} \right) = 0, \forall i, d, j \\
& (5)\, x_{i_1,d_1} + x_{i_2,d_2} - y_{i_1,d_1,i_2,d_2} - 1 \leq 0, \forall i_1, i_2 : i_1 \neq i_2, \forall d_1, d_2 : d_1 \neq d_2 \\
& (6)\, y_{i_1,d_1,i_2,d_2} \leq x_{i_1,d_1}, \forall i_1, i_2 : i_1 \neq i_2, \forall d_1, d_2 : d_1 \neq d_2 \\
& (7)\, y_{i_1,d_1,i_2,d_2} \leq x_{i_2,d_2}, \forall i_1, i_2 : i_1 \neq i_2, \forall d_1, d_2 : d_1 \neq d_2 \\
& (8)\, f_{inter}(q_j) * f_{num}(q_j) * \left(\sum_{i_1=1}^{N} \left(\sum_{i_2=1}^{N} (y_{i_1,d_1,i_2,d_2} * q_j) - C_{r_j,l} \right) \right) \leq 0, \forall d_1, d_2 : d_1 \neq d_2, \forall j, l \\
& (9)\, f_{inter}(q_j) * (1 - f_{eq}(q_j)) * \left((y_{i_1,d_1,i_2,d_2} * q_j) - C_{r_j,l} \right) \leq 0, \forall d_1, d_2 : d_1 \neq d_2, \forall i_1, i_2 : i_1 \neq i_2, \forall j, l \\
& (10)\, f_{inter}(q_j) * f_{eq}(q_j) * \left((y_{i_1,d_1,i_2,d_2} * q_j) - C_{r_j,l} \right) = 0, \forall i_1, i_2 : i_1 \neq i_2, \forall d_1, d_2 : d_1 \neq d_2, \forall j, l \\
& (11)\, x_{i,d} \in \{0,1\} \\
& (12)\, y_{i_1,d_1,i_2,d_2} \in \{0,1\}
\end{aligned}
$$

Figure 5: A general form of the MILP optimization problem solved by the CSOS solver plug-in. The groups of equations from 1 to 12 are related to different combinations of constraint type and modelling purpose. The semantics of each group are described in the main text.

In order to express our constraints, we employ a set of auxiliary functions characterizing the set of QoS requirements. The first such function, called f_{intra}, distinguishes between *intra* and *inter* QoS requirements: it is defined to be 1 in case it is an *intra* requirement, 0 otherwise. A second characterization function is used to distinguish between ascending/descending QoS requirements and equality QoS requirements; as the previous one, it is defined to be 1 in case the requirement is of eq type, 0 otherwise. Finally, we consider *numerical* QoS requirements, that imply using up some quantity of an available resource. We note that numerical requirements in this modelization can be ascending or descending, but not of the equality type, so we introduce the third and last characterization function f_{num}, qualifying numerical versus non-numerical QoS requirements. Using the notation introduced in this section, we have the optimization problem shown in *Figure 5*.

In the definition above, we employ a set of integer binary decision variables $x_{i,j}$. each one representing whether the component c_i, $i \in \{1, \dots, P\}$, of application *A* has been deployed on the datacenter d_j, $j \in \{1, \dots, M\}$.

Our problem has overall 12 groups of constraints. The constraints of group (1) force each valid solution to unequivocally deploy each component of the application *A* on exactly one datacenter.

Constraints in (2), (3) and (4) ensure that any component allocation shall not exceed the resource limits of the datacenter on which it will be deployed; they differ in the type of QoS and resources they model. In constraint group (2) we deal with *numerical* (ascending or descending) requirements: the characterization functions f_{num} and f_{inter} are applied to each QoS requirements and multiplied with each other such that the constraint is ignored when the result is zero, since in this case it is always satisfied independently from the value assigned to the decision variable. Constraints in (3) handle *non numerical* (ascending or descending) requirements, while constraints in (4) apply to *non-numerical* requirements of the equality kind (i.e., the presence of a certain OS or graphics card, codifying a specific card model or OS with a unique integer value).

Constraints on requirements and resources defined over communication links are encoded by equation groups (8), (9) and (10) that, in a similar way to constraint groups (2), (3) and (4), model respectively *numerical* (i.e., bandwidth), *ascending/descending* (i.e. latency), and *equality* requirements. In both groups, numerical and non-numerical requirements of type "greater than" are modelled by changing the sign of both the requirement and the corresponding resource coefficient. For example, the latency requirement is an ascending non-numerical requirement of the type $q_{lat} \geq C_{r_{lat},l}$ that is transformed in the constraint $-q_{lat} \leq -C_{r_{lat},l}$.

In our problem modelling we need to check for a requirement over a certain communication channel between two different components i_1 and i_2 of the application *A*, if and only if (a) the two components are deployed on different datacenters d_1 and d_2, and (b) a network link exists between d_1 and d_2.

Some constraints of our model result from applying standard techniques to turn the initial formulation into one suitable for MILP optimization.

> In a straightforward modelization of the optimization problem, that we do not show here, each *inter* requirement is modelled by multiplying the two decision variables $x_{i_1,d_1} x_{i_2,d_2}$, if and only if a network link exists between d_1 and d_2. However, this approach leads to a non-linear programming problem. Such constraints are easily linearized by introducing a set of auxiliary binary decision variables y_{i_1,d_1,i_2,d_2} , such that
>
> $y_{i_1,d_1,i_2,d_2} = x_{i_1,d_1} x_{i_2,d_2}$, $\forall i_1, i_2 \in \{1, \dots, N\}$ and $d_1, d_2 \in \{1, \dots, P\} : i_1 \neq i_2$ and $d_1 \neq d_2$.

To restrict the new problem to the same set of solutions of the first modelization, we also need to introduce the new sets of constraints (5), (6) and (7). The constraints in (6) and (7) ensure that $y_{i_1,d_1,i_2,d_2} = 0$ if $x_{i_1,d_1} = 0$ or $x_{i_2,d_2} = 0$. On the other hand, the constraints in (5) ensure that $y_{i_1,d_1,i_2,d_2} = 1$ if and only if $x_{i_1,d_1} = 1$ and $x_{i_2,d_2} = 1$.

Finally, constraint groups (11) and (12) restrict to Boolean values our decision variables **x** and auxiliary variables **y**.

The objective function is a general utility function over the two vectors of the decision variables of the optimization problem, **x** and **y**. Such utility function can encode a utility value specified by the owner of the application, exploiting a linear combination of the coefficient of the optimization problem, The objective function thus embeds the policy metrics that we want to be optimized by the orchestration.

2.2.2.3 HLO Approach Evaluation

To assess the viability of a multi-layered version of the HLO in overcoming the potential scalability limitation of MILP-based matchmaking processes, we show two sets of experiments, each set run using a different scenario. The *LARGE* scenario considers a larger set of different edge virtual clusters (70 EMCs = 700 computing nodes), but lower contention (500 app instances) with respect to the *MEDIUM* scenario, where a smaller set of virtual clusters (50 EMCs = 500 computing nodes) manages a higher contention for resources (400 app instances).

We used fixed structure applications, where the requirements declared by each application in terms of resources are parametric and randomly generated as follows in *Table 1*:

Table 1: Application and Resource parameters for the evaluation.

App parameter / Cluster Capabilities	Distribution	Application range	Virtual cluster resource range
CPU cores	random uniform	[1,8]	[20,100]
RAM	random uniform	[128,1024] Mbytes	[2048,4096] Mbytes
Storage	random uniform	[10,128] Gbytes	[200,1000] Gbytes

The experiments have been performed using a relatively homogeneous set of resources. We fixed the number of nodes managed by each EMC to 10, and randomly generated the features of the EMC nodes, as also reported in *Table 1* above.

Both scenarios have been simulated by running an instance of the MILP solver on a local Intel 4-cores i7600K CPU machine, with 16 Gbytes of RAM and feeding it with a TOSCA file describing each application instance, taking each one from a file of randomly generated deployment requests and applying the request parameters to the TOSCA Blueprint. The test performs the matchmaking with MILP-based solver and produces an output file with the final deployment plan, where each unique application request is associated with a virtual cluster. In the output file each result is associated with the time in seconds that the matchmaking process required.

We compare a “flat” version of the orchestration with a layered one, reproducing the HLO/LLO interaction by having a central HLO that delegates local orchestration to each virtual cluster. In the flat version a single MILP module must solve the matchmaking over the whole group of resources in the platform. In the layered version each MILP instance is instead invoked once to matchmake on the groups of aggregated resources, i.e., the groups associated to the EMC virtual clusters, and then once per each involved EMC (to simulate the local orchestration by LLO on the 10 nodes of the EMC).

As we show in *Figure 6*, MILP solver execution time both for the *MEDIUM* (left) and *LARGE* (right) scenarios, the time needed to solve a single MILP problem for the whole platform and set of requests is not acceptable, showing that optimal orchestration with MILP does not easily scale to large platforms if we model the architecture in a flat way. The size of the MILP problem, whose key characteristics are shown in *Table 2* for the LARGE scenario, grows too quickly. On the other hand, if we group resources by virtual clusters and optimize the MILP first at the high level and then locally for each domain, the service time of the optimization process is well below one second for all tests. We can see in *Figure 6* that the cost of the central MILP solution (marked as Sauron in the plot) is around 10^{-2} - 10^{-1} s. The MILP approach cost for the local orchestration is a safe over estimation of the time required for the low level orchestration, which in the CHARITY platform is delegated to the LLO, and it is comparable with the high-level MILP execution time.

The results validate the solution of layered orchestration according to the HLO/LLO decomposition even for the case of detailed QoS models and costly optimization algorithms, as long as resources are grouped together by their virtual clusters, allowing the high-level orchestrator to deal with one or more orders of magnitude less variables via its (MILP) solver plugin.

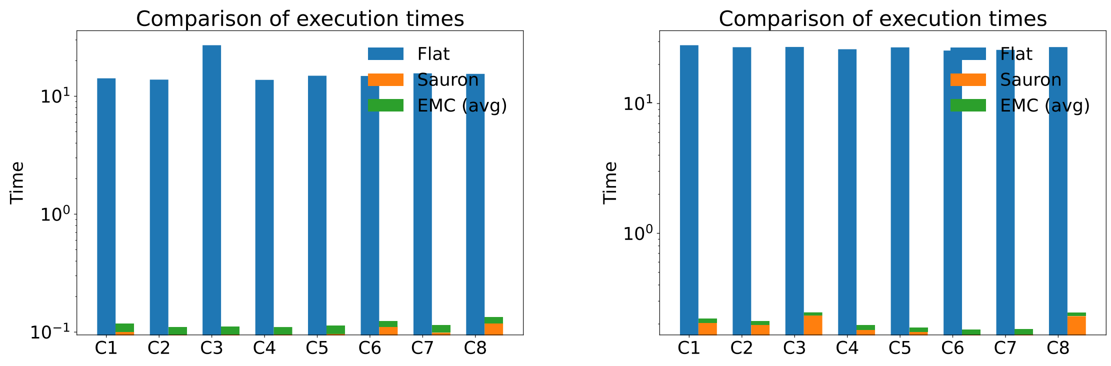


Figure 6: MILP solver execution time in seconds, flat vs layered execution (Sauron = main HLO); (left) MEDIUM scenario (right) LARGE scenario.

Table 2: Number of MILP variables and constraints in the LARGE scenario, flat vs layered orchestration.

Orchestration Version	#apps	#resources	Decision variables	#constraints in group 1	#constraints in group 2
Flat (single MILP instance)	500	700	350000	500	2100
Layered (Sauron = top level HLO)	500	70	35000	500	210
Layered (domain HLO or LLO) ***worst case**	*500	10	*5000	*500	30

To conclude, the above-mentioned Complex Scenario Optimal Solver (CSOS) tackles the Multi-Resource Multi-Objective Generalized Assignment Problem (MRMOGAP) for optimally deploying application components across available resources in the CHARITY federation. This is done by formulating the MRMOGAP as a Mixed-Integer Linear Programming (MILP) problem. CSOS parses the application's TOSCA blueprint and gathers data on available resources to generate the MILP instance, encoding constraints such as not exceeding resource capacities, satisfying component requirements (e.g. CPU, memory, latency), and ensuring each component maps to exactly one resource. It then solves the MILP problem using solvers like COIN-OR CBC to find the optimal allocation. If a solution exists, CSOS augments the blueprint with allocation details for deployment; otherwise, it requests additional resources from the orchestrator. The CSOC facilitates the HLO module to perform the high-level phase of resource allocation in the CHARITY platform by deciding on the most suitable deployment topology with regard to the specific XR application deployment requirements and resources available in the CHARITY deployment targets.

3 Low Level Orchestration

This section delves into the solution design for the Low-Level Orchestrator (LLO) intending to provide the means to enforce proper resource instantiation in multi-domain infrastructures. The Low-Level Orchestrator is the component of the system responsible for infrastructure provisioning, networking, and application deployments. It receives application specifications and orchestration decisions from the High-Level Orchestrator. Subsequently, it implements these decisions made regarding the distribution of application components across the existing infrastructure. The LLO can also serve as an interface for system administrators to interact with the system, leveraging their technical knowledge and familiarity with low-level topics. This provides them with more control and advanced customization capabilities.

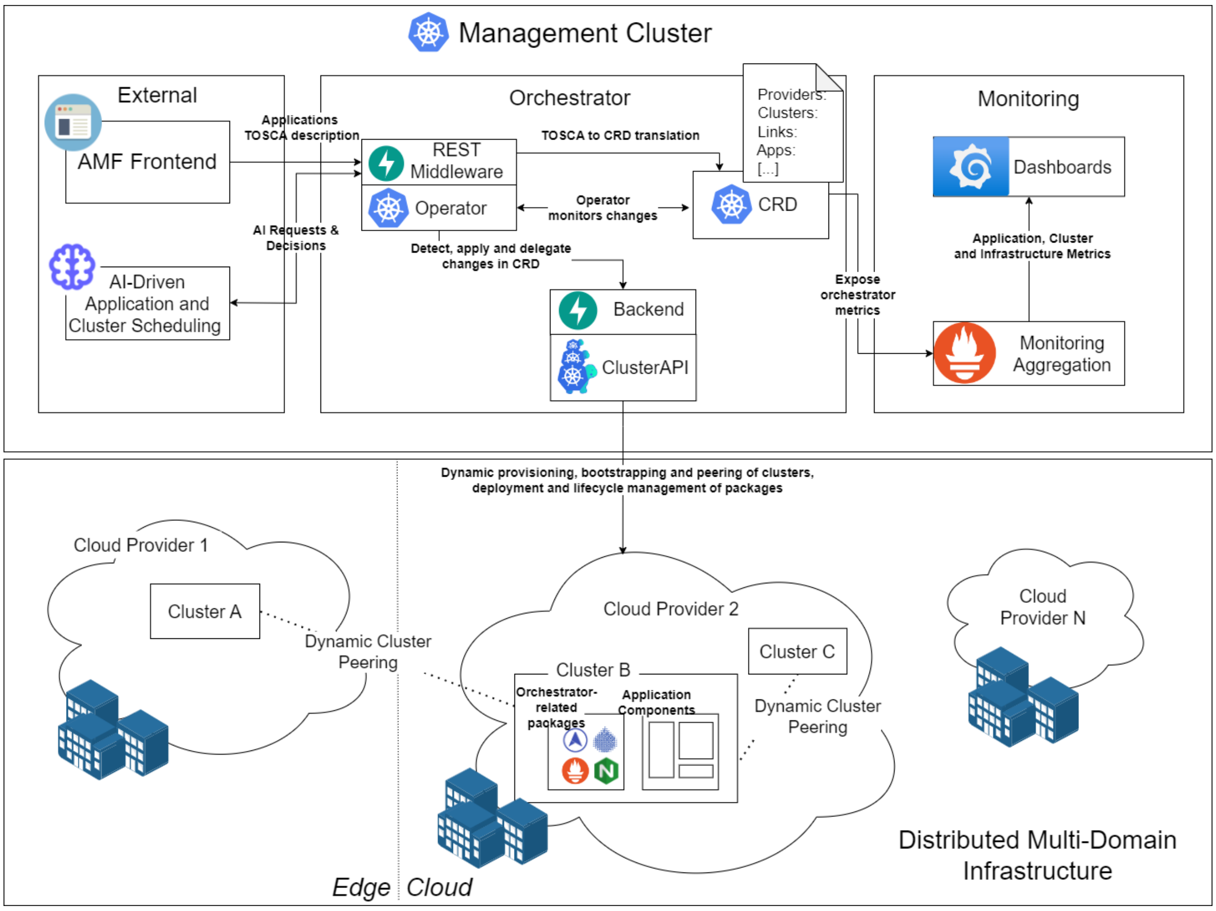

Management Cluster
External
AMF Frontend
AI-Driven Application and Cluster Scheduling
Applications TOSCA description
AI Requests & Decisions
Orchestrator
REST Middleware
Operator
TOSCA to CRD translation
Operator monitors changes
Providers: Clusters: Links: Apps: [...]
CRD
Detect, apply and delegate changes in CRD
Backend
ClusterAPI
Expose orchestrator metrics
Monitoring
Dashboards
Application, Cluster and Infrastructure Metrics
Monitoring Aggregation
Dynamic provisioning, bootstrapping and peering of clusters, deployment and lifecycle management of packages
Cloud Provider 1
Cluster A
Dynamic Cluster Peering
Cloud Provider 2
Cluster B
Orchestrator-related packages
Application Components
Cluster C
Dynamic Cluster Peering
Cloud Provider N
Distributed Multi-Domain Infrastructure
Edge
Cloud

Figure 7: The High Level Concept behind CHAIRTY´s LLO – Management Cluster and Distributed Clusters on a Multi-Domain Infrastructure.

A brief description of the components composing the LLO architecture is given below (see *Figure 7*):

• **Custom Resource Definition**: The CRD contains the most up-to-date information regarding the status of the infrastructure.

• **Operator**: The operator is responsible for monitoring the CRD and handling its changes (i.e., insert, delete and update). When a change to the CRD is detected, the operator acts accordingly. For instance, adding a cluster definition to the CRD will trigger a new cluster provisioning. The operator was inspired by the ETSI ZSM standard and closed-loop concept as a way to orchestrate and automate a Kubernetes based environment.

• **Backend**: The backend is responsible for executing the changes to the infrastructure detected by the operator. The backend is complemented by a REST API that the operator uses to request the changes detected, and the backend uses the data received to translate the changes (e.g., the operator requests the creation of a cluster, sending the data collected from the CRD, the backend uses the data to create the cluster in the infrastructure and reports the feedback).

• **Monitoring Aggregation**: It is responsible for collecting metrics exposed by the different components composing the orchestrator, as well as infrastructure metrics (e.g., number of clusters running, number of providers, number of applications).

• **Dashboards**: The dashboard allows visual feedback of the metrics collected by the monitoring aggregation component.

• **Distributed Multi-Domain Infrastructure**: The infrastructure relies on different cloud providers that will host the clusters for the applications. An application can be distributed or replicated across different clusters and cloud providers. The clusters can be connected across different providers, if the distribution of the application demands it, as to guarantee communication between the components comprising the application.

The orchestrator is prepared to be distributed across clusters, as each of the components composing the orchestrator is independent. For example, the backend could be in a different cluster than the operator and it will not affect the orchestration system's operation, as long as the clusters have access to each other. This independence is given through the developed middleware, acting as a bridge between the components. The orchestrator is also prepared to support different cloud providers and as such, the environments created by the orchestrator can be distributed across the supported providers.

3.1 Operator Concept

The low level part of the CHARITY Orchestration systems receives the orchestration decision from the high level ones. The corresponding decision is formalized using an orchestration blueprint that defines the end state that the low-level orchestration should achieve. This later analyses the current orchestration state and creates an execution plan to reach the communicated target state. In case of a failure, a remediation plan is followed.

The core of the low level orchestration is based on a Kubernetes Operator implementation and handles the automatic lifecycle management of a given orchestration. It is important to note that operators are custom extensions to built-in Kubernetes controllers which are the main components of the control plane in Kubernetes. It is considered a state of the art for the orchestration of static applications workload in a cluster. Our operator implementation extends Kubernetes to manage multi-domain cloud orchestration and inter-domain connectivity as explained below.

Since Kubernetes represents the core of our system, specifications for clusters and applications need to be defined in a way that is compatible and readable by the Operator and transformed in resources that can be used by the Kubernetes infrastructure. In our case, Custom Resource Definitions (CRDs) serve as the specification format. They extend the Kubernetes API, enabling users to define and use their own custom resources alongside the default resources managed by Kubernetes itself. As shown in *Figure 8*, we have defined a CRD containing three different elements: Clusters, Links, and Applications. From the CRD, multiple instances of resources can be created depending on the required specification.

Leveraging both CRDs and Operator concepts, the LLO monitors changes to the resources and takes actions to reconcile the current state of the resource with its desired state, providing major automation in the management of the infrastructure and the deployments. These components play a crucial role in streamlining complex tasks through the use of dedicated frameworks designed to simplify their implementation. This feature enables developers to focus more on the application logic, leading to accelerated iteration and development cycles.

Following this idea, the operator assumes two primary responsibilities. Firstly, it verifies whether Custom Resources (CR) created by the user adhere to the custom resource definition (CRD). Failure to comply with the CRD leads to unsuccessful resource creation. Secondly, the operator actively monitors Custom Resource (CR) events, including creation, update, deletion, and more. For each event, it executes the predefined logic, ultimately facilitating the reconciliation of the resource's current state to its newly specified desired state. This dual role ensures the integrity and consistency of the system's resources throughout their lifecycle.

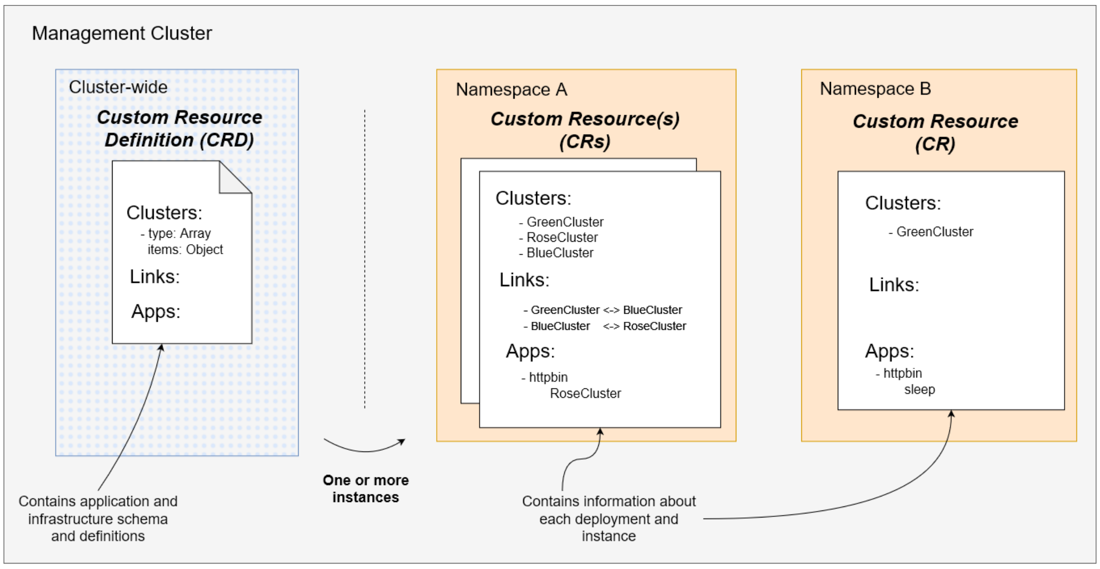


Figure 8: Custom Resource Definition (CRD) in the CHAIRTY´s LLO.

The implementation of such operator, constantly monitoring the defined CRs, was though to respect the Observe, Orient, Decide, Act (OODA) loop, represented in *Figure 9*. First, the operator is waiting for the monitored resources to change (observe phase). If any change is detected, the operator is responsible for admitting and validating the change, which consists of checking if the change is allowed and syntactically valid (orient phase). Based on the changes made to the resources, it detects what type of operation is needed (i.e. create, update, delete) and decides how to act based on the type of operation, outlining the execution plan (decide phase). Finally, based on the decision from the previous step, instead of executing the decided action himself, it requests action from the backend component, responsible for changing the infrastructure according to the data and demands received from the operator (act phase). This describes the recurring working process of the operator.

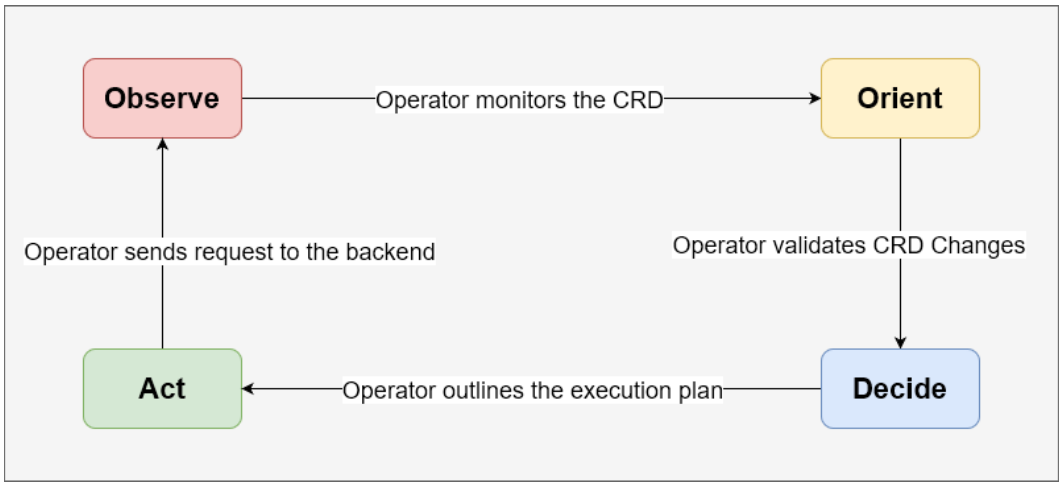

Observe
Operator monitors the CRD
Orient
Operator validates CRD Changes
Decide
Operator outlines the execution plan
Act
Operator sends request to the backend

Figure 9: Operator's OODA Loop.

3.2 Multi Domain Cloud Orchestration

To materialize the automation capabilities brought by the operator, there is the backend which is a containerized component running on the management cluster designed to meet the needs of the operator. Each event (creation, deletion, update) that the operator monitors undergo a complex process in the backend, performing all the necessary tasks for the event to be successfully completed.

The backend handles all the heavy-duty operations of the orchestration system involved in updating the infrastructure (i.e., when requested to create a cluster, the backend communicates with Cluster API and OpenStack to create said cluster). The backend exposes a REST interface used by the operator to request changes to the infrastructure. The REST interface is implemented with FastAPI. To deliver most of the operations, the backend integrates with Cluster API, Openstack and Liqo. The backend working cycle starts when the operator detects a change in the CRD, which also demands a change in the infrastructure. This triggers a request to the backend REST interface, making then the backend act accordingly (activating a functionality) to the request itself and data received within the request. This should be assumed for all the functionalities explained in more detail, further in this Section. All the cluster operations made by the backend (i.e., create, update, delete, interact) integrate the Cluster API framework, enabling more powerful and robust functionalities, and a better-managed infrastructure as the clusters and its resources (i.e., worker machines, control-plane machines) are stored as CRs and are independent per cloud provider (i.e., Openstack, AWS, BYOH) and per bootstrap provider (i.e., Kubeadm, MicroK8S, K3S). As default bootstrap provider and cloud provider for the functionality's explanation, found further in this Section, it should be assumed Kubeadm and Openstack, respectively.

The LLO relies heavily on Cluster API which provides a declarative API for cluster creation, configuration, and management. It allows users to define the desired state of their Kubernetes clusters using YAML manifests. The Cluster API controllers then work to reconcile the actual state with the desired state, ensuring that the clusters are provisioned and maintained according to the specified configuration. This abstraction layer makes it easier to automate cluster management tasks, deploy clusters across different cloud providers or on-premises environments, and maintain consistency in cluster configurations.

The cluster creation process includes the creation of the VM, which will host the cluster and the bootstrapping of the cluster itself. The creation of a cluster starts by loading the correct bootstrap and cloud provider (e.g., Kubeadm and Openstack, respectively) from Cluster API based on the data received from the operator, which is also stored in the CRD. Following this, the backend runs a script that generates the cluster CR based on a CRD from ClusterAPI, and filled using the data (i.e., name, image, machine count) from the request. During the generation of the manifest, it is also added to PostKubeadmCommands field of the generated cluster CR, bash commands to download, install and configure the CNI of the cluster automatically. Although this makes the creation of the cluster slightly slower, it is rewarded with the creation of a cluster ready to use. After the generation of the cluster manifest, it is then applied using the kubernetes CLI. The management cluster is able to interpret the manifest as it is based on the Cluster API CRD. During the cluster deployment, after the host VM is created, it is also created a floating IP. The floating IP is used by Cluster API to generate the cluster access file (kubeconfig). Although the creation of the floating IP is done automatically by Cluster API and Openstack, the assignment of this IP to the VM is not. The automation of this step is central for seamless cluster creation, so a solution was implemented. This assignment is done by checking if the VM is created, by checking the Cluster API CRs, and when it is created, it assigns the floating IP previously created to the host VM, automating the floating IP assignment step. Finally, the backend waits for the cluster to be ready which is when all the nodes of the cluster achieve a READY status. The orchestrator supports the installation of add-on packages (i.e. Liqo, NGINX, MetalLB, Prometheus, Kafka) and these packages can be installed after the cluster is ready. Due to infrastructure limitations, these packages are all installed by default with every cluster apart from Kafka as it is a more resource intensive add-on. The cluster creation is truly finished when all the selected packages are installed. After the cluster is created and ready, it is possible to scale the cluster. The orchestrator supports scaling the control plane nodes and the worker nodes, which can be scaled individually. The operator sends the cluster's name, the control plane node count and the worker node count saved within the CRD. As the control plane and worker machines are stored in individual CRs, the backend is capable of distinguish between them and checks for the resources individually. If the resources exist, the backend changes their replica count to the desired value, Cluster API detects changes to the CRs and acts accordingly, scaling the cluster nodes, up or down. For cluster deletion, the operator sends the cluster's name registered in the CRD. As the cluster resources are uniquely named, the backend checks for the CR residing in the management cluster and proceeds to delete the Kubernetes resource. Subsequently, this triggers the deletion of the cluster in the infrastructure.

The orchestrator is prepared to deploy containerized applications in clusters hosted within the providers integrated with Cluster API. If the applications are composed of more than one component, the whole application can be deployed in a single cluster or distributed across different clusters. The application deployment process begins with a request sent from the operator to the backend with the information regarding the application registered within the CRD. The backend translates the information received into native Kubernetes resources (i.e., deployment, services, ingresses) and deploys each component individually, as not all the components need to be exposed through an ingress or even a service. It is also during this step that the backend distributes the components across different clusters using Liqo's offloading feature according to the information registered within the CRD. The clusters where the components of an application should be deployed are included in the information gathered from the CRD, as each component may have a different cluster associated. The backend uses Kubernetes labels and adds the name of the cluster as a label to the Kubernetes deployment resources of each component (the Kubernetes deployments are created from custom templates) with the name of the cluster associated to the component. As labels are native to Kubernetes, Liqo already knows how to leverage them during the offloading phase done by the backend using a bash command via the Liqo CLI, installed in the backend component. As every application and its components are namespaced Kubernetes resources, when the deletion process is requested by the operator, the backend receives only the name of the application, which is used to identify the namespace where the application is deployed. Even if the application is distributed across clusters, as the identification of the namespace is kept across clusters, the backend unoffloads the namespace regardless of the application being distributed or not, as it does not affect the outcome or the performance of the process. After the unoffloading, the backend deletes the namespace and all the Kubernetes resources within.

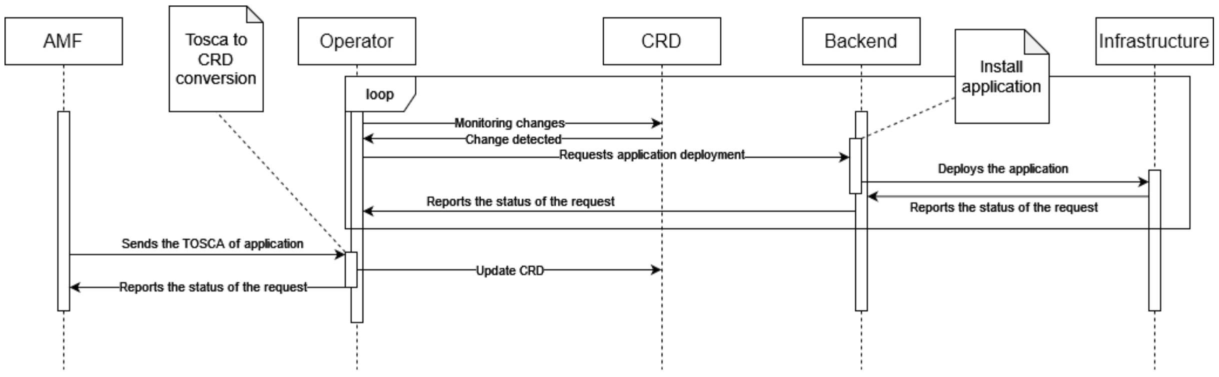
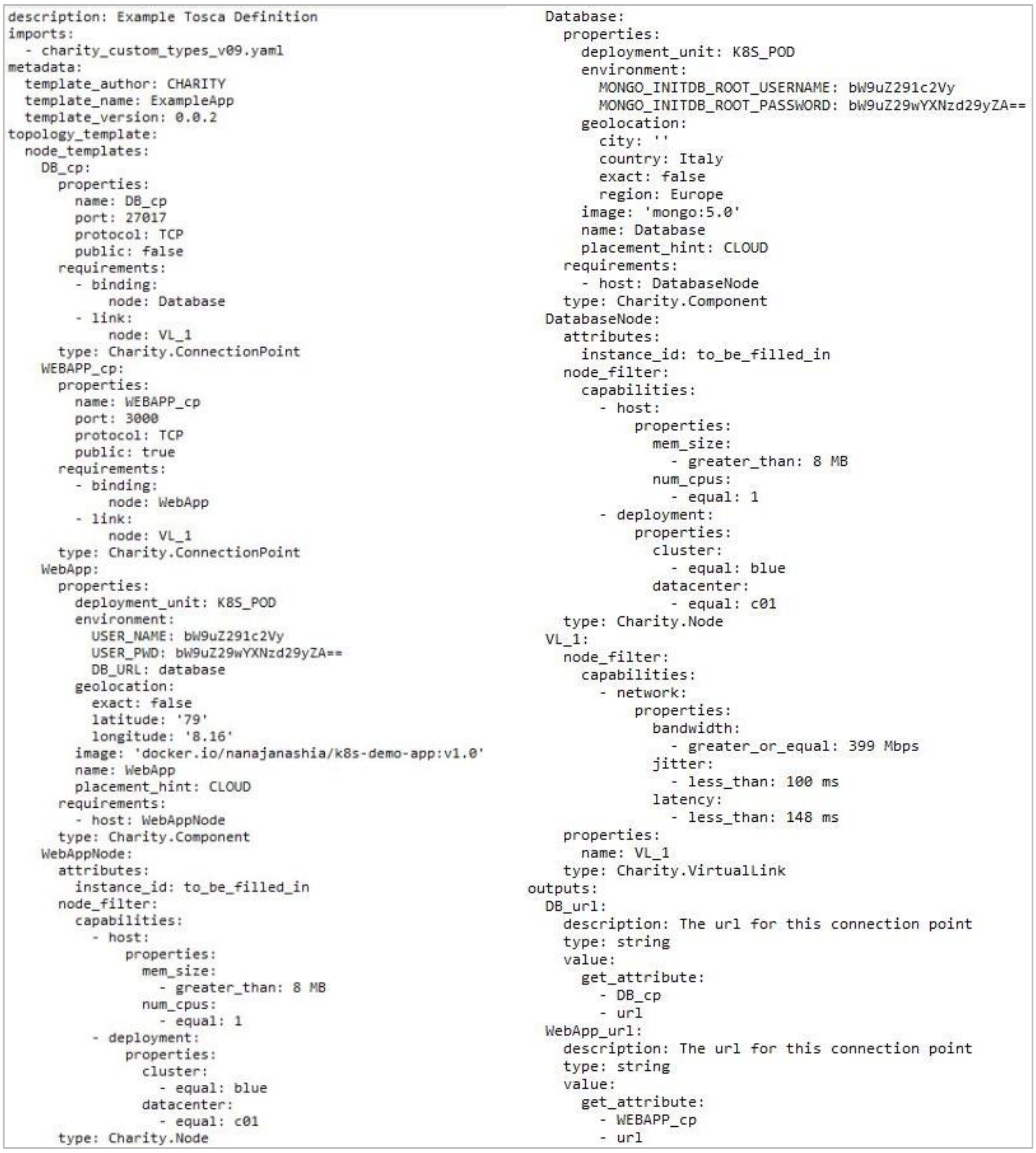

AMF
Tosca to CRD conversion
Operator
CRD
Backend
Install application
Infrastructure
loop
Monitoring changes
Change detected
Requests application deployment
Deploys the application
Reports the status of the request
Reports the status of the request
Sends the TOSCA of application
Update CRD
Reports the status of the request

Figure 10: Orchestration system handling the AMF Input.

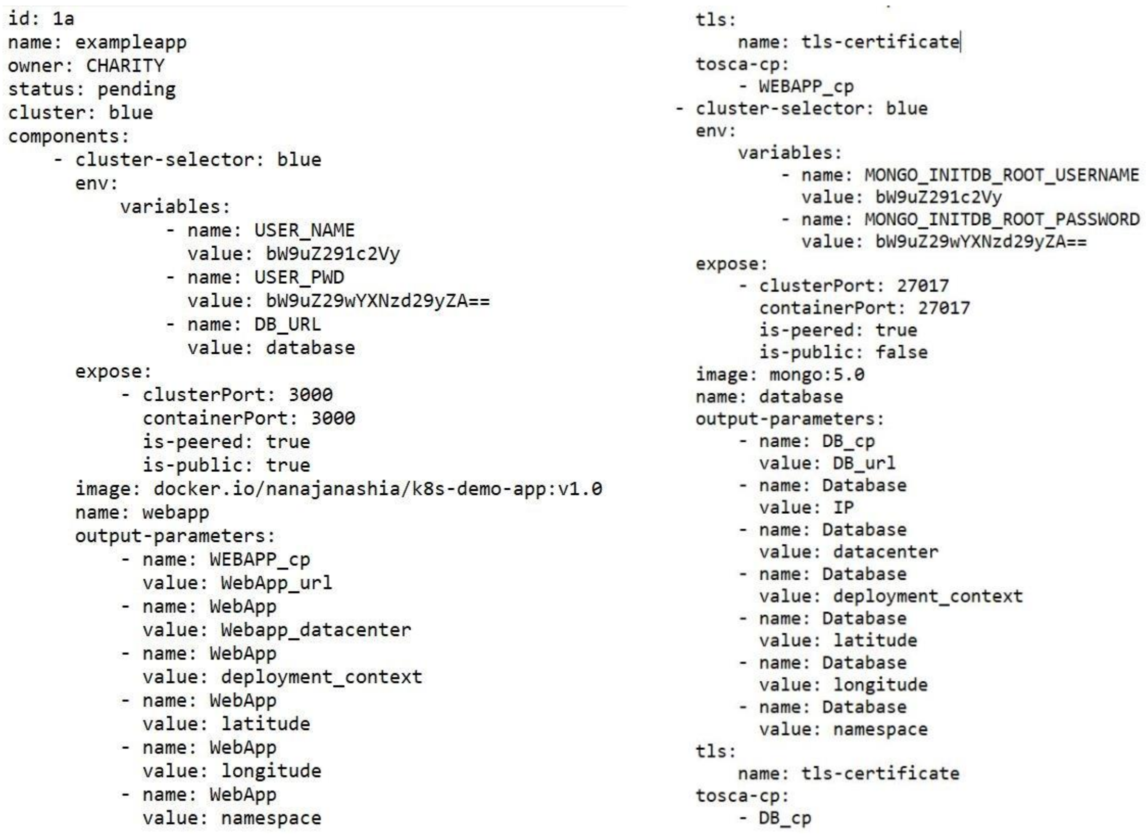
In order to enable the trigger of the LLO's operations and following the communication standards within the context of the CHARITY project, the LLO has the responsibility to transform high-level data sent from the AMF/HLO to the infrastructure level (i.e., Kubernetes Custom Resources (CR)), bringing abstract concepts from NFVs (e.g., Virtual Links, Connection Points) to Kubernetes, where there's not a direct mapping. To fulfil this responsibility, a parser was developed to enable communication between the LLO and, subsequently, with the infrastructure. The parser is used whenever a deployment-related endpoint is called. The input of these endpoints are TOSCA blueprints which are a standard to define and specify cloud applications. The TOSCA parser acts as a translator of the TOSCA blueprint to changes in the LLO CRs. Following the changes in the CRs, the LLO detects the required operations at the infrastructure level and creates/updates/deletes the Kubernetes resources needed to deploy such changes. For instance, as aforementioned, these TOSCA blueprints hold attributes such as Virtual Links and Connection Points which are ultimately converted to Kubernetes Services to be applied to the actual infrastructure, this workflow can be seen in *Figure 10*. Moreover, attributes such as Nodes and Components, present in the TOSCA specification, are converted to Kubernetes resources (e.g. deployments) which allow the actual instantiation of applications in Kubernetes-based environments. Besides the application's specification, the TOSCA blueprint also allows the definition of output variables to be returned by the LLO. The parser extracts the variables specified in the blueprint so that the LLO can retrieve them to the AMF/HLO. The output parameters can range, for instance, from the URL of a service to the Kubernetes namespace where the services are deployed.



```
description: Example Tosca Definition
imports:
  - charity_custom_types_v09.yaml
metadata:
  template_author: CHARITY
  template_name: ExampleApp
  template_version: 0.0.2
topology_template:
  node_templates:
    DB_cp:
      properties:
        name: DB_cp
        port: 27017
        protocol: TCP
        public: false
      requirements:
        - binding:
            node: Database
        - link:
            node: VL_1
      type: Charity.ConnectionPoint
    WEBAPP_cp:
      properties:
        name: WEBAPP_cp
        port: 3000
        protocol: TCP
        public: true
      requirements:
        - binding:
            node: WebApp
        - link:
            node: VL_1
      type: Charity.ConnectionPoint
    WebApp:
      properties:
        deployment_unit: K8S_POD
        environment:
          USER_NAME: bW9uZ291c2Vy
          USER_PWD: bW9uZ29wYXNzd29yZA==
          DB_URL: database
        geolocation:
          exact: false
          latitude: '79'
          longitude: '8.16'
        image: 'docker.io/nanajanashia/k8s-demo-app:v1.0'
        name: WebApp
        placement_hint: CLOUD
      requirements:
        - host: WebAppNode
      type: Charity.Component
    WebAppNode:
      attributes:
        instance_id: to_be_filled_in
      node_filter:
        capabilities:
          - host:
              properties:
                mem_size:
                  - greater_than: 8 MB
                num_cpus:
                  - equal: 1
          - deployment:
              properties:
                cluster:
                  - equal: blue
                datacenter:
                  - equal: c01
      type: Charity.Node
    Database:
      properties:
        deployment_unit: K8S_POD
        environment:
          MONGO_INITDB_ROOT_USERNAME: bW9uZ291c2Vy
          MONGO_INITDB_ROOT_PASSWORD: bW9uZ29wYXNzd29yZA==
        geolocation:
          city: ''
          country: Italy
          exact: false
          region: Europe
        image: 'mongo:5.0'
        name: Database
        placement_hint: CLOUD
      requirements:
        - host: DatabaseNode
      type: Charity.Component
    DatabaseNode:
      attributes:
        instance_id: to_be_filled_in
      node_filter:
        capabilities:
          - host:
              properties:
                mem_size:
                  - greater_than: 8 MB
                num_cpus:
                  - equal: 1
          - deployment:
              properties:
                cluster:
                  - equal: blue
                datacenter:
                  - equal: c01
      type: Charity.Node
    VL_1:
      node_filter:
        capabilities:
          - network:
              properties:
                bandwidth:
                  - greater_or_equal: 399 Mbps
                jitter:
                  - less_than: 100 ms
                latency:
                  - less_than: 148 ms
      properties:
        name: VL_1
      type: Charity.VirtualLink
  outputs:
    DB_url:
      description: The url for this connection point
      type: string
      value:
        get_attribute:
          - DB_cp
          - url
    WebApp_url:
      description: The url for this connection point
      type: string
      value:
        get_attribute:
          - WEBAPP_cp
          - url
```

Figure 11: TOSCA Model Example.

In *Figure 11*, an example of a TOSCA specification can be seen. The TOSCA model possesses four major types of elements: the Components, the Nodes, the ConnectionPoints, and the VirtualLinks. All these elements have diverse attributes and requirements that need to be translated to actual components that can be perceptible to Kubernetes. The Components are the elements that provide information about the service itself, such as the image and the name of the service. The Nodes have information about the components of the application, such as the image and the technical requirements of the service. The VirtualLinks provide mainly information about the technical requirements of the connection, such as the minimum bandwidth and the maximum latency required. The ConnectionPoints have information about the exposition of the components. Specifications such as the port and the protocol of the service, as well as the VirtualLinks to which the service is associated. In this way, it is possible to know which components are connected to which and how the communication between them is supposed to be done. To accomplish the propagation of the output parameters mentioned above, there is the outputs field. Each output instance holds the name of the element related to the needed variable (e.g. a specific connection point) and the actual variable (e.g. URL) to be returned by the LLO in a specific deployment. Additionally, each output parameter has a description and the type expected by the HLO/AMF.



```
id: 1a
name: exampleapp
owner: CHARITY
status: pending
cluster: blue
components:
    - cluster-selector: blue
      env:
          variables:
              - name: USER_NAME
                value: bW9uZ291c2Vy
              - name: USER_PWD
                value: bW9uZ29wYXNzd29yZA==
              - name: DB_URL
                value: database
      expose:
          - clusterPort: 3000
            containerPort: 3000
            is-peered: true
            is-public: true
      image: docker.io/nanajanashia/k8s-demo-app:v1.0
      name: webapp
      output-parameters:
          - name: WEBAPP_cp
            value: WebApp_url
          - name: WebApp
            value: Webapp_datacenter
          - name: WebApp
            value: deployment_context
          - name: WebApp
            value: latitude
          - name: WebApp
            value: longitude
          - name: WebApp
            value: namespace
      tls:
          name: tls-certificate
      tosca-cp:
          - WEBAPP_cp
    - cluster-selector: blue
      env:
          variables:
              - name: MONGO_INITDB_ROOT_USERNAME
                value: bW9uZ291c2Vy
              - name: MONGO_INITDB_ROOT_PASSWORD
                value: bW9uZ29wYXNzd29yZA==
      expose:
          - clusterPort: 27017
            containerPort: 27017
            is-peered: true
            is-public: false
      image: mongo:5.0
      name: database
      output-parameters:
          - name: DB_cp
            value: DB_url
          - name: Database
            value: IP
          - name: Database
            value: datacenter
          - name: Database
            value: deployment_context
          - name: Database
            value: latitude
          - name: Database
            value: longitude
          - name: Database
            value: namespace
      tls:
          name: tls-certificate
      tosca-cp:
          - DB_cp
```

Figure 12: Parsed Input Example.

With the parsing of the TOSCA model, the system obtains a YAML similar to that shown in *Figure 12*. In this example, the information represents the TOSCA data in a Kubernetes-like language. This information is split by the components obtained via TOSCA specification. Within the "components" field, there can be a great variety of attributes regarding each component including the "service", which represents some attributes that the system needs to define how to expose the component within the context of a Kubernetes service, the "env" that contains the environment variables needed by the component and the output-parameters which, although not useful in the scope of Kubernetes resources, are stored to be retrieved ahead. Moreover, attributes such as the image, the name, and the destination cluster of the components are also extracted. Furthermore, LLO converts this information into actual Kubernetes resources (e.g. deployments).

3.3 Inter Cloud Domain Connectivity

Regarding cluster interconnectivity, the backend integrates Liqo framework, which, in addition to providing a solution to cluster interconnectivity, provides support to workload distribution across the connected clusters. Using a peer-to-peer approach, Liqo provides interconnectivity between clusters, allowing workload offloading, service distribution across clusters, and multicluster applications traffic routing. Liqo becomes essential to enable seamless and secure communication among services deployed across various clusters distributed in multiple geographical zones. Liqo is an open-source project that enables dynamic and seamless Kubernetes multi-cluster topologies, supporting heterogeneous on-premises, cloud, and edge infrastructures. It provides several key features such as:

- Peering: Automatic establishment of VPN tunnel between two clusters (local and remote). The local cluster is the consumer of resources, and the remote cluster is the provider. The peering process results in the creation of a virtual node in the local cluster acting as an abstraction over the resources provided from the remote cluster.

- Offloading: Offloading occurs after the peering process concludes. Once the virtual node is operational in the local cluster, workloads can be offloaded by scheduling them to run on that virtual node. Subsequently, Liqo will automatically transfer these workloads to the remote cluster where they will be executed. Networking configurations of the workloads are also synchronized between the two clusters, enabling services to communicate seamlessly, as if they are running on the same cluster.

The features implemented by the orchestrator that integrate with the Liqo framework, are also exposed by the backend to the operator via HTTP endpoints. The starting point of the interconnectivity is the cluster peering. The backend receives the names of the clusters that should be peered with Liqo from the operator which are contained within the CRD. For the peering process, the backend does a series of checks, to make sure the peering can begin. The checks are done individually by cluster and only when both clusters pass all the checks, the peering process begins. First, the backend checks if the clusters are already deployed and ready. If this first check is verified, the backend generates the ".kubeconfig" files to access the clusters via the Cluster API CLI and perform the remaining checks. Accessing the cluster, the backend checks for the list of components composing the Liqo framework, verifying if they are ready and available. After all the checks for both clusters are validated, the clusters are peered, using the Liqo CLI (installed in the backend). As it functions similarly to the Kubernetes CLI, the ".kubeconfig" files used to access the clusters can be inserted as a parameter and the clusters are peered. If the cluster checks are not valid, the peering process stops. With peered clusters, it is possible to distribute components of an application, using Liqo's offloading capabilities. These capabilities allow the reflection and execution of workloads in remote clusters. Liqo allows to offload namespaces, services, and pods. For example, when offloading a namespace, Liqo extends it by creating an identical twin namespace in the remote cluster. This enables pods and services to operate within this shared cross-cluster namespace.

4 Monitoring & Forecasting

Monitoring encompasses the process of collection, analysis and use of information systematically, that provides the continuous visualization and perception of the status of an application, service or infrastructure. Such continuous monitoring process provides a way to analyse the environment to check whether applications and infrastructure run as expected. Indeed, the real-time monitoring of the environment allows, for instance, to minimize the response time to incidents (e.g., the detection and mitigation of cyber-attacks). Whenever an incident happen beyond the expected behaviour, it is possible to timely take the appropriate actions and decisions. In past, monitoring fundamentally served as a decision support for manual interventions of service and infrastructure administrators. As we progress into more complex and challenging scenarios, as envisioned in CHARITY, monitoring and prediction algorithms (based on the instrumented metrics) assume a new relevance in the orchestration and life-cycle management of next-generation applications. The input is formed upon which all the intelligent orchestration mechanisms are built. The concept of closed loops control and the envisioned automation of intelligent orchestration highly depends on a comprehensive real-time monitoring approach and on the quality of collected metrics.

This section explains the monitoring framework implemented for CHARITY to reach a dynamic multi-cluster architecture. Also, it details the data gathered, used as a data source for forecasting algorithms and responsible for activating the custom alerting system that leverages real-time and predicted performance information.

4.1 Goals and Research Challenges

In a Cloud Native environment, the monitoring process has a critical role to provide the required observability over the complex (and potentially large) number of micro-services spanning across distinct domains. Manually monitoring such an environment is no longer a viable task, instead the usage of monitoring tools allows the achievement of the required degree of automation. As we move towards intelligent orchestration platforms such as the CHARITY case, resource monitoring would play a critical role into supporting not only automation but has also a valuable input for all the resource prediction algorithms, as detailed later.

Monitoring tools provide observability over metrics at different levels, such as excessive or unusual CPU utilization patterns which might impact the system performance, memory-related metrics to detect memory leaks and other unexpected behaviours, disks running out of space, unauthorized network traffic flows or slow/bogus service APIs responses. In Cloud Native environments, such monitoring tools are used to ensure that infrastructure assets including servers, nodes, pods, containers behave as expected.

Resource monitoring, especially within a Kubernetes cluster, is a daunting and challenging task. There are literally hundreds of metrics which can be extracted from all the layers, components and applications and not all of them are always relevant for every single orchestration task. For instance, a network state prediction algorithm will likely need only a subset of network-related metrics. Additionally, dynamic, ephemeral and loosely coupled micro-services pose an architectural challenge of how to extract and collect all the different service-related metrics.

Hence, in a Cloud Native environment, one of the biggest challenges is not only to understand which metrics should be used but the efficiency and efficacy of the collection, aggregation and all the preprocessing, which ideally should occur near real-time. This challenge is further aggravated when one considers the traits of next generation of XR applications such as their latency constrains and the amount of involved data. Likewise, the monitoring of specific XR metrics poses different challenges. For instance, measuring the quality of a video streaming as perceived by users as part of the overall QoE assessment is a major challenge [1].

Moreover, monitoring tools should also have the following characteristics [2]: extensibility to accommodate scalable and dynamic environments in a flexible way; portability to enable the monitoring of distinct heterogeneous platforms and services; non-intrusiveness to avoid the interference with the resources that are being monitored (e.g., monitoring tools should not impact the already-constrained latency of XR applications);multi-tenancy to allow the monitoring of shared resources, accessibility; usability; robustness and achievability. The premises of the CHARITY project add new degrees of complexity:

- Multi-cloud: The CHARITY platform seeks to achieve the extreme KPIs required by XR applications enabling dynamic deployment of microservices, which implies a changing architecture of virtual applications through time. Each provider offers native monitoring tools to control and visualize the performance of their Kubernetes clusters and the services deployed. Hence, the use of cloud providers monitoring tools is not viable since the change of a server from one cloud to another should not be perceived by UC owners.
- Heterogeneity: The focus on XR applications supposes the opening to technologies beyond the well documented traditional ones. Being an area in full development, the components, languages and KPIs are still being defined and will not stop advancing as the use of XR reaches the personal use. The monitoring platform must adapt to the demands of development of the XR itself.

The goal of CHARITY's monitoring tool is to meet these architecture requirements without involving UC owners in the complex architecture wherein their application components are deployed. Therefore, agnosticity is the main requirement for monitoring: it must support all kinds of technologies, hardware and software, languages and service provider companies. CHARITY's monitoring tool should reduce the complexity of the underlying architecture, at least abstracting the UC owners from it into this constant point of contact they will have to control the performance of their applications and each of the microservices that make them up.

Prevention and reactivity are the objectives to reach with CHARITY and monitoring is a key part of it as it is one more piece in the chain of analysis, prediction, reaction and modification. The tool must adapt itself as dynamically as the application architecture itself will do based on multi-cloud performance and requirements. *Table 3* summarizes the above discussion and gives an overview on the monitoring requirements.

Table 3: Overview of monitoring requirements.

CHARITY Feature	Requirements	Orientation
Heterogeneity	Native monitoring Support for all clouds Multiple providers and technologies	Cloud agnostic Monitor services, resources and network
Complexity	Traceability of problems	Certain level of abstraction on the different clouds in use Single panels
Prevention	Customer impact Interrupted services	Analyse data and trends Proactive architecture
UC abstraction	Human involvement XR scopes continuously evolving	Cloud agnostic templates to automatize changes in monitoring system

4.2 Enablers and Tools

This section provides an overview on the considered monitoring tools, mainly focused on open-source tools that were investigated for supporting the monitoring of the infrastructure, applications and services as part of the CHARITY framework. Namely, it presents Prometheus and Grafana, ELK Stack, Kubewatch, Weave Scope, Zabbix, cAdvisor, Jaeger, Dynatrace and Datadog [3][4][5].

- Prometheus[5] is an open-source tool for monitoring (and alerting events) systems, services and applications. Prometheus collects the target metrics at certain intervals, evaluates the configured thresholds, and triggers alerts if any condition is true. Prometheus relies on the concept of exporters to export and send metrics from third-parties components to a central server. The communication between exporters and Prometheus Server is done via HTTP by default. In a Kubernetes deployment, a Prometheus server can leverage Kubernetes API and service discovery capabilities to directly pull specific node and service metrics. Grafana[6], often used to complement such metric collection, provides a flexible way to query the metrics persisted by Prometheus (using PromQL) and visualizes them into graphical dashboards. Grafana can be also leveraged for implementing alert functions.
- ELK stack[7] comprises the combination of ElasticSearch, Logstash, and Kibana. Together, they allow the collection of data from different sources in different formats and provides real-time data analysis and search capabilities. First, Logstash is used for collecting, transforming and sending data in real-time from data sources to ElasticSearch. Then, ElasticSearch, a search and analysis engine, supports the implementation of distinct analytics capabilities on the top of the collected data. Finally, Kibana, similar to Grafana, allows visualizing all of that data in the form of dashboards. Likewise, the combination of ElasticSearch, Fluentd and Kibana, known as EFK stack [6], can be also adopted. In that case, Fluentd replaces Logstash, as the component used to retrieve and ingest data into the ElasticSearch engine (e.g., it supports the ingestion of specific Kubernetes node related metrics). Moreover, Elastic Cloud on Kubernetes (ECK) [7], an ElasticSearch managed service, built on top of the Kubernetes Operator, allows to further streamline the ElasticSearch and Kubernetes integration by providing features such as the management and monitoring of multiple clusters or the cluster scale-in/down configuration changes.
- Kubewatch[8] is a Kubernetes specific open-source watcher used to track and notify changes of Kubernetes specific resources (e.g., pods, services, deployments, replica sets, replication controllers or even configuration maps). Whenever it detects configuration changes, it generates a notification to predefined collaboration hubs. Although this tool does not offer long term storage, trending, or analysis capabilities, it provides a simple and Cloud-Native approach to monitor for unexpected service modifications (e.g., as a result of a cyber-attack).
- Weave Scope[9] is a monitoring and visualization tool capable of providing operational insights from Kubernetes clusters. Weave Scope automatically generates topology maps of applications and infrastructure, enabling to intuitively understand and monitor context details (such as metrics, data or tags) and control containerized applications in real-time (e.g., stop, restart and pause containers as needed).

[5] https://prometheus.io/

[6] https://grafana.com/

[7] https://github.com/elastic/elasticsearch

[8] https://github.com/bitnami-labs/kubewatch

[9] https://www.weave.works/oss/scope/

- Zabbix[10] is an open-source software tool to monitor various devices, systems and applications through a large number of available integrations, both agent-based and agentless. For instance, it allows collecting from resource consumption and application-specific metrics up to auto-discovery of pods, deployments, services in Kubernetes deployment. Designed with automation in mind, Zabbix supports the monitoring of large and dynamic environments by offering auto-registration and discovery capabilities.
- cAdvisor[11] is an open-source agent to collect, process and export resource usage and performance information relative to running containers. Part of the Kubelet binary of Kubernetes, cAdvisor agent, can auto-discover containers in execution and collect resource-related metrics, such as memory, CPU, disk, files or network. Although cAdvisor does not offer long term storage, trending, or analysis capabilities, thus requiring a complementary monitoring solution. Nevertheless, its Kubernetes integration makes it a simple but effective tool for exposing container-level resource consumption metrics in a Cloud Native environment as a Kubernetes deployment.
- Jaeger[12] is an open-source solution that provides end-to-end distributed tracing capabilities such as consistent upfront sampling with individual per service/endpoint probabilities. Amongst others, Jaeger features can be leveraged to optimize the performance and latency of the services, an important aspect of the underlying concept of CHARITY. Jaeger provides a native Kubernetes integration through the implementation of a Kubernetes Operator (i.e., Jaeger Operator) which is composed of the Agent, Collector, Query components. The agent, which can be automatically injected as a sidecar, interfaces with the Jaeger clients (implementing an OpenTracing API for each application) and abstract the routing and discovery of the collectors. The Jaeger Collectors are responsible to receive the information from the agent, process it and store it in a specific storage backend. Finally, Jaeger Query provides the UI interface with the stored traces. Despite the benefits of such a tracing monitoring solution, the applications need to be aware of it and need to be designed to include the Jaeger client which exposes the OpenTracing API.
- Dynatrace[13] is a platform that has a solution to Infrastructure Monitoring that provides a unified view across the full Kubernetes stack, from applications to infrastructure and user experience. Dynatrace enables automated and intelligent observability with continuous auto-discover of hosts, virtual machines, cloud servers, containers and Kubernetes. Similar to Jaeger, Dynatrace provides a native integration for Kubernetes clusters, by implementing a Kubernetes Operator, named OneAgent. This agent can run as a DaemonSet in a Kubernetes cluster. It provides observability over the infrastructure and application levels in an automated and continuous way. Dynatrace allows for collecting resource consumption metrics of cluster components (i.e., containers, pods, nodes) up to controlling costs or root cause analysis of detected issues. This tool is not open-source as the aforementioned ones.
- Datadog[14] is a service that provides data observability to applications in the cloud and enables the monitoring of servers, databases, tools and services through a SaaS-based data analysis platform. This solution provides a platform that helps to monitor and track end to end requests, application performance, automatically collection, correlation and search of logs. To deploy Datadog, there are two different options: the deployment of Datadog agents as

[10] https://www.zabbix.com/

[11] https://github.com/google/cadvisor

[12] https://www.jaegertracing.io/

[13] https://www.dynatrace.com/

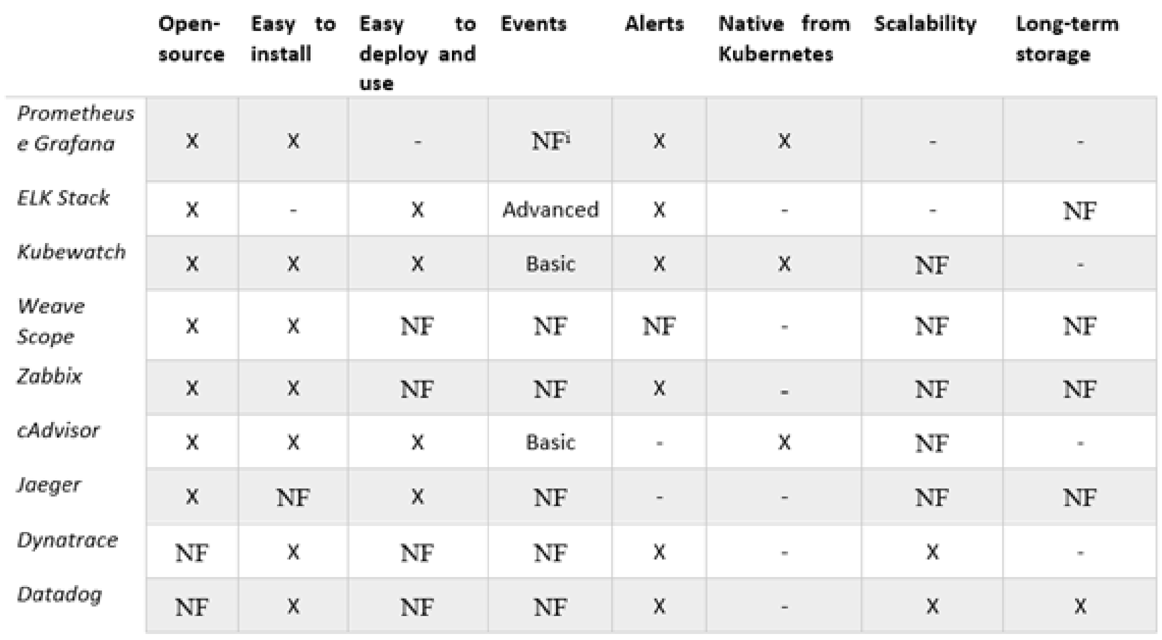
[14] https://www.datadoghq.com/

sidecars in all pods or the deployment of agents at the host level. Similar to Dynatrace, Datadog is not open-source.

- Kafka[15] is not a monitoring specific tool but an open-source distributed event streaming platform. Kafka was considered as part of the implementation Integration Fabric concept in the CHARITY framework. It is here referred to as a pivotal enabler to support not only the efficient communication between components but also the collection and aggregation of metrics from different heterogeneous assets and monitoring tools. Kafka is a widely used solution for implementing the messaging bus pattern. Clients (producers and subscribers) can asynchronously exchange messages with a common bus (i.e., Kafka topics and partitions). This provides a more decoupled communication strategy where each part (i.e., the CHARITY components) can publish and consume data as needed in a shared and dynamic environment. From a monitoring standpoint, such a model can be used to allow different monitoring components to expose their observed metrics. Then, each of the orchestration and management functions, according to their specific goals, can subscribe to such metrics. Moreover, Kafka provides a strategical role in the monitoring process itself as it allows to have a scalable and intermediate persistent layer for storing metrics with fault-tolerance capabilities. Additionally, Kafka Connect and Kafka Streams can also play an important role to facilitate the process of retrieving and consuming data from distinct tools in different formats and to support the pre-processing capabilities.

Table 4 presents a brief comparison of the aforementioned monitoring tools based on the initial findings and survey work [8].

Table 4: Comparison between monitoring tools.



	Open-source	Easy to install	Easy to deploy and use	Events	Alerts	Native from Kubernetes	Scalability	Long-term storage
Prometheus e Grafana	X	X	-	NF[i]	X	X	-	-
ELK Stack	X	-	X	Advanced	X	-	-	NF
Kubewatch	X	X	X	Basic	X	X	NF	-
Weave Scope	X	X	NF	NF	NF	-	NF	NF
Zabbix	X	X	NF	NF	X	-	NF	NF
cAdvisor	X	X	X	Basic	-	X	NF	-
Jaeger	X	NF	X	NF	-	-	NF	NF
Dynatrace	NF	X	NF	NF	X	-	X	-
Datadog	NF	X	NF	NF	X	-	X	X

[i] Information not found

One of the first parameters compared was the alerting capabilities. In CHARITY, these alerts are an important input for triggering automated response actions. cAdvisor and Jaeger do not natively support them, thus if needed it needs to be developed. Nevertheless, Jaeger offers unique service tracing capabilities not found in other tools and it might be of interest to have them in CHARITY.

[15] https://kafka.apache.org/

Despite the benefits of Dynatrace and Datadog, they are not open-source which can pose adoption barriers or customization issues. Kubewatch is useful for monitoring basic configuration changes but it does not fit into the purpose of collecting distinct metrics from a Kubernetes cluster. Zabbix although somehow more complex based on the initial observations, it offers a large range of integrations which might prove useful for certain task. In other ways, Weave Scope seems to be more focused on the graphical representations of elements (e.g., the topology maps) and it lacks other useful monitoring capabilities. Prometheus and the EFK stack are typically referred as the most widely adopted choices for a generic monitoring solution given their extensive list of features, large open-source communities and the range of integration options with third-party components (e.g., Prometheus exporters[16] or Fluentd datasources[17]).

Figure 13 illustrates the envisioned implementation of the EFK (Elasticsearch, Fluentd and Kibana) in CHARITY. Fluentd collects the metrics and send them to Elasticsearch. Elasticsearch can be leveraged to implement different analysis mechanisms, whereas Kibana can be used to visualize such information in a graphical interface. Likewise, Kibana can also feed the orchestration tasks by providing alerts.

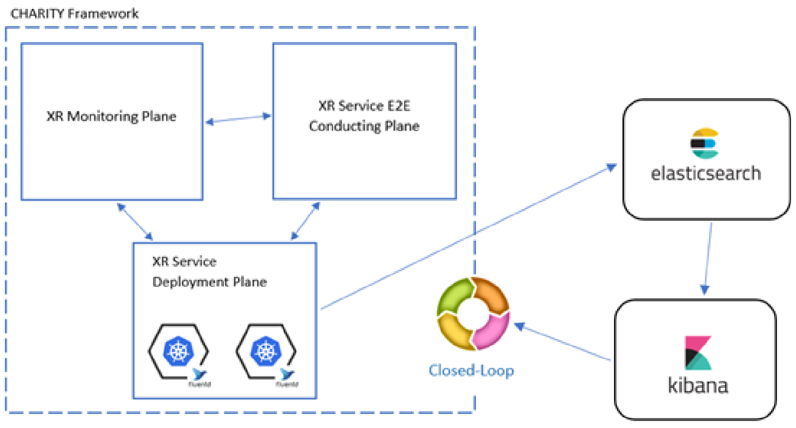


Figure 13: Implementation of EFK based on Kubernetes, adapted from[18].

Similarly, *Figure 14* illustrates how Prometheus and Grafana tools fit in the CHARITY project. Different Prometheus exporters are used to collect metrics from the deployment plane. These metrics, persisted by the Prometheus server, are meant to be used by the orchestration tasks. Likewise, the AlertManager Components of Prometheus can be also leveraged as an important input for orchestration tasks.

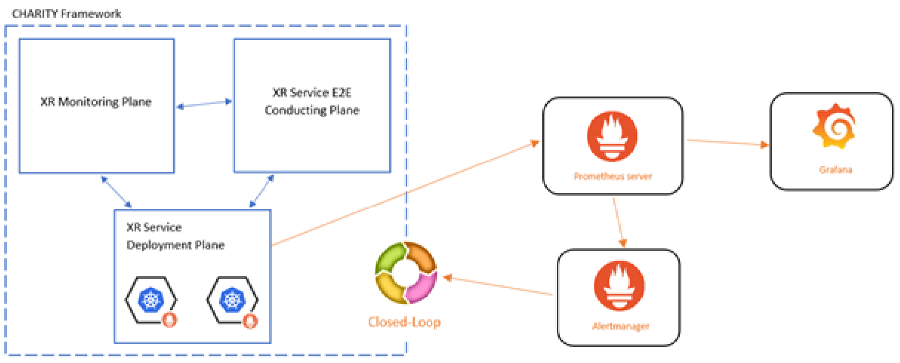


Figure 14: Implementation of Prometheus and Grafana based on Kubernetes adapted from[19].

[16] https://prometheus.io/docs/instrumenting/exporters/

[17] https://sematext.com/blog/kubernetes-monitoring-tools/

[18] https://dytvr9ot2sszz.cloudfront.net/wp-content/uploads/2018/06/kuberbetes-monitoring-arch-1.jpg

[19] https://sysdig.com/wp-content/uploads/Blog-Kubernetes-Monitoring-with-Prometheus-4-Architecture-Overview.png

Finally, it is important to refer that despite of the advantages of using a generic solution like Prometheus for persisting and exposing numerous heterogeneous metrics, the choice will actually depend of the exact metrics that are needed by each orchestration mechanism. Thus, distinct solutions will be further considered and investigated as part of the overall CHARITY research and development.

4.3 Monitoring Solutions

4.3.1 Cloud Data Monitoring

Crossing the existing tools with the demands of a XR monitoring tool, the heterogeneity is achieved with Prometheus, an open-source tool that collects metrics from configured endpoints, stores them as time series and provides a functional query language to select, aggregate and filter data. The default Prometheus deployment, as shown in *Figure 15*, also provides its own alerting system to set limits over monitored metrics and handle alert notifications.

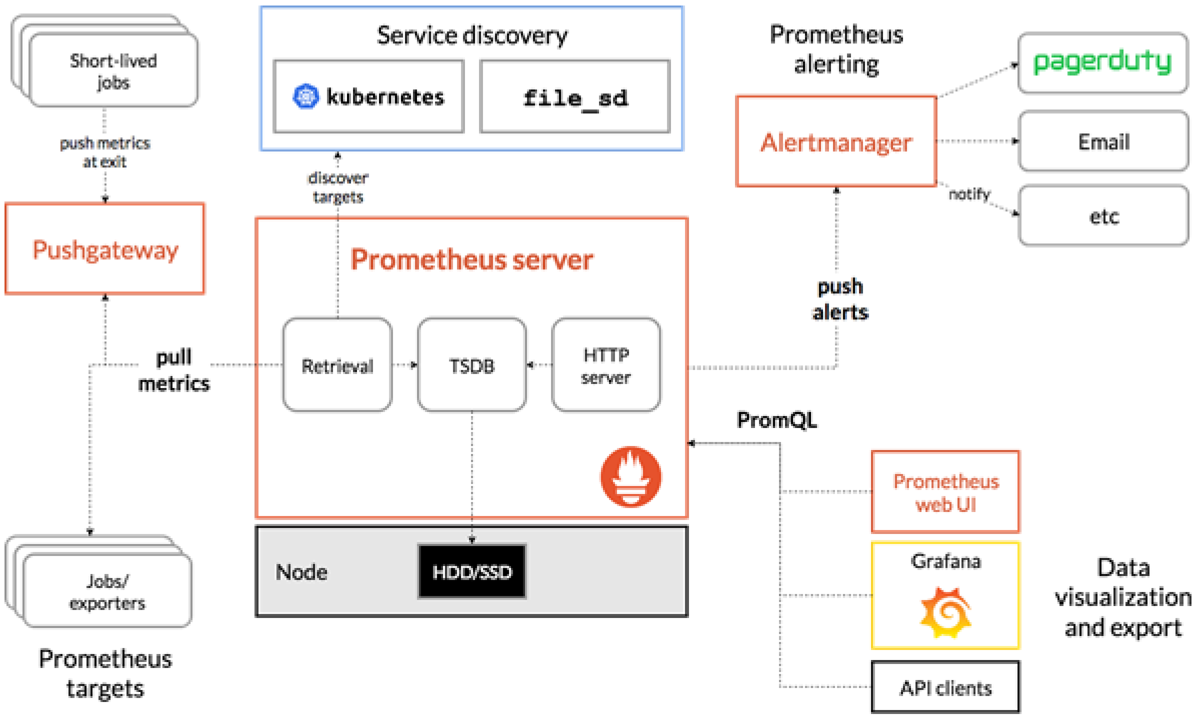


Figure 15: Prometheus architecture (Prometheus.io)

The pull mechanism is another of the strong points of Prometheus, it scrapes data from endpoints and limits the push mechanism to short-lived jobs that are not always available. In an ecosystem with so many microservices deployed and in so many places, the tool could become a bottleneck, but pulling metrics allows backward error traceability against network failures by being able to check them from the origin.

Figure 16 shows the monitoring architecture of CHARITY, where the multicloud and agnosticity premises are fulfilled by adding Thanos, a storage Prometheus setup that collects data from the Prometheus server deployed in each cluster. Thanos collects data from servers outside the cluster where is deployed by using sidecars, components deployed along with Prometheus that collect data from the monitoring server and share them with the central data storage.

Prevention and reaction are the most complex objectives to reach in a highly heterogeneous ecosystem like CHARITY. The adaptative network requires the minimum human involvement given the difficulty of locating problems in such a complex architecture. For this, a new tool has been added to convert the static configurations of Prometheus into a dynamic system capable of set new endpoints to scrape. The so-called Monitoring Manager works with the high-level orchestrator and adapts the configuration of the Prometheus servers deployed in each cluster using a template repository to automatize all the changes in the monitoring system and applying them to the server. The functionalities of the Monitoring Manager, described in deliverable D3.2, are available through an HTTP API.

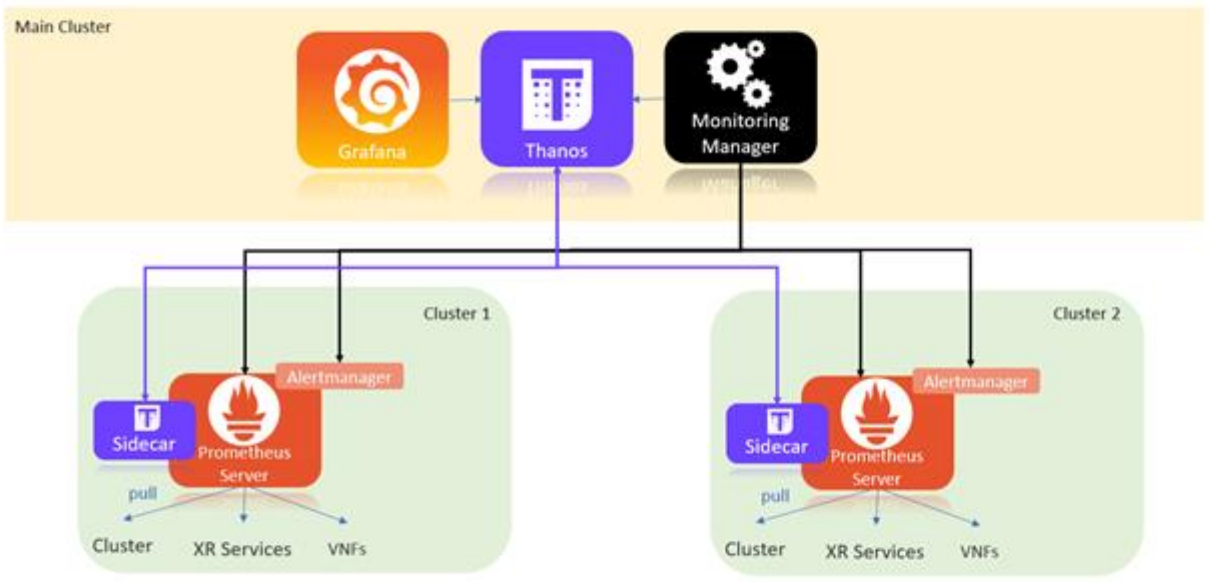


Figure 16: CHARITY monitoring architecture.

Prevention is also granted by Prometheus Alertmanager, that triggers alerts according to the performance thresholds defined by UC owners in the AMF. Migrating a service susceptible of failure is the reactive mechanism allows to guarantee XR extreme performance requirements. In CHARITY, this protection is at two levels: current performance and predicted performance. The first one is triggered by the current performance of the monitored component, which means that the migration it triggers aims to correct a certain condition, like exceeding a limit value established by the UC owner. The second one by forecasting predictions, where the migration does not seek to reverse an existing situation, rather to prevent it from occurring. The notification types are described in deliverable D3.2.

Grafana is the visualization tool that completes the monitoring framework, given its extensive use in the open-source community. Grafana allows to use both Thanos and Prometheus as data sources, provides powerful functions to create dashboards and multiple panels to display data in the most convenient way for engineers in charge of system observability.

4.3.2 Network and Infrastructure Data Monitoring

The monitoring framework, as illustrated in *Figure 17*, provides data to the Data Analytics Engine, the High-Level Orchestrator and the Resource Indexing. The first one requests time series of monitored data to perform predictions, the other two use monitored and predicted data to take deployment decisions regarding the resources available. All the data consumed by these components is exposed by Prometheus exporters.

Since Prometheus is one of the leading monitoring solutions, many third-party systems offer custom Prometheus exporters along with their tools. Many other exporters have been developed by the large and proactive community behind Prometheus that enriches the project. These exporters are tools that collect, process and expose metrics, that are available to Prometheus through an HTTP endpoint. The configuration with the endpoints to scrape is specified in a ConfigMap, that modify the list of targets monitored by Prometheus, as shown in *Figure 18*. The metrics consumed by the previously mentioned components are extracted from two tools: cAdvisor and Liqo exporter.

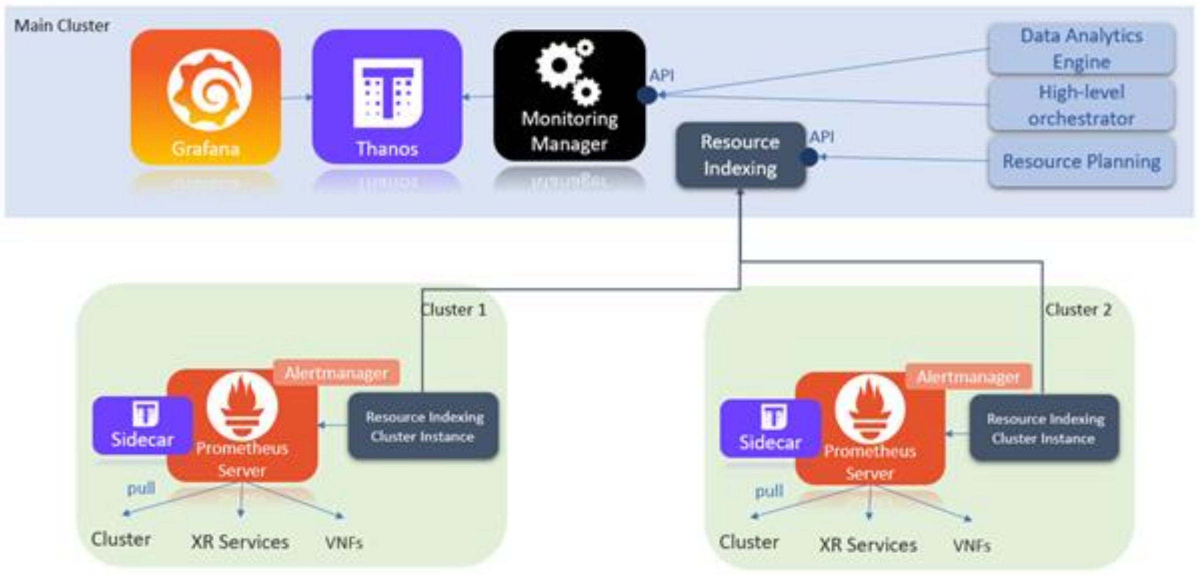
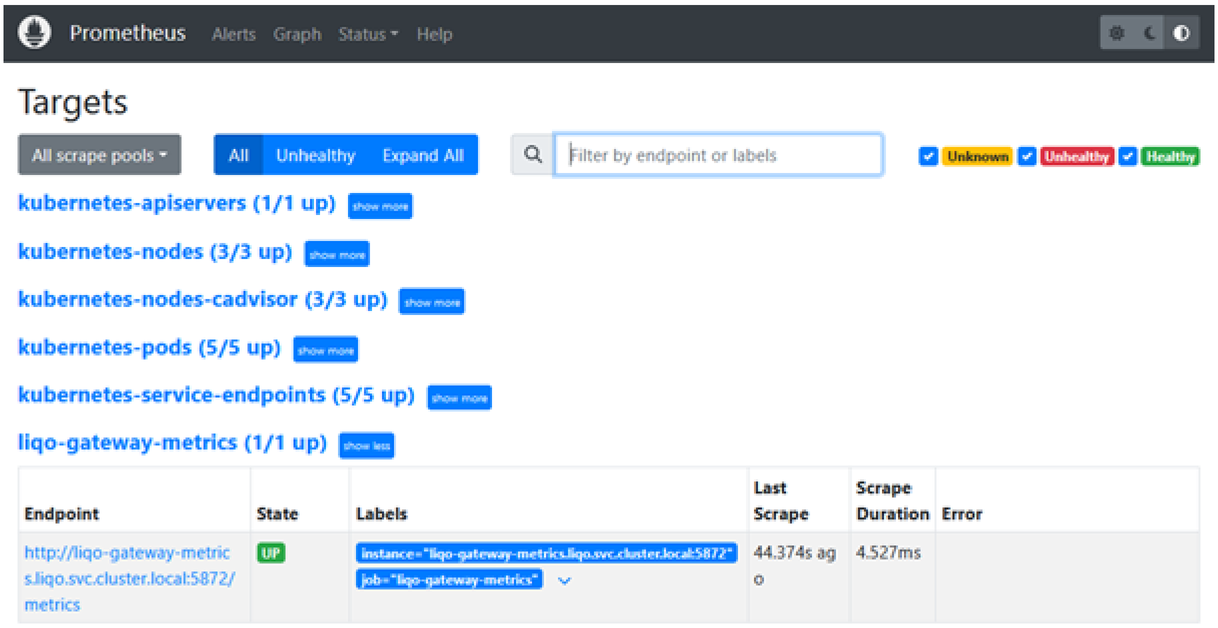


Figure 17: Monitoring framework integration with other components of CHARITY.



Targets

kubernetes-apiservers (1/1 up) show more

kubernetes-nodes (3/3 up) show more

kubernetes-nodes-cadvisor (3/3 up) show more

kubernetes-pods (5/5 up) show more

kubernetes-service-endpoints (5/5 up) show more

liqo-gateway-metrics (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://liqo-gateway-metrics.liqo.svc.cluster.local:5872/metrics	UP	instance="liqo-gateway-metrics.liqo.svc.cluster.local:5872" job="liqo-gateway-metrics"	44.374s ago	4.527ms	

Figure 18: List of targets monitored by Prometheus.

cAdvisor, or Container Advisor, is the tool developed by Google to monitor container performance. The monitoring framework uses the metrics of *Table 5* to monitor cluster CPU, memory and storage, values used to perform predictions and provide the cluster status to the Resource Planning. Regarding network monitoring metrics, Liqo[20], the tool that manages cluster interconectivity, exposes its own metrics regarding links status, traffic and latency.

Table 5: Prometheus metrics used in CHARITY.

Metric	Definition	Exporter
machine_cpu_cores	Number of logical CPU cores	cAdvisor
machine_memory_bytes	Amount of memory installed on the machine	cAdvisor

[20] https://docs.liqo.io/en/stable/usage/prometheus-metrics.html

container_memory_working_set_bytes	Current working set in bytes	cAdvisor
container_cpu_usage_seconds_total	Cumulative cpu time consumed in seconds	cAdvisor
container_fs_usage_bytes	Number of bytes that are consumed by the container on this filesystem	cAdvisor
container_fs_limit_bytes	Number of bytes that can be consumed by the container on this filesystem	cAdvisor
liqo_peer_is_connected	boolean keeping the status of the network interconnection between clusters	Liqo
liqo_peer_latency_us	the round-trip (RTT) latency between the local cluster and a remote cluster	Liqo
liqo_peer_transmit_bytes_total	the total number of bytes transmitted to a remote cluster.	Liqo
liqo_peer_receive_bytes_total	the total number of bytes received from a remote cluster	Liqo

4.3.3 XR Service Monitoring

The infrastructure metrics described in the previous section do not require UC involvement to perform monitoring, the observability is over the containers that are part of the application architecture. However, security restrictions to guarantee the integrity of the data managed and user privacy limit to the edge the components that can be monitored by Prometheus. To monitor final user devices is necessary to add an intermediated trusted proxy that collects and expose data.

The metrics that involve XR devices are treated as custom metrics. The strategy is the same as in Prometheus exporters: collect data, convert it to Prometheus readable format and expose it through an endpoint within the UC cluster. This additional metrics are configured as infrastructure ones through the AMF, including the HTTP endpoint.

4.3.4 Aggregated Monitoring

CHARITY and its monitoring framework spreads across clouds, datacenters and clusters, offering to UCs the possibility to migrate its services between them to maintain the defined performance values. Therefore, an application deployed in CHARITY isn’t restricted to a certain cluster, it can go through different cluster during its lifecycle.

The Monitoring Manager component adds dynamism to Prometheus, but data gathered by the monitoring server is still limited to the cluster where is deployed. To solve this, Improbable[21] created Thanos[22], an open-source tool that gathers data from different Prometheus servers. Thanos extends Prometheus storage capacity, but also implements its query language to retrieve data. Consequently, the data returned when querying a certain metric show compatible results in the different clusters that make up CHARITY.

Aggregating monitoring data with Thanos allows us to follow the performance of a service through the different clusters to which it migrates, as shown in *Figure 19*. This is key to the forecasting algorithms, which receive reliable information that represents the fluctuation the metric has undergone so the prediction results are accurate.

[21] https://www.improbable.io/

[22] https://thanos.io

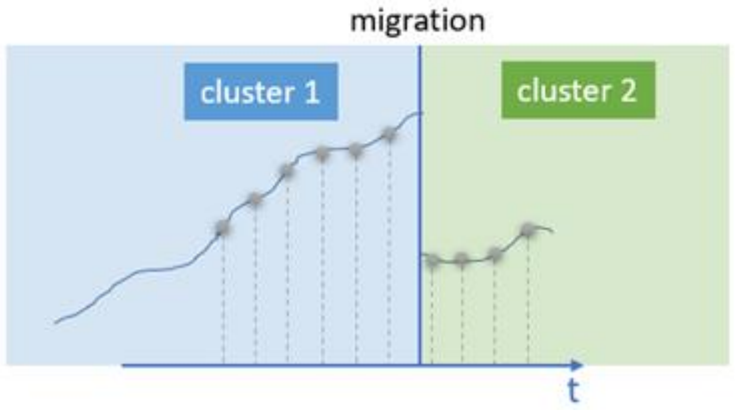


Figure 19: Example time series returned querying Thanos about a migrated service.

4.4 Forecasting Algorithms

4.4.1 Computation Utilization Forecasting

The efficiency of managing and orchestrating Edge & Cloud resources can be greatly enhanced by accurately predicting their perspective time-evolving resource utilization metrics. According to scientific literature, some metrics that can be employed in the prediction process are CPU, RAM, bandwidth and disk I/O. By implementing a predictive approach in regards to management and orchestration, it is possible to dynamically allocate cloud resources in a manner which is efficient in terms of resource utilization and is compliant with the constraints imposed by Service Level Agreements (SLAs).

The performance of applications is closely intertwined with the resources that they run on. Thus, it is of paramount importance for computational resources to operate within a specific range. This range is formulated in a manner which prevents resource under-utilization and over-utilization. In order to guarantee that resource utilization shall be kept in this specific range, it is possible to allocate additional computational resources in case of over-utilization or to deallocate some resources in case of under-utilization. This process is referred to as Horizontal scaling. In many cases, the process of deploying a virtual machine requires time in the order of several seconds which may make a reactive approach rather inefficient in terms of properly handling sudden bursts in resource usage. Fortunately, the use of predictive mechanisms solves this issue by providing predictions regarding the utilization that is expected to take place during the specific time-frame. These predictions are then leveraged in order to conduct proactive scaling. Another prominent functionality that benefits from utilizing a resource utilization prediction mechanism is task offloading. Task offloading is the process of choosing which computational resources shall handle specific tasks. By doing so, the overall workload can be distributed across the various resources in a manner which guarantees better response time. The selection process is based on the requirements of the tasks and the processing capabilities of the resources.

There have been numerous approaches in regards to how to properly predict resource utilization [9]. Some of these frameworks are based on the use of ARIMA models, such as the ARIMA-DEC [10]. which is VM provisioning technique based on load prediction. In [11], a proactive approach is proposed to cope with the dynamic resource provisioning which requires the use of the ARIMA model in order to perform predictions. Another approach is the use of recurrence-based neural networks such as the LSTM networks. In [12], an LSTM-based model is used in order to predict future CPU utilization. Furthermore, in [13], unidirectional and bidirectional multivariate LSTMs were used in order to forecast resource usage in Cloud datacenters. Contrary to these approaches examined thus far, there is also the choice of leveraging multi-step forecasting. In [14], an Encoder-Decoder network (GRUED) is used in order to perform sequence to sequence modelling. GRUED uses two Gated Recurrent Unit (GRU) networks which operate as a pair of GRU Encoder and GRU Decoder.

Computation Utilization Prediction can provide substantial benefits in terms of guaranteeing reduced latency and advanced fault tolerance. Violos et al. [15] showcases how Deep Learning (DL)-based Computation Utilization Prediction can be leveraged in the context of Horizontal Autoscaling in order to provide reduced latency. *Figure 20* shows the architecture of the proposed double tower neural network.

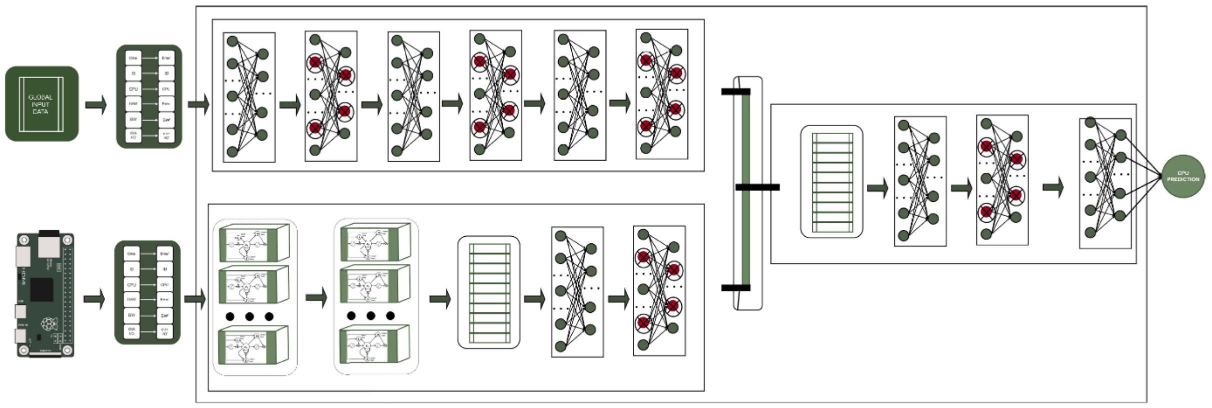


Figure 20: Architecture of the proposed Double Tower Neural Network [15].

The composite DL network is designed to satisfy the particularities of the Edge infrastructure and the resource usage metrics. Since resource metrics like CPU, RAM, disk, and bandwidth have sequential dependence, Recurrent Neural Network (RNN) can provide an appropriate type of neural layers. RNN combines the advantages of DL with the characteristics of time-series forecasting. There are different types of RNN architectures and the two most prominent are the GRU and LSTM approaches. Each individual processing node is examined separately for future resource usage. However, in order to trigger the node replication, the DL model of each node should be aware of its own status and the whole Edge infrastructure status. The Edge node which is examined is also called local and the whole Edge infrastructure is called global. Because the local and the global status affect each other we propose the use of a composite DL model that combines the two in order to provide the local resource utilization predictions. Details regarding the specifics of the proposed model can be found in [15].

In order to support these claims with regards to the efficiency of the proposed model, we performed a large-scale experimental evaluation in a simulated edge computing environment with CloudSim Plus. The simulation lasted one week and more than 1,500,000 tasks were generated and offloaded onto the edge processing nodes. The tasks were generated by a mixture of Gaussian probability distributions for the workload to simulate the working behaviour of employees who have peak of application requests at 11:00 am in the morning. The simulation begins with five running processing nodes and there are 15 more backup nodes for potential replication.

When a resource over-utilization is predicted, the scale up takes place proactively in order to keep the QoS in an acceptable range. We set the predictions of the proposed model to have a time frame of 10 minutes. In our experiments we compared the Reactive Horizontal Scaling Mechanism, the Kubernetes Horizontal Pod Autoscaller and the proposed IHPA. Hereunder and for the sake of brevity, we will call the first two as Reactive and Kubernetes. The experiments took place offloading the tasks with the MinMin, MaxMin and the RoundRobin (RR) task scheduling algorithms.

In the experimental evaluation, we see the accuracy of the double tower deep learning model, in terms of error metrics, compared with other baseline and state of the art prediction models. Next, we see the concrete outcomes of the proposed method in terms of Edge computing performance metrics i.e., execution time, throughput and the number of active resources per hour. We compare these outcomes with the reactive and the Kubernetes autoscaling methods for the three task offloading mechanisms.

The error metrics we use are the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE). The MAE expresses the average absolute difference between the target values and the predicted values. Squaring the prediction errors and averaging the squares we have the RMSE. RMSE expresses the standard deviation of the errors emphasizing on the spread out of the errors. MAE is preferred when all the errors have the same importance, while RMSE is prioritized when we should penalize the large errors even if they are just a few.

The execution time has been evaluated through different statistical measurements. Average Execution time (Avg. Ex. time) declares the mean time for all the tasks of the experiment. Median Ex. time refers to the middle values. Standard deviation declares how much the execution times of the tasks differ from the average value. Two additional statistical measures are the skewness and kurtosis. Skewness indicates the symmetry of the values in execution time and a right-skewed distribution is better than a left one. Kurtosis indicates if the distribution of the time values is heavy-tailed or light-tailed. An additional evaluation metric is the tail latency. Tail latency is the 98th percentile and represents the 2% longest response times in the system. It is an important metric because these longest responses affect in a significant way the SLAs and the QoS. Throughput declares the average number of tasks completed per second for all the processing edge nodes. Active VMs per hour refers to the number of VMs that are active and running per hour. This metrics is closely intertwined with the pricing that is charged by the infrastructure providers.

The Reactive Autoscaling mechanism decides every 60 seconds whether the network should allocate additional nodes, release some running nodes or continue with the same topology. The decision is being made reactively and independently of each node and is based on its average CPU utilization recorded during the last minute. The main objective is to ensure that each processing node operates in 40%-70% capacity in order to avoid under-provisioning and over-provisioning. If the current CPU utilization exceeds the 70% upper threshold the scaling mechanism in CloudSim Plus decides to allocate additional nodes. If the predicted value is below 40%, the scaling mechanism decides to release the under-utilization nodes after its running tasks have been completed. Because the scaling decisions take place after the resource metrics exceed the threshold, there will be a significant delay in the instantiation of the new nodes. As we will see in the next subsection, these delays regarding the deployment and startup time of the node will also affect the tasks execution times.

Each pod is a representation of a single instance of a running process in the Edge infrastructure and runs at least one container. In the CloudSim Plus experiments with the Kubernetes autoscaling mechanism, each pod was designed to contain a single container in order to facilitate a computational paradigm similar to the one used in the Reactive Horizontal Scaling Mechanism. By doing so there is a direct analogy formed between pods and network hosts with containers and VMs respectively. Firstly, the incorporation of the Intelligent Scaling Mechanism in the experiments requires the additional integration of the DL prediction model in CloudSim Plus. Next, in the same way with the previous autoscalers, once every 60 seconds information regarding each node is gathered. In addition, for each node the last 20 sampling metrics are stored in order to form the time-series of the local nodes. These sampling metrics are multivariate and include the following six features: VM ID, timestamp, CPU, RAM, Bandwidth utilization and the number of processed tasks that correspond to the last minute.

In the context of the experiments there are 20 local representation vectors, each one corresponding to a different node. By aggregating their values, we create the unique global representation vector. The global representation vector describes the Edge infrastructure in a single timestamp and consists of the metrics of all local nodes. Thus, once every 60 seconds, 20 local representation vectors and a single global representation vector are created. All the representation vectors bear the same dimensions (6*1). This fact allows the concatenation of each one of the local representation vectors with the global. The result is the creation of 20 hybrid representation vectors that are later used as input for the proposed model. The proposed produces 20 distinct predictions, each one describing the expected CPU utilization for its corresponding node. Similarly, to the scaling mechanisms described before, the Intelligent autoscaler also keeps the CPU utilization in the 40%-70% zone. The fundamental difference is that the Intelligent Scaling Mechanism utilizes the predicted values of CPU to make proactively the scaling decisions instead of the Reactive and Kubernetes autoscalers that utilize the current CPU metrics.

We make a comparison of the proposed Intelligent Proactive Autoscaling against the Reactive and the Kubernetes approach with different versions of the execution time, throughput and number of resources. The outcomes are summarized in *Table 6*.

Table 6: Experimental results of the proposed Intelligent Autoscaling method [15].

	Algorithm	Makespan	Tput	Tail latency	Avg. Ex. time	Std. Ex. time	Median Ex. time	Num. Tasks	Max. Ex. time	Skewness Ex. time	Kurtosis Ex. time
Min Min	Reactive	574801,29	16	32,4386	2,9982	12,8765	1,11	1532308	186,83	8,3002	75,2270
	Kubernetes	574796,76	3167	29,2900	2,8198	11,6182	1,11	1530509	172,16	8,2046	73,4336
	Intelligent	574797,15	**74443**	**5,7700**	**1,5704**	3,3223	1,11	1531764	**90,84**	12,6256	206,8956
Max Min	Reactive	574800,11	7087	103,8100	6,1388	22,9556	1,22	1531095	240,91	5,4973	32,3819
	Kubernetes	574795,22	6886	67,1358	5,7495	33,9438	1,11	1530972	576,74	10,1786	116,4899
	Intelligent	574796,17	**7466**	**3,8700**	**1,3819**	2,0919	1,11	1531331	**46,80**	9,6935	113,9497
RR	Reactive	574795,40	203	119,4900	7,4070	26,0314	1,24	1531237	251,19	5,0120	26,5954
	Kubernetes	574799,95	3902	75,8500	5,5398	29,3488	1,11	1529289	419,82	8,8722	87,9680
	Intelligent	574800,62	**7194**	**3,3100**	**1,4378**	2,2548	1,11	1530368	**49,72**	10,2477	124,7710

The Double Tower DL model that runs in the Intelligent Autoscaler excels at speeding up all tasks in general, something that can be seen at the results of the Average Execution Time. The improvement in the execution time means that tasks complete faster in general, regardless of their computational needs. Moreover, the significantly lower standard deviation shows that the Intelligent Autoscaler provides consistency in the execution time, reinforcing our claim of fewer outliers of delayed tasks. This adds predictability to our system and enhances our abilities of being in control of the resources. The tail latency metric also highlights the IHPA's efficiency, as it was improved by a factor of 500-3600% depending on the task offloading algorithm. That implies a vast improvement on the computationally heavier tasks, which can really affect user experience. This claim can be also supported further by evaluating the maximum Execution Time of the methods tested, where the proposed method exhibits similar results. By dropping said metric to at least roughly half of the Reactive/Kubernetes values, one can safely assume that the proposed method handles the available resources in a way which ensures maximum availability for longer tasks. There would be no point in examining the results in regards to throughput in the context of the entirety of the simulation, since all tasks would have eventually been completed successfully by the end of it. So, we chose to examine the throughput that is achieved when the sudden bursts in task production take place. More specifically, the throughput metric corresponds to the number of tasks completed during the three minutes time-slots when the ten most sudden and violent bursts in task production took place. As one can see in *Table 6*, the proposed algorithm managed to greatly improve this specific type of Throughput (Tput). It is worth mentioning that the results which correspond to the Round Robbin Reactive mechanism and the MaxMin Reactive mechanism are indicative of their inability to handle sudden bursts in task production, during which the use of these Horizontal Scaling mechanisms led to extended system failures. Our proposed model, on the other hand, managed to not only ensure that the task processing operations will remain unaffected by changes in the rate of task production, but to also enable these operations to be conducted in an optimal manner. In addition to the above, *Table 6* provides metrics such as skewness and kurtosis. Skewness shows us that the main part of the execution time distribution values resides on the left part, which translates to lower execution times, whereas kurtosis implies that those values are concentrated on the smaller central part of the distribution, resulting in a steeper "bell". Ultimately, these metrics inform us that the tasks are completed faster, and at a relatively steadier pace. We can arrive at this same conclusion by using the average value and the standard deviation of execution time as well. The average number of VMs that were used during the entirety of the simulations remained relatively the same across all the Horizontal Scaling Mechanisms that we examined, with a slight deviation of about 5% of the total number of VMs per hour. Even if the Intelligent method is slightly resource heavier than the reactive and Kubernetes ones due to the proactive autoscaling, we see that it elastically scales down fast enough once it understands that there are resources that might go unused in the immediate future.

In regards to Computation Utilization Prediction being able to provide advanced Fault Tolerance capabilities, we produced the "Intelligent Proactive Fault Tolerance at the Edge through Resource Usage Prediction". The composite DL network is proposed to provide accurate resource utilization predictions for the Intelligent Proactive Fault Tolerance (IPFT) method.

The composite DL model was integrated in an Edge simulation of CloudSim Plus. We simulated an Edge infrastructure that consists of a set of nodes, 5 available to us by default, and another 15 that can be activated for intelligent replication when needed. We simulated the local and global resource monitoring process, measuring CPU, RAM and bandwidth values, and saving those values every 60 seconds (time-step). The task offloader of the infrastructure was receiving incoming traffic and was assigning each task on a node, based on the following scheduling algorithms: RoundRobin, MinMin, and MaxMin.

The local and global resource usage metrics are being monitored and then fed to the IPFT mechanisms of each processing node. During every single time-step we use the monitoring data in order to formulate the appropriate data representations, featuring the past time-series measurements of a single node, as well as the state of the infrastructure as a whole. The input is then fed to the composite DL model, enabling it to make predictions of resource usage for every node in a time horizon of 10 minutes. In this experimental set up we made the assumption that the preparation time for the infrastructure to assure its availability and robustness to faults is 10 minutes.

The simulation lasted for seven days and the tasks were generated by a mixture of Gaussian probability distributions that simulate a realistic application workload behaviour. The processing nodes simulated the processing capabilities of Raspberry Pis. We defined a process fault if the time execution of a task lasts more than one second. The selection of one second is a reasonably acceptable latency for several data analytic applications. Trying different latency times for the process faults, we noticed that the IPFT performance was in a similar way better than the reactive approach. In the reactive Fault Tolerance approach, a node replication is triggered in case of a fault is taking place. In *Table 7*, as we will thoroughly discuss in the next section, we compare the IPFT mechanism to the reactive FT approach.

In order to evaluate the performance of the IPFT mechanisms, we used a set of fault tolerance evaluation metrics. Mean Time To Failure (MTTF) is defined as the expected time to failure given that the system functions properly. MTTF is an evaluation metric which corresponds to the overall inability of the Edge infrastructure to operate properly and thus, it is calculated by taking into consideration the number of faults regardless of the actual processing node that failed. Mean Time To Repair (MTTR) is defined as the expected time required to repair the system after a failure occurs. For the MTTF the higher values are the better and for MTTR the lower values are the better. These evaluation metrics are calculated in terms of seconds.

Two additional Fault Tolerance evaluation metrics are the Reliability and Maintainability. Reliability refers to the ability of an Edge infrastructure to run continuously without any failure. Maintainability refers to how easily a failed system can be repaired. Both Reliability and Maintainability are numbers with no units and higher values mean better performance.

Table 7: Experimental results of the proposed IPFT.

	MTTF	MTTR	Reliability	Maintainability
RFT RR	2.864	19.657	0.741	0.048
IPFT RR	9.506	3.343	0.904	0.230
RFT MinMin	8.733	36.169	0.897	0.026
IPFT MinMin	8.919	5.656	0.899	0.150
RFT MaxMin	3.721	24.239	0.788	0.039
IPFT MaxMin	13.309	7.425	0.930	0.118

The experimental results are summarized in *Table 7*. We compared the IPFT mechanism to the Reactive Fault Tolerance (RFT) approach. The RFT approach performs node replications after a fault occurs. Regarding the task offloading algorithm we used the Round Robin (RR), the MinMin and MaxMin. The experimental result shows the superiority of IPFT compared to the RFT in all evaluation metrics. In addition, we see that the outcomes are significantly affected from the task offloading mechanism. This happens because the task offloading algorithms also integrate a workload balancing methodology with different criteria as we discuss in the following paragraphs.

In *Table 7*, we can see that in RR, MaxMin and MinMin the MTTF in IPFT has been increased compared to the RFT. This means that leveraging the resource usage predictions, faults occur more sparsely and rarely. We can see from the MTTR metric that in the event of a fault, the infrastructure will recover very quickly, scheduling the new tasks in processing nodes with low resource utilization. The reliability metric shows that by using the IPFT, the Edge infrastructure can provide the expected results up to 93% of the simulation length, even during the stressing time periods of the simulated days. The significant improvement noticed for the Maintainability metric, declares that even if a fault occurs, the IPFT will increase the robustness of the Edge infrastructure. In other words, the IPFT will take timely the right measures by triggering node replication and task migration, in order to reduce the likelihood of subsequent fault occurrence.

A fault is recorded taking into consideration all the Edge nodes that are currently active. This means that the MTTF value of 13.309 seconds in IPFT MaxMin includes the faults of different Edge nodes. In addition to that, some generated tasks had a large number of million instructions that would have provoked a fault because of the CPU unavailability in the processing nodes. In this case, we wanted to know how these tasks affect the MTTF and MTTR. From the analysis of the results, we saw that the variance of the task size is the reason that we see that the three different task offloading mechanism have different performance. In particular, the MaxMin algorithm gives higher priority in big tasks, thus we see a significantly better MTTF metric.

During the simulation we examined the IPFT decisions and how the Edge environment operates. The simulation confirmed that the infrastructure takes advantage of the timely decision to trigger proactive actions, such as intelligent node replication and task migration before the number of tasks overwhelms the processing nodes. This can be particularly important for the infrastructure provider as it can save cost and energy, by shutting down nodes when they are no longer needed. Additionally, the provider can achieve a smoother flow of on-time completed tasks, avoiding crashes and minimizing QoS deterioration.

While the aforementioned solutions provided exceptional results in terms of forecasting within the frame of single resource consumption metric at a time, when the need to conduct predictions that involve multiple metrics, they presented certain limitations. Towards tackling these limitations, members of the CHARITY project investigated the use of novel Deep Learning Encoder - Decoder architectures that are based on Graph Neural Networks. One of these architectures (GCN-LSTM ED) is showcase in *Figure 21*. The showcased forecasting model architecture is capable of providing multi-step multi-variate forecasting regarding CPU, Memory, and Network metrics. More details regarding the proposed architecture can be referred to [16].

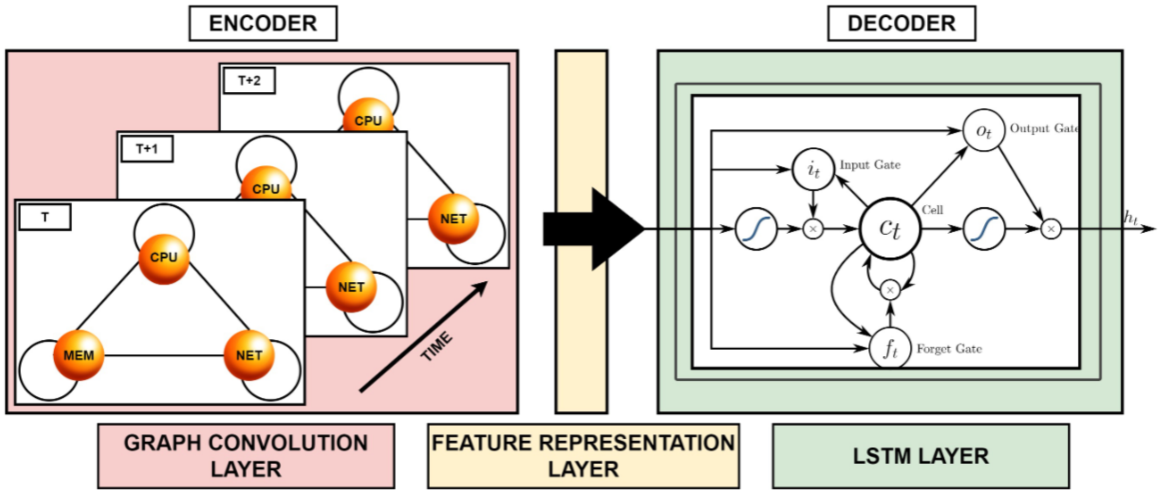


Figure 21: Architecture of the proposed GCN-LSTM.

Figure 22 showcases the experimental results that examine the forecasting efficiency of the proposed solution. As one can see, the proposed solution (GCN-LSTM ED) managed to outperform its competitors in terms of multi-variate multi-step forecasting.

	RMSE	MAE
LSTM	6.007	5.049
BD-LSTM	6.735	5.300
GRU	6.507	5.129
LSTM ED	6.239	4.908
BD-LSTM ED	6.157	4.964
HYBRID LSTM ED	6.040	4.755
HYBRID LSTM ATT ED	6.134	4.952
CNN-LSTM ED	6.729	5.288
GCN-LSTM ED	**5.922**	**4.665**

Figure 22: Experimental Results of the proposed GCN-LSTM (prediction accuracy).

Furthermore, we examined the efficiency of the proposed solution within the frame of a simulation for proactive horizontal autoscaling using the aforementioned CloudSim Plus framework. The settings of the experiments were similar to the ones that were previously explored. The corresponding experimental results are showcased in *Figure 23*. As one can see the proactive intelligent approach that leverages the proposed GCN-LSTM ED manages to outperform the standard reactive approach.

Autoscaling Method	Tail latency	Avg. Ex. time	Std. Ex. time	Median Ex. time	Num. Tasks	Max. Ex. time	Skewness Ex. time	Kurtosis Ex. time
Reactive	5.610	1.767	1.944	1.539	1624817	47.849	9.385	129.665
Intelligent	**5.060**	**1.525**	**1.123**	**1.260**	1624735	**18.909**	**3.513**	**20.166**

Figure 23: Experimental Results of the proposed GCN-LSTM (horizontal autoscaling).

4.4.2 Network State Forecasting

Network state prediction is vital for optimally managing the computational, storage and network resources that the various services run on. The network state corresponds to the amount of network traffic in relation to the various nodes. The term of traffic in the context of Edge and Cloud computing has two different interpretations. The first one refers to the amount of data which is traversing the network infrastructure. The second one refers to the amount of user requests / sessions conducted. The metadata corresponding to both traffic interpretations can be collected in the transport layer of the Transmission Control Protocol/Internet Protocol (TCP/IP) suite by using a traceroute network diagnostic tool.

The service traffic prediction has a long history dating back to the 1990s. For many years, different methods have been used for modelling and forecasting the service traffic. In the beginning, point process statistical models like Poisson processes were used but they presented the limitation that they do not capture the self-similarity characteristic [17] of the sequence values. Afterwards, time-series models such as Autoregressive-Moving-Average (ARMA) and their variations Autoregressive Integrated Moving Average (ARIMA) and Seasonal ARIMA (SARIMA) [18] were used for traffic prediction and managed to minimize the operation cost taking into account two types of cost: i) the cloud resource costs which occur when non-essential resource provisioning is performed due to traffic overestimation and ii) the QoS degradation cost which occurs when the traffic is underestimated, resulting to fewer resources than actually needed being allocated and thus jeopardizing the satisfaction of the users of the data services.

With the advent of Deep Learning, many decision-making models after being experimentally compared and redesigned, were ultimately replaced by Artificial Neural Network (ANN). The first studies showed that ARIMA performs better than simple feed-forward ANN [19]. The reason is that simple feed-forward ANN approaches are not designed for sequential tasks. They allow information to travel one way and cannot capture the periodic and autocorrelation patterns that characterize network traffic. Recurrent Neural Networks (RNNs) are a different class of ANN that models temporal sequencing of data so that each observation is dependent on the previous ones running in both directions by loops in their network. Information derived from earlier input is fed back into the network providing a kind of memory of the previous observation sequences in order to predict the next one. Complex RNN models that leverage interactive and temporal behaviour of data centres have been used successfully for single-service traffic prediction and interactive network traffic prediction [20].

Many data transfer, storage and processing services include short- and long-range time dependencies, making the multi-step prediction a prominent solution [21]. Multi-step prediction using RNN with iterated prediction over many time steps has been applied for IoT traffic time-series prediction [22]. This approach is based on the assumption that for each prediction step, the output of the RNN is merged with the newer input in order to make the next step prediction. The limitation is that this feedback approach is not directly designed for sequence prediction and as a result tends to accumulates errors over steps. Contemporary cloud resource management mechanisms can provide resource allocation policies by leveraging multi-step traffic prediction.

A sequence to sequence (seq2seq) architecture can capture the temporal dependencies and provide predictions for different time steps. A prominent approach for seq2seq is the encoder-decoder which consists of one neural network that maps the input sequence of previous steps to an intermediate vector and the decoder which maps the intermediate vector to a sequence prediction. Encoder-decoders have been used in multiple fields for multi-step prediction but up until now they had not been used in service traffic prediction. *Figure 24* illustrates a contemporary service chain scenario. The traffic monitoring tool provides the current traffic values to the encoder-decoder, which then outputs the traffic prediction sequence. The traffic prediction sequence is being leveraged by the Intelligent Network Function Resource Allocation to provide the necessary resources on the fly, thus keeping the fulfilment of QoS requirements at acceptable levels. The Intelligent Network Function Resource Allocation mechanism performs horizontal or vertical scaling, by dynamically allocating resources to keep up with the data-flows of the next time periods.

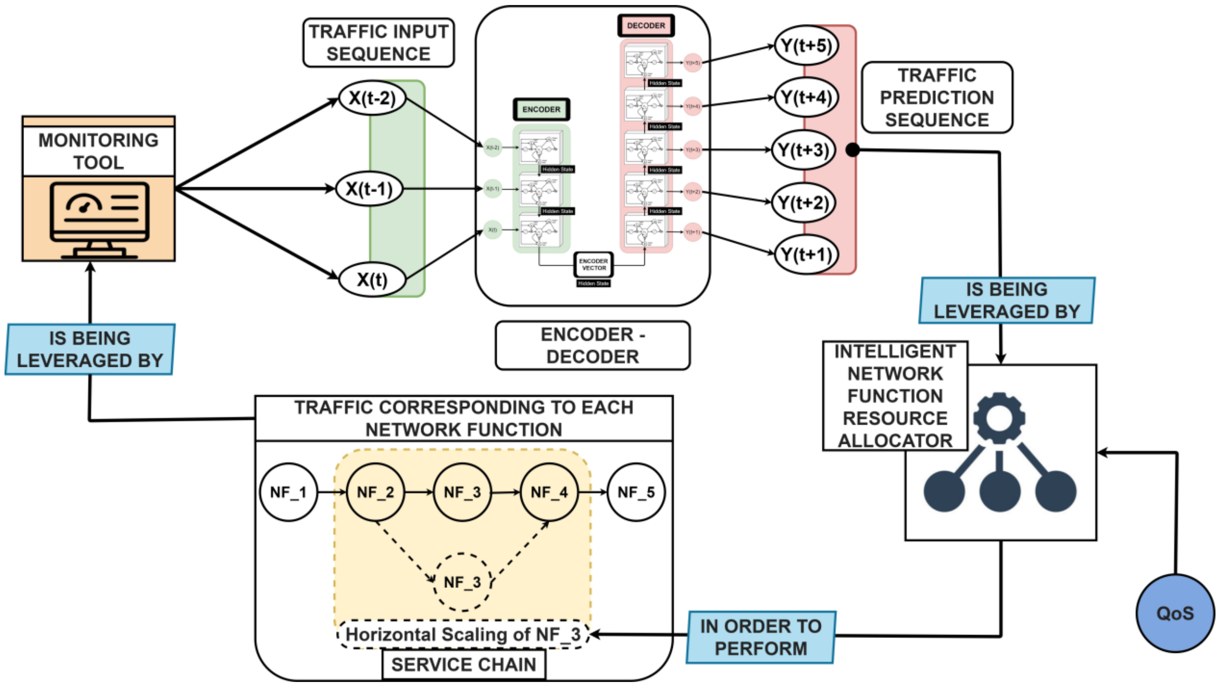


Figure 24: Leveraging Service Traffic Prediction for Horizontal Scaling of Network Functions [23][23].

CHARITY aims to leverage the Encoder-Decoder paradigm in order to establish robust multi-step traffic prediction mechanisms. What makes encoder-decoder models an ideal candidate for sequence-to-sequence prediction is their inherent ability to map sequences of different lengths to each other. This functionality is the result of the model's architecture. The encoder takes the input sequence and represents the information as latent variables. The decoder is set to the final states of the encoder and is trained to generate the output based on the information gathered by the encoder.

Furthermore, CHARITY shall introduce a novel Hybrid architectural paradigm which is the result of using both of bidirectional and unidirectional LSTMs instead of just one of the two. The input layer is a bidirectional LSTM. A unidirectional LSTM layer is then stacked on top of the bidirectional one. The bidirectional layer will provide one hidden state output for each time-step in 3-dimensional form which is then used as input by the unidirectional layer. The core idea behind this architectural choice is the fact that by introducing heterogeneous layers the model will be able to exploit the temporal correlations present in the various time-series in a more sophisticated way when compared to the rest of the models. Furthermore, the fact that multiple layers are being used allows the features of the input sequence to be represented in a more robust way. The same design logic is implemented in the decoder part in order to mirror the encoder morphology. Instead of the basic LSTM model used in the previously explored decoders, the Hybrid model uses a bidirectional layer stacked on top of a unidirectional layer. This structural symmetry enables the decoder to properly reconstruct the underlying temporal motifs of the input sequence.

In order to evaluate the accuracy of the proposed model we used error metrics and time metric. The error metrics are the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE). The MAE expresses the average absolute difference between the target values and the predicted values. Squaring the prediction errors and averaging the squares we have the RMSE which expresses the standard deviation of the errors emphasizing on the spread out of the errors. MAE is preferred when all the errors have the same importance, while RMSE is preferred when we should penalize the large errors even if they are just a few. In our experiments, the amount of transmitted data was in terms of megabytes and the duration of the services was in seconds. In *Table 8*, *Table 9*, and *Table 10*, we evaluated each forecasted time step independently in order to see the models prediction skill at each specific time-step and to contrast models based on their accuracy at different time-steps.

Table 8: Request number [23].

Requests	1st step		2nd step		3rd step		4th step		5th step	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
LSTM Vec	22.794	**15.704**	25.808	**17.885**	26.637	**18.956**	27.539	**19.378**	28.249	**20.066**
ED LSTM	22.675	16.147	25.807	18.422	27.119	19.454	27.901	20.157	29.071	20.705
ED CNN-LSTM	23.348	16.584	25.877	18.311	26.906	19.282	28.006	20.063	28.873	20.749
ED Bid-LSTM	22.827	16.071	26.241	18.615	27.191	19.275	28.311	20.401	29.102	21.177
ED Hybrid	**22.656**	16.173	**25.495**	18.145	**26.489**	19.098	**27.504**	19.754	**28.114**	20.268

Table 9: Transmitted data [23].

Traffic	1st step		2nd step		3rd step		4th step		5th step	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
LSTM Vec	781968	605345	798364	628584	832546	652228	846999	**660414**	862397	**669140**
ED LSTM	770830	**581789**	801711	**614400**	845053	**649910**	876312	672496	889506	680190
ED CNN-LSTM	801861	615874	815760	631545	843934	657737	864654	671775	867361	671956
ED Bid-LSTM	785167	595622	824090	632736	855272	658113	878195	673668	880516	672885
ED Hybrid	**757915**	601998	**788857**	632948	**812605**	651148	**831969**	666148	**843414**	673666

Table 10: Session duration [23].

Duration	1st step		2nd step		3rd step		4th step		5th step	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
LSTM Vec	43604	11091	43085	10710	43105	10924	43105	10924	43437	11078
ED LSTM	44816	11377	44473	11354	44518	11407	44723	11398	44740	11416
ED CNN-LSTM	43232	10591	42702	10139	42409	**10018**	**42142**	**9697**	**42012**	**9701**
ED Bid-LSTM	**42407**	**10125**	42702	10152	42594	10125	42434	9983	42380	9814
ED Hybrid	42437	10179	**42324**	**10126**	**42363**	10212	42429	10161	42425	10184

Extended information regarding the architecture of the Hybrid Encoder-Decoder and the overall experimental evaluation process can be found in [23]. The fact that the Hybrid model utilizes a greater number of layers allows it to better encapsulate the signal's characteristics when compared to the other models. This effect is amplified by the fact that the Hybrid model consists of heterogeneous layers (bidirectional and unidirectional) which allows the encapsulation of temporal motifs in a more robust manner. This claim is supported by the fact that the Hybrid model produces the best RMSE scores in regards to traffic and number of requests. On the other hand, the best MAE scores in regards to traffic and number of requests were produced by various LSTM-based models whose simpler and more shallow architecture enabled them to tune into the fundamental oscillation of the time-series. Yet, the Hybrid model was able to follow the signal more accurately by being able to produce predictions closer to the actual values.

Similar to resource usage predictions that were previously explored, while the previously mentioned solutions yielded impressive results in forecasting in the frame of a singular service, they encountered limitations when extending predictions to involve multiple services. In addressing these challenges, researchers involved in the CHARITY project explored the utilization of innovative Deep Learning Encoder-Decoder architectures grounded in Graph Neural Networks. *Figure 25* showcases one such architecture, known as GCN-LSTM ED. This highlighted forecasting model architecture demonstrates the capability to forecast service demand that corresponds to multiple services. Further insights into the proposed architecture can be found in the respective journal publication.

Figure 26 showcases the experimental results that examine the forecasting efficiency of the proposed solution. As one can see, the proposed solution (GCN-LSTM ED) managed to outperform its competitors in terms of multi-service multi-step forecasting.

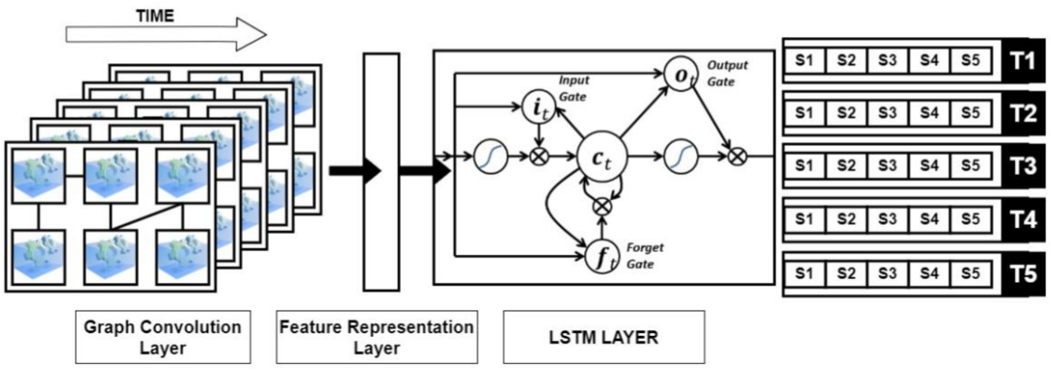
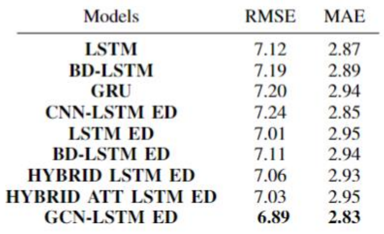


Figure 25: Architecture of the proposed GCN-LSTM.



Models	RMSE	MAE
LSTM	7.12	2.87
BD-LSTM	7.19	2.89
GRU	7.20	2.94
CNN-LSTM ED	7.24	2.85
LSTM ED	7.01	2.95
BD-LSTM ED	7.11	2.94
HYBRID LSTM ED	7.06	2.93
HYBRID ATT LSTM ED	7.03	2.95
GCN-LSTM ED	**6.89**	**2.83**

Figure 26: Experimental Results of the proposed GCN-LSTM (prediction accuracy).

4.4.3 SLA Violation Forecasting

Crucial to ensuring the agreed service standards are Service Level Agreements (SLAs), which delineate the metrics against which service is measured, as well as the corrective measures or sanctions in case the agreed service levels are not achieved. The SLAs constitute the key documents between the service provider and the consumer, clarifying the quality and terms of the services agreed between the two parties for a specific time period. SLAs serve as regulatory procedures, establishing trust between the parties involved and specifying various guarantees that providers are required to offer to their customers. The agreements may encompass a wide range of Quality of Service (QoS) metrics that the provider aims to guarantee, including response time, throughput, availability, mean time between failures, or energy consumption, among others. In many cases, multiple of these QoS metrics are combined to form Service Level Objectives (SLOs). Additionally, SLAs detail corresponding service pricing and the penalties imposed in cases where providers are unable to offer the agreed-upon QoS. Furthermore, SLAs cover aspects such as service guarantee time period, credit, exclusions, and granularity, with the latter referring to the resource scale specified in the service guarantee, which may correspond to data centre, service, instance, or transaction level.

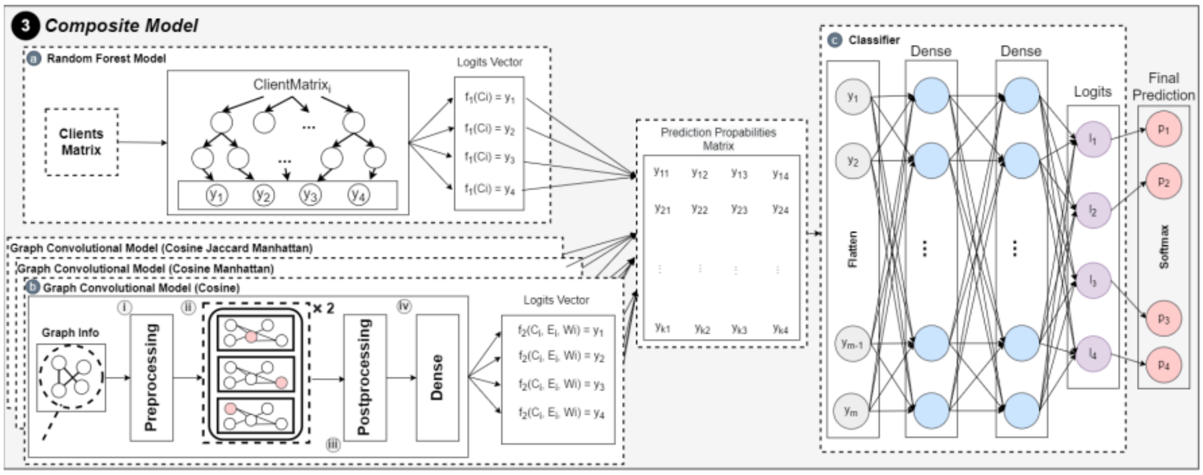


Figure 27: Architecture of the proposed Composite Model [24].

For service providers, minimizing instances of SLA violations is vital to enhancing customer satisfaction, avoiding financial penalties, and preserving their reputation. To prevent service violations and their consequences, service providers must establish mechanisms for monitoring and predicting QoS parameters. In cases of severe indications of violations, they must take immediate and effective measures to prevent them. Therefore, effective prediction methods are necessary for service providers to anticipate potential violations before they occur, enabling proactive countermeasures. Various prediction techniques are employed for predicting SLA violations, specifically for forecasting QoS parameters in future time intervals. However, the configuration of each technique varies based on several aspects, such as the choice of the prediction method, input formulation, and data patterns.

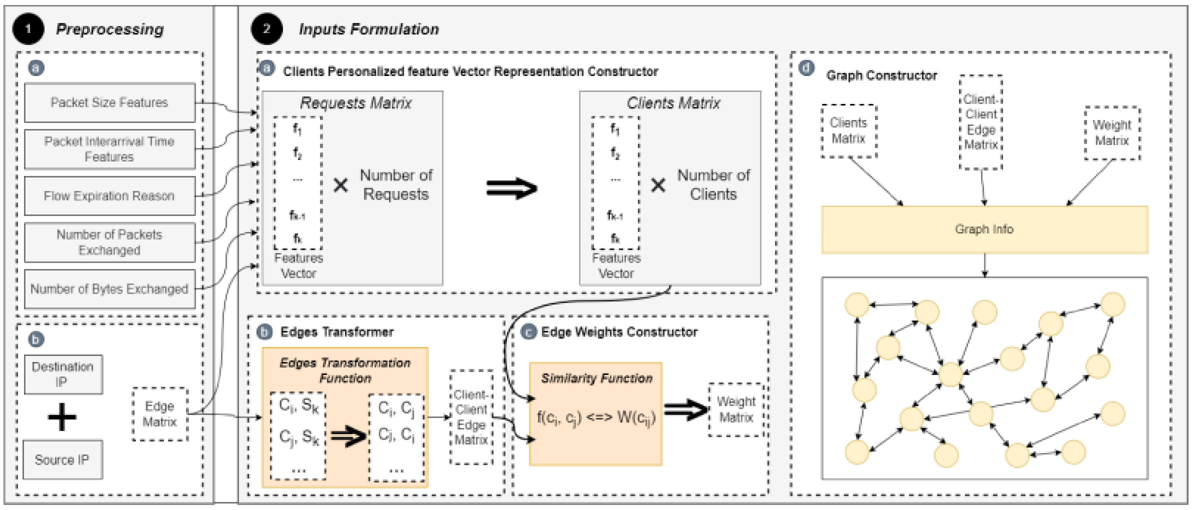
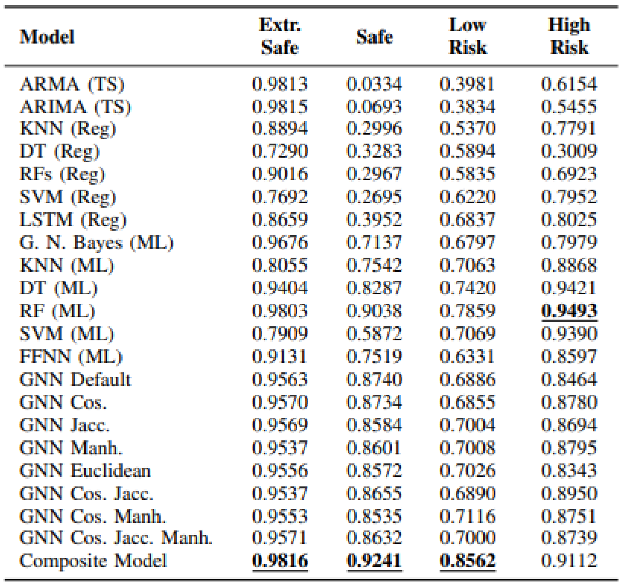


Figure 28: Required input preprocessing and formulation.

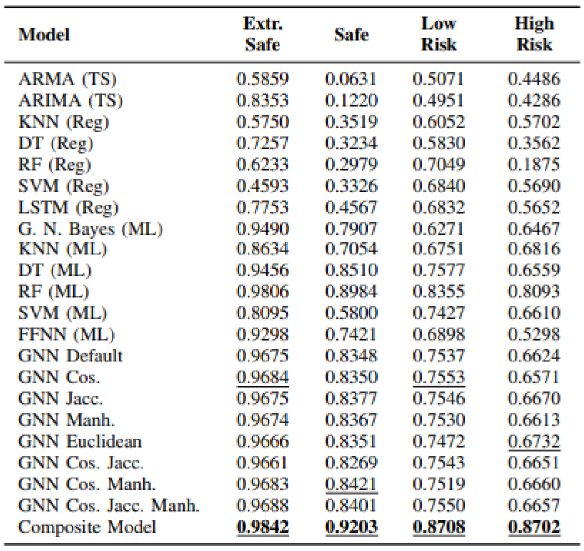
We explored various methods for predicting SLA violations occurring during the service provisioning between cloud customers and providers. While there are numerous network metrics pertaining to server-client interactions, the effective utilization of these metrics by an SLA prediction mechanism remains an open research question. We investigate three distinct data representation models for network characteristics: time series, content, and context representations. We find that a context-based approach utilizing graph representations adeptly captures client associativity, thereby enhancing the performance of conventional SLA violation prediction models when integrated with them. Our SLA violation prediction employs neural networks, leading us to propose a composite model leveraging Graph Neural Networks (GNNs). In our investigation, we place particular emphasis on constructing graphs in various ways. We conduct a comprehensive performance evaluation of 23 SLA prediction models categorized into three representation types: vector models based on network features, sequential models exploiting temporal QoS metric evolution, and graph models considering client associativity.

The architecture of the proposed composite model is displayed in *Figure 28*. In order to provide the appropriate input to the proposed composite model it is required to follow the two steps that are displayed in *Figure 28*. Experimental results demonstrate that our GNN-based model notably enhances SLA violation prediction accuracy, rendering it a valuable tool for cloud and service providers. To conduct the performance comparison, we employ a range of classification and regression evaluation metrics. These diverse evaluation metrics enable us to delve into the specific strengths and weaknesses of each model. The SLA violation prediction mechanism endeavours to categorize the testing instances into four categories: Extremely Safe, Safe, Low Risk, and High Risk, based on the feature vectors. Therefore, we utilize Accuracy, Precision, Recall, and F1-score metrics. Additionally, to assess QoS metrics such as response time, we utilize Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), which are well-suited for continuous numerical data. It is noteworthy that these QoS metrics often serve as the focal parameters in SLA modelling and prediction efforts. For instance, *Figure 29* displays the experimental results that correspond to Precision. *Figure 30* display the F1-scores that derived from the experimental evaluation process.



Model	**Extr. Safe**	**Safe**	**Low Risk**	**High Risk**
ARMA (TS)	0.9813	0.0334	0.3981	0.6154
ARIMA (TS)	0.9815	0.0693	0.3834	0.5455
KNN (Reg)	0.8894	0.2996	0.5370	0.7791
DT (Reg)	0.7290	0.3283	0.5894	0.3009
RFs (Reg)	0.9016	0.2967	0.5835	0.6923
SVM (Reg)	0.7692	0.2695	0.6220	0.7952
LSTM (Reg)	0.8659	0.3952	0.6837	0.8025
G. N. Bayes (ML)	0.9676	0.7137	0.6797	0.7979
KNN (ML)	0.8055	0.7542	0.7063	0.8868
DT (ML)	0.9404	0.8287	0.7420	0.9421
RF (ML)	0.9803	0.9038	0.7859	**0.9493**
SVM (ML)	0.7909	0.5872	0.7069	0.9390
FFNN (ML)	0.9131	0.7519	0.6331	0.8597
GNN Default	0.9563	0.8740	0.6886	0.8464
GNN Cos.	0.9570	0.8734	0.6855	0.8780
GNN Jacc.	0.9569	0.8584	0.7004	0.8694
GNN Manh.	0.9537	0.8601	0.7008	0.8795
GNN Euclidean	0.9556	0.8572	0.7026	0.8343
GNN Cos. Jacc.	0.9537	0.8655	0.6890	0.8950
GNN Cos. Manh.	0.9553	0.8535	0.7116	0.8751
GNN Cos. Jacc. Manh.	0.9571	0.8632	0.7000	0.8739
Composite Model	**0.9816**	**0.9241**	**0.8562**	0.9112

Figure 29: Experimental Results of the proposed Composite Model (precision).



Model	**Extr. Safe**	**Safe**	**Low Risk**	**High Risk**
ARMA (TS)	0.5859	0.0631	0.5071	0.4486
ARIMA (TS)	0.8353	0.1220	0.4951	0.4286
KNN (Reg)	0.5750	0.3519	0.6052	0.5702
DT (Reg)	0.7257	0.3234	0.5830	0.3562
RF (Reg)	0.6233	0.2979	0.7049	0.1875
SVM (Reg)	0.4593	0.3326	0.6840	0.5690
LSTM (Reg)	0.7753	0.4567	0.6832	0.5652
G. N. Bayes (ML)	0.9490	0.7907	0.6271	0.6467
KNN (ML)	0.8634	0.7054	0.6751	0.6816
DT (ML)	0.9456	0.8510	0.7577	0.6559
RF (ML)	0.9806	0.8984	0.8355	0.8093
SVM (ML)	0.8095	0.5800	0.7427	0.6610
FFNN (ML)	0.9298	0.7421	0.6898	0.5298
GNN Default	0.9675	0.8348	0.7537	0.6624
GNN Cos.	0.9684	0.8350	0.7553	0.6571
GNN Jacc.	0.9675	0.8377	0.7546	0.6670
GNN Manh.	0.9674	0.8367	0.7530	0.6613
GNN Euclidean	0.9666	0.8351	0.7472	0.6732
GNN Cos. Jacc.	0.9661	0.8269	0.7543	0.6651
GNN Cos. Manh.	0.9683	0.8421	0.7519	0.6660
GNN Cos. Jacc. Manh.	0.9688	0.8401	0.7550	0.6657
Composite Model	**0.9842**	**0.9203**	**0.8708**	**0.8702**

Figure 30: Experimental Results of the proposed Composite Model (F1-score).

4.4.4 Optimal Image Placement Prediction

The rise of data-intensive and latency-sensitive applications, known as NextGen applications, is a significant driver behind the expansion of Edge Computing. Edge computing promises to extend utility computing to applications that were not fully accommodated by the Cloud computing revolution. It achieves this by enabling the deployment of applications in proximity to users and data sources, addressing concerns such as latency, battery life, bandwidth costs, security, and privacy. These applications typically adopt a microservice architecture, comprising independently deployable and loosely coupled services. This architecture supports on-demand and dynamic behaviour to facilitate data and user movement. Consequently, microservices must be activated only when a specific number of users access them from the vicinity of a particular edge resource. As a result, edge devices are required to run multiple microservices over time, necessitating access to sets of VNF/application/microservice images (regardless of their nature, e.g., containers, Unikernels, etc.) to instantiate applications proactively. Providing access to these images poses several challenges, including limiting transfer time to align with the dynamic behaviour of the application, minimizing bandwidth costs for image downloads, and accommodating potentially limited storage capacities of edge resources. These challenges are compounded in scenarios with a high number of edge devices.

Therefore, centralized image repository download may not always be feasible due to potential bandwidth costs and transfer time constraints. We advocate for a distributed approach where images are replicated across a subset of edge resources. Specifically, we propose an approach that divides edge resources into distinct groups and assigns an image to each group. The objective is to balance storage usage with the latency induced by image transfer. Ideally, these edge groups should be determined (and sized) automatically based on actual resource availability and application instance requirements (e.g., size, latency requirements, etc.).

To achieve this, we model image placement as a Minimum Vertex Cover (MVC) problem [25] on the network connecting the Edge nodes. Subsequently, we introduce GNOSIS, a learning approach that combines Graph Neural Networks and Deep Reinforcement Learning. GNOSIS leverages the representation capabilities of Graph Neural Networks (GNNs) along with the strength of actor-critic Reinforcement Learning to provide robust solutions. We extensively evaluate GNOSIS across various network topologies and sizes, comparing it against a greedy approach considered state-of-the-art. While the Greedy algorithm demonstrates superior speed across all network configurations, GNOSIS outperforms it in terms of vertex cover quality for random and small-world networks, albeit falling behind for preferential attachment networks.

Figure 31 displays the execution times (GNOSIS vs Greedy approach) in the context of various network topologies, across various number of vertices. *Figure 32* displays the cost function (GNOSIS vs Greedy approach) in the context of various network topologies, across various number of vertices.

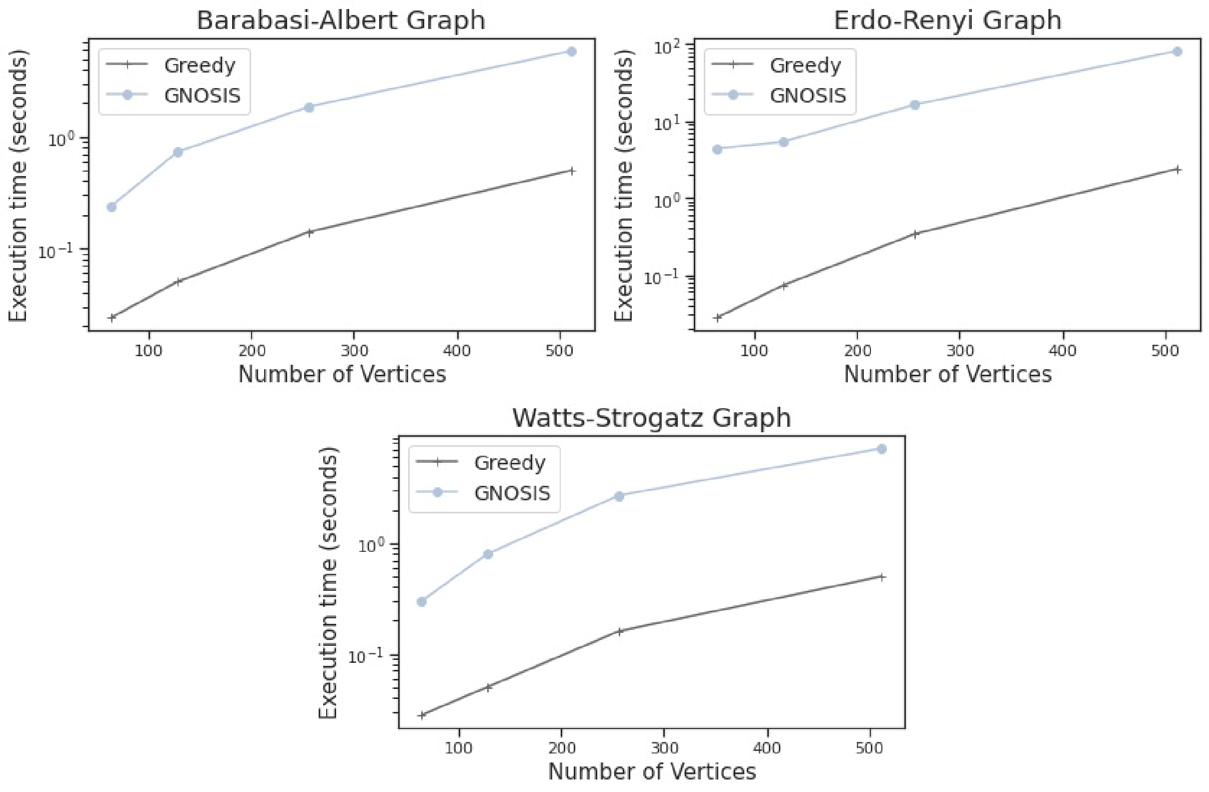


Figure 31: Experimental Results of the proposed GNOSIS (execution time).

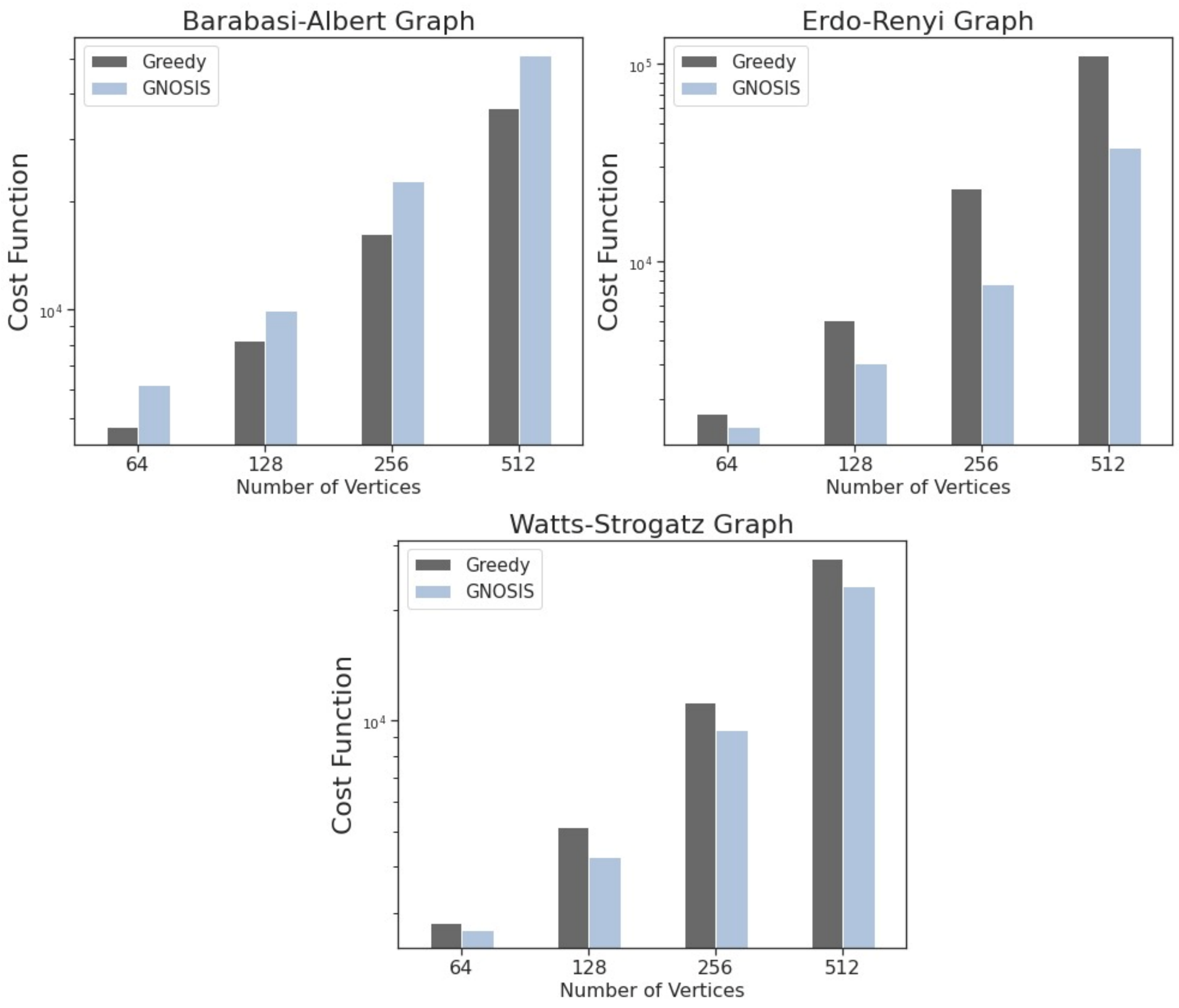


Figure 32: Experimental Results of the proposed GNOSIS (cost function).

4.4.5 Mobility Prediction

Mobility forecasting is of paramount importance in the context of optimally orchestrating edge resources. Regional traffic forecasting presents a burgeoning challenge within the realm of urban mobility, holding significance across various domains including smart cities, Internet of Things, edge computing, wireless networks, and more. It involves predicting future traffic conditions across diverse geographic areas, characterized by grid-based or non-uniform partitioning, and spanning multiple time periods, ranging from minutes to hours. Dedicated forecasting models are utilized for this purpose.

Despite its numerous advantageous applications, regional traffic forecasting is a complex endeavour. It necessitates accurate predictions of traffic conditions across different areas over extended time periods, owing to the intricate and interlinked nature of traffic systems, which exhibit both spatial and temporal characteristics. Spatially, traffic conditions in one region can be influenced by events occurring in neighbouring or distant areas, necessitating an understanding of the spatial dependencies between regions. Temporally, traffic patterns undergo dynamic changes influenced by time cycles of varying lengths, requiring models to capture both short-term fluctuations and long-term trends.

Hence, constructing regional traffic forecasting models requires incorporating information regarding the topology of various regions and the populations traversing them. Technologies such as advanced traffic sensor networks and integrated geographic information systems facilitate the availability of such information by continuously monitoring, documenting, and archiving spatial and temporal data across multiple instances. By combining advanced modelling techniques, real-time data streams, and domain-specific knowledge, robust and accurate forecasting systems capable of handling the intricacies of regional traffic can be developed.

In recent years, spatio-temporal graph neural networks, like GCN-LSTM, have demonstrated state-of-the-art performance in various traffic forecasting problems, primarily due to their ability to effectively incorporate contextual information. However, only a small portion of these endeavors focuses on regional traffic forecasting. Additionally, the use of spatio-temporal graph neural networks for regional traffic forecasting has been limited, with prior attempts typically focusing on capturing either the spatial or temporal aspects of the problem.

Members of the CHARITY project have managed to enhance the GCN-LSTM architecture to enable the integration of information pertaining to diverse populations (temporal aspect) and the regions they traverse (spatial aspect). This enhancement aims to develop more precise and refined prediction models by effectively fusing and distilling information. The outcome of this research effort is a novel spatio-temporal graph neural network architecture called WEST (WEighted STacked) GCN-LSTM. Moreover, the integration of the aforementioned information is accomplished through the utilization of two innovative algorithms known as the Shared Borders Policy and the Adjustable Hops Policy. The proposed solution is showcased in *Figure 33*.

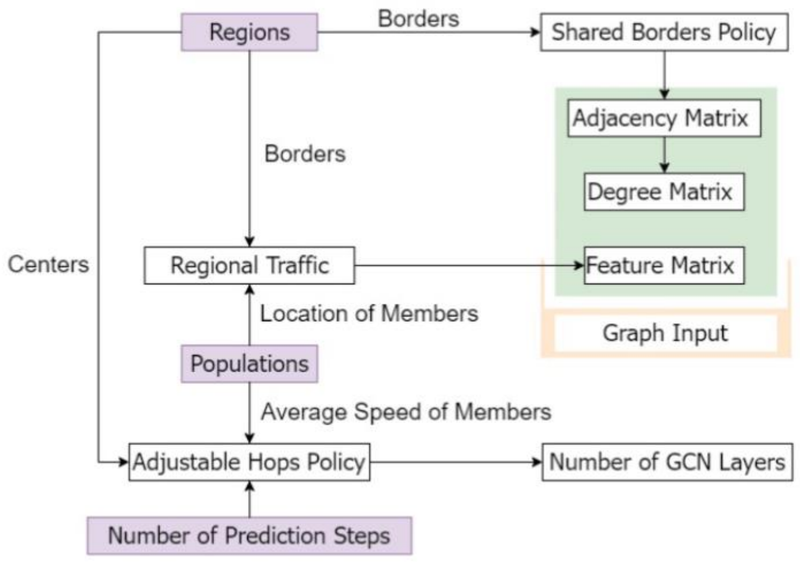
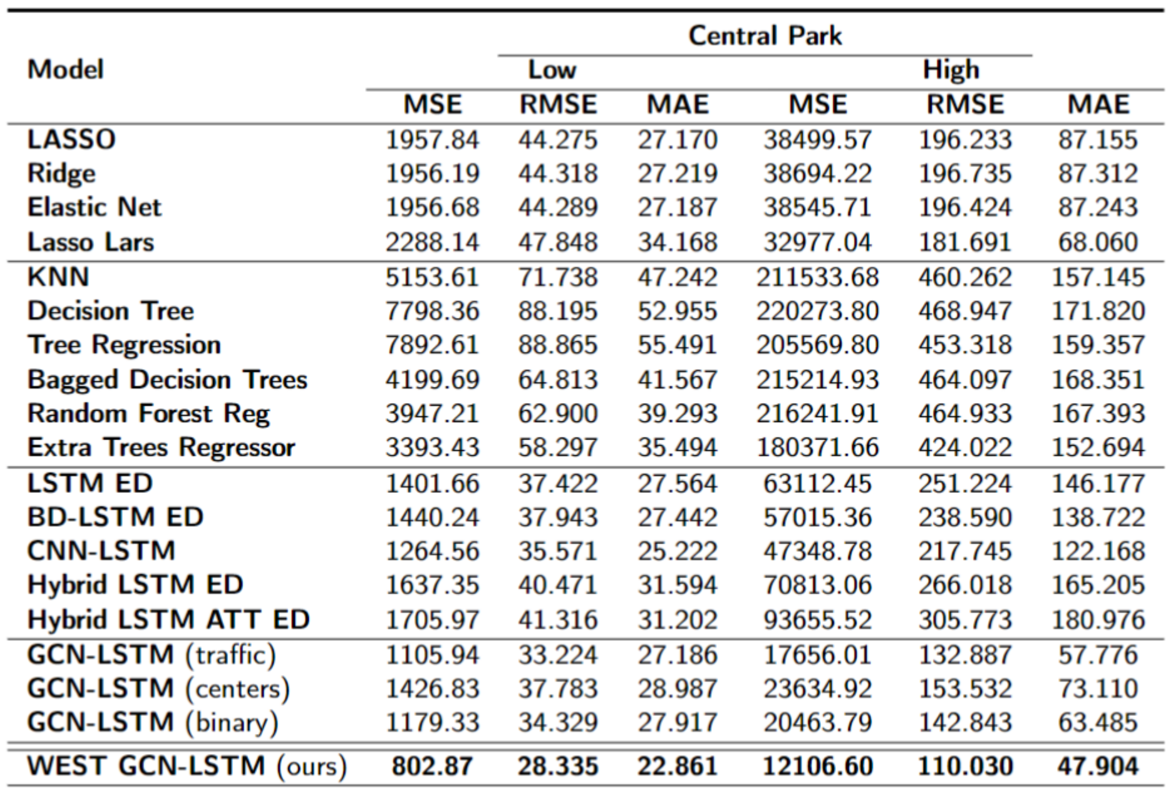


Figure 33: Architecture of the proposed solution.

Through an extensive experimental evaluation process, the ability of the proposed solution to significantly surpass its competitors was established. Figure 34 displays some of the experimental results that derived on the basis of a large scale simulation. The conducted simulation involved the divide of Central Park into 6 distinct regions. The showcased results are indicative of the fact that the proposed solution managed to outperform its competitors in terms of predicting the number of people in each one of these 6 regions across numerous time-steps. More details regarding the architecture of the proposed solution, as well as the specifics of the experimental settings can be referred to at [26].



Model	Central Park					
	Low			High		
	MSE	RMSE	MAE	MSE	RMSE	MAE
LASSO	1957.84	44.275	27.170	38499.57	196.233	87.155
Ridge	1956.19	44.318	27.219	38694.22	196.735	87.312
Elastic Net	1956.68	44.289	27.187	38545.71	196.424	87.243
Lasso Lars	2288.14	47.848	34.168	32977.04	181.691	68.060
KNN	5153.61	71.738	47.242	211533.68	460.262	157.145
Decision Tree	7798.36	88.195	52.955	220273.80	468.947	171.820
Tree Regression	7892.61	88.865	55.491	205569.80	453.318	159.357
Bagged Decision Trees	4199.69	64.813	41.567	215214.93	464.097	168.351
Random Forest Reg	3947.21	62.900	39.293	216241.91	464.933	167.393
Extra Trees Regressor	3393.43	58.297	35.494	180371.66	424.022	152.694
LSTM ED	1401.66	37.422	27.564	63112.45	251.224	146.177
BD-LSTM ED	1440.24	37.943	27.442	57015.36	238.590	138.722
CNN-LSTM	1264.56	35.571	25.222	47348.78	217.745	122.168
Hybrid LSTM ED	1637.35	40.471	31.594	70813.06	266.018	165.205
Hybrid LSTM ATT ED	1705.97	41.316	31.202	93655.52	305.773	180.976
GCN-LSTM (traffic)	1105.94	33.224	27.186	17656.01	132.887	57.776
GCN-LSTM (centers)	1426.83	37.783	28.987	23634.92	153.532	73.110
GCN-LSTM (binary)	1179.33	34.329	27.917	20463.79	142.843	63.485
WEST GCN-LSTM (ours)	**802.87**	**28.335**	**22.861**	**12106.60**	**110.030**	**47.904**

Figure 34: Experimental Results of the proposed solution.

4.5 Relation to CHARITY

As highlighted several times, CHARITY aims to achieve a platform, inspired by the ETSI Zero-touch specification, which can be used to reduce the human interaction in the expected complex life-cycle management of next-generation XR applications. To fulfil such a vision, different orchestration and management functions will leverage the notion of control closed loops (e.g., MAPE-K loops) as a structured chain of steps from retrieving environment information up to the decision actuation. Orchestration and management functions require constant analysis of the managed entities. Monitoring is the initial step of control closed loops and is essential to track the real-time QoE of XR applications and to ensure they run as expected. In this vein, this section discussed different monitoring tools with a brief comparison among them and the monitoring solution with regards to cloud data, network, and service data/metrics. It then discussed how the obtained monitoring data can be leveraged for the prediction of computation and network resources, as well as the SLA violation and mobility feature of mobile XR applications with various effective prediction methods, e.g., AI-based prediction algorithms, in the scope of the CHARITY framework. As mentioned before, the realization of more autonomous and intelligent orchestration mechanisms, as envisioned in CHARITY, highly depends on the inputs provided by such monitoring and prediction components. Hence it becomes critical to understand how they fit in the proposed architecture and how they can be used as the inputs for the proposed mechanisms. Together with the intelligent prediction algorithms, monitoring tools are key elements to automate the network and service management, reduce detection and decision times and minimize the need for human intervention in the overall service orchestration.

5 XR Application Management Framework

The CHARITY project aims to put in place a technological platform which deals with XR applications (AR, VR and Holography based applications): XR applications are provided with fully integrated edge computing solutions offering high performance, lower latency, scalability and security. The realization of this vision is empowered by the most recent technologies and by a new way to rethink standard models. This journey begins by offering to Next Generation Application Developers (NextGen) a set of services and tools to be used to speed up release rates, reduce costs, and mitigate risks throughout the development process. In CHARITY, a DevOps/ Continuous Integration/Continuous Delivery (CI/CD) concept has been adopted and integrated into an Application Management Framework (AMF), offering to NextGen developers an environment for rapid design, testing and integration of highly interactive and collaborative next-generation services. As part of this toolset, NextGen developers will be offered functions to design and manage, at design time, network slice blueprints.

The AMF framework can be considered an entry point for XR application developers, from which they shape and manage their XR service: through a portal named Application Management Portal (AMP), they can start their CHARITY lifecycle. In CHARITY, NextGen developers are the owners of Use Cases who, through AMP functionalities, will take advantage of the service enablers provided by CHARITY and will validate the results through their targeted demonstrators. *Figure 35* depicts the CHARITY high level architecture, in which the AMF framework is highlighted.

Interaction with Network Service Orchestrator is implemented in a loosely coupled way: the mechanisms to communicate inside CHARITY platform is REST APIs.

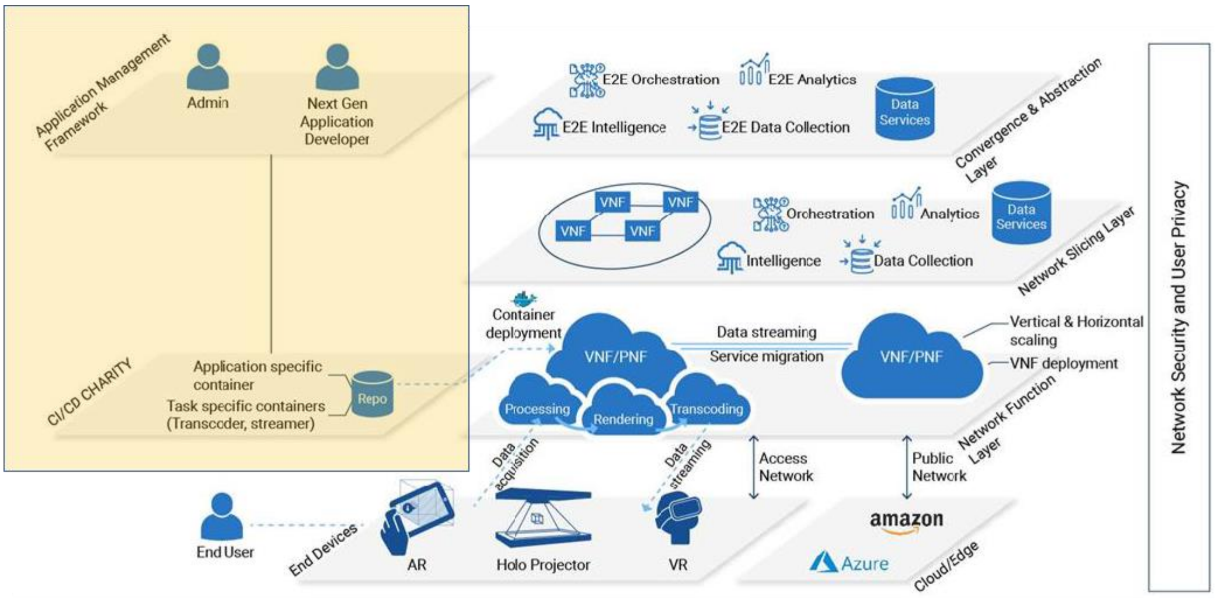


Figure 35: CHARITY High Level Architecture (as initially envisioned in the description of work).

We outline here the provided functions, distinguishing between CHARITY specific slice blueprint management and other application composition functions.

Regarding application composition:

- Use cases application **registration/onboarding** by using available tools for a CI/CD chain wherever possible.
- Definition of application model (templates describing the different application bricks and their interconnection).
- Whenever possible, validation of composed applications in a dedicated environment. This strongly depends on the technological requirements of the applications: in some cases, it may

not be possible to set up a validation runtime, e.g., for components requiring dedicated hardware (e.g., specific GPU models), or high demand in physical resources.

- Management of dynamic changes to the application model.

Registration/onboarding consists in the possibility, for NextGen developer, to upload the application components, packaged as virtual machine or container images (aka artefacts), in a repository where they can univocally be registered with an abstract description of the artefact itself (eventually guided by GUI). A normal input for a CI/CD chain is the source code for which CI is a consistent and automated way to build, package, and test it: in CHARITY, the input is however expected to be wrapped up in VNF or CNF images (VMs or containers). After the onboarding process, the artefact is available and tagged in the artefact repository.

Whenever possible, each described artefact (that will need to be validated internally through unit and E2E tests before being uploaded into the CHARITY repository) will be validated with tests, possibly along with native CHARITY components. Validation should be done in a test environment e.g.:

- via single component smoke test run (if provided by NextGen Application Developers)
- if possible in terms of resource requirements, via integration tests provided by NextGen Application Developers, running with CHARITY components (mocks or full)
- via a security scan

For a registered and validated artefact, the owner can request to deploy, renew the deployment with a new version, and to decommission the artefact. The same request can be also issued programmatically by a device, for the use cases that require a more dynamic activation mechanism. In this case, if the request is triggered by an and-user device (e.g., a customer of an organization providing the XR Application), it will be mediated by a proxy owned by the same organization, which will present the request on behalf of the end user with the organization CHARITY authentication credentials. *Figure 36* represents the above-mentioned interactions allowed to the XR Developer for the definition and management of the developed artefacts.

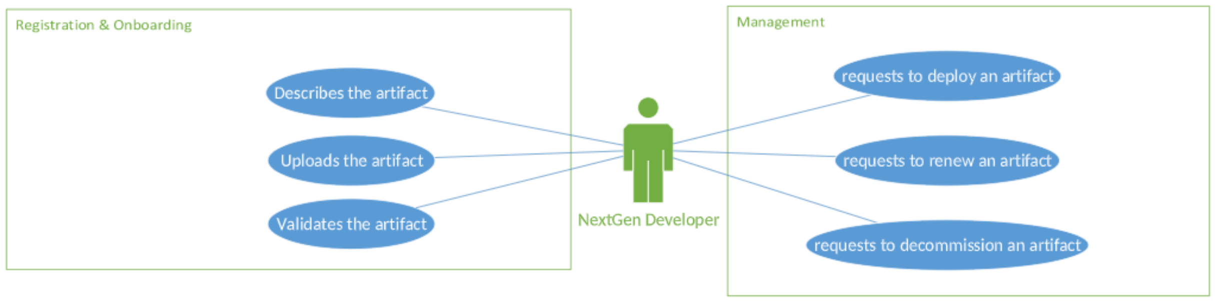

Registration & Onboarding
Describes the artifact
Uploads the artifact
Validates the artifact
NextGen Developer
Management
requests to deploy an artifact
requests to renew an artifact
requests to decommission an artifact

Figure 36: Developers' Activities.

After the NextGen Application Developer decides to deploy/decommission an artefact, the Application Management Framework alerts orchestration components that an artefact is ready to be deployed-renewed/decommissioned. Mechanisms has been developed to pass run-time instance specific deployment parameters not included in the abstract application model e.g., geo-location parameters. The interface between AMF and Network Service Orchestrator is e based on REST API calls. *Figure 37* depicts the high-level deployment sequence.

In terms of blueprint management, the following functions will be provided:

- Designing - from scratch or via copy - a blueprint as a composition of Network Services, with WP3 artefacts -as VNF-, virtual links, connection endpoints; the slice blueprint might or not include also bare metal infrastructure element descriptors as PNFs.
- Registering, modifying, validating and storing abstract blueprints.

NextGen application developers can choose, picking in the AMP GUI, all the needed elements to compose the desired Network Service but not its actuation: VNFs, Virtual links between elements, specific network service (routing, encoding, streaming…) and also service enabling artefacts provided by CHARITY platform. Selected abstract objects will be collected and will represent inputs for the design tools whose outputs will be network slice blueprints. AMF stores such blueprints, see *Figure 38*, in a NSL catalogue (XR Service Blueprint Templates Repository) that can be accessed by the orchestration layer (see **Section 3**).

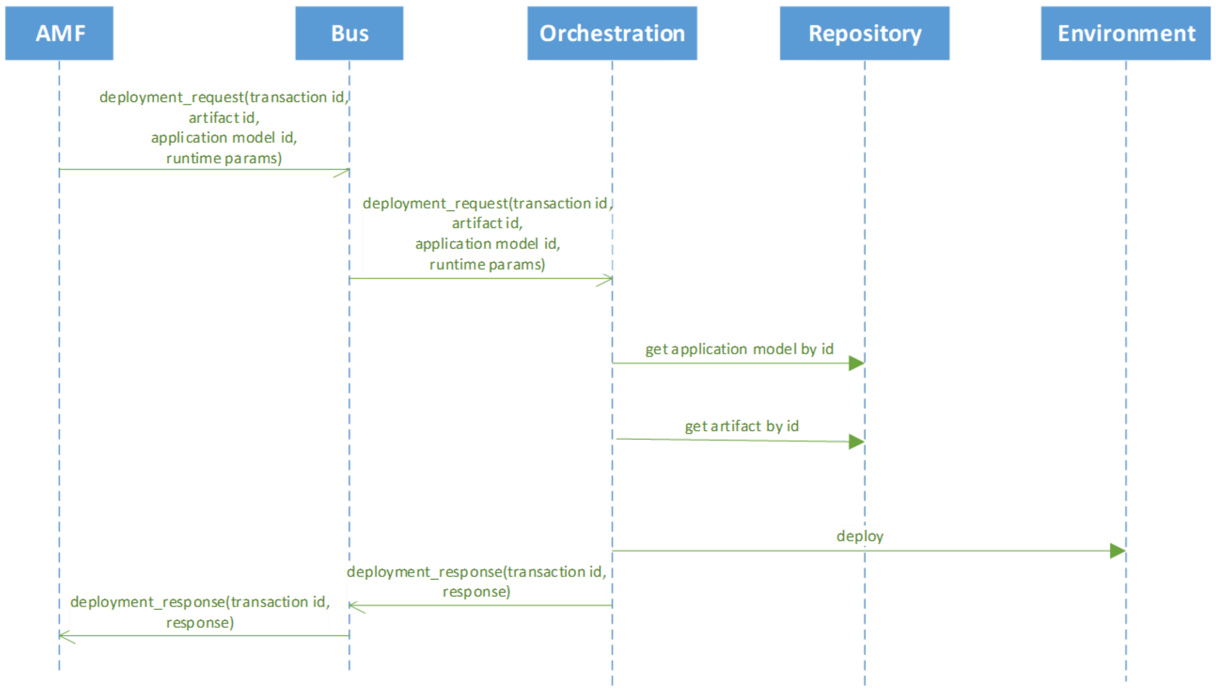

AMF
Bus
Orchestration
Repository
Environment
deployment_request(transaction id, artifact id, application model id, runtime params)
deployment_request(transaction id, artifact id, application model id, runtime params)
get application model by id
get artifact by id
deploy
deployment_response(transaction id, response)
deployment_response(transaction id, response)

Figure 37: Deployment sequence.

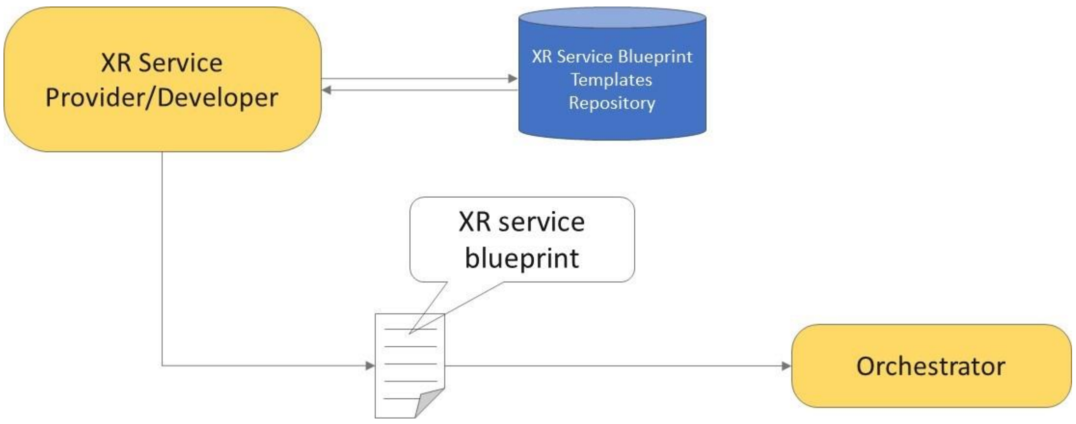

XR Service Provider/Developer
XR Service Blueprint Templates Repository
XR service blueprint
Orchestrator

Figure 38: AMF and network blueprints.

Blueprints do not contain runtime objects or artefacts; they contain an abstract definition of the network service: for example, each time a NS Admin request for an encoding service is received, the specific blueprint template is sent to a “translator” engine that transforms the blueprint capabilities into a detailed blueprint containing reference to the encoding software version and so on. The detailed blueprint is represented in a TOSCA compliant format and is then consumed by the Network Orchestrator. In this sense, the network slice blueprint created by the AMF can be comparable with a Network Service Descriptor according to the ETSI MANO specification.

The multi-cloud perspective adds degrees of complexity since it is not feasible to use the basic Prometheus visualization and alerting tools. The CHARITY project contemplates a dispersed architecture between different cloud domains for the applications deployed to achieve its best performance. Therefore, we must create adapted tools that collect and display only the data that affects the microservices of each application, as explained in **Section 4**.

5.1 XR Application Management Framework Architecture

The AMF relies on the architectural components depicted in *Figure 1* and is described in the following (see *Figure 39*):

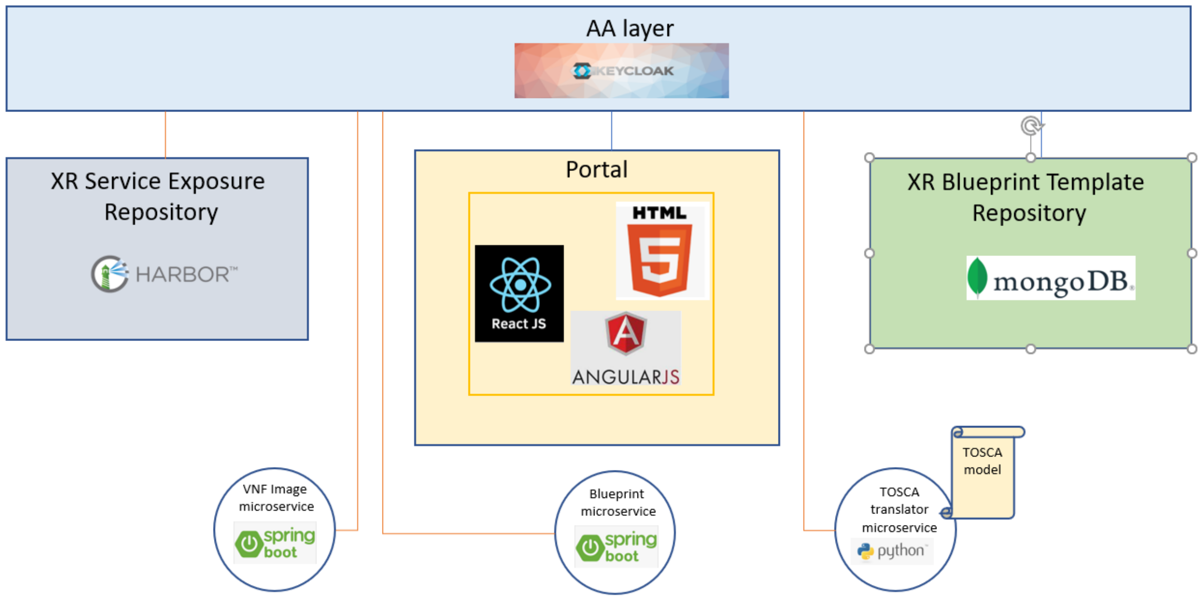


Figure 39: Architectural components of the XR Application Management Framework.

- The WEB Portal acts as the entry point to the CHARITY platform for end users (typically XR developers), enabling them to upload into the platform the software artefacts representing the building blocks of their applications, and to define an application model (Blueprint Template) for the application itself. The model defines which building blocks (VNFs) compose the application, how they are interrelated in terms of connectivity, and the requirements in term of resource needs, hardware constraints and service level expected. All these pieces of information are needed by the Orchestration and Monitoring layers to deploy the application on the most suitable runtime and to enable the monitoring of the relevant QoS parameters. In addition, the Portal also enables an end user of the CHARITY platform (an XR Service developer or administrator) to trigger a deployment request for a specific Blueprint. This functionality is also offered via REST API endpoints, to allow machine-to-machine activation for the use cases that require automated workflow of deployments on behalf of an end user, started by a device (in this case, the request will be mediated by a proxy owned by the same organization offering the XR Application).
- The XR Service Enabler repository is the component responsible for storing and making available the VNF images developed and packaged by the XR Service developers, in the form of container or virtual machine images. Images can be uploaded to the repository by the developers, tagged and enriched with metadata. Such images can be referred to during the editing of an application Blueprint template.
- The XR Service Blueprint Template repository is the component responsible for the storage, versioning and access of the application Blueprint Templates created by the XR developers.
- The backend microservices are the components implementing the business logic of the Portal. Each microservice is specific for a set of functionalities provided by the user interface. Two microservices have been implemented for the interaction with the XR Service Enabler repository and with the XR Blueprint Template repository. An additional microservice is dedicated to the translation at runtime or a upon request of a blueprint template into the TOSCA representation expected by the orchestration layer for deployment actions.

- The TOSCA Model is the model that allows to define an application Blueprint Template through a TOSCA-compliant representation. Being compliant to an official standard supported by the ETSI Mano specification is critical not only from the technical viewpoint, but also for dissemination purposes, as it allows to use a language shared and understood by the scientific community.
- The TOSCA Translator is a component that converts, upon a deployment request, the representation of the application Blueprint created by the XR developer from the AMF internal format (JSON) into the TOSCA format defined for CHARITY.
- The Authentication and Authorization layer ensures that all the access to resources provided by the portal and by the backend components are made by users or services which have proper permissions. For both end users and automated services, a role-based policy is enforced and checked at every request leveraging a centralized component which denies access whenever a condition is not met.

5.2 XR Application Management Framework Portal

The Portal has been designed with extensibility and separation of roles in mind. Due to the large number of services and components envisioned for the CHARITY platform, the Portal architecture leaves space for any additional functionality made by independent development teams, each one responsible for specific areas of the platform.

To provide this level of flexibility and independence, the Portal adopts the Micro Frontend architecture[23] paradigm. With Micro Frontends, the visualization layer (Web GUI) is composed by a shared main application (Web App Shell) which acts just as a placeholder for views defined and managed by independent applications. As shown in *Figure 40*, this paradigm allows different development teams to work in parallel, implementing their pipelines independently and in a technology-agnostic way and to integrate seamlessly the visualization of their services in the GUI, without the support of teams dedicated to the GUI layer.

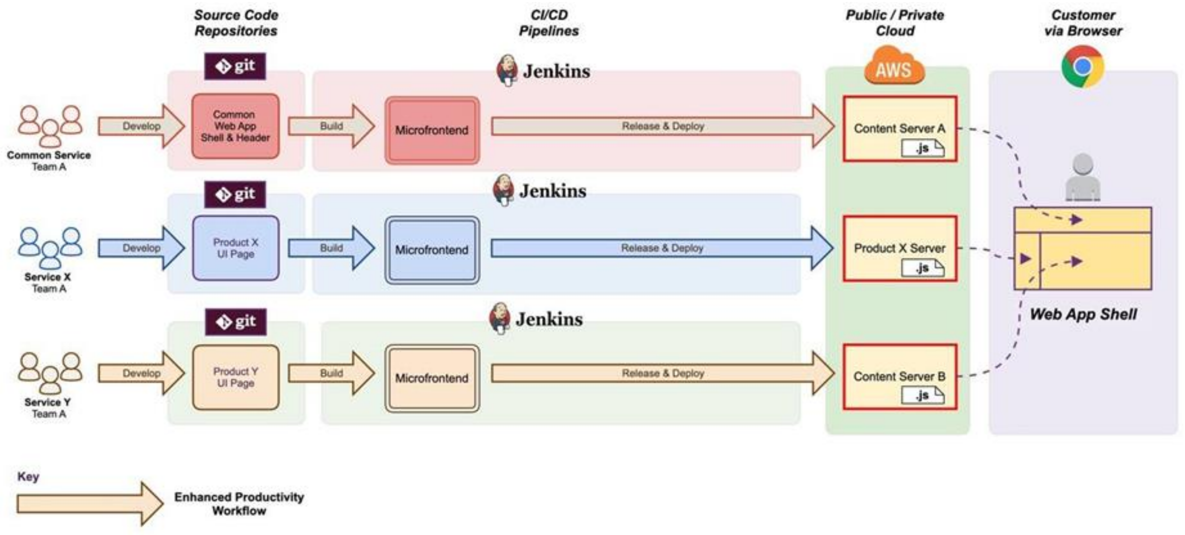


Figure 40: How a Micro frontend architecture enables independent end to end development workflows[24].

[23] https://micro-frontends.org/

[24] Picture taken from: https://www.trendmicro.com/it_it/devops/21/h/micro-frontend-guide-technical-integrations.html

To implement the CHARITY Micro Frontend architecture, a set of tools and technologies based on Html and JavaScript have been selected (*Figure 41*), namely:

- AngularJS: for the visualization layer
- Webpack: a static module bundler for modern JavaScript applications, specifically the Webpack Module Federation functionality, which allows forming a single application by the composition of several separate builds. These separate builds should not have dependencies between each other, so they can be developed and deployed individually.

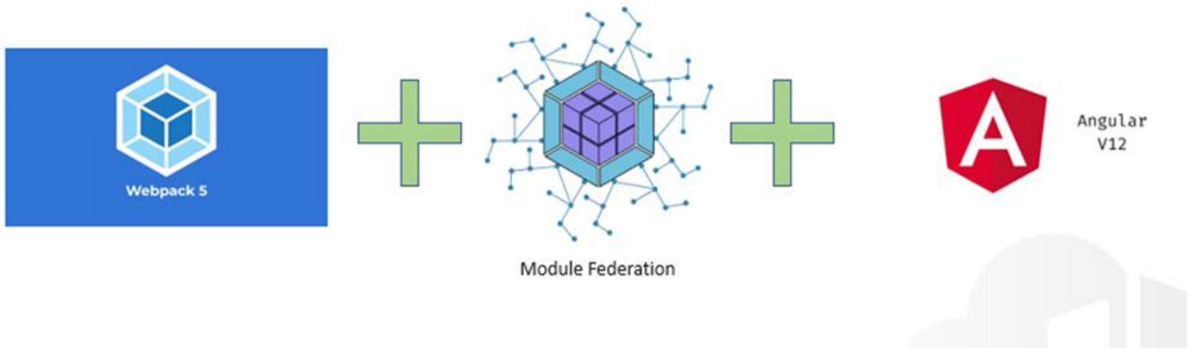


Figure 41: Technologies allowing Micro Frontends in CHARITY.

In the following we show, with the help of screenshots, the AMF Portal functionalities implemented in this project.

The entry point to the platform is the Login page (*Figure 42*). From the end user standpoint, the login mask is a typical form requiring user and password for entering the portal. Indeed, the login is not directly implemented into the Portal, but is delegated to the external Authentication and Authorization system which knows about users, services, roles and the related assignments according to the openid-connect protocol.

Every call to an XR AMF endpoint, comprising invocations coming from and to the backend microservices, must contain a token returned by the external openid-connect authenticator. Such token is self-contained, meaning that it contains, among many pieces of information, also declarations about the roles of the callers.

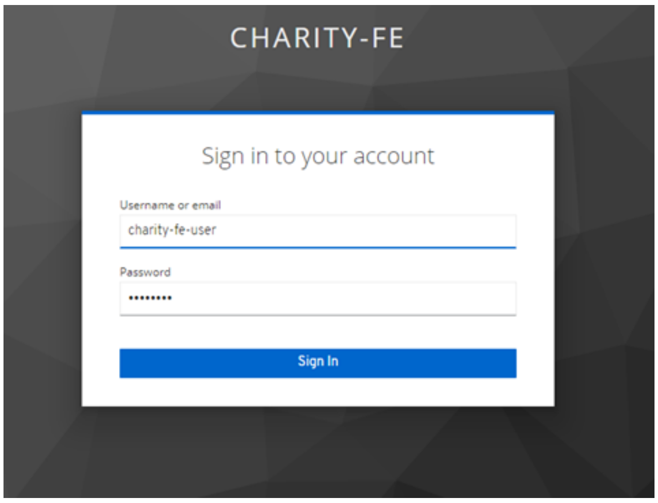
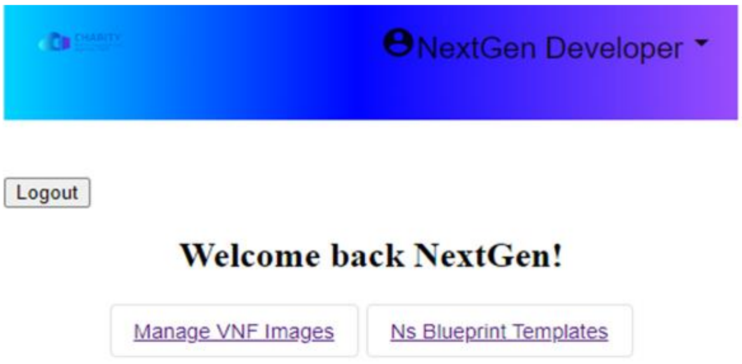


Figure 42: Login page mediated from the openid-connect authentication provider.

For users with role properly configured, the login will provide a token that allows access to the portal GUI sections for the management of VNFs and creation and management of Blueprints. Furthermore, the operations performed by the Portal on behalf of the user towards the backend services are checked against the roles of the end user and are filtered accordingly.

Once a successful authentication is performed, the user is redirected to the AMF GUI landing page, which is currently offering the "Manage VNF Images" and "NS Blueprint Templates" high level functionalities (*Figure 43*).

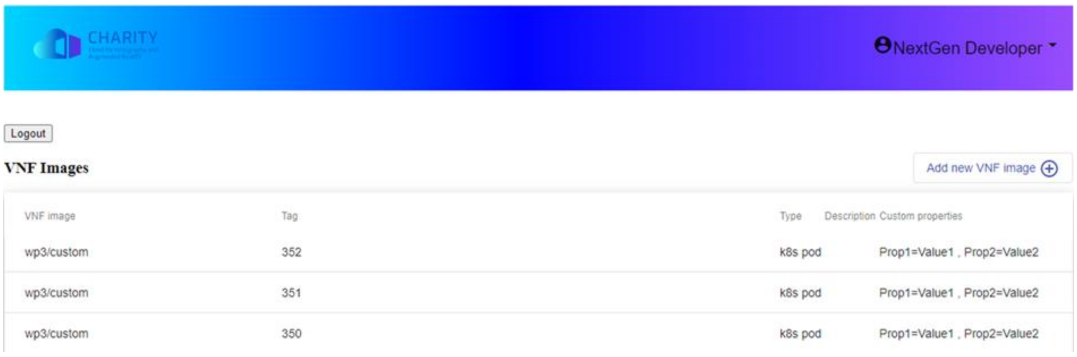


Welcome back NextGen!

Figure 43: Landing page for an XR Developer.

The "Manage VNF Images" area allows a developer to:

- List (*Figure 44*) the VNF images available in the system (private to the developer or public for their organization).

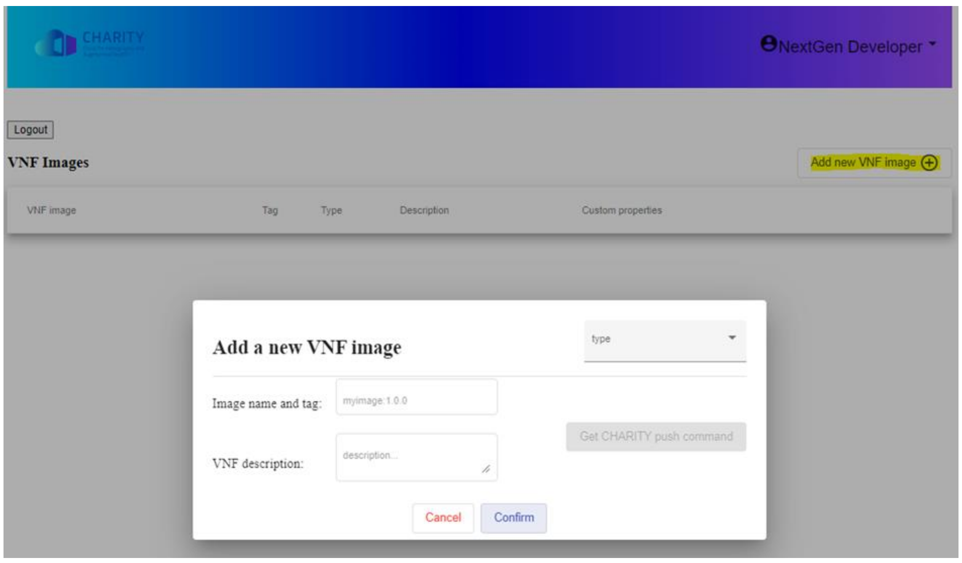


VNF Images

VNF image	Tag	Type	Description	Custom properties
wp3/custom	352	k8s pod		Prop1=Value1 , Prop2=Value2
wp3/custom	351	k8s pod		Prop1=Value1 , Prop2=Value2
wp3/custom	350	k8s pod		Prop1=Value1 , Prop2=Value2

Figure 44: List of VNF images available.

- Upload (*Figure 45*) a VNF image, packaged as container image, into the CHARITY XR Service Enabler repository. They can also specify additional data to enrich the basic information (image name and tag) representing the VNF.



VNF Images

VNF image	Tag	Type	Description	Custom properties

Add a new VNF image

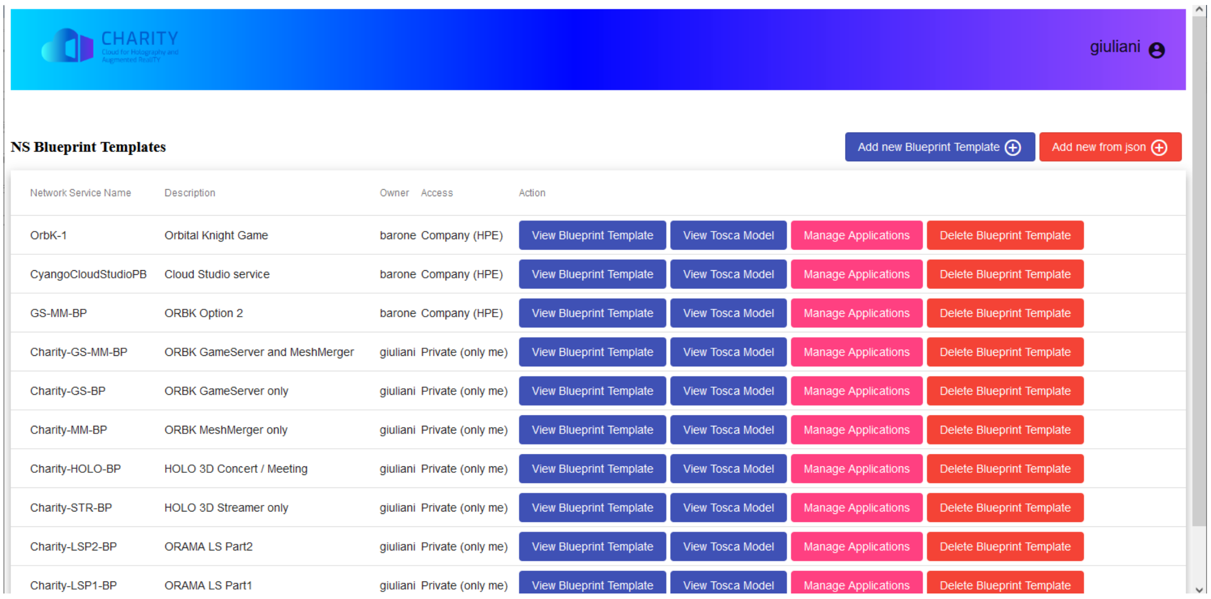
type	
Image name and tag:	myimage:1.0.0
VNF description:	description...

Figure 45: Add new image form.

Following the standard for the management of container images, the user does not upload the VNF image data directly into the Portal, rather they get back a command to push the image into the image registry where images are backed. This allows quick uploads when existing images are updated or tagged, as only the new layers are appended to the existing ones. To allow this, the end user is also automatically authenticated to the image registry in the XR Service Enabler Repository, through proper configuration of roles in the authentication provider component. They will get a secret that will be used as password in the image related commands (push/pull/tag).

The “NS Blueprint Templates” functionality allows for the visualization, creation, and maintenance of Blueprint Templates.

The main page (*Figure 46*) lists the Blueprint Templates available to the end user, depending on his/her role. For every item, the user can view the details and, depending on the access rights, modify the template. In addition, a button in the top right area allows for the creation of a new Template.



NS Blueprint Templates

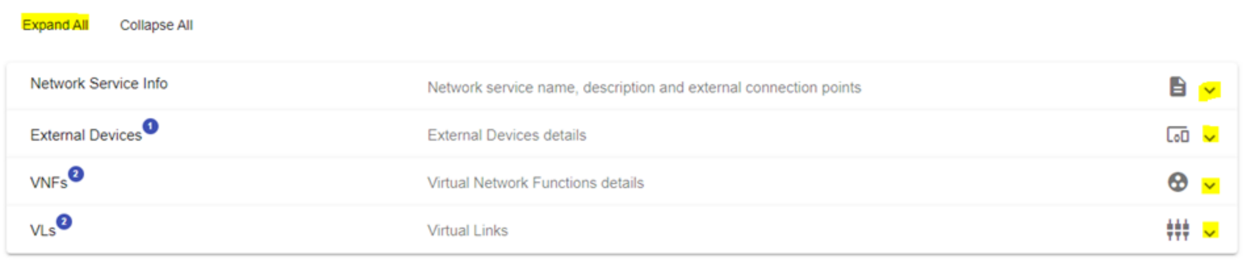
Network Service Name	Description	Owner	Access	Action			
OrbK-1	Orbital Knight Game	barone	Company (HPE)	View Blueprint Template	View Tosca Model	Manage Applications	Delete Blueprint Template
CyangoCloudStudioPB	Cloud Studio service	barone	Company (HPE)	View Blueprint Template	View Tosca Model	Manage Applications	Delete Blueprint Template
GS-MM-BP	ORBK Option 2	barone	Company (HPE)	View Blueprint Template	View Tosca Model	Manage Applications	Delete Blueprint Template
Charity-GS-MM-BP	ORBK GameServer and MeshMerger	giuliani	Private (only me)	View Blueprint Template	View Tosca Model	Manage Applications	Delete Blueprint Template
Charity-GS-BP	ORBK GameServer only	giuliani	Private (only me)	View Blueprint Template	View Tosca Model	Manage Applications	Delete Blueprint Template
Charity-MM-BP	ORBK MeshMerger only	giuliani	Private (only me)	View Blueprint Template	View Tosca Model	Manage Applications	Delete Blueprint Template
Charity-HOLO-BP	HOLO 3D Concert / Meeting	giuliani	Private (only me)	View Blueprint Template	View Tosca Model	Manage Applications	Delete Blueprint Template
Charity-STR-BP	HOLO 3D Streamer only	giuliani	Private (only me)	View Blueprint Template	View Tosca Model	Manage Applications	Delete Blueprint Template
Charity-LSP2-BP	ORAMA LS Part2	giuliani	Private (only me)	View Blueprint Template	View Tosca Model	Manage Applications	Delete Blueprint Template
Charity-LSP1-BP	ORAMA LS Part1	giuliani	Private (only me)	View Blueprint Template	View Tosca Model	Manage Applications	Delete Blueprint Template

Figure 46: Main page for the “NS Blueprint Templates” functionality.

The Blueprint Editor is based on an “accordion” visualization, which is a collection of expandable items. Every item corresponds to a specific element of the template and can in turn comprise a secondary-level accordion to represent a collection of sub elements.

This allows to maintain the visualization compact, expanding only the element that the user wants to edit.

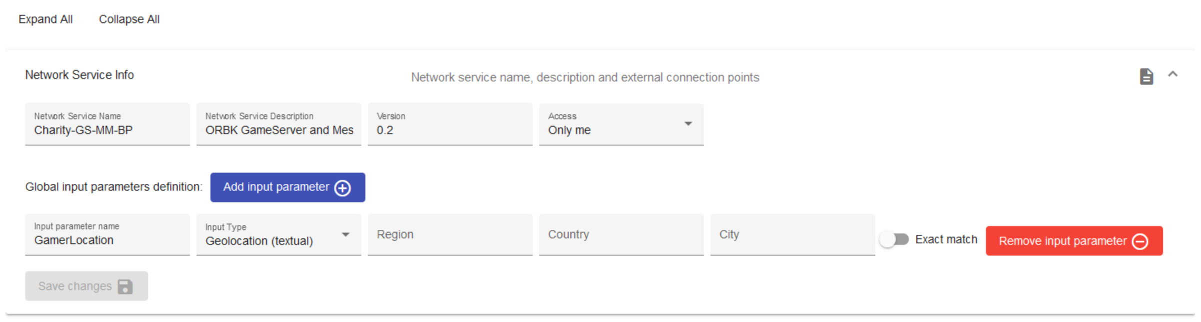
Figure 47 depicts the “summary” view that the user gets when selecting a Blueprint Template from the list. The GUI shows only top-level components, and for each of them, the number of defined elements (blue numbered icon) and an arrow for expanding into the details. At the same time, it is also possible to expand all the elements to have a complete view of the whole definition.



Network Service Info	Network service name, description and external connection points
External Devices 1	External Devices details
VNFs 2	Virtual Network Functions details
VLs 2	Virtual Links

Figure 47: Summary view (“Collapsed”) of a Blueprint Template definition.

By expanding the “Network Service Info” panel (*Figure 48*), the user can edit general information such as the service name, a description, and a version number. They also can define the input variables whose values might be initialized with instance specific values at deployment time (e.g. the GamerLocation in this example).


Expand All Collapse All

Network Service Info — Network service name, description and external connection points

Network Service Name: Charity-GS-MM-BP
Network Service Description: ORBK GameServer and Mes
Version: 0.2
Access: Only me

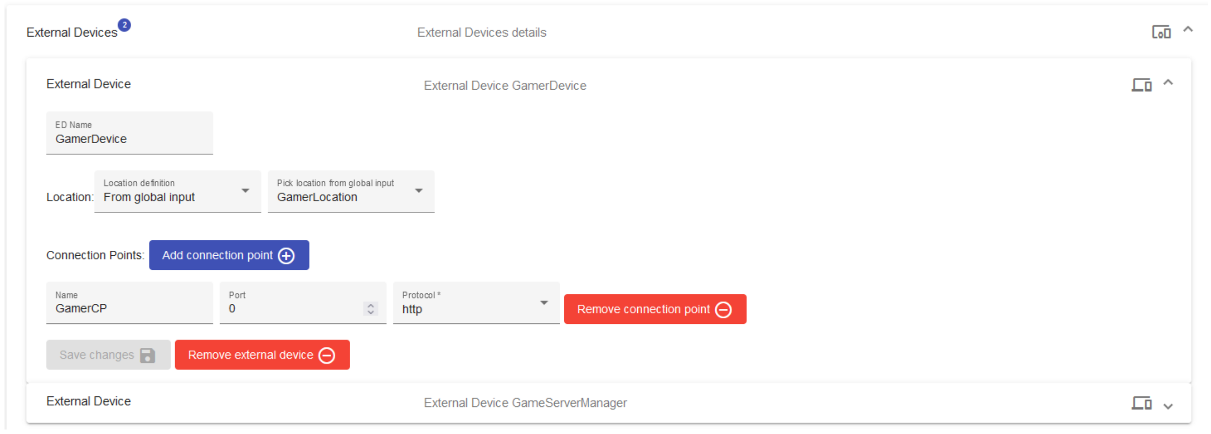
Global input parameters definition: Add input parameter

Input parameter name: GamerLocation
Input Type: Geolocation (textual)
Region
Country
City
Exact match
Remove input parameter

Save changes

Figure 48: Editing of the NS general information.

The next panel (*Figure 49*) allows to describe the “External Devices”. This step is needed whenever it is necessary to describe to CHARITY XR applications how external devices interact with them. In particular, they can define a set of connection points describing which ports are used for the communication.


External Devices 2 — External Devices details

External Device — External Device GamerDevice

ED Name: GamerDevice

Location: Location definition: From global input; Pick location from global input: GamerLocation

Connection Points: Add connection point

Name: GamerCP
Port: 0
Protocol *: http
Remove connection point

Save changes
Remove external device

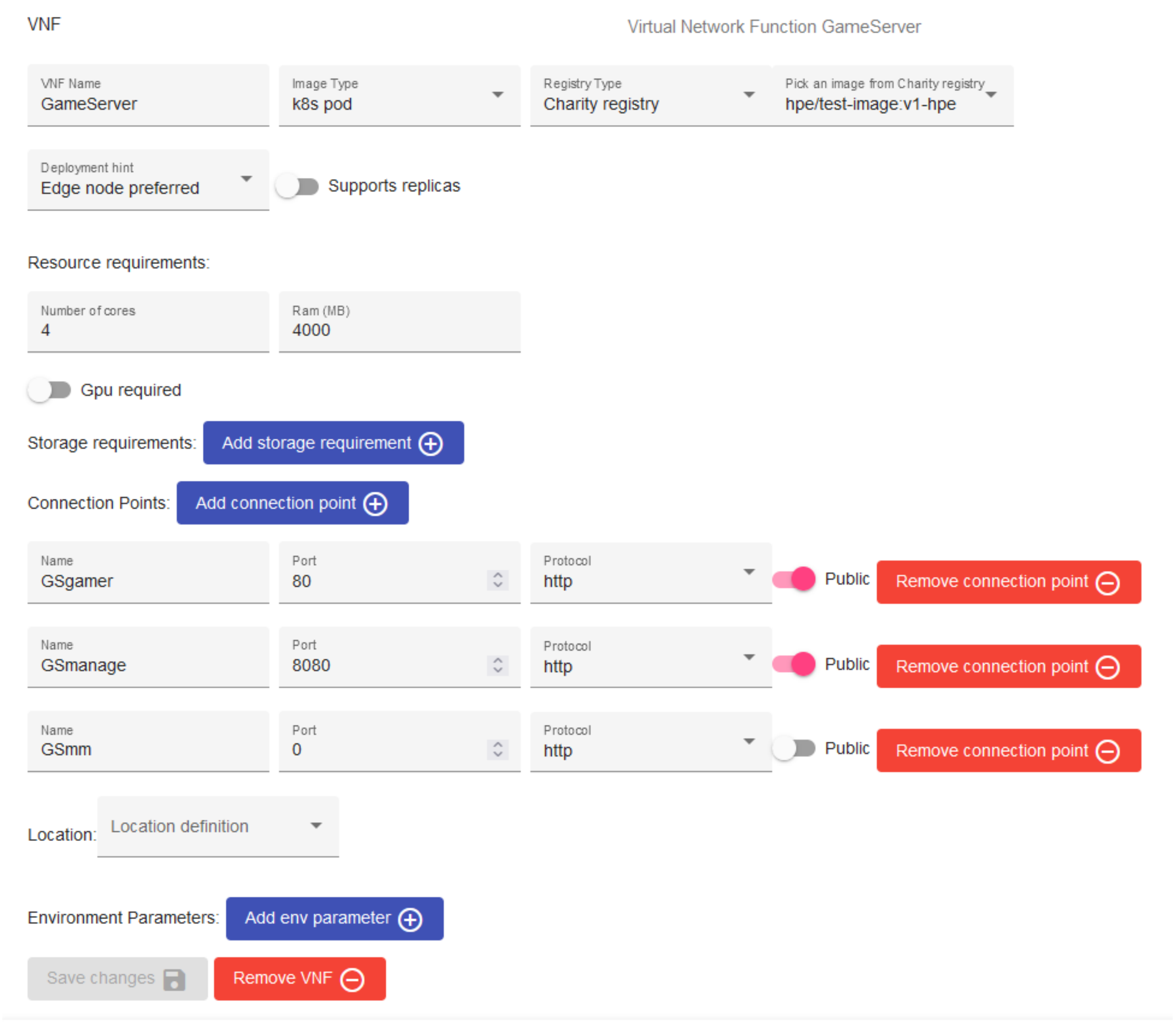
External Device — External Device GameServerManager

Figure 49: Editing of the NS “External devices” Section.

The “VNFs” panel (*Figure 50*) is where the XR developer specifies which set of VNF composes the application. For every VNF to be added to the template, the user can:

- Specify a name.
- Search for the VNF image from the XR Service Enabler Repository.
- Specify some recommendations for the deployment (e.g., edge node/cloud node preferred).
- The set of connection points (ports description) used by the VNF image.
- The input data they should receive, or other relevant information that may be needed (e.g., location) which are not part of the application but are needed to better specify the application components deployment.
- Custom parameters to be injected at image deployment time (e.g., environment vars).
- Requirements in terms of hardware or software resources (e.g., CPU, RAM, GPU, Operating System, … – not shown in the figure).

The developer can iteratively repeat the editing for each VNF to be added, by clicking the “Add VNF” button in the bottom right area of the VNF panel.


VNF

Virtual Network Function GameServer

VNF Name: GameServer
Image Type: k8s pod
Registry Type: Charity registry
Pick an image from Charity registry: hpe/test-image:v1-hpe

Deployment hint: Edge node preferred
Supports replicas

Resource requirements:

Number of cores: 4
Ram (MB): 4000

Gpu required

Storage requirements:

Connection Points:

Name	Port	Protocol	Public
GSgamer	80	http	on
GSmanage	8080	http	on
GSmm	0	http	off

Location: Location definition

Environment Parameters:

Figure 50: Editing of the VNFs section.

The “Virtual Links” panel (*Figure 51*) allows the developer to specify how all the components defined should communicate to each other, and what are the requirements of the connection in terms of quality of service.

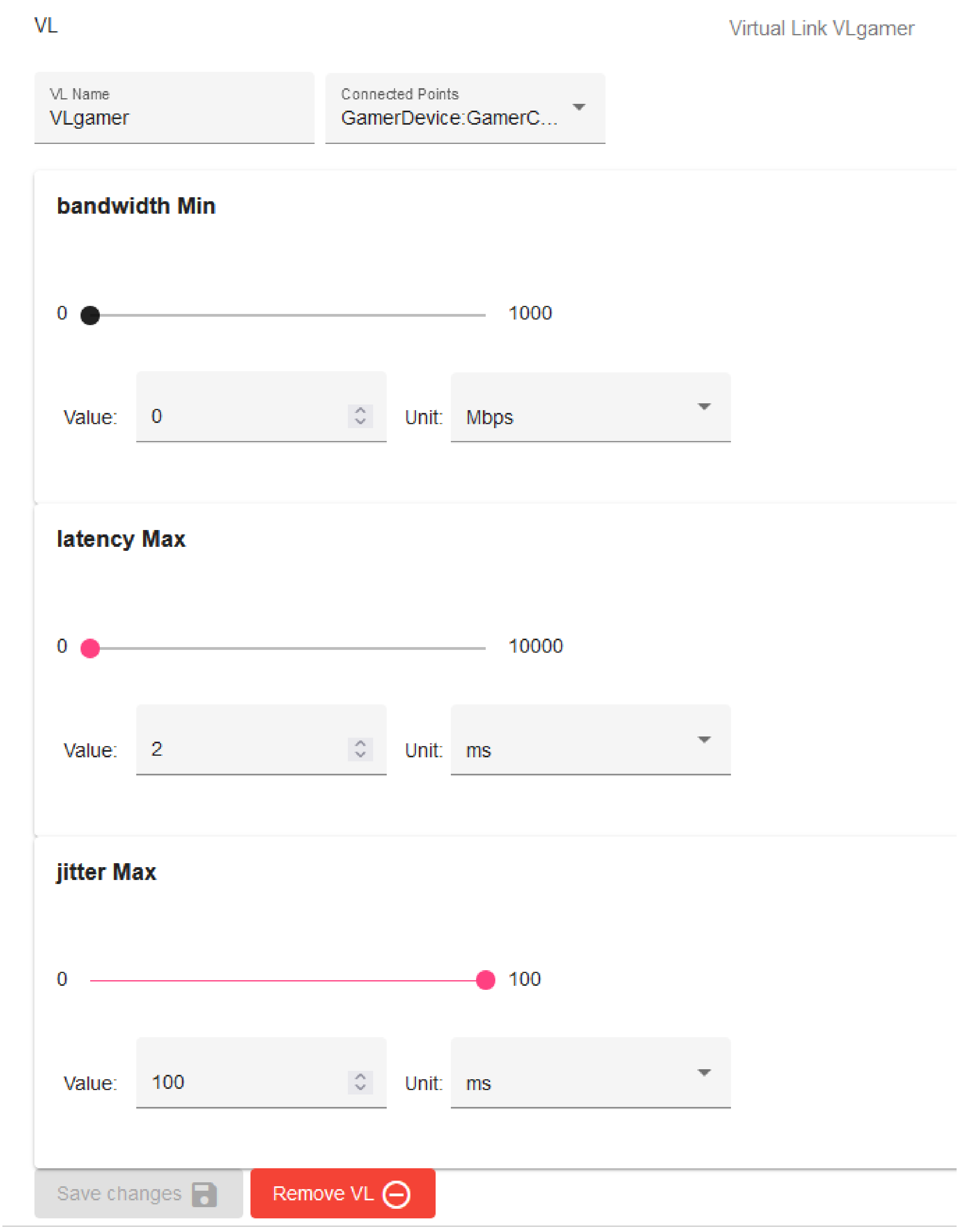
For every virtual link, the user can define a name and, most importantly, can easily define through a multi-select drop down list the VNFs and related connection points that the virtual link itself is expected to connect. A detail of the multi-select drop down list is shown in *Figure 52*: the multi-select shows all the connection points defined in the Blueprint, grouped by VFN, and provides checkboxes for each of them. The user has a clean view of all the connectable endpoints and can easily select all the ones to be connected by the current Virtual Link.

After defining the relationship between the Virtual Link and the connected points, the user can define requirements related to the QoS via a slider-based interface.

Currently, it is possible to specify requirements for:

- Minimum bandwidth (Kbps, Mbps, Gbps)
- Maximum latency (ms)
- Maximum jitter (ms)

For bandwidth, it is also possible to specify the unit via a drop-down list.


VL

Virtual Link VLgamer

VL Name: VLgamer

Connected Points: GamerDevice:GamerC...

bandwidth Min

0 — 1000

Value: 0 Unit: Mbps

latency Max

0 — 10000

Value: 2 Unit: ms

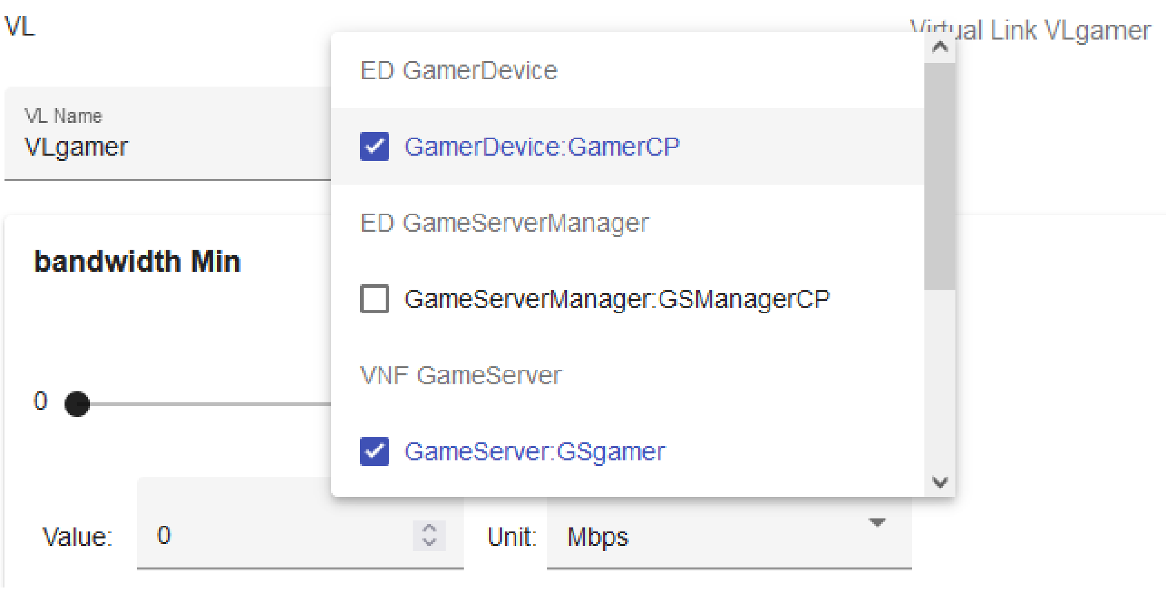
jitter Max

0 — 100

Value: 100 Unit: ms

Figure 51: Editing of the Virtual Links section.

Once the Blueprint Template editing has been completed, AMP displays a simple graph representing the network topology relations between the Blueprint Template objects (VNFs, CPs, VLs and Externals). *Figure 53* shows the screenshot of the Blueprint Template described in this chapter.


VL

Virtual Link VLgamer

VL Name
VLgamer

bandwidth Min

0

Value: 0 Unit: Mbps

ED GamerDevice
- [x] GamerDevice:GamerCP

ED GameServerManager
- [ ] GameServerManager:GSManagerCP

VNF GameServer
- [x] GameServer:GSgamer

Figure 52: Detail of the multi-select drop down list for specifying Virtual Link connectivity.

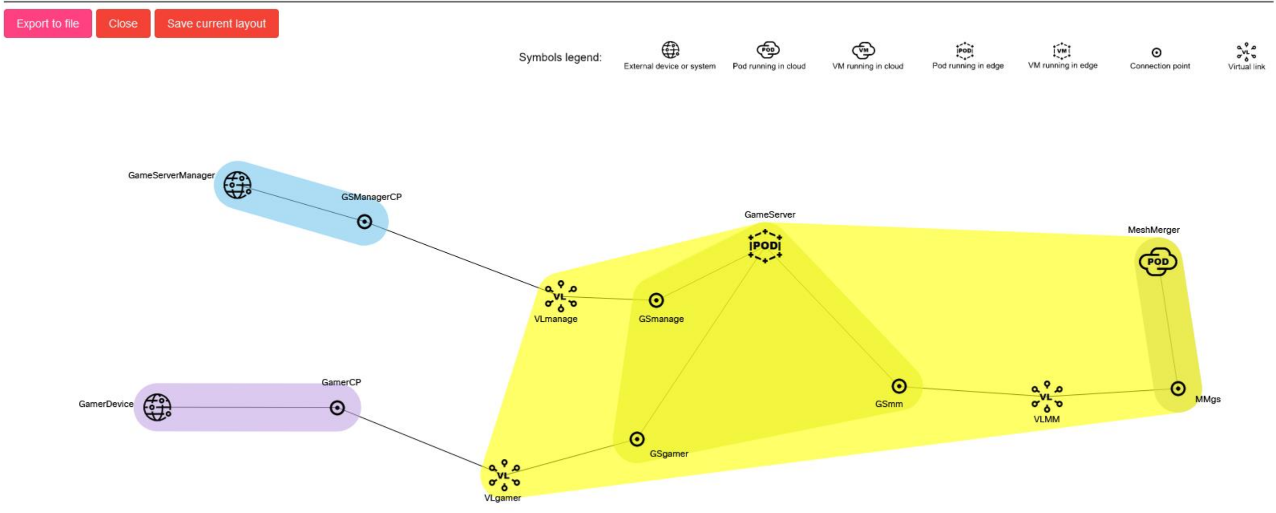
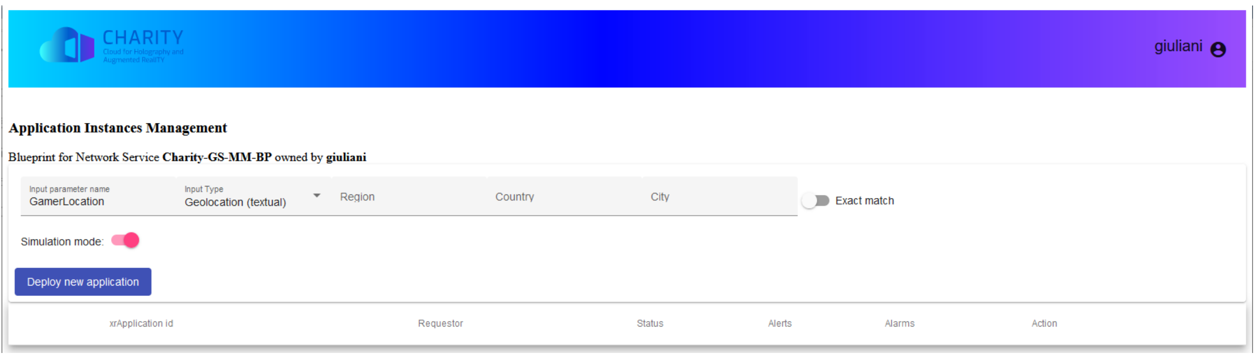


Figure 53: Blueprint Template simple topology graph.

After the design phase, AMP also guides the deployment phase of XR applications, from the Application Management Web GUI page. *Figure 54* shows a screenshot of this page for the previously described Blueprint Template, notice the presence of the input variable GamerLocation, whose value needs to be filled in to provide the Orchestrator geo-location information about the clients to allow the placement of EDGE components optimizing their network proximity.


CHARITY

giuliani

Application Instances Management

Blueprint for Network Service **Charity-GS-MM-BP** owned by **giuliani**

Input parameter name	Input Type	Region	Country	City	Exact match
GamerLocation	Geolocation (textual)				

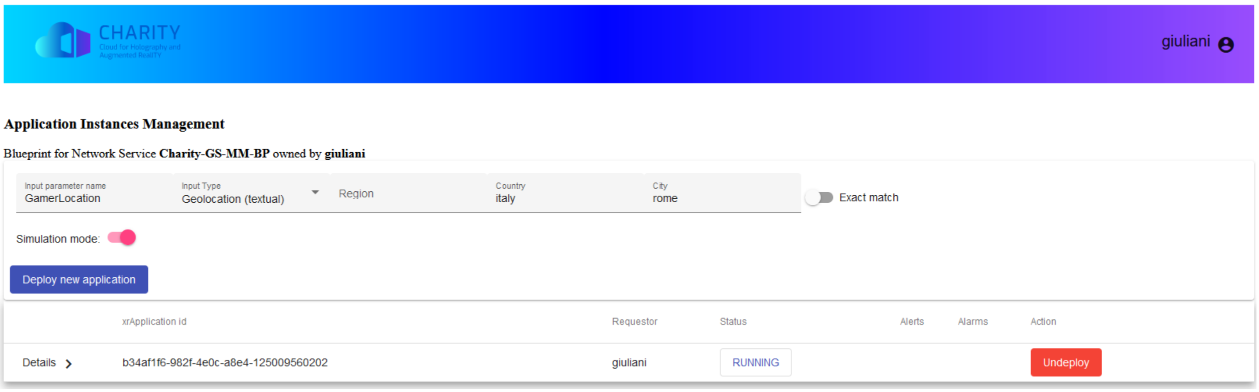
Simulation mode:

Deploy new application

xrApplication id	Requestor	Status	Alerts	Alarms	Action

Figure 54: XR Application Management AMP page.

After the input of the GamerLocation information, using the “Deploy new application” button the deployment request for the Blueprint template is requested by providing the specified input value(s). The REST API used to start the deployment (see next section) will return an ‘xrapplicationid” that can be used to retrieve further deployment status progress information, as well as output parameters (e.g. IP addresses/URLs of VNFs/CPs, etc). *Figure 55* shows a screenshot of a deployment summary.



Application Instances Management

Blueprint for Network Service **Charity-GS-MM-BP** owned by **giuliani**

Input parameter name	Input Type	Region	Country	City
GamerLocation	Geolocation (textual)		italy	rome

Exact match

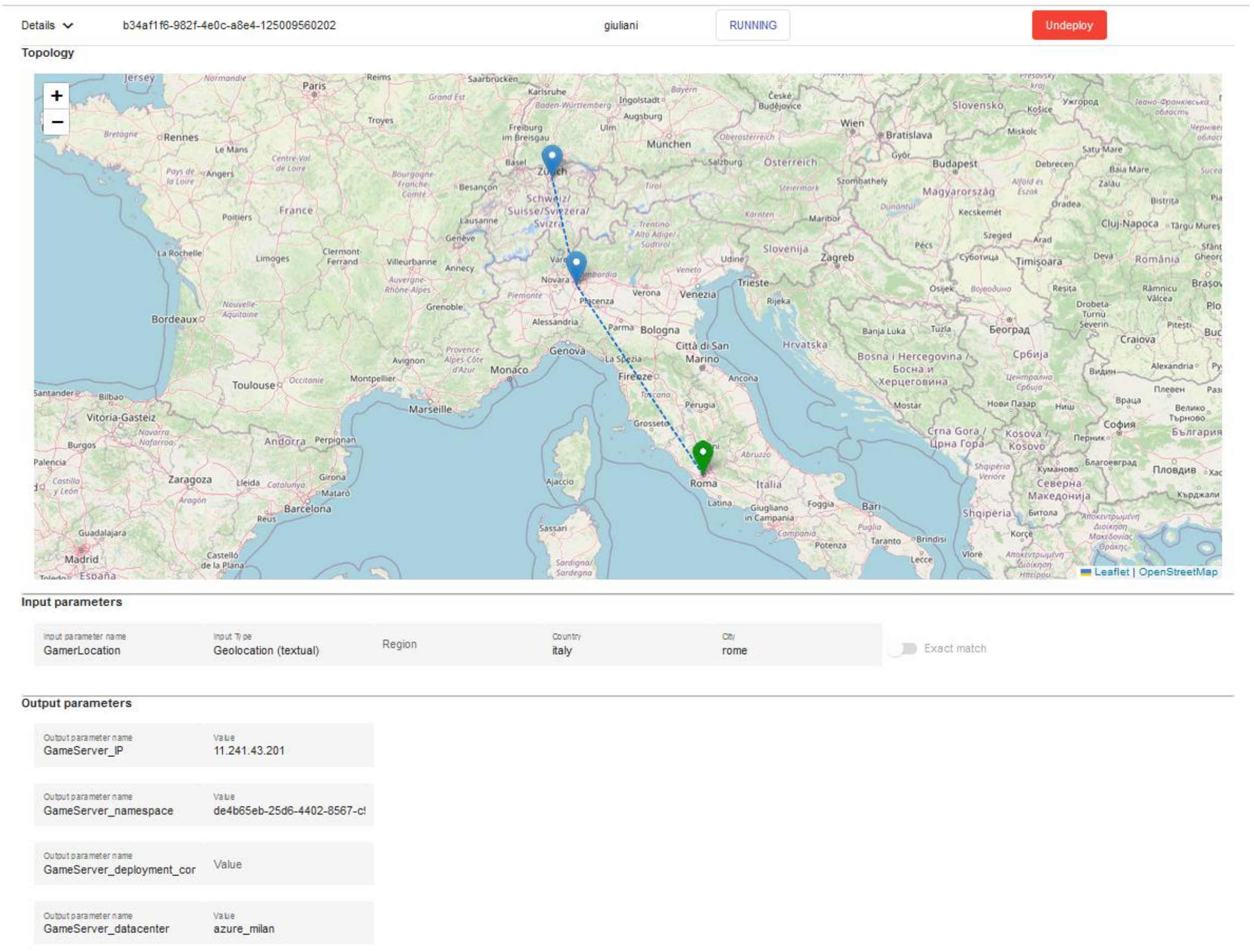
Simulation mode:

Deploy new application

	xrApplication id	Requestor	Status	Alerts	Alarms	Action
Details	b34af1f6-982f-4e0c-a8e4-125009560202	giuliani	RUNNING			Undeploy

Figure 55: XR Application deployment status summary.

Finally, the “Details” section can be expanded to display a geographical map showing where the Blueprint components have been deployed, followed by the table of the out variables, as shown in *Figure 56*.



Details b34af1f6-982f-4e0c-a8e4-125009560202 giuliani RUNNING Undeploy

Topology

Input parameters

Input parameter name	Input Type	Region	Country	City
GamerLocation	Geolocation (textual)		italy	rome

Exact match

Output parameters

Output parameter name	Value
GameServer_IP	11.241.43.201
GameServer_namespace	de4b65eb-25d6-4402-8567-c!
GameServer_deployment_cor	
GameServer_datacenter	azure_milan

Figure 56: XR Application deployment details.

Notice the green marker for the GamerLocation (“Italy/Rome” has been selected), and the blue marker for the VNFs: the GameServer in Milan datacenter, and the MeshMerger (in this example requiring a specific GPUs not available in Milan DC) deployed at Zurich DC (providing the NVIDIA GPU model).

5.3 XR Service Enabler Repository

The XR Service Enabler Repository is the CHARITY component responsible for storing and managing the VNF images developed by the XR developers to the CHARITY platform. The images are stored in the XR Service Enabler Repository via the Portal and are accessed from the Blueprint Editor page when defining a Blueprint. The repository is also accessed at deployment time by the Orchestration and Deployment layer, to retrieve the images to be deployed on the selected nodes.

CHARITY being a platform targeting cloud-native applications, the majority of the images implementing a VNF are expected to be containers. For this reason, the XR Service Enabler Repository relies on Harbor, a well-established and universally adopted open-source registry that secures artifacts with policies and role-based access control.

The XR Service Enabler Repository provides a wrapping layer around Harbor, to allow developers to enrich the information related to VNF images with additional meta data that Harbor does not allow to specify. To support this scenario, the XR Service Enabler Repository tracks the information related to the VNF images created by XR developers in a private database (NoSQL e.g., MongoDB).

The integration between the XR Service Enabler Repository and the AMF happens via REST API at microservice backend level (described later in this section). The access to Harbor is mediated via the same Keycloak Authorization and Authentication Realm used for the Portal.

5.4 XR Service Blueprint Template Repository

The XR Service Blueprint Template Repository is the place where the Blueprint Templates defined by XR developers are stored. The templates created are stored in a private, non-relational MongoDB, which is accessed via a REST API provided by a dedicated microservice.

Template data are stored in the database in the form of JSON objects, as this format is the most suitable for communication and integration with web interfaces.

At deployment time, the JSON representation of the Blueprint Template is translated into the TOSCA model of the application, which is the format understood by the orchestration layer.

5.5 Backend Microservices

The functionalities exposed to end users via the GUI are enabled by a set of microservices, representing the backend layer and each one dedicated at a specific business function.

5.5.1 Microservices Architecture

The microservices implemented so far are providing their services based on REST communication, exposing to the GUI layer all the operations needed to create, update, get, delete data required to implement the business function they are dedicated to. They are stateless, where the data persistence is delegated to external database or file storage systems.

5.5.2 Selected Technologies

The microservices implemented so far are based on the following technologies:

- Spring boot, an open-source framework based on Java which simplifies the development of micro services and their packaging into container images.
- REST API, a Web API conforming to the REST architectural style (REST is an acronym for REpresentational State Transfer and an architectural style for distributed hypermedia systems).

- MongoDB Community Edition, an open-source NoSQL database management program that can manage document-oriented information.
- Keycloak connectors for enabling Keycloak-based Authorization and Authentication to microservices endpoints.

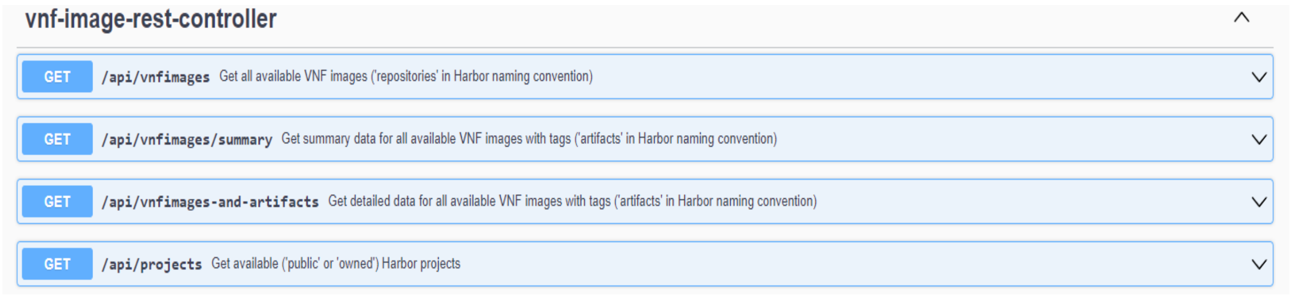
5.5.3 VNF Image Microservice

The VNF Image microservice provides means to manage all the information needed by the XR Service Developer for the representation and storage of a VNF Image within the CHARITY ecosystem, in connection with the XR Service Enabler Repository. The core of a VNF Image is the container created by the developer outside CHARITY. This core image is then enriched with metadata relevant for the CHARITY platform and uploaded into the CHARITY Harbor registry within the XR Service Enabler Repository, so that it will be available:

- to XR developers, when defining a Blueprint Template for an XR Service.
- to the CHARITY Orchestration layer, at runtime when the images must be retrieved, deployed and started in the form of containers in the CHARITY Cloud or Edge nodes.

Figure 57 depicts some of the operations provided by the VNF Image microservice:

- getting all the VNF images available to the user, depending on their permissions, in the repository.
- getting a summary data for available images.
- getting detailed data for the images and tags.
- getting available Harbor projects.


vnf-image-rest-controller

GET /api/vnfimages Get all available VNF images ('repositories' in Harbor naming convention)

GET /api/vnfimages/summary Get summary data for all available VNF images with tags ('artifacts' in Harbor naming convention)

GET /api/vnfimages-and-artifacts Get detailed data for all available VNF images with tags ('artifacts' in Harbor naming convention)

GET /api/projects Get available ('public' or 'owned') Harbor projects

Figure 57: Sample list of REST 'GET' operations provided by the VNF image microservice.

The enrichment of VNF Image information with additional metadata and the storage of such representation is stored into a private MongoDB instance.

5.5.4 Blueprint Microservice

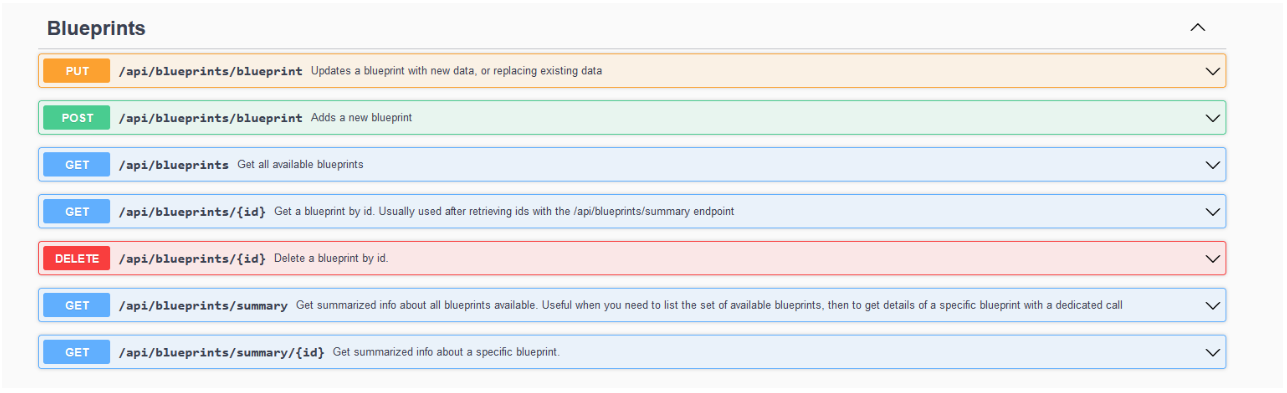
The Blueprint microservice provides all the endpoints and functionalities to create, update, delete, persist the representation of a service Blueprint Template. Based on the information provided by the XR Service Developer via interaction with the Web GUI, a JSON representation of the Blueprint is created and stored into a private MongoDB instance.

The JSON representation of the Blueprint Service will be then translated, when needed (upon request or at deployment request time), into a TOSCA representation which is suitable for the Orchestration layer.

Figure 58 shows the endpoints provided by the Blueprint microservice for:

- updating an existing Blueprint Template.
- adding a new Blueprint Template.
- getting all available Blueprint Templates (filtered by permissions).
- getting a specific Blueprint Template.
- deleting a blueprint.

- getting summary data on Blueprint Templates.


Blueprints

PUT /api/blueprints/blueprint Updates a blueprint with new data, or replacing existing data

POST /api/blueprints/blueprint Adds a new blueprint

GET /api/blueprints Get all available blueprints

GET /api/blueprints/{id} Get a blueprint by id. Usually used after retrieving ids with the /api/blueprints/summary endpoint

DELETE /api/blueprints/{id} Delete a blueprint by id.

GET /api/blueprints/summary Get summarized info about all blueprints available. Useful when you need to list the set of available blueprints, then to get details of a specific blueprint with a dedicated call

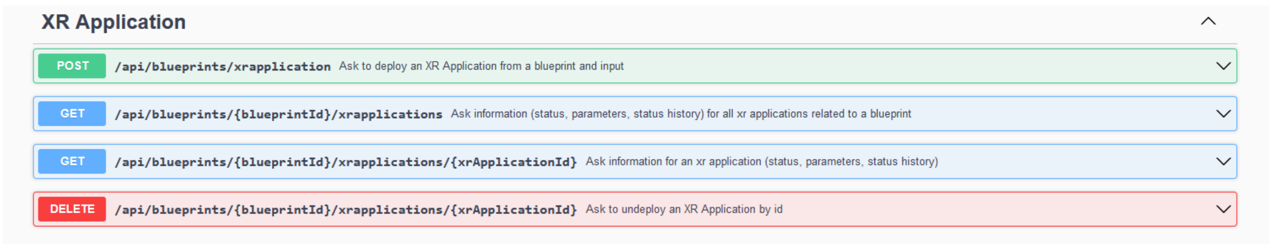
GET /api/blueprints/summary/{id} Get summarized info about a specific blueprint.

Figure 58: Endpoints of the Blueprint microservice REST API.

5.5.5 XR Application Microservice

This microservice is responsible for the management of XR applications, i.e. deployment instance of Blueprint templates. *Figure 59* shows the endpoints provided by the XR Application microservice for:

- Deployment of a new XR Application from a Blueprint Template, providing (if configured) a set of input variables values specific for the new instance
- Retrieve information about one (or all) XR applications: deployment status, oputput parameters, status history, alarms and alerts)
- Undeploy an XR Application instance


XR Application

POST /api/blueprints/xrapplication Ask to deploy an XR Application from a blueprint and input

GET /api/blueprints/{blueprintId}/xrapplications Ask information (status, parameters, status history) for all xr applications related to a blueprint

GET /api/blueprints/{blueprintId}/xrapplications/{xrApplicationId} Ask information for an xr application (status, parameters, status history)

DELETE /api/blueprints/{blueprintId}/xrapplications/{xrApplicationId} Ask to undeploy an XR Application by id

Figure 59: Endpoints of the XR Application microservice REST API.

Notice that the deployment of an XR application is a long lasting operations, thus the REST API will not wait for its termination, rather it will quickly return an “xrapplicationid” to be used with the information queries to retrieve deployment status progess information.

5.6 TOSCA Model for Blueprint Templates

As introduced in **Section 3**, the TOSCA Model allows to define an application Blueprint Template through a TOSCA-compliant representation[25].

The Model has been defined taking into account the capabilities of the TOSCA specification, focusing in particular on the Tosca Simple Profile v1.3 specification[26].

Citing from the TOSCA Simple Profile definition written by the OASIS Standards dedicated group:

[25] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

[26] https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/os/TOSCA-Simple-Profile-YAML-v1.3-os.html

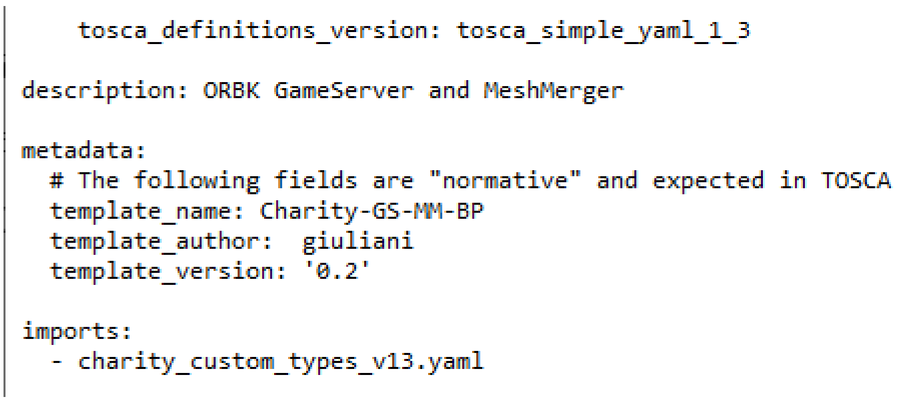
"The TOSCA metamodel uses the concept of service templates that describe cloud workloads as a topology template, which is a graph of node templates modeling the components a workload is made up of and of relationship templates modeling the relations between those components. TOSCA further provides a type system of node types to describe the possible building blocks for constructing a service template, as well as relationship types to describe possible kinds of relations. Both node and relationship types may define lifecycle operations to implement the behavior an orchestration engine can invoke when instantiating a service template [...]. The TOSCA simple profile assumes a number of base types (node types and relationship types) to be supported by each compliant environment such as a 'Compute' node type, a 'Network' node type or a generic 'Database' node type. Furthermore, it is envisioned that a large number of additional types for use in service templates will be defined by a community over time. Therefore, template authors in many cases will not have to define types themselves but can simply start writing service templates that use existing types. In addition, the simple profile will provide means for easily customizing and extending existing types [...]".

In the CHARITY scenario, we selected the TOSCA Simple Profile v1.3 specification[26] as the basis to represent the XR Services and the relationships between the components, while we defined custom types to tailor the TOSCA capabilities to the specific needs of the XR Applications deployment scenarios in CHARITY.

Leveraging the support for the YAML representation of the TOSCA specification, we defined such custom types in a "charity_custom_types.yaml" document, to be imported in the TOSCA model according to the specification rules. The contents of the custom type definition are documented in Appendix A.1.

In the following, we give some examples of how the TOSCA specification defined for CHARITY can be used to represent an XR Service.

Figure 60 shows an example of a TOSCA Blueprint Template header section, where general information are set and the custom types specification file is imported into the model.



```
    tosca_definitions_version: tosca_simple_yaml_1_3

description: ORBK GameServer and MeshMerger

metadata:
  # The following fields are "normative" and expected in TOSCA
  template_name: Charity-GS-MM-BP
  template_author:  giuliani
  template_version: '0.2'

imports:
  - charity_custom_types_v13.yaml
```

Figure 60: Sample TOSCA Blueprint Template header section.

Figure 61 shows an example about how to represent a CHARITY Component (a Game Server needed by a gaming use case) and a CHARITY Node (a runtime node where the component should be run). The Component section specifies what must be executed in terms of VNF image, some deployments preferences (e.g. EDGE), and the requirements in terms of runtime node.

The Node section specifies the characteristics required for a valid runtime node.

```
node_templates:
  GamerDevice:
    type: Charity.External
    properties:
      name: gamerdevice
      geolocation: { get_input: GamerLocation }
  GameServerManager:
    type: Charity.External
    properties:
      name: gameservermanager
  GameServer:
    type: Charity.Component
    properties:
      name: GameServer
      deployment_unit: K8S_POD
      placement_hint: EDGE
      image: harbor.hpe.charity-project.eu/hpe/test-image:v1-hpe
    requirements:
      - host: GameServerNode
  GameServerNode:
    type: Charity.Node
    node_filter:
      capabilities:
        - host:
            properties:
              num_cpus:
                - equal: 4
              mem_size:
                - greater_than: 4000 MB
```

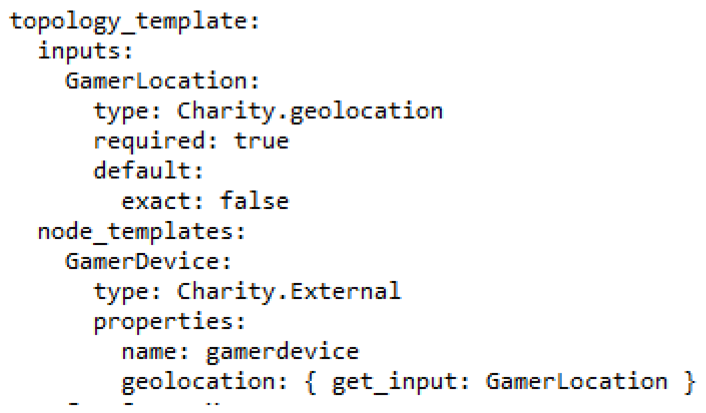
Figure 61: Sample representation of a CHARITY Component and a CHARITY Node.

Figure 62 shows an example similar to the previous one, specifying a container image as VNF image type, bur suggesting a CLOUD deployment, and the request for a specific GPU.

```
MeshMerger:
  type: Charity.Component
  properties:
    name: MeshMerger
    deployment_unit: K8S_POD
    placement_hint: CLOUD
    image: harbor.hpe.charity-project.eu/hpe/test-image:v1-hpe
  requirements:
    - host: MeshMergerNode
MeshMergerNode:
  type: Charity.Node
  node_filter:
    capabilities:
      - host:
          properties:
            num_cpus:
              - equal: 8
            mem_size:
              - greater_than: 12000 MB
      - gpu:
          properties:
            model:
              - equal: NVIDIA_GEFORCE_RTX_3080
            dedicated:
              - equal: true
            units:
              - equal: 1
```

Figure 62: Another sample representation of a CHARITY Component and a CHARITY Node.

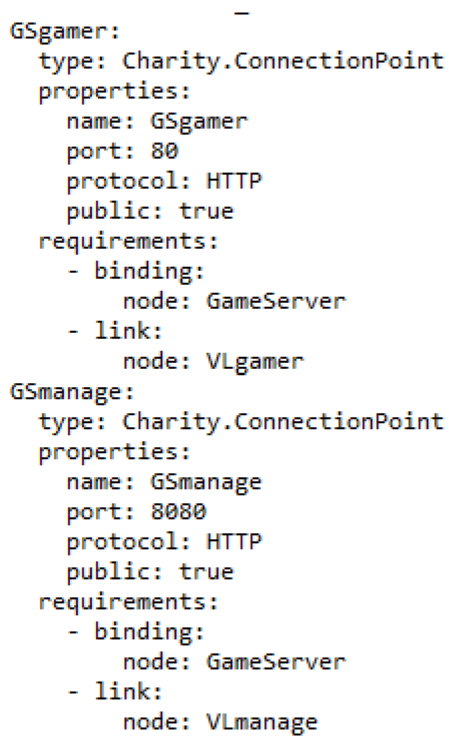
Figure 63 shows how to represent an "external device". This kind of representation may be required to model the interaction with an external system that has to pass some parameters (in the example the location) to the application deployed by CHARITY.



```
topology_template:
  inputs:
    GamerLocation:
      type: Charity.geolocation
      required: true
      default:
        exact: false
  node_templates:
    GamerDevice:
      type: Charity.External
      properties:
        name: gamerdevice
        geolocation: { get_input: GamerLocation }
```

Figure 63: Representation of an external device.

Figure 64 shows how to represent connection points, which are elements needed to define how a VNF communicates with other VNFs or systems. In the most simplified representation, they specify the port numbers used by a VNF for inputs and outputs, and the names of the (virtual) links that connect such ports.



```
GSgamer:
  type: Charity.ConnectionPoint
  properties:
    name: GSgamer
    port: 80
    protocol: HTTP
    public: true
  requirements:
    - binding:
        node: GameServer
    - link:
        node: VLgamer
GSmanage:
  type: Charity.ConnectionPoint
  properties:
    name: GSmanage
    port: 8080
    protocol: HTTP
    public: true
  requirements:
    - binding:
        node: GameServer
    - link:
        node: VLmanage
```

Figure 64: Representation of connection points.

Figure 65 shows how to represent a virtual link, which is the communication link between two VNFs or with external system. The virtual link can be qualified with requirements in terms of quality of service parameters, e.g., for specifying bandwidth or latency requirements.

```
  VLMM:
    type: Charity.VirtualLink
    properties:
      name: VLMM
    node_filter:
      capabilities:
      - network:
          properties:
            bandwidth:
            - greater_or_equal: 10000 Mbps
            latency:
            - less_than: 10000 ms
            jitter:
            - less_than: 100 ms
```

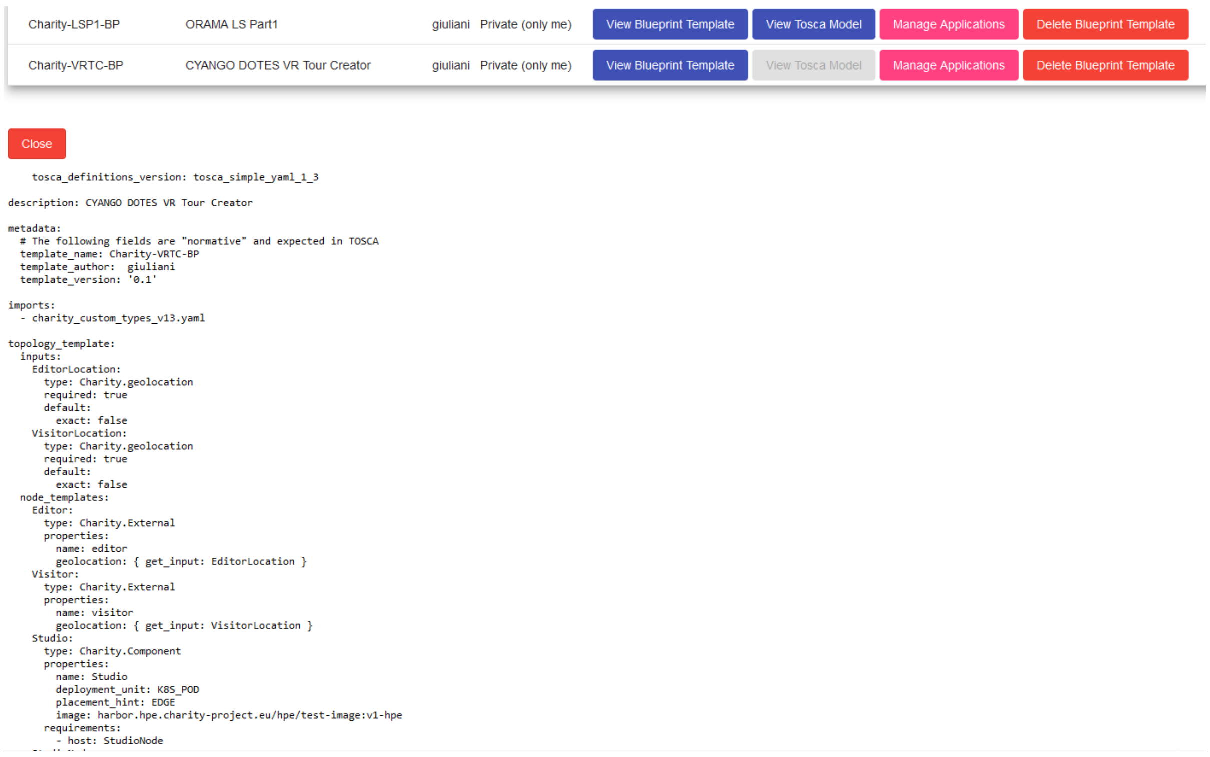
Figure 65: Representation of a virtual link with QoS requirements.

5.7 TOSCA Translation

The TOSCA Translator is a component that converts, upon a deployment request, the representation of the application Blueprint created by the XR developer from the AMF internal format (JSON) into the TOSCA format defined for CHARITY. The design principles behind this component are:

- It is a dedicated microservice, specific for the translation from the JSON representation of a Blueprint Template into a TOSCA YAML representation.
- The translation is be based on a templating engine (e.g., Jinja in case of a microservice based on a Python stack, or Jinjava for Java based microservices, depending on the selected technologies for implementation), with placeholders and extended capabilities for expressing variables, statements and expressions.
- It provides a REST API to the Web GUI layer to perform translation requests

Trevious sections TOSCA snippets were extracted from Web GUI snapshots, provided by AMP using this API: *Figure 66* displays a sample view of this feature.



Charity-LSP1-BP	ORAMA LS Part1	giuliani	Private (only me)	View Blueprint Template	View Tosca Model	Manage Applications	Delete Blueprint Template
Charity-VRTC-BP	CYANGO DOTES VR Tour Creator	giuliani	Private (only me)	View Blueprint Template	View Tosca Model	Manage Applications	Delete Blueprint Template

```
    tosca_definitions_version: tosca_simple_yaml_1_3

description: CYANGO DOTES VR Tour Creator

metadata:
  # The following fields are "normative" and expected in TOSCA
  template_name: Charity-VRTC-BP
  template_author:  giuliani
  template_version: '0.1'

imports:
  - charity_custom_types_v13.yaml

topology_template:
  inputs:
    EditorLocation:
      type: Charity.geolocation
      required: true
      default:
        exact: false
    VisitorLocation:
      type: Charity.geolocation
      required: true
      default:
        exact: false
  node_templates:
    Editor:
      type: Charity.External
      properties:
        name: editor
        geolocation: { get_input: EditorLocation }
    Visitor:
      type: Charity.External
      properties:
        name: visitor
        geolocation: { get_input: VisitorLocation }
    Studio:
      type: Charity.Component
      properties:
        name: Studio
        deployment_unit: K8S_POD
        placement_hint: EDGE
        image: harbor.hpe.charity-project.eu/hpe/test-image:v1-hpe
      requirements:
        - host: StudioNode
```

Figure 66: Example of display of TOSCA model.

6 Algorithms for Service Orchestration

6.1 Deterministic Networking in Service Orchestration

Optimal SFC (Service Function Chain) embedding onto physical networks has drawn much attention in the recent literature. However, most of the existing work in the literature focused on maximizing network throughput/resource efficiency, regardless latency requirements; whereas maximizing network resource efficiency while keeping the service latency and latency variation (jitter) within a deterministic bound has received less attention. In traditional networks, the end-to-end latency/jitter curves have a wide probability distribution with a long tail. A deterministic orchestration mechanism would ensure bounded end-to-end latency and delay variation with no long tails in an end-to-end converged network; ultimately supporting deterministic services such as XR services. Also, it is important to increase the profits of service providers by optimizing the resource allocation and placement of VNF instances while ensuring deterministic latency performance. Moreover, due to the highly dynamic nature of network traffic load, it is a challenge to embed SFC requests with deterministic latency bounds and lower jitter (i.e., SFC of XR services) during the SFC lifetime.

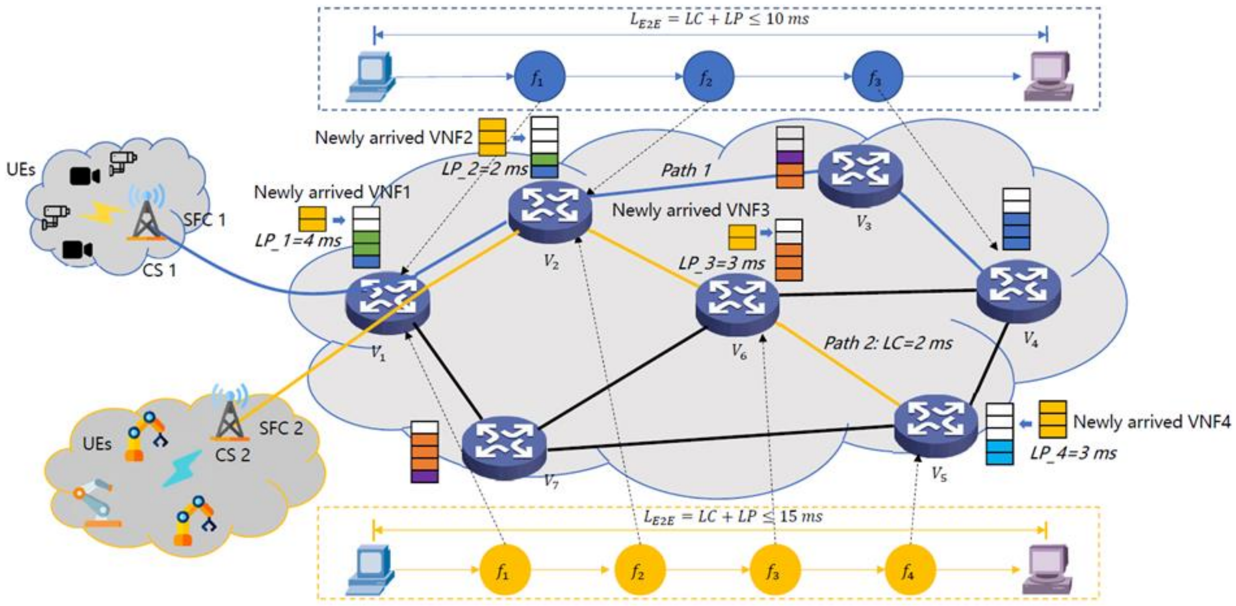


Figure 67: Example of SFC requests in 5G edge networks [27].

Figure 67 shows an example of SFC request embedding in a 5G network. Edge nodes are equipped with a certain amount of processing resources and switching ability. At cell sites side, user devices generate SFC requests over time, and the SFC requests are featured with latency requirements, which are comprised of communication latency and VNF processing latency. These SFC requests can be also made in response to a request from XR application developers. Once a new SFC request arrives, optimal path selection and processing resource allocation should be performed in order to meet the latency requirement of the XR service supported by the SFC. For example, when a request for SFC 2 with an end-to-end latency requirement of 15ms arrives, considering the network load status and distance between source and destination nodes, Path 2 would be selected with communication latency of 2ms. Then the VNF processing budget would be 13ms. Therefore, it is important to allocate an appropriate amount of processing resources to each VNF composing the SFC in order to make sure the latency would remain less than 13ms while minimizing the cost of VNFs processing.

To achieve deterministic orchestration, and as detailed in [27], we followed an approach that consists in separating this problem into two distinct sub-problems: (i) how to optimize the resource allocation and path selection for SFC deployment; (ii) how to adjust resource allocation to ensure the bounded service latency and jitter under the traffic variation, which can ultimately maximize the overall incomes for ISP. These two aspects form the whole procedure of lifetime management for SFCs. The first sub-problem is to be solved in two directions: (i) improving service acceptance ratio (i.e., increase the revenue derived from providing services to users); (ii) reducing resource consumption by optimizing resource allocation to VNF instances (i.e., reduce the network cost for ISPs). Given that propagation and transmission latency can be considered as being deterministic, we need to bound the non-deterministic VNF processing latency in order to achieve an overall deterministic end-to-end latency and jitter for time-sensitive services. For the second sub-problem, we investigate the optimal VNF scaling up/down scheme in response to the traffic variation to keep the bounded latency by considering the historical network load, which shall help avoid resource bottleneck and reduce network congestion.

In [27], the problem was formulated as follows. Given a physical network topology and a set of SFC requests, we need to determine: 1) the path between source and destination nodes and place VNF instances along the path, 2) the right amount of processing and bandwidth resources for corresponding VNFs and traffic, 3) how to adjust resource allocation when traffic load varies; while maximizing the profits of the ISP from running SFC requests and ensuring deterministic e2e latency performance. The traffic of SFC request will traverse a series of ordered VNF instances and this selected path of nodes will influence the resource consumption on edge nodes and physical links. How to select suitable paths and allocate resources for SFC requests remain a challenge for deterministic latency performance and maximum resource efficiency.

As stated previously, the Lifecycle management procedures of the SFCs are divided into two phases: SFC deployment and SFC adjustment, which are solved by the proposed Det-SFC deployment (Det-SFCD) and the Det-SFC adjustment (Det-SFCA) algorithms, respectively. In SFC deployment phase, 1) optimal paths need to be selected to avoid the resource bottleneck when deploying SFCs, ultimately increasing SFC acceptance rate; 2) VNF instances need to be sized optimally to minimize the resource costs while ensuring the latency requirements. In the SFC adjustment phase, optimal VNF instance scaling up/down scheme should be designed so latency variation would be controlled within a small range.

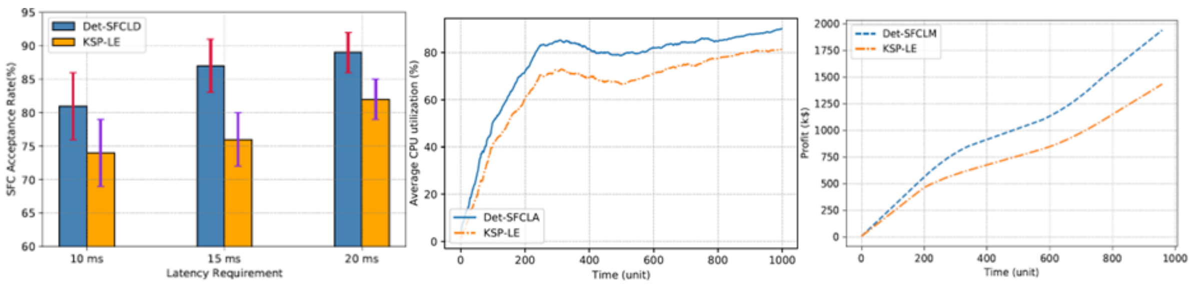


Figure 68: Performance evaluation of DET-SFCD and DET-SFCA algorithms [27].

The conducted performance evaluation (*Figure 68*) showed that the proposed algorithms achieved more than 15% enhancement in SFC acceptance rate and an average 35 % more overall profits in comparison to the baseline solution. Also, Det-SFCA results in a higher utilization. Without considering deployment cost, the baseline solution exhibits a lower utilization of CPU resources by rejecting more SFC requests due to resource bottleneck. This is due to the fact that during the adjustment phase, the resource adjustment is performed without taking into account the history of network load dynamics.

6.2 Service Placement & Resource Scheduling

Managing and allocating resources for the network processes and functions is an important aspect to XR applications. This is mainly due to the fact that XR applications compose tasks of image processing, high-quality display, resource-hungry computations, and faster packet forwarding. Further, onboarding and scaling these complex ecosystems of cloud-native applications, based on microservices, is also an important factor to consider.

Traditional network resource management involves a simple system model and low-level design where an application determines the amount of network resources, for example, bandwidth needed for the data flow of the application. The network manager acts as a reservation system to allocate the computed requirement of the resources based on the resource availability. This tends to be inefficient especially for applications with distributed service placements.

An alternative approach is, instead of directly specifying the network resources to the network manager, the application can submit the requirements in terms of both constraints and objectives. The network then calculates the optimal resources based on the requirements. While this reduces the load on the application, this increases the burden of in-network compute to calculate the optimal allocation. Additionally, the application could end up sharing some of its proprietary information with the network.

Given the distributed nature and heterogeneity of resources from one side and the distribution of XR services across network elements on the other side, it is not trivial to use the existing datacenter resource scheduling techniques without a careful tailoring to requirements of CHARITY. For example, *Figure 69* shows a Deep Learning (DL) task within an XR application. The variants of the request can be the devices requesting, accuracy and latency requirements of the DL model. Based on these goals and constraints in terms of load spread across the network devices, the decision needs to be made at runtime.

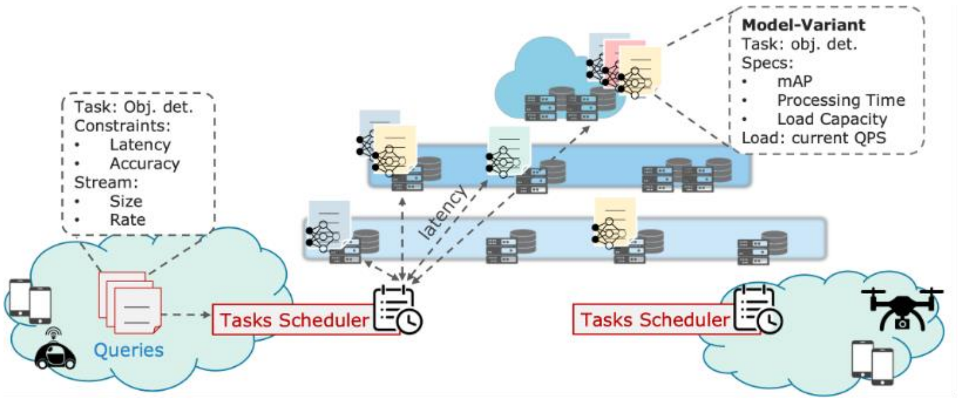


Figure 69: Request scheduling for optimal resource allocation at real time.

To this end, we are building a learning paradigm based on uncertain network dynamics and algorithms that can learn and adapt to the environment based on resource availability. We call this Adaptive Scheduling of Edge Tasks (ASET), which runs a smart RL agent trained using real-world network topology and identify the best policy to schedule the workloads in a network leveraging Deep Reinforcement Learning (DRL) techniques. The policy can be as simple as executing a task at the closest edge cluster to schedule based on latency and load at real time.

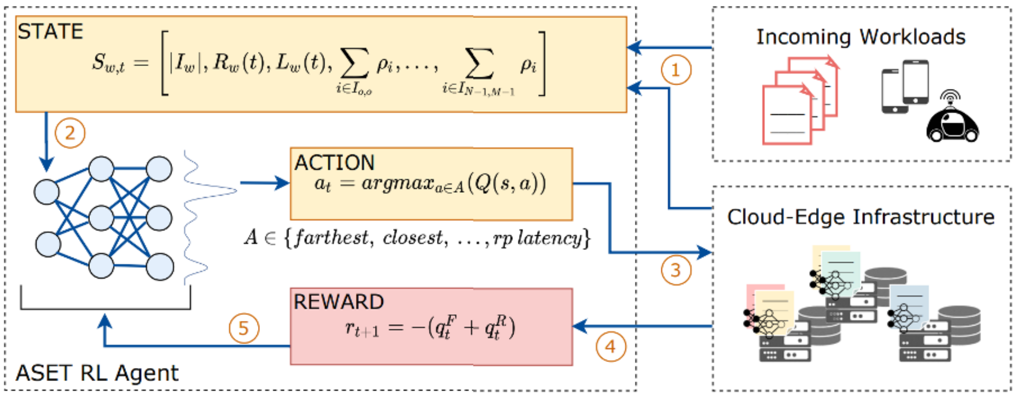

STATE
$S_{w,t} = \left[|I_w|, R_w(t), L_w(t), \sum_{i \in I_{o,o}} \rho_i, \ldots, \sum_{i \in I_{N-1,M-1}} \rho_i \right]$
ACTION
$a_t = argmax_{a \in A}(Q(s,a))$
$A \in \{farthest,\ closest,\ \ldots, rp\, latency\}$
REWARD
$r_{t+1} = -(q_t^F + q_t^R)$
ASET RL Agent
Incoming Workloads
Cloud-Edge Infrastructure
1
2
3
4
5

Figure 70: Adaptive Scheduling of Edge Tasks (ASET) workflow.

Our adaptive scheduling approach aims to learn the optimal policy depending on current system conditions, e.g., current applications, network topology, and stream arrivals that vary over time. Due to the lack of labelled data, the optimal policy learning is formulated as a RL problem; hence, an intelligent agent tries to learn the optimal policy selection strategy according to the observed state of the environment. This is accomplished by an RL policy that estimates a probability distribution of each possible action (policy selection) that cumulatively maximizes a reward (typically maximizing the fraction of queries that are served successfully), as shown in *Figure 70*.

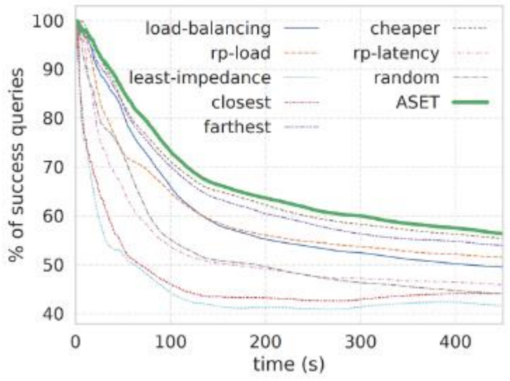


Figure 71: Percentage of successful queries over time for ML task with users arriving in real-world pattern.

Initial results suggest, even with partial view of the network resources, ASET performs better than traditional scheduling mechanisms (*Figure 71*). We simulate the scenario of users arriving in real-world pattern. The work is still on-going to implement better selection of policies through multi-agent communication, security & privacy-aware and real-world deployments. Furthermore, we envisage to integrate the scheduler with AIRO to render both smart orchestration and scheduling for XR applications facilitating cloud-edge continuum.

6.3 Decentralized Service Replica Management

The pace in the adoption of Edge Computing is rapidly increasing in the attempt to bring computation as close as possible to the data producers and consumers (e.g., end-users) [28][29] . In principle, the transition to the Edge showcases exciting properties: it is cost-effective for the application providers while being more convenient for the end-users, who enjoy more personal applications. For example, interactive applications (such a multiplayer games and VR) have strong requirements on latency to keep their immersiveness and might benefit from being placed at the Edge.

However, for these properties to hold, applications must be placed and replicated correctly by matching their requirements with resource capabilities and the position of the users. Such a process can result in the spawn of many replicas of the same service in different resources of the edge platform, which can rapidly erode the promised cost-effective benefit. Also, the problem becomes even more challenging when considering that application requirements and resource capability can change over time, even when the application is running and accessed by the moving end-users. Therefore, there is a need to adapt the application placement during runtime to maintain the promises regarding the quality of experience.

In the context of CHARITY, we consider that the system at-the-edge is made of entities with a specific geographic location representing a small-to-mid pool of potentially heterogeneous resources (i.e., EM – Edge Miniclouds). For the sake of simplification, one can model a system in which each EM is modelled as the sum of all its available resources. EMs build a decentralized system by employing a set of point-to-point communication with each other. Users of the system request the services provided by a set of applications. Each application offers a different type of service. Several application instances (or replicas) can be deployed on several EMs, based on users' requests and QoE constraints. Each EM supervises the execution (and all the related optimization aspects) of the application replicas received.

In this context, we are interested in an approach that, proactively and in a decentralized fashion, controls the number of application replicas of the same service in an Edge Computing platform, while meeting the requested QoE promises. The approach should decide which replicas to maintain active and to ensure that the matching of application requirements and resource capability is respected. This approach can also trigger live migration of an application between resources of the Edge platform, by considering the impact of migration on the quality of service.

As a first step to study and evaluate our approach, we plan to use extensive simulations to validate our approach within CHARITY framework, but it is a line of investigation that it is still open. Our idea is to use the PureEdgeSim simulator [30], a discrete-event simulator for Edge environments. The simulated scenario would consist of a federation of EMs. Each EM is composed of a heterogeneous set of resource-constrained edge devices and servers, able to host various types of applications. Each EM can be simulated as a single aggregated entity with a PureEdgeSim Datacenter object with a capacity equal to the sum of the resources of the devices and servers that compose it. Each user device can be simulated with a PureEdgeSim EdgeDevice object (e.g., a tablet or a smartphone). An early experiment along this path is reported in [31].

6.4 Managing Network Slice Mobility for 6G Networks

Efficient network resource provisioning should be investigated to support the strict latency and bandwidth requirements of future 6G XR services, such as in holographic communications and industrial automation scenarios. Compared with 5G, 6G will face more challenging situations, including an open communication ecosystem, higher network management levels, and ubiquitous intelligence. The role of network slices in supporting diverse 6G use cases and the importance of managing these slices to maintain service level agreements (SLAs) should be investigated. The network slices are foreseen as a network solution to support advanced XR users and applications with strict latency requirements. To achieve the necessary SLAs, network slices are provisioned with specific resources that are coupled with services, as well as the corresponding mobile users, during the whole lifetime of the slices. Especially for some dynamic scenarios where the user demands rise or drop, it needs strict slice adjustment and migration strategies, for both the network resources and services running on them, which can be referred to as network slice mobility (NSM). While a slice is migrated from one service region to another, all these factors, i.e., service migration and network resource migration, must be considered to guarantee service continuity.

We devise a two-level network architecture consisting of a physical layer and a slice layer [32]. It defines the state of MEC servers and slices and introduces two types of triggers for NSM: Slice Resource Trigger (SRT) and Slice Migration Trigger (SMT). The primary objective of designing SRT is to monitor the slices themselves, allowing for more flexibility and exploration of a wider variety of new activities, such as scale up/down operations. The SMT trigger may result in slice migration based on the CPU and RAM utilization of the MEC server if the current residual resources in a MEC server cannot support the scaling up of some slices. We then formulate the NSM problem as a distributed system with multiple MEC servers running various network slices, aiming to maximize system profits over time.

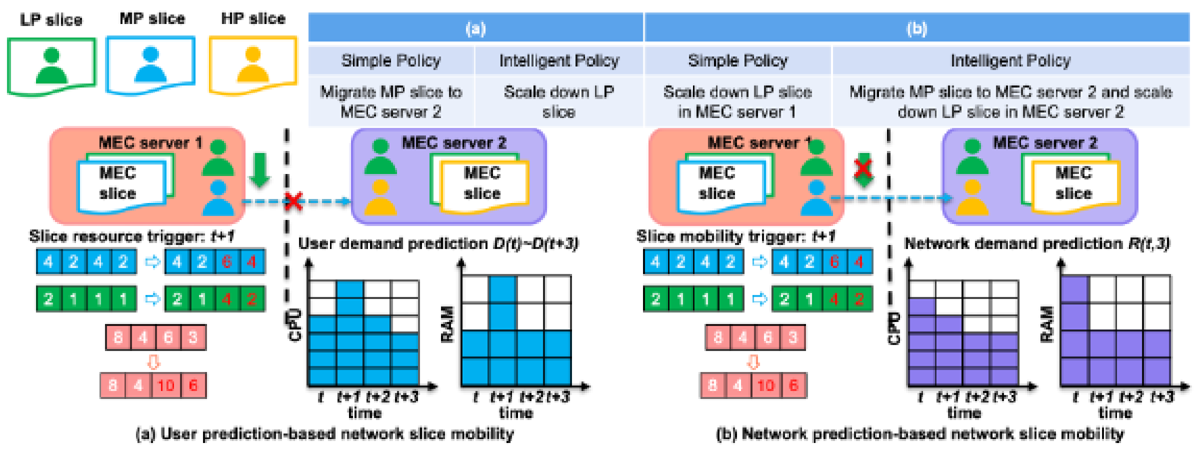

LP slice
MP slice
HP slice
(a)
(b)
Simple Policy
Intelligent Policy
Migrate MP slice to MEC server 2
Scale down LP slice
Scale down LP slice in MEC server 1
Migrate MP slice to MEC server 2 and scale down LP slice in MEC server 2
MEC server 1
MEC server 2
MEC slice
Slice resource trigger: t+1
Slice mobility trigger: t+1
User demand prediction D(t)~D(t+3)
Network demand prediction R(t,3)
CPU
RAM
t t+1 t+2 t+3
time
(a) User prediction-based network slice mobility
(b) Network prediction-based network slice mobility

Figure 72: The motivational example of prediction based NSM scheme [32].

Different from the traditional solution, we propose a scheme that leverages user and network demand predictions to determine NSM policies. It assumes a time-slotted system and use traffic prediction methods to obtain accurate prediction information. The scheme prioritizes slices based on their importance and uses the prediction information to decide on scaling up/down or migrating slices to different MEC servers. As shown in *Figure 72*, two motivational examples are presented to illustrate the prediction-based slice mobility scheme. Three types of network slices with high priority (HP), medium priority (MP) and low priority (LP) exist in the networks. In *Figure 72(a),* the MP slice needs to be scaled up at time t+1 due the increasing resource demands of users, i.e., (8, 4), and the resource demand of LP slice is also increasing in the following time slots. Due to the physical resource capacity constraints, there are not enough network resources, e.g., CPUs, for this scaling up operation. Thus, an intuitive policy might be migrating this MP slice to MEC server 2, which still has adequate network resources. However, if we consider the future resource demand of this MP slice, we will find there is a downward tendency of resource demand in the next time slot, e.g., from t + 2. As a result, an intelligent policy should be scaling down the LP slice in MEC server 1 at time t+1 since slice migration operation will also induce some cost. In the other case, if the resource demand of an MP slice increases continuously in t + 1, t + 2..., there will be resource competition in the next time slots with LP slices for a relatively long time, which will induce much degradation cost for this LP slice. At the same time, the total resource demands of MEC server 2 present a decreasing tendency from t + 1 as shown in *Figure 72(b)* if we have the prediction information of all the slices in MEC server 2. Under this circumstance, an intelligent policy should be migrating the MP slice towards MEC server 2 to accommodate the increasing user demands, eventually minimizing the total system cost.

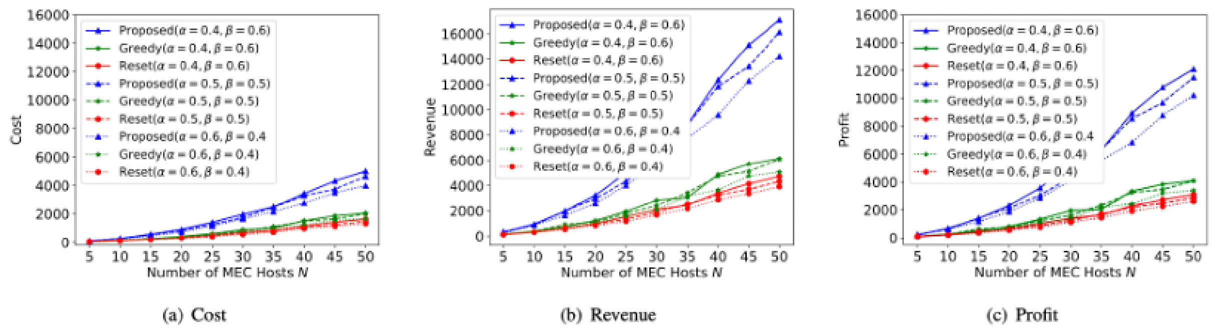


Figure 73: The cost, revenue, and profit of the considered system versus N [32].

We compare the proposed scheme with two benchmark solutions: 1) reset scheme: the slices in t scale up/down based on user demand, and the slices cannot migrate to other MEC servers even the resources are exhausted and 2) greedy scheme: the slices scale up/down firstly, if the resources are exhausted, the slices greedily migrate to the nearest MEC servers of its users. The performance is evaluated in terms of system cost, revenue, and profit. The results show that the proposed scheme outperforms the benchmarks, with higher overall profits due to efficient resource utilization and lower system latency. We set α and β as the cost coefficient of the CPU and RAM. From *Figure 73(a)*, we can observe that the system cost of different schemes and combinations gradually increase with N, since the increasing MEC server serve much more users, who request more resources. Moreover, in *Figure 73(a)*, each scheme with lower α results in higher cost, which reflects that the requested RAM resources cost more than the requested CPU cores. In *Figure 73(b)*, we compare the revenue of different schemes with different parameter combinations. We can observe that each scheme with lower α have higher revenue, and the proposed scheme with all (α, β) combinations has much higher revenue compared to the reset scheme and greedy scheme, since the proposed scheme's strategy considers migrating all the slices to their nearest MEC server and thus achieves much lower system latency. To further evaluate the performances of different schemes, we investigate profit of these schemes, as shown in *Figure 73(c)*. We can observe that the proposed scheme outperforms other schemes in all parameter combinations. As a result, the proposed scheme averagely increases the system profit by up to 30 times and 19 times compared to reset scheme and greedy scheme, respectively.

6.5 Self-sustaining Multiple Access for Dynamic Metaverse Applications

XR services, particularly metaverse-relevant applications, pose unique challenges for network infrastructure. Efficient and reliable communication is crucial for supporting diverse applications, e.g., XR services, within the metaverse, requiring innovative solutions for managing network resources. Multiple access refers to the technique of allowing multiple users to share the same radio spectrum efficiently. In dynamic environments, where the number of active users and their demands constantly change, traditional access control methods struggle to adapt effectively.

We propose using Continual Deep Reinforcement Learning (CL-DDQL), a specific type of deep reinforcement learning, to address the dynamic nature of multiple access in metaverse applications [33]. The RL system is modelled as an agent interacting with the network environment. The agent observes network states (e.g., channel conditions, user locations) and takes actions (e.g., assigning channels to users) to maximize its reward (e.g., overall throughput). Unlike traditional DRL, CL-DDQL enables the agent to continuously learn and adapt to new situations and changing network dynamics. This is crucial for handling the evolving nature of user activity in the metaverse. In addition, double deep Q-Learning (DDQL) technique within CL-DDQL helps to address the overestimation problem commonly encountered in DRL, leading to more stable and reliable learning, as shown in *Figure 74*.

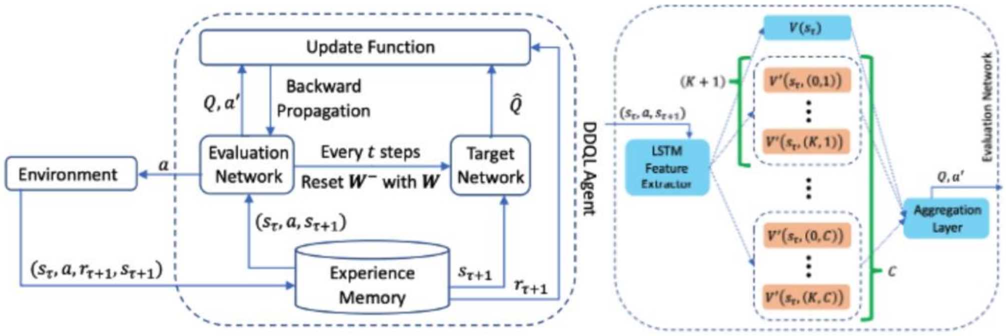


Figure 74: The DDQL agent and evaluation network of DDQL [33] .

Decentralized decision-making process is also involved in the agent training. Each user acts as an agent, learning and adapting its access strategy based on local observations and interactions with other users. The CL-DDQL algorithm helps users learn to select channels and adjust transmission power to minimize interference and maximize overall throughput for the entire network. In addition, CL-DDQL is effective in tackling the non-stationary nature of metaverse environments, where user activity and network conditions constantly change. Compared to traditional methods, the proposed approach demonstrates significantly higher throughput, especially in highly dynamic scenarios with fluctuating user numbers.

The CL-DDQL algorithm enables the agents to learn and adapt quickly, achieving optimal performance within a shorter timeframe.

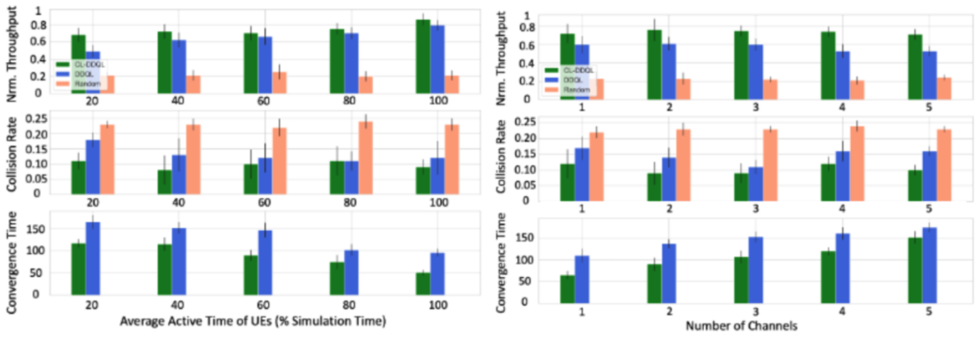


Figure 75: Normalized throughput, collision rate, and convergence time vs. (a) the average active time, (b) the number of channels of UEs for CL-DDQL, DDQL, and Random. The results are the all-time average of the values [33].

Consequently, the results demonstrate that CL-DDQL outperforms other multiple access methods in terms of throughput, particularly in dynamic scenarios with varying user densities. The system exhibits faster convergence, meaning the agents learn and adapt to new situations efficiently. The self-sustaining nature of the approach allows for efficient resource utilization and reduces the need for centralized control mechanisms. As *Figure 75(a)* demonstrates, the more frequent the context transitions, the more continual learning improves the performance. This is due to the increased likelihood of encountering repetitive contexts. In addition, the performance of CL-DDQL is hardly impacted by an increase in the rate at which contexts are transited, making it suitable for the highly dynamic environments of the Metaverse. Nonetheless, both algorithms perform better in environments with less variability. For the second experiment, *Figure 75(b)* illustrates that as the number of channels increases, the CL-DDQL algorithm becomes marginally more advantageous than DDQL. Notwithstanding, the performance of the two algorithms is not significantly impacted by the number of channels, leading us to conclude that while a greater number of channels provides more idle time slots for the agent, it also increases problem dimensions and thus the number of novel contents can be explored.

6.6 Dependency-aware Microservice Deployment in Edge Computing

The growing popularity of microservices architecture enables the development of complex applications, e.g., XR services, by breaking them down into smaller, independent services. Edge computing brings computation closer to data sources, offering low latency and improved responsiveness for resource-intensive applications. Deploying microservices efficiently in edge computing environments requires careful consideration of 1) interdependencies: microservices often rely on each other for data and functionality, necessitating coordinated deployment to ensure proper execution, 2) resource constraints: edge nodes typically have limited computational resources compared to traditional cloud data centers.

We first model the problem of microservices deployment (MSD) as the max-min joint optimization problem, which minimizing the overall system cost to promise the QoS of all the UE in the system. The problem can be regarded as a mixed binary integer linearly constrained programming (MBILP). Meanwhile, the objective can be seen as a binary integer linearly constrained quadratic programming (LCQP) problem which is proved as NP-hard, thereby making it not feasible to solve it by heuristic algorithm or dynamic programming because of its high computational and spatial complexity and large scale. Thus, we carry out a Deep Reinforcement Learning (DRL) for microservice deployment to address the complexities of dependency-aware microservice deployment in edge computing [34].

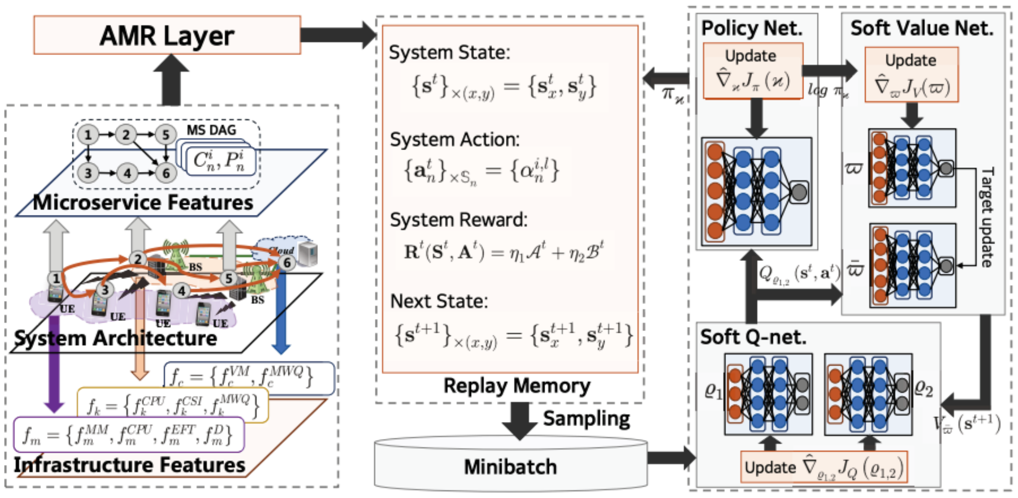

AMR Layer
System State:
$\{\mathbf{s}^t\}_{\times(x,y)} = \{\mathbf{s}_x^t, \mathbf{s}_y^t\}$
System Action:
$\{\mathbf{a}_n^t\}_{\times S_n} = \{\alpha_n^{i,l}\}$
System Reward:
$\mathbf{R}^t(\mathbf{S}^t, \mathbf{A}^t) = \eta_1\mathcal{A}^t + \eta_2\mathcal{B}^t$
Next State:
$\{\mathbf{s}^{t+1}\}_{\times(x,y)} = \{\mathbf{s}_x^{t+1}, \mathbf{s}_y^{t+1}\}$
Replay Memory
Sampling
Minibatch
Policy Net.
Update $\hat{\nabla}_\varkappa J_\pi(\varkappa)$
Soft Value Net.
Update $\hat{\nabla}_\varpi J_V(\varpi)$
Target update
Soft Q-net.
Update $\hat{\nabla}_{\varrho_{1,2}} J_Q(\varrho_{1,2})$
MS DAG
C_n^i, P_n^i
Microservice Features
System Architecture
Infrastructure Features
$f_c = \{f_c^{VM}, f_c^{MWQ}\}$
$f_k = \{f_k^{CPU}, f_k^{CSI}, f_k^{MWQ}\}$
$f_m = \{f_m^{MM}, f_m^{CPU}, f_m^{EFT}, f_m^{D}\}$
UE
BS
Cloud

Figure 76: Implementation of attention modified soft actor-critic (ASAC) scheme [34].

The reinforcement learning framework is modelled as an agent interacting with the edge computing environment as shown in *Figure 76*. The agent observes the state of the environment (e.g., available resources, microservice dependencies) and takes actions (e.g., deploying microservices to specific edge nodes) to maximize its reward (e.g., overall system performance). The agent utilizes a deep neural network to learn and represent the complex relationships between microservices, resource constraints, and system performance. We also introduce an attention mechanism within the deep neural network. This mechanism helps the agent focus on the most relevant aspects of the environment state, particularly the critical interdependencies between microservices, when making deployment decisions. The DRL agent learns to consider the following factors when making deployment decisions: 1) resource availability: the agent assesses the available computational resources at each edge node, 2) microservice dependencies: the agent considers the interdependencies between microservices to ensure they are deployed on nodes that enable efficient communication and data exchange. 3) performance impact: the agent evaluates the potential impact of each deployment decision on overall system performance metrics like latency and throughput.

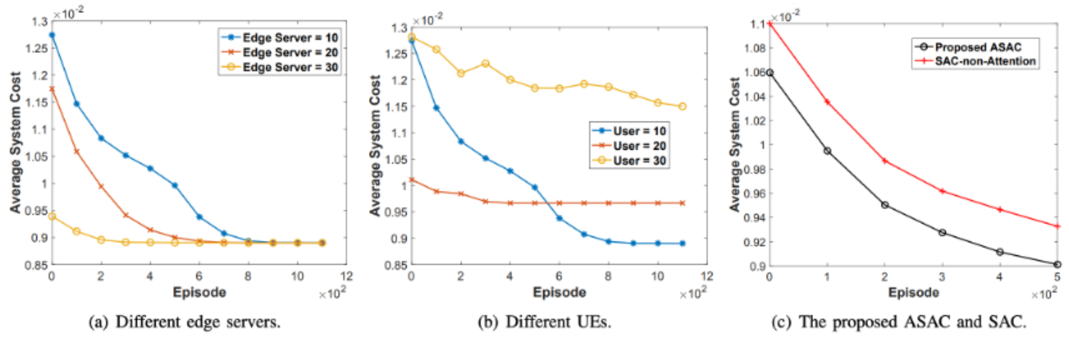


Figure 77: Performance of average system cost under different number of edge servers, UEs and comparison between the proposed ASAC and SAC [34].

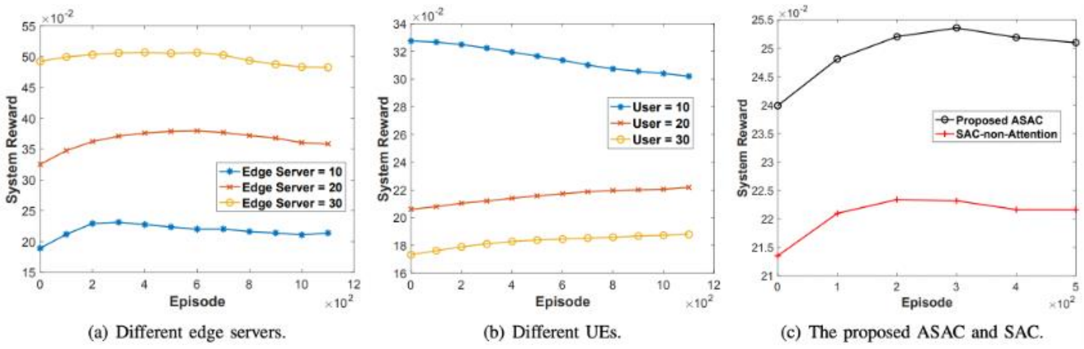


Figure 78: Performance of system reward under different number of edge servers, UEs and comparison between the proposed ASAC and SAC [34].

We demonstrate the simulation results in terms of the performance of average system cost and the system reward. All the results are obtained by the average values of 20 times. (1) Performance of average system cost: We deploy 20 UEs and 30 microservices in the system for the demonstration. From *Figure 77(a)*, it is observed that ASAC reaches the maximum average system cost when the number of Edge servers is 10, while the optimal cost occurs when the number is 30. With the increase in the number of Edge servers, the system cost becomes smaller until it converges. This is mainly because fewer edge servers mean less disposable network resources, so as the number of edge servers increases, system cost becomes smaller. *Figure 77(b)* shows the average system cost performance with different UEs, 10 edge servers and 30 microservices are deployed in this case. When the number of UE is 10, ASAC has the worst performance in the beginning but achieves the lowest cost in the end, while UE is 30, the performance is the worst. This is because the AMR algorithm occupies many network resources at the beginning of the system representation, and finally, the resources are optimally allocated with the system operation. We deploy 10 edge serves, 20 UEs, and 30 microservices in the following case. *Figure 77(c)* shows that the proposed ASAC reduces 5% average system cost compared to the original SAC. The main reason is that ASAC extracts the critical features from the system, accelerating the deployment speed at a lower cost. (2) Performance of system reward: For the same settings as the above demonstration, respectively, we carry out the performance of system reward, shown as *Figure 78*. It can be observed that when the edge server is 30, ASAC achieves the best performance in *Figure 78(a)*, the performance improvement is positively correlated with the number of edge servers while there is a negative correlation with the increase of users in *Figure 78(b)*. The reason is that when UE is 10, it occupies many network resources, which leads to the high system cost in the beginning, while when UE is 30, the average system cost is over-consumed, resulting in the low QoS for each UE. *Figure 78(c)* shows that the priorities of the proposed ASAC outperform the SAC algorithm for almost 30%. The work concludes by highlighting the potential of DRL for enabling efficient and dependency-aware microservice deployment in edge computing. It suggests potential future research directions, such as exploring the integration of additional factors like real-time resource monitoring and dynamic workload variations into the learning process.

6.7 GPU-based Primitives Supporting AI-based Service Placement

A proper placement of the micro-services of XR applications is highly important in the context of CHARITY. Multiple edge resources, potentially heterogeneous and available at different locations, must collaborate to deliver the right QoE to the end-users of the XR applications. Optimization and AI-based approaches to find the proper location for edge applications have emerged as a relevant field of study. Effectively, when using these approaches, there is the need for a consistent amount of processing power. Optimization approaches (e.g., mixed-integer programming solvers) require many operations to find the optimal solution, especially when the dimensions of the problem are high. On the other hand, machine learning approaches require a training phase whose processing needs depend on the complexity of the model and the size of the inputs. Distributed algorithms are well suitable to solve this problem efficiently. Such class of algorithms, falling under the category of MRMOGAP, have been briefly explained and analysed in Section 2. The process for creating an Aggregation could be iterated if the MRMOGAP could not be solved, or the distributed algorithm raises exceptions during execution or for example a set of resources are lost due to network connection failures or the occurrence of disruptive events. In this case, it is possible to expand the actual Aggregation with additional neighbours, or create a new Aggregation which is a superset of the old one, creating a hierarchical structure with multiple levels of orchestration. The Aggregation could either persist or be destroyed after the resolution of the application request.

In this context, solutions that help speed up the running time of the approaches mentioned above are of high interest. In particular, solutions based on GPUs are promising, and that is according to recent findings in the literature. More discussion and findings related to this can be found in deliverable **D3.2**, as part of the research activities conducted in WP3 – T3.1 Efficient exploitation of CPUs, GPUs and FPGA.

6.8 Application Resource Management

Applications designed according to the principles of cloud native architecture and distributed across computing nodes ranging from devices to the edge and the cloud need the support of container orchestration to ensure they are adequately resourced to fulfil their objectives. However, the resource usage demands of an application can ebb and flow according to usage patterns that often require intimate knowledge of the application to usefully analyse. Through detailed monitoring, we can externally observe resource fluctuations but fall short of diagnosing the appropriate course of action to deal with them. This difficulty is exacerbated when we seek to accommodate applications that dynamically adapt their use of resources according to business or operational imperatives. Such circumstances arise, for example, when applications encounter scaling constraints in dealing with rises in simultaneous users caused by resource shortages or budgetary constraints. In circumstances whereby applications self-adapt to reduce resource usage (or increase it to exploit unused resource availability), Kubernetes needs to 'be in the loop' so that resources can be efficiently managed. Self-adaptation may not be as straightforward as detecting anomalies through monitoring and instructing an application to reduce its fidelity or to toggle resource-intensive features while remaining in situ. It may involve replacement of service instances or groups of them with differently configured variations or a more radical change to the deployment topology with services being relocated across the edge-cloud. In the remainder of this subsection, we present a model of partnership between application self-adaptation and container orchestration to enable efficient application-aware orchestration.

6.8.1 Resource Usage Budgets

Recent years have witnessed increasing disaggregation of the cloud computing pricing model. We have moved on from the days of paying per VM (offered in various tiers or so called "t-shirt sizes") which often resulted in organizations dimensioning for their pressure point (e.g., memory) and needlessly paying for increases in the other dimensions of storage and computing muscle. Cloud providers now offer a far more granular model in which multiple dimensions can be customized to a user's projected needs. In addition, we have moved on from organizations needing to dimension for their peak demand while needlessly paying for idling resources outside of this time window. Hyperscalers are now offering "burst" capacity to cater for workloads which experience varying demands over time.

Beyond the simplistic marketing headlines, however, lies a deep morass of pricing complexity. Cloud spend is still wasted but in places that are not so easily monitored and detected. It involves many contributing factors such as level of support, choice of services, choice of features, tiered usage pricing, service level agreements, backup, and redundancy preferences, monitoring, and alerting. Organizations are offered increasingly rich catalogues of services ranging from machine learning to data analytics to quantum computing and all have their own pricing vagaries. Across just five hyperscalers, the Cloud Price Index tracks over two million pricing variability points [35] [1] and it remains notoriously difficult to accurately estimate costs pre-deployment. Edge computing infrastructure, by its nature, is highly distributed and the resources available in each locality cannot match the relatively endless scalability of large cloud computing data centres. For application providers seeking to deploy across a cloud-edge continuum, it is reasonable to expect that pricing complexity will not reduce and that they will seek mechanisms to instil overall resource usage constraints.

To stabilize costs, many organizations adopt a hybrid cloud approach in which some of the computing resources are owned by the organization and typically reside on their premises, while the public cloud resources are used to provide surge capacity and externally facing resources such as public web sites. We will likely see enterprises deploy their own edge resources for campus-style scenarios to manage costs and availability. The key concern is that edge resources, whether operated by the enterprise or provided by a third party, cannot scale in a cloud fashion. They will consist of high-end, relatively expensive physical resources (such as advanced GPUs), and they will need to be used judiciously. Otherwise, application providers may quickly exhaust their budget or the available resources. CHARITY vision is to integrate into the same platform resources provided by cloud providers with resources that can be provided by the XR service providers. As explained above, this would allow more flexibility for

XR providers by easily leveraging cloud resources when needed while being able to extend the resources at the edge. All of which will be managed and integrated into the same CHARITY platform.

6.8.2 Usage Segmentation

Over-dimensioning of resources is common amongst application providers as they are often faced with uncertainty about future usage patterns and conditions. Monitoring actual application usage and comparing to reserved resources can reveal opportunities for resource repurposing and drive savings. However, there are circumstances in which application usage can vary from one instance to another.

An application provider faced with reaching scaling limits due to inherent problems with the application design[27] or available resource saturation (whether it is physical or fiscal) has a narrow range of options open to them. Either they throttle overall usage and discourage new clients from connecting or they reduce the fidelity of the application so that it has a lighter resource footprint and thus able to accommodate additional users. There is also of course, the option to mix the two strategies. If there are different classes of users of an application, then a provider may elect to prioritize resources for their most important users and reduce the fidelity of experience for the remaining users. For instance, in CHARITY UC 3.2 (see D1.2), Manned-Unmanned Operations Trainer, there are two classes of users – Priority and Best Effort. Priority users demand a full fidelity experience and may be undergoing certification sessions. Best Effort users, on the other hand, would typically be experimenting with the simulator. We would like to deliver the best experience we can to best effort users when the capacity is available but step it down when resource availability or resource budget constraints demand. Such adaptation requires both the application instances and the Orchestrator to be on the same page.

Perhaps in the most straightforward approach, an application stepping down its fidelity and thus resource requirements, would announce this change to the Orchestrator which would correlate the new fidelity level with appropriate resource requirements using a lookup table and reify this in the resource allocation. The work in [36] explores how Real-Time Interactive Applications (ROIA) can be augmented with a module that enables them to communicate directly to an SDN controller when the usage dynamic of an application instance requires a change in QoS. The reasoning is that the resource demands of an application instance can vary according to what activity the user is performing. If in the exploratory phase of a game, for example, then higher network latencies can be tolerated while lower latencies are essential when the player enters a first-person shooter phase. The authors propose a custom C++ library linked into the application under test which communicates real-time application usage metrics to an SDN Controller through the Northbound interface enabling the controller to leverage custom rules and make resourcing decisions according to the application needs.

In [37], the authors introduce a Machine Learning component alongside an SDN controller to deduce correlations between an “arbitrary performance metric from an application” and observed network characteristics. The metrics, which capture how well the application is performing in attainment of its goals (e.g., resiliency, responsiveness, availability), is continually reported from the application to the SDN Northbound interface and fed to the ML component along with current network application usage with the goal of building a model that can inform the controller of potential resource reallocation opportunities that could reduce resource consumption without compromising application goals. The authors assume that applications can be instrumented in such a fashion that potentially complex and holistic metrics can be efficiently gathered.

There is a difficulty with approaches that depend on application instances announcing their need for an increase or reduction in resources because they are dynamically toggling features or altering their fidelity levels. They assume that applications are sophisticated enough to adapt their behaviour dynamically and consume less resources while still delivering an appropriate quality of experience to their users. In practice, applications are rarely written from scratch and leverage legacy code and third-

[27] Reliance on a shared resource which suffers from scalability constraints for example which acts as a bottleneck.

party components so that developers can reserve their efforts for their business differentiating functionality. It is not uncommon to require an application service to be restarted if a significant configuration change is required such as disabling a key feature or operating at lower fidelity because the ability to dynamically change is not built into the software and the source code is not available for modification. In modern service-based architectures, altering the behaviour of one service can have consequences for its peers and multiple service restarts may be required. It is also possible that a change to the execution model of an application distributed across the device-edge-cloud continuum will require the migration of services horizontally across the continuum. These various scenarios are depicted in *Figure 79* which represents a high-level deployment view and possible adaptation scenarios for a distributed service-based application[28].

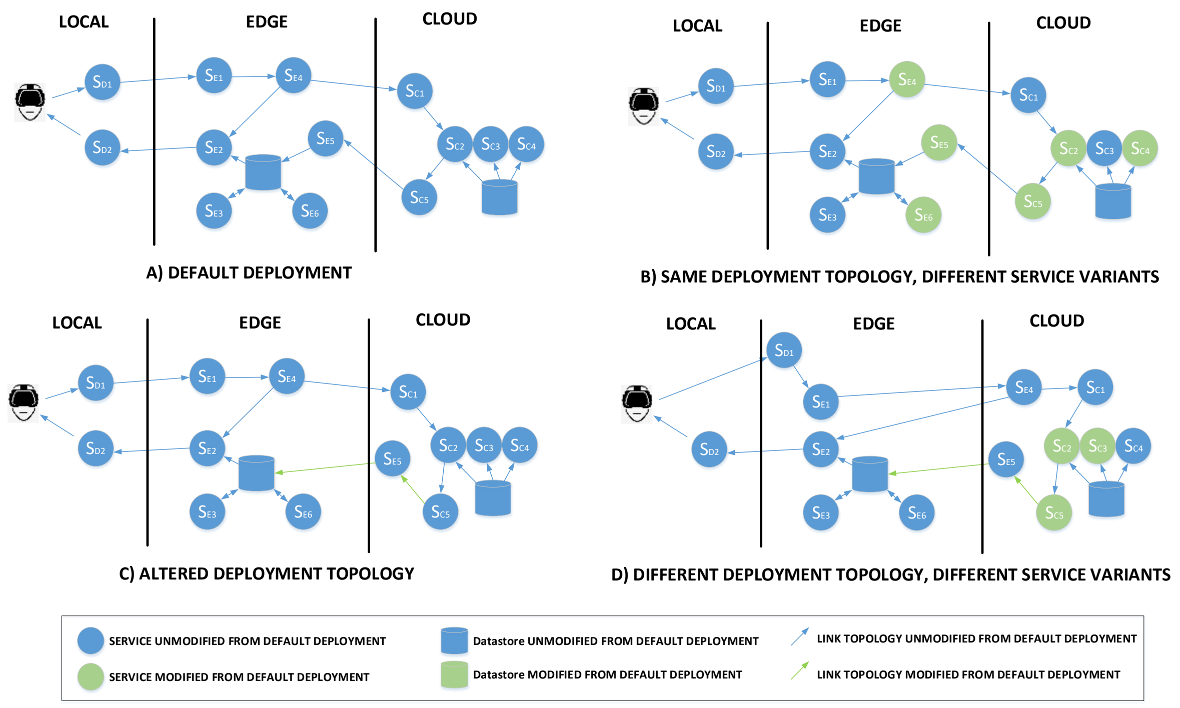

LOCAL
EDGE
CLOUD
A) DEFAULT DEPLOYMENT
B) SAME DEPLOYMENT TOPOLOGY, DIFFERENT SERVICE VARIANTS
C) ALTERED DEPLOYMENT TOPOLOGY
D) DIFFERENT DEPLOYMENT TOPOLOGY, DIFFERENT SERVICE VARIANTS
SERVICE UNMODIFIED FROM DEFAULT DEPLOYMENT
SERVICE MODIFIED FROM DEFAULT DEPLOYMENT
Datastore UNMODIFIED FROM DEFAULT DEPLOYMENT
Datastore MODIFIED FROM DEFAULT DEPLOYMENT
LINK TOPOLOGY UNMODIFIED FROM DEFAULT DEPLOYMENT
LINK TOPOLOGY MODIFIED FROM DEFAULT DEPLOYMENT

Figure 79: Application adaptation scenarios.

In *Figure 79*, we depict a number of different adaptation scenarios that could play out according to environment circumstances. In each scenario, we depict changes from the default deployment in green

a) The default deployment. This is our starting state and represents the topology and software component configuration when the application is initially deployed.

b) Different Service Variants. In this scenario, we maintain the same deployment topology (services currently deployed on the edge stay located on the edge, those running on the cloud remain running on the cloud). However, we change the configuration for some of the services so that the overall application resource footprint is modified.

c) Altered Deployment Topology. In this scenario, we do not modify or replace any of the default service configurations but instead migrate one or more services from the edge to the cloud or vice versa.

d) Altered Topology and Service Variants. In this scenario, quite a lot of changes occur at once. Not only do we replace some services with differently configured copies, we also consider moving the location of a service from the edge to the cloud.

[28] Approximately modeled on a real flight simulator application.

6.8.3 Application Aware Orchestration

For an application to be adapted from its existing state, we need to describe the new state and have the orchestrator move us to it. In a service-based distributed application, it is unlikely that a wholesale restart of the entire collection of services would be required, and different applications will offer different opportunities for service rewiring. Some infrastructural services (such as databases, caches, and webservers) may not be operable at lower fidelity; some may be statefully recording and utilizing internal state for servicing successive requests and not be amenable to restarts; others may be perfectly amenable to restarts and migrations.

To cover the adaptation use cases summarized in *Figure 79*, we need to support the following functionality from the perspective of the orchestrator:

- Identify the need for adaptation
- Identify the changes needed to result in adaptation
- Allocate any new resources required to satisfy the changes
- Relay new resource details (e.g., IP addresses) to the application
- Deallocate any resources no longer required

With Kubernetes, containers are deployed in pods and their resource requirements (such as the need for GPU, memory, storage) are relayed to Kubernetes through YAML configuration files. These are then referenced by Kubernetes in its decision making about where to deploy the pods and how much resources to reserve. However, the resource needs of an application can vary depending on its configuration and workload at a given point in time. The configuration can, in turn, depend on the resource performance and availability at a given point in time. For example, there may be bandwidth or link latency issues. There may be budgetary limits that require the temporary reining in of resource usage. Addressing such issues may necessitate an application to adapt to different resource budgets while keeping Kubernetes in the loop to dynamically alter the profile of resources reserved.

As discussed earlier, depending on application APIs to expose dynamic adaptation functionality unnecessarily narrows the solution to those applications that have and expose such functionality. However, there are other problems with this approach. Firstly, it works in opposition to the 'cattle versus pets' ideal in cloud native design which essentially states that we should be able to terminate any individual service and immediately replace it with another identical instance while maintaining service continuity. If there has been manipulation of the service configuration using runtime APIs then identical manipulation would need to occur on any restarted service instances. Secondly, Kubernetes has no awareness or interest in such manipulation. If the services in a pod are dynamically reconfigured to use less resources, Kubernetes still holds the originally requested resource allocations which significantly negates the benefits of instructing the services running in a pod to use less resources. A more efficient and far more robust approach is to update the static configuration settings for the services in a pod by requesting Kubernetes to reference new Deployment and ConfigMap files for the pod. This can initiate a rolling update of the pod by Kubernetes causing the pod to be safely replaced with a reconfigured pod with a new resource allocation budget.

We introduce an Application Orchestrator that receives alerts from the Prometheus monitoring platform and directs Kubernetes to update one or more pods to transition those pods from one resource usage profile to another. This is depicted below in *Figure 80*.

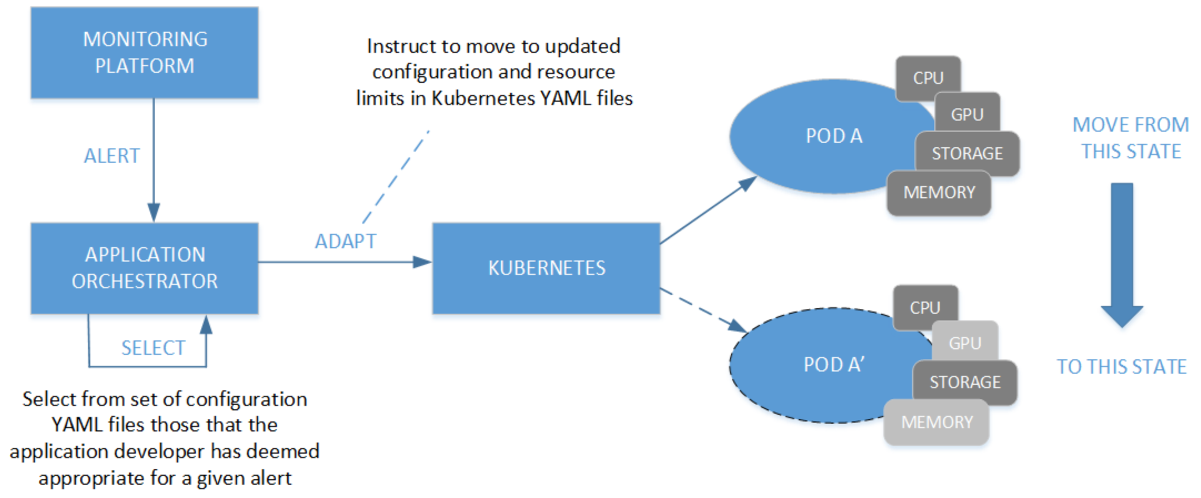

MONITORING PLATFORM
ALERT
APPLICATION ORCHESTRATOR
SELECT
Select from set of configuration YAML files those that the application developer has deemed appropriate for a given alert
Instruct to move to updated configuration and resource limits in Kubernetes YAML files
ADAPT
KUBERNETES
POD A
CPU
GPU
STORAGE
MEMORY
POD A'
CPU
GPU
STORAGE
MEMORY
MOVE FROM THIS STATE
TO THIS STATE

Figure 80: Application Orchestrator directing movement of pod from one state to another.

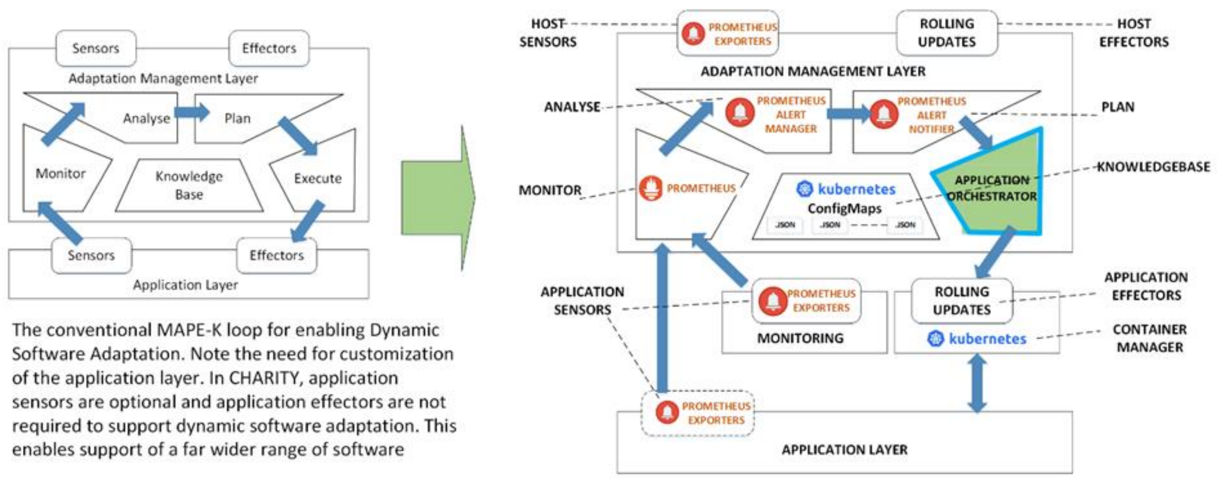
One of the challenges we outlined earlier with service-based architectures is the dependence between some services such that a reconfiguration of one may necessitate a simultaneous reconfiguration of another to maintain the platform in a coherent state. Kubernetes pods enable us to group tightly dependent containers together to facilitate such scenarios; Kubernetes rolling updates[29] provide the means to stand up such container groupings in a safe manner; Kubernetes Services allow us to manage interdependencies between pods such that pod replacements happen transparently to clients of that pod[30].

The Application Orchestrator is provided by the Application Provider. Its responsibilities are to decide when instances of an application need to be adapted, how they are to be adapted, and conduct the necessary dialogue with Kubernetes to affect the adaptation. How the orchestrator makes its decisions and maps circumstances to a particular choice of Kubernetes configuration files is completely up to the Application Provider. It can receive alerts and query metrics from Prometheus and track application specific data such as the number of active users, what they are doing, what their privileges are; track the overall resource usage (and possibly financial spend) according to the resources allocated to active users; respond to quality of experience and resource fluctuations observed by the Prometheus Alert Manager.

In *Figure 81* below we contrast the traditional approach employed in Dynamic Software Product Lines [38] with that proposed in CHARITY.

[29] Kubernetes rolling updates allow deployment updates to take place with zero downtime by safely replacing pod instances with new ones. They can be equipped with readiness checks to ensure the replacement pod is in a healthy condition before switching over

[30] Kubernetes Services essentially sit between pods and clients of those pods. Clients can communicate with pods through Services. It nicely conceals the pod topology and ip address concerns from clients. A load balancer service is a common use case in which a single service sits in front of a pool of pods and distributes traffic amongst them. Services play a key role in rolling updates by concealing the replacement of one instance of a pod with another from users of the pod.



The conventional MAPE-K loop for enabling Dynamic Software Adaptation. Note the need for customization of the application layer. In CHARITY, application sensors are optional and application effectors are not required to support dynamic software adaptation. This enables support of a far wider range of software

Figure 81: Application Aware Orchestration.

Dynamic software adaptation, although a component of the CHARITY platform, has not been integrated with the wider CHARITY platform. It has however, paid close attention to and adopted many of the technologies employed in CHARITY such as Prometheus, Alert Manager and Kubernetes. This offers a visible path to integration in which application owners would configure alerts in the CHARITY platform and have them routed to their Application Orchestrator. From here, a blueprint would be selected describing the updated pod configuration and a rolling update would be requested from the AMF which in turn conducts a dialogue with the CHARITY Orchestrator to deploy the updated pod.

6.9 Algorithms for Service Migration

Service migration consists in moving a service across locations, regions, or even infrastructure providers. It is one of the pillars to ensure the continuity of the service while maintaining the Quality of Service. A service is composed of many VNFs chained together that offer the said service. The placement of each VNF is important. For instance, delay sensitive VNFs should be placed at the edge close to the end users, while delay-tolerant ones can be placed in distant cheap clouds. With service migration, these VNFs can be moved according to end-users' mobility, resources shortage, and many other reasons. While SDN is used to maintain the connectivity between the chained VNFs. It is worth noting that VNFs migration comes with a cost, service disruption may occur, network resources overhead due to moving VNFs, a reconfiguration needed to reroute the traffic, and the momentary increase in resources consumption due to the migration. Service migration patterns can be split into three principal categories: i) full slice mobility; ii) partial slice mobility, which includes slice breathing, slice splitting, and slice merging; and iii) slice mobility optimizer, which contains slice shrinking pattern.

In what follows, we will show how selecting triggers of service migration can improve the overall performance. More specifically, Deep Reinforcement agents are used to learn how to use these triggers. We will also show how service migration can be improved by optimizing the resources allocated to the migration.

6.9.1 Service Migration Triggers

In this section, the triggers of service migrations are discussed, it summarizes Addad et al.'s work [39]. These mainly relate to users' mobility, resources availability, utilization efficiency, cost and energy reduction, and service reliability and security. Nevertheless, these triggers are non-orthogonal and can overlap, the mobility action selection process becomes complex and unambiguous.

Figure 82 depicts an overview of the envisioned architecture, incorporating an agent capable of autonomously selecting triggers and actions for allowing various Network Slice Mobility (NSM) patterns. It is divided into two separate layers, the Orchestration layer and MEC layer. This layering model helps manage applications by casting MEC in NFV paradigms, hence complying with ESTI's MEC and NFV standards. Considering the MEC-NFV standards, both the Mobile Edge Platform (MEP) and

MEC applications (MEC app) are VNFs. Therefore, elements of the NFV domain hosted in the MEC layer, i.e., the Virtualized Infrastructure Manager (VIM), NFV Infrastructures (NFVI), and VNF Manager (VNFM), manage their life-cycle. The Mobile Edge Platform Management (MEPM - V) acts as Element Management (EM) in the NFV architecture, thus providing application management features to the MEP. The NFV Orchestrator (NFVO) and the Mobile Edge Application Orchestrator (MEAO), in the Orchestration layer, share service application information and the network service information in the MEC-NFV domain to provide a reliable orchestration system. It is worth noticing that we omitted the reference points details between MEC and NFV components for clarity.

The Slice Mobility Decision Maker (SMDM) agent is an additional plugin to MEAO [40]. The main components of SMDM are the Request Manager (RM), the Learning and Exploration (LE) module, the Trigger Selector and Exploitation (TSE) module, and the DRL Algorithms Comparator (DAC) module. The SMDM agent interacts with the MEC layer through the RM module. It retrieves states, selects decisions such as scaling up/down or migrating MEC apps, and receives rewards for its decisions. The SMDM agent communicates with the Operation/Business Support Systems (OSS/BSS) for executing administrative and billing instructions. It also leverages the NFVO to perform the migrations and scaling operations. In this architecture, the information between the SMDM agent and the MEC hosts transits via the standardized interfaces of MEAO and MEPM elements.

Three triggers have been used to constitute the state of a MEC host. These are Resource Availability Trigger (RAT), Service Consumption Trigger (SCT), and Request Overload Trigger (ROT). As it can be seen on *Figure 83*, the RAT trigger deals with the aggregate system-level resources related to the under-laying nodes. It provides details about the CPU, RAM and Disk capacities of the MEC and their current consumption. The SCT and ROT triggers cope with the performance of a single service. The main idea behind developing these triggers is to monitor the services themselves instead of watching only the MEC hosts, allowing bigger flexibility and exploring a more comprehensive range of new actions such as scale-up/down operations. The details of the triggers, i.e., SCT and ROT, are the CPU and RAM of each container-based MEC app and their current consumption. Moreover, we can expand these details to cover different parameters such as the number of requests/MEC app. Appending all these triggers together from each MEC will result in a feature vector that represents the state of the edge infrastructure that can be used by RL agents.

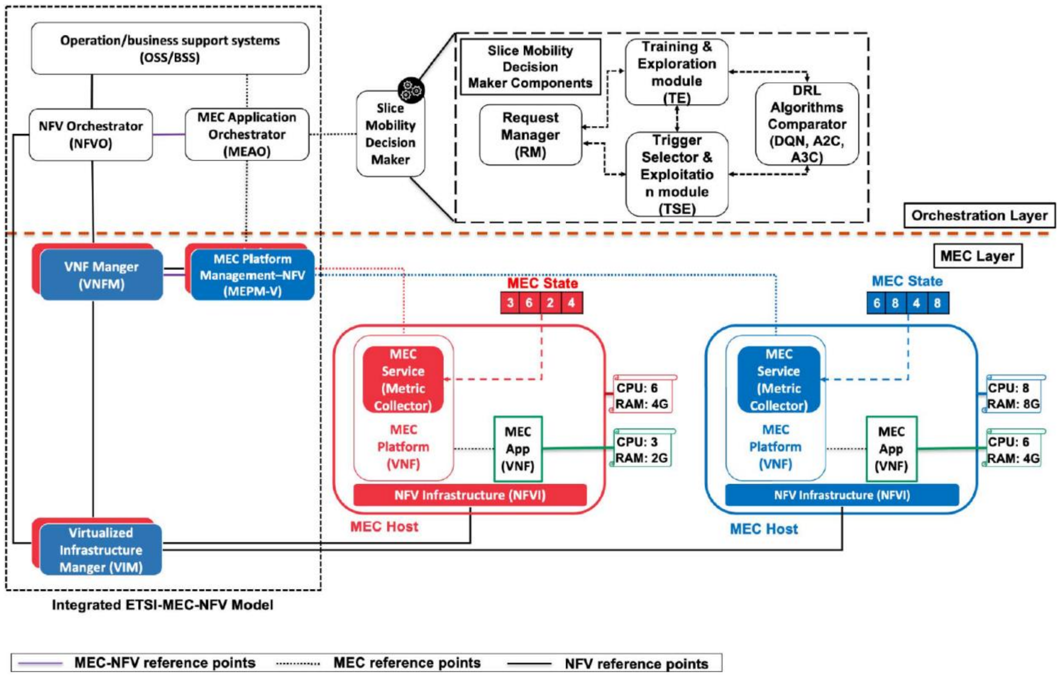

Operation/business support systems (OSS/BSS)
NFV Orchestrator (NFVO)
MEC Application Orchestrator (MEAO)
Slice Mobility Decision Maker
Slice Mobility Decision Maker Components
Training & Exploration module (TE)
DRL Algorithms Comparator (DQN, A2C, A3C)
Request Manager (RM)
Trigger Selector & Exploitation module (TSE)
Orchestration Layer
MEC Layer
VNF Manger (VNFM)
MEC Platform Management–NFV (MEPM-V)
MEC State
3 6 2 4
MEC State
6 8 4 8
MEC Service (Metric Collector)
MEC Platform (VNF)
MEC App (VNF)
CPU: 6 RAM: 4G
CPU: 3 RAM: 2G
CPU: 8 RAM: 8G
CPU: 6 RAM: 4G
NFV Infrastructure (NFVI)
MEC Host
MEC Host
Virtualized Infrastructure Manger (VIM)
Integrated ETSI-MEC-NFV Model
MEC-NFV reference points
MEC reference points
NFV reference points

Figure 82: Architecture of smart triggers selection for service migration [39].

The state-space defined above allows to obtain a state at each time-step. An action space is needed to be able to transit from one state to another. The action space is represented by:

- no-action, i.e., conserves the current resources distribution.
- migrate from a given source MEC host to a given target MEC host.
- scale up/down various resource types such as CPU and RAM.

Finally, for the reward, it is a combination of the time needed to complete a migration operation and state of consumption of the resources (e.g., CPU, RAM, …).

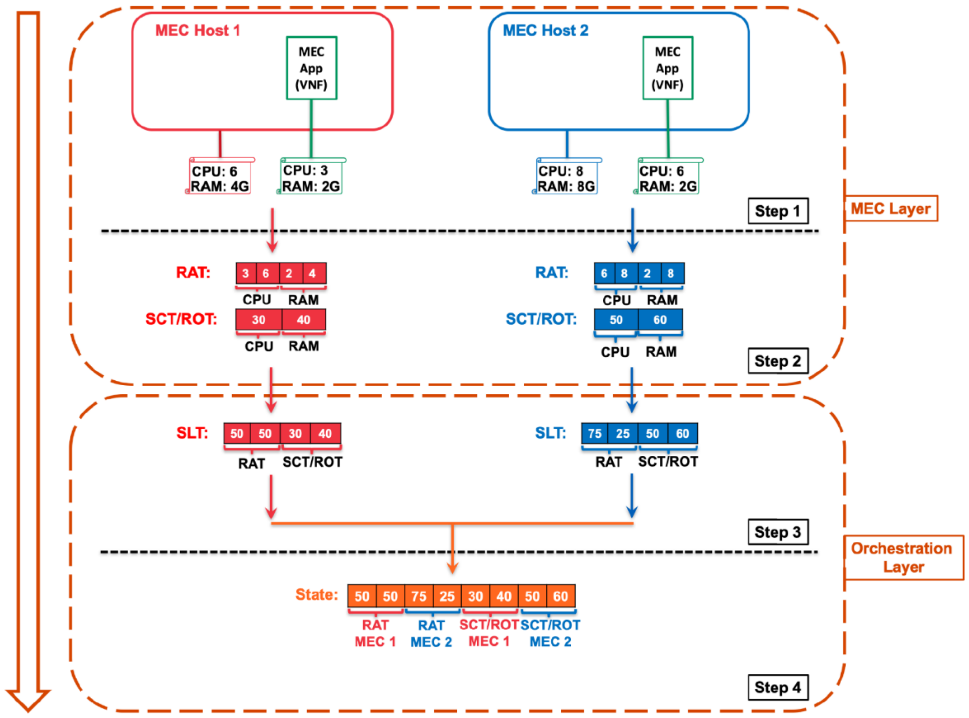


Figure 83: Triggers for RL agent [39].

To evaluate DRL agent using the triggers, experiments have been carried out. In this experiment, 10000 episodes are run, changing the resources randomly for MEC host and MEC apps in the underlying layer. *Figure 84* shows the result of two agents, the first is based on DQN and the second on A2C. In *Figure 84*, the Y-axis represents the rewards, while the X-axis portrays the number of episodes in the training process. A 100-episodes average is plotted on the same figure (i.e., orange colour). The results showed the efficiency of the A2C-based agent compared to the DQN-based agent in terms of average/cumulative rewards and learning stability.

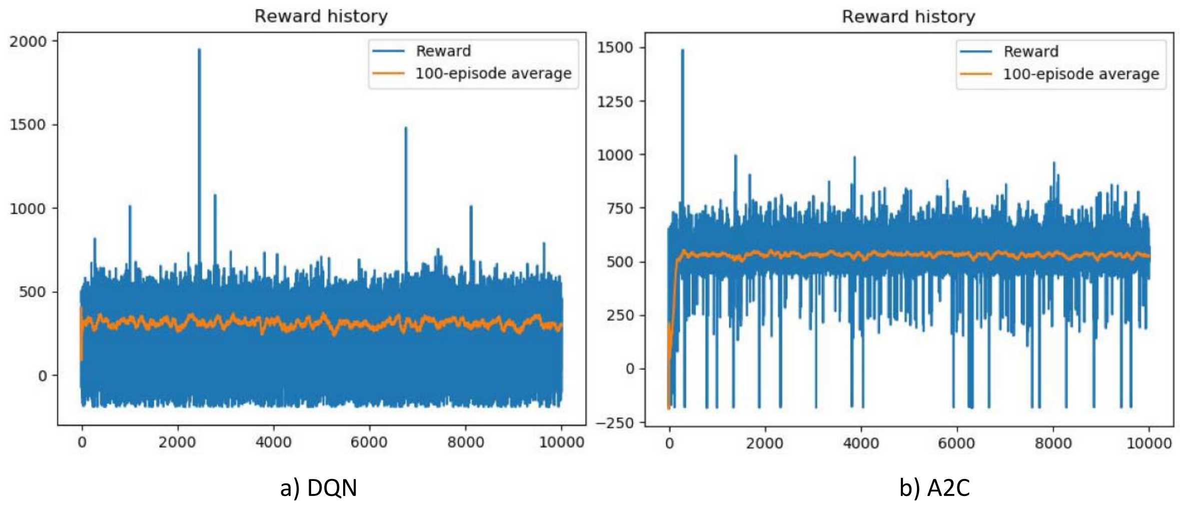


Figure 84: Reward history of SMDM agents [39].

Note that, this work constituted an effort toward transforming the service migration triggers into intelligent ones that can be used by RL agents. Two agents have been proposed, and many other RL agents can be also leveraged and tailored specifically towards XR services for the service migration, such as DDPG, TRPO, or PPO.

6.9.2 Service Migration Strategy in the Core Cloud

Based on SDN/NFV technologies, Service Function Chaining (SFC), standardized by the Internet Engineering Task Force (IETF), is regarded as an important networking concept to provide users with flexible services provisioning, e.g., XR application functions. Typically, SFC comprises a sequence of VNFs, and the traffic needs to be steered to traverse these VNFs in a predefined order. As it is essential to deploy SFC onto the physical network, at present, there has been some research on the SFC deployment problem to focus on how to reduce network providers' operational costs, increase resource utilization rate, and guarantee users' Quality of Service (QoS). When a SFC request is received, the management and orchestration (MANO) layer of the network system allocates resources and takes responsibility for recycling resources after the lifecycle of a request. Furthermore, to avoid QoE and Service Level Agreement (SLA) violations caused by traffic load variations, MANO should dynamically adjust the allocated resources of SFC requests. In addition, to save the allocated resources, and adapt to the users' mobility, MANO sometimes even must migrate some VNFs of a request. However, the above adjustment process may result in the unbalanced distribution of physical resources. Different from previous research work, this work mainly answers the following two questions. A: Will uneven distribution of physical resources adversely affect subsequent SFC requests and network operators? B: Can we reduce the potential negative impact of the uneven distribution of physical resources by migrating SFC requests in the initial service queue?

To model the SFC migration problem into an integer linear program (ILP) problem, the following key components, involved in the migration process, are considered: 1) Physical machines – represented as variables indicating their resource availability; 2) SFC instances – represented as variables indicating their resource requirements and migration costs; and 3) Migration decisions – represented as binary variables indicating whether to migrate an SFC instance from one machine to another. The ILP is designed to 1) minimize the total migration cost, 2) maximize the resource utilization across all physical machines, and 3) ensure sufficient resources are available on the destination machine after migration. Due to the computational complexity of solving the ILP for large-scale scenarios, we propose a heuristic-based migration algorithm [41], as shown in *Figure 85*. This algorithm efficiently approximates the optimal solution while maintaining acceptable performance. The algorithm iteratively performs the following steps: 1) identify overloaded machines and underutilized machines. 2) evaluate potential migrations based on benefit-to-cost ratio, considering resource improvement and migration overhead. 3) select the migration with the highest benefit-to-cost ratio and update resource availabilities. Repeat steps 1-3 until a stopping criterion is met (e.g., resource utilization threshold reached).

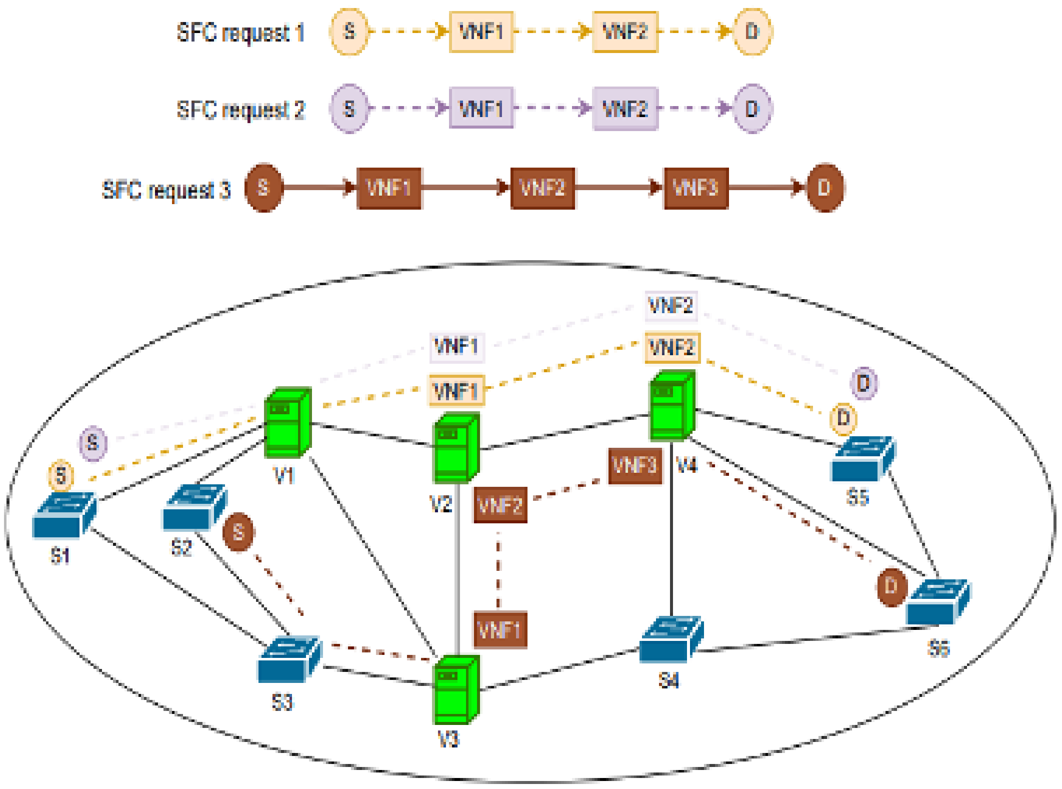


Figure 85: Process of aggressive migration scheme [41].

The proposed strategy was demonstrated through extensive simulations. We then evaluate the proposed strategy through simulations against two benchmark solutions using real-world network traffic traces and various performance metrics: 1) resource utilization – the aggressive strategy demonstrates significantly higher and more balanced resource utilization compared to conservative strategies. 2) service acceptance ratio – the aggressive strategy leads to a notable increase in the number of successfully accepted SFC requests due to improved resource availability. 3) migration overhead – while the strategy incurs some overhead due to migrations, the benefits in terms of resource utilization and service acceptance outweigh the costs. 4) impact on service latency – the work acknowledges the potential for slight increases in service latency during migration. However, the overall improvement in resource utilization and service acceptance often outweighs these potential latency penalties. The performance of the proposed strategy is compared with two benchmark solutions: 1) BestFit – First, the physical node with the most remaining computational resources is greedily selected to place the VNFs requested by a request, and then the shortest path between these nodes is calculated to connect these VNFs in series under the premise of meeting the bandwidth. 2) CN – CN similarly divides resource allocating into two stages. First, it calculates the importance of each node according to the degree, betweenness centrality of the nodes, the remaining computational resources of the nodes, and the remaining bandwidth resources of the links to which the nodes are directly connected. Then, the VNFs requested by SFC requests are greedily placed on physical nodes with high importance. Finally, under the premise of meeting the bandwidth requirements, the shortest path between these physical nodes is calculated to connect the VNFs in series.

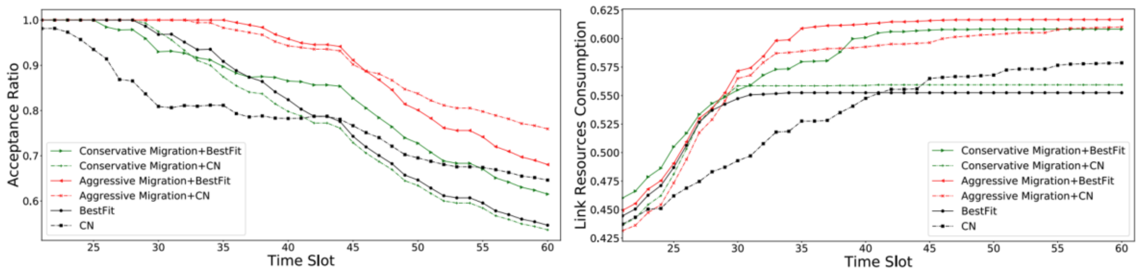


Figure 86: (a) Different acceptance ratios and (b) different link resource utilization for an experimental instance [41].

We first analyze the simulation results for a single experimental instance. As shown in *Figure 86*(a), as time goes by, MANO receives more and more SFC requests, and the physical resources become more and more strained. Therefore, after a period, the acceptance ratio of SFC requests starts to decrease. By migrating SFC requests in the service queue at the initial moment through our proposed strategy, we can make the operators receive more SFC requests, and thus increase the final acceptance ratio. However, the conservative migration strategy has a negative impact on the final acceptance ratio. We then analyze changes in physical resource utilization. As shown in *Figure 86*(b), *Figure 87*(a), and *Figure 87*(b), for both “BestFit” and “CN” heuristics, our proposed migration strategy improves final resource utilization and long-term profit. The “BestFit” heuristic, although the conservative migration strategy reduces the acceptance ratio of SFC requests, improves the final resource utilization. For the “CN” heuristic, the conservative migration strategy reduces both the acceptance ratio of SFC requests and the final resource utilization. Interestingly, however, the conservative migration strategy advances the time when the physical resource utilization reaches a plateau. From time slot 30 to time slot 44, the resource utilization of the “Conservative Migration + CN” strategy is higher than the pure “CN” strategy. Although the resource utilization of the pure “CN” strategy exceeds that of the “Conservative Migration + CN” strategy from time slot 45, it still needs a long period of time for the pure “CN” strategy to catch up with the long-term profit of the “Conservative Migration + CN” strategy. From time slot 45 to time slot 60, the gap between the pure “CN” strategy and the “Conservative Migration + CN” strategy in long-term profit gradually narrows, but until time slot 60, the long-term profit of the “Conservative Migration” strategy is still higher than that of the pure “CN” strategy.

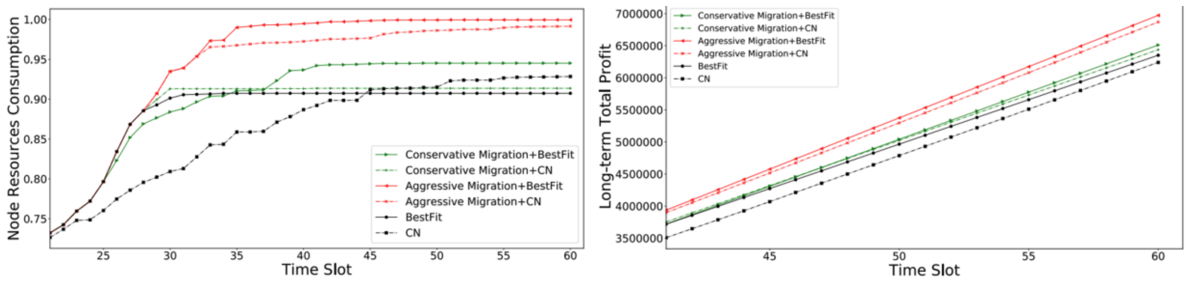


Figure 87: (a) Different node resources utilization and (b) different long-term profit for an experimental instance [41].

6.9.3 Network Aware Service Migration

Once the decision of a service migration is made and a target cloud/edge has been chosen, the next step consists in orchestrating the migration. This consists in moving the VNFs to the new edge/cloud and also redirecting the traffic to the new VNFs. In order to move these VNFs, network and computing resources should be available. It also takes some time to do this moving, which can be even a downtime (i.e., users requests are not served) if it is not a live migration. Therefore, it is important to find the right balance between how much resources to allocate to this migration versus how long the migration would take. In order to do this, RL based agents have been investigated to find this right balance. In what follows, we summarise Addad et al.’s work [39] that was done in this vein. A system is proposed to host the RL agents. The proposed system, depicted in *Figure 88*, complies with ETSI-NFV standards. In the defined system, the MEC layer is controlled through the interaction between the components of the Orchestration layer and the elements constituting the NFV architecture. Several components of the Orchestration layer have been omitted to focus on the Smart Network-Aware (SN-A) agent that is supposed to fine-tune the bandwidth allocation process.

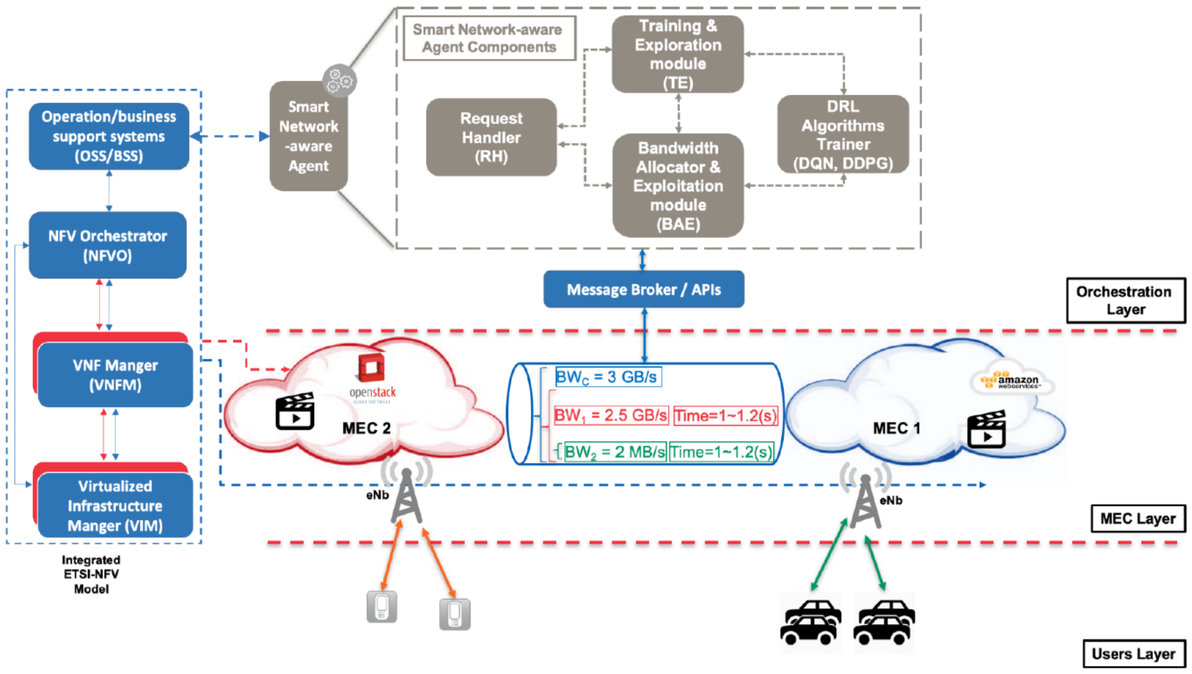

Smart Network-aware Agent Components
Training & Exploration module (TE)
Request Handler (RH)
Bandwidth Allocator & Exploitation module (BAE)
DRL Algorithms Trainer (DQN, DDPG)
Smart Network -aware Agent
Operation/business support systems (OSS/BSS)
NFV Orchestrator (NFVO)
VNF Manger (VNFM)
Virtualized Infrastructure Manger (VIM)
Integrated ETSI-NFV Model
Message Broker / APIs
Orchestration Layer
openstack
MEC 2
MEC 1
amazon
BWc = 3 GB/s
BW1 = 2.5 GB/s Time=1~1.2(s)
BW2 = 2 MB/s Time=1~1.2(s)
eNb
eNb
MEC Layer
Users Layer

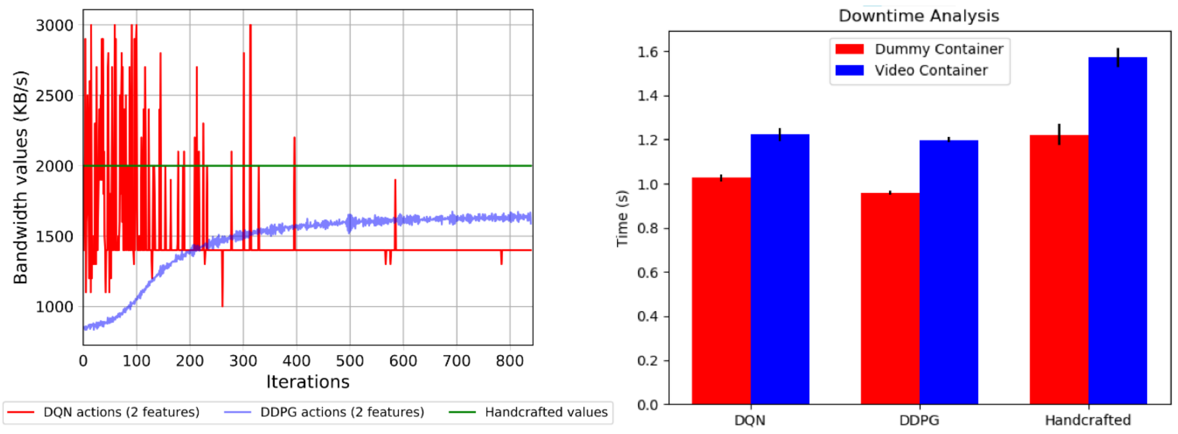
Figure 88: Architecture of a network aware service migration orchestrator [42].

The Request Handler (RH) module offers to the SN-A agent a technology-agnostic abstraction to access MEC-layer entities, i.e., public or private cloud platforms. Therefore, regardless of the underlying MEC platform, the SN-A agent retrieves states, accordingly outputs decisions of bandwidth values, and receives rewards for each decision. The SN-A agent also receives administrative instructions from the Operation/Business Support Systems (OSS/BSS) as defined in the ETSI-NFV model. The RH module must ensure reliable communication and synchronization between the SN-A agent and the MEC layer. It can achieve this through a message broker functionality, e.g., RabbitMQ, or a standardized Application Programming Interface (API). In the NFV model, the MEC layer components are hosted on distributed NFV Infrastructure (NFVI) and would be controlled by one or more Virtualized Infrastructure Managers (VIMs). The Orchestration layer is hosted separately and communicates with the NFV domain through the NFV Orchestrator (NFVO) to emit corrective decisions and actions. VNF Managers (VNFMs) manage life-cycles of the SFC services carried out on VNFs over multiple administrative domains. Furthermore, users in the users' layer benefit from the distributed aspect of computations in the MEC layer, which reduces latency while following end-users' mobility patterns.

The Training and Exploration (TE) module is responsible for creating identical digital twin environments used for the training phase of the SN-A agent. Initially, the TE module, through the RH module, gathers all the bandwidth capacity and latency information between each pair of MEC nodes to obtain a global knowledge of the distributed infrastructure. A client/server-based IPerf test integrated with the TE module in this scouting stage is used. This step is a reconnaissance phase that generates most of the network information that is used as an upper-bound for selecting bandwidth actions. Then, after each migration decision in the test environment, the TE module reserves the network resources to successfully complete the SFC migration operations while improving the global bandwidth utilization. Finally, the used resources are released whenever migrations are completed. Note that a practical implementation of the SFC migration schemes is used, which basically means that in addition to ensuring service migration, there was need to also guarantee predetermined order of SFC components and their respective network and system dependencies. The presented process allows the SN-A agent to learn how to attribute optimal/near-optimal bandwidth values over time through the TE module. It should be also noted that it is possible to replicate these offline trial and error achievements in other environments, e.g., 5G networks, as the training and testing phases share the same input features and output decisions.

Once obtaining preliminary results, the TE module shares its learned model with the Bandwidth Allocator and Exploitation (BAE) module to minimize network resource utilization. Therefore, the results' usability can be validated by comparing them to their handcrafted counterpart, defined in [39]. The SN-A agent compares the learned policies against the handcrafted values; if both downtime and total migration time of the SFC migration increase, the TE module will continue the learning process without reporting its current findings to the BAE module. Reversely, if the TE finished learning a fully working model, the SN-A agent will use BAE to forward the accurate decisions to the MEC layer. Finally, both TE and BAE use the "DRL Algorithms Trainer (DAT)" module, which trains DRL algorithms based on the received inputs and delivers adequate bandwidth values.

The preliminary results demonstrate that both DQN and DDPG achieved better results compared to the baseline solution. From *Figure 89(a),* it can be seen that DDPG is stable compared to DQN and explores a broader range of actions during the training phase. However, it also indicates that in convergence, DQN is selecting lower bandwidth values than the DDPG-based agent, i.e., 1,400 KBps. Based only on action selection, we cannot determine the best approach in terms of resource efficiency. Thus, we extend our evaluation to cover downtime comparison. In *Figure 89(b)*, the DQN-based agent, the DDPG-based agent, and the baseline solution downtimes can be viewed in the X-axis, while the Y-axis presents the downtime in seconds. The downtimes of video-streaming container are larger when compared to the blank container for all variants. The difference in these results is due to the additional copies of the network connections status. It can be also seen that the agent based on the DDPG algorithm outperforms the remaining proposed approaches in terms of downtime. Indeed, the DDPG-based agent is the only DRL algorithm which reached less than one-second downtime when migrating blank containers.



a) Selected bandwidth b) Downtime

Figure 89: Performance evaluation [42].

In the context of the XR applications, the amount of bandwidth used by the concurrent services is quite large. Therefore, carefully reserving, for service migrations, the right amount of bandwidth that minimizes the downtime is important. Some components of XR services can be very large (e.g., terrainDB of UC3.2); migrating such images should be done carefully due to the number of consumed resources such a migration operation entails, which can disturb many concurrent running XR services. As such, a balance should be found between preserving the QoS of running XR services and minimizing the transition time of the service migration while ensuring the continuous functioning of the XR service.

7 Algorithms for Network Orchestration

7.1 Deterministic Traffic Scheduling in 6G-Integrated Terrestrial and Non-terrestrial Networks

The emergence of 6G, with its promise of significantly higher data rates, lower latency, and massive connectivity, is expected to revolutionize communication experiences. One such transformative application is holography, which enables the creation of realistic three-dimensional projections that can interact with the real world. However, supporting holography effectively necessitates a robust network infrastructure that can guarantee the stringent performance requirements of these applications. Therefore, the growing demand for deterministic networking (DetNet) capabilities in future communication networks should be highlighted. DetNet guarantees specific levels of bandwidth, latency, and reliability, which are crucial for supporting mission-critical applications like remote surgery, autonomous vehicles, and industrial control systems.

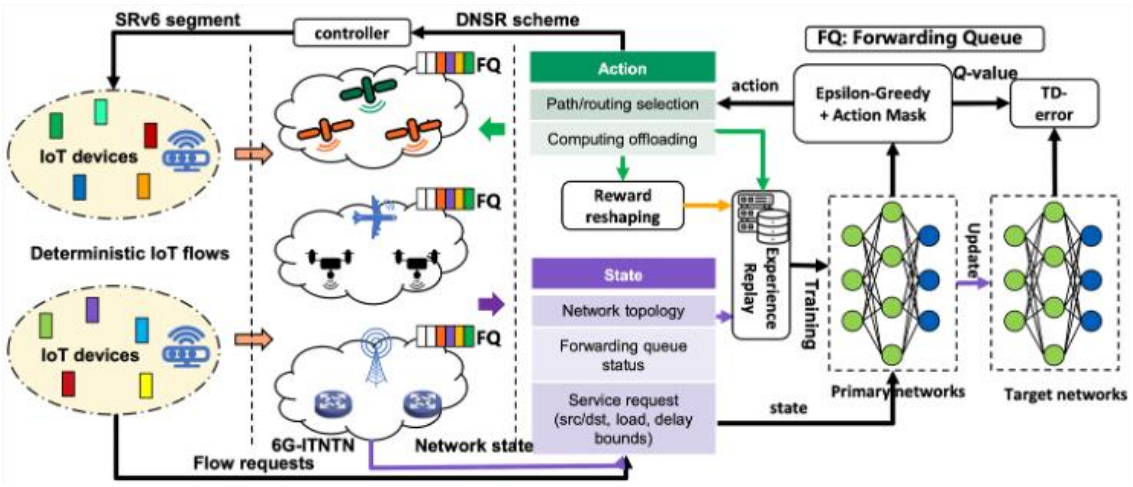


Figure 90: Deep reinforcement learning-based network selection and routing for deterministic holographic services [43].

6G-integrated terrestrial and non-terrestrial networks (6G-ITNTN), as a promising 6G network paradigm for supporting deterministic holographic services, is attracting much attention. The network and service orchestration of this network architecture should be also studied and aligned within the CHARITY framework. For this purpose, the unique feature and attribute of 6G-ITNTN will be investigated in this section. 6G-ITNTN integrates two key network types: 1) Terrestrial Networks (TNs), which include cellular communication systems, provide high capacity and wide coverage, making them suitable for handling large data volumes associated with holographic transmissions within densely populated areas. 2) Non-Terrestrial Networks (NTNs), which include Low-Earth Orbit (LEO) satellites, High-Altitude Platforms (HAPs), and Unmanned Aerial Vehicles (UAVs), can provide shorter signal transmission paths compared to terrestrial (congested) links, leading to reduced latency. They can also extend network coverage to remote and underserved areas where terrestrial infrastructure might be limited. By applying DetNet technologies over the 6G-ITNTN, network requirements for holographic services are supposed to be met, which include: 1) high bandwidth, holographic transmissions involve vast amounts of data to create realistic and detailed three-dimensional projections. The network must have sufficient capacity to handle this data flow without compromising quality, 2) low and deterministic latency, any delays in data transmission can lead to perceivable lags and disruptions in the holographic experience, requiring strict latency guarantees, 3) tight synchronization, different components within the holographic system, such as capture devices, display units, and processing servers, need to be synchronized precisely to maintain coherence and prevent visual artifacts.

In addition, two distinct communication scenarios are considered utilizing NTNs to support holographic services: 1) backhaul offloading, NTNs are primarily used for backhauling connectivity between edge servers and core networks. This offloads traffic from congested terrestrial links, particularly beneficial in densely populated areas where terrestrial networks might experience bottlenecks. By utilizing the low latency and wider coverage offered by NTNs, this scenario helps to reduce overall latency experienced by holographic traffic. 2) direct communication, this scenario explores the use of NTNs for direct communication between holographic endpoints, bypassing terrestrial networks altogether. This approach is particularly suitable for remote locations with limited or no terrestrial network coverage, enabling the deployment of holographic services in geographically diverse areas.

To guarantee the required end-to-end delays for holographic data streams, a novel Deep Reinforcement Learning-based Deterministic Network Selection and Routing (DNSR) scheme was proposed [43]. As shown in *Figure 90*, this scheme leverages deep reinforcement learning (DRL) to dynamically select network and route holographic traffic across the 6G-ITNTN with a cycle-specified queuing and forwarding (CSQF)-based traffic shaping mechanism. In detail, a DRL agent continuously interacts with the network environment, observing network metrics like link congestion, latency, and available bandwidth. Based on these observations, the agent learns to make optimal decisions about routing holographic traffic across different network paths within the 6G-ITNTN. The DRL agent's decision-making process is continuously refined through trial and error, enabling it to adapt to changing network dynamics and ensure deterministic performance for holographic services. This work delves into the unique characteristics of different NTN options, presents the trade-offs between bandwidth, computational power, and latency. This analysis helps in selecting the most suitable NTN technology for specific holographic service requirements and deployment scenarios.

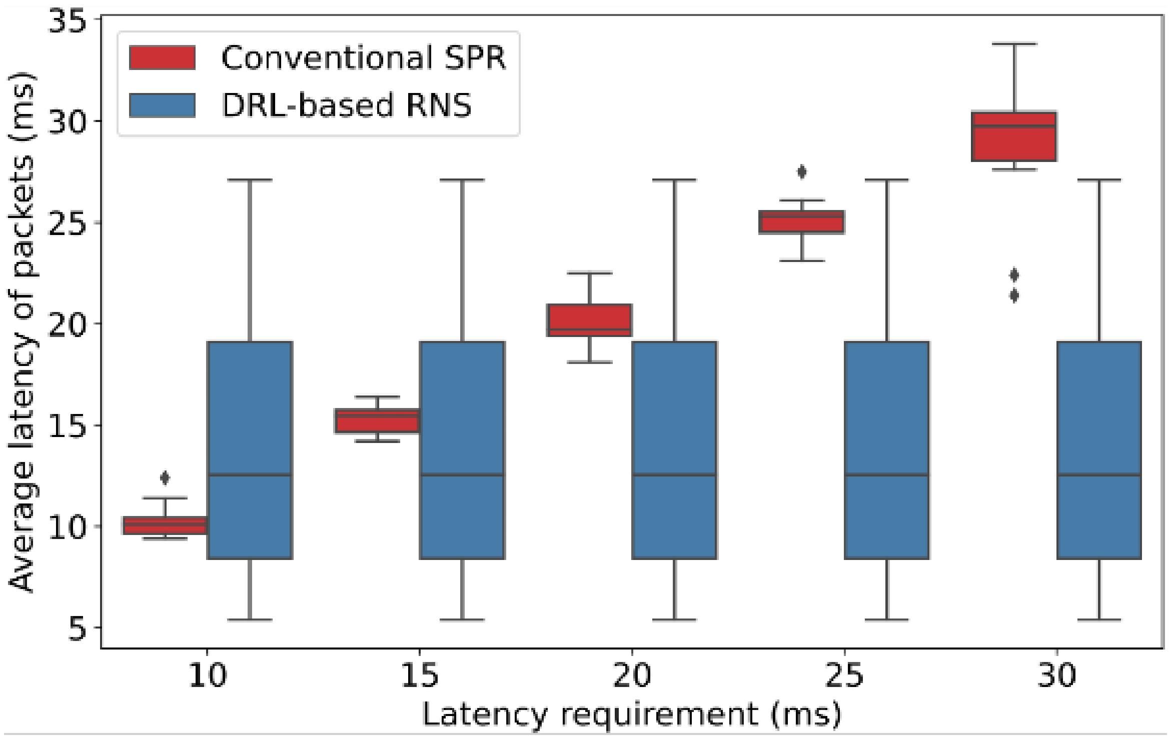


Figure 91: Comparison of latency and jitter of sub-flows with hard delay bound between conventional SPR and DRL-based DNSR [43].

The effectiveness of the proposed DRL-based DNSR approach has been evaluated. The holographic service requests are randomly generated. For each service request, two flows originated from different areas should be routed to the same application host. In the source of each flow, we assume three Kinect v2 sensors are capturing a dynamic scene from three directions at 30 fps. Each sensor provides point-cloud data with 217,088 points per frame, which gives a total of 651,264 points per frame for three sensors. For each single point, geometry characteristics are represented by 32-bit *X*, *Y*, and *Z* values, and color attributes are described with 8-bit R, G, and B values. The calculation for the total amount of data at 30 fps is 651,264 × (3 × 32 + 3 × 8) × 30 = 2.344 Gbps. Then traditional video coding techniques are applied to compress the video stream. We assume they can offer lossy compression ratio of 1:200. The packet length follows the Ethernet standard 1,500 Bytes. Fewer than 50 *ms* end-to-end latency, are defined as the latency requirement. We compare the proposed scheme to the conventional shortest-path routing (SPR) scheme. The simulations demonstrate that, compared to conventional routing methods, the DRL-based scheme significantly improves the end-to-end delay performance for holographic traffic, as shown in *Figure 91*. This highlights the potential of DRL in managing complex network dynamics and ensuring reliable support for time-sensitive applications such as XR services. As it is important to synchronize the multiple holographic images from different transmission paths, after the training of the DRL agents with the generated service requests, the DRL-based DNSR scheme can achieve a deterministic performance with much lower jitters for the sub-flows with hard delay bound than conventional SPR schemes. This is due to the fact that the conventional SPR scheme tries to search for the shortest path for each flow, regardless of the jitter of joint flows within one holographic service. On the one hand, the RL-based routing and network method will coordinate the scheduling of flows within one single service with the objective of minimizing the jitter. On the other hand, the NTN platform also advances the deterministic routing scheme with fewer intermediate forwarding nodes. The flows with deterministic latency and lower jitter can thus provide a reliable guarantee for the execution of holographic services.

7.2 Deterministic Routing and Scheduling for Mix-Criticality Flows

To support critical data flows generated by XR applications, that the CHARITY orchestration framework is supposed to support, with bounded latency/jitter and high bandwidth, the IEEE Time-Sensitive Networking (TSN) and the IETF Deterministic Networking (DetNet) work group have been initiated to study timing guarantee for critical traffic. DetNet guarantees specific levels of bandwidth, latency, and reliability for time-critical applications like remote medicine and online education. XR applications often involve a mix of flows with different criticality levels, requiring the network to prioritize and manage them effectively. Traditional scheduling methods might struggle to adapt to dynamic network conditions and changing traffic patterns, potentially compromising the performance of critical flows. In this work, we investigate the deterministic flow routing and scheduling problem in a CSQF-enabled DetNet system [44]. Initially, Cycle Queuing and Forwarding (CQF, i.e, IEEE 802.1Qch) is proposed as a peristaltic shaper which considers two queues on ports to be open and closed alternatively in a cyclic fashion. It divides the time into different cycles with an equal duration T. A packet sent from the precedent node in cycle c must be received during the same cycle in the subsequent node and then transmitted in cycle $c+1$. Although CQF can control well the delay over each hop (at most two cycles), the scalability of this mechanism is not enough since it only works well for small networks and assumes perfect synchronization between nodes. To improve scalability and flexibility, the Cycle Specified Queuing and Forwarding (CSQF) mechanism has been devised as an emerging standard draft from the IETF DetNet working group as the evolution of the CQF mechanism. CSQF is proposed to delay packets with more queues and specify a certain cycle to transmit packets. Inside a CSQF-enabled router, N queues will be equipped in each output port and N_D queues out of N ($N_D \leq N$) queues are reserved for time-critical traffic, while the remaining Non-critical (NC) queues are for best effort (BE) traffic. These N queues transmit packets in a round-robin fashion, that is, during each cycle, only one queue is active for emitting a packet to the physical link, the other ($N-1$) inactive queues are closed and enqueue packets for future transmissions. Note that the number of packets that are enqueued in each inactive queue is related with the buffer size of each queue, and improper enqueuing will incur packet loss. The N_D time-sensitive queues are dedicated to the time-critical flows by resource reservation. The assignment of packets to specific queues decides their transmission cycle, and a packet can be delayed by at most ($N-1$) cycles. This assignment can be determined by a centralized controller in advance, while the BE flows without critical timing requirements will not be scheduled in advance by the controller. When the packets of BE flows arrive at each node, they will be directly inserted into the ($N-N_D$) NC queues, whose queuing delay is not controllable or deterministic. Note that unlike CQF, CSQF operates at layer 3, as it allows to specify the routing and cycle scheduling of packets (e.g., with Segment Routing).

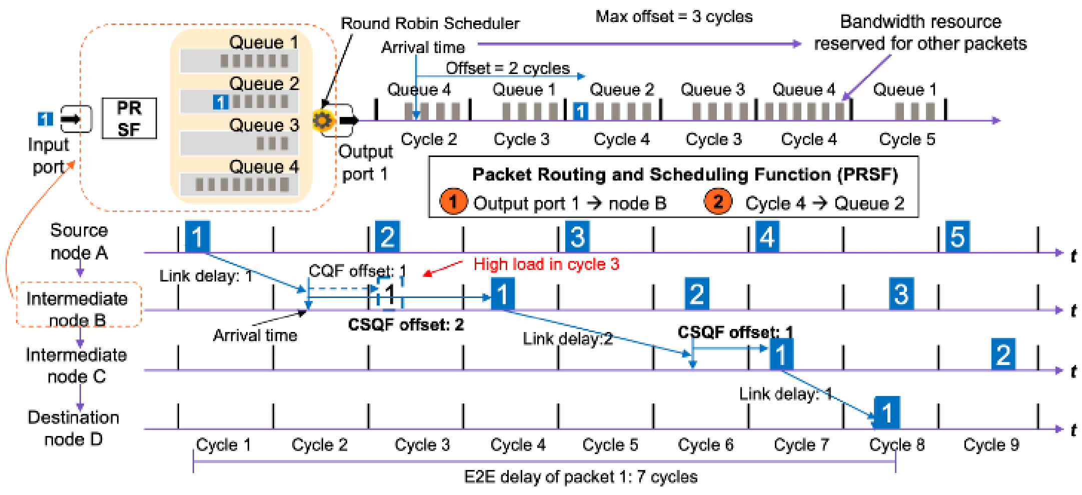


Figure 92: CSQF-based cycle scheduling of a DN flow [44].

To ensure that no collision or congestion can happen, the controller needs to decide, for each packet, where and when it will be transmitted in each node, i.e., if a packet is sent in the first available cycle or delayed by one or more additional cycles before transmission. In *Figure 92*, we show an example of how a packet is propagated from node *A* to node *D* through node *B* and *C*. We assume that 1) the link delays of $d_{A,B}$, $d_{B,C}$ and $d_{C,D}$ are one cycle, two cycles and one cycle, respectively; 2) the period of the flow of interest (FOI) is 2 cycles. Once the packets of FOI are sent from *A*, they are received at B in the next cycle (e.g., packet 1 is sent in cycle 1 at node *A* and received in cycle 2 at node *B*), since the link delay between node *A* and *B* is one cycle. Upon receiving packet 1 in cycle 2, the controller can decide to forward packet 1 in the next cycle (cycle 3). However, considering the high traffic load in cycle 3 of Node *B*, it is better to choose to delay the packet forwarding by 2 cycles (i.e., CSQF offset), that is, packet 1 is forwarded in cycle 4. Then it is received at cycle 6 due to two cycles delay between node *B* and *C*. The E2E delay of a packet is calculated as the number of cycles it costs along the path. For example, the E2E of packet 1 is 7 cycles (cycle 8 - cycle 1). What the controller should accomplish is, on the one hand, to ensure that the E2E delay of packets are within the delay bounds of this flow; on the other hand, to avoid the network congestion on certain cycles or edges.

Driven by the huge amount of the ever-increasing multimedia traffic over the Internet, particularly in the context of XR services, the network administrators need to rely on humans to design, configure and manage sophisticated and dynamic scenarios, which is not efficient and sustainable. Next-generation network automation represented by artificial intelligence (AI) based technologies is proposed to tackle this challenge. Along with the advent of the network programmability of 6G networks, the AI-enabled paradigm will carry out the intelligent automated network configuration, optimization, and management in the 6G era. We propose using Deep Reinforcement Learning (DRL) to address the challenges of routing and scheduling mixed-criticality flows in DetNet [44]. DRL-based deterministic flow scheduling (Deep-DFS) enables the network to learn and adapt to dynamic environments, making it suitable for handling complex network scenarios. The DRL process is designed as follows. **Network State Representation**: The network state is represented using features that capture relevant information, such as queue lengths, link capacities, and flow deadlines. **Action Space**: The Deep-DFS agent can take various actions, including routing decisions (selecting paths for flows) and scheduling decisions (determining the order in which flows are transmitted). **Reward Function**: The agent receives rewards based on its actions' impact on the network performance. The reward function is designed to incentivize the agent to prioritize critical flows while maintaining overall network stability. **Learning Process**: Through continuous interaction with the network environment, the Deep-DFS agent learns to make optimal routing and scheduling decisions that maximize the reward.

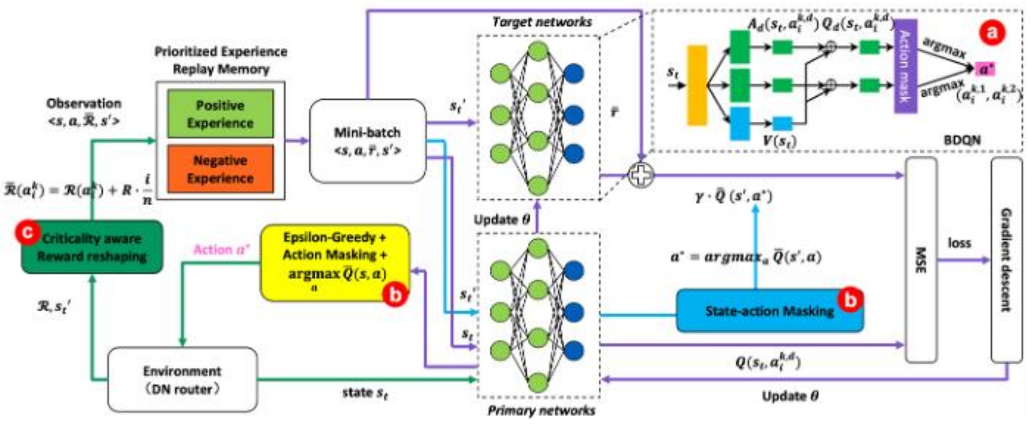

Target networks
Prioritized Experience Replay Memory
Observation <s, a, R̃, s'>
Positive Experience
Negative Experience
Mini-batch <s, a, r̃, s'>
$\tilde{\mathcal{R}}(a_t^k) = \mathcal{R}(a_t^k) + R \cdot \frac{i}{n}$
Criticality aware Reward reshaping
Action a'
Epsilon-Greedy + Action Masking + argmax Q̃(s, a)
R, s_t'
Environment (DN router)
state s_t
Update θ
Primary networks
$\gamma \cdot \bar{Q}(s', a^*)$
$a^* = argmax_a \bar{Q}(s', a)$
State-action Masking
$Q(s_t, a_i^{k,d})$
Update θ
MSE
loss
Gradient descent
BDQN
Action mask
argmax
$V(s_t)$
a
b
c

Figure 93: Branching dueling Q-networks based learning process [44].

As shown in *Figure 93*, in the RL training design, we leverage a branching dueling Q-networks to generate the policy, we also propose a) a novel delay-aware network representation that incorporates information about flow deadlines and queueing delays, enabling the agent to make informed decisions considering both bandwidth and latency constraints; b) action masking to ensure the agent only takes feasible actions, in other words, restricting the agent from selecting invalid routing or scheduling options based on the current network state; and c) criticality-aware reward function that explicitly considers the criticality levels of flows, encouraging the agent to prioritize critical flows while still ensuring fairness and efficient resource utilization for non-critical flows.

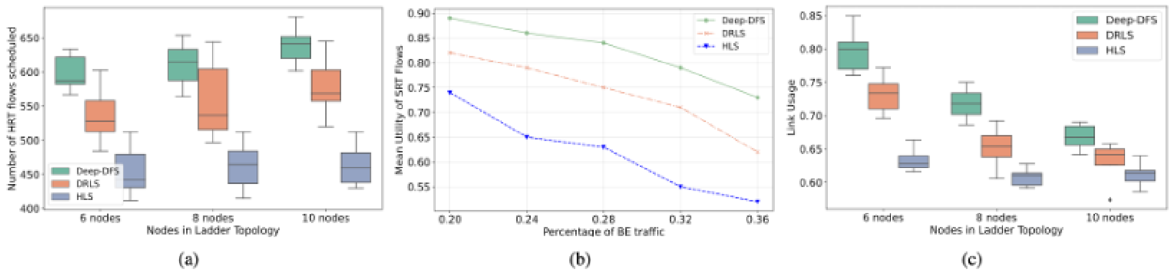


Figure 94: (a) Number of HRT flows scheduled; (b) Utility of SRT flows; (c) Link Usage with different nodes in ladder topology [44].

We evaluated the performance of Deep-DFS through extensive simulations. The results demonstrate that Deep-DFS outperforms traditional heuristic-based: heuristic list scheduler (HLS) and existing RL-based scheduling methods (DRLS) in terms of 1) Flow acceptance rate – Deep-DFS schedules more flows successfully, especially critical flows, compared to other methods; 2) Average flow completion time – Deep-DFS achieves lower average completion times for critical flows, ensuring timely delivery of critical data; and 3) Network resource utilization – Deep-DFS efficiently utilizes network resources while maintaining fairness among different flow types. As shown in *Figure 94(a)*, Deep-DFS can schedule more HRT flows than the other two methods in general, specifically, 14.8% more on average than DRLS, 32.1% more on average than HLS in ladder networks. Since HLS always selects the first available time slot to transmit the packets on the shortest path, it will derive the minimum latency for all flows regardless of the flow criticalities and their delay bounds. Therefore, HLS will saturate the cycles and edges soon, and increase the probability of flow blocking by some fully occupied cycles. DRLS considers the cycle usage on the edge, it avoids selecting the cycles with high degree to save more bandwidth for the flows with different periods. However, DRLS still tries to select the first available cycle for packet forwarding and minimizes the E2E delay of the flow, which conflicts with the long-term objective of maximizing the number of flows scheduled in this system. To this end, Deep-DFS redesigns the network state representation and reward function to make delay-aware decisions on edge and cycle selection, so that the HRT flows are prioritized, and no bandwidth resources are wasted on minimizing the E2E delay. When the size of the ladder topology becomes larger, Deep-DFS schedules more flows (39.1% more than HLS on average) in the ladder topology with 10 nodes than with 6 nodes (29.3% more than HLS on average). This is because Deep-DFS has more exploration space in a larger topology, while HLS only selects the shortest paths to route the flows regardless of topology size. We also evaluate the average utility of SHR flows with the percentage of BE traffic. We set the probability of generating BE traffic from 0.2 to 0.36, with HRT and SRT flows are generated with the same probability. As we can see in *Figure 94(b)*, it is obvious that the average utility will decrease with more BE traffic inserted in the network. With the increasing BE traffic, the network bottleneck will come earlier and the utility of SRH flows in the HLS case will decrease a little faster than the other two methods due to the selection of shortest paths, as discussed above. The link and cycle usages are also shown in *Figure 94(c)*. The results show that although the HLS method will lead to more cycles with high traffic load (>60%), the link usage induced by HLS is lower than that of Deep-DFS, which is not intuitional. This is because, on the one hand, the resources are exhausted in earlier cycles by minimizing the E2E delay with HLS, though, HLS also stops to schedule flows earlier than Deep-DFS. That makes the overall link usage of HSL lower than Deep-DFS by 21.3% on average in the ladder topology. We also find that the link usage decreases slightly with more nodes in the ladder topology for Deep-DFS, as it prefers to choose a longer route to balance the link load and avoid the bottleneck.

7.3 XR-aware Dynamic Routing Strategy

Routing schemes on SDN are generally classified into two types, namely static and dynamic. In static routing, existing solutions focus on extending well-known path searching algorithms such as Dijkstra or Depth First Search (DFS) to find paths from source-to-destination while considering edge and/or node weights. However, the selected path will not change unless a link failure is detected. In this way, the route failure or link congestion results either in drop or waiting for packet transmission. To overcome the drawback of static routing, dynamic routing (also known as adaptive routing) has been proposed, where routing is performed based on the current situation of the network. Research in dynamic routing aims at providing more efficient usage of network resources by considering the current load of each link in the network while making routing decisions.

In a SDN-based routing framework, the SDN controller has three common routing tasks [45]:

- Obtaining the global view of the network: the SDN controller needs to acquire the accurate global view of the underlying network in order to make routing decisions and compute the new paths. The global view of the network mainly contains the network topology and link status information from the switches in the data plane. Various protocols are available for the SDN controller to discover the network topology and obtain the static link status information (e.g., hop count, link capability) at the same time since these static information does not change[46]. To collect the dynamic link status information (e.g., available bandwidth, utilization, and delay) which do often change, the SDN controller needs to continuously query the switches at very short intervals to maintain an accurate view. This imposes significant protocol overhead. To address this challenge, the SDN controller usually implements a periodic monitoring mechanism to obtain dynamic metrics from each switch at a predetermined rate. The value of this rate should be selected carefully to balance the trade-off between accuracy and protocol overhead.
- Computing the routing paths: the SDN controller computes the optimal path(s) for a given flow using routing algorithms which assume that the global view of the network is available. Based on the number of output optimal paths between source-to-destination nodes, existing solutions can be classified into two categories, namely, single-path and multipath. While single-path routing protocols are designed to discover and use single path between a source and a link, the multipath routing protocols make use of multiple routes so that the traffic is balanced among the number of available paths. Therefore, multipath routing provides better overall performance by allowing better sharing of available network resources.
- Installing the forwarding rules: After computing the optimal paths, the SDN controller needs to install or update the necessary rules on the forwarding table of each switch using OpenFlow. The switches then use these rules to forward packets. The mechanism of updating routing

information (i.e., who and how to start computing new routes and installing forwarding rules) is an essential part of any routing protocol. There are three modes of updating operation:

- Reactive (on-demand) mode: the routing path is discovered by the SDN controller only when a source node needs to send some data to a specific destination node. The nodes do not maintain table regarding routing information. In this way, this mode consumes less resources due to the absence of large routing tables. However, it causes performance delay because of continuous communication between nodes and the controller.
- Proactive (table driven) mode: the SDN controller installs the forwarding rules to switches for possible traffic in advance. The advantage of this mode is faster routing decision and less delay in route setup process. However, each node is required to keep the routing table up to date which needs large routing overhead.
- Hybrid mode: It combines reactive and proactive techniques. Accordingly, this mode obtains the flexibility of reactive mode in providing fine-grained control while benefiting from proactive mode by avoiding significant burden on the controller.

With the emergence of SDN, the flow routing in the network can be flexibly managed and adjusted in a timely manner according to the current network status. Various existing works have been proposed in the routing optimization that can be classified into different categories according to the underlying approach that they use in their algorithms:

- Path searching algorithm – based approach: Existing works in this approach implement the concept of multipath routing on SDN mainly based on modifying well-known path search algorithms such as Dijkstra and DFS. The authors in [47] apply Equal-cost multipath (ECMP) routing scheme to find all available paths on Fat-Tree network topology. ECMP utilizes modified Dijkstra's algorithm to search for the shortest path and uses the modulo-n hashing method to select the delivery path. In [48], a modification of DFS to adapt multipath routing concept and Open shortest path first (OSPF) distance estimation technique is used to estimate the minimum distance. Similarly, the work in [49] implements the modified DFS and measures the paths weight by combining the node, edge, path, and bucket weight using port statistics available in OpenFlow standard and manual calculation.
- Constraint-based approach: Due to finite resources in the network, main attributes of a routing algorithm are determined by the flow's characteristics such as demand or the type of application data that flow is carrying. These parameters define the constraints in finding network paths for flows. Most of algorithms simply eliminate the links whose residual static and dynamic metrics are less than the requested demand and then uses path search algorithms to find the optimal paths [45][50] . Other existing works [51][52] follow the declarative and expressive approach which applies Constraint Programming (CP) techniques to find the optimal paths. Accordingly, the constraint-aware routing problem is represented as constraint satisfaction and optimization problems in CP. The developers only state the constraints and optimization statements that the solution should have and do not specify a step-by-step solution of the problem. The solution is provided by a powerful general purpose CP solver.
- Heuristic algorithm – based approach: Searching for exact optimal paths may be unfeasible in a reasonable time for a large network. Heuristic routing determines close-to-optimal, although not always optimal, solutions in a fixed amount of time. Most of existing research on heuristic-based routing are based on evolutionary algorithms in which among of them, Genetic Algorithm (GA) and Ant Colony Optimization (ACO) are the most popular used. The work in [53] developed an evolutionary multipath routing algorithm based on GA to solve the multi-commodity flow problem while authors in [54] leveraged GA and incorporates a fitness function inspired by RL for the priority flow admission and routing problems. The solution in [55] proposed a dynamic routing algorithm based on ACO with three modules to compute multi paths, select optimal path, and validate the optimal path. Following similar approach,

authors in [56] introduced different ant colonies to ACO to calculate multiple paths and reduce the coincidence rate between these paths.

- Machine learning –based approach: Many studies have proposed to optimize the routing problem on SDN using ML-based algorithms in order to enhance learning ability from past experiences and smart route-decision capability. The existing solutions can be classified into two categories[57]:
    - Supervised learning – based solutions mainly consist of three phases: (1) collecting labeled training datasets, (2) establishing ML-based model in the control plane with the training data, (3) applying the trained model for dynamic routing. Preparing a set of adequate data for training is an essential step in this approach. In general, to construct a training dataset, the network and traffic states are often considered as input and the corresponding routing solution (normally provided by heuristic algorithms) are the output. In [58][62], authors introduced NewRoute, a ML-based dynamic routing Framework, which applies Long Short-Term Memory (LSTM) networks to estimate future network traffic. NewRoute uses this traffic estimation together with the network state and corresponding routing solution calculated by a so-called baseline heuristic algorithm to train the DNN model which is responsible of selecting optimal routes. In the same principal, Awad et al. [59] proposed ML-based multipath routing framework which learns the mapping function between network configuration and their routing solution calculated by a column generations-based heuristic algorithm.
    - Reinforcement learning – based solutions consider the routing optimization as a decision-making task, the SDN controller as an agent and the network as the environment. In this approach, the state space contains the network and traffic states. The action is the routing solution and the reward is defined based on optimization metrics. Research in[60] proposed a mechanism dubbed DROM, a routing optimization mechanism based on Deep Deterministic Policy Gradient (DDPG) [61], to realize the global, real-time and customized network intelligent control and management in continuous time. A routing algorithm based on Deep Q-Learning in [62] combines NN with RL via replacing Q-tables with an approximate function trained by NN. Rischke et al. [63] designed and evaluated QR-SDN, a tabular RL approach, which directly represents the flow routes in Q-Learning state and action spaces to enable multipath routing.

In CHARITY, we aim at developing a dynamic multipath routing framework (*Figure 95*) to improve the end-to-end communication in the context of the strict requirements of AR, VR, and holography-based applications. To do so, it is essential to develop mechanisms which can facilitate the scheduling and routing of latency-sensitive and / or bandwidth-sensitive traffic. The component which shall be in charge of providing these functionalities is referred to as the Intelligent Traffic Routing mechanism. The Intelligent Traffic Routing mechanism leverages information regarding the various traffic flows, the network topology and the network state in order to establish traffic routing and scheduling functionalities in a manner which is compliant with QoS requirements. The required information which relates to the traffic flows are their corresponding source, destination and QoS requirements. Furthermore, the Intelligent Traffic Routing mechanism shall also utilize network traffic predictions which are provided by the Traffic Prediction mechanism.

The Intelligent Traffic Routing Mechanism leverages SDN to have access to vital information regarding the traffic and topology of the network. The SDN controller is able to use Northbound APIs to establish communication with the application plane and Southbound APIs, such as OpenFlow, in order to communicate with the forwarding devices. Furthermore, the SDN controller examines the network state and flow-related information and then alters the flow table of the forwarding devices accordingly.

The Intelligent Traffic Routing Mechanism is designed to leverage RL to conduct these functionalities in an optimal manner which is in line with the QoS requirements. The centralized control provided by SDN greatly enhances the quality of RL-based traffic engineering by enabling network policies to be centrally generated and then transferred to the forwarding devices. The formulation of the agent's Action Space is made in a manner which is in accordance with the SDN paradigm. Two different implementations of the Action Space have been created up to this point. The first one matches different available paths to pre-defined Actions that the agent may take, in order to achieve multi-path routing. The second one is a novel approach that we developed that allows the implementation of weighted multi-path routing in the context of DRL. According to this approach each potential action is matched to a distinct combination of potential percentages, each of which corresponds to a specific path. That way, all of the various available paths can be leveraged at the same time. In both of the aforementioned cases, the action is applied during specified time-intervals.

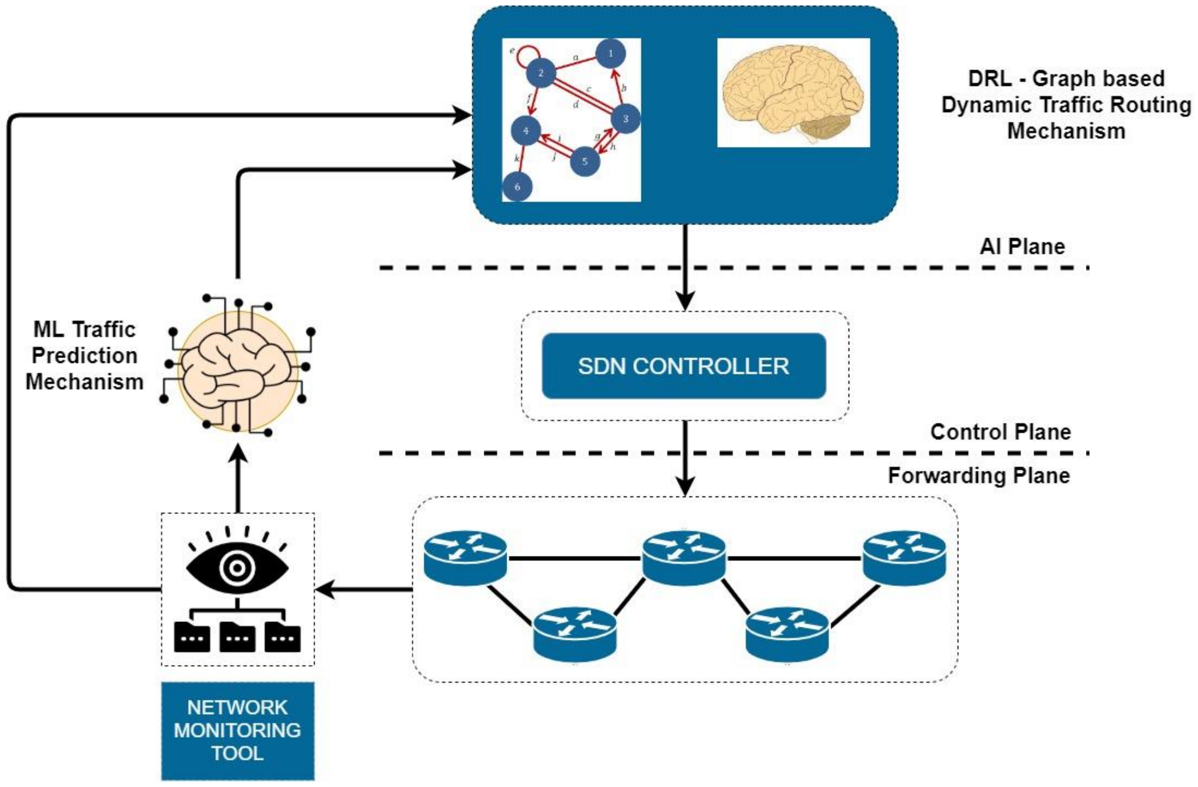

DRL - Graph based Dynamic Traffic Routing Mechanism
AI Plane
ML Traffic Prediction Mechanism
SDN CONTROLLER
Control Plane
Forwarding Plane
NETWORK MONITORING TOOL

Figure 95: Dynamic multipath routing framework.

Although there have been numerous scientific endeavours applying RL-based paradigms in the context of SDN, only a few of them are designed to accommodate multipath routing while taking into consideration the QoS constraints. CHARITY aims to expand upon the current scientific literature in regard to developing QoS-aware RL-based structures which support multipath routing. To that end, the Action Space should be also modelled in a manner which can properly reflect the intricacies of multipath routing. Furthermore, the State Space shall be implemented in a manner which includes the traffic predictions. By doing so, it is possible to enable the creation of policies that take into consideration the future state of the network as well as the ongoing one. Finally, the Intelligent Traffic Routing Mechanism shall also leverage Graph Neural Networks (GNNs) to enhance the efficiency of the RL-based routing algorithm. The use of GNNs shall enable the network structures to be represented in a more accurate way by properly encapsulating the intricate relations which are established among graph-based structures.

7.4 Asynchronous Traffic Scheduling for Deterministic Networking

In this section, we delve into the application of asynchronous Time Sensitive Networking (TSN). Although asynchronous scheduling increases the latency compared to synchronous one, it improves the network scalability and the link utilization as it does not require a network-wide coordinated time to schedule the traffic transmission of each stream over reserved time slots. Asynchronous TSN is ideal for conveying sporadic traffic with real-time constraints and allowing its coexistence with best-effort streams. It deemed to be of high importance for the support of highly-interactive holographic communication services, particularly over small-scale networks.

The building block of asynchronous TSN is the IEEE 802.1Qcr Asynchronous Traffic Shaper (ATS), which is based on the Urgency-Based Shaper (UBS) proposed by Specht and Samii [64]. ATS enhances traditional asynchronous scheduling, in which a set of First Come, First Served (FCFS) queues, each associated with a traffic class and a priority level, are arbitrated by a strict priority transmission selection scheme. Specifically, ATS adds traffic regulation to conventional asynchronous schedulers cost-effectively. In this way, per-hop traffic regulation is enabled in the network, thus avoiding the burst size or burstiness of the streams grows when they traverse the network, and the worst-case delay becomes arbitrarily large [65] .

Given an optimization goal, such as the maximization of the flow acceptance ratio, the flow allocation involves the optimal selection of one or several paths for every Traffic Class and optimally finding the configuration for every ATS included in paths. These decisions are subject to the QoS constraints fulfilment of the incoming flows and all the ongoing flows. For critical flows, typical E2E performance requisites are the following:

- Frame delay budget: the upper bound for the time the network takes to transport a packet between the source and the destination.
- Maximum jitter delay: the permitted delay variation in the frame delivery from the source to the destination.
- Frame loss ratio: the fraction ofthe frames that are lost when they traverse the network.
- Reliability: the probability of network success to carry out the communication and fulfil the flow’s required service level during its entire lifetime.

Synchronous TSN is suitable to transport performance sensitive traffic with periodic patterns such as closed-loop control systems in Industry 4.0. Conversely, asynchronous TSN networks perform well in scenarios where deterministic aperiodic (or sporadic) and best-effort traffics are predominant. However, the exact number of flows to be allocated, and their features are often unknown in these scenarios. Thus, the flow allocation in asynchronous TSN networks is a stochastic optimization problem in nature. There are two approaches for performing the flow allocation in TSN networks, namely offline and online methods. Online methods compute the flow’s allocation configuration right after it arrives at the network. Hence, they might run an optimization algorithm to find the allocation for every incoming flow. Conversely, offline methods compute a long-term configuration for the whole network for each type of traffic. Offline methods require less state information (i.e., same configuration for all the flows of a traffic type), and the access control mechanism becomes a lightweight process that only needs to check whether there are enough resources (links capacities and buffer space) for the incoming flow. Conversely, online methods offer higher flexibility (i.e., flows with the same traffic type might have different configurations) and greater agility to adapt to the changing network conditions.

Figure 96 sketches a blueprint of a possible management and orchestration framework for transport networks based on ETSI ZSM and IETF ACTN (Abstraction & Control of Transport Networks) reference models. This architecture enables the customer to create and operate Virtual Networks (VNs) (Transport Network slicing) while hiding the complexity of the underlying physical infrastructure. Also, it provides cross-domain coordination, which is crucial to ensure the cohesion and satisfiability of the configurations applied to the distinct domains. For instance, the E2E delay budgets imposed by the services need to be distributed among the different network domains. A fully centralized (SDN-like) TSN network is considered given that we are targeting deterministic single digit delays (i.e., less than

9ms).

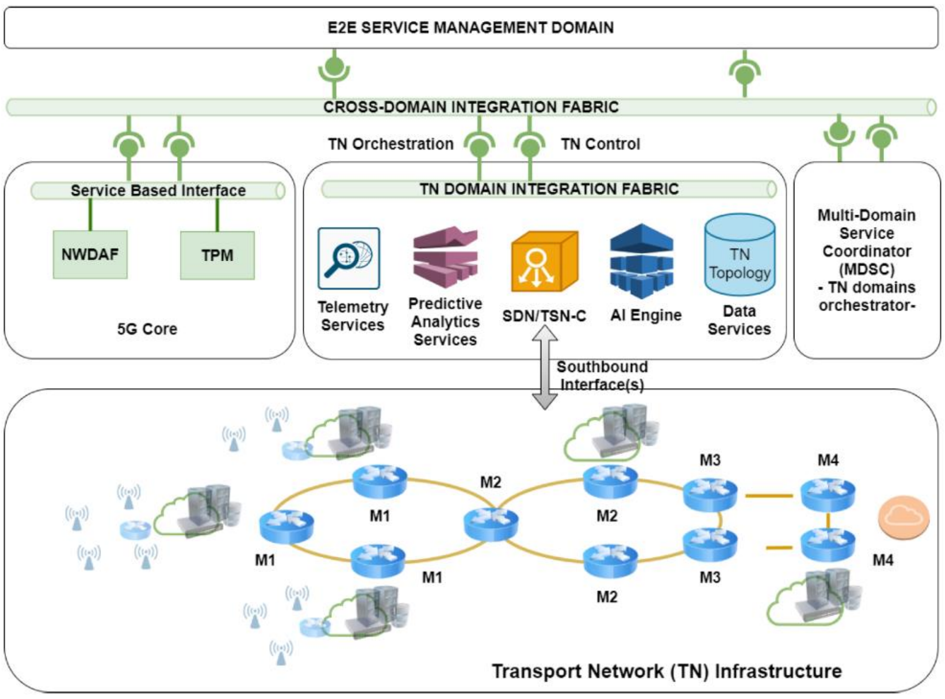

E2E SERVICE MANAGEMENT DOMAIN
CROSS-DOMAIN INTEGRATION FABRIC
TN Orchestration
TN Control
Service Based Interface
NWDAF
TPM
5G Core
TN DOMAIN INTEGRATION FABRIC
Telemetry Services
Predictive Analytics Services
SDN/TSN-C
AI Engine
TN Topology
Data Services
Multi-Domain Service Coordinator (MDSC) - TN domains orchestrator-
Southbound Interface(s)
M1
M2
M3
M4
Transport Network (TN) Infrastructure

Figure 96: Transport network management and orchestration architecture [66].

In the ambit of CHARITY, a study [67] was conducted where deep RL was employed to solve the flow allocation problem in asynchronous TSN networks as its features are well suited for that problem. In contrast to alternative ML techniques, deep RL supports online learning efficiently, which is advisable for the model adaptability to the changing network conditions. In the same way, the RL exploration capability also allows adapting the agent's decision policy. On the other side, deep RL can handle large state-action spaces as required in medium and large scale TSN networks. Last, RL might act alone to output the solution directly from the input without any restriction on the optimization objective.

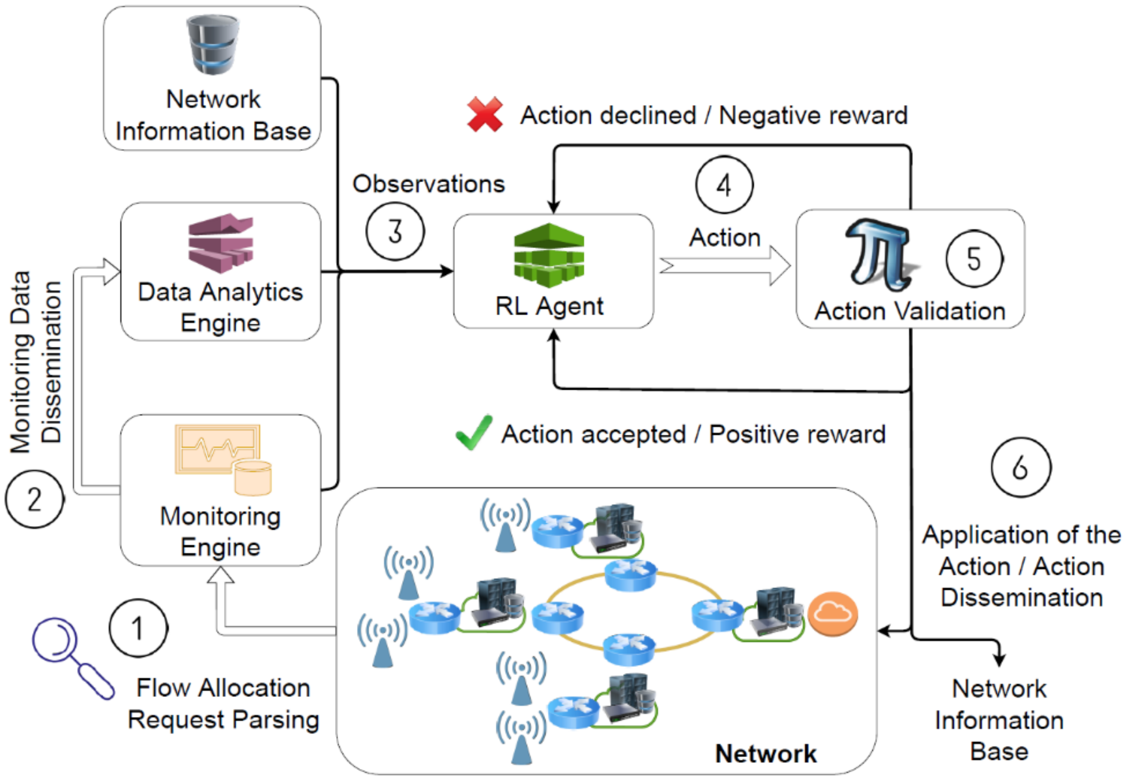
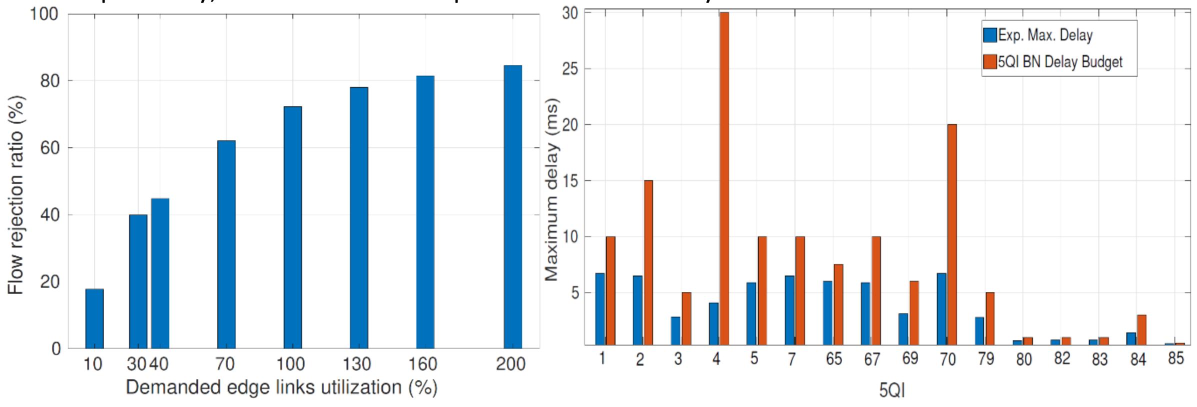

Network Information Base
Action declined / Negative reward
Observations
3
4
Action
RL Agent
5
Action Validation
Data Analytics Engine
Monitoring Data Dissemination
2
Monitoring Engine
Action accepted / Positive reward
6
Application of the Action / Action Dissemination
1
Flow Allocation Request Parsing
Network
Network Information Base

Figure 97: RL for flow allocation and time-sensitive networks optimization [67].

Figure 97 shows an online RL-based solution for the flow allocation policy-making in ATS-based networks [67]. First, every incoming flow allocation request is parsed to determine the flow type and characteristics (step 1). Then the flow characteristics, along with the traffic predictive data analytics and the network information and status, are taken by the agent as observations. Next, the agent outputs an action, which is validated through verifying analytically that the action would not impact anyhow the deterministic performance requirements of this and already existing flows' allocations. If the action is validated, the agent will be positively rewarded, and the action applied. Otherwise, it is simply not applied. In this way, the analytical models' information is transferred to the agent, and, most importantly, the flow allocation process becomes fully reliable.



a) Flow rejection ration

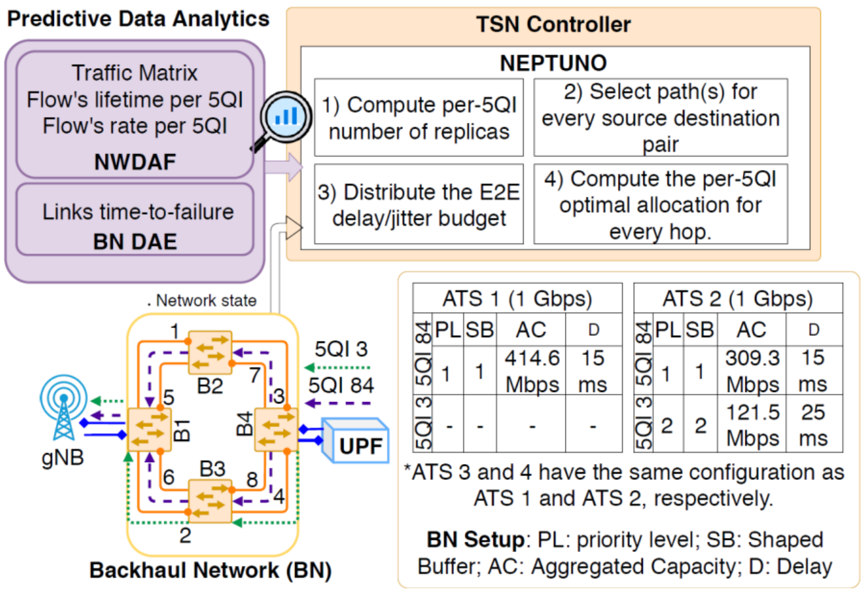
b) Worst case delay

Figure 98: ATS-based Backhaul Network (BN) performance [67].

Figure 98(a) depicts the flow rejection ratio as a function of the demanded link utilization at the access links interconnecting M1 devices and gNodeBs (see *Figure 96*). Every point shown in *Figure 98(a)* was obtained via simulating the arrivals and departures of 1.8M of flows. As observed, the flow rejection ratio depends logarithmically on the demanded edge link for the setup. It can be seen that the algorithm offers high rejection probability (penalizes) flows with high data rate demands, e.g., those with 5QIs 2, 7, and 5 (5G QoS Identifier – as defined in 3GPP TS 23.501[31]), as it seeks for maximizing the number of accepted flows. *Figure 98(b)* shows both the BN delay budget per 5QI (labelled as "5QI BN Delay Budget"), which is 10% of the E2E delay budget defined in 3GPP standards, and the worst-case delay per 5QI obtained through simulation (labelled as "Exp. Max. Delay"). As observed, the delay constraint is met for every 5QI. The maximum delay experienced by each 5QI primarily depends on its priority level in the TSN network, which is assigned by the algorithm. This fact explains the variability observed in the obtained maximum delay for the different 5QIs.

In a second work [67] also performed in the context of CHARITY, the allocation problem was also investigated in 5G backhaul, wherein an offline solution dubbed "Next Generation Transport Network Optimizer" (NEPTUNO) was proposed. It combines exact optimization methods and heuristic techniques and leverages data analytics to solve the flow allocation problem. NEPTUNO aims to maximize the flow acceptance ratio while guaranteeing the deterministic QoS requirements of the critical flows. NEPTUNO makes the following decisions: i) clustering of 5G streams into IEEE 802.1Q classes according to the 5GQI value, ii) flow-to-shaped buffer and flow-to-priority assignments at each ATS of the network, iii) paths selection to interconnect every source and destination, and iv) distribution of the end-to-end delay/jitter budget of the flows among the hops comprising each path in the network.

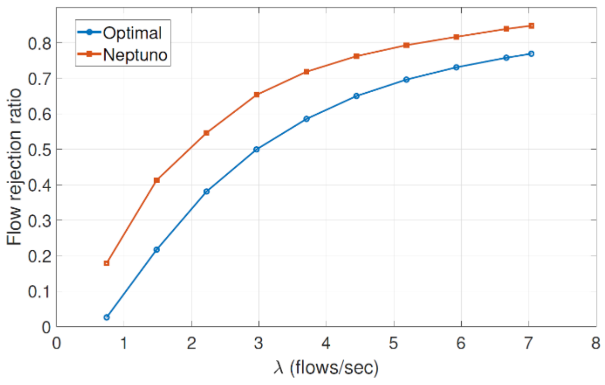
[31] For example, 5QI 3, 7, and 80 refer to real-time gaming services, live video, and augmented reality services, respectively.


Predictive Data Analytics
Traffic Matrix
Flow's lifetime per 5QI
Flow's rate per 5QI
NWDAF
Links time-to-failure
BN DAE
TSN Controller
NEPTUNO
1) Compute per-5QI number of replicas
2) Select path(s) for every source destination pair
3) Distribute the E2E delay/jitter budget
4) Compute the per-5QI optimal allocation for every hop.
Network state
5QI 3
5QI 84
gNB
UPF
B1
B2
B3
B4
Backhaul Network (BN)
*ATS 3 and 4 have the same configuration as ATS 1 and ATS 2, respectively.
BN Setup: PL: priority level; SB: Shaped Buffer; AC: Aggregated Capacity; D: Delay

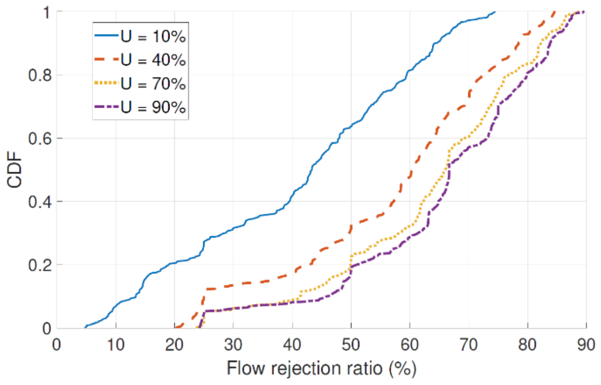
	ATS 1 (1 Gbps)				ATS 2 (1 Gbps)			
	PL	SB	AC	D	PL	SB	AC	D
5QI 84	1	1	414.6 Mbps	15 ms	1	1	309.3 Mbps	15 ms
5QI 3	-	-	-	-	2	2	121.5 Mbps	25 ms

Figure 99: Main stages of NEPTUNO for computing the optimal configuration of the network and an example illustrating the primary configuration parameters for two 5Qis [67].

Figure 99 shows the main steps of NEPTUNO to make its decisions. First, NEPTUNO collects the data analytics of interest and network state information. Next, it executes the optimization algorithm for finding the optimal configuration of the network. The first step of the optimization process is to compute the number of packet replicas required for each 5QI in order to assure its minimum reliability. It runs, then, a path selection algorithm whose objective is to balance the workload through the different transit links of BN. To that end, it uses the number of required flow replicas computed in the previous step and data analytics A1 and A2 as input. After that, it distributes the delay/jitter budget among the hops of each path chosen in the previous step. Finally, it computes the optimal configuration for each last hop by solving the MILP problem.



a) Degree of optimality



b) Empirical CDF of the flow rejection when varying workload

Figure 100: Performance of NEPTUNO [67].

Figure 100(a) compares the flow rejection ratios offered by NEPTUNO and by the optimal solution versus the flow arrival rate. As observed, the flow rejection ratio exhibited by NEPTUNO is roughly 20% above the optimal one for low workloads and 10% above for high workloads. That is because of NEPTUNO's operation. More precisely, NEPTUNO configures the last hops to minimize the flow rejection probability, whereas the configuration of the transit ATSs is set in such a way that the per-5QI reserved capacity in the last hops can be accommodated. Thus, it seems reasonable that NEPTUNO performs better for high workloads where the configuration of the bottleneck link (last hop) becomes increasingly important. *Figure 100(b)* depicts the empirical cumulative distribution functions of the flow rejection ratio offered by NETPUNO under different workloads. As observed, the characteristics and performance requirements of the flows matter. For instance, for the same characteristics of the flows (e.g., committed rate and burst size), the rejection ratio increases when the flows' constraints are more stringent. Even if NEPTUNO is targeted towards 5G networks, the solution is general enough to be adapted towards other networks as long as the underlying networking infrastructure supports ATS switching with priority queues. This also depends on the level of ownership of the network elements.

8 Algorithms for Cloud Orchestration

In this section, we will outline design challenges that should be considered while proposing a computation resource management system that is capable of addressing the requirements of the different workloads brought by the use cases introduced by CHARITY and to meet the target KPIs defined in D1.2. The most appropriate resource management techniques need to be selected to ensure the efficient sharing of distributed computational resources between multiple users. In this regard, an autonomous orchestration framework, based on Artificial Intelligence (AI), is envisaged to orchestrate and manage the computation resources. To achieve the objectives and KPIs of the target use cases (as in D1.2), we look to use a combination of machine learning, cloud computing, micro-services, and the ETSI Zero-touch network and Service Management (ZSM) concept (reference D1.3 architecture).

AI and Machine learning (ML) techniques (supervised, unsupervised, and deep learning) can be used primarily for predicting computation utilization, then for resource allocation. The framework is executed on top of geographically distributed infrastructures consisting of heterogeneous computation resources. Technology enablers and concepts, such as Multi-access Edge Computing (MEC) will play an important role in meeting the design challenges, while the use of micro-services allows for efficient service deployment across MEC environments and clouds. The framework will use AI and ZSM for the orchestration and management of distributed computation resources across MEC environments and cloud infrastructure. This should provide the necessary abstraction layer to hide the resource heterogeneity while optimizing geographical distribution.

In terms of the infrastructure layer, this spans across public cloud infrastructure with virtually limitless processing power to end-user devices with relatively limited computational capabilities. CHARITY use cases bring advanced media applications, each with their own computational requirements. According to this, we must take into account the resource requirements of these applications as any data processing, either at the device end or in the cloud, must be able to meet the defined KPIs, and QoE. It is therefore important to cope with the relevant challenges, due to the limitations in terms of both network and computing capabilities. Effectively, for XR services, compute capacity is not the only issue. The efficient utilisation of the underlying networking resources is equally highly important. To tackle this challenge, the resource management system will need to handle load balancing and optimise resource availability in a more autonomous way. Essentially, the network would use AI and machine learning to learn and adapt to changes in the network in real time. Automation should extend across the entire network requiring very little human interaction.

From a cloud service provider (CSP) perspective, support of hybrid, multi-cloud environments is of fundamental importance to ensure maximum flexibility and to mitigate the risk of vendor lock-in. Containerized environments will go some way towards ensuring increased portability, as they provide a streamlined way to build, test, and deploy applications across multiple environments offering seamless integration with a CI/CD (Continuous Integration/Continuous Development) pipeline.

From here on, we investigate various resource management techniques as well as open-source software tools to improve the usability and utilisation of resources to efficiently deploy XR applications across a large heterogeneous resource pool. The key here will be to reduce the need for traditional cloud infrastructures as much as possible while meeting the KPIs and QoE of the target XR services.

An edge/cloud approach facilitates the AI to be distributed across the network from the core to the access layer. Such distribution of AI across the network can be beneficial only if it is done in an optimal manner; i.e., if implemented badly, the necessary data sharing could consume substantial network capacity. It should then alleviate the low-latency considerations required by the use cases and ensure that resources are appropriately allocated in real time. The cloud is also essential to creating dynamic and flexible network environments with real-time updated network performance and management dashboards.

In this regard, an intent-based automated management and orchestration (MANO) approach may be essential to achieve optimal network performance allowing for autonomous operations and a zero-touch operational model. NFV MANO (refer to D1.3) includes all the essential management modules to coordinate network resources in the NFV architectural framework. It includes three Managers: NFV Orchestrator (NFVO), VNF Manager (VNFM), and Virtualized Infrastructure Manager (VIM).

The NFV Orchestrator is responsible for the on-boarding of new Network Services (NS), VNF-Forwarding Graph (VNF-FG) and VNF Packages NS lifecycle management (including instantiation, scale-out/in, performance measurements, event correlation, and termination), global resource management, validation and authorization of NFVI resource requests policy management for NS instances. The VNF Manager provides lifecycle management of VNF instances, overall coordination and adaptation role for configuration and event reporting between NFVI and the E/NMS (Element/Network Management System). The Virtualised Infrastructure Manager (VIM) controls and manages the NFVI compute, storage and network resources, within one operator's infrastructure sub-domain and handles the collection and forwarding of performance measurements and events.

There are also four repositories that hold different information in NFV MANO: VNF Catalog, Network Service Catalog, NFV instances, and NFVI resources. The VNF Catalog is a repository of all usable VNF Descriptors (i.e., a deployment template that describes the requirements of a VNF deployment and operational behaviour). It is primarily used by VNFM in the process of VNF instantiation and lifecycle management of a VNF instance, as well as by the NFVO to manage and orchestrate Network Services and virtualized resources on NFVI. The Network Services (NS) Catalog provides the usable Network services (i.e., a deployment template for a network service in terms of VNFs and description of their connectivity through virtual links is stored in NS Catalog for future use). The NFV Instances list holds all the details about Network Services instances and related VNF Instances. NFVI Resources is a repository of NFVI resources used for the purpose of establishing NFV services.

8.1 Cloud-Network Integrated Resource Allocation for Latency-Sensitive B5G

B5G networks are expected to support a diverse range of applications including ultra-bandwidth demanding service such as XR applications. These applications necessitate guaranteed low latency and high bandwidth, posing significant challenges for traditional resource allocation schemes. Existing approaches often struggle to effectively manage the complex interplay between cloud and network resources, potentially leading to performance bottlenecks and service disruptions.

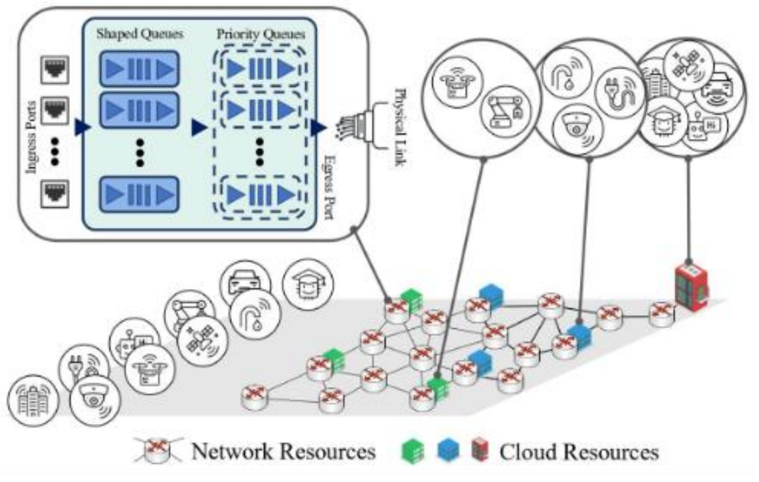


Figure 101: System model [68].

To cope with this issue, we first model the main components of the system studied: infrastructure, services, and requests as shown in *Figure 101*. Based on this, we then model the joint problem of VNF placement and assignment, traffic prioritization, and path selection by formulating a mathematical model that jointly optimizes the allocation of 1) computational resources: processing power and memory within the cloud infrastructure. 2) network resources: bandwidth and transmission time across the network infrastructure. We propose a near-optimal cloud-network integrated resource allocation scheme that jointly considers both cloud and network resources for B5G service provisioning to efficiently solve the complex optimization problem [68]. This algorithm iteratively searches for resource allocation solutions to 1) minimize the overall service execution time, 2) satisfy the latency requirements of B5G applications, 3) ensure efficient utilization of both cloud and network resources.

The Communication and Computing Resource Allocation (CCRA) problem specified is NP-hard (the multidimensional knapsack problem can be reduced to it) and finding its optimal solution in polynomial time is mathematically intractable. One potential strategy for addressing such a problem is to restrict its solution space using the branch and bound (B&B) algorithm, which relaxes and solves the problem to obtain lower bounds, and then improves the bounds using mathematical techniques to reach acceptable solutions. In this algorithm, the solution space is discovered by maintaining an unexplored candidate list $N = \{N_t | t \geq 1\}$, where each node N_t contains a problem, denoted by Φ_t, and t is the iteration number. This list only contains the root candidate N1 at the beginning with the primary problem to be solved. To reduce its enormous computational complexity, instead of directly applying the B&B algorithm to CCRA, we consider its integer linear transformation as the problem of N_1. Since the B&B method searches the problem's solution space for the optimal solution, its complexity can grow up to the size of the solution space in the worst case. Therefore, finding its optimal solution for large-scale instances using B&B is impractical in a timely manner, and the goal of this section is to devise an efficient approach based on the Water Filling (WF) concept to identify near-optimal solutions for this problem. The first step is to initialize the vectors of parameters and variables in the model. Following that, two empty sets, R' and Ω, are established. The former maintains the set of accepted requests, and the latter stores the feasible resource combinations for each request during its iteration. Now, the algorithm iterates through each request in R, starting with the one with the most stringent delay requirement, and keeps track of the feasible allocations of VNF, priority, as well as request and reply paths based on the constraints. The final steps of each iteration are to choose the allocation with the lowest cost and fix it for the request, as well as to update remaining resources and the set of pending and accepted requests. When there is no pending request, the algorithm terminates.

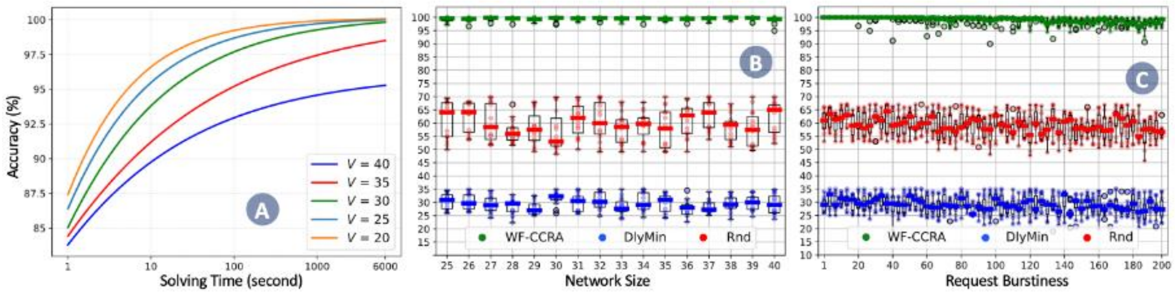


Figure 102: B&B-CCRA accuracy vs. solving time (A), the accuracy of WF-CCRA, DlyMin, and Rnd vs. network size (B) and request burstiness (C) [68].

The accuracy of the B&B-CCRA and WF-CCRA methods is numerically investigated through extensive simulations using various performance metrics: 1) average service execution time, 2) resource utilization, and 3) scalability. The proposed methods are evaluated based on the accuracy of the solutions they provide. Note that the accuracy of a solution for a scenario (η) is defined as $1-((\eta - \eta^*)/\eta^*)$, where η^* is the scenario's optimal solution, which is obtained by solving it with CPLEX 12.10. In *Figure 102(a)*, the accuracy of B&B-CCRA is plotted *vs*. the solving time for five scenarios with different network sizes. In this simulation, the number of requests is set to 200. As illustrated, the accuracy of B&B-CCRA starts at 80% after the first iteration, which is obtained by solving the LP transformation, dubbed LiCCRA, with CPLEX 12.10 in just a few milliseconds, and increases as the solving time passes, reaching 92% for all samples after 100 seconds. It proves that this method can be easily applied to provide baseline solutions for small and medium size use cases. However, the accuracy growth is slowed by increasing the network size, which is expected given the problem's NP-hardness and complexity. In the two remaining sub-figures, the accuracy of WF-CCRA is depicted against the number of requests and network size. In addition, these sub-figures illustrate the outcomes of two more approaches, called DlyMin and Rnd. In the DlyMin method, allocations are performed to minimize delay regardless of other constraints, while Rnd is used to allocate resources randomly to requests. Note that the number of requests in *Figure 102(b)* is 200, and the number of network nodes in *Figure 102(c)* is 20. For each number of nodes or requests, 50 random systems are formed, and the problem is solved for them using the techniques. It is evident that regardless of network size, WF-CCRA has an average accuracy of greater than 99%, implying that it can be used to allocate resources in a near-optimal manner even for large networks. For different numbers of requests, the average accuracy remains significantly high and greater than 96%. It does, however, slightly decrease as the number of requests increases, which is the cost of decomplexifying the problem by allocating the resources through separating requests. For the Rnd method, because it consumes the resources of all tiers uniformly, its accuracy is slightly above 50%. DlyMin is the least efficient method according to the results. The reason is that this method always utilizes the costly tier-one nodes to minimize E2E delay. In conclusion, it is shown that the WF-CCRA algorithm is capable of efficiently allocating resources for large numbers of requests compared to other approaches.

8.2 Joint Task and Computing Resource Allocation in Distributed Edge Computing Systems

Distributed task and resource allocation are critical to ensuring application requirements and maintaining resource efficiency in edge systems when cloud centers are unable to provide in-time management because of unpredicted communication latency. Meanwhile, compared with QoS like latency and throughput, quality of experience (QoE) is more critical for user-centric applications such as holography communication. Besides, conducting resource allocation to satisfy the QoE of applications rather than achieving extreme QoS is significant for improving resource efficiency, especially in resource-restricted edge systems. Therefore, to ensure resource efficiency and the QoE of applications, we investigate the distributed joint task and computing resource allocation problem for maximizing QoE. In addition to the quantitative correlation between QoS and QoE, we must address the limited state observation and resource management restrictions in actual systems. For example, each edge server can only obtain the state of its real-time associated users and can only decide its own resource allocation.

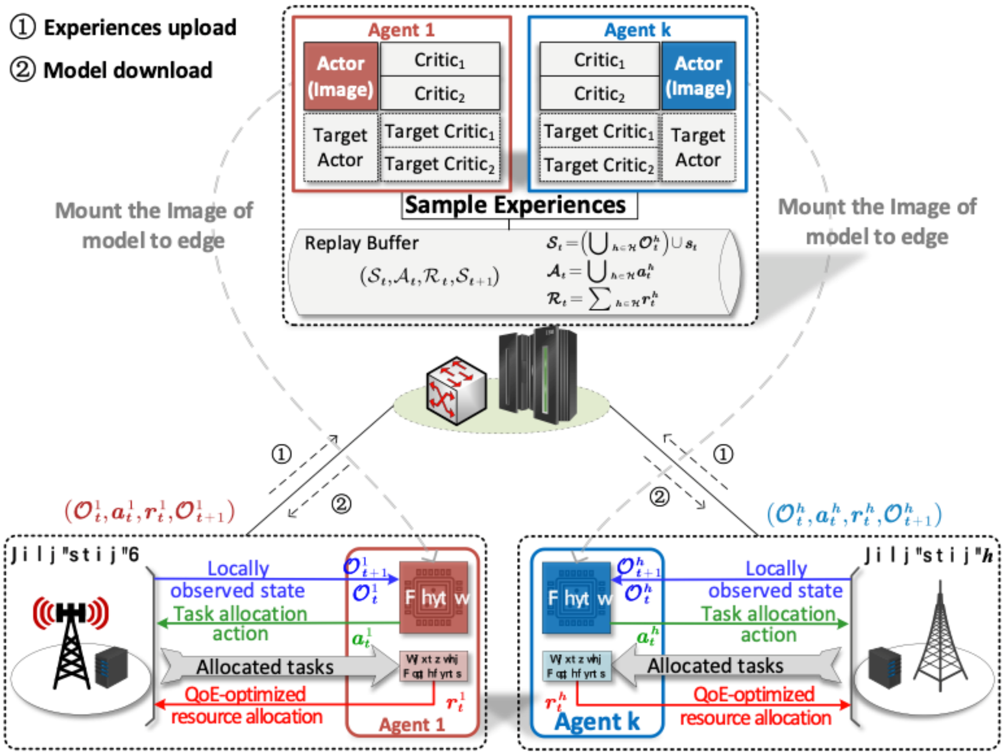

① Experiences upload
② Model download
Agent 1
Actor (Image)
Critic1
Critic2
Target Actor
Target Critic1
Target Critic2
Agent k
Sample Experiences
Replay Buffer
$(\mathcal{S}_t,\mathcal{A}_t,\mathcal{R}_t,\mathcal{S}_{t+1})$
$\mathcal{S}_t=(\bigcup_{h\in\mathcal{H}}\mathcal{O}_t^h)\cup s_t$
$\mathcal{A}_t=\bigcup_{h\in\mathcal{H}}a_t^h$
$\mathcal{R}_t=\sum_{h\in\mathcal{H}}r_t^h$
Mount the Image of model to edge
$(\mathcal{O}_t^1,a_t^1,r_t^1,\mathcal{O}_{t+1}^1)$
$(\mathcal{O}_t^h,a_t^h,r_t^h,\mathcal{O}_{t+1}^h)$
Locally observed state
Task allocation action
Allocated tasks
QoE-optimized resource allocation
Agent 1
Agent k

Figure 103: The framework of distributed task and resource allocation based on MADRL [69].

We proposed an approach based on multi-agent deep reinforcement learning (MADRL) [69]. *Figure 103* presents the framework of the proposed approach, which mainly consists of centralized training and distributed execution with the assistance of a remote cloud. Different from existing works that set each terminal device as an executor, we set the edge server as the management unit to ensure that the necessary resource state can be acquired, and resource allocation action will be accepted. Meanwhile, the number of policies that need to be trained is greatly reduced as edge servers are much fewer than users, which can greatly reduce the policy training cost and parameter synchronization cost. Besides, the complexities of policy training are greatly increased in multi-agent systems when the number of agents increases, and it is still challenging to obtain massive agents in one system. Therefore, our proposed approach can tackle the problem of massive users by setting edge severs as agents.

To satisfy the state observation constraint and resource capacity constraints, we decompose the problem into subproblems of task allocation and computing resource allocation. Then, we develop a two-step approach, including a MADRL-driven task allocation step and a computing resource allocation step based on sigmoidal programming. To deal with the issue of huge and discrete action spaces in multi-agent task allocation problems, we designed the approach based on the MADDPG method. Moreover, to further enhance the exploration during policy training, we integrate the entropy of massive user task allocation into policy updates. We prove that the resource allocation for maximizing QoE is a problem of maximizing a sum of sigmoids, making us able to optimize resource allocation using the sigmoidal programming method. The main workflow of our proposed approach includes the several steps. First, each edge server observes a local state (O_t^h) of its associated users, including the real-time requests, QoE and QoS correlations, locations, and communication conditions. Then, each edge server decides task allocations (a_t^h) for its associated users with its local state observation based on the installed policy (i.e., actor). After implementing task allocation actions, each edge server can identify the users whose tasks are allocated to be processed on it and acquire the corresponding state. Then, the resource allocation model based on sigmoidal programming optimizes the computing resource allocation among these users. After completing these procedures, the QoE of every user can be obtained by edge servers to calculate corresponding rewards (r_t). Then, each edge server collects the experience of the above procedures and uploads it to the cloud server. Once collecting enough data, an off-policy training process is triggered, and the parameters of actors are synchronized to the corresponding edge server after each update on the actor. Finally, the distributed policies are gradually optimized and can work cooperatively to maximize applications' QoE.

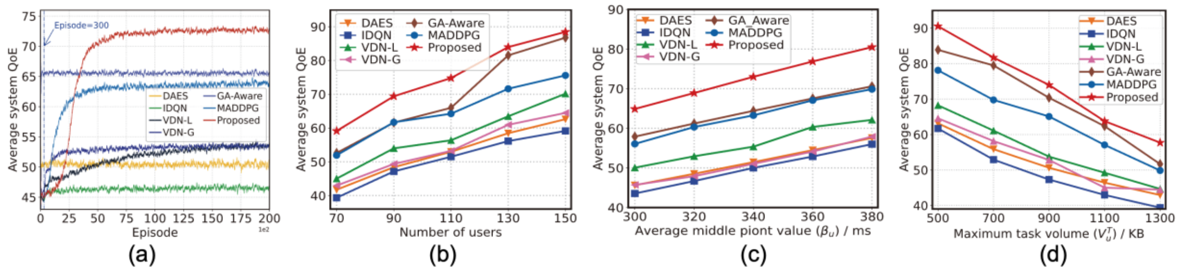


Figure 104: Average system QoE over training process, and under different numbers of users, average middle point values, and task volumes [69].

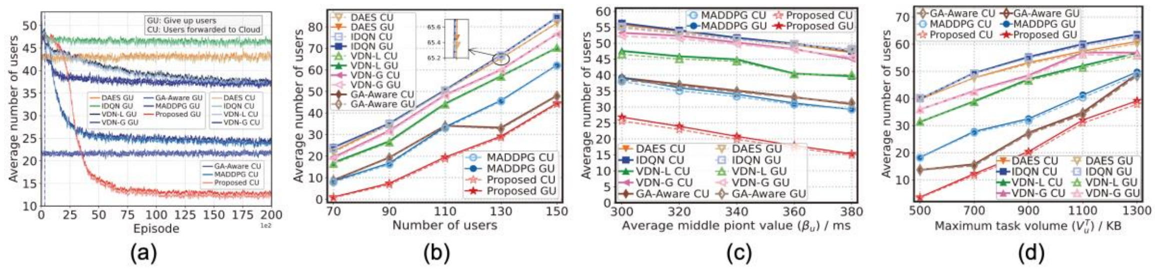
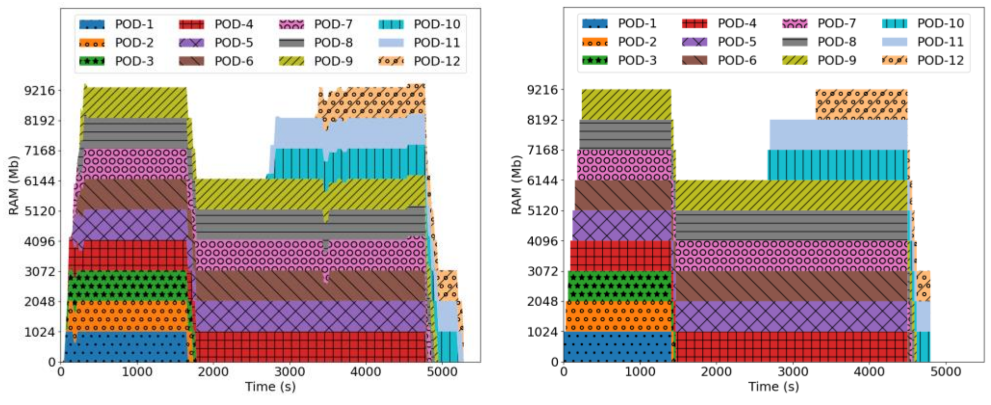


Figure 105: Average number of GU and CU over training process, and under different numbers of users, average middle point values, and task volumes [69].

We evaluated the proposed MADRL-based distributed joint task and resource allocation approach through extensive simulations [69]. The results demonstrate that our proposed approach can effectively establish distributed collaboration among edge servers by sharing resources to release computing load after policies are well trained and outperforms other benchmarks in terms of 1) Average system QoE: the proposed approach achieves a significantly higher sum of applications' QoE than others, especially when the resources of the edge system are sufficient to support a large proportion of all applications. 2) Average number of users who give up due to low QoE (GU) and users whose tasks are forwarded to the cloud (CU): the proposed approach can minimize the number of users who achieve unacceptable QoE and the number of users whose tasks are forwarded to the cloud because of inappropriate task allocation actions that break the resource constraints of edge servers, ensuring the resource utilization of the edge system. As presented in *Figure 105*, the users who give up because of unacceptable QoE are mainly attributed to their tasks being forwarded to the cloud, resulting in considerable latency. Our proposed approach can minimize the number of such kinds of users and support most applications in the edge system. This is because our approach can allocate tasks to edge servers more appropriately, which reduces the probability of breaking the resource constraints of edge servers. *Figure 104* and *Figure 105* reveal that our proposed approach converges to obviously better performance than other DRL approaches, including MADDPG, VDN, and IDQN. This is because we employ the policy-based method to address the issue of huge and discrete action spaces, and we integrate the action entropy of distributed task allocation to enhance exploration.

8.3 Simulations Tools and Experiments on Cloud Resource Management

To build an efficient cloud resource orchestration system, it is important to populate it with efficient AI-based algorithms and mechanisms that autonomously take decisions on the resource allocation and service placement. These AI techniques need to be intuitively evaluated, above all in simulated environments close to real life systems. In this regard, a simulation platform that accurately mimics K8s microservices clouds is proposed [70]. This platform is aimed to the RL agents. It helps them learn quickly and converge to a policy that can be used in real environments. The simulated environment is meant to start the agent learning; it is not meant to replace the real environments. Indeed, after finding a decent policy, the agent needs to continue learning once it is deployed in a real environment. Using a simulator instead of using a real deployment would greatly reduce the time needed by the agent to learn a good policy. Furthermore, as described in Section 2.2, the envisioned AIRO framework aims to alleviate the scheduling and placement problem in very large system where the number of possible configurations is huge. In order to evaluate and quicken development of the AIRO framework, the simulation platform is needed. Therefore, a performance evaluation is carried out to show how close the cloud-native simulator is to the real deployment.



a) Real testbed b) Simulated Testbed

Figure 106: Memory consumption per POD [70].

Figure 106 shows memory consumption for each POD. From *Figure 106(a)* and *Figure 106(b)*, it is clear that the real and simulated testbeds are almost identical. The only notable difference between the two is that the real testbed can show random behaviour such as in the case of POD6 and POD9 between 3500s and 4000s. Likewise, CPU utilization of PODs in both testbeds is quite similar. The real testbed shows also some noisy behaviour compared to the simulated testbed.

Additionally, an experiment was carried out with the main objective of investigating and evaluating a resource monitoring demonstration in a cloud native deployment. This experience proves to be important, as continuous monitoring can be useful to minimize incident response time and ensure that applications and infrastructure behave as expected. Namely, tracking cluster resources, such as memory, CPU, storage, and bandwidth, facilitates the process of managing cloud-native environments. Considering the multi-domain context, Rancher[32] was used, taking into account it has support for multiple multi-cloud providers. This feature allows the facilitation and intermediation of the orchestration of different domains, which can even be from different providers. Rancher allows not only the creation but also the orchestration of multiple Kubernetes Clusters, both k3s and k8s, by installing a cluster agent on all cluster nodes. In this sense, Rancher was used to set up a two-node Kubernetes cluster that was used to deploy and monitor distinct microservice-based applications. Additionally, a Prometheus[33] and a Grafana[34] installation was performed, to provide a simple and efficient way to visualize several natively supported cluster and pod-specific metrics. By leveraging this approach, and considering the Prometheus architecture, additional XR-specific instrumentation was achieved by having additional libraries and Prometheus Exporters to expose virtually all kinds of XR related metrics.

First, and to evaluate the monitoring resources in a more comprehensive way, three different (topology-wise and purpose-wise) applications were deployed [71]. These applications were chosen as reference scenarios for next-generation XR applications, being composed of multiple microservices and with different topologies. In this sense, it is possible to assess how different applications can be effectively monitored in a cloud-native environment and how their orchestration can be performed, to support a new wave of predicting, scheduling, and intelligent orchestration mechanisms. Therefore, these applications have different primary objectives, such as:

- 2-tier application with two main goals, achieved with iper3[35] and stress-ng[36]. The first tool allows generating realistic network traffic, while the second one allows generating excessive use of resources.
- 3-tier application with the purpose of providing a simple-yet-realistic standard architecture as a starting point for demonstration purposes.
- 12-tier application represents the implementation of a web-based e-commerce application, to showcase a complex and realistic application.

Considering XR services and cloud-native environments, it is important that the availability factor is taken into account, because downtime values (e.g., due to service migrations) can have a negative impact on users' experience. Therefore, it is extremely important that it is possible to analyse and evaluate the deployment time (the time that a service takes from creation to proper functioning), to properly assess and regulate performance. However, the deployment time that is usually debated does not consider the availability of the service, that is, it only measures the time until the pod is running, and not until the service is fully capable of accepting any type of requests (e.g., some databases need time to migrate, web servers need time to initialize). Thus, for the validation of this metric, a component was deployed that allows the calculation of the deployment time based on the event log performed by Kubernetes, as out-of-the-box metrics reported by Kubernetes (i.e., kube-state-metrics) neither provide such a mechanism nor account for time spent pulling a container image.

[32] https://rancher.com/

[33] https://prometheus.io/

[34] https://grafana.com/

[35] https://iperf.fr/

[36] https://wiki.ubuntu.com/Kernel/Reference/stress-ng

Figure 107 shows this time, in seconds, per pod and per application. These values are an average of the results obtained in five tests. As mentioned, these are the values per pod, however, the deployment time of the application as a whole is obtained through the maximum deployment time of its pods (the pod that took the longest to deploy). It should be also noted that in these tests the images were not pulled, a process that would lead to an increase in the recorded time.

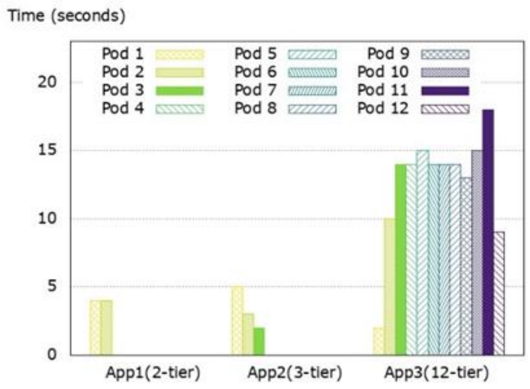


Figure 107: Average pod deployment time of each application [71].

The recorded values and their differences can be explained by the level of complexity of each application (the more tiers, the longer it takes to deploy). The first application only has two pods and does not present relevant differences between the deployment time of the two. The second one has three deployments, which have dependencies on each other, which explains the difference between the times of each pod. The last one has twelve pods, also with dependencies between them, which once again explains the difference in its deployment time. These dependencies between services in cloud-native applications are quite common and have an impact on numerous operations (e.g., service migration, scaling), and in this way, microservice-based XR applications are not expected to be different. Indeed, they are expected to have complex topologies and numerous dynamic constraints. Thus, their management in (near) real-time is a fundamental aspect of the envisioned orchestration and should be an aspect to consider.

In order to understand the discrepancy between deployment times between services and applications, it is important to dissect the various states considered in the deployment process. This process includes scheduling the pods, pulling the container(s) image(s), and finally creating and starting it. In *Figure 108*, it is possible to visualize the various states of this process and their time on each pod.

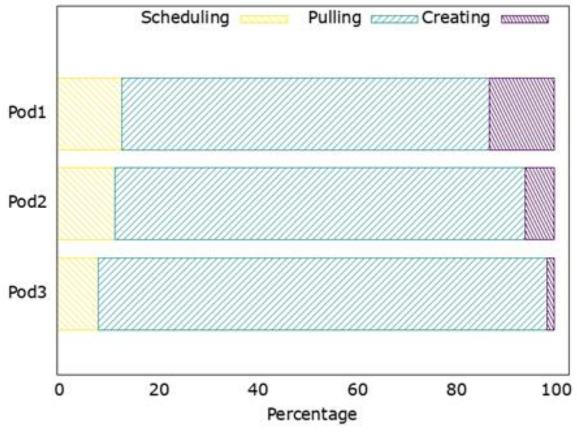


Figure 108: Stages included in deployment time [71].

Although the pulling state time depends on factors such as image size, this is the state that has the longest time. However, to counteract this factor, it is possible to configure the pods to only pull the image when it is not present locally. In this case, only the first time its deployment was carried out would the image be pulled, in subsequent deployments, this process would no longer be accomplished, since the image would already be present locally, which consequently causes a shorter deployment time.

Nevertheless, maintaining local images across nodes becomes ever-more difficult when dealing with multi-node, highly complex cloud-native environments. Prioritizing deployment time at the expense of complexity is a matter that demands its proper evaluation, and understanding its impact is crucial to properly adapt cloud-native environments to specific use-cases. To get around this problem, one of the possibilities is the use of smart caching techniques. The use of these techniques during orchestration enables the prefetch of the required images of XR services and places them in the nodes or in close vicinity of the nodes.

In addition to measuring and analysing the deployment time, it is also important to evaluate mechanisms that allow the visualization of resources (e.g., CPU and memory usage) to predict the behaviour of applications and infrastructure, to enable active decision making. Thus, the effectiveness and efficiency of the application can be maintained. In this sense, Grafana provides out-of-the-box dashboards to visualize such metric. Thereby, *Figure 109* and *Figure 110* show the visualization of the graphs obtained from CPU usage and the memory usage of the deployed 12-tier application.

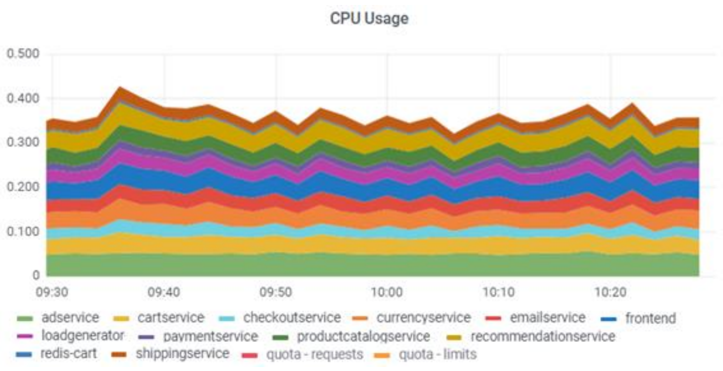


Figure 109: CPU Usage graph from 12-tier application [71].

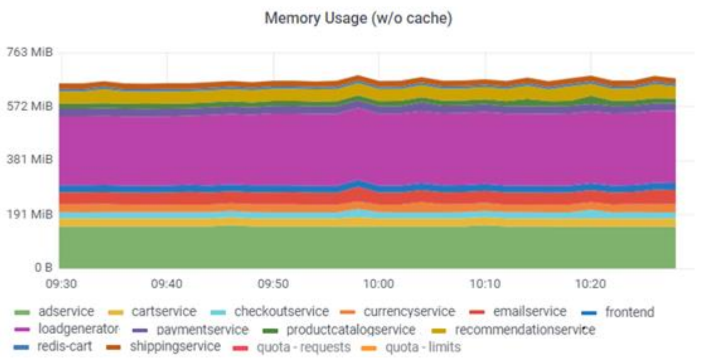


Figure 110: Memory Usage graph from 12-tier application [71].

The observed graphs show different values for each container, for memory and CPU usage, which can be explained by the differentiation of each pod function, since there will be pods that need more resources to perform their functions.

Likewise, a multi-user XR application might behave differently according to multiple factors, such as the environment, number of instances, users or even different settings of each user. Monitoring the resource usage at the pod-level and/or application-level is useful to detect early deviations that might indicate unhealthy situations or be used to predict individual service behaviours. Effectively, through the deployment of different cloud-native XR applications, more interesting observations can be made regarding the number of tiers, caching, and availability. These observations can be translated into recommendations for *i)* how cloud-native XR applications should be designed (in addition to the guidelines stated in subsection 6.8) and also on *ii)* how corresponding clusters should be formed, configured and orchestrated to ensure a certain level of QoE for the target XR applications. Like the above-mentioned applications we experimented on, XR applications will benefit from deployments that can take advantage of locally present images. From an orchestration standpoint, clusters should be also designed in order to take advantage of smart caching mechanisms that can *i)* make those images available at different application lifecycle and *ii)* also minimize the burden of having them always persisted at every single node of a multi-cluster and multi-node setup. Moreover, clusters should be considered and designed as highly dynamic environments, capable of seamlessly reallocating XR applications components taking into consideration runtime factors such as cluster or individual component resource consumptions and application-specific characteristics (e.g., current number of users). Ideally, such overall orchestration should not impose changes on XR components themselves, which are likely not aware of such needs. Nevertheless, by following microservice best practices (e.g., single responsibility principle), XR applications can be modelled in a way that facilitate their orchestration (e.g., using stateless services when appropriated).

9 Security & Privacy

CHARITY strongly focuses on the orchestration and scheduling of XR services, so it is crucial to analyse the security and privacy challenges of delivering these services. Security and Privacy are different words with different means. They complement each other. On one side, security is the set of measurements to protect against the risks and loss of information. In another way, privacy is the right to maintain the information protected.

The following sections discuss the challenges inherent to using XR technology, the security and privacy requirements of XR applications, and how CHARITY intends to explore the Zero-Trust security model and the specific Security as a Service Mechanisms to enforce the security of XR applications within the CHARITY framework.

Moreover, the following sections also present some Cloud-Native security challenges covering the inevitable need for security at the micro-services level and security in orchestration and scheduling processes. It also discusses the DevSecOps concepts as increasing relevant security aspects of the next generation of XR applications. Finally, this section also presents the proposed approaches and mechanisms considered for integration as part of the CHARITY framework.

9.1 Security of XR Applications, Zero-Trust and Security as a Service

The progress of XR has significantly increased in the past years due to the advances in available XR hardware (e.g., HMDs), the introduction of more capable sensors, and the advances in computing graphics technologies. XR-based applications, including AR and VR, combine the real-word with virtual reality elements to support more immersive experiences in healthcare, education, and industry. Nevertheless, it is of the utmost importance to understand the challenges of XR technology from a security and privacy perspective. This technology presents new attack vectors that must be considered, evaluated and adequately mitigated. Hence, in what follows, we first present the security challenges inherent to XR applications and then discuss how the recent Zero Trust security model allows addressing some of these challenges.

9.1.1 Security of XR applications

In the same way as other applications, XR applications are susceptible to different kinds of classical threats such as malware, DDoS or Man-in-the-Middle (MITM) attacks. Hence, the design of a comprehensive security strategy should not put those threats aside. Nevertheless, it is relevant to analyse XR-specific needs and challenges. XR applications typically collect and process large amounts of sensitive data (e.g., user location, biometrics, medical data, unwanted information about private spaces, and recording private conversations). For instance, a VR-based training class can reveal several types of sensitive data, like the trainer's surrounding environment. Likewise, unintentionally collected images from private spaces can reveal sensitive information about players in an AR game. This information, for instance, can be used to understand when a potential target is at home, posing a danger if these images reach malicious people. Hence, the design of an XR system should consider such an amount of potentially sensitive data. Unlike other traditional applications, XR applications involve such an increased amount of data, which from a security perspective, poses a challenge to have the proper way to process them in, ideally, real-time. Latency, a significant constraint for XR applications, is also a challenge from a security standpoint. Security approaches tailored for such latency-sensitive environments should not themselves require additional verification steps such as active network filtering mechanisms) which would otherwise create a potentially more secure but lower perceived QoE and ruin the expected immersive experience.

According to [72], XR applications need the following list of security and privacy requirements:

- **Integrity:** Stored and in transit data must not be tampered with or modified;

- **Non-repudiation:** The responsible entity involved in any data operation must be identified appropriately, thus avoiding the negation of the action;
- **Availability:** An adversary should not be able to make the system unavailable;
- **Authorization, Authentication and Access Control:** All actions must be verified, and only authorized users or services should perform operations.
- **Confidentiality:** Unauthorized users must not have access to confidential resources;
- **Unlinkability:** An adversary must not be able to link and identify users through existing data.

Different approaches exist for each security and privacy requirement. In [72], the authors proposed a taxonomy with five main categories: *Data Protection, Input Protection, Output Protection, User Interfaces and Device Protection* (*Figure 111*), which considers what security and privacy approaches should protect. In CHARITY, more than covering all, the main goal is to understand how some of them can be leveraged and integrated within the overall notion of a more autonomous and intelligent orchestration platform.

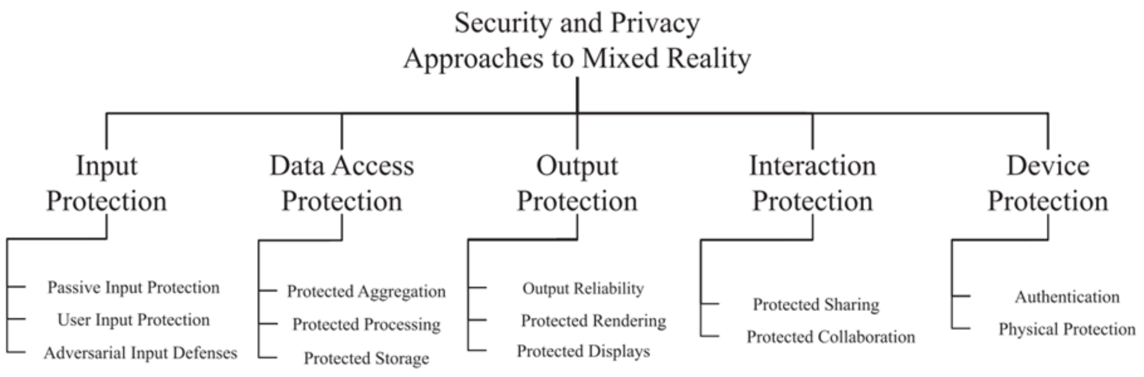


Figure 111: Mapping security and Privacy approaches to XR applications [72] .

Input Protection: The category considers that XR applications could contain sensitive information and focuses on ensuring the security and privacy of data entered into the XR application. Input protection techniques relate to input sanitization (e.g., context-based and video-based sanitization). They are used to evaluate whether there is sensitive information in images and videos entering the application. The input protection mechanisms also seek to distinguish legitimate inputs from adversarial inputs.

Data Access Protection: Comprises all the techniques used to prevent unauthorised users from accessing data in storage, in transit and during the computation stage. Data Access protection approach can range from classical encryption-based techniques, which allow preserving the confidentiality of data in rest or transit, to Homomorphic encryption schemes, which allow executing some operations without needing to decrypt the data. Other techniques such as K-Anonymity and differential privacy also fall into this category to ensure the unnecessary leak of personally identifiable information.

Output Protection: This covers the security and privacy aspects of protecting data outputs. For instance, output control policies could provide a guideline for devices to know how to deal with outputs from third-party applications. Likewise, Least-Privilege approaches to prevent the rendering of unnecessary elements.

Interacting Protection: Encompasses the approaches used to protect collaborative interactions in a multi-user environment. They establish boundaries and, for instance, provide confidentiality for sensitive data in a shared context or non-repudiation to ensure the correct identification of interactions.

Device Protection: Security mechanisms to ensure only known and authorized users access the XR devices. Hence, it primarily concerns the identifiability of users, and in a second stage, with the correct authorization and authentication. Device protection approaches include, for instance, secure authentication mechanisms based on gestures, physiological characteristics or biometric information.

9.1.2 Zero-Touch Network and Service Management (ZSM)

In cloud-native environments, it is crucial to automate detecting and mitigating network anomalies intelligently. Moreover, since they are typically multi-tenancy environments, a failure in a given service/level can harm the rest of the environment. Proposed by ETSI, the Zero-touch network and Service Management (ZSM) specification [73] defines an End-to-End (E2E) management reference architecture that aims to provide a more flexible and automated approach for E2E service orchestration in multi-domain deployments. ZSM specification addresses how a set of management services (and their respective capabilities) can enable a more consistent and standardized method to manage the entire life-cycle of services spanning over those multi-domain scenarios. Indeed, ZSM defines the concept of a Management Domain (MD) by encompassing the existing management functions of each domain and an Integration Fabric to expose such capabilities and facilitate the communication between service consumers and producers.

Moreover, ZSM reference architecture includes an E2E Service Management Domain (and a Cross-Domain Integration Fabric) to support the overall E2E orchestration capabilities. Furthermore, ZSM specification builds upon the principle of closed-loop management automation and feedback-driven processes (e.g., OODA loop model) to realize fully automated (and intelligent) management functionalities. Here, we discuss how security, a relevant concern within service orchestration, benefits from the same principle and reasoning.

In order to enforce and support particular security attributes, the service orchestration regarding to the security criteria and requirements must be studied. Specifically, putting a greater emphasis on VNF placement would prevent user data from traveling towards the core of the network, which is deemed as a safer way to store and process the data. This would also need additional infrastructure, and it could also have an impact on the overall experience or performance of the XR applications. For instance, this may be utilized to fulfil the particular statutory or geopolitical needs, or at the absolute least, it could be utilized to lessen the quantity and distance of user-specific data that are transmitted via the network in order to maintain secrecy.

9.1.3 Zero Trust Security Model

The previous section provided an overview of the security-related challenges and requirements of XR applications. This section discusses how the concept of Zero Trust supports the implementation of such security approaches. In the past, the traditional perimeter security model was based on the *"trust but verify"* approach. As long as components and equipment belong to a specific network or segmented group, they are considered trustworthy. In such a model, network segmentation, used to establish a trust zone, was assumed to be sufficient. The focus was on whether or not to authorize access to network segments through perimeter security mechanisms such as firewalls. The perimeter model can protect against threats from outside but does not prevent unauthorized lateral movement. The implicit perimeter trust model does also not fit the increasingly complex and dynamic cloud computing environments [74] .

In such environments, no default trust should occur. Instead, adopting the principle of least privilege (POLP) [75] and the Zero-trust model for granting only the minimum required privileges, thus, limiting the visibility and accessibility of assets. Zero-trust model intends to prevent unauthorized lateral movement as not implicitly or by default trusted zones exist. Moreover, Zero Trust also intends to provide a more granular and dynamic segmentation of the different resources (i.e., even when network traffic belonging to the same network should go through a validation process) [76] .

According to NIST, the Zero Trust architecture must follow the basic zero trust principles [77] : data, assets and services are considered resources; should be secured communications regardless of whether networks are considered enterprise-owned or non-enterprise-owned; per session resource access; access depends on multiple aspects including the observable state of the user, the service requesting asset; no resource is inherently trusted, there is always an assessment for each access request, which implies monitoring the integrity of the resources [77].

Figure 112 further illustrates the difference between the classical perimeter and Zero Trust models. The perimeter model focuses on communications from outside the Local Area Network (LAN), assuming the network is trusted. In the Zero-Trust model, the network is never trusted, each communication should undergo an authentication process. Moreover, in a Service-Based Architecture (SBA), security policies should be enforced in all network communications between services (and containers).

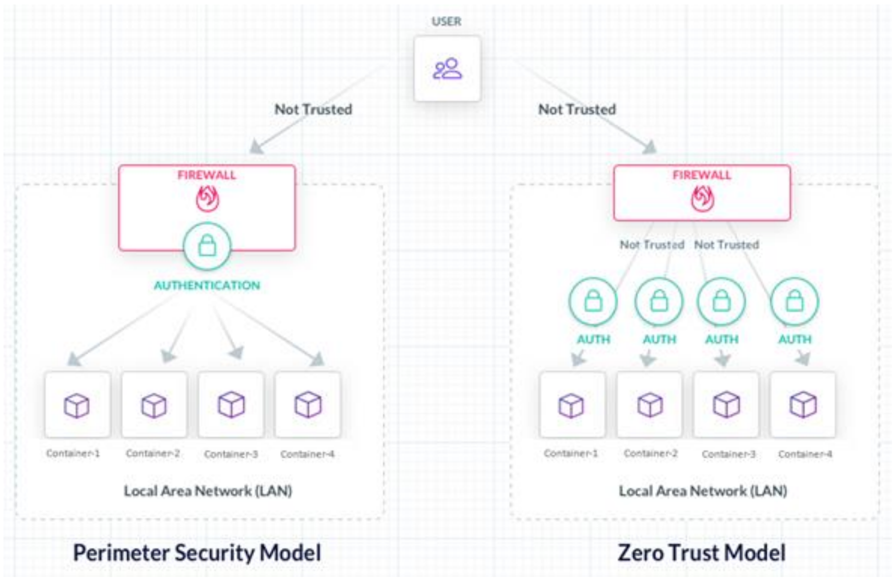


Figure 112: Comparison between Perimeter Security Model and Zero Trust Model, adapted from[37]*.*

Even though, Zero trust does not aim to replace the security perimeter model but complements it with a more granular approach to enforcing security policies. Whilst zero trust often refers to authentication and authorization policy enforcement, in CHARITY, we investigated the idea of having the means for observing the various XR service components. Including the means for continuous monitoring of all the network traffic (i.e., both north-south and east-west traffic) within a Cloud-Native environment. Such a continuous evaluation of network traffic is relevant to timely detect cyber-attacks (e.g., due to compromised components or API abuses) and thus complements additional policy enforcement strategies, fulfilling the zero trust overall principle of “always verify”.

9.1.4 Service Mesh

Service mesh is an increasingly widely used solution to address the challenges of complex and exponential relationships between components [78] . The service mesh concept has gained much strength in achieving network and resource observability, a key feature within any cloud-native environment. For instance, Eunji Kim *et al.* [6] used Istio and Envoy for data collection and complete visibility into infrastructure resources, network traffic and application behaviour in the environment t.

Service Mesh is an infrastructure layer for handling service-to-service communication without imposing changes on services [78]. Service Meshes provide a more Cloud-Native and comprehensive strategy to control network traffic, apply distinct policies and cope with the complexity and dynamism of service communications. Service mesh addresses such challenges by shifting standard orchestration-related functions from the application logic to the infrastructure layer. The immediate benefit is simplifying applications that would otherwise need to include them. Moreover, by abstracting

[37] https://goteleport.com/gravitational/images/diagrams/teleport/zero-trust-vs-firewalls-vpns.png

standard functions to the infrastructure layer, we might envision a more standard approach to implementing them.

In summary, service meshes provide the following key features:

- **Service Registry & discovery:** Mechanisms used to facilitate the discovery and registry of new components and services.
- **Load balancing:** The ability to balance network traffic depending on latency and infrastructure health status.
- **Fault tolerance:** The ability to redirect requests to an alternate instance when the original is not available or degraded.
- **Traffic monitoring:** The ability to monitor all the network traffic and key metrics between the mesh of microservices.
- **Encryption, Authentication and Access Control:** Dynamically encryption of network communications on the fly. Authorize and authenticate network communication between services.

Although conceptually not restricted, the rise of service meshes emerged with Cloud-native applications [79] . As shown in *Figure 113*, a service mesh is constituted of two planes: data and control planes. The data plane comprises a set of proxies known as sidecars, co-located into each microservice. The sidecars are proxies that intermediate the communication with other proxies. The combination of several proxies forms a mesh that intercepts the communication between microservices, meditating and controlling all network communication and collecting telemetry. Together, they have complete visibility over all microservice communications and can perform health checking, filtering, routing, load balancing, service discovery, authentication, authorization, and collecting telemetry. The control plane is responsible for the overall orchestration of the sidecar's behaviour. In comparison, the proxies provide the capabilities to configure the components to collect telemetry, enabling observability and applying routing traffic policies. The Service Mesh concept is used as an architectural approach to enforce security policies on top of microservices network traffic.

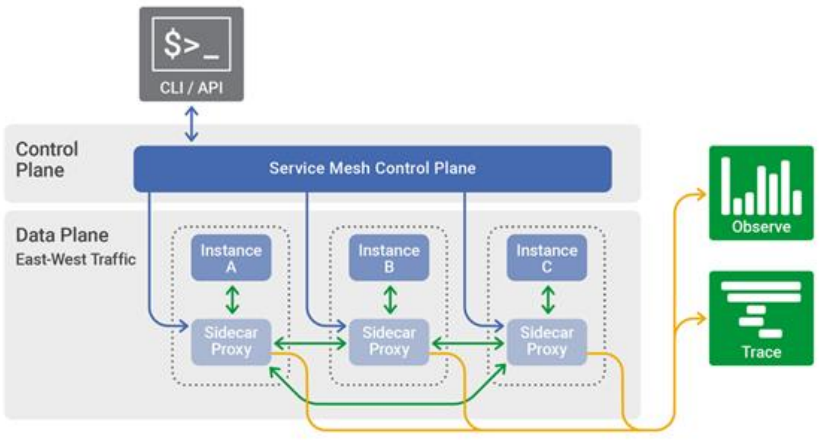


Figure 113: Data and Control planes of a Service Mesh[38].

Although this approach has many benefits, they require further investigation. Chandramouli *et al.*[80] address this issue in a NIST publication and presents additional guidance on security solutions for cloud-native environments, specifically, how proxy-based Service Mesh components can collectively form a security infrastructure to support micro-services.

As discussed in several works, anomaly detection in Cloud-Native environments has new challenges and relevance. Harlicaj [81] discussed this problem for detecting web-based attacks on micro-services

[38] https://www.nginx.com/blog/what-is-a-service-mesh/

using Kubernetes and Istio. Similarly, in [82], the authors proposed an anomaly detection mechanism in API traffic to improve its security in microservices environments. Characteristics such as bandwidth and the number of consecutive requests were analysed, concluding that such a mechanism is more accurate than a rule-based anomaly detection mechanism. On the other hand, both works do not cover automated mitigation of incidents (e.g., based on policies).

Two essential widely-used tools that enable the service mesh concept are Istio and Envoy proxies. Istio[39] is a realization of an open-source Service Mesh platform enabling control of how microservices share data between them. It comprises a set of layered distributed applications providing traffic management, security and observability at the service mesh level. Istio exposes APIs to enforce access control at mesh, namespace and service levels. As a result of applying policies over a Kubernetes network, it becomes possible to configure security for service communications at the network and application layers. Istio APIs support integration with other components, such as telemetry or policy systems. Whereas, Envoy proxies bring the sidecar functionalities [40] . Envoy provides built-in functionality to cope with challenges, such as retries, delays, circuit breakers and fault injection, dynamic service discovery and load balancing, traffic management and routing, security policies and rate limiting. Istio offers security advantages through strong identity, transparent TLS encryption, robust policy, and authentication, authorization, and auditing (AAA) tools to protect services and data exchanged between them. However, another feature to highlight is its support for heterogeneous environments, including VMs, Kubernetes and multi-domains setting. In the ambit of this project, Istio was used to deploy a Service Mesh in a Kubernetes environment to conduct the experimental work.

9.1.5 Policy-Based Control

Not only is it a challenge to efficiently observe and understand, for instance, all the network traffic of an environment, but it also is crucial to have the means to apply mitigation measures and security policies effectively. One example of policy enforcement in a cloud-native environment is the architecture proposed by Loıc Miller *et al.* [83] that uses Istio combined with the Open Policy Agent (OPA) to execute policies. Although this architecture presents key ideas focused on the authorization of communications through the defined policies, it does not present ways of detecting anomalies.

Open Policy Agent (OPA)[41] is an open source, general-purpose policy engine that unifies the implementation of policy enforcement procedures across IT environments, such as the ones involving Cloud-Native applications. OPA enables decoupling policy decisions from policy enforcement. Hence, distinct analytics mechanisms (e.g., AI/ML-based orchestration functions) can support the decision process. In contrast, OPA can uniformise how those decisions are specified and enforced. For instance, whereas Istio can be used to support network traffic management, Istio policies are limited to networks. On the other hand, OPA allows a more comprehensive strategy to implement distinct policies and have more control over deployments and containers. OPA was used with Istio to enforce network policies on a Kubernetes environment as part of the experimental work.

9.1.6 Security as a Service (SECaaS)

In line with the underlying idea of CHARITY architecture as a more intelligent and autonomous framework to address the orchestration challenges of the next generation of XR applications, this section discusses the benefits of the Security as a Service (SecaaS) model as an approach to support the security and privacy requirements of XR applications.

From Identity and Access Management (IAM), Information Security, Network Security, Intrusion Supervision or Encryption, the SECaaS model brings numerous benefits [84][85] . For XR applications,

[39] https://istio.io/

[40] https://www.envoyproxy.io/

[41] https://www.openpolicyagent.org

the benefits are manifold. In XR, the capability to seamless process all data is paramount to achieving the recommended security and privacy requirements. From computing vision algorithms capable of processing video streams from HMDs and automatically flagging potential bystander-sensitive information up to the network security methods to detect anomalies and cyber-attacks, SECaaS offers an opportunity to incorporate such mechanisms without design time intrusions [86] .

The SECaaS model is also a cost-efficient solution that allows next-generation developers and application providers to shift the onus of security functions to infrastructure providers. This way, various security and privacy mechanisms could be considered to take advantage of the hybrid edge/cloud network, compute, and storage resources and the notion of closed loops to have more autonomous and intelligent security coverage.

9.1.7 Autonomic and Cognitive Security Management

The amalgamation of many of the technologies that were presented during the previous section of this work shall serve as the building blocks for the emergence of frameworks that are capable of establishing cybersecurity in cloud-native services. This section is dedicated to exploring such a framework that is based on paradigms, such as the enabling technologies that were previously explored. Modern applications are being designed into smaller, more manageable microservices due to a plethora of requirements, such as portability, scalability, or reliability, in the context of cloud-native environments. From a security standpoint, such an emerging paradigm raises new challenges in terms of managing the volume and complexity of cloud-native applications. As such, this demands intelligent and automated solutions to lower the burden assigned to humans in the context of managing the security of cloud-native applications. With the objective of establishing autonomous secure management of resources in various domains, a framework that adheres to the key design principles of ETSI ZSM was proposed, as depicted in *Figure 114*. This framework introduces AI-powered closed loops with various different scopes, from node level to end-to-end and inter-slice level. Thus, it allows the rapid and effective detection and mitigation of security threats close to the source, which prevents their proliferation in the network.

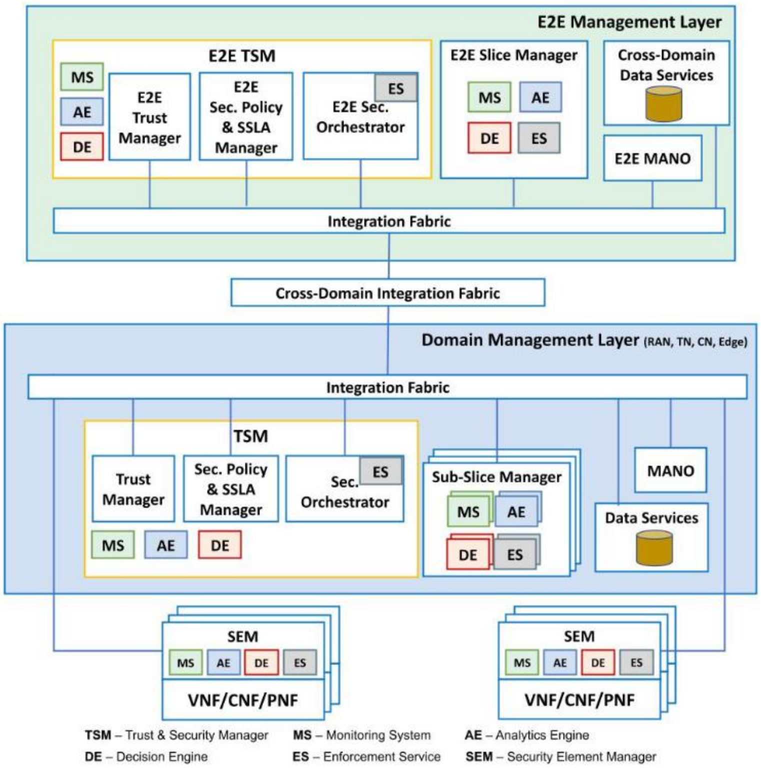


Figure 114: Architecture of an autonomic and cognitive security management framework, adapted from [87].

The framework enhances E2E security management capabilities across various domains in a hierarchical manner. It builds upon the domain vision presented in and introduces AI-driven closed loops with varying scopes, ranging from individual network nodes to end-to-end connections and inter-slice levels. This fine-grained approach enables efficient and rapid detection and mitigation of security threats at their source, preventing their spread throughout the network. It is important to note that this framework aligns with the aforementioned fundamental design principles of ETSI ZSM. It achieves this by supporting the segregation of security management concerns and adopting a service-based architecture. This architecture allows authorized consumers to access and utilize the provided security management services through an integration fabric. The integration fabric facilitates the registration, discovery, and invocation of security management services. It also facilitates communication between these services and other management services. The framework leverages historical data and knowledge generated by various security management services, which are stored and made available through data services within the same domain or across multiple domains. In the following sections, we outline the main functional components of this framework. In this architecture, the closed-loop automation manifests in the context of four functions, Monitoring System (MS), Analytics Engine (AE), Decision Engine (DE), and Enforcement Service (ES). MS takes responsibility for gathering, preprocessing, and presenting security-related data obtained from the managed entity. AE offers services that enable the identification or prediction of potential security anomalies and attacks, as well as determining the root causes behind observed security incidents, utilizing the collected data. DE determines the most effective mitigation policy required to address the detected or predicted security issue, ensuring the desired level of security is maintained. ES allows triggering/updating implementations of specific Virtual Security Functions (VSFs), such as vFirewall and vIDS through the management and orchestration platform (MANO). Each network function is associated with a Security Element Manager (SEM) that will be responsible for managing security within its scope. As mentioned, in closed loops, cognitive resources are incorporated for security analysis and decision-making. Furthermore, to increase the cognitive level of the environment, AI/ML techniques can be implemented in MS and ES.

In order to have greater security and mitigation at different levels, multiple closed loops can be coordinated at different levels. These loops are managed and orchestrated by the “Trust & Security Manager” (TSM),which comprises three functional modules, the “Security Orchestrator”, “Security Policy and SSLA Manager”, and “Trust Manager”. The “Security Orchestrator” is responsible for designing, instantiating, and managing the runtime lifecycle of circuits. Security Policy and SSLA Manager are responsible for violating SSLAs (Security Service Level Agreements) and security policies defined by external entities. Trust Manager is responsible for the continuous assessment of the reliability of the network services and associated circuits (this trust is calculated based on the trust attributes specified in the Trust Level Agreement).

Based on this framework, some open-source solutions that enable zero-touch security management in environments were analysed. A cloud-native architecture, based on stateless microservices implemented as containers, is a technology recognized for being suitable for the cost efficiencies, flexibility, and scalability required in the operation and management of environments. In this sense, the concept of Platform-as-a-Service (PaaS) emerges, a cloud-native architecture layer that allows developers to implement, run, and manage different applications without the complexity of configuring and maintaining the cloud. In this follow-up, Kubernetes appears as a de facto standard for the implementation and orchestration of applications in containers, which allows scalability, high availability, and fault tolerance features.

For the implementation of the Fabric Integration features, including facilitating interoperability and communication between management services (within and between domains), service mesh solutions such as Istio or Linkerd[42] were analyzed, as well as an event streaming platform, like Apache Kafka. On the one hand, the service mesh will allow the management of traffic between services, while enhancing security and observability. The event streaming platform will handle asynchronous

[42] https://linkerd.io/

communications between applications and services. In addition, the use of the event streaming platform is also important for security use cases, including monitoring, analyzing, and reacting to security threats in real time. In this sense, the use of Istio and Kafka as candidates for integration fabric implementation was considered.

In the context of management and orchestration platforms, the Open Network Automation Platform (ONAP)[43] and Open Source MANO (OSM)[44] are highlighted. ONAP provides a framework for real-time, policy-driven orchestration, management, and automation of network services and edge computing. This platform includes different ecosystems, such as POLICY, CLAMP, and DCAE, which together support closed-loop automation. POLICY provides policy creation and validation capability. CLAMP designs and manages closed-control loops. DCAE (Data Collection, Analytics, and Events) collects and analyses data. On the other hand, OSM allows modelling and automating the life cycle of network functions, network services, and network slices through MON and POL mod- ules. MON leverages monitoring tools to collect VNFs and infrastructure metrics. POL is a policy management module. Another tool to highlight is Ansible https://www.ansible.com (accessed on 15 August 2023), which allows production-level automation in a cloud-native environment, and as such, allows increasing the automation capabilities of NFVM and NFVO in the management and orchestration of network functions and services. To provide services in AI and analytics, the Platform for Network Data Analytics (PNDA) [45] and Acumos AI Platform[46] were analysed. The first is a scalable big data analytics platform for networks and services that brings together multiple technologies (e.g., Kafka). PNDA was used to enable closed loop control for an ETSI NFV environment. In addition, ONAP is integrating the PNDA as part of the DCAE to provide its analysis services to the ecosystem. On the other hand, Acumus AI Platform allows you to create, share, and deploy AI/ML models, capable of packaging ML models into microservices in portable containers. An "Acumos-DCAE Adapter" is developed to integrate ML models from an Acumos catalog to the ONAP DCAE. Through these open-source solutions, the architecture represented in *Figure 5* was designed. In such a configuration, Kubernetes can act as VIM and CISM. NSMF, NSSMF, NFVO, and NFVM functions can be provided by ONAP or OSM. Regarding the closed loop, MS and AE functions can be implemented using the MON and DCAE modules of OSM and ONAP, respectively, or directly using open-source monitoring tools (e.g., Prometheus and ELK) and analytics platforms (e.g., PNDA and Accumos). Integration fabric will collaborate with management functions deployed as services through the combination of features from Istio and Kafka. In turn, through Envoy sidecar proxies, the management functions are connected to form a service mesh managed by Istio. Synchronous communication between services can be enabled via Istio, whereas asynchronous communication can be performed via Kafka.

9.1.8 Researched Solution Synergy

Establishing cybersecurity in cloud-native services is a multifaceted endeavour where various technologies and methodologies converge synergistically. This section is dedicated to showcasing how the various solutions that were explored in the previous sections of this study work in synergy in order to facilitate the emergence of the features that are required for establishing cybersecurity in cloud-native services.

Strong Access Controls

Together, ZTA, service meshes, Native AI and SECaaS, and Security Risk Profiling and Mitigation establish a holistic approach to strong access controls that emphasizes continuous verification, least privilege, and adaptive responses, enhancing the organization's overall security posture. ZTA redefines

[43] https://www.onap.org/

[44] https://osm.etsi.org/

[45] http://pnda.io/ (accessed on 15 August 2023)

[46] https: //www.acumos.org/ (accessed on 15 August 2023)

access controls by adopting a “never trust, always verify” approach, ensuring stringent verification of identity, device health, and context for all entities. Service meshes complement this philosophy by providing granular access control mechanisms for microservice-based applications, enforcing secure communication between authorized services. Native AI and SECaaS continuously monitor behaviours, promptly adjusting access permissions based on real-time threat detection, and fortifying strong access controls against evolving threats. Concurrently, Security Risk Profiling and Mitigation assesses vulnerabilities and risks, guiding access control decisions, and ensuring that access controls align with identified security concerns, thereby reducing the attack surface.

Network Segmentation

Service meshes ETSI ZSM and ZTA are interconnected elements that contribute to network segmentation and bolster network security and management. Service meshes excel at securing and governing communication between microservices within applications, granting service-level segmentation within the application architecture. ETSI ZSM serves as a comprehensive framework and standard for automating network and service management, enabling the efficient orchestration of network segmentation policies and ensuring consistent application of security measures across network segments. Zero-trust architecture, a security paradigm that assumes no inherent trust, can be applied to both service-to-service communication (inside service meshes) and network-level access control (within network segmentation), thereby establishing a robust security posture where trust is never taken for granted, and access control is meticulous. The convergence of these three concepts facilitates a unified approach to network segmentation, fortifying security and management capabilities by securing communication at both the application and network layers while automating and orchestrating network segmentation.

Secure Communications

ETSI ZSM, ZTA, service mesh, and Shift Left and Static Testing Techniques all play vital roles in enhancing the security of communications within modern network and application ecosystems. ZSM’s automation framework ensures that network resources are optimized and security policies are consistently enforced, actively contributing to the establishment of secure communication channels. Meanwhile, ZTA, with its fundamental principle of dis- trust by default, strengthens secure communications through rigorous identity verification and continuous authentication, ensuring only authorized entities can access and exchange data. Service mesh further reinforces secure communications by applying authentication, authorization, and encryption mechanisms to inter-microservice communication within applications, guaranteeing data confidentiality during transit. Lastly, Shift Left and Static Testing Techniques, when integrated into the development process, proactively identify and rectify security vulnerabilities, thereby establishing secure communication as an inherent aspect of application design and development, fortifying the overall security posture across diverse IT environments.

Data Encryption

Service meshes, Native AI, SECaaS, and Security Risk Profiling and Mitigation all have a role in fortifying data encryption strategies to bolster data security. Within the context of data encryption, service meshes serve as a protective layer for data in transit within microservice-based applications, ensuring secure communication through mecha- nisms like TLS or mTLS. Native AI, when integrated into security solutions, contributes to data encryption by detecting anomalies in encryption key usage or protocol utilization, continuously monitoring and adapting encryption policies to evolving threats. SECaaS providers offer encryption solutions for various data storage and communication channels, simplifying the implementation of robust encryption measures without requiring extensive in-house expertise. Meanwhile, Security Risk Profiling and Mitigation efforts identify encryption as a pivotal control to mitigate specific security vulnerabilities and risks, particularly concerning data protection and compliance. Collectively, these elements converge to create a comprehensive approach to data encryption, enhancing overall data security across diverse IT environments.

Continuous Monitoring and Intrusion Detection

ZTA, service mesh, Continuous Security practices, Data Preserving and Data Compliance measures, and Shift Left and Static Testing Techniques all play integral roles in the context of Continuous Monitoring and Intrusion Detection. ZTA's fundamental principle of mistrusting all entities aligns seamlessly with continuous monitoring efforts, where all network activity, both internal and external, undergoes vigilant scrutiny for anomalies and potential intrusions. Meanwhile, service mesh offers insights into communication patterns within microservice applications and enforces security policies like encryption, contributing to enhanced intrusion detection capabilities. Continuous Security practices encompass the ongoing assessment of security, including monitoring network traffic, system logs, and user activities to swiftly identify and respond to threats. Data Preserving and Compliance measures help safeguard data integrity and compliance during monitoring, ensuring data remain secure and compliant. Lastly, Shift Left and Static Testing Techniques, integrated into the development process, serve to pre-emptively uncover and address vulnerabilities, reducing potential intrusion points and bolstering the overall efficacy of Continuous Monitoring and Intrusion Detection. Together, these elements collectively fortify an organization's security posture by providing the necessary tools and strategies for detecting, mitigating, and preventing security breaches in a continuously evolving threat landscape.

Process Automation

ETSI ZSM, service mesh, Shift Left, and Static Testing Techniques for Enhancing Security are interconnected in their roles within process automation, particularly in the context of bolstering security within organizational workflows. ETSI ZSM, functioning as a network and service management automation framework, streamlines processes by efficiently allocating resources and enforcing security policies, minimizing manual intervention, and mitigating potential human errors. Service mesh further enhances automation by automating security aspects of microservice-based communication, ensuring consistent policy application across services, and simplifying complex application environments. Shift Left and Static Testing Techniques, integrated early in the software development lifecycle, contribute to process automation by automating security testing, swiftly identifying and rectifying vulnerabilities before deployment, expediting security assessments, and reducing the necessity for manual code scrutiny. Together, these components drive process automation by automating various network, service, and security tasks, ultimately enhancing efficiency, consistency, and security throughout an organization's operations and development lifecycle.

Vulnerability Management

Security Risk Profiling and Mitigation, Shift Left, and Static Testing Techniques for Enhancing Security, Continuous Security, Data Preserving, and Data Compliance are intricately interwoven with vulnerability management, creating a cohesive strategy for identifying, prioritizing, and mitigating vulnerabilities. Vulnerability Identification is fortified through early detection in the software development lifecycle (Shift Left) and ongoing vulnerability scanning and real-time monitoring (Continuous Security). Security Risk Profiling aids in prioritization based on the risk posed by identified vulnerabilities, directing vulnerability management efforts toward the most critical threats. Mitigation strategies benefit from Shift Left practices, ensuring security controls and configurations are rigorously tested before deployment, whereas Data Preserving and Data Compliance measures safeguard sensitive information during mitigation. Continuous Monitoring offered by Continuous Security maintains a vigilant stance, identifying new vulnerabilities and promptly incorporating them into vulnerability management processes. Together, these elements establish a comprehensive approach to vulnerability management that efficiently addresses vulnerabilities, prioritizes them based on risk, and maintains data security and compliance throughout the vulnerability lifecycle

Configuration Management

Shift Left and Static Testing Techniques for Enhancing Security, in conjunction with ETSI ZSM, play a pivotal role in fortifying Configuration Management's security and reliability aspects. Shift Left practices advocate the integration of security assessments into the early stages of the software development lifecycle, employing static code analysis and other static testing methods to uncover vulnerabilities and security flaws. These techniques are equally applicable to Configuration

Management, where they ensure the early detection and remediation of security-related issues in configuration files and settings. Meanwhile, ETSI ZSM automates network and service management, encompassing configuration management for network devices and services. This automation facilitates the consistent application of security configurations, such as access controls and firewall rules, reducing the risk of misconfigurations that could lead to security breaches. Collectively, Shift Left, Static Testing Techniques, and ETSI ZSM synergize to bolster Configuration Management, ensuring that security configurations are validated, accurate, and compliant, thus enhancing the overall security and reliability of IT environments.

Continuous Security Testing

Security Risk Profiling and Mitigation, Shift Left and Static Testing Techniques for Enhancing Security, Continuous Security, Data Preserving, and Data Compliance are all interconnected components of Continuous Security Testing. Security Risk Profiling and Mitigation contribute by assessing vulnerabilities and prioritizing them based on risk, guiding testing efforts to address the most critical threats. Shift Left practices ensure early integration of security testing, including static code analysis, into the development process, reducing vulnerabilities introduced in code changes. Continuous Security encompasses real-time threat detection and vulnerability scanning during testing, ensuring continuous assessment of security. Data Preserving measures secure data during testing whereas Data Compliance ensures regulatory adherence. Together, they form a holistic approach to maintaining an ongoing, effective, and compliant security testing process, minimizing security risks and vulnerabilities throughout the software development lifecycle.

9.2 Cloud Native Security Mechanisms

XR applications are moving towards micro-services-based architectures and Edge/Cloud environments. Despite the benefits and flexibility, the rise of (Cloud-Native) micro-services-based architectures is not without its challenges. They are composed of numerous and dynamic components that need to communicate and interact with each other. Hence, one of the biggest challenges is dealing with the exponential and complex relationship between all the moving parts forming a Cloud Native application.

In addition to the specific security and privacy discussed in section 9.1.1, it is crucial to understand how the security of XR applications fits into (secure) Cloud-Native environments. XR applications have specific traits such as the vast amount of involved data (e.g., audio/video), low-latency requirements or the multi-user and network bandwidth-hungry scenarios requiring tailored security mechanisms.

Thus, the following section provides an overview of some of the security best practices and research challenges that focus on bringing security to cloud environments and how that supports the required security of XR applications. Moreover, for each challenge, we discuss candidate open-source enablers that support their realization. Finally, this section provides an overview of the proposed security mechanisms, including a proposal for an Autonomic and Cognitive Security Management Framework for detecting and mitigating anomalies, the proposed approach for a Cloud-Native anomaly detection and mitigation framework, and the conducted experiments.

Cloud-Native Security splits into four key levels, the so-called “4C’s” of Cloud-Native Security[47]: the security of Cloud, Clusters, Containers and Code security.

***Cloud/Datacenter* security** concerns the security of the underlying infrastructure. Here, it is essential to protect and control, amongst others, the (network) access to the environment APIs, Cloud Provider APIs, and infrastructure. Indeed, Cloud Environments rarely target a single user but rather a highly complex multi-tenancy environment. Hence, it is crucial to guarantee that only authorized people have

[47] https://kubernetes.io/docs/concepts/security/overview/

access to this environment and, consequently, access to some data or service. Proper authentication and granular authorization play a vital role in this.

Moreover, it is crucial to adopt the best practices and act according to legislation, never forgetting that a geographically distributed environment must comply with regulatory standards under industry guidelines and local/national/international laws. For instance, in European Union, GDPR and the ePrivacy directive ensure that the involved actors and parties follow security and privacy best practices. Cloud compliance ensures that cloud computing services meet compliance requirements being valuable and necessary in the case of personal processing information. In that regard, security assessments, such as Kube-bench, can be used to attest compliance by running automated tests. Alternatively, InSpec also allows testing and auditing infrastructure (and applications).

***Cluster* security** includes the protection of the components of the cluster, for instance, by leveraging authentication and admission control mechanisms or network policies for controlling clusters, nodes, and pod communications and operations. To help in accomplishing that, the Center for Internet Security (CIS) Benchmarks provides guidelines for cybersecurity best practice recommendations for configuring Kubernetes with the primary goal of providing tailored recommendations for Kubernetes to improve security against threats. Then, tools such as Kube-bench[48] or InSpec allow the automation of such security assessments. Kube-bench allows automating the implementation of CIS Benchmarks, offering recommendation actions for detected issues. Similarly, InSpec allows the automation of infrastructure testing, auditing of applications and policy conformance. InSpec compares the actual state of the systems with the desired state, and whenever it detects deviations, it issues a report. Likewise, Kube-hunter[49], a vulnerability scanning tool, aims to increase the awareness and visibility of security controls in Kubernetes environments.

Furthermore, Cloud-Native environments require continuous security, meaning it is not enough to worry about security before the execution. It is equally important to pay attention to security during the lifetime of the environment by analysing and monitoring, for instance, unauthorized access and anomalous traffic. Even so, such security monitoring should not compromise the availability of the applications, especially in XR, where such mechanisms should not affect the already strict latency requirements of XR applications. Finally, tools such as Zabbix[50] or Falco[51] are pivotal in monitoring many aspects of the Cloud-Native environment and XR applications. Falco, a Cloud-Native threat detection engine, relies on monitoring Linux system calls in containers to flag unexpected behaviours (e.g., privilege escalation events, suspicious read/writes to known directories). To accomplish that, Falco uses a set of predefined rules. Similar, Curiefense is a Cloud-Native tool capable of monitoring HTTP API requests, allowing the detection of suspicious behaviours on HTTP traffic between containers.

***Container* security** refers to implementing the best security practices to ensure containers are free from vulnerabilities, for instance, by using signed and trustable images or avoiding running containers as privileged users. Container vulnerabilities may be due to incorrect infrastructure configurations, vulnerabilities inherited from container base images, or the application code itself. Tools like Curiefense[52] or Falco[53] are fundamental for detecting and quickly reporting unexpected application runtime behaviours. Despite all the benefits of such tools, most existing tools mainly rely on the usage

---

[48] https://github.com/aquasecurity/kube-bench

[49] https://kube-hunter.aquasec.com/

[50] https://www.zabbix.com/

[51] https://falco.org/

[52] https://www.curiefense.io/

[53] https://falco.org/

of hard-coded rules, which might limit them to only known vulnerabilities. Moreover, XR-specific tools that could help to detect XR-specific code and behavioural vulnerabilities are still an open challenge.

***Code* security** encompasses methods to automate the testing of source code for known security issues or the automated verification of insecure third-party libraries. In this regard, different tools can help to automate the process and vulnerability search. Static Application Security Testing (SAST) tools, like Checkmarx[54], allow verifying source code against known harmful patterns that risk the security of applications. Likewise, Dynamic Application Security Testing (DAST) tools OWASP Zed Attack Proxy (ZAP[55]) can be used to dynamically assess the running behaviour of applications as an attacker would do. Moreover, tools like Microsoft Credential Scanner (CredScan) help identify credential-related vulnerabilities and leaks in source code and configuration files (e.g., default passwords, SQL connection strings or exposed private keys). Likewise, tools such as WhiteSource Bolt[56] or Black Duck[57] are pivotal in searching for known vulnerabilities.

Hereunder, we introduce three approaches for protecting cloud-based XR applications. The proposed solutions, namely the FortisEDoS Framework, Autonomic and Cognitive Security Management Framework, and Security as a Service for Service Mesh aim to enhance security and resilience by implementing security measures at different levels, using deep learning-based DDoS detection models, and relocating microservices. These approaches aim to provide uninterrupted and high-quality XR experiences while mitigating the impact of security threats like DDoS attacks.

9.2.1 Autonomic and Cognitive Security Management Framework

Modern XR applications are being designed into smaller, more manageable microservices for different reasons, such as portability, scalability, or reliability, considering the Cloud-Native environments. From a security standpoint, such an emerging paradigm raises new, open challenges in managing the volume and complexity of Cloud-Native applications. As such, this demands intelligent and automated solutions to lower the burden assigned to humans on managing the security of Cloud-Native applications. With the objective of autonomous security management that enables hierarchical end-to-end security self-management resources in various domains, a framework that adheres to the key design principles of ETSI ZSM was proposed, as shown in *Figure 115*. This framework introduces AI-powered closed-loops with different scopes, from node to end-to-end and inter-slice levels. Thus, it allows the rapid and effective detection and mitigation of security threats close to the source, which prevents their proliferation in the network.

In this architecture, the closed loop is represented by four functions, Monitoring System (MS), Analytics Engine (AE), Decision Engine (DE) and Enforcement Service (ES). ES allows triggering/updating implementations of specific virtual security functions (VSFs), such as vFirewalld, vIDS, through the management and orchestration platform (MANO). Each network function is associated with a Security Element Manager (SEM) that will be responsible for managing security within its scope. As mentioned earlier, cognitive resources are incorporated into closed loops for security analysis and decision-making. Furthermore, AI/ML techniques can be implemented in MS and ES to increase the cognitive level of the environment.

In order to have greater security and mitigation at different levels, multiple closed loops can be coordinated at different levels. These loops are managed and orchestrated by the “Trust & Security Manager” (TSM), which comprises three functional modules, the “Security Orchestrator”, “Security Policy and SSLA Manager”, and “Trust Manager”. The “Security Orchestrator” is responsible for designing, instantiating, and managing the runtime lifecycle of circuits. The Security Policy & SSLA

[54] https://checkmarx.com/

[55] https://owasp.org/www-project-zap/

[56] https://www.whitesourcesoftware.com/free-developer-tools/bolt/

[57] https://www.blackducksoftware.com/

Manager is responsible for coping with violations of SSLAs (Security Service Level Agreements) and security policies defined by external entities. The Trust Manager is responsible for the continuous assessment of the reliability of the network services and associated circuits (this trust is calculated based on the trust attributes specified in the Trust Level Agreement).

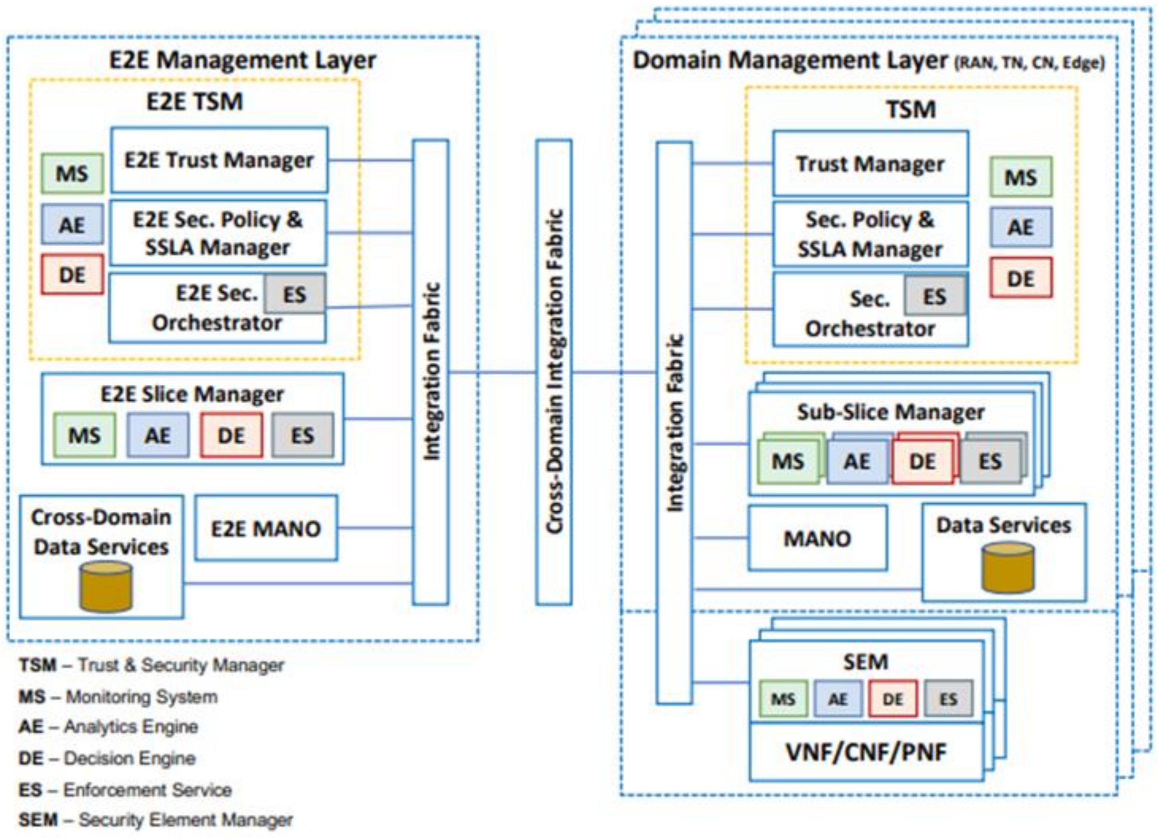


Figure 115: Architecture of envisioned Autonomic and Cognitive Security Management framework [87].

Based on this framework, some open-source solutions that enable zero-touch security management in environments were analysed. A cloud-native architecture, based on stateless microservices implemented as containers, is a technology recognized for being suitable for the cost efficiencies, flexibility and scalability required in the operation and management of environments. In this sense, the concept of Platform-as-a-Service (PaaS) emerges as a cloud-native architecture layer that allows developers to implement, run and manage different applications without the complexity of configuring and maintaining the cloud. In this follow-up, Kubernetes appears as a de-facto standard for the implementation and orchestration of applications in containers, which allows scalability, high availability, and fault tolerance features.

For the implementation of the Fabric Integration features, including facilitating interoperability and communication between management services (within and between domains), Service Mesh solutions such as Istio or Linkerd[58] were analysed, as well as an event streaming platform like Apache Kafka. On the one hand, the service mesh will allow the management of traffic between services while enhancing security and observability. The event streaming platform will handle asynchronous communications between applications and services. In addition, the use of the event streaming platform is also important for security use cases, including monitoring, analysing and reacting to security threats in real-time. In this sense, the use of Istio and Kafka as candidates for Integration Fabric implementation was considered.

[58] https://linkerd.io/

In the context of management and orchestration platforms, Open Network Automation Platform (ONAP)[59] and Open Source MANO (OSM)[60] are highlighted. ONAP provides a framework for real-time, policy-driven orchestration, management and automation of network services and edge computing. This platform includes different ecosystems, such as POLICY, CLAMP, DCAE, which together support closed-loop automation. POLICY provides policy creation and validation capability. CLAMP designs and manages closed control loops. DCAE (Data Collection, Analytics and Events) collects and analyses data. On the other hand, OSM allows modelling and automating the life cycle of network functions, network services and network slices, through MON and POL modules. MON leverages monitoring tools to collect VNFs and infrastructure metrics. POL is a policy management module. Another tool to highlight is Ansible, which allows production-level automation in a cloud-native environment, and as such, allows increasing the automation capabilities of NFVM and NFVO in the management and orchestration of network functions and services.

From a monitoring point of view, some tools have already been mentioned in Section 5.1.2. Therefore, in this context, a combination of Prometheus, Elasticsearch, and Grafana was adopted in order to build a more efficient monitoring system.

To provide services in AI and analytics, Platform for Network Data Analytics (PNDA)[61] and Acumos AI Platform[62] were analysed. The first is a scalable big data analytics platform for networks and services that brings together multiple technologies (e.g., Kafka). PNDA was used to enable closed-loop control for an ETSI NFV environment. In addition, ONAP is integrating the PNDA as part of the DCAE to provide its analysis services to the ecosystem. On the other hand, Acumus AI Platform allows you to create, share and deploy AI/ML models, capable of packaging ML models into microservices in portable containers. An “Acumos-DCAE Adapter” is developed to integrate ML models from an Acumos catalog to the ONAP DCAE.

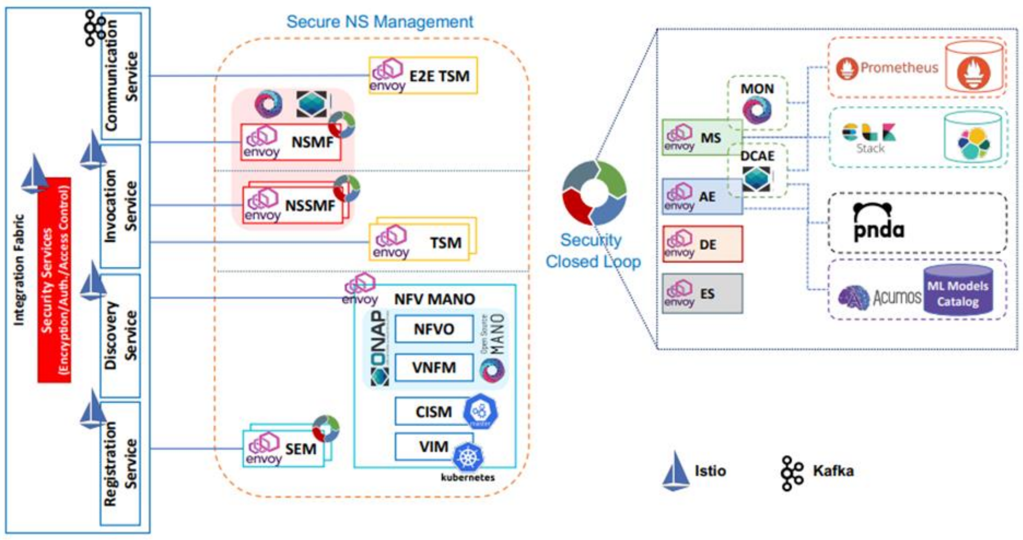


Figure 116: Architecture of the framework with the enablers and tools [87] .

Through these open-source solutions, the architecture represented in *Figure 116* was designed. In such a configuration, Kubernetes can act as VIM and CISM. NSMF, NSSMF, NFVO and NFVM functions can be provided by ONAP or OSM. Regarding the closed loop, MS and AE functions can be implemented

[59] https://www.onap.org/

[60] https://osm.etsi.org/

[61] http://pnda.io/

[62] https://www.acumos.org/

using the MON and DCAE modules of OSM and ONAP, respectively, or directly using open-source monitoring tools (e.g., Prometheus and ELK) and analytics platforms (e.g., example, PNDA and Accumos). Integration Fabric will collaborate with management functions deployed as services through the combination of features from Istio and Kafka. In turn, through Envoy sidecar proxies, the management functions are connected to form a service mesh managed by Istio. Synchronous communication between services can be enabled via Istio, while asynchronous communication can be performed via Kafka.

9.2.2 Cloud Native Anomaly Detection and Mitigation (FortisEDoS Framework)

XR services include VR, AR, and MR. They are highly sensitive to delays, require significant bandwidth, and process a large amount of data. Network slicing plays a key role in deploying XR services on the cloud, but it also poses significant security challenges due to the dynamic nature of slicing and the sophistication of cyber threats. Effectively, to efficiently use the cloud resources allocated for a XR application, one can change the number of replicas and the cloud resources such application uses to handle them. However, in such conditions, the application becomes vulnerable to EDoS (Economic Denial of Sustainability) attacks, whereby an attacker may flood the relevant servers with a huge number of requests, and causes the need for scaling up the resources. Then, the illegitimate requests are reduced and this cycle is repeated multiple times. In such a situation, the service providers face economic damages. To solve this problem and not let an adversary affect the functionality of the applications as well as the cost they incur, a Cloud Native Anomaly Detection and Mitigation method is proposed in this section.

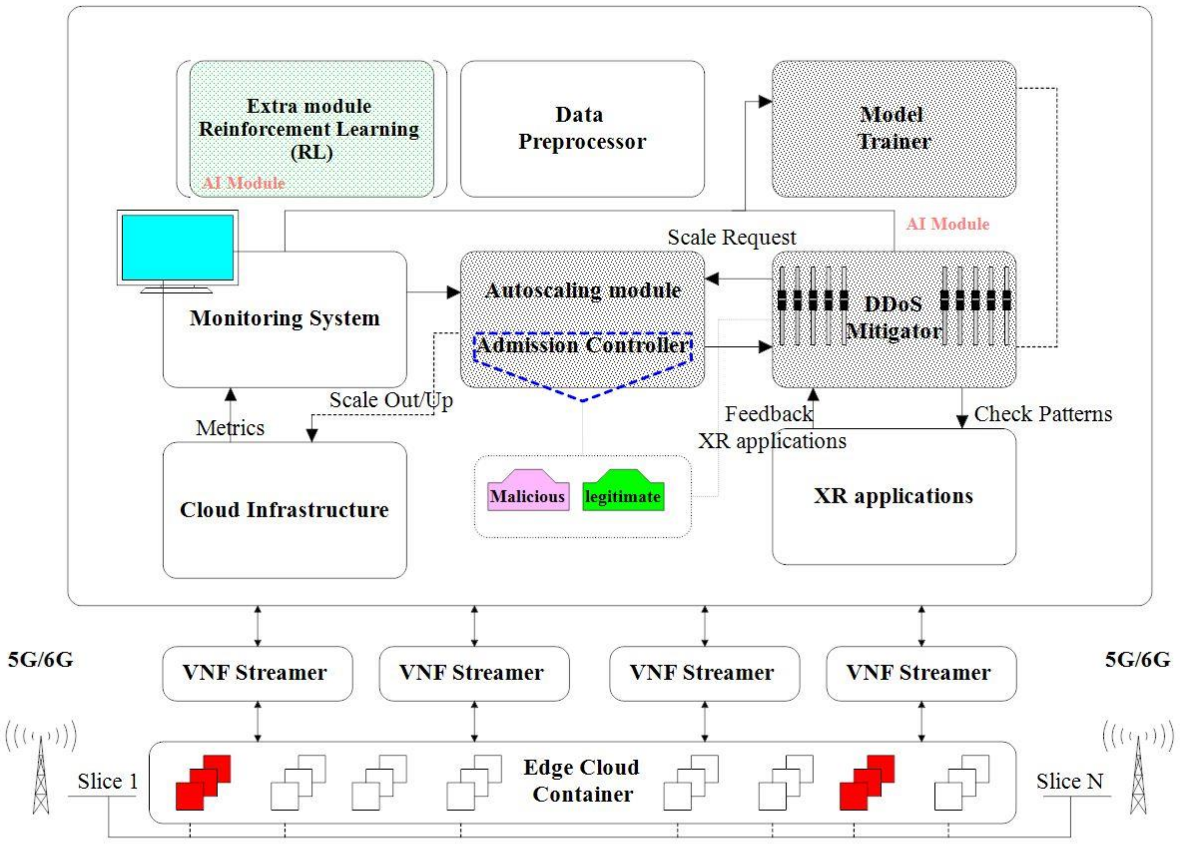


Figure 117: Boosting Anomaly Detection Accuracy for XR Applications' EDoS Mitigation with AI.

As stated earlier, XR services, especially VR applications, heavily rely on real-time data transmission and are highly sensitive to latency. Thus, they are vulnerable to DDoS attacks that can disrupt real-time data delivery, causing severe degradation or interruption of the XR experience. FortisEDoS's deep learning-powered DdoS anomaly detection model, CG-GRU, can be adapted to recognize patterns of DdoS attacks targeting network slices hosting XR services [88][89]. FortisEDoS is a cutting-edge platform that utilizes a deep learning-powered forecast-based anomaly detection model called CG-

GRU. This model is designed to accurately identify any malicious scaling requests that may be made to the system. The platform leverages Convolutional Neural Networks (CNNs), Graph Neural Networks (GNNs), and Recurrent Neural Networks (RNNs) to extract both temporal and spatial dependencies in the data. By leveraging spatio-temporal correlations, it can accurately detect and mitigate these attacks in real-time, ensuring uninterrupted XR experiences. Also, FortisEDoS's utilization of transfer learning can enhance the security of newly deployed XR slices by leveraging insights gained from previous deployments, thereby improving the resilience of XR services against evolving security threats.

Furthermore, the integration of Reinforcement Learning (RL) models holds promises for enhancing anomaly detection accuracy tailored to the requirements of XR applications, such as minimizing delay and optimizing resource capacity. This augmentation underscores security mechanisms' adaptive and responsive nature in catering to the dynamic demands of XR environments (*Figure 117*).

At each time step, the values of the Virtual Network Function's (VNF – composing the network slice running the micro-services of the XR application) metrics are analyzed, and if any anomalies are detected, they are flagged as anomalous. This flagging process is based on a dynamic thresholding technique, whereby the global forecasting error is calculated, and if it exceeds the threshold, then the corresponding metric is considered anomalous. In case of any malicious scaling requests, the system will tag and refuse the scaling operation as malicious. This advanced anomaly detection system ensures that FortisEDoS remains a secure and reliable platform for all users.

This framework could incorporate various approaches, such as transferring affected micro-services to safer edge cloud nodes and rejecting the auto-scaling operation. To prevent adversarial attacks against ML models, it would be beneficial to use MTD-based robust ML models. To enhance the accuracy of the EDoS detection and mitigation framework, it is a good idea to include metrics that are relevant to the cloud infrastructure. Also, connecting with the XR service provider can make the framework even more effective by providing data on significant changes in XR service consumption patterns. For example, when an EDoS mitigator needs to auto-scale the operation, it can use the trained model and the collected infrastructure-related data to determine its legitimacy. The EDoS mitigator can then inquire the relevant XR service provider about any notable changes in the XR service consumption before rejecting the operation. Changes in XR service consumption may include an increase in active consumers of the XR service, leading to higher generation of XR stream data, which would require additional cloud resources for processing. Alternatively, it could be an influx of users accessing the system and consuming XR stream data. If the XR service provider confirms such changes, the EDoS mitigator can reassess its decision and allow the necessary scaling up of resources. If the XR service provider indicates that there was no major change in the XR service consumption, the EDoS mitigator can reject the auto-scaling operation as it may be considered potentially malicious. By communicating with the XR service provider and validating the details, the EDoS detection and mitigation framework can improve its accuracy and make better decisions regarding resource scaling.

Generally speaking, FortisEDoS is a framework designed to ensure the resilience of elastic B5G services (including XR services) against EDoS attacks. It integrates CGGRU, a deep learning-powered DDoS anomaly detection model that accurately discerns malicious behavior. Transfer learning enhances detection sensitivity while accelerating the training process. FortisEDoS provides intuitive explanations for its decisions, fostering trust in deep learning-assisted systems.

We compared the CG-GRU model with other variants, including G-GRU and GRU models, and a LSTM-based model proposed in a previous work. We conducted a layer ablation study to assess the impact of different layers on the performance of the models. The evaluation was done using Precision, Recall, and F1 metrics, and the results showed that CG-GRU outperformed all other models (*Figure 118*). The model demonstrated high sensitivity in identifying anomalous CNF's status while maintaining an acceptable Precision and a reasonable F1 score. We also evaluated the computation and storage overhead induced by CG-GRU and its counterparts, and the results showed that CG-GRU model brought a significant reduction in training time, inference time, and model size compared to the best-performing LSTM-based model (*Figure 119*). The study concludes that the quality of the spatio-temporal features learned by the feature extraction block is crucial for achieving high performance in multivariate time series anomaly detection, and that capturing both spatial and temporal dependencies is important. The results also suggest that adding Conv1D layer and GAT layers can boost the model's performance and improve the estimation of the anomaly threshold for discriminating anomalous CNF's status. We have analyzed the computation and storage overhead associated with CG-GRU model and its counterparts. The study measured various variables, including the average training time, average inference speed, and model size. The results are presented in *Figure 119*, where we can compare the data. One of the key observations is that GRU-based models are quicker to train and make inferences, and occupy much less storage space than LSTM-based models. This can be attributed to GRU cells using fewer parameters and gates than LSTM cells while accomplishing the same task. The data in *Figure 119* indicates that CG-GRU model is up to 12.42%, 23.73%, and 21.54% more efficient in terms of training time, inference time, and model size, respectively, as compared to the best-performing LSTM-based model (i.e., CG-LSTM).

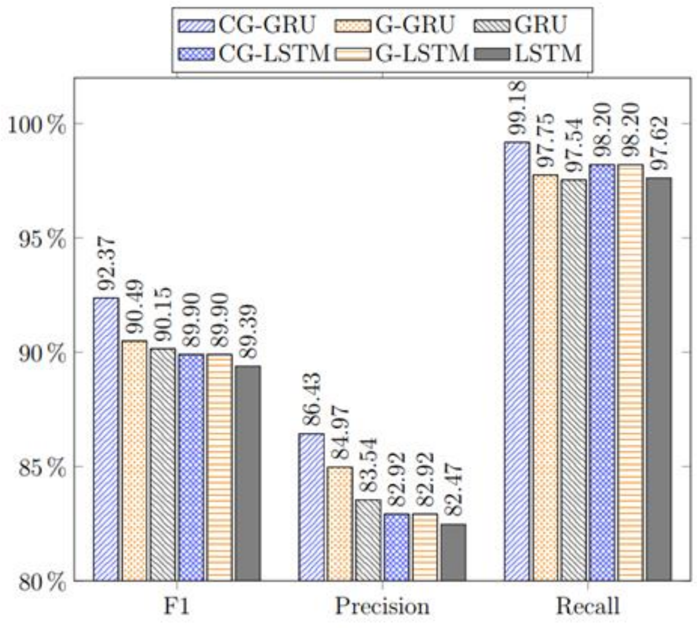


Figure 118: Attack Detection Performances [89].

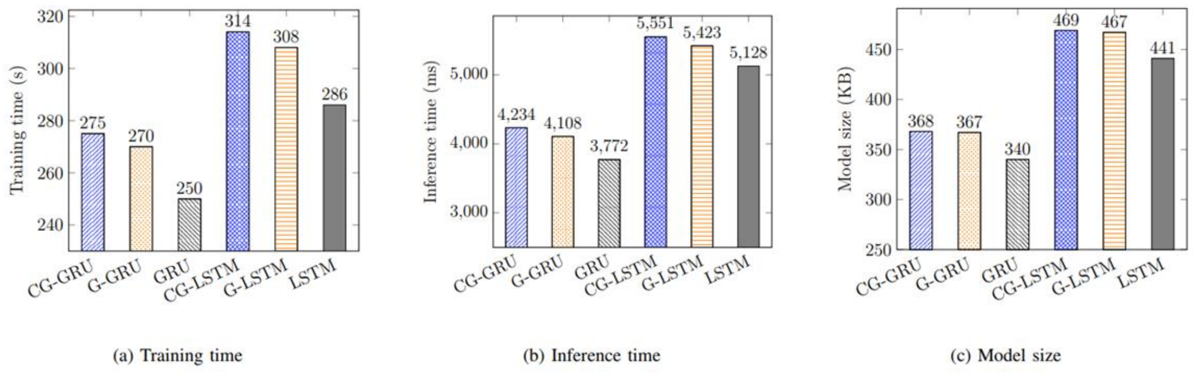


Figure 119: Computation and storage costs of the models [89].

The CG-GRU model effectively predicts a normal CNF's status with high accuracy. It achieved a low prediction RMSE of 0.039 and MAE of 0.1189 on the validation dataset.

The CG-GRU model's high forecasting quality is due to its ability to capture time dependencies and spatial correlations among CNF's metrics. The model's accuracy is evident from the closely matched forecast and ground truth values for the video streamer's metrics in *Figure 120*.

During DDoS attack periods, the deviation between forecast and real values becomes pronounced, indicating an increase in prediction error due to abnormal metric values. The framework can use different methods to tackle adversarial attacks, like transferring services to safer nodes and rejecting auto-scaling. MTD-based robust ML models can improve resilience against such attacks. Enhancing EDoS detection and mitigation, including cloud infrastructure metrics and connecting with XR service providers, can improve the framework's effectiveness. For instance, when an EDoS mitigator needs to auto-scale the operation, it can utilize the trained model and collected infrastructure-related data to determine its legitimacy. By communicating with the XR service provider and validating the details, the EDoS detection and mitigation framework can improve its accuracy and make better decisions regarding resource scaling.

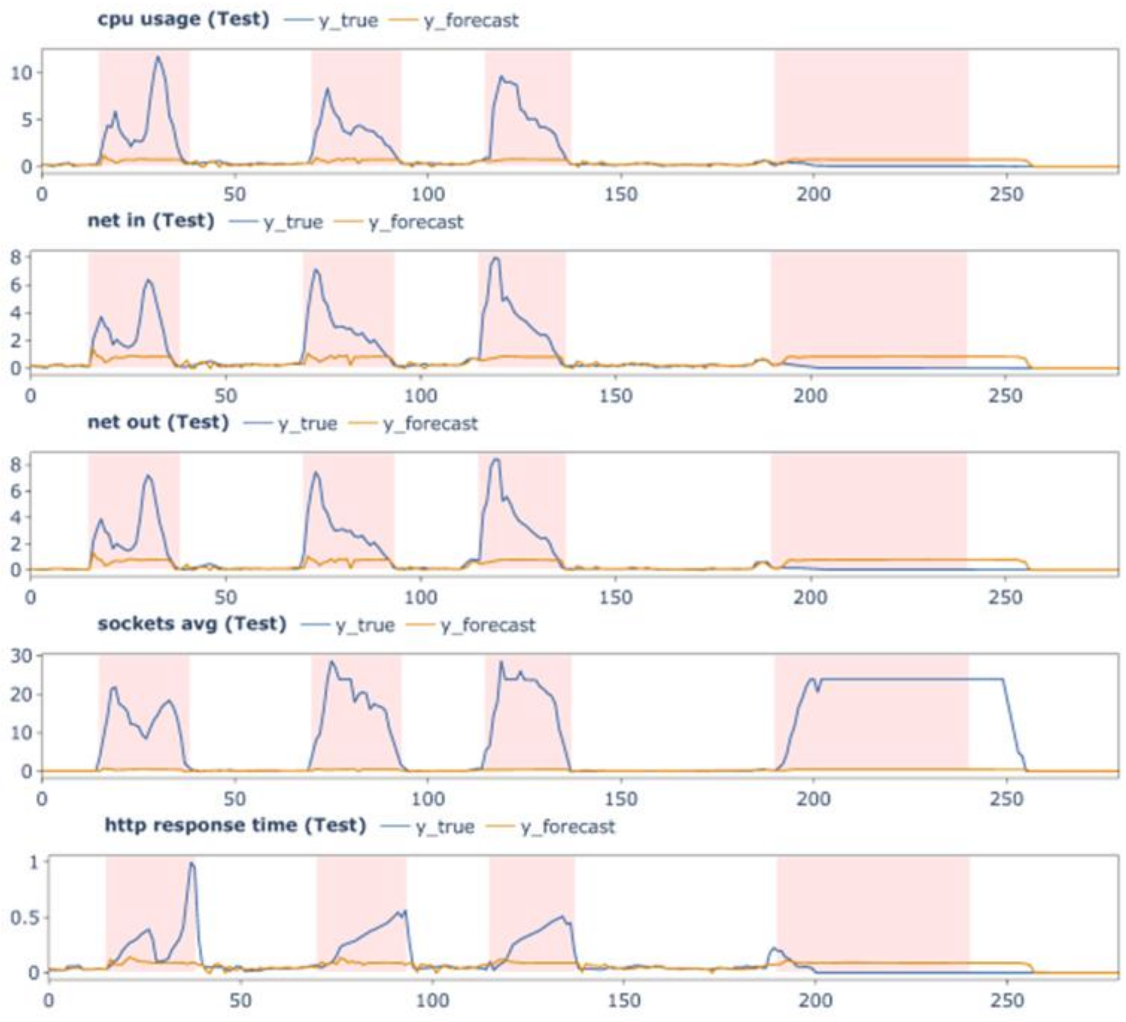


Figure 120: Predictive Analysis of Video Streamer CNF Performance and detection of Hulk and Slowloris attacks in testing data [89].

Figure 121 shows a heatmap of forecasting errors over time for slice 1's video streamer. The x-axis represents time steps, and the y-axis shows CNF metrics. Dark blue indicates zero error, while dark red signifies higher error. The heatmap visually identifies attack patterns and impacted metrics. The largest errors occur during attacks, with clear distinctions between Hulk and Slowloris patterns. *Figure 121* shows a heatmap of forecasting errors over time for slice 1's video streamer. The x-axis represents time steps, and the y-axis shows CNF metrics. Dark blue indicates zero error, while dark red signifies higher error. The heatmap visually identifies attack patterns and impacted metrics. The largest errors occur during attacks, with clear distinctions between Hulk and Slowloris patterns. *Figure 122* reveals the top three metrics contributing to forecasting errors during Hulk and Slowloris attacks, aligning with their respective natures. Hulk's high HTTP request rate, open sockets, and CPU usage reflect its high-rate nature, while Slowloris's impact on open sockets mirrors its low-rate strategy. These findings confirm the distinct behaviors of Hulk and Slowloris attacks, as evidenced by their respective effects on forecasting errors. Reveals the top three metrics contributing to forecasting errors during Hulk and Slowloris attacks, aligning with their respective natures. Hulk's high HTTP request rate, open sockets, and CPU usage reflect its high-rate nature, while Slowloris's impact on open sockets mirrors its low-rate strategy. These findings confirm the distinct behaviors of Hulk and Slowloris attacks, as evidenced by their respective effects on forecasting errors.

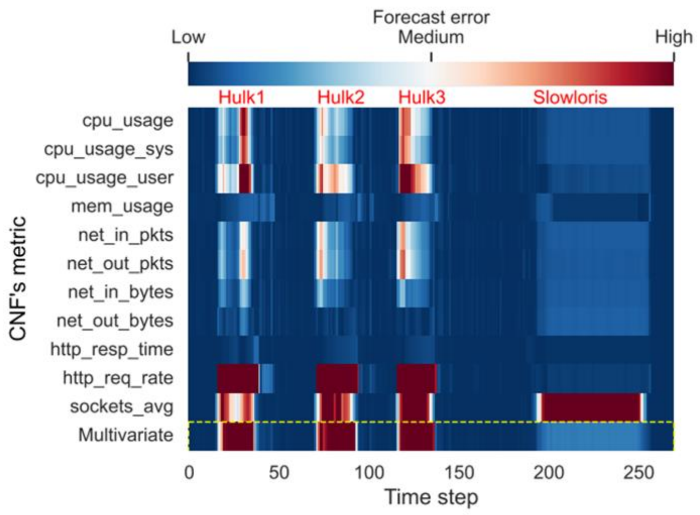


Figure 121: A heat map visualization of forecasting errors per CNF's metric for interpretability [89].

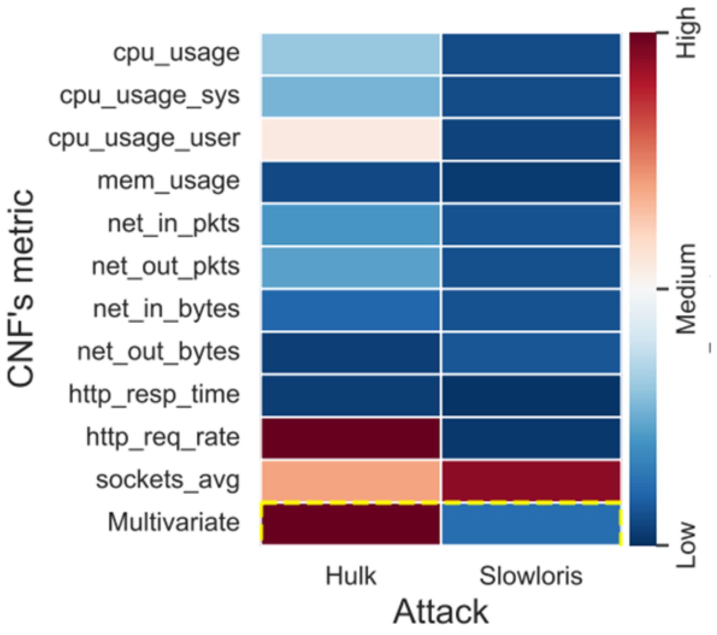


Figure 122: The visual signature of Hulk and Slowloris attacks using the forecast errors heat map [89].

9.2.3 Security as a Service for Service Mesh

The large amount of data processed by XR applications, and the crucial need for keeping the latency low, inspire us to shift the security mechanisms to other microservices rather than the XR microservices. In other words, we can provide a security service to the XR microservices and do not involve them in the details of threat mitigation procedures. This approach maintains the scalability of the whole application while efficiently utilizing the resources to fulfill the needs of XR services.

The first point to consider is the protection of data transferred between microservices of an application. As communication between microservices can be implemented based on service mesh infrastructure, the most basic approach could be to use the mTLS (mutual Transport Layer Security) protocol. Based on this protocol, the connection between two microservices is encrypted and only the end-points of that connection can decrypt it. However, due to the large amount of data transferred among XR microservices, they may run out of resources for performing encryption/decryption processes and have to scale-up their resources. This is against the scalability goal of 5G and beyond services (including XR services), and hence, we can provide microservices that serve related security utilities. This approach is a sort of security-as-a-service (SEaaS) concept.

The main challenge in designing a SEaaS solution for XR services is the delay that the extra connections between XR and SEaaS microservices may cause. We plan to relocate the microservices on the pods with lower end-to-end delay with the XR pods, and also provide multiple replicas to distribute the requests among them. Moreover, when the number of requests grows, we can also increase the number of replicas. To avoid EDoS attacks against this solution, the above-mentioned Cloud Native Anomaly Detection and Mitigation method (i.e., FortisEDoS) can be leveraged.

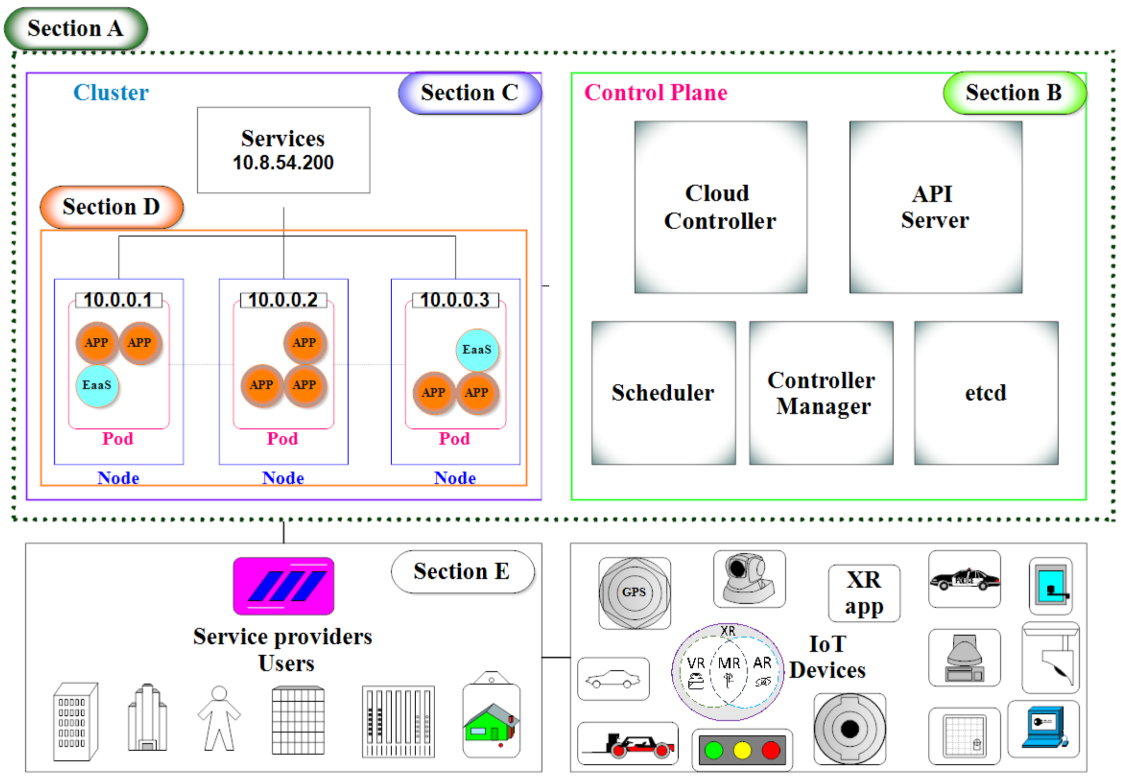


Figure 123: The architecture of deploying SEaaS in Kubernetes for securing XR applications [90].

We can consider Service Mesh (SM) as tools designed to address the challenges of managing traffic, security, and observability in microservices and distributed architectures. The architecture of such solution in Kubernetes can be as shown in *Figure 123*. Our architecture uses Encryption as a Service (EaaS) through Kubernetes Service Meshes to provide effective cryptography services to microservices on resource-limited devices [91]. In *Figure 123*, Section A is considered the core of Kubernetes, as it performs coordination and management tasks. Its functionality is based on various components that work together seamlessly to ensure the system's smooth operation. These components include the Kubernetes API server, which acts as the central control point for all Kubernetes-related operations, and the etcd key-value store, which provides a distributed data store for the system's configuration data. Moreover, Kubernetes uses various controllers and schedulers to manage the deployment and scaling of applications and services on the cluster, while ensuring that resources are allocated efficiently and fairly across all nodes in the system.

The section B in *Figure 123* refers to a cluster of multiple servers operating under Kubernetes' management. These servers are designed to host a set of applet applications that can be seamlessly deployed to other servers by Kubernetes and be immediately ready to run. This system allows

Kubernetes to manage load-balancing efficiently, ensuring maximum resource utilization and preventing downtime. EaaS is a secure tool for managing resource requests, with Kubernetes checking and approving them before providing access via Kubectl commands. It offers a seamless user experience and ensures the security of sensitive information.

In the context of service providers, it is possible for users to also act as service providers (Section E). For instance, a user may have many Internet of Things (IoT) devices that can be utilized for EaaS-related tasks. After appropriate resource allocation, these devices can communicate with servers and perform the necessary work associated with EaaS. This can include a variety of tasks, such as data processing, storage, and analysis. This approach allows for more efficient and cost-effective delivery of EaaS, while also leveraging the capabilities of distributed IoT networks.

9.2.4 Cloud Native Anomaly Detection and Mitigation

As discussed before in section 9.2, the recent shift of modern applications to highly distributed and dynamic Cloud-Native environments and the growing volume of heterogeneous communications resulting from this shift increases the complexity of managing security. Indeed, Cloud-Native applications demand tailored security measures to protect their microservices, which can span multiple domains.

According to these challenges, a highly-needed security mechanism consists of a security-focused analytics service offered as part of the overall managed functions of the CHARITY framework. Such a service constitutes an essential building block in the overall security and privacy strategy for XR applications. Such an integrated service is of the utmost importance, for instance, to enable various processing mechanisms (e.g., anomaly-detection algorithms) in a more efficient and scalable way. Numerous scenarios require security, such as the real-time video processing of surrounding environments captured by the XR HMDs to find bystander-sensitive information. Likewise, as initially investigated, such a service can also be used to detect network traffic anomalies.

Figure 124 illustrates how the preliminary SECaaS reference architecture maps into the overall CHARITY architecture and the initial mapping of the surveyed open sources for each component focused on the network anomaly detection problem.

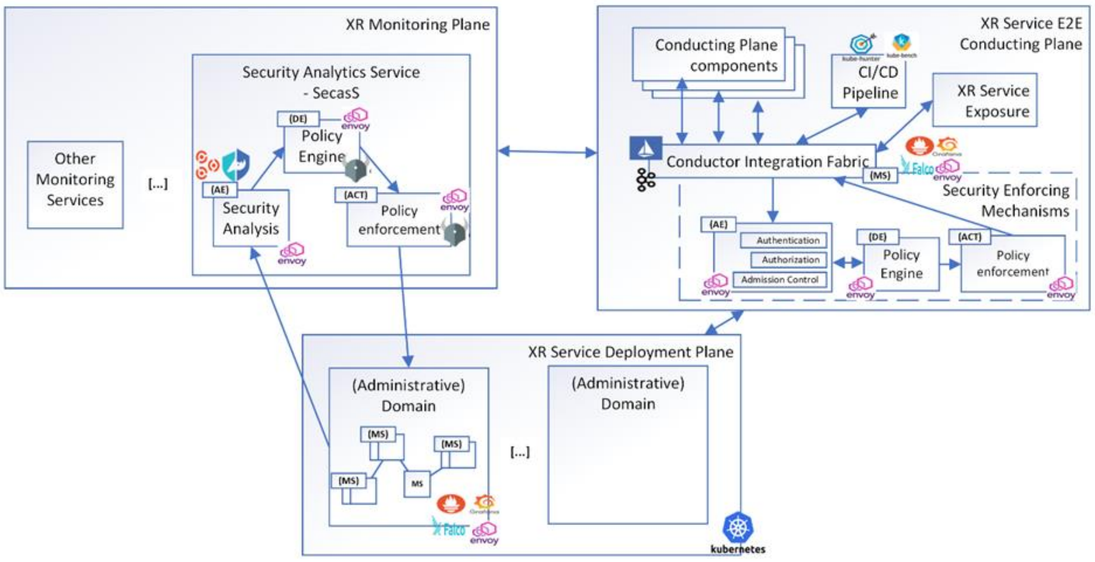


Figure 124: Mapping open-source enablers for network security in CHARITY architecture.

The proposed network anomaly service builds upon closed loops, as described in the ETSI ZSM specification, to implement an autonomous and intelligent process of detecting and reacting to network security issues. It comprises the monitoring agents, an analytics component, and a policy engine. More, it leverages the concept of a Service Mesh and sidecars as strategically placed elements to extract network data and enforce security policies.

The Istio/Envoy pair were part of the initial experimental works. Istio provided traffic management capabilities (e.g., ingress traffic control) and Envoy the sidecars functionalities. Service Mesh makes it possible to support the collection of network data and the application of network security policies without changing the services to be monitored. Open Policy Agent (OPA) was also tested as a more decoupled and forward-looking alternative to enforce and fine-control distinct security policies at different levels of a Cloud-Native environment. For network traffic, instead of solely relying on Envoy and Istio filtering capabilities, an Envoy's External Authorization filter was investigated to delegate the network authorization decision to OPA. In the initial scenario, the analysis component included the usage of NFStream[63] to aggregate and extract features from network packets and a machine learning-based network classification model to classify network packets into normal or anomalous traffic. The output of such classification supported the decision of whether the network communication should be blocked.

Additional security tools explicitly tailored for Cloud Native environments will be further investigated. Falco will be considered for detecting malicious activity attempts and signature-based CVE exploits as an input for the Analytics Engines (AE). Kube-Hunter will be considered to provide information on the cluster's vulnerabilities and whether pods are vulnerable. Kube-Bench will be assessed for the automation of security CIS Benchmark tests in a Kubernetes cluster. Finally, the monitoring system Prometheus and Grafana will be further evaluated as pivotal tools for collecting security-related metrics and logs and feeding the AE (Analytics Engine) and other CHARITY components.

The proposed analytics service takes inspiration from the closed-loop pattern and OODA loop model, as discussed in the ZSM framework, to intelligently enforce the application of network security policies based on network metrics. *Figure 125* depicts the intermediate functions which correspond to the steps of the OODA loop model: Data Collection (Observe), Analytics (Orient), Intelligence (Decide), Orchestration & Control (Act). The first is responsible for observing network traffic from the underlying Cloud-Native environment. The Analytics function allows the detection of anomalies in previously collected and observed data through ML (Machine Learning) mechanisms. The Intelligence function makes decisions for mitigating anomalies (e.g., blocking network traffic with a particular origin) based on input received from Analytics. Finally, the Orchestration & Control function applies the measures decided to mitigate the anomaly, control the communication of microservices, and avoid dangerous situations quickly and efficiently.

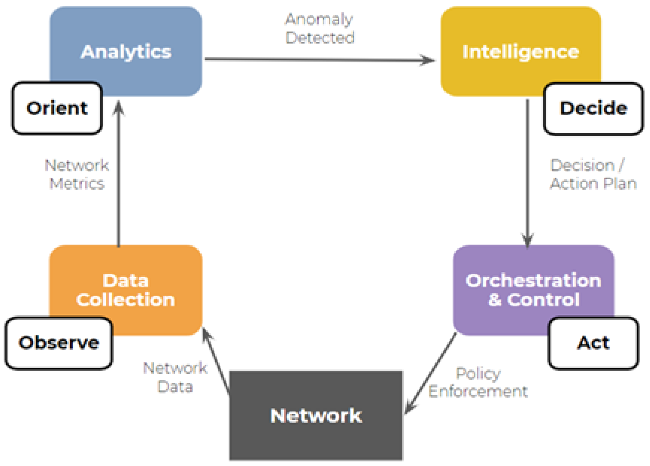


Figure 125: Network anomaly detection and mitigation functions mapped into OODA loop and closed-loop pattern.

OODA model can be applied at a security level to detect network anomalies and implement mitigation mechanisms. Thus, it becomes possible to manage the lifecycle of resources more consistently,

[63] https://www.nfstream.org/

encompassing scenarios from various domains, as in Cloud Native environments. This model can be used together with other specified approaches to automate anomaly detection and mitigation. Through Service Mesh, it is possible to obtain the necessary network observability to detect network anomalies through Machine Learning to mitigate possible cyber threats (e.g., application of network policies at the Service Mesh level).

One possible architecture for this implementation would be the one represented in *Figure 126*. One of the critical components of this architecture is the Collection agent that allows the capture of network traffic for later delivery (as part of the Data Collection function) to the AICO (Analytics Intelligence Control and Orchestration). In turn, AICO process the data and analyse it for possible anomaly detections (as part of the Analytics function), decide the next steps for its mitigation (as part of the Intelligence function) and imposes the necessary changes (as part of the Control & Orchestration function).

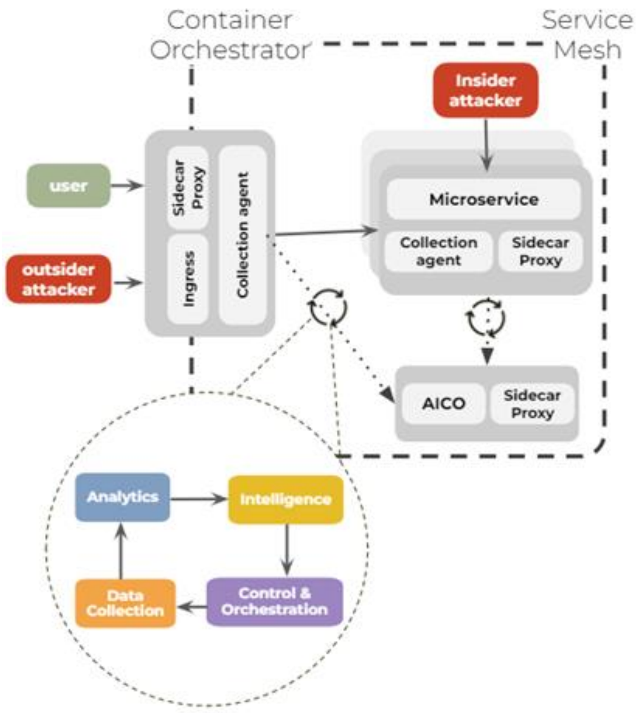


Figure 126: Reference architecture of the proposed framework.

AICO is a component implemented centrally and separately from cloud-native applications, unlike collection agents that must be alongside them. In addition, the edge of the mesh also requires a collection agent o allow protection against external threats. Microservice-level collection agents have only visibility into internal traffic, and as such, they cannot provide adequate security for external traffic. The collection agent at the edge brings observability over north-south traffic.

In this approach, there are some security aspects to consider, such as:

- Separation of concerns - Adopting decoupled closed-loop functions in different components is possible. However, this causes increased network traffic between components, which requires additional resources to support these communications.
- Data Collection - Another possible approach would be redirecting network traffic directly to the AICO component at the fabric level. However, a more complex network configuration would be required. The same case would apply to extending the functionality of sidecar proxies to allow access to traffic data, which requires complex configurations since the metrics reported by sidecars may not even include all the information contained in the network packets. Also, using an additional sidecar proxy for this situation would increase the processing delay. Another alternative could be configuring AICO to use the node's network interface.

However, this violates the isolated nature of containers, which would pose a security risk (e.g., in multi-tenancy scenarios).

- Northbound-southbound traffic - Typically, microservices are exposed only internally, and external traffic goes through a load balancer at the edge of the mesh, which presents a single input. However, when network packets go through ingress, the source IP address is replaced by the ingress address, which makes microservice-level protection from external traffic impossible. The collection agent placed on the ingress makes it possible to capture these packets and preserve the source IPs.
- Analytics function must allow the detection of anomalies comprehensively enough to deal with the vulnerabilities of each service, which may mean handling multiple algorithms. One algorithm may not be able to detect all anomalies, but multiple algorithms increase the detection rate.

An experimental case was carried out in the context of CHARITY research activities to test this approach, where Kubernetes was used to implement the infrastructure to support the Cloud-Native environment, Istio and Envoy proxies for the use of the Service Mesh concept. In that regard, a publication entitled "Cloud-Native Intelligent Anomaly Detection and Mitigation" is under preparation. Istio's out-of-the-box security mechanisms (authorization, authentication, and certificate management) do not broadly cover security vulnerabilities in modern Cloud-Native environments. As such, OPA was also used alongside Istio to ensure enhanced security. OPA offers a more remarkable set of features (e.g., admission control and container security), which significantly increases the level of protection of the environment.

The reference application under protection used in the experiments includes an MQTT message broker component, handling sensitive data over the North-South (with the external IoT devices) and East-West (between internal components) network traffic. This component poses a significant security risk as an attack against such a service can compromise communication between IoT devices and lead to sensitive information leakages. *Figure 127* depicts the practical use case scenario.

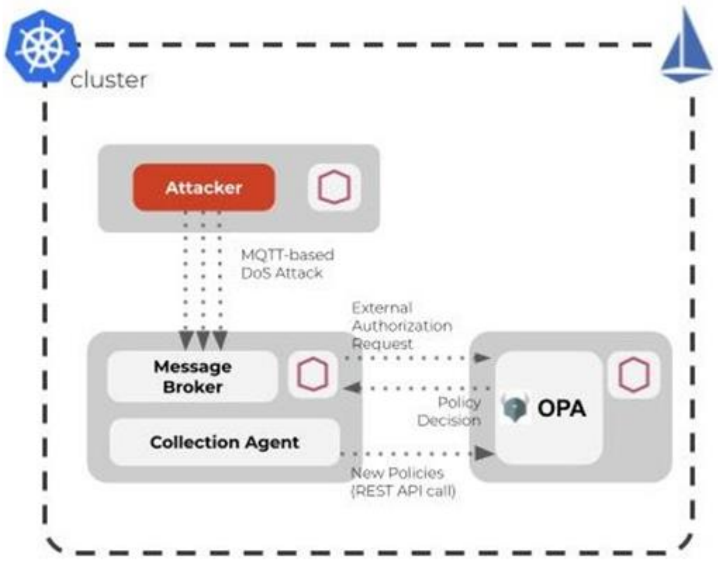


Figure 127: Experimental Use Case scenario.

A DoS attack against the message broker was generated to evaluate the architecture's behaviour and reaction. *Figure 128* shows the evolution of TCP connection to the message broker. The Collection agent captured and stored the network packets, which were later submitted to a pre-trained ML model (using Random-Forest). Through the analysis, it is possible to see that an anomaly was detected. A blocking policy for the offending IP address was registered to mitigate the anomaly using the OPA REST API. Subsequently, it was verified that the traffic was blocked since incoming connections from the respective IP address started to be rejected.

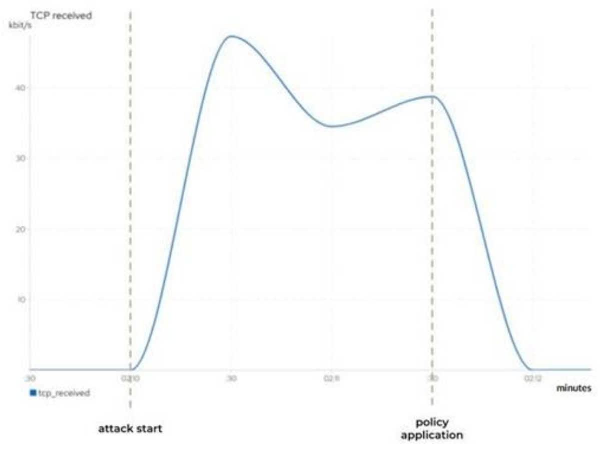


Figure 128: Evolution of TCP connections registered in the message broker.

Using the discussed open-source enablers made it possible to conclude that threats can be effectively stopped with the proposed approach.

9.3 Security Aware Orchestration

Orchestration and scheduling of XR services form the core of the CHARITY architecture. Thus, it is important to enclose security for end-to-end service delivery. Apart from the default security of the application (Section 7.1), we plan to achieve secured orchestration. We assume the application is built of modular tasks, and the tasks are composed of several functions.

9.3.1 Microservice Security

Tasks are deployed as microservices, and an important step is to see whether the containers are secure. Adversaries with access to a container (running inside a host that is part of a large cluster) might be able to exploit vulnerabilities at different layers to conduct powerful attacks (e.g., Remote Code Execution (RCE) attacks). Among those vulnerabilities, those that target the OS kernel of the host where the (malicious) container resides are the most dangerous. Adversaries who send maliciously crafted syscalls can trigger memory leaks, write arbitrary contents into shared files with the host or gain elevated privileges in the host (among others). One prominent example is the weakness found in the *waitid* system call, which allowed adversaries to run a privilege escalation attack to gain access to the host. The fundamental reason why this type of attack exists is that containers have too many capabilities enabled by default. Several Linux kernel security modules, such as AppArmor and SELinux, have been released to restrict the capabilities of containers. Yet, none of these modules tackles the fundamental question of how to minimise the containers' capabilities.

To tackle this, we are designing a tool that automatically finds the minimum set of capabilities that containers need for executing their applications correctly while minimising their interactions with the OS kernel. We hypothesise that the fewer capabilities containers have, the harder it will be for adversaries to carry out attacks against the host's OS kernel. Specifically, the proposed tool would:

- Be sufficiently generic that it can be applied to any type of application or container (i.e., not only Docker).
- Accurately and efficiently characterise the syscall patterns of containerised applications.
- Satisfy the previous two requirements while preventing cloud providers from inferring any information about applications from their syscall patterns.

We will integrate this tool with the orchestrator for both static and dynamic analysis of the microservices while executing the task. This would enable the CHARITY to be much more secure and pave the way for secured multi-tenant deployments.

9.3.2 Secured Orchestration and Scheduling

The second most important thing is to secure communication between the learning module (section 3.2) and the application. We achieve this by enabling network-topology obfuscation using EqualNet.

Distributed Denial of Service (DDoS) attacks constitute one of the Internet's major threats today. One prominent example is Link Flooding Attacks, which aim to disrupt the network connectivity of as many users as possible by congesting network links. More concretely, adversaries aim to inject a large number of flows so that they all traverse a set of core network links at once (to overload them). Noticeably, adversaries can use low-volume, separate flows that are indistinguishable from regular traffic, making it difficult for network operators to develop defences to protect against Link Flooding Attacks (LFAs).

According to Meier et al. [92], performing LFA against an arbitrary link without any knowledge of the network topology requires five times more flows than when the adversary possesses this information. Equally, the number of flows needed to perform an LFA against a target link is higher when the topology is unknown. Indeed, knowledge of the network topology is an important prerequisite for executing such attacks effectively, efficiently and stealthily. This has important implications for adversaries, as their goal is always to cause significant damage while minimising the cost of their attacks (i.e., the number of flows they have to create) and the chances of being detected.

Intuitively, one could think that keeping the network topology confidential would be an effective mechanism to increase the cost of performing successful LFAs. Note that this type of defence would align well with today's Internet Service Providers (ISPs) (as they regard their network topologies as confidential). Unfortunately, researchers have demonstrated that existing path-tracing tools (e.g., traceroute) can be used "maliciously" to infer previously unknown ISPs' network topologies, including their forwarding behaviour and tracing flow distributions (i.e., the number of traceroute flows received by each router's interface). Hence, it becomes apparent that adversaries will apply these techniques to carry out more efficient and effective LFAs.

Over the last few years, researchers have proposed several proactive countermeasures against LFAs which mitigate such attacks by exposing a virtual (false) network topology that conceals potential bottleneck links and nodes while in some cases also attempting to maintain the utility of the information provided by path tracing tools. We began by analysing three state-of-the-art proactive network obfuscation defences, namely NetHide [92], Trassare et al.'s solution [93] and LinkBait [94]. This resulted in the identification of four common weaknesses which can be used to significantly lower the security and utility of the virtual topologies they expose.

Motivated by the weaknesses we found in previous work, we proposed and implemented EqualNet, a secure and practical proactive defence for long-term network topology obfuscation that alleviates LFAs within a single network domain. The fundamental idea behind EqualNet is to equalise tracing flow distributions over nodes and links so that adversaries cannot distinguish which of them are the most important ones, thus significantly increasing the cost of performing LFAs. Meanwhile, EqualNet preserves subnet information, helping network operators who use path tracing tools to debug their networks. To demonstrate its feasibility, we implemented a full prototype of it using Software-Defined Networking (SDN) and performed extensive evaluations both in software and hardware. Our results show that EqualNet is effective at equalising the tracing flow distributions of small, medium and large networks even when only a small number of routers within the network support SDN. Finally, we proved the security of EqualNet under various attacks.

9.4 Security in the Software XR Application

The utilities made available by CHARITY for XR Application developers are primarily included in the XR Application Management Framework (**Section 5**). Security is conceptually part of that lifecycle. Hence the functionalities described in the current paragraph might be implemented in the AMF.

Nevertheless, we treat them as a specific set of security-related capabilities and give a short overview in this section.

Several open security assurance tools and software packages can potentially be part of a DevSecOps cycle for XR Application Developers. The main security related areas to be included in the CHARITY DevSecOps cycle:

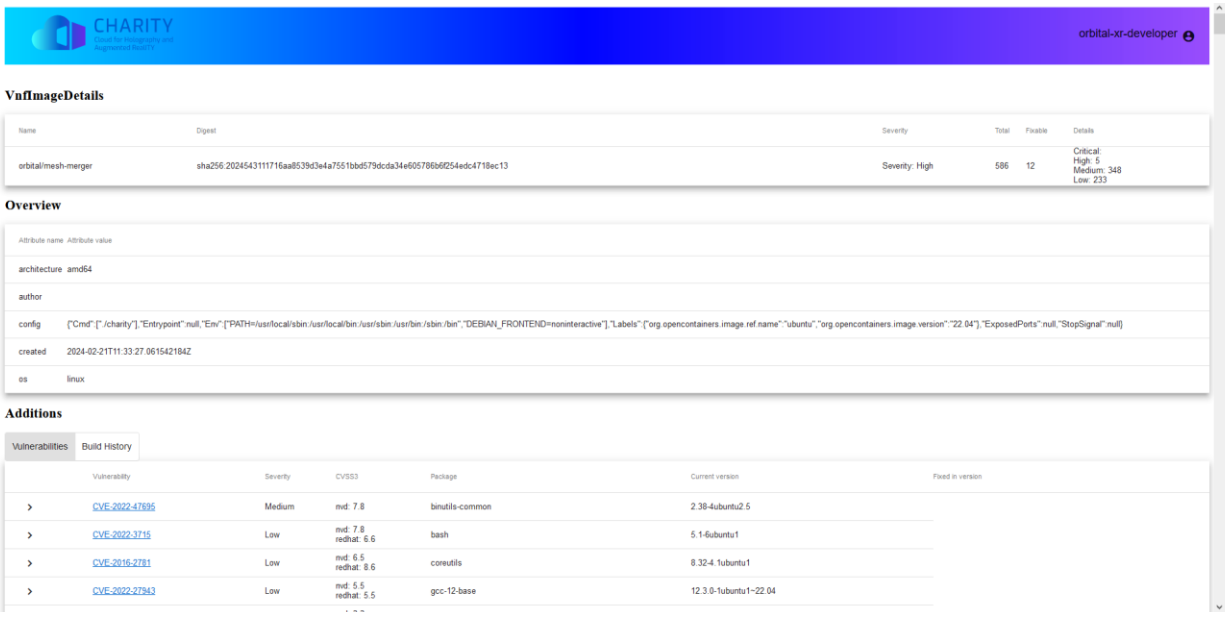
- Static Application Security Testing (SAST):

This step performs a static (application at rest) scan of an application component's source to assess the general code quality and detect potential security vulnerabilities. This technique is limited to the application code and does not seek environment or run time-related vulnerabilities. Issues are detected at the early stages of the software development life cycle, reducing the overall impact and the cost of mitigation. SAST capabilities in CHARITY are available as self-standing services, not fully integrated with the general CI/CD pipeline that is supposed to be fed by wrapped-up images rather than source code, but provided as general project guidelines to CHARITY Platform developers.

- Static container image security testing:

The container images of application microservice components as well as all XR enablers provided by CHARITY project, are statically scanned, searching for known vulnerabilities, typically by parsing through image packages or other dependencies. Each time a developer, through the support of CHARITY AMF Editor WebGUI, uploads a new container image to the CHARITY XR enabler repository (based in Harbor open source container registry[64]), a vulnerability scan is automatically performed (using embedded open source Trivy for vulnerability analysis[65]). The results are collected inside Harbor/Trivy vulnerability repository and exported to the AMF Editor using Harbor REST API, to be made directly available to XR application developers through their usual CHARITY WebGUI.

Figure 129 shows a sample screenshot of the CHARITY AMD Editor web page showing the results of the vulnerability scan of an XR sample enabler (MeshMerger in this case)



VnfImageDetails

Name	Digest	Severity	Total	Fixable	Details
orbital/mesh-merger	sha256:2024543111716aa8539d3e4a7551bbd579dcda34e605786b6f254edc4718ec13	Severity: High	586	12	Critical: High: 5 Medium: 348 Low: 233

Overview

Attribute name	Attribute value
architecture	amd64
author	
config	{"Cmd":["./charity"],"Entrypoint":null,"Env":["PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin","DEBIAN_FRONTEND=noninteractive"],"Labels":{"org.opencontainers.image.ref.name":"ubuntu","org.opencontainers.image.version":"22.04"},"ExposedPorts":null,"StopSignal":null}
created	2024-02-21T11:33:27.061542184Z
os	linux

Additions

Vulnerabilities Build History

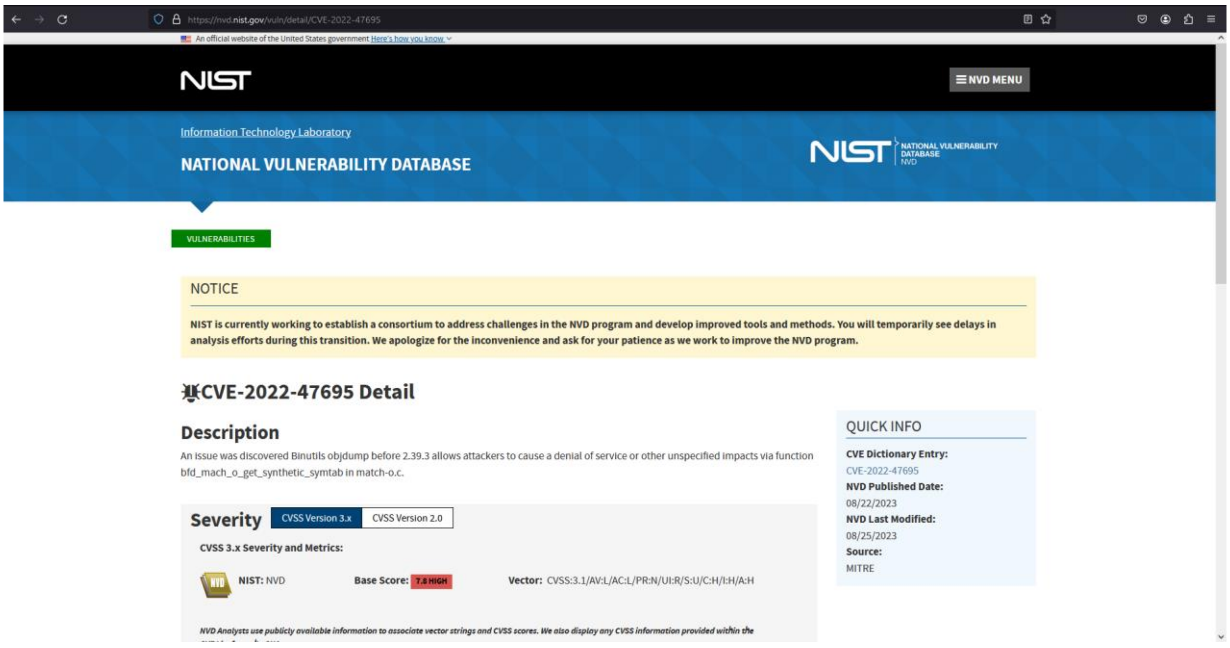
	Vulnerability	Severity	CVSS3	Package	Current version	Fixed in version
›	CVE-2022-47695	Medium	nvd: 7.8	binutils-common	2.38-4ubuntu2.5	
›	CVE-2022-3715	Low	nvd: 7.8 redhat: 6.6	bash	5.1-6ubuntu1	
›	CVE-2016-2781	Low	nvd: 6.5 redhat: 8.6	coreutils	8.32-4.1ubuntu1	
›	CVE-2022-27943	Low	nvd: 5.5 redhat: 5.5	gcc-12-base	12.3.0-1ubuntu1~22.04	

Figure 129: Vulnerability scan example.

[64] Harbor open source container registry: https://goharbor.io/

[65] Trivy open source vulnerability scanner: https://trivy.dev/

In the top right bar the summary counters of the vulnerabilities by severity is reported, while in the bottom part of the window (partially clipped due to its length) each vulnerability is described, with a link to a trusted site containing full details and reference to CWE (Mitre) and NVD (NIST) sites where to retrieve hints on how to fix the issues. *Figure 130* is the screenshot of the NIST web page for the first vulnerability reported by Trivy (CVE-2022-47695) retrieved following the link shown in the previous Web page.


https://nvd.nist.gov/vuln/detail/CVE-2022-47695
An official website of the United States government Here's how you know
NIST
NVD MENU
Information Technology Laboratory
NATIONAL VULNERABILITY DATABASE
NIST NATIONAL VULNERABILITY DATABASE NVD
VULNERABILITIES

NOTICE

NIST is currently working to establish a consortium to address challenges in the NVD program and develop improved tools and methods. You will temporarily see delays in analysis efforts during this transition. We apologize for the inconvenience and ask for your patience as we work to improve the NVD program.

CVE-2022-47695 Detail

Description

An issue was discovered Binutils objdump before 2.39.3 allows attackers to cause a denial of service or other unspecified impacts via function bfd_mach_o_get_synthetic_symtab in match-o.c.

Severity CVSS Version 3.x CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD Base Score: 7.8 HIGH Vector: CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the

QUICK INFO

CVE Dictionary Entry:
CVE-2022-47695
NVD Published Date:
08/22/2023
NVD Last Modified:
08/25/2023
Source:
MITRE

Figure 130: Vulnerability description at Nist site.

Automated static application testing combined with container testing techniques allow the detection of a broad range of different vulnerabilities.

Regarding the security aspects for the Application Management Framework, a centralized Authentication and Authorization mechanism based on the OpenID-connect[66] protocol (an identity layer on top of the OAuth 2.0[67] protocol, the industry-standard for authorization) has been adopted.

The Authentication and Authorization layer ensures that all the access to resources provided by the portal and by the backend components are made by users or services which have proper permissions. For both end users and automated services, a role-based policy is enforced and checked at every request leveraging a centralized component which denies access whenever a condition is not met. Keycloak, supported by Red Hat, is the open-source framework selected and adopted for the AMF. By configuring proper realms and, within them, specific roles, policies and permissions, it is possible to specify fine-grained rules for the endpoints exposed by the system both for external (internet-exposed) endpoints and for internal (also service-to-services) ones.

All the systems and frameworks participating in the AMF ecosystem (the Harbor container registry, Spring Boot microservices, the Jenkins CI/CD orchestration engine) have been integrated with Keycloak and inherit the same access policies defined there.

[66] https://openid.net/connect/

[67] https://oauth.net/2/

9.5 Holistic Security and Privacy Framework

The emergence of new technologies also reflects the emergence of new threats, increasingly more complex and unpredictable. With artificial intelligence methods, it is possible to detect threats and preserve user privacy. Thus, the HSPF framework is introduced, which uses deep learning algorithms to detect threats in network traffic.

9.5.1 HSPF Overview

The Holistic Security and Privacy Framework is composed of three main components: Aggregator, Collector and Agent.

The aggregator corresponds to the main framework Unit, being responsible for coordinating the Federated Training procedure. It performs the management and distribution of the different ML models used for anomaly detection by different federated agents, saving in the database the different training iterations of these models for analysis and comparison purposes.

The Agent component, where the federated agent resides, is responsible for inferring the inbound and outbound traffic and training, accordingly, being the network traffic collected by the Collector. It also is the main trainee of the Federated Training procedure along with other Agent components inserted in applications of the same type which are utilizing the same ML model.

Both components, the Agent and the Collector, are injected as sidecars next to any existing container where the to-be-secured application is executing.

The first step towards the detection of traffic anomalies is to perform the traffic capture itself. As such, the Collector is injected into all the micro-services of the application, with the sole purpose of registering the incoming and outgoing traffic for the respective service. Also called as “side-cars”, their final output is a *pcap* file, containing all the network traffic previously obtained.

Considering all the *pcap* file produced by the sidecars, for each one, a csv formatted file is generated, which is used later on to train AI approaches. Such conversion is made using NFStream tool, aiming to produce datasets with a set of features. As there are a set of features that are not relevant for the identification of traffic anomalies (since their value is not relevant with the traffic characteristics itself and rather represent the origin, endpoint and other metadata of the flow), those have been removed.

With a focus on unsupervised approaches, the implemented classification model is based on an Autoencoder. An Autoencoder is a ML algorithm based on Unsupervised Deep Learning, which is known to present a great balance between classification performances and fast classification times.

For supervised approaches, it was concluded that it is necessary to previously train the algorithms with a set of attacks, which translates into an inability of the algorithm to identify unseen attacks. Also, considering that every time the algorithm needed to be train in runtime, it was necessary to have the dataset of known attacks locally, or send the collected data (from the micro-service communications) to an external place to allow this training. Both options had some limitations. The first presented an issue with storage space availability, and the second a major privacy (and potencial security) issue.

9.5.2 UC Integration Scenario

With this framework and its components in mind, this section presents the integration of the HSPF tool in the context of the CHARITY project. For this purpose, a scenario of integration was built to provide an overview of the capabilities of HSPF in the context of a CHARITY use case, specifically, Use-Case IV - VR Tour Creator. This use case (UC) is composed of five components: cyango-cloud-editor, cyango-backend, cyango-database, cyango-worker, and cyango-story-express.

In this integration scenario, our objective is to deploy the UC IV microservices in a Kubernetes cluster and instantiate the HSPF framework within those microservices. This integration entails deploying the various HSPF components within the UC namespace and configuring them to capture and analyse network traffic for anomaly detection. This configuration is showcased in *Figure 131.*

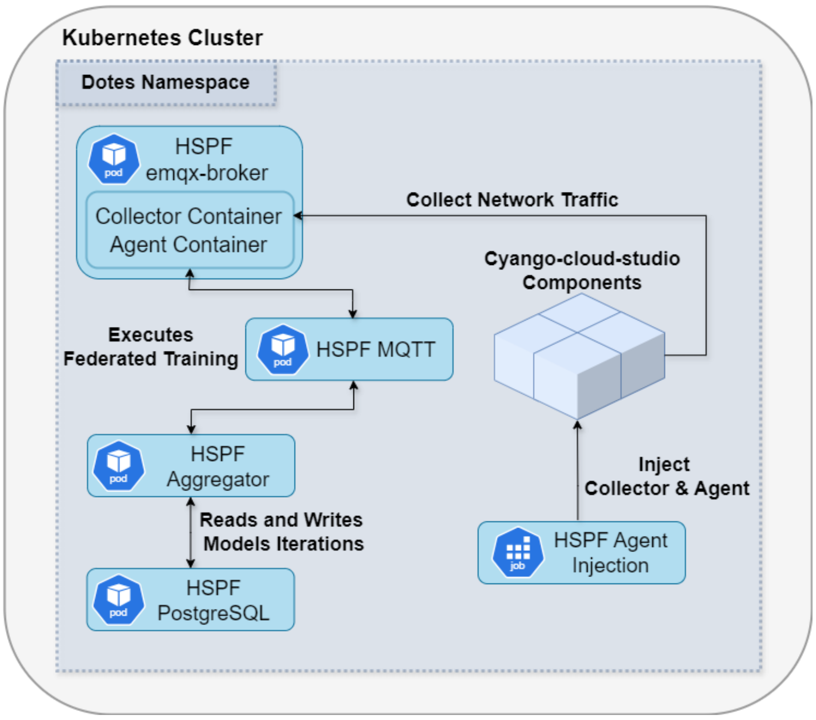


Figure 131: UC Integration Scenario.

Firstly, the deployment of the UC took place, where a dedicated namespace was created in order to hold the components of this application. The microservices were then configured and instantiated in that namespace. Moreover, we deployed the HSPF components within the UC namespace. The HSPF Aggregator component is instantiated to manage and distribute ML models for anomaly detection, alongside the HSPF MQTT broker, to allow the communication between the Aggregator and the Agent, and the HSPF PostgreSQL to enable the storage of Model Training Iterations. Once these components are ready, we deployed the HSPF *emqx-broker* composed of the Agent and the Collector containers which are then injected as sidecars alongside each microservice pod by the HSPF Agent injection (*Figure 132*), being the last component to be deployed. The Agent component is tasked with deducing both inbound and outbound traffic and adjusting training, accordingly, using the traffic data collected by the Collector container. *Figure 133* showcases the initial behaviour of both the Aggregator and the Agent within the Kubernetes environment configured after everything is instantiated. After a certain amount of training the classifier starts to provide high-accuracy anomalies reports, providing reliable information about possible attack flows or anomalies in the network traffic (*Figure 134*).

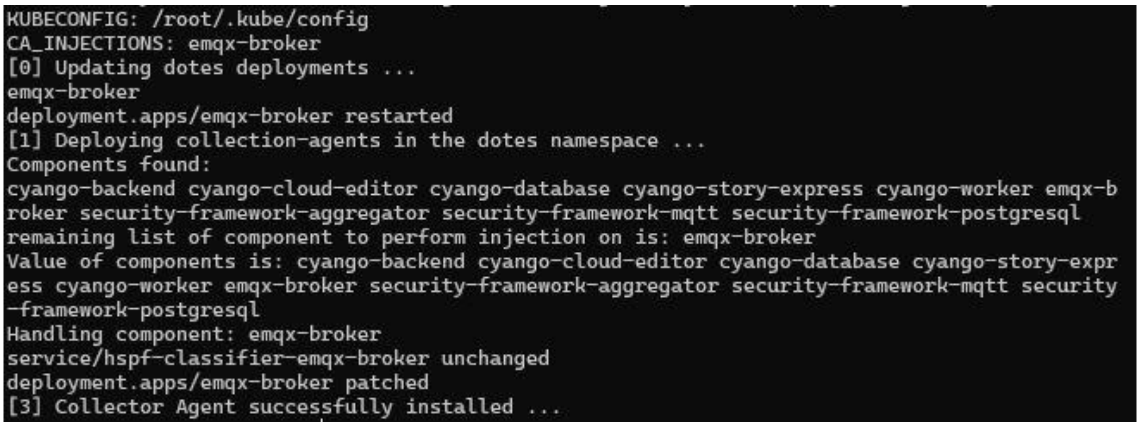
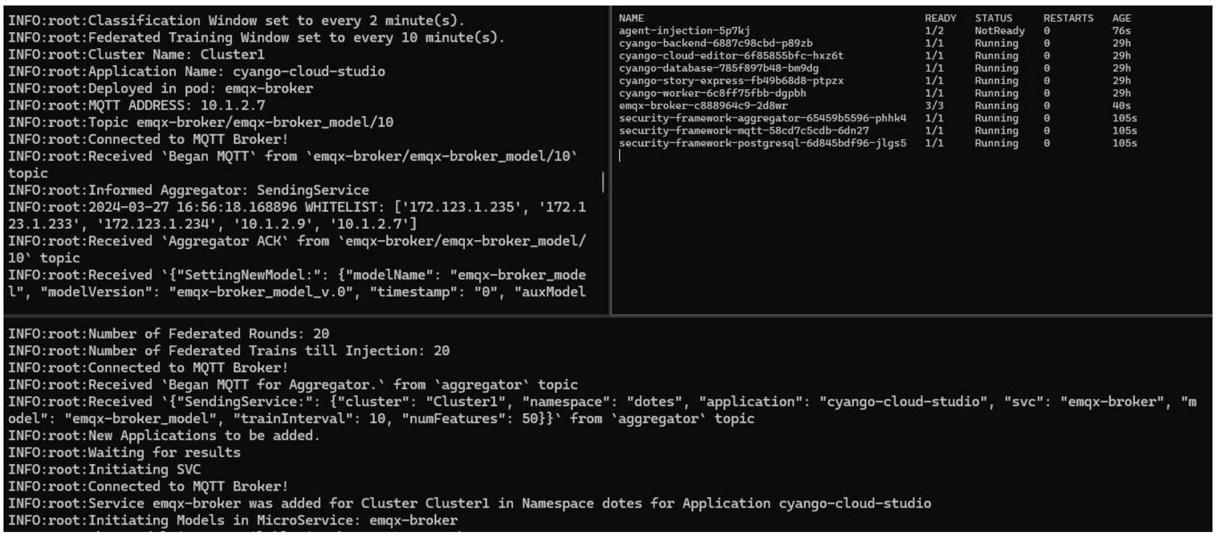

```
KUBECONFIG: /root/.kube/config
CA_INJECTIONS: emqx-broker
[0] Updating dotes deployments ...
emqx-broker
deployment.apps/emqx-broker restarted
[1] Deploying collection-agents in the dotes namespace ...
Components found:
cyango-backend cyango-cloud-editor cyango-database cyango-story-express cyango-worker emqx-b
roker security-framework-aggregator security-framework-mqtt security-framework-postgresql
remaining list of component to perform injection on is: emqx-broker
Value of components is: cyango-backend cyango-cloud-editor cyango-database cyango-story-expr
ess cyango-worker emqx-broker security-framework-aggregator security-framework-mqtt security
-framework-postgresql
Handling component: emqx-broker
service/hspf-classifier-emqx-broker unchanged
deployment.apps/emqx-broker patched
[3] Collector Agent successfully installed ...
```

Figure 132: HSPF Agent Injector – HSPF Agent and Collector injection into UC microservices.

With the HSPF framework instantiated within the UC IV microservices, the stage was set for comprehensive anomaly detection and security monitoring within the CHARITY project. This integration not only showcased the capabilities of HSPF but also demonstrated its potential to enhance the security posture of complex, distributed applications such as the VR Tour Creator.

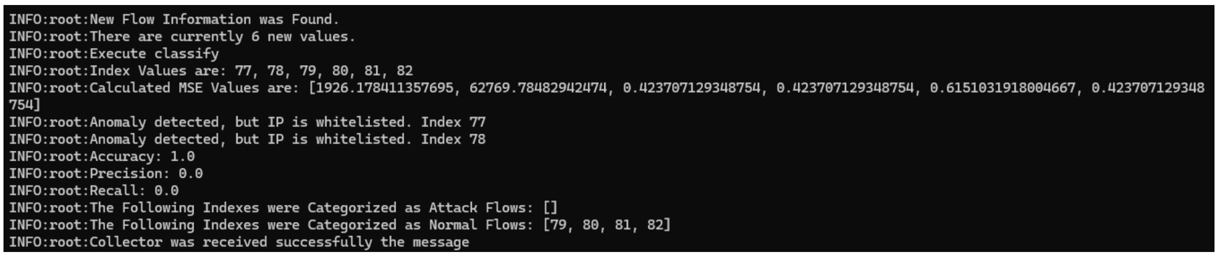


```
INFO:root:Classification Window set to every 2 minute(s).
INFO:root:Federated Training Window set to every 10 minute(s).
INFO:root:Cluster Name: Cluster1
INFO:root:Application Name: cyango-cloud-studio
INFO:root:Deployed in pod: emqx-broker
INFO:root:MQTT ADDRESS: 10.1.2.7
INFO:root:Topic emqx-broker/emqx-broker_model/10
INFO:root:Connected to MQTT Broker!
INFO:root:Received `Began MQTT` from `emqx-broker/emqx-broker_model/10`
topic
INFO:root:Informed Aggregator: SendingService
INFO:root:2024-03-27 16:56:18.168896 WHITELIST: ['172.123.1.235', '172.1
23.1.233', '172.123.1.234', '10.1.2.9', '10.1.2.7']
INFO:root:Received `Aggregator ACK` from `emqx-broker/emqx-broker_model/
10` topic
INFO:root:Received `{"SettingNewModel:": {"modelName": "emqx-broker_mode
l", "modelVersion": "emqx-broker_model_v.0", "timestamp": "0", "auxModel
```

```
NAME                                           READY   STATUS      RESTARTS   AGE
agent-injection-5p7kj                          1/2     NotReady    0          76s
cyango-backend-6887c98cbd-p89zb                1/1     Running     0          29h
cyango-cloud-editor-6f85855bfc-hxz6t           1/1     Running     0          29h
cyango-database-785f897b48-bm9dg               1/1     Running     0          29h
cyango-story-express-fb49b68d8-ptpzx           1/1     Running     0          29h
cyango-worker-6c8ff75fbb-dgpbh                 1/1     Running     0          29h
emqx-broker-c888964c9-2d8wr                    3/3     Running     0          40s
security-framework-aggregator-65459b5596-phhk4 1/1     Running     0          105s
security-framework-mqtt-58cd7c5cdb-6dn27       1/1     Running     0          105s
security-framework-postgresql-6d845bdf96-jlgs5 1/1     Running     0          105s
```

```
INFO:root:Number of Federated Rounds: 20
INFO:root:Number of Federated Trains till Injection: 20
INFO:root:Connected to MQTT Broker!
INFO:root:Received `Began MQTT for Aggregator.` from `aggregator` topic
INFO:root:Received `{"SendingService:": {"cluster": "Cluster1", "namespace": "dotes", "application": "cyango-cloud-studio", "svc": "emqx-broker", "m
odel": "emqx-broker_model", "trainInterval": 10, "numFeatures": 50}}` from `aggregator` topic
INFO:root:New Applications to be added.
INFO:root:Waiting for results
INFO:root:Initiating SVC
INFO:root:Connected to MQTT Broker!
INFO:root:Service emqx-broker was added for Cluster Cluster1 in Namespace dotes for Application cyango-cloud-studio
INFO:root:Initiating Models in MicroService: emqx-broker
```

Figure 133: HSPF Agent and Classifier Initial Behaviour.



```
INFO:root:New Flow Information was Found.
INFO:root:There are currently 6 new values.
INFO:root:Execute classify
INFO:root:Index Values are: 77, 78, 79, 80, 81, 82
INFO:root:Calculated MSE Values are: [1926.178411357695, 62769.78482942474, 0.423707129348754, 0.423707129348754, 0.6151031918004667, 0.423707129348
754]
INFO:root:Anomaly detected, but IP is whitelisted. Index 77
INFO:root:Anomaly detected, but IP is whitelisted. Index 78
INFO:root:Accuracy: 1.0
INFO:root:Precision: 0.0
INFO:root:Recall: 0.0
INFO:root:The Following Indexes were Categorized as Attack Flows: []
INFO:root:The Following Indexes were Categorized as Normal Flows: [79, 80, 81, 82]
INFO:root:Collector was received successfully the message
```

Figure 134: HSPF Classifier Anomaly Detection.

10 Conclusions

This document introduced a summary of the different activities carried out in the four tasks of WP2. It introduced the CHARITY platform along with the evaluation of its components, particularly in terms of four fundamental aspects: i) the orchestration framework along with the supporting algorithms and mechanisms; ii) the monitoring framework; iii) the security of the platform and XR services; iv) the interface connecting the CHARITY platform and XR service providers.

The design of the orchestration framework has been presented, along with the introduction of the high level orchestration, the low level orchestration, and the supporting algorithms such as service scheduling, dynamic routing, and service migration. The CHARITY framework is designed to combine edge, cloud, and network resources into a single continuum of resources that can be easily used by XR services. Additionally, it oversees the entire lifespan of XR applications. The achievement of key performance indicators (KPIs) of an XR application is accomplished through meticulous placement of its components. The Key Performance Indicators (KPIs) are consistently monitored to maintain the Quality of Experience (QoE) by carefully adjusting the Extended Reality (XR) service to the platform's condition. This can be achieved by dynamically modifying the route of data streams, moving certain components, or enabling the XR services to adjust its own behaviours in coordination with the orchestrator.

The second part, which pertains to the monitoring framework and the prediction methods for computation and networking consumption, serves as the fundamental component that would provide an efficient orchestration framework. Due to the decentralized structure of the CHARITY network, utilizing a centralized monitoring server is impractical. Hence, the monitoring framework is structured in a manner that guarantees the dispersion of monitored data over several locations, while simultaneously upholding the comprehensive observability of the orchestration system. The monitoring framework's architecture was introduced, along with the tools that will be utilized for its implementation. Several prediction mechanisms were presented and assessed for both computing and networking predictions. These techniques utilize innovative neural network models that leverage continuous time-series data to provide accurate predictions. The investigation focused on estimating CPU consumption for computational prediction, with the aim of extending it to include memory prediction. Network prediction techniques focus on determining the traffic load placed on the network. This involves predicting bandwidth utilization and anticipating users' requests or sessions.

A substantial amount of effort was dedicated to investigating the privacy and security issues of operating XR services on Edge/Cloud resources. Therefore, a proposed solution to tackle these difficulties involves implementing an architecture that utilizes service-mesh and Open Policy Agent (OPA). This design extensively utilizes closed control loops to provide autonomous security capabilities. Additionally, several security techniques have been developed for cloud-native settings. A proposed approach aimed to secure containers by restricting them to the essential set of capabilities necessary for proper functionality. Another approach involves the deliberate concealment of the networking infrastructure. The purpose of this approach is to restrict the amount of information an attacker can obtain by scanning the network, while still retaining subnet information for debugging purposes. Additional approaches were suggested to provide Authentication and Authorization to XR providers, to do static application and container image security testing, and so forth.

The final part pertains to the application administration framework, which involves the development of a frontend application. This application allows XR service enablers to connect to the CHARITY platform and install their services. The AMF incorporates several key components of the CHARITY platform architecture, including the XR Service Enabler repository, XR Service Blueprint Template Repository, and XR service exposure component.

References

[1] Z. Li, A. Aaron, I. Katsavounidis, A. Moorthy and M. Manohara, “Toward A Practical Perceptual Video Quality Metric,” Netflix Technology Blog, 6 June 2016. [Online]. Available: https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652. [Accessed 15 October 2021].

[2] K. Fatema, V. Emeakaroha, P. Healy, J. Morrison and T. Lynn, “A survey of Cloud monitoring tools: Taxonomy, capabilities and objectives,” Journal of Parallel and Distributed Computing, vol. 74, no. 10, pp. 2918-2933, 2014.

[3] L. Toka, G. Dobreff, D. Haja and M. Szalay, “Predicting Cloud-Native Application Failures Based on Monitoring Data of Cloud Infrastructure,” in 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2021.

[4] D. Berman, “Kubernetes Monitoring: Best Practices, Methods, and Existing Solutions,” logz.io, 19 March 2020. [Online]. Available: https://logz.io/blog/kubernetes-monitoring/. [Accessed 15 October 2021].

[5] D. Berman, “Top 11 Open Source Monitoring Tools for Kubernetes,” logz.io, 4 October 2019. [Online]. Available: https://logz.io/blog/open-source-monitoring-tools-for-kubernetes/. [Accessed 15 October 2021].

[6] E. Kim, J. Han and J. Kim, “Visualizing Cloud-Native AI+ X Applications employing Service Mesh,” in 2020 International Conference on Information and Communication Technology Convergence (ICTC), 2020.

[7] A. Gupta, “Elasticsearch on Kubernetes: A new chapter begins,” elastic, 20 May 2019. [Online]. Available:https://www.elastic.co/blog/introducing-elastic-cloud-on-kubernetes-the-elasticsearch-operator-and-beyond. [Accessed 15 October 2021].

[8] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini and H. Flinck, “Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions,” IEEE Communications Surveys & Tutorials, vol. 20, no. 3, p. 2429–2453, 2018.

[9] M. Masdari and A. Khoshnevis, “A survey and classification of the workload forecasting methods in cloud computing,” Cluster Computing, vol. 23, p. 2399–2424, 2020.

[10] S. Li, Y. Wang, X. Qiu, D. Wang and L. Wang, “A workload prediction-based multi-VM provisioning mechanism in cloud computing,” in 15th Asia-Pacific Network Operations and Management Symposium (APNOMS), Hiroshima, Japan, 2013.

[11] R. N. Calheiros, E. Masoumi, R. Ranjan and R. Buyya, “Workload prediction using ARIMA model and its impact on cloud applications’ QoS,” IEEE Transactions on Cloud Computing, vol. 3, no. 4, pp. 449-458, 2015.

[12] Y. S. Patel and R. Misra, “Performance comparison of deep VM workload prediction approaches for cloud,” in Progress in Computing, Analytics and Networking, 2018, pp. 149-160.

[13] S. Gupta and D. A. Dinesh, “Resource usage prediction of cloud workloads using deep bidirectional long short term memory networks,” in 2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), Bhubaneswar, India, 2017.

[14] C. Peng, Y. Li, Y. Yu, Y. Zhou and S. Du, “Multi-step-ahead Host Load Prediction with GRU Based Encoder-Decoder in Cloud Computing,” in 10th International Conference on Knowledge and Smart Technology (KST), Chiang Mai, Thailand, 2018.

[15] J. Violos, S. Tsanakas, T. Theodoropoulos, A. Leivadeas, K. Tserpes and T. Varvarigou, “Intelligent Horizontal Autoscaling in Edge Computing Using a Double Tower Neural Network,” Computer Networks, vol. 217, p. 109339, 9 November 2022.

[16] T. Theodoropoulos, A. Makris, I. Kontopoulos, J. Violos, P. Tarkowski, Z. Ledwoń, P. Dazzi, and K. Tserpes, “Graph neural networks for representing multivariate resource usage: A multiplayer mobile gaming case-study,” International Journal of Information Management Data Insights, Volume 3, Issue 1, 2023, 100158.

[17] V. Paxson and S. Floyd, “Wide-area traffic: the failure of Poisson modeling,” in SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications, London, United Kingdom, 1994.

[18] V. Eramo, T. Catena, F. Lavacca and F. di Giorgio, “Study and Investigation of SARIMA-based Traffic Prediction Models for the Resource Allocation in NFV networks with Elastic Optical Interconnection,” in 22nd International Conference on Transparent Optical Networks (ICTON), Bari, Italy, 2020.

[19] P. Sekwatlakwatla, M. Mphahlele and T. Zuva, “Traffic flow prediction in cloud computing,” in 2016 International Conference on Advances in Computing and Communication Engineering (ICACCE), Durban, South Africa, 2016.

[20] X. Cao, Y. Zhong, Y. Zhou, J. Wang, C. Zhu and W. Zhang, “Interactive Temporal Recurrent Convolution Network for Traffic Prediction in Data Centers,” IEEE Access, vol. 6, pp. 5276-5289, 2018.

[21] F. Pilka and M. Oravec, “Multi-step ahead prediction using neural networks,” in Proceedings ELMAR-2011, Zadar, Croatia, 2011.

[22] A. R. Abdellah, O. A. K. Mahmood, A. Paramonov and A. Koucheryavy, “IoT traffic prediction using multi-step ahead prediction with neural network,” in 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Dublin, Ireland, 2019.

[23] T. Theodoropoulos, A.-C. Maroudis, J. Violos and K. Tserpes, “An Encoder-Decoder Deep Learning Approach for Multistep Service Traffic Prediction,” in IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService), Oxford, United Kingdom, 2021.

[24] A. -C. Maroudis, T. Theodoropoulos, J. Violos, A. Leivadeas and K. Tserpes, "Leveraging Graph Neural Networks for SLA Violation Prediction in Cloud Computing," in IEEE Transactions on Network and Service Management, vol. 21, no. 1, pp. 605-620, Feb. 2024.

[25] T. Theodoropoulos et al., “GNOSIS: Proactive Image Placement Using Graph Neural Networks & Deep Reinforcement Learning,” 2023 IEEE 16th International Conference on Cloud Computing (CLOUD), Chicago, IL, USA, 2023, pp. 120-128.

[26] T. Theodoropoulos, A. Maroudis, A. Makris, and K. Tserpes, “WEST GCN-LSTM: Weighted Stacked Spatio-Temporal Graph Neural Networks for Regional Traffic Forecasting,” available on: https://arxiv.org/abs/2405.00570.

[27] H. Yu, T. Taleb and J. Zhang, “Deterministic Latency/Jitter-aware Service Function Chaining over Beyond 5G Edge Fabric,” IEEE Transactions on Network and Service Management, 2022.

[28] F. Salaht, F. Desprez and A. Lebre, “An Overview of Service Placement Problem in Fog and Edge Computing,” ACM Computing Surveys, vol. 53, no. 3, pp. 1-35, 2020.

[29] G. Z. Santoso, Y.-W. Jung, S.-W. Seok, E. Carlini, P. Dazzi, J. Altmann, J. Violos and J. Marshall, “Dynamic Resource Selection in Cloud Service Broker,” in 2017 International Conference on High-Performance Computing & Simulation (HPCS), Los Alamitos, CA, USA, 2017.

[30] C. Mechalikh, H. Taktak and F. Moussa, “PureEdgeSim: A Simulation Toolkit for Performance Evaluation of Cloud, Fog, and Pure Edge Computing Environments,” in International Conference on High Performance Computing Simulation (HPCS), Dublin, Ireland, 2019.

[31] L. Ferrucci, M. Mordacchini, M. Coppola, E. Carlini, H. Kavalionak and P. Dazzi, “Latency Preserving Self-Optimizing Placement at the Edge,” in Proceedings of the 1st Workshop on Flexible Resource and Application Management on the Edge (FRAME), Virtual Event, Sweden, 2020.

[32] H. Yu, Z. Ming, C. Wang, and T. Taleb, “Network Slice Mobility for 6G Networks by Exploiting User and Network Prediction,” in Proc of IEEE ICC, Rome, Italy, Jun. 2023.

[33] H. Mazandarani, M. Shokrnezhad, T. Taleb, and R. Li “Self-Sustaining Multiple Access with Continual Deep Reinforcement Learning for Dynamic Metaverse Applications,” in Proc. IEEE Int’l Conf. on Metaverse Computing, Networking and Applications (IEEE MetaCom 2023), Kyoto, Japan, Jun. 2023.

[34] C. Wang, B. Jia, H. Yu, X. Li, X. Wang, and T. Taleb, “Deep Reinforcement Learning for Dependency-aware Microservice Deployment in Edge Computing,” in Proc. of IEEE Globecom’22, Rio De Janeiro, Brazil, Dec. 2022.

[35] S. Global, “2021 RESEARCH PLAN – Cloud Price Index,” 451research S&P Global, 2021.

[36] S. Gorlatch, H. Tim and F. Glinka, “Improving QoS in real-time internet applications: from best-effort to Software-Defined Networks,” in International Conference on Computing, Networking and Communications (ICNC), Honolulu, HI, USA, 2014.

[37] H. Z. Jahromi and D. T. Delaney, “An Application Awareness Framework Based on SDN and Machine Learning: Defining the Roadmap and Challenges,” in 10th International Conference on Communication Software and Networks (ICCSN), Chengdu, China, 2018.

[38] Bashari, M., Bagheri, E., Weichang Du, W. Dynamic Software Product Line Engineering: A Reference Framework. International Journal of Software Engineering and Knowledge Engineering, Vol. 27, No. 2 (2017) 191–234.

[39] R. A. Addad, D. L. C. Dutra, T. Taleb and H. Flinck, “Toward Using Reinforcement Learning for Trigger Selection in Network Slice Mobility,” IEEE Journal on Selected Areas in Communications, vol. 39, no. 7, pp. 2241-2253, 2021.

[40] D. Sabella and e. al., “Developing Software for Multi-Access Edge Computing,” ETSI White Paper No. 20, Sophia Antipolis, France, 2019.

[41] H. Feng, Z. Shu, T. Taleb, Y. Wang and Z. Liu, "An Aggressive Migration Strategy for Service Function Chaining in the Core Cloud," in IEEE Transactions on Network and Service Management, vol. 20, no. 2, pp. 2025-2039, June 2023.

[42] R. A. Addad, D. L. C. Dutra, T. Taleb and H. Flinck, “AI-based Network-aware Service Function Chain Migration in 5G and Beyond Networks,” IEEE Transactions on Network and Service Management, vol. 19, no. 1, pp. 472 - 484, 2021.

[43] H. Yu, T. Taleb, K. Samdanis and J. Song, "Toward Supporting Holographic Services Over Deterministic 6G Integrated Terrestrial and Non-Terrestrial Networks," in IEEE Network, vol. 38, no. 1, pp. 262-271, Jan. 2024.

[44] H. Yu, T. Taleb, and J. Zhang, “Deep Reinforcement Learning based Deterministic Routing and Scheduling for Mixed-Criticality Flows,” in IEEE Transactions on Industrial Informatics, Vol. 19, No. 8, Aug. 2023, pp. 8806-8816.

[45] A. Erdal and T. Korkmaz, “Comparison of Routing Algorithms with Static and Dynamic Link Cost in SDN - Extended Version,” in 16th IEEE Annual Consumer Communications & Networking Conference, Las Vegas, NV, USA, 2019.

[46] P. Goransson and C. Black, “Software Defined Networks: A Comprehensive Approach,” Morgan Kaufman, 2014.

[47] R. Fiqih, N. A. Suwastika and M. A. Nugroho, “Equal-cost multipath routing in data center network based on software defined network,” in 6th International Conference on Information and Communication Technology (ICoICT), Bandung, Indonesia, 2018.

[48] S. Hossen, H. Rahman, Al-Mustanjid, A. Shakil Nobin and A. Habib, “Enhancing Quality of Service in SDN based on Multi-path Routing Optimization with DFS,” in 2019 International Conference on Sustainable Technologies for Industry 4.0 (STI), Dhaka, Bangladesh, 2019.

[49] S. Syaifuddin, M. F. Azis and F. Sumadi, “Comparison Analysis of Multipath Routing Implementation in Software Defined Network,” Kinetik Game Technology Information System Computer Network Computing Electronics and Control, vol. 6, no. 2, 2021.

[50] T. Slavica, I. Radusinovic and N. Prasad, “Performance comparison of QoS routing algorithms applicable to large-scale SDN networks,” in 2015-International Conference on Computer as a Tool (EUROCON), 2015.

[51] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp and P. Francois, “A Declarative and Expressive Approach to Control Forwarding Paths in Carrier-Grade Networks,” in ACM SIGCOMM computer communication review 45.4, New York, NY, USA, 2015.

[52] L. Siamak, F. Pakzad and M. Portmann, “SCOR: software-defined constrained optimal routing platform for SDN,” arXiv preprint arXiv:1607.03243, 2016.

[53] N. Farrugia, J. A. Briffa and V. Buttigieg, “An Evolutionary Multipath Routing Algorithm using SDN,” in 9th International Conference on the Network of the Future (NOF), 2018.

[54] López, Jorge and e. al., “Priority Flow Admission and Routing in SDN: Exact and Heuristic Approaches,” in 19th International Symposium on Network Computing and Applications (NCA), 2020.

[55] T. Shreya and e. al., “Ant colony Optimization-based dynamic routing in Software defined networks,” in 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2020.

[56] Y. W. Chuan and S. Yao, "A Multi-path Routing Algorithm based on Ant Colony Optimization in Satellite Network," in IEEE 2nd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE), 2021.

[57] J. Xie and e. al., "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges," IEEE Communications Surveys & Tutorials, vol. 21, no. 1, pp. 393-430, 2018.

[58] A. Abdelhadi, R. Boutaba and G. Pujolle, "NeuRoute: Predictive dynamic routing for software-defined networks," in 13th International Conference on Network and Service Management (CNSM), 2017.

[59] M. K. Awad and e. al., "Machine learning-based multipath routing for software defined networks," Journal of Network and Systems Management, vol. 29, no. 2, pp. 1-30, 2021.

[60] C. Yu and e. al., "DROM: Optimizing the routing in software-defined networks with deep reinforcement learning," IEEE Access, vol. 6, pp. 64533-64539, 2018.

[61] T. P. Lillicrap and e. al., "Continuous control with deep reinforcement learning". Patent U.S. Patent 0 024 643 A1, 2 Feb 2017.

[62] C. Fang, C. Cheng, Z. Tang and C. Li, "Research on Routing Algorithm Based on Reinforcement Learning in SDN," Journal of Physics: Conference Series, vol. 1284, no. 1, p. 012053, 2019.

[63] J. Rischke and e. al, "Qr-sdn: towards reinforcement learning states, actions, and rewards for direct flow routing in software-defined networks," IEEE Access, vol. 8, pp. 174773-174791, 2020.

[64] J. Specht and S. Samii, "Urgency-based scheduler for time-sensitive switched ethernet networks," in 28th Euromicro Conference on Real-Time Systems (ECRTS), Toulouse, France, 2016.

[65] J.-Y. Le Boudec, "A Theory of Traffic Regulators for Deterministic Networks With Application to Interleaved Regulators," IEEE/ACM Transactions on Networking, vol. 26, no. 6, pp. 2721-2733, 2018.

[66] J. Prados-Garzon and T. Taleb, "Asynchronous Time-Sensitive Networking for 5G Backhauling," IEEE Network, vol. 35, no. 2, pp. 144-151, 2021.

[67] J. Prados-Garzon, T. Taleb and M. Bagaa, "Optimization of Flow Allocation in Asynchronous Deterministic 5G Transport Networks by Leveraging Data Analytics," IEEE Transactions on Mobile Computing, 2021.

[68] M. Shokrnezhad, and T. Taleb, "Near-optimal Cloud-Network Integrated Resource Allocation for Latency-Sensitive B5G," in Proc. of IEEE Globecom'22, Rio De Janeiro, Brazil, Dec. 2022.

[69] Y. Chen, Y. Sun, H. Yu, and T. Taleb. "Joint Task and Computing Resource Allocation in Distributed Edge Computing Systems via Multi-Agent Deep Reinforcement Learning." In IEEE Transactions on Network Science and Engineering. (Early Access)

[70] A. Boudi, B. Miloud, P. Pöyhönen, T. Taleb and H. Flinck, "AI-Based Resource Management in Beyond 5G Cloud Native Environment," IEEE Network, vol. 35, no. 2, pp. 128 – 135, 2021.

[71] T. Taleb, A. Boudi, L. Rosa, L. Cordeiro, T. Theodoropoulos, K. Tserpes, P. Dazzi, A. Protopsaltis and R. Li, "Towards Supporting XR Services: Architecture and Enablers," in IEEE Internet of Things Journal, vol. 10, no. 4, pp. 3567-3586, 15 Feb.15, 2023.

[72] J. A. De Guzman, K. Thilakarathna and A. Seneviratne, "Security and Privacy Approaches in Mixed Reality: A Literature Survey," ACM Computing Surveys, vol. 52, no. 6, pp. 1-37, 2020.

[73] ETSI, "Zero-touch network and Service Management (ZSM); General Security Aspects," ETSI, Sophia Antipolis, France, 2021.

[74] C. DeCusatis, P. Liengtiraphan, A. Sager and M. Pinelli, "Implementing Zero Trust Cloud Networks with Transport Access Control and First Packet Authentication," in 2016 IEEE International Conference on Smart Cloud (SmartCloud), New York, NY, USA, 2016.

[75] M. Sanders and C. Yue, "Automated Least Privileges in Cloud-Based Web Services," in Proceedings of the Fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies, San Jose, USA, 2017.

[76] S. Mehraj and M. T. Banday, "Establishing a Zero Trust Strategy in Cloud Computing Environment," in 2020 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2020.

[77] S. Rose, O. Borchert, S. Mitchell and S. Connelly, "NIST Special Publication 800-207 - Zero Trust Architecture," NIST, USA, 2020.

[78] W. Li, Y. Lemieux, J. Gao, Z. Zhao and H. Yanbo, “Service Mesh: Challenges, State of the Art, and Future Research Opportunities,” in 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), San Francisco, CA, USA, 2019.

[79] W. Morgan, “Service mesh: A critical component of the cloud native stack,” Buoyant.io, April 2017. [Online]. Available: https://www.cncf.io/blog/2017/04/26/service-mesh-critical-component-cloud-native-stack/. [Accessed October 2020].

[80] R. Chandramouli and Z. Butcher, “NIST Special Publication 800-204A - Building Secure Microservices-based Applications Using Service-Mesh Architecture,” NIST, USA, 2020.

[81] E. Harlicaj, “Anomaly Detection of Web-Based Attacks in Microservices,” Aalto University, Espoo, Finland, 2021.

[82] G. Baye, F. Hussain, A. Oracevic, R. Hussain and S. Ahsan Kazmi, “API Security in Large Enterprises: Leveraging Machine Learning for Anomaly Detection,” in 2021 International Symposium on Networks, Computers and Communications (ISNCC), Dubai, United Arab Emirates, 2021.

[83] L. Miller, P. Mérindol, A. Gallais and C. Pelsser, “Towards Secure and Leak-Free Workflows Using Microservice Isolation,” in 2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR), Paris, France, 2021.

[84] K. M. Musa, “Evaluating Security-as-a-Service (SECaaS) Measures to Increase the Quality of Cloud Computing,” International Journal of Science and Engineering Applications (IJSEA), vol. 6, no. 12, pp. 350-359, 2017.

[85] M. Baby and I. Memon, “Security-as a-service in Cloud Computing (SecAAS),” International Journal of Computer Science and Information Security, vol. 15, no. 2, 2017.

[86] K. A. Torkura, M. I. H. Sukmana, C. F. and C. Meinel, “Leveraging Cloud Native Design Patterns for Security-as-a-Service Applications,” in 2017 IEEE International Conference on Smart Cloud (SmartCloud), New York, NY, USA, 2017.

[87] C. Benzaïd, T. Taleb and J. Song, "AI-Based Autonomic and Scalable Security Management Architecture for Secure Network Slicing in B5G," in IEEE Network, vol. 36, no. 6, pp. 165-174, November/December 2022.

[88] C. Benzaid, T. Taleb, A. Sami, and O. Hireche, “A Deep Transfer Learning-powered EDoS Detection Mechanism for 5G and Beyond Network Slicing,” in Proc. of IEEE Globecom’23, Kuala Lumpur, Malaysia, Dec. 2023.

[89] C. Benzaid, T. Taleb, A. Sami, and O. Hireche, “FortisEDoS: A Deep Transfer Learning-empowered Economical Denial of Sustainability Detection Framework for Cloud-Native Network Slicing,” in IEEE Transactions on Dependable and Secure Computing.

[90] C. Benzaid, T. Taleb, and J. Song, “AI-based Autonomic & Scalable Security Management Architecture for Secure Network Slicing in B5G,” IEEE Network Magazine, Vol. 36, No. 6, Dec. 2022, pp. 165 - 174.

[91] Javadpour, A., Ja’fari, F., Taleb, T., Zhao, Y., Bin, Y., & Benzaïd, C. (2023). Encryption as a Service for IoT: Opportunities, Challenges and Solutions. IEEE Internet of Things Journal.

[92] M. Roland, P. Tsankov, V. Lenders, L. Vanbever, and M. Vechev. "NetHide: Secure and practical network topology obfuscation." In 27th USENIX Security Symposium (USENIX Security 18), pp. 693-709. 2018.

[93] S. T. Trassare, R. Beverly and D. Alderson, "A Technique for Network Topology Deception," MILCOM 2013 - 2013 IEEE Military Communications Conference, San Diego, CA, USA, 2013, pp. 1795-1800.

[94] X. Ding, F. Xiao, M. Zhou and Z. Wang, “Active Link Obfuscation to Thwart Link-flooding Attacks for Internet of Things,” 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Guangzhou, China, 2020, pp. 217-224.

Appendix A Additional info

A.1 Tosca custom types for CHARITY

```
tosca_definitions_version: tosca_simple_yaml_1_3
description: CHARITY custom types

data_types:

  charity.geolocation:
    # geolocation of Components (both deployed by CHARITY and External ones)
    properties:
      # name of the continent/region
      Region:
        type: string
        required: false
      # name of the country
      Country:
        type: string
        required: false
      # name of the city
      City:
        type: string
        required: false
      # latitude in string
      latitude:
        type: string
        required: false
      # longtitude in string
      longtitude:
        type: string
        required: false
      exact:
        type: boolean
        required: false

# capability types defined to be used by CHARITY
capability_types:
```

```
# gpu describes the gpu of a node and property describes the model of a gpu.
charity.gpu:
  properties:
    # describes the gpu model
    model:
      type: string
      required: false
    # Dedicated describes if the gpu is a different part from the cpu or not.
    dedicated:
      type: boolean
      default: true

# NetworkMetric is used for Virtual Link KPIs suitable for capability comparisons
CHARITY.NetworkMetric:
  properties:
    # Bandwidth for Virtual Links, notice the scalar-unit.bitrate type, e.g., 10 GBs
    bandwidth:
      type: scalar-unit.bitrate
      required: false
    # Latency for Virtual links, notice the scalar-unit.time type, e.g., 10 ms
    latency:
      type: scalar-unit.time
      required: false
    # Jitter for Virtual Links, notice the scalar-unit.time type, e.g., 1 ms
    jitter:
      type: scalar-unit.time
      required: false

# Preliminary definition of Monitoring KPIs
CHARITY.MonitoringMetric:
  properties:
    # The monitor map will contain sets of key/value pairs suitable for capability comparisons
    monitor:
      type: map
      required: false
      entry_schema:
```

```
      type: string



node_types:

  # Node type is being used to describe CHARITY nodes
  CHARITY.Node:
    derived_from: tosca.nodes.Compute
    capabilities:
      # gpu capabilities
      gpu:
        type: charity.gpu
    # other capabilities are inherited from tosca.nodes.Compute
        #host:
          #properties:
            #num_cpus: 1
            #mem_size: 512 MB
            #disk_size: 20 GB
        #os:
          #properties:
            #architecture: x86_64
            #type: linux
            #distribution: centos
            #version: 7.0



  # Component type is being used to describe a component of an application.
  # Components marked as EXTERNAL are not deployed by CHARITY Orchestrator
  CHARITY.Component:
    derived_from: tosca.nodes.SoftwareComponent
    capabilities:
      # this is required to 'bind' a Software component with a Connection Point
      binding:
        type: tosca.capabilities.network.Bindable
      # the image_os field describes the OS inside the image (VM or Container)
      # this information is automatically extracted by the Editor from image metadata
      image_os:
```

```
      type: tosca.capabilities.OperatingSystem
    # preliminary support to express monitoring KPI comparisons
    monitoring:
      type: CHARITY.MonitoringMetric
  properties:
    # with name property developers will have to provide how the component should be uniquely named inside CHARITY template
    name:
      type: string
      required: true
    # with the deployment_unit property developers can describe if their component is a K8s VM or a K8s Pod
    # EXTERNAL means that thes components do not need to be deployed by CHARITY Orchestrator
    deployment_unit:
      type: string
      required: true
      constraints:
        - valid_values: [ EXTERNAL, K8S_VM, K8S_POD]
    # for CHARITY deployed components developers need to indicate the image url
    image:
      type: string
      required: false
    # with geolocation property developer can either specify the desired location for the Component (at edit or input time)
    # or get the orchiestrator assigned location at run-time
    # if 'exact' is true, the deployment will fail if the location can't be matched exactly, otherwise the closest is selected
    geolocation:
      type: charity.geolocation
      required: false
    # Hint for orchestrator to deploy on Edge or Cloud - not imperative
    placement_hint:
      type: string
      required: false
      constraints:
        - valid_values: [ EDGE, CLOUD]
    # with ip property developers can either provide an external ip to the component (at edit or input time)
    # or get the orchestrator assigned ip at run-time
```

```
      ip:
        type: string
        required: false
      # preliminary support for  environment variables (arbitrary key/value pairs)
      environment:
        type: map
        required: false
        entry_schema:
          type: string
    attributes:
      # instance_id is actually an attribute initialized at deployment time
      # the Orchestrator will compose its value starting from template and resource name, adding other unique tokens
      instance_id:
        type: string


  # CHARITY Connection Points inherit both 'binding' and 'link' requirements from tosca.nodes.network.Port
  # Binding will refer the CHARITY.Node, while Link will refer the CHARITY.VirtualLink
  CHARITY.ConnectionPoint:
    derived_from: tosca.nodes.network.Port
    properties:
      # with name property developers will have to provide how the component should be uniquely named inside CHARITY template
      name:
        type: string
        required: true
      # port number (if required)
      port:
        type: PortDef
        required: false
    attributes:
      # instance_id is actually an attribute initialized at deployment time
      # the Orchestrator will compose its value starting from template and resource name, adding other unique tokens
      instance_id:
        type: string
```

```
# CHARITY Virtual links also collect the NetworkMetric KPIs in addition to tosca.nodes.network.Network information
CHARITY.VirtualLink:
  derived_from: tosca.nodes.network.Network
  properties:
    # with name property developers will have to provide how the component should be uniquely named inside CHARITY template
    name:
      type: string
      required: true
  attributes:
    # instance_id is actually an attribute initialized at deployment time
    # the Orchestrator will compose its value starting from template and resource name, adding other unique tokens
    instance_id:
      type: string
  capabilities:
    # Network capabilities for Virtual Link (Bandwidth, Latency, Jitter)
    network:
      type: CHARITY.NetworkMetric
```

[end of document]