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

D2.1: Edge and cloud infrastructure resource and computational continuum orchestration system (preliminary)

Version: v1.0

Deliverable type	R (Document, report)
Dissemination level	PU (Public)
Due date	30/09/2022
Submission date	30/09/2022
Lead editor	Tarik Taleb (ICT-FI)
Authors	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), Claudia Tavares (ONE), Pedro Sa (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, 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 cognitive resources-aware orchestration mechanisms 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

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 reports some of the findings that will enable the implementation of a platform for offering a smooth and efficient life-cycle management of the computational and networking resources as well as the XR services hosted in the platform. These shall steer both the development of the actual platform and also the XR services which are developed by the XR providers. The resulting platform leverages and supports multiple network technologies and concepts such as Artificial Intelligence (AI) techniques for orchestration and XR microservices. The automation is another important aspect that is investigated in this report.

A framework of cognitive resources-aware orchestration is introduced. This would offer a cloud-native framework where XR services can be deployed. This shows how the resources are to be managed in order to continuously satisfy the desired KPIs of the running XR services. Also, this framework can leverage many other AI based algorithms for carrying out different tasks. For instance, a smart scheduling mechanism is presented further below.

Given the nature of the data generated in users' locations, the protection and privacy of such data is of utmost importance. Therefore, a suitable security and privacy framework is also presented and discussed. Similar to the orchestration, the automation of the security aspects of CHARITY is fundamental and is an integral part of the orchestrator. Thus, this report also covers how this security framework leverages concepts such as Zero-Trust and SECaaS (Security as a Service).

The orchestration framework and the security framework need an efficient monitoring service that can continuously monitor the behaviour of the moving parts of the platform and the XR services. In this vein, a monitoring and resource usage prediction platform is introduced. This platform gathers all the important metrics and stores them to be used by intelligent algorithms, such as the prediction mechanisms, and that is in order to draw insights from the data.

The last aspect of this report deals with the interface between the proposed platform and the XR providers and developers. The proposed application management framework allows the XR developers to define the blueprint for their XR services. The blueprints are used by the orchestration framework for the deployment and the life-cycle management of the XR services.

Table of Contents

Executive Summary **3**

Table of Contents **4**

List of Figures **6**

List of Tables **9**

Abbreviations **10**

1 Introduction **13**

2 Cognitive and resource aware orchestration **14**

2.1 Computation resources management 14

2.2 The orchestration framework 15

2.3 CHARITY AI Resource Aware Orchestrator (AIRO) 18

2.4 GPU-based primitives supporting AI-based service placement 28

2.5 Application Resource Management 29

3 Algorithms for service orchestration **34**

3.1 Deterministic networking in service orchestration 34

3.2 Service placement & resource scheduling 35

3.3 Decentralized service replica management 37

3.4 Simulations tools and experiments on cloud resource management 38

4 XR Aware networking **45**

4.1 Dynamic routing 45

4.2 Deterministic-Networking and TSNs 48

5 Monitoring and Prediction **54**

5.1 Resource Monitoring 54

5.2 Computation utilization prediction 63

5.3 Network state prediction 68

6 Service migration **72**

6.1 Service migration triggers 72

6.2 Network aware service-migration 75

7 Security & Privacy **78**

7.1 Security of XR Applications, Zero-Trust and Security as a Service 78

7.2 Cloud Native Security Mechanisms 84

7.3 Security Aware Orchestration 93

7.4 Security in the Software XR Application 94

8 XR Application management framework **96**

8.1 XR Application Management Framework Architecture 99

8.2 XR Application Management Framework Portal 100

8.3 XR Service Enabler Repository 108

8.4 XR Service Blueprint Template Repository 109
8.5 Backend microservices 109
8.6 TOSCA model for blueprint templates 111
8.7 TOSCA translation 114
9 Conclusions 116
References 117
Appendix A Additional info 123
A.1 Tosca custom types for CHARITY 123

List of Figures

Figure 1: General overview of the AIRO framework [1]. 16
Figure 2: Communication system overview [1]. 18
Figure 3: Conceptual representation of Edge Miniclouds, Network and Application models. 19
Figure 4: Internal architecture and external interfaces of the E2E-CCROM/CNROM. 22
Figure 5: Architecture of the XR Service Enabler. 23
Figure 6: Application adaptation scenarios. 31
Figure 7: Application Aware Orchestration. 33
Figure 8: Example of SFC requests in 5G edge networks [17]. 34
Figure 9: Performance evaluation of DET-SFCD and DET-SFCA algorithms [17]. 35
Figure 10: Request scheduling for optimal resource allocation at real time. 36
Figure 11: Adaptive Scheduling of Edge Tasks (ASET) workflow. 36
Figure 12: Percentage of successful queries over time for ML task with users arriving in real-world pattern. 37
Figure 13: Memory consumption per POD [1]. 40
Figure 14: Average pod deployment time of each application [22]. 42
Figure 15: Stages included in deployment time [22]. 42
Figure 16: CPU Usage graph from 12-tier application [22]. 43
Figure 17: Memory Usage graph from 12-tier application [22]. 43
Figure 18: Dynamic multipath routing framework. 48
Figure 19: Transport network management and orchestration architecture [44]. 50
Figure 20: RL for flow allocation and time-sensitive networks optimization [44]. 50
Figure 21: ATS-based Backhaul Network (BN) performance [44]. 51
Figure 22: Main stages of NEPTUNO for computing the optimal configuration of the network and an example illustrating the primary configuration parameters for two 5QIs [45]. 52
Figure 23: Performance of NEPTUNO [45]. 52
Figure 24: Implementation of EFK based on Kubernetes, adapted from. 60
Figure 25: Implementation of Prometheus and Grafana based on Kubernetes adapted from. 60
Figure 26: Pull vs Push mechanism. 61
Figure 27: CHARITY monitoring architecture. 63
Figure 28: Architecture of the proposed Double Tower Neural Network [59]. 64
Figure 29: Leveraging Service Traffic Prediction for Horizontal Scaling of Network Functions [66]. ... 70
Figure 30: Architecture of smart triggers selection for service migration [67]. 73
Figure 31: Triggers for RL agent [67]. 74
Figure 32: Reward history of SMDM agents [67]. 74
Figure 33: Architecture of a network aware service migration orchestrator [69]. 75
Figure 34: Performance evaluation [69]. 76

Figure 35: Mapping security and Privacy approaches to XR applications [71]. 79
Figure 36: Comparison between Perimeter Security Model and Zero Trust Model, adapted from..... 81
Figure 37: Data and Control planes of a Service Mesh. 82
Figure 38: Architecture of envisioned Autonomic and Cognitive Security Management framework [86]. 86
Figure 39: Architecture of the framework with the enablers and tools [86]...................................... 88
Figure 40: Mapping open-source enablers for network security in CHARITY architecture. 89
Figure 41: Network anomaly detection and mitigation functions mapped into OODA loop and closed-loop pattern. 90
Figure 42: Reference architecture of the proposed framework. 91
Figure 43: Experimental Use Case scenario. 92
Figure 44: Evolution of TCP connections registered in the message broker. 93
Figure 45: CHARITY High Level Architecture (as initially envisioned in the description of work)......... 96
Figure 46: Developers' Activities.. 97
Figure 47: Deployment sequence. 98
Figure 48: AMF and network blueprints. 98
Figure 49: Architectural components of the XR Application Management Framework...................... 99
Figure 50: How a Micro frontend architecture enables independent end to end development workflows.. 101
Figure 51: Technologies allowing Micro Frontends in CHARITY. 101
Figure 52: Login page mediated from the openid-connect authentication provider. 102
Figure 53: Landing page for an XR Developer. 102
Figure 54: List of VNF images available.. 103
Figure 55: Add new image form.. 103
Figure 56: Main page for the "NS Blueprint Templates" functionality. 104
Figure 57: Summary view ("Collapsed") of a Blueprint Template definition..................................... 104
Figure 58: Editing of the NS general information. 104
Figure 59: Editing of the NS "External devices" Section. 105
Figure 60: Editing of the VNFs section. 106
Figure 61: Editing of the Virtual Links section.. 107
Figure 62: Detail of the multi-select drop down list for specifying Virtual Link connectivity. 108
Figure 63: Sample list of REST 'GET' operations provided by the VNF image microservice. 110
Figure 64: Endpoints of the Blueprint microservice REST API. 111
Figure 65: Sample TOSCA Blueprint Template header section. 112
Figure 66: Sample representation of a CHARITY Component and a CHARITY Node. 112
Figure 67: Another sample representation of a CHARITY Component and a CHARITY Node............. 113
Figure 68: Representation of an external device. 113
Figure 69: Representation of connection points.. 114

Figure 70: Representation of a virtual link with QoS requirements. 114

List of Tables

Table 1: Overview of monitoring requirements. 55
Table 2: Comparison between monitoring tools. 59
Table 3: Example monitoring metrics for CHARITY use cases 61
Table 4: Experimental results of the proposed Intelligent Autoscaling method [59] 66
Table 5: Experimental results of the proposed IPFT 68
Table 6: Request number [66] 71
Table 7: Transmitted data [66] 71
Table 8: Session duration [66] 71

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 first version of the deliverable 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.

This deliverable is structured as follows. In section 2, the orchestration framework is introduced. The architecture of the orchestration framework is given along with the interplay that happens between the XR service and the orchestration framework. Section 2 introduces some smart closed loops algorithms that composes the CHARITY orchestration framework. They consist of several algorithms such as smart service scheduling and replica management. Section 2 also shows some implemented simulation tools and presents and discusses some conducted experiments on resource management in a cloud infrastructure.

Section 4 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. Monitoring and resource usage prediction define the focus of Section 5. 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 orchestration framework to take pro-active decisions.

Section 6 introduces some 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 migration while considering the status of the network. Section 7 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 8, the application management framework is presented. It defines the main entry point for XR developers to CHARITY platform. In section 9, concluding remarks are drawn.

2 Cognitive and resource aware orchestration

2.1 Computation resources management

With the advancement of mobile applications and the Internet of Things, it is now not uncommon to integrate computation resources into end devices capable of communicating with one another, resulting in a geographically distributed pool of compute, storage and network resources. This type of distributed computing provides the opportunity for us to process data at the edge as opposed to transferring the data to a traditional centralized computing system, processing it there, before transferring the results back to the end devices, which is far from ideal when running latency-critical and real-time applications. There are a number of advantages to processing on the edge of the network with particular regard to scalability and perceived latency. This is where computation resource management becomes crucial to get right, as workloads must be distributed efficiently across the network. However, considering the limited network resources and the increasing computational requirements, processing *everything* locally or at a remote server may not be the most efficient approach.

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. This is discussed further in Section 8.

2.2 The orchestration framework

In current 5G systems, the exhibited scale and heterogeneity of resources and users demands results in further complex systems with a huge space of possible settings. Indeed, in a scenario of task scheduling that has multi-dimensional resource requirements, coupled with a telco-grade system, the scheduling problem would prove to be too complex. Using traditional ad-hoc scheduling mechanisms may underperform in such complex systems. Therefore, ML techniques are being widely adopted to replace heuristic approaches. Due to the large set of parameters, the use of AI techniques is becoming of utmost importance. With ML techniques, it is possible for the scheduling framework to automatically understand both the workload and the environment.

An Artificial Intelligence based Resource aware Orchestration (AIRO) framework in Cloud Native Environment (CNE) is needed. The AIRO framework leverages ZSM concepts, cloud-native approaches, and ML techniques for efficiently managing network and computation resources. This AIRO framework is implemented by deploying a monitoring and management agent alongside the master node at each cluster for creating a single management domain. The latter receives the high-level controls and commands generated from the End-to-End (E2E) management domain using domain integration fabric.

Figure 1 depicts a general overview of the envisioned AIRO framework to create Cognitive Cloud Native. The AIRO framework leverages both ZSM and ML for creating self-orchestrated and self-optimized CNEs that are able to cohabit and adapt according to the network state and industrial verticals' KPIs (e.g., XR services). AIRO mainly consists of two planes, which are *i)* the orchestration and management planes that present E2E service management domain; and *ii)* the management domain that presents the user plane. The management domain consists of a set of clouds and edges, whereby different vertical micro-services and applications run. For instance, K8s and K3s can host services at the cloud and edge level, respectively. While the communication components within the edge and cloud present the internal domain integration fabric, the "Monitoring Bus" and "Command & Control Bus" present external domain integration fabric.

In AIRO framework, there are two resource orchestration and management levels that aim to optimize the deployment of the services. The first level consists of E2E service management domain that has the global vision and information about various verticals, CNE clusters states, network and computational resources, as well as running services. AIRO framework has two scheduling levels; the first for E2E service management domain and the second for CNE cluster.

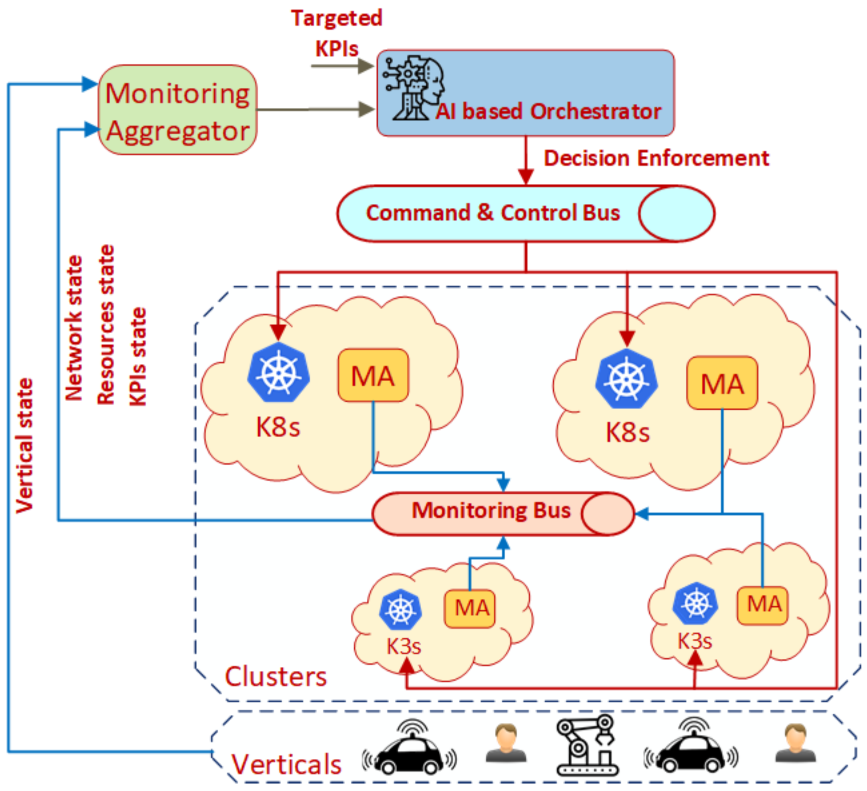


Figure 1: General overview of the AIRO framework [1].

In the first scheduling level, different services and POD intents would be generated and queued up to serve various verticals. Then, according to the vertical state (e.g., user location), clusters states, and network and computational resources, the AI based Orchestrator (i.e., the first scheduling level) decides at which time and on which CNE cluster the PODs and services should be created. In order to take the right decisions, the AI based Orchestrator would employ various AI techniques including deep learning and deep reinforcement learning techniques, such as value-based (e.g., DQN, DDQN, and Duel-DDQN) and policy-based (e.g., A2C, A3C, DDPG, and PPO) approaches. The AI based Orchestrator requires two inputs. The first input consists of the target KPIs per vertical and network slice and the second one consists of the aggregate logging network state that includes information about resources in different clusters and perceived KPIs. Based on the observation that in a closed loop management system, the AI based Orchestrator deals with continuous time-series data, there is a need for data pre-processing that includes data cleaning, integration, smoothing and reduction. For this purpose, the "Monitoring Aggregator" entity has been suggested. In order to remove the noise in this module, different data pre-processing strategies and methods would be applied including dimensionality reduction, discrete Fourier transform, numerosity reduction, and discrete wavelet transform.

In the second scheduling level, the PODs are instantiated at the single management domain (cloud native cluster) from the received POD intents. In each single management domain, there is a Monitoring Agent (MA) that is responsible to gather local information related to network and computation resources and applications, such as QoE, E2E delay and bandwidth, to ensure the desired KPIs. In each cloud-native cluster (i.e., K8s or K3s), the required services (i.e., POD intents) are deployed as PODs and exposed to the outside world. Besides the master node, there is a management agent that exposes REpresentational State Transfer (REST) API and is able to apply different strategies for selecting the appropriate worker node for each POD deployment. Therefore, according to the general guidelines received from the E2E service management domain (i.e., AI based Orchestrator) through the REST API, the cloud native management agent will deploy PODs in various worker nodes to meet the desired KPIs while preventing service overprovisioning and resources underutilization.

The single management domain offers close-loop self-management automation platform to deploy received POD intents. In vanilla K8s and K3s cluster, the PODs are scheduled and deployed in a predefined order and using two cycles. An enhancement is proposed to ensure the close-loop self-management automation by adding two new modules, which are i) Monitoring Agent, and ii) Management Agent. The former gathers logging information within that cluster and reports them to the latter and to "Monitoring Aggregator" modules. Meanwhile, the Management Agent leverages the outputs of the Monitoring Agent for updating the internal strategies of PODs scheduling to reduce and prevent deployment failures and to ensure the satisfaction of desired KPIs. Similar to E2E service management domain, AI techniques including deep learning and deep reinforcement learning techniques would be explored for providing the required strategies. Taking the decisions at the cluster level helps AIRO framework to be a hierarchical, extensible and scalable platform by light-weighting the overhead on the E2E service management domain.

Figure 2 depicts a detailed communication diagram between the two developed modules and vanilla K8s and K3s clusters. With the help of these new communication links, it becomes possible to exploit AI based techniques together with ML to assist the K8s cluster's workload scheduling. In such system, both K8s and external entity have specific well-defined roles based on their interactions that are carried out via the specified Application Programming Interfaces (APIs). These roles and the respective APIs are depicted in Figure 2. The external entity (from the K8s cluster's point of view) called AI/ML logic provides guidance to the K8s scheduler on how to evaluate candidate nodes (workers). This guidance currently consists of mathematical formulas and the related KPIs based on which a rank value for each candidate node is calculated. In this way, the AI/ML logic is continuously maintaining up-to-date guidance information in the K8s scheduler so that at any moment, a new workload scheduling can be done accurately.

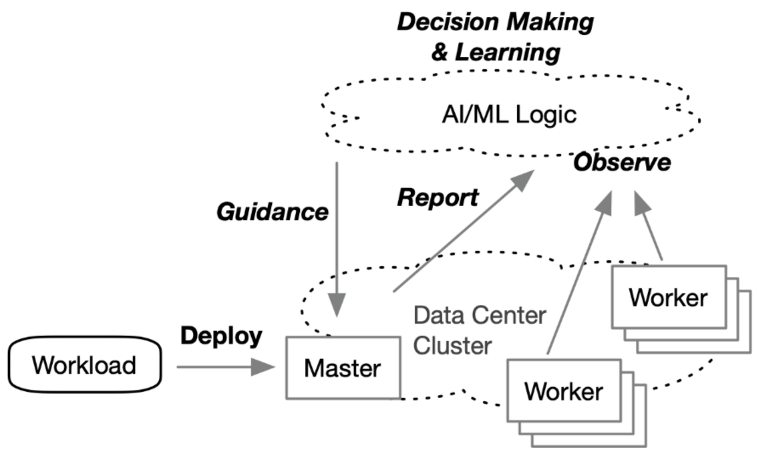


Figure 2: Communication system overview [1].

This approach does not slow down the scheduling cycle by introducing external entities in the scheduling loop, since any communication (synchronous or asynchronous) to fetch external information for the scheduler's decision-making phase may delay the execution of the internal scheduling functionalities. The feedback for the AI/ML logic is achieved in two steps; i) the results of the scheduler's decision making are communicated back (Report) and ii) the cluster's conditions are actively monitored (Observe). This feedback is then used to further refine the AI/ML logic's "knowledge" affecting to the related ML among other things. All this is then used to update the guidance to the scheduler.

Additionally, this kind of system enables a single AI/ML logic to govern multiple clusters that are operating autonomously, i.e., there are no federation or co-operation between clusters. However, even if clusters operate autonomously, certain management actions could be implemented in a distributed fashion resulting in a federated management operation on top of the clusters at the AI/ML logic level. For instance, federated scheduling can be achieved by having an external entity in the scheduling loop, even if the scheduling process becomes slower than the autonomous approach, the advantages of such distributed operation could far outweigh the slower execution times. Indeed, by considering multiple clusters where the scheduling is done by independent Reinforcement Learning (RL) agents, wherein the clusters would be operating autonomously, and by adding an entity that manages the RL agents by modifying their models and policies, we can achieve some level of federation. We can go further and introduce a central entity that would split up the service to be scheduled into a set of sub-services and would subsequently dispatch these sub-services (i.e., micro-services) to the different clusters to be scheduled.

Cluster operators are interested in driving the cluster towards some equilibrium of multiple objectives. These objectives represent operator's incentives and goals, and one of the main incentives is to reduce costs and increase the overall revenue. In what follows, we present some objectives that the operators may chase:

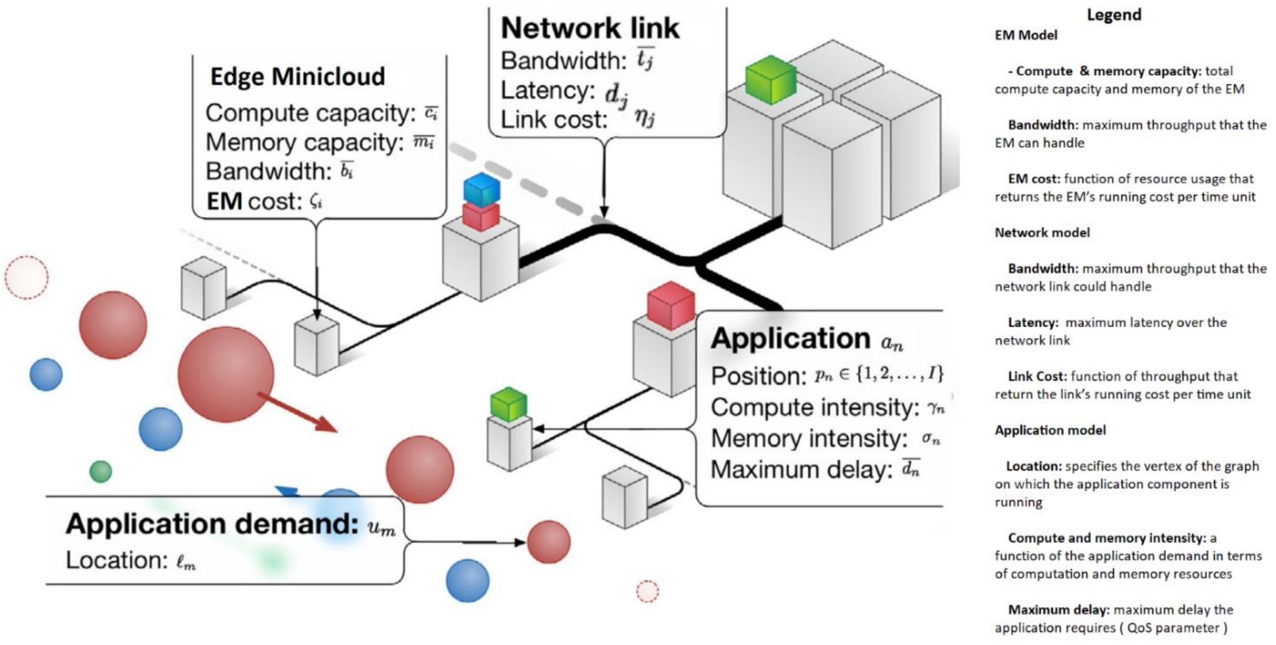
- **Utilization and load distribution among nodes:** Distributing the load among the nodes consists in keeping a low utilization across all nodes. In fact, the Euclidean distance of the percentage of resource utilization among nodes should be reduced as much as possible. This helps in maximizing resource utilization while mitigating and reducing points of congestion. The accurate load distribution ensures a good distribution of network traffic which may result in a positive impact on the QoS and QoE.
- **Reduce operation costs:** In a scenario where nodes have different running costs, it may be useful to shift load from nodes with high running costs to nodes with smaller running costs.
- **Reduce energy consumption:** Likewise, this objective may help the operator in choosing nodes with small energy footprints. It can also clear some nodes from all the PODs so they can be turned off.
- **Service satisfaction:** In this instance, the objective is to maximize the service satisfaction. In other words, regardless of the cluster and node status, the best node would be the one that satisfies all the preferences of the service.
- **Selecting healthy nodes:** We may consider the nodes that have less probability of failure. Such a probability can be calculated from the current load exerted on the node and its history of failures. This objective will help increase the reliability of deployed services.

2.3 CHARITY AI Resource Aware Orchestrator (AIRO)

The CHARITY federation is composed of a "continuum" of resources belonging to different domains or clusters at the edge of the network, closer to users, and one or more public Central Clouds. This continuum enables the convergence of heterogeneous infrastructures, stretching all the way from cloud to edge devices, including intermediate steps such as ISP (Internet Service Provider) gateways, cellular base stations, and private cloud deployments. The CHARITY AI Resource Aware Orchestrator (AIRO) framework is at the heart of the CHARITY platform.

The role of AIRO is to manage all types of resources of the CHARITY Federation, to allow their allocation to the applications (VNFs) running on the Federation. Ultimately, the actual objectives of AIRO are to deliver allocation plans matching application requirements (e.g., in terms of QoE/QoS, etc.) while at the same time achieving an efficient exploitation of these resources. Such, potentially conflicting, objectives are particularly complex to achieve in the highly dynamic, distributed and heterogeneous environment, targeted by the CHARITY project. In order to be properly instrumented to manage such a complexity, AIRO has a compound and hierarchical structure suitable to offer solutions that are tailored to a specific type of resources, while providing a flexible tool enabling the definition and enforcement of high-level management policies across the entire continuum of resources. To reach such objective, the AIRO framework leverages the ZSM concept and Machine Learning (ML) techniques, among others, for efficiently managing network and computation resources.

The AIRO framework consists of two planes, which are the orchestration and management planes, composed of an E2E service management and orchestration plane and a user level management domain plane. The user level management domain consists of a set of clouds and edge domains (called Edge Miniclouds, EM) on which NFV applications are deployed and run. Figure 3 depicts a general overview of it.


Network link
Bandwidth: $\overline{t_j}$
Latency: d_j
Link cost: η_j
Edge Minicloud
Compute capacity: $\overline{c_i}$
Memory capacity: $\overline{m_i}$
Bandwidth: $\overline{b_i}$
EM cost: ς_i
Application a_n
Position: $p_n \in \{1, 2, \ldots, I\}$
Compute intensity: γ_n
Memory intensity: σ_n
Maximum delay: $\overline{d_n}$
Application demand: u_m
Location: ℓ_m

Legend

EM Model

- Compute & memory capacity: total compute capacity and memory of the EM

Bandwidth: maximum throughput that the EM can handle

EM cost: function of resource usage that returns the EM's running cost per time unit

Network model

Bandwidth: maximum throughput that the network link could handle

Latency: maximum latency over the network link

Link Cost: function of throughput that return the link's running cost per time unit

Application model

Location: specifies the vertex of the graph on which the application component is running

Compute and memory intensity: a function of the application demand in terms of computation and memory resources

Maximum delay: maximum delay the application requires (QoS parameter)

Figure 3: Conceptual representation of Edge Miniclouds, Network and Application models.

AIRO takes, as input, applications descriptions, each organized in the form of a Blueprint. The blueprint defines all the entities associated with an application: the modules composing the applications and the links among these modules. In CHARITY, such blueprints are written in the Tosca description language as elaborated in Section 8. To each entity are also associated its own specific requirements, e.g., the requested network bandwidth, the computing capacity, specialized hardware devices, etc. Additional requirements can be also specified in the blueprint, e.g., which part of the application should be executed at the edge, maximum network delay that an application can tolerate. AIRO builds an internal representation of the resources belonging to the CHARITY federation. A definitive formal version of such representation will be defined later in the project. The work conducted so far has been based on a conceptual representation that considers the following items, which is represented graphically in Figure 3:

1. Compute resources:

- **Compute capacity**: total compute capacity of an EM or CC.
- **Memory capacity:** total memory capacity of an EM or CC.
- **Bandwidth**: maximum throughput that an EM or CC can handle.
- **Cost**: function of resource usage that returns the EM or CC running cost per time unit.

2. Network resources:

- **Bandwidth**: maximum throughput that the network link could handle.
- **Latency**: maximum latency over the network link.
- **Link Cost**: function of throughput that returns the link's running cost per time unit.

AIRO achieves its goals finding proper allocation schemas for the application blueprint by leveraging the information collected on the available resources belonging to the CHARITY federation. To this end, AIRO provides specialized approaches and modules that are differentiated on the basis of the resources actually exploited, producing a **CHARITY Compute Resource Orchestrator Module** (CCROM) and a **CHARITY Network Resource Orchestrator Module** (CNROM).

CCROM is the module of the AIRO framework that focuses on the assignment of applications to the computational resources belonging to the CHARITY federation. It defines an internal representation for the federation resources and an application blueprint internal representation; the latter is an internal translation of the Tosca high level blueprint representation. Such representations are needed to approach the problem of allocating application to resources as a **Multi-Resource Multi-Objective Generalized Assignment Problem** (MRMOGAP). To solve the MRMOGAP, various types of algorithms have been investigated to establish a trade-off between the execution complexity of the algorithm and its precision to find a suitable optimized deployment solution. Such approach defines multiple objectives, i.e., reduce latency between application components or with the end-user, reduce resources utilization, reduce costs, etc.

Another function of CCROM is to proactively avoid the degradation of the QoS/QoE levels of an application by taking into consideration a set of indices, which are provided by other CHARITY architecture components, such as the monitoring framework or other Analytical Engines (AEs) built on top of it. Anyway, if the degradation of QoS/QoE cannot be avoided, the CCROM has to cope with the runtime violation of the QoS/QoE of an application or other disrupting events by preparing and executing a set of recovery actions, such as the scaling up or the scaling out by increasing/decreasing the amount or size of instances of components.

The role of CNROM is instead to manage network resources, such as bandwidth, latency, jitter, and network physical channels that also take part in the deployment and runtime of CHARITY applications. It assures that the deployment and runtime of applications do not violate the network requirements of applications. CNROM provides a multi-domain AI-based orchestration framework of the network elements by ensuring reliability and latency, providing automated orchestration and intelligent management operations and facilitating the life cycle management of the network slices with the aim of rapid slice creation and activation, enabling application developers, Use Case owners, to define blueprints for their VR/AR ready network slices.

Furthermore, it relies on monitoring the resources at the edge/public cloud for any potential QoS degradation (e.g., congested links, node capacity excess, etc.) and accordingly plan operations to fix these issues (e.g., service migration, remove congested links, scaling, etc.) and guarantee the network requirements of these applications. These events could be provided through monitoring agents deployed on the Domain-specific XR Service monitoring & Reaction Plane of each specific domain or by the application itself. To enable self-configuration and self-optimization capabilities of network resources, this component also considers the exploitation of machine learning techniques and their integration in the CHARITY framework.

In the remainder of this section, several research challenges related to the actual objectives of these components are addressed. Some of these challenges are common to the two modules (CCROM, CNROM), whereas others are specific to computing or network aspects. Potential solutions to these challenges are devised and introduced in Section 4, Section 5, and Section 6.

2.3.1 CHARITY Computing Resource Orchestrator module for a heterogeneous Cloud/Edge continuum

Cloud computing resources are typically provided through virtualization and there is an illusion of infinite resource availability. Nevertheless, while the cloud can provide vast computing capacity through elasticity, accessing these resources may involve multiple hops of network communication, leading to prohibitive latency in the processing of client requests. In contrast, in edge computing, computational resources are limited and then have to be managed very efficiently. Nevertheless, in edge computing, a client typically accesses resources that are under the same network coverage, be it cellular (e.g., 5G) or local (e.g., domestic or office), allowing to mitigate network latency. So, one of the first research challenges is how to design the architecture of the CCROM such as it could manage such heterogeneity in resources and could be adapted indifferently to EMs and CCs, taking into consideration their different peculiarities, in particular the limited resource capacity of Edge cloud environments.

To maintain a global vision over the entire set of resources of the federation and of the positions of all the application components, the CCROM module is logically viewed as **centralized**, but the potentially large size of a CHARITY federation and the number of resource management parameters render a fully centralized resource allocation strategy practically infeasible. A distributed design for the CCROM will provide some clear advantages, for example:

- **Fault-tolerance:** a decentralized solution is more resistant to failure with respect to a centralized one.
- **Increase scalability:** a centralized solution does not scale in respect to the size of the CHARITY federation.
- **Reduce network congestion:** a centralized solution could be a bottleneck for network traffic.

For these reasons, and similar in spirit to the overall CHARITY architecture (devised in D1.3 as per a thorough study conducted on relevant system architectures and standards), the CCROM module is logically separated in two components: **i)** a **Domain Specific CCROM** (DS- CCROM) component, and **ii)** an **End-to-End CCROM** (E2E- CCROM) component. Of these, the E2E-CCROM component is a centralized component, deployed in a unique instance in a Central Cloud of the federation, i.e., on the XR Service E2E Conducting Plane of the CHARITY architecture. Figure 4 illustrates the internal architecture of such component, which is integrated with the E2E-CNROM (Section 2.3.2), composing a unique module called E2E-IO (E2E Intelligent Orchestrator) and the external interfaces and connection with other CHARITY components.

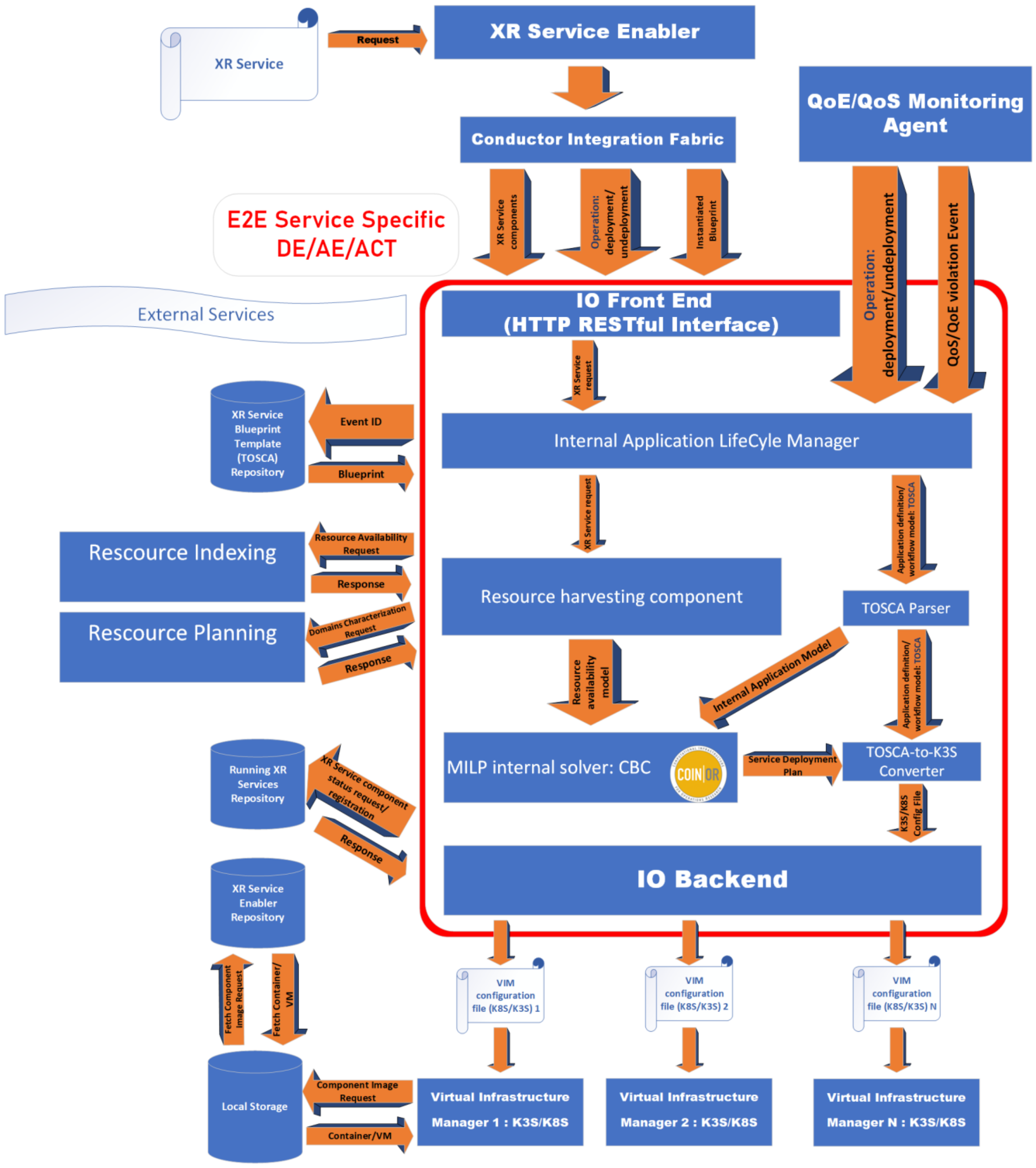

XR Service
Request
XR Service Enabler
Conductor Integration Fabric
QoE/QoS Monitoring Agent
E2E Service Specific DE/AE/ACT
XR Service components
Operation: deployment/ undeployment
Instantiated Blueprint
Operation: deployment/undeployment
QoS/QoE violation Event
External Services
IO Front End (HTTP RESTful Interface)
XR Service request
XR Service Blueprint Template (TOSCA) Repository
Event ID
Blueprint
Internal Application LifeCyle Manager
XR Service request
Application definition/ workflow model: TOSCA
Rescource Indexing
Resource Availability Request
Response
Resource harvesting component
TOSCA Parser
Rescource Planning
Domains Characterization Request
Response
Resource availability model
Internal Application Model
Application definition/ workflow model: TOSCA
Running XR Services Repository
XR Service component status request/ registration
Response
MILP internal solver: CBC
Service Deployment Plan
TOSCA-to-K3S Converter
K3S/K8S Config File
IO Backend
XR Service Enabler Repository
Fetch Component Image Request
Fetch Container/ VM
VIM configuration file (K8S/K3S) 1
VIM configuration file (K8S/K3S) 2
VIM configuration file (K8S/K3S) N
Local Storage
Component Image Request
Container/VM
Virtual Infrastructure Manager 1 : K3S/K8S
Virtual Infrastructure Manager 2 : K3S/K8S
Virtual Infrastructure Manager N : K3S/K8S

Figure 4: Internal architecture and external interfaces of the E2E-CCROM/CNROM.

As depicted in Figure 4, such architecture is composed by the following internal modules:

- E2E-IO Front End: As the name infers, it is the front end of the entire E2E-IO and the unique entry point. It is composed of a HTTP RESTful Interface to receive JSON formatted requests to deploy/un-deploy/scale-up/scale-down/migrate a certain set of XR service components of a Network Service Descriptor (NSD). It is also possible to receive, through the same interface, specific criteria of optimization of the request, such as to deploy the component on a certain geographic location, or to minimize latency or to maximize bandwidth or other network resources between some components connected with a Virtual Link (see the TOSCA specification). Figure 5 shows the front-end part of the entire orchestration process for the invocation of an operation on a VNF: the request from a user/developer of a CHARITY Use Case is received by the XR Service Enabler (XRSE) component which works as a portal for requests. Then, it selects the proper Blueprint Template written in Tosca from the XR Service Blueprint Template Repository component (XRSBTR), instantiating it dynamically on the basis of the needed optimization requests and/or deployment time parameters (i.e., required geographical location, etc…). Then, the instantiating Blueprint is passed, with the request details, to the E2E-IO Front End.

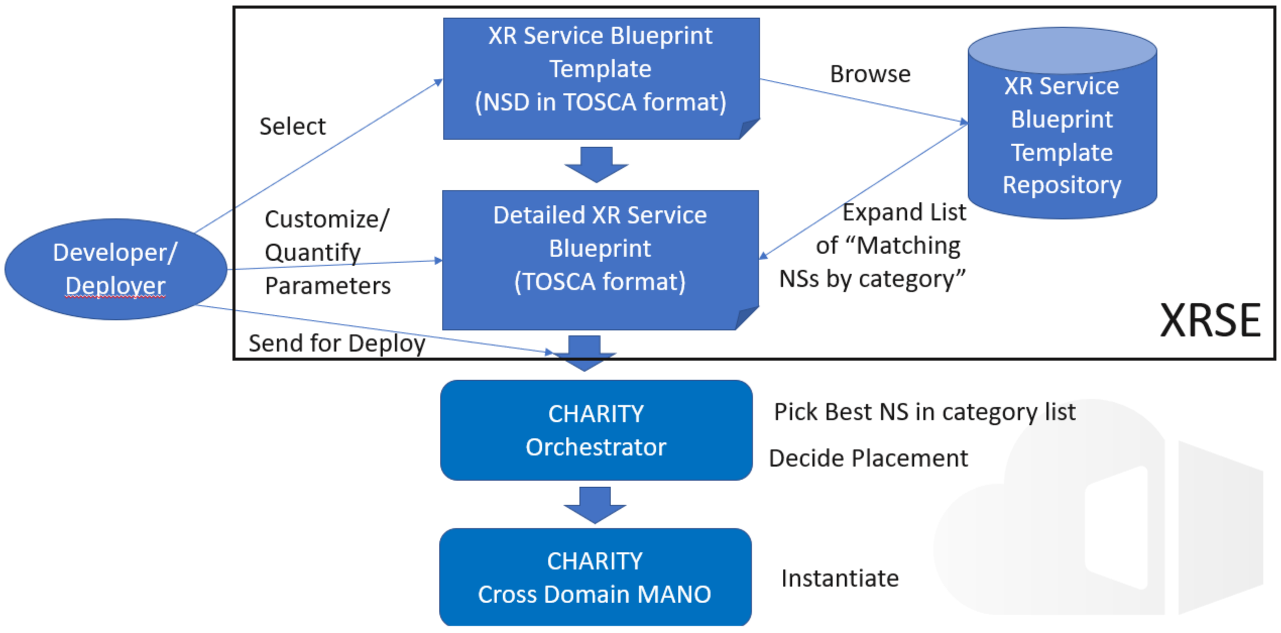

XR Service Blueprint Template (NSD in TOSCA format)
Browse
XR Service Blueprint Template Repository
Select
Customize/ Quantify Parameters
Detailed XR Service Blueprint (TOSCA format)
Expand List of "Matching NSs by category"
Developer/ Deployer
XRSE
Send for Deploy
CHARITY Orchestrator
Pick Best NS in category list
Decide Placement
CHARITY Cross Domain MANO
Instantiate

Figure 5: Architecture of the XR Service Enabler.

- Internal Application Lifecycle Manager: Its role is to analyse the request, parsing it for errors and identifying uniquely the NSD and its components, and eventually other additional components to be deployed/undeployed/migrated if needed by the invoked operation. It exploits an internal TOSCA parser to translate the Blueprint into an internal model to be passed to the MILP (Mixed Integer Linear Programming model) Internal Solver components, validating and detecting errors in the TOSCA specification, and identifying the set of internal actions to be performed to execute the required operation. Usually, the required operation is translated into a set of internal actions, starting an internal iterative process. Furthermore, it receives directly from a QoE/QoS Monitoring Agent/s, deployed on the XR Service Deployment Plane, the QoE/QoS violation notifications for every running VNFs. This event triggers a direct request to the XRSBTR to fetch a Blueprint to instruct the E2E-IO LifeCycle Manager on what are the recovery actions to be performed for that specific event.
- Resource Harvesting component: Its role is to retrieve information about the actual availability of resources over the entire CHARITY federation, to perform the matchmaking between actual availability and the requested resources to perform the required operation on the specified NSD components (when needed, i.e., for the deployment operation). In order to receive information about system resources, it queries the Resources Indexing service (deployed on the XR Service E2E Conducting Plane). Resources Indexing service contains an aggregated information about resources per EM/CC such as number of available CPU nodes/cores, quantity of available RAM, disk storage size, and availability of special resources (i.e., a particular model of GPU or hardware, such as Intel's NUC – Next Unit of Computing). In some cases, Resource Harvesting component queries the Resources Planning service deployed on the XR Service E2E Conducting Plane, for characterization info (i.e., static info such as ID of a

certain EM, number of hosts, maximum quantity of RAM, CPU cores, external IP of the K8S/K3S Masternode, etc).

- MILP Internal Solver: This is the internal solver, which is actually the COIN-OR Branch and Cut (CBC) solver, based on an exact Cplex algorithm. It is used to solve the MRMOGAP problem, to do the matchmaking between the required resources and the available resources on the EMs/CCs. The solver is based on a MILP model. Other mathematical models and related state of the art can be also used as reported in Sections 4, 5 and 6. As inputs, it takes the model (tailored on the operation and the optimizations to be performed) composed by a set of linear constraints and an objective function, which is based on two other internal models: a Resource availability model and an Application model, which is the output of the TOSCA Parser component. In the next developed cycle, it will be assisted by an AI based scheduler to proactively avoid deterioration of QoS/QoE. Its output, in the case of the deployment operation, is a Service Allocation Plan (SAP), whose structure is explained later. Such structure depends on the exact operation performed.
- TOSCA Parser/Converter: The TOSCA Parser's role is to translate the specifications contained in the blueprint to an internal representation, producing two different outputs: (1) the Application model to be submitted to the internal solver, and (2) a series of internal application definitions, which are the input of the second component, the TOSCA-to-K3S Converter. The Converter has the role to translate the application definitions in one or more configuration files defined in the Kubernetes API YAML formatted language, grouped by VIM (Virtual Infrastructure Manager). Each VIM manages locally a different EM/CC. Such files contain a set of Kubernetes objects needed to perform the required operation: deployment objects, service objects to expose IP addresses, Persistent Volume and Persistent Volume Claims object for the storage and so on.
- E2E-IO Back End: Its role is to execute the SAP, identifying the set of domains involved and encapsulating the Kubernetes object related to the same VIM in a single configuration file. Then, it sends such configuration file to each involved VIM through a dedicated message. For operations different from deployment, it could need to query the Running XR Services Repository, to know where running components have been deployed, and their status.

Instead, the DS-CCROM component is a domain specific component which is in charge of performing a matchmaking within the application resources request, its QoS/QoE indicators and the availability of the resources of a specific EM/CC domain. Each DS-CCROM instance is deployed on a certain EM at the Domain-Specific XR Service Monitoring & Reaction Plane. The DS-CCROM component has a full and detailed knowledge of these resources, their availability and their localization. To realize this matchmaking, the DS-CCROM needs the specification of the application blueprint as well as the representation of the resources and their actual availability. Due to the high dynamicity of the environment and the high volatility of the availability of the resources, DS-CCROM queries the Domain Resources Indexing service of CHARITY, deployed on each EM at the Domain-Specific XR Service Monitoring & Reaction Plane, to obtain information about resource availability in an online fashion. The DS-CCROM then approaches the MRMOGAP problem. Indeed, the multi-resource generalized assignment problem is encountered when a set of tasks has to be assigned to a set of agents in a way that permits assignment of multiple tasks to an agent subject to the availability of a set of multiple resources assigned to that agent, as it happens with our orchestrator.

The MRMOGAP could be modelled mathematically in various ways. However, finding its solution can be very complex from a computational perspective, being a well-known NP-hard problem. The models that have a prominent role in literature are:

- The Mixed Integer Linear Programming model (MILP) [2, 3, 4, 5]
- The Mixed Integer Non-Linear Programming model (MINLP) [6]
- Graph models [7, 8].

As described in [2], at the E2E-CCROM level each vertex of the graph is an EM or CC able to host applications using a finite set of resources. Each EM is modelled as a single administrative domain composed by a set of hosts, that can run one or more virtualized resources, which could be VMs or Containers, depending on their availability of resources, but the view of E2E-CCROM over the set of resources of each EM and CC is limited to aggregated values. Examples of such information are:

- Number of CPUs
- Total amount of RAM
- Total amount of Storage
- Presence (or not) of a determined type of a special resource, such as a particular type of GPU.

Furthermore, each CC/EM specifies its cost model to allow an evaluation of the overall cost to deploy an application, and to minimize the use of resources (if specified in the Objective function). We consider providers adopting a per-resource cost model. In the per-resource case, we model EM/CC providers as capable of running each type of VM/Container provided by customers, up to availability of resources limit, and charging them per unit of used resources over time. For this reason, we introduce a set of variables C_{ir}, which is the unit cost for the resource r of the EM/CC associated to the vertex of index i.

The logical network links connecting CCs and EMs are represented as edges of the graph. So far, we assumed undirected edges, but it is also possible to model the directed edge case, by assigning the proper values to the resources in the two directions. Examples of such network resources are:

- Aggregated maximum latency over the link
- Aggregated maximum bandwidth estimations over the link
- An indicator for the level of congestion of the link, in percentage and normalized to 1: LC, $0 \le LC \le 1$
- Other similar indicators.

Also, in this case, to each CC/EM is associated a cost for the communications inter-providers. Such value could be a function of the different cost specified by the providers at the vertices connected by a given edge. The representation graph is updated exploiting the information obtained by the Resource and Indexing service as well as the Monitoring and Network. At the DS-CCROM level, each vertex of the graph represents a single host of an EM/CC, while each edge represents the internal network connections between hosts of the referred EM/CC. The application is logically represented with an **application blueprint** annotated by the set of requested resources, QoS and QoE constraints.

The following are the classes of algorithms used to solve the MRMOGAP service allocation problem which had been investigated so far:

- **Exact algorithms:** these algorithms produce an exact solution to the problem. We will not take them into consideration for CHARITY since they are too expensive in terms of computation complexity.
    - ***CPlex***
- **Approximated algorithms:** these algorithms produce a near-optimal solution with a limited computational effort. This is achieved by exploiting heuristic, random processes or other types of techniques to reduce the available solution space.
    - ***Centralized:***
        - ***Simulated annealing***: the method models the physical process of heating a material and then slowly lowering the temperature to decrease defects, thus minimizing the system energy. At each iteration, a new point in the reference space is randomly generated. The distance of the new point from the previous point is based on a probability distribution that is proportional to the simulated temperature. The temperature progressively decreases from an initial positive value to zero. Then, it measures the quality of the solution, and moves to it according to the temperature-dependent probabilities of selecting better or worse solutions, which during the search respectively remain at 1 (or positive) and decrease towards zero. The algorithm accepts all new points that foster the achievement of the objective, but also, with a given probability,

those points that move away from the objective, avoiding being trapped in local minima.

- **_Iterative local search_** [2]: the method tries to apply a local search routine, starting from a candidate “local” solution, to find a better “neighbour” solution heuristically, exploiting only local information to the set of neighbours. If no solutions improving the current one are found, to avoid being trapped in local minima, the algorithm iteratively applies the local routine from a different local solution, which is decided without taking in consideration the information collected by neighbours.

- ***Distributed:***
  - **_Distributed version of simulated annealing_**
  - **_Voting-based consensus algorithms_** [4]: In this type of algorithms, a set of agents try to find a consensus over a certain decision, applying one of a large set of distributed voting techniques such as the Boyer–Moore majority vote algorithm, which is an exact algorithm, and other approximated algorithms. In case of orchestration, each agent is represented by an application, which starts a voting procedure with the aim of acquiring the resources needed to deploy its assignment vector, participating to a resource election protocol.
  - **_Hierarchical, two-level genetic algorithms_** [6]: A genetic algorithm (GA) is an iterative process, in which each iteration is a generation composed of a population of individuals that represents a candidate solution. Each individual is represented by its own chromosome, which is characterized by a set of variable components known as genes. The quality of an individual in the population is determined by a fitness function, the resulting value specifies how an individual compares to others in the population. By applying genetic operators an old generation can evolve into a new one until convergence is reached. GAs have been applied to service orchestration in Cloud federations before [9]. In two-level genetic algorithms, two GA are nested to solve an assignment problem: the first level applies a distributed GA at the federation level; the second level is executed locally for each EM.
  - **_Graph coloring techniques_** [8]: In this type of algorithm, a distributed version of the well-known Graph Coloring Problem is applied to a bipartite graph, with the two set of vertices constituted by, respectively, the application components and the EM (or the hosting nodes on EM), solving node load conflicts over resources of EM: each feasible coloring represents a set of no conflicting node.

The output of the DS-CCROM is a collection of SAP (Service Allocation Plan): each plan is composed by a vector of couples (c_i, x_j), where $c_i \in A_p$ is the i^{th} component of the application A_p, while x_j is a hosting server belonging to a specific domain.

DS-CCROM selects the best SAP plan, among the ones calculated, during the optimization process: in case a QoS/QoE violation is detected or a disrupting event has occurred, e.g., a security violation, a failure of a hardware component or a risk of a privacy data leakage, DS-CCROM can consider the possibility of executing another SAP, if available.

When a SAP enactment fails, the DS-CCROM could execute one or a set of recovery actions. The QoS/QoE violation or disrupting event may also happen during the lifecycle of the application, i.e., after the deployment has been done with success. In this case, a modification to the executed SAP or the execution of another one could be also triggered and a set of recovery actions could be launched, taking in consideration the failed SAP. Similarly, forecasting techniques to proactively detect failures or QoS/QoE parameters deterioration could be employed applying ML/AI algorithms over a set of indicators provided by the proper CHARITY services; these techniques will be further investigated.

If no SAPs are computed or all the SAPs failed to be executed, the DS-CCROM is in charge to try to find additional resources to satisfy the application deployment request from other groups of remote resources, belonging to other EMs or CCs, managed by a different DS-CCROM.

Another challenge is to design a CCROM that can optimize resources' allocations. Federated Learning (FL) is considered to be a game-changer for an intelligent and efficient resource management. FL is a ML technique that enables the distribution of the learning process among different Edge nodes (e.g., Edge Mini Clouds) for two main reasons, namely 1) to ensure privacy and 2) to reduce latency. Without the need for sharing local data, these nodes will collaborate by sharing their models to a central node to train their model. This latter, the central node, will learn from these models in order to help improving the local models [10]. However, due to the limitation of resources of MC (Compute and network), it may be challenging to leverage AI at the edge.

2.3.1.1 Technical Challenges and Roadmap for Violation Mitigation

As a DS-CCROM fails to execute an SAP during its deployment, or a recovery action is required due to a QoS/QoE detected violation, it may not be possible to undertake recovery actions, if we limit the scope of such actions to the local resources belonging to an edge minicloud.

If the "royal road" is to let this violation or issue to emerge to a higher-level (E2E-CCROM), we are already investigating the possibility of creating point-to-point decentralized interactions between DS-CCROMs to address these kinds of issues. When the extent of the violation is limited, we shall avoid to redeploy the entire application, which is often a very expensive action and can cause the E2E-CCROM to become a bottleneck for the entire Federation. A solution that we envisage is to contact a selected group of DS-CCROMs belonging to "neighbour" domains, chosen following certain selection criteria and creating what we called an Aggregation of DS-CCROMs.

The creation and management of these Aggregations introduces a set of new technical challenges:

- **Discovering of "neighbours"**: to solve this problem, it is possible to exploit an Overlay Network, maintained between the various DS-CCROM of the Federation. A plethora of technological solutions are available to enable this kind of interaction, which will be investigated to identify some candidate techniques, such as gossip-based protocols. Another possibility is to create this Overlay Network "on the fly" exploiting the E2E-CCROM and its information about the geographical location of the EMs.
- **Management of the Aggregation**: when a set of candidate neighbours DS-CCROM are identified, a new MRMOGAP must be solved over the union of all the resources of the DS-CCROMs of the Aggregation. The new aggregation could be managed by:
    - One of the DS-CCROMs, chosen with a distributed election algorithm,
    - A new ad-hoc DS-CCROM, different from all the DS-CCROM of the Aggregation,
    - All the DS-CCROMs of the Aggregation, exploiting a distributed algorithm to coordinate the solution of the MMORGAP.

Similar techniques to "Aggregation" have been also applied in Fog computing for an automatic and spontaneous creation of virtual clusters. They often exploit

- algorithms based on the concept of betweenness centrality, i.e., measuring the centrality of a node in a graph where each vertex represents a Fog device, and
- techniques such as Spectral clustering, as in SmartFog [13].

2.3.2 CHARITY Network Resource Orchestrator module for a heterogeneous Cloud/Edge continuum

The main role of the CNROM is to enable applications' providers to define blueprints for slices of network resources, and to facilitate an efficient life cycle management of the slices with the aim of rapid slice creation and activation. As previously described, the CNROM SW architecture is linked with the CCROM SW architecture. However, reliable and efficient network orchestration poses specific scientific challenges that should be taken into consideration. Similar to CCROM, the CNROM module is logically separated in two components: (i) Domain-specific CNROM (DS-CNROM) component and (ii) End-to-End CNROM (E2E-CNROM).

2.3.2.1 E2E network orchestrator (E2E-CNROM)

An end-to-end (E2E) CNROM provides a service delivery all the way from a service provider to the end users. This orchestration stretches across different domains unifying various network layers and heterogenous technologies enabling an overlaid layer and providing an efficient convergence of networking and services [11]. An E2E-CNROM is responsible for defining the slice blueprint in terms of network resources.

2.3.2.2 Domain-specific network orchestrator (DS-CNROM)

After defining the slice blueprint, E2E-CNROM issues a request from the corresponding DS-CNROM. Each DS-CNROM, on its turn, forwards the request to the corresponding entity (e.g., NFVI controller, Domain-specific SDN controller) to handle the requests and allocate the resources. CHARITY UC applications have strict network requirements in terms of bandwidth and latency. CNROM relies on closed-loop automation to ensure the applications' KPIs.

A network slice is composed of several VNFs interconnected and optimally placed to meet specific KPIs defined by the application provider (latency and bandwidth). A key challenge for developing this component is how to map an application blueprint as described by the application developer to a concrete slice blueprint that defines the compute and network resources, and network functions [12].

Identifying the infrastructure-domains with the required resources that can meet the application requirements is also challenging. Most importantly, CHARITY use case applications require strict KPIs in term of bandwidth and latency, and compute resources which could be accommodated by creating customized slices across the multi-domain architecture (CCs, EMs) that could be composed of several network instances where each slice is on its turn composed of network resources and network functions.

2.4 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. **D**istributed 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 2.3.1. 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. In CHARITY, we plan to explore this type of solutions, particularly within WP3 – T3.1 Efficient exploitation of CPUs, GPUs and FPGA.

2.5 Application Resource Management

Applications designed according to the principles of the Service-Based Architecture and distributed across computing nodes ranging from devices to the edge and the cloud need the support of resource orchestration to ensure they are adequately resourced to fulfil their objectives. To the Orchestrator, applications operate somewhat as a black box. While the ebbs and flows of their actual resource usage can be observed through detailed monitoring, the exact causes of peaks and troughs in their resource demands can be only confidently deduced through knowledge of the application logic and user interaction consequences. This difficulty is exacerbated when we seek to accommodate applications that can 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), the Orchestrator needs to 'be in the loop' so that resources can be efficiently managed. Self-adaptation may not be as straightforward as services within the application reducing their fidelity or toggling 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 network orchestration to enable efficient application-aware resource orchestration.

2.5.1 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 [14] 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[2]. The key concern is that edge resources, whether operated by the enterprise or

[2] Indeed, this is a model envisaged by Collins Aerospace for Flight Simulator training centers.

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.

2.5.2 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[3] 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 [15] 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 [16], 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

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

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-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 6 which represents a high-level deployment view and possible adaptation scenarios for a distributed service-based application[4].

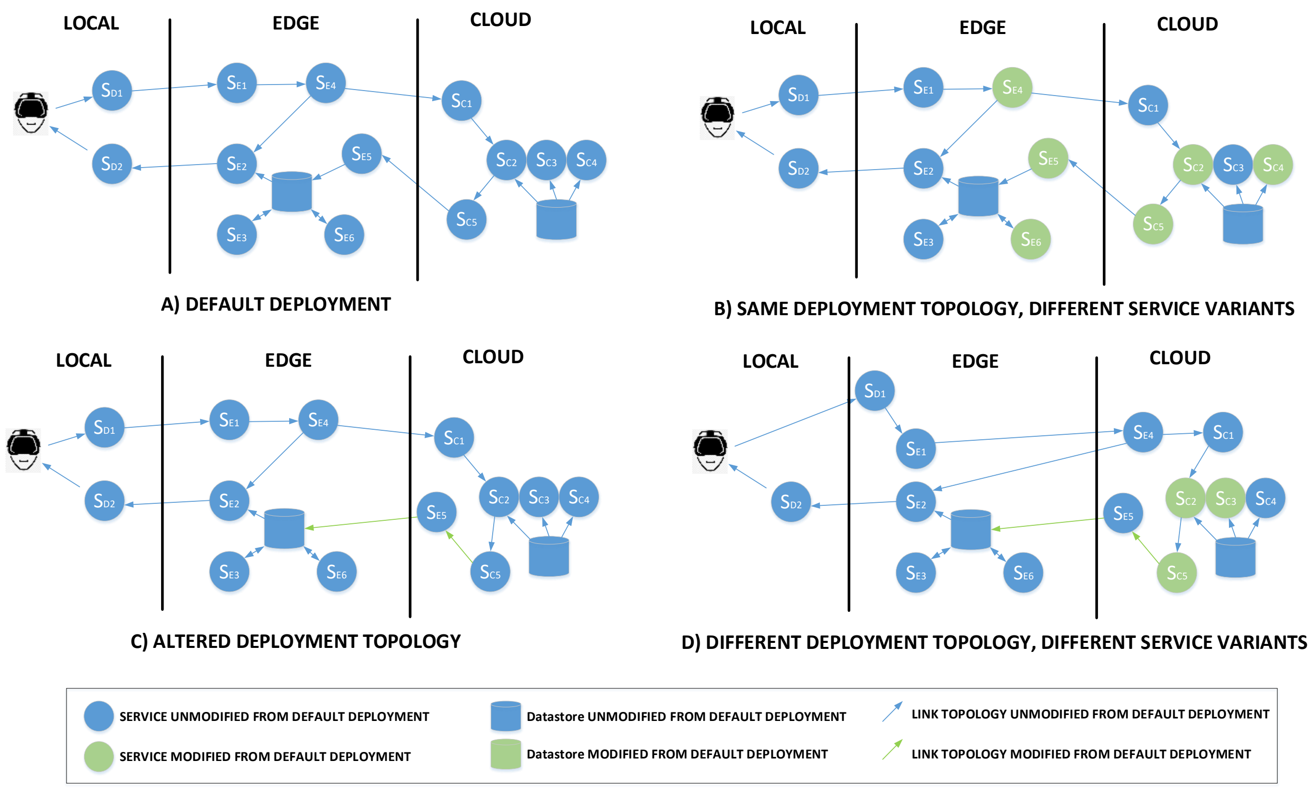

LOCAL
EDGE
CLOUD
SD1
SD2
SE1
SE2
SE3
SE4
SE5
SE6
SC1
SC2
SC3
SC4
SC5
A) DEFAULT DEPLOYMENT
B) SAME DEPLOYMENT TOPOLOGY, DIFFERENT SERVICE VARIANTS
C) ALTERED DEPLOYMENT TOPOLOGY
D) DIFFERENT DEPLOYMENT TOPOLOGY, DIFFERENT SERVICE VARIANTS
SERVICE UNMODIFIED FROM DEFAULT DEPLOYMENT
SERVICE MODIFIED FROM 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 6: Application adaptation scenarios.

In Figure 6, 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

[4] Approximately modelled on a real flight simulator application.

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.

2.5.3 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 large 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[5].

To cover the adaptation use cases summarized in Figure 6, we need to support the following functionality from the perspective of the orchestrator:

- 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

Resources are allocated to application instances by the Orchestrator in network slices. To support the increased elasticity that we need from the slice model to allow topological or resource sizing changes to a running application instance, we propose to introduce the concept of *subset slices*. The concept is straightforward and requires application providers to separate which parts of their deployed application may be altered to affect a particular target adaptation and which parts should remain static. We refer to these as the dynamic and static subsets. Those services within the dynamic subset can collectively have their resources recalibrated or even reallocated on a completely different machine. We capture the static and dynamic subsets in separate blueprints[6] and send/identify them both to the Orchestrator when seeking to launch the application. When adapting the application, we then send a new dynamic subset blueprint to the Orchestrator, receive back the details on resource location changes, perform any necessary inter-service traffic routing changes within the application and finally request the Orchestrator to reclaim any resources that were reserved for use before adaptation but not afterwards. This is summarized in Figure 7.

---

[5] The notion of disposable services that can be released and respawned anew lies at the heart of Microservice architecture tenets. The oft-referred-to “pets versus cattle” metaphor captures this ideal.

[6] Logically separate if not physical. The detail of this is yet to be formalized as we feel the decision on whether to use multiple blueprints or a single more robust one is best made when we have progressed further along with the implementation of the CHARITY Orchestrator.

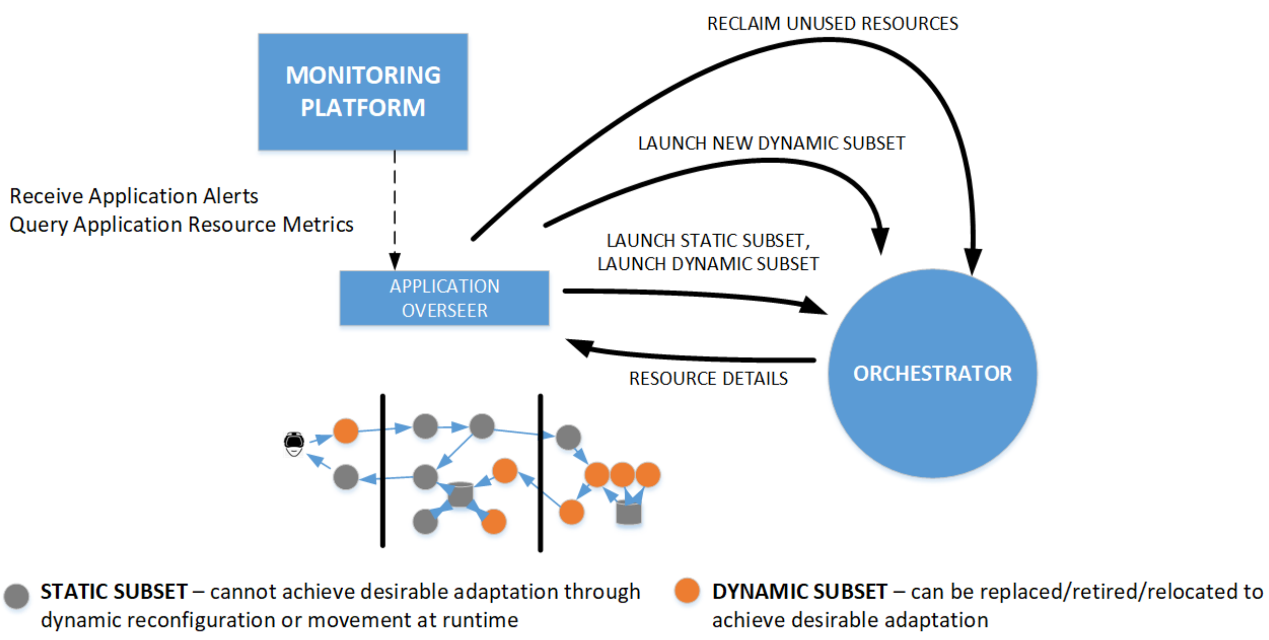

RECLAIM UNUSED RESOURCES
MONITORING PLATFORM
LAUNCH NEW DYNAMIC SUBSET
Receive Application Alerts
Query Application Resource Metrics
LAUNCH STATIC SUBSET, LAUNCH DYNAMIC SUBSET
APPLICATION OVERSEER
RESOURCE DETAILS
ORCHESTRATOR
● STATIC SUBSET – cannot achieve desirable adaptation through dynamic reconfiguration or movement at runtime
● DYNAMIC SUBSET – can be replaced/retired/relocated to achieve desirable adaptation

Figure 7: Application Aware Orchestration.

The Application Overseer 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 the Orchestrator to affect the adaptation. How the Overseer makes its decisions and maps circumstances to a particular choice of dynamic subset is completely up to the Application Provider. It can receive alerts and query metrics from the CHARITY Monitoring Platform 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 slices allocated to active users; respond to quality of experience and resource fluctuations observed by the Monitoring Platform.

3 Algorithms for service orchestration

3.1 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.

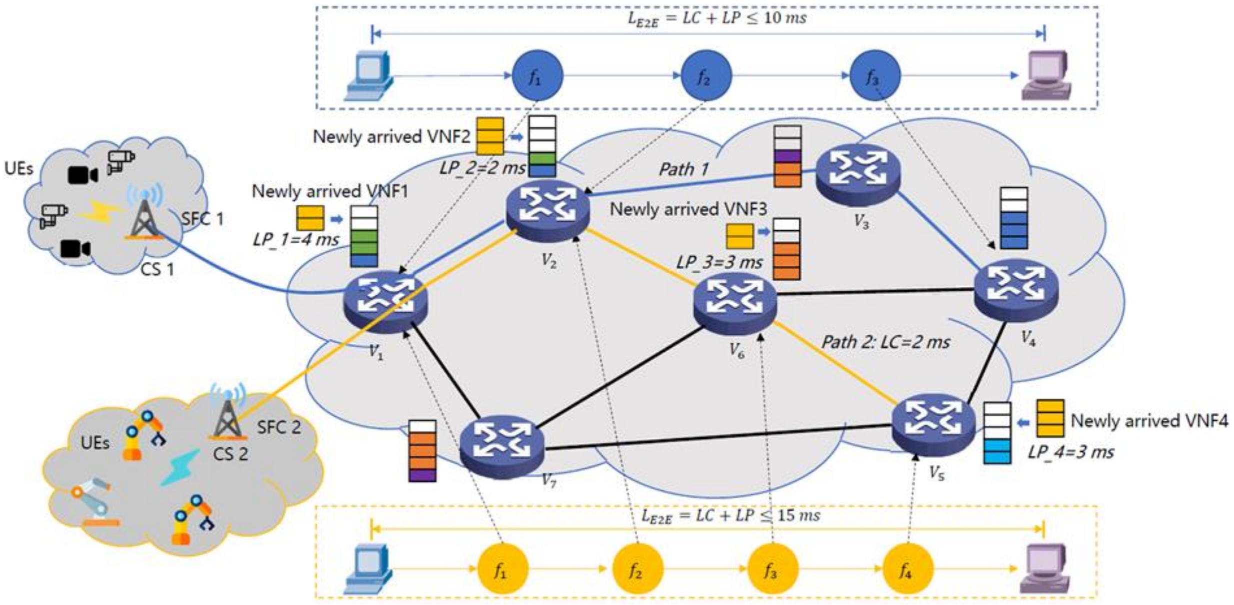


Figure 8: Example of SFC requests in 5G edge networks [17].

Figure 8 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 [17], 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 [17], 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.

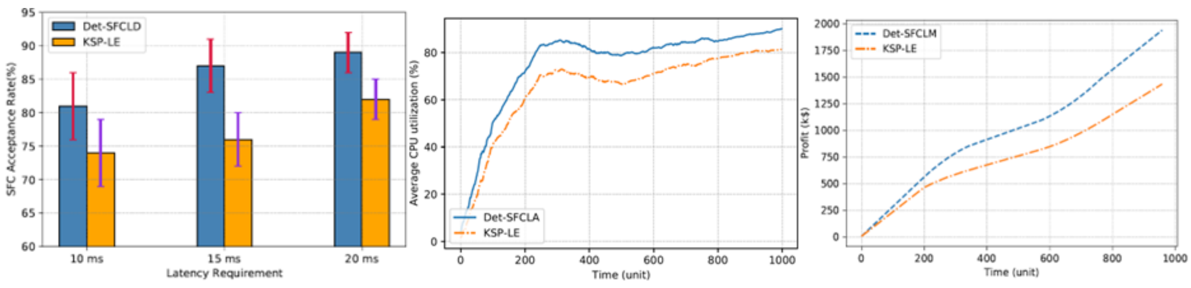


Figure 9: Performance evaluation of DET-SFCD and DET-SFCA algorithms [17].

The conducted performance evaluation (Figure 9) 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.

3.2 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 10 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.

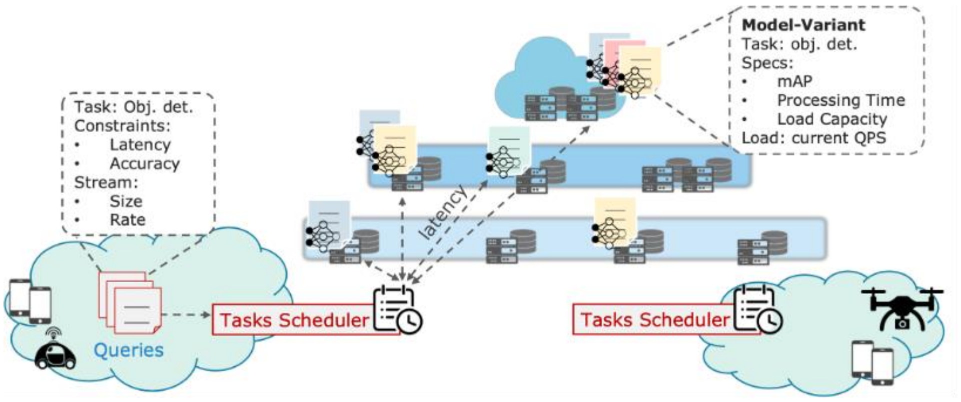


Figure 10: Request scheduling for optimal resource allocation at real time.

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.

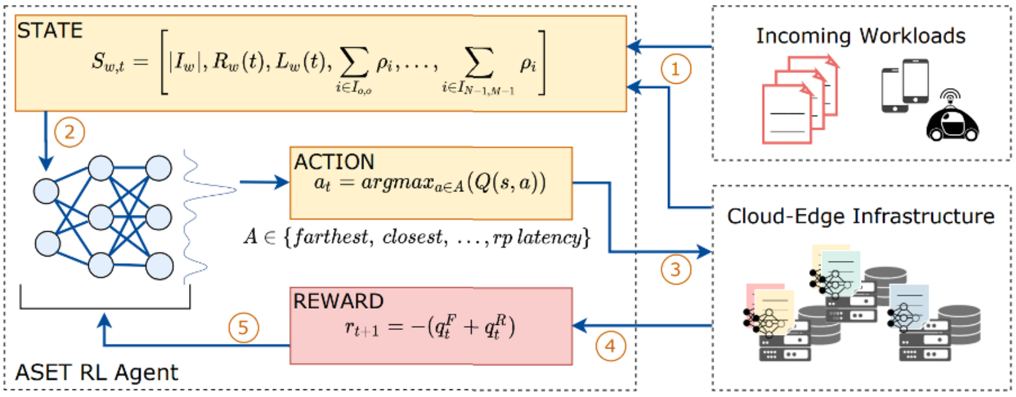

STATE
$S_{w,t} = \left[|I_w|, R_w(t), L_w(t), \sum_{i \in I_{o,o}} \rho_i, \ldots, \sum_{i \in I_{N-1,M-1}} \rho_i \right]$
ACTION
$a_t = argmax_{a \in A}(Q(s,a))$
$A \in \{farthest,\ closest,\ \ldots, rp\, latency\}$
REWARD
$r_{t+1} = -(q_t^F + q_t^R)$
ASET RL Agent
Incoming Workloads
Cloud-Edge Infrastructure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

Figure 11: 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 11.

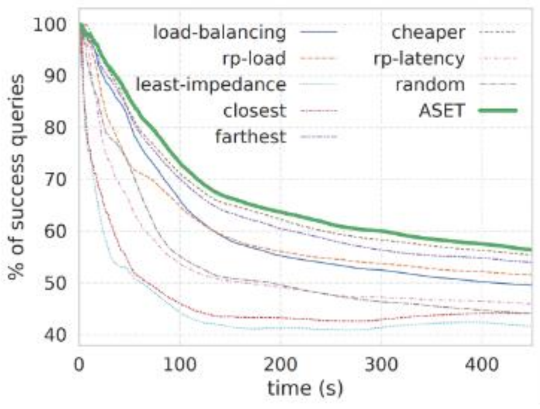


Figure 12: Percentage of successful queries over time for ML task with users arriving in real-world pattern.

Initial results suggest, even with partial view of the network resources, ASET performs better than traditional scheduling mechanisms (Figure 12). 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 (section 0) 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.

3.3 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) [18, 19]. 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.

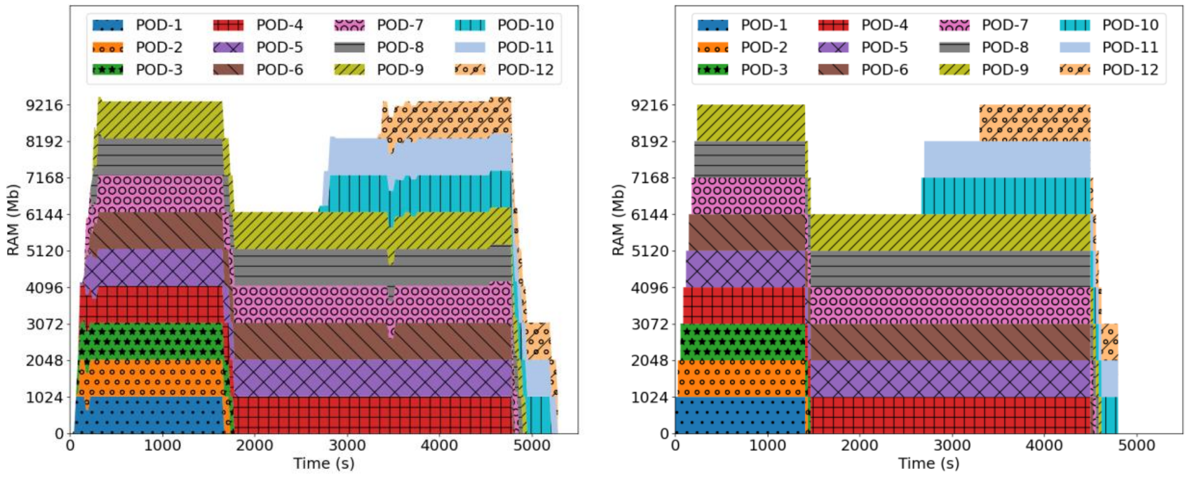
In the context of CHARITY, we consider that the system at-the-edge is made of entities with a specific geographic location representing a small-to-mid pool of potentially heterogeneous resources (i.e., EM – Edge Miniclouds). For the sake of simplification, one can model a system in which each EM is modelled 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. Our idea is to use the PureEdgeSim simulator [20], 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 [21].

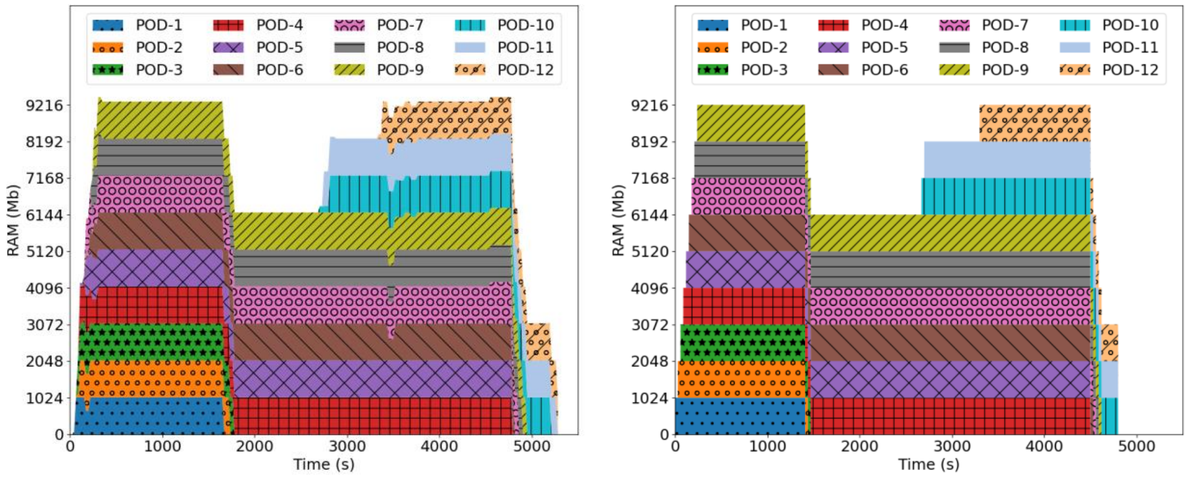
3.4 Simulations tools and experiments on cloud resource management

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 [1]. 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 13 shows memory consumption for each POD. From



a) Real testbed b) Simulated Testbed

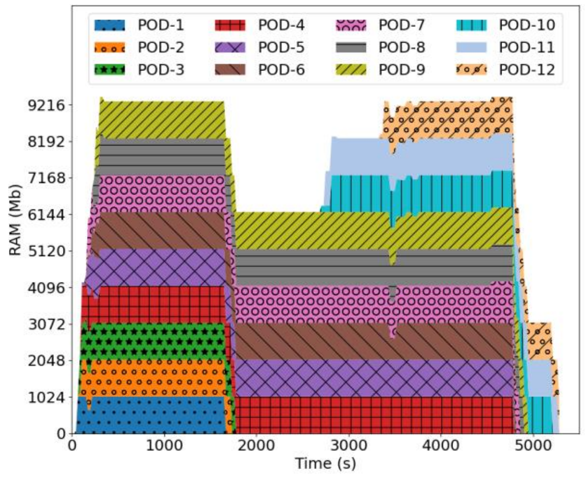
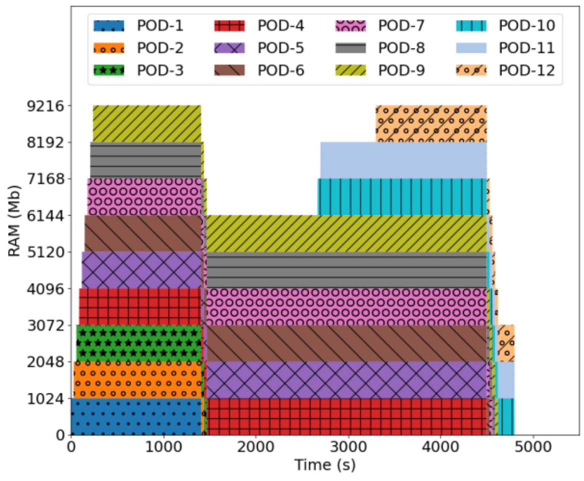
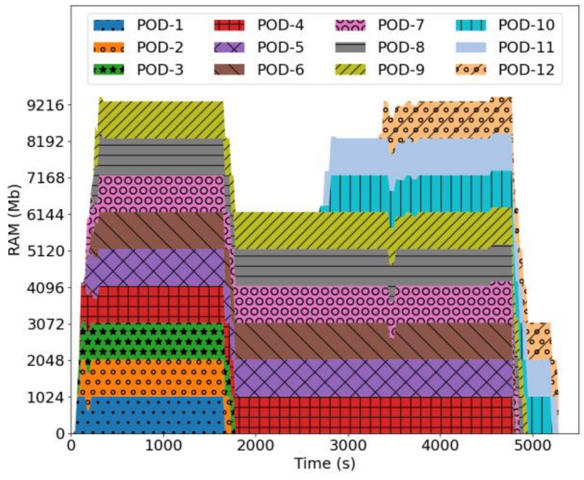


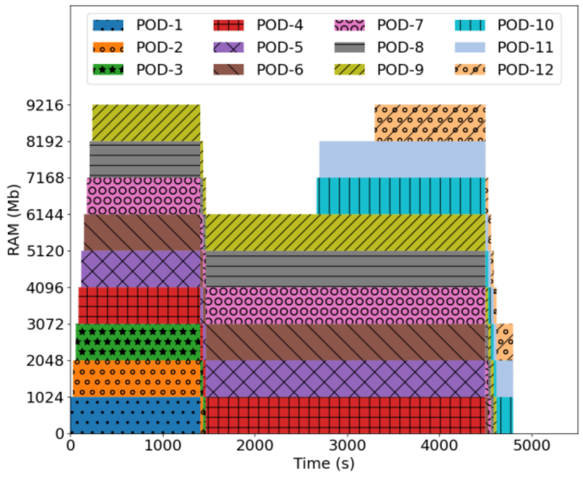
Figure 13(a) and



a) Real testbed b) Simulated Testbed

Figure 13(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.





a) Real testbed b) Simulated Testbed

Figure 13: Memory consumption per POD [1].

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[7] 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[8] and a Grafana[9] 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 [22]. 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[10] and stress-ng[11]. 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.

Figure 14 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

[7] https://rancher.com/

[8] https://prometheus.io/

[9] https://grafana.com/

[10] https://iperf.fr/

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

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.

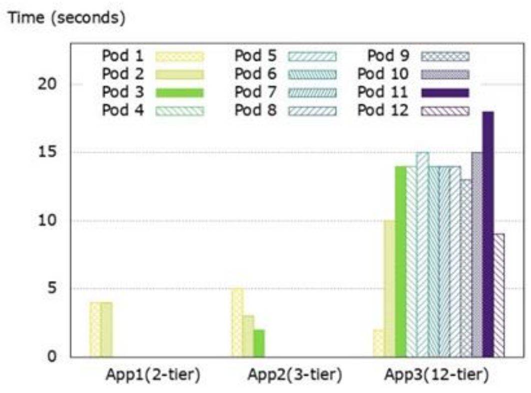


Figure 14: Average pod deployment time of each application [22].

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 15, it is possible to visualize the various states of this process and their time on each pod.

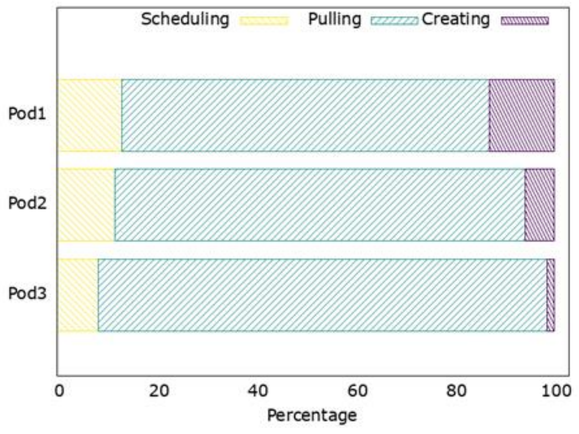


Figure 15: Stages included in deployment time [22].

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 16 and Figure 17 show the visualization of the graphs obtained from CPU usage and the memory usage of the deployed 12-tier application.

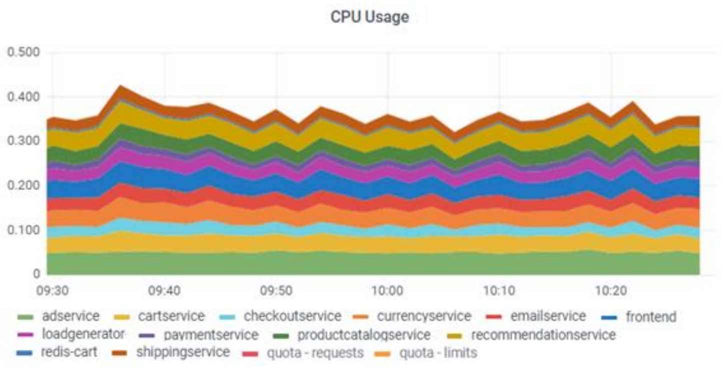


Figure 16: CPU Usage graph from 12-tier application [22].

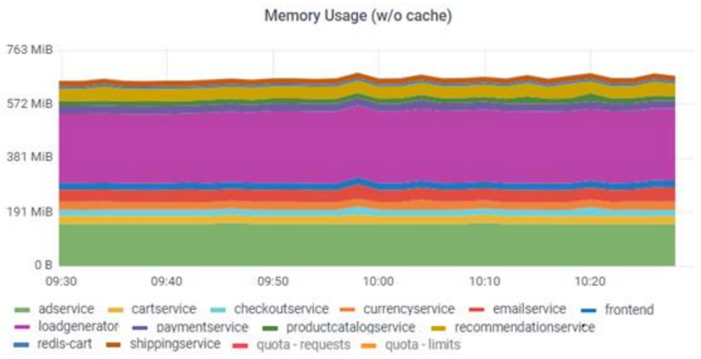


Figure 17: Memory Usage graph from 12-tier application [22].

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 2.5) 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).

4 XR Aware networking

4.1 Dynamic routing

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 [23]:

- 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 [24]. 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 [25] 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 [26], 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 [27] 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 [23, 28]. Other existing works [29, 30] 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 [31] developed an evolutionary multipath routing algorithm based on GA to solve the multi-commodity flow problem while authors in [32] leveraged GA and incorporates a fitness function inspired by RL for the priority flow admission and routing problems. The solution in [33] 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 [34] 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 [35]:
    - 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 [36], 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. [37] 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 [38] proposed a mechanism dubbed DROM, a routing optimization mechanism based on Deep Deterministic Policy Gradient (DDPG) [39], 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 [40] combines NN with RL via replacing Q-tables with an approximate function trained by NN. Rischke et al. [41] 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 18) 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.

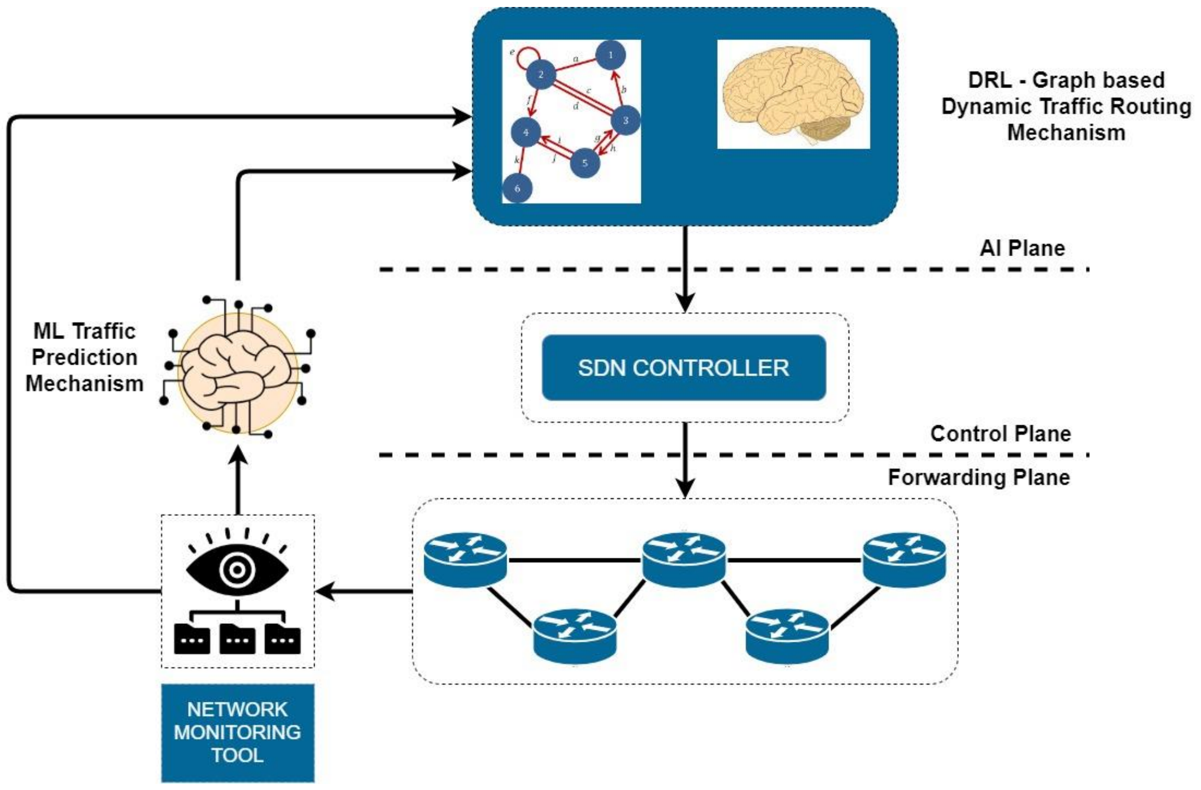

DRL - Graph based Dynamic Traffic Routing Mechanism
AI Plane
SDN CONTROLLER
Control Plane
Forwarding Plane
ML Traffic Prediction Mechanism
NETWORK MONITORING TOOL

Figure 18: 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.

4.2 Deterministic-Networking and TSNs

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 [42]. 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 [43].

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 of the 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 19 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).

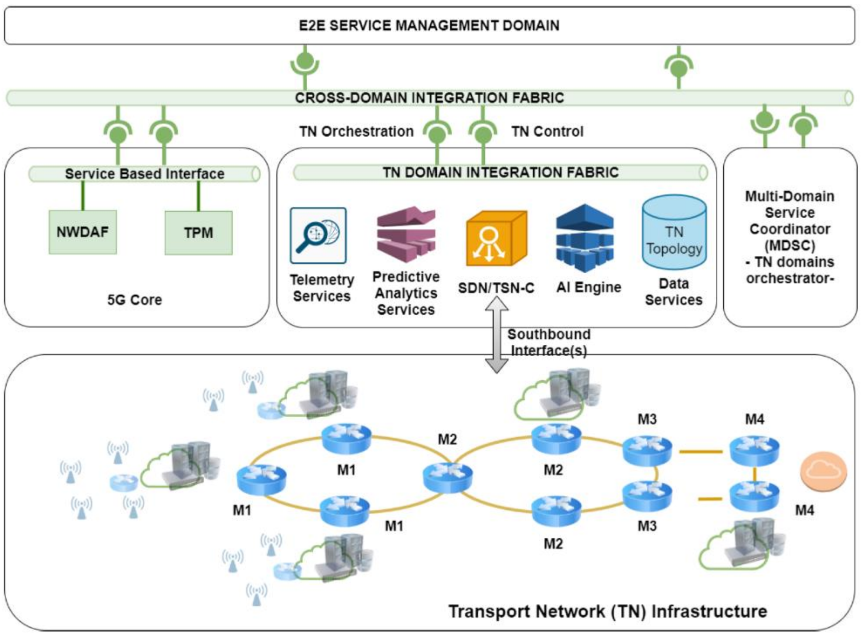

E2E SERVICE MANAGEMENT DOMAIN
CROSS-DOMAIN INTEGRATION FABRIC
TN Orchestration
TN Control
Service Based Interface
NWDAF
TPM
5G Core
TN DOMAIN INTEGRATION FABRIC
Telemetry Services
Predictive Analytics Services
SDN/TSN-C
AI Engine
TN Topology
Data Services
Multi-Domain Service Coordinator (MDSC) - TN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

Figure 19: Transport network management and orchestration architecture [44].

In the ambit of CHARITY, a study [44] 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.

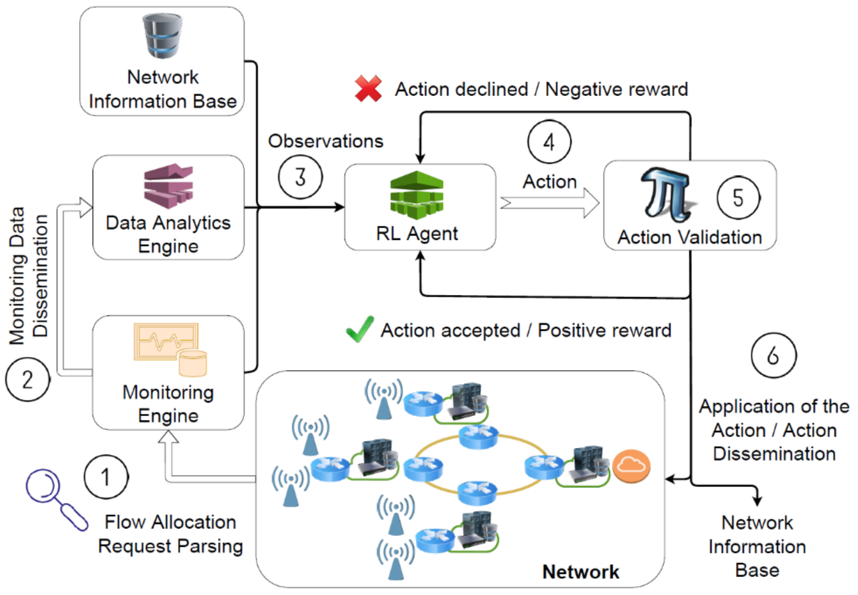
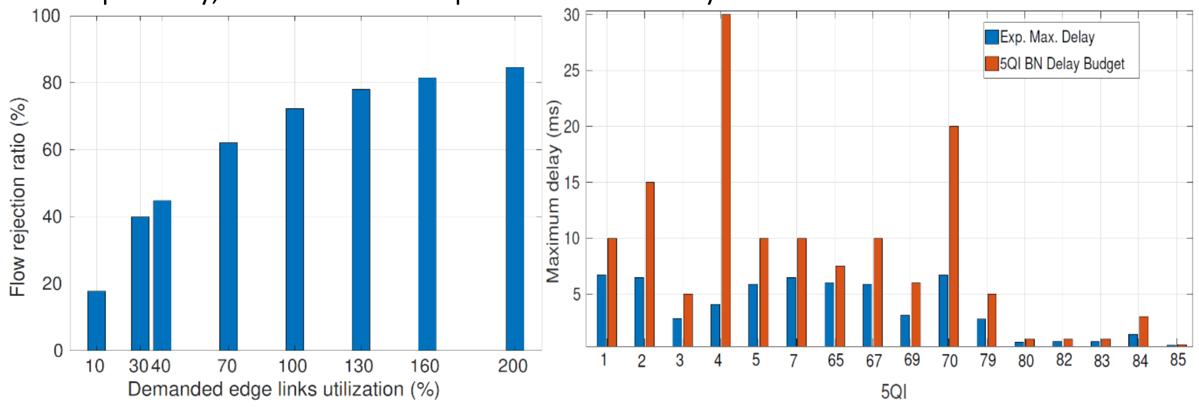

Network Information Base
Action declined / Negative reward
Observations
3
4
Action
Data Analytics Engine
RL Agent
Action Validation
5
Monitoring Data Dissemination
Action accepted / Positive reward
6
2
Monitoring Engine
Application of the Action / Action Dissemination
1
Flow Allocation Request Parsing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

Figure 20: RL for flow allocation and time-sensitive networks optimization [44].

Figure 20 shows an online RL-based solution for the flow allocation policy-making in ATS-based networks [44]. 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.



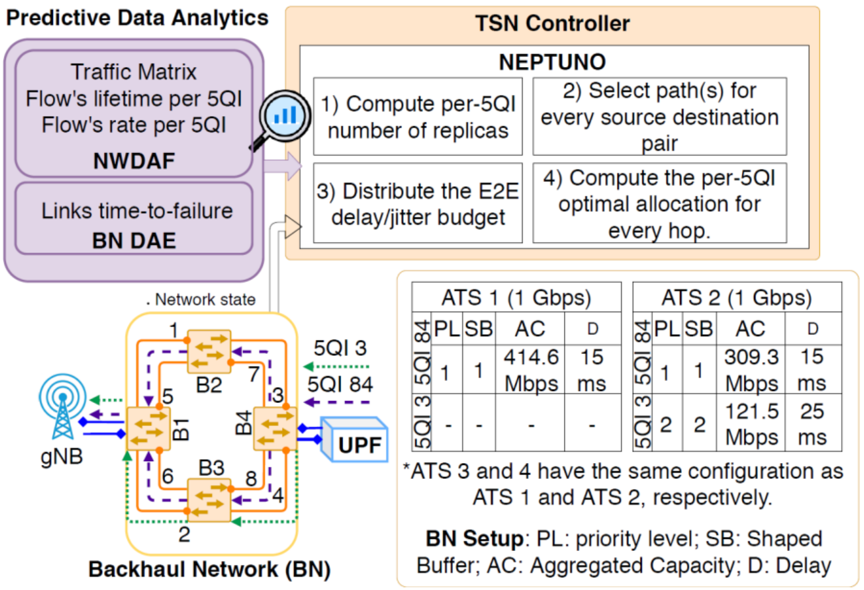
a) Flow rejection ration b) Worst case delay

Figure 21: ATS-based Backhaul Network (BN) performance [44].

Figure 21(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 19). Every point shown in Figure 21(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[12]), as it seeks for maximizing the number of accepted flows. Figure 21(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 [45] 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.

[12] 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
gNB
B1
B2
B3
B4
5QI 3
5QI 84
UPF
Backhaul Network (BN)

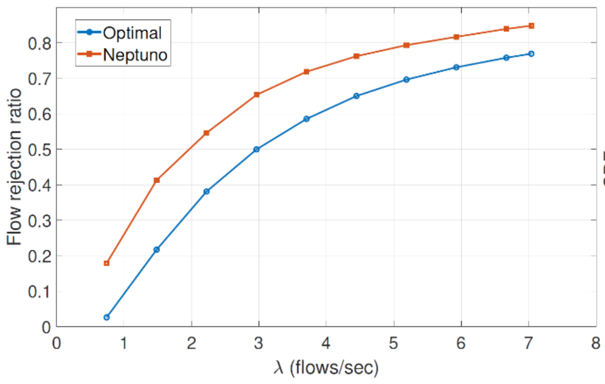
	ATS 1 (1 Gbps)				ATS 2 (1 Gbps)			
	PL	SB	AC	D	PL	SB	AC	D
5QI 84	1	1	414.6 Mbps	15 ms	1	1	309.3 Mbps	15 ms
5QI 3	-	-	-	-	2	2	121.5 Mbps	25 ms

*ATS 3 and 4 have the same configuration as ATS 1 and ATS 2, respectively.

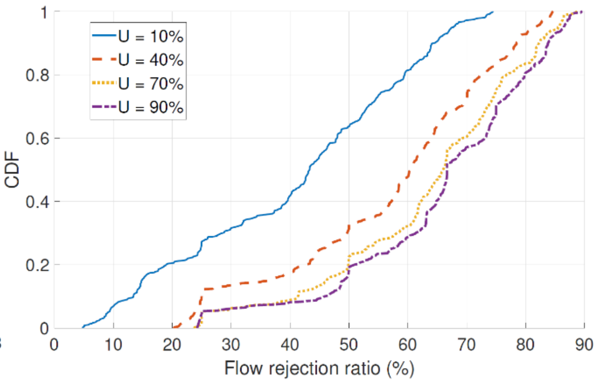
BN Setup: PL: priority level; SB: Shaped Buffer; AC: Aggregated Capacity; D: Delay

Figure 22: Main stages of NEPTUNO for computing the optimal configuration of the network and an example illustrating the primary configuration parameters for two 5QIs [45].

Figure 22 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 23: Performance of NEPTUNO [45].

Figure 23(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 23(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.

5 Monitoring and Prediction

Monitoring encompasses the process of collection, analysis and use of information systematically, that provides the continuous visualization and perception of the status and the progress 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). This way, as soon as something happens outside the expected behaviour, it is possible to 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 whole new relevance in the orchestration and life-cycle management of next-generation applications. They form the input upon which all the intelligent orchestration mechanisms are built. The concept of control closed loops and the envisioned automation of such intelligent orchestration highly depends on a comprehensive real-time monitoring approach and on the quality of collected metrics. In what follows, we discuss the importance of monitoring for resource consumption prediction algorithms, the open research challenges and the surveyed specific enablers and tools for Cloud-Native environments.

5.1 Resource Monitoring

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.

5.1.1 Goals and Research Challenges

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 [46].

Moreover, monitoring tools should also have the following characteristics [47]: ***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* ***archivability***. 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 1 summarizes the above discussion and gives an overview on the monitoring requirements.

Table 1: 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

5.1.2 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 [48, 49, 50].

- ***Prometheus***[13] 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[14], 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**[15] 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 [51], 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) [52], 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***[16] 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***[17] 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).

[13] https://prometheus.io/

[14] https://grafana.com/

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

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

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

- ***Zabbix***[18] 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***[19] 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***[20] 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***[21] 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***[22] 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

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

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

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

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

[22] 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***[23] 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.

5.1.3 Relation to CHARITY

This section completes the discussion of the aforementioned monitoring tools with a brief comparison among them. It then discusses how they can be leveraged in the scope of the CHARITY framework. As before, the realization of more autonomous and intelligent orchestration mechanisms as envisioned in CHARITY highly depends on the input provided by such monitoring components. Hence it becomes critical to understand how they fit in the proposed architecture and how they can be used as the input for the proposed mechanisms.

Inspired by the ETSI Zero-touch specification, CHARITY aims to achieve a platform 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. This monitoring, the initial step of control closed loops, is essential to optimize the QoE of XR applications and to ensure they run as expected. Together with the intelligent prediction and orchestration algorithms, monitoring tools are key elements to automate, reduce detection and decision times and minimize the need for human intervention in the overall service orchestration. Table 2 presents a brief comparison of the aforementioned monitoring tools based on the initial findings and survey work [24].

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

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

Table 2: 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. 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[25] or Fluentd datasources[26]).

Figure 24 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.

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

[26] https://www.fluentd.org/datasources

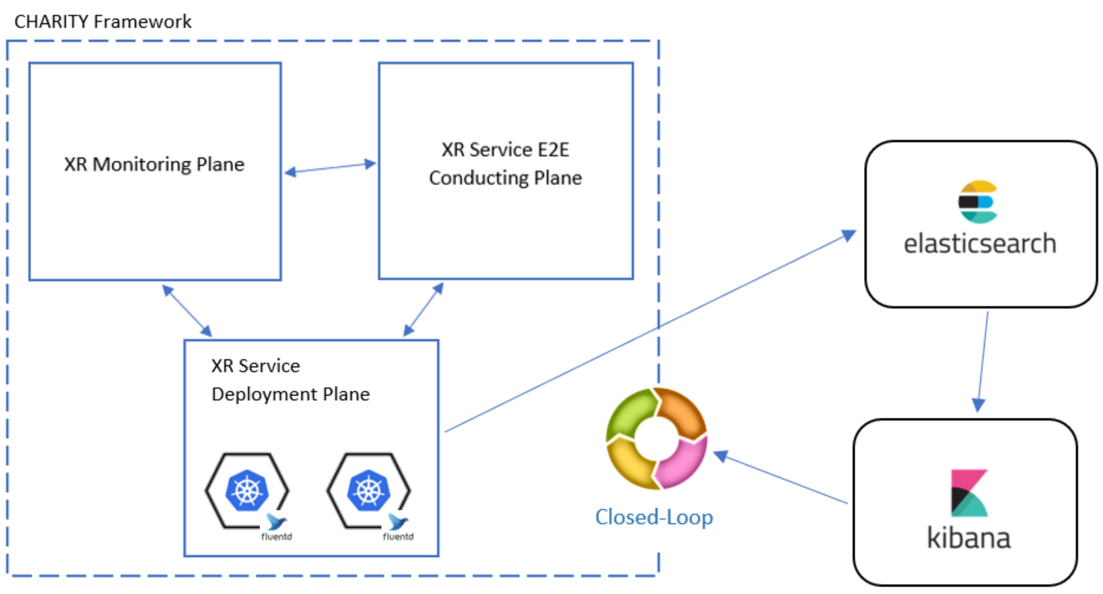


Figure 24: Implementation of EFK based on Kubernetes, adapted from27.

Similar, Figure 25 illustrates how Prometheus and Grafana tools fit in 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.

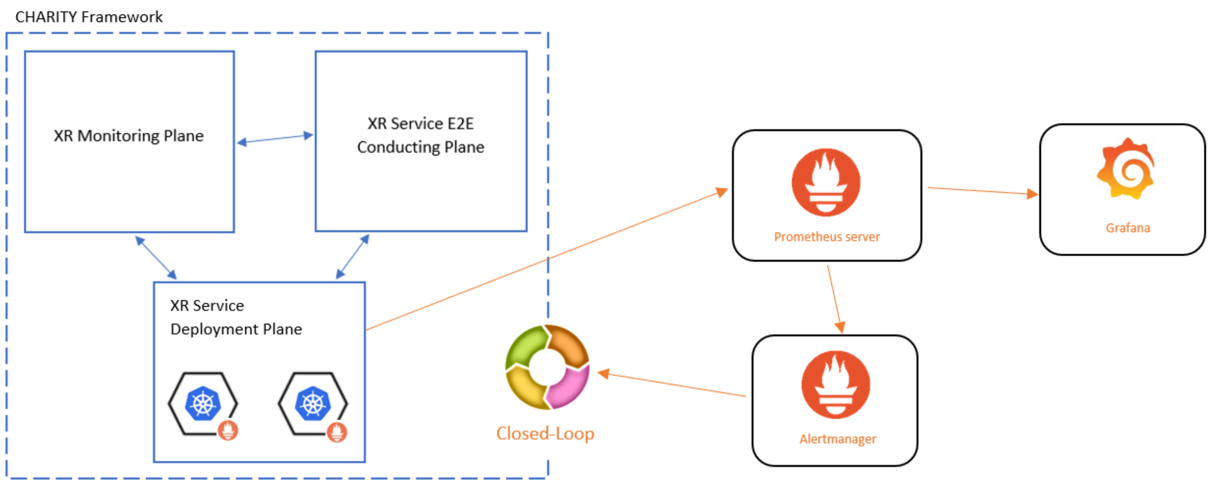


Figure 25: Implementation of Prometheus and Grafana based on Kubernetes adapted from28.

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.

5.1.4 Monitoring Architecture

The previous analysis of existing tools has been crossed with the demands of a XR monitoring tool. The **heterogeneity** is achieved with an existing tool that is extensively documented and with hundreds of elements already developed to extract metrics from the components to be monitored. This is the case

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

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

of **Prometheus**, which has a large and proactive community that continues to develop and update the open-source tool to offer native Prometheus endpoints to expose data to be collected. The pull mechanism is another of the strong points of Prometheus (Figure 26), since in an ecosystem with so many microservices deployed and in so many places, the tool would become a bottleneck. In addition, it allows backward error traceability against network failures by being able to check them from the origin, which would not be possible with the push method in which each microservice would send its performance data to the monitoring tool on its own.

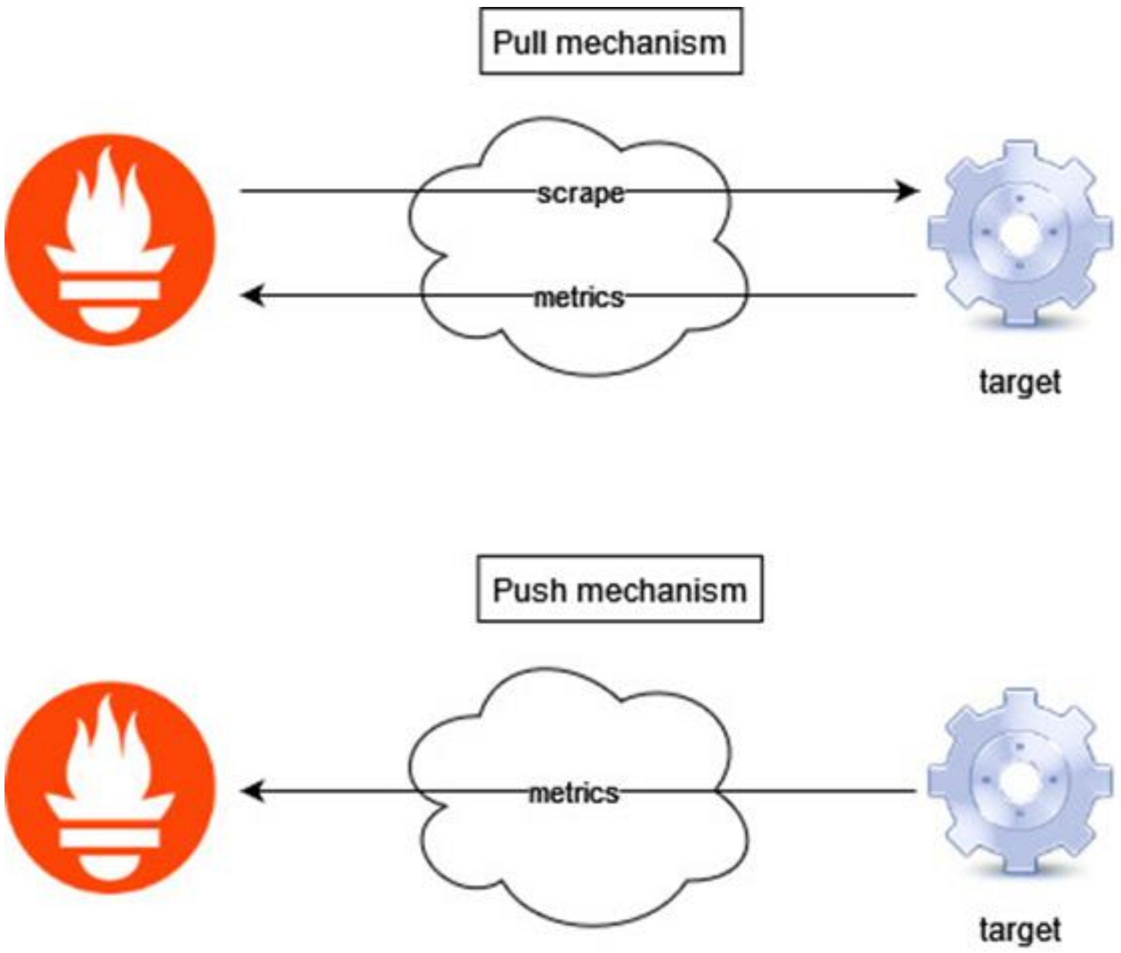


Figure 26: Pull vs Push mechanism.

The use of an existing tool has allowed us to establish from the outset the format of the data that will be monitored to facilitate the parallel development of other elements of the architecture that depend on them, such as the orchestrator. The analysis of the architectures of the use cases and the needs of their developers has allowed us to establish the list of example metrics presented in Table 3, also its formats, names and units have been agreed upon at the project level.

Table 3: Example monitoring metrics for CHARITY use cases

METRICS	DEFINITION	OUTPUT NAME	OUTPUT UNITS	FORMAT	EXAMPLE
Latency	Time it takes for a request to reach the destination and return, including the operation time of the destination to respond to the request	latency	miliseconds -ms	three decimals	125.123
RTT	Round trip time. Time it takes for a request to reach the destination and return. It does not include	rtt	miliseconds -ms	three decimals	125.123

	the operation time of the destination to respond to the request				
Bandwidth	Maximum capacity that can be transmitted over a link	bw	Mbps	three decimals	1000.000
CPU	Percentage of used CPU	cpu	percentage	positive integer	0-100
GPU	Percentage of used GPU	gpu	percentage	positive integer	0-100
Memory	Percentage of used memory	memory	percentage	positive integer	0-100
Resolution	Number of pixels a screen is capable of displaying	resolution	Megapixel	three decimals	4.096
Color bit depth	Number of bits needed to represent the color of a pixel	colorbitdepth	bits per pixel	positive integer	24
Frame-rate	Frequency at which a device displays images	framerate	Frames per second - fps	positive integer	240
Petitions per second	Number of requests per second	petitionspersecond	Requests per second	positive integer	1000

Figure 27 depicts the monitoring architecture of CHARITY. The **multicloud and agnosticity** premises require performance data to be maintained by an element outside the cloud providers, a task that **Thanos** can certainly do. It is a storage Prometheus setup that brings together the data from the servers deployed in the different clouds and a query system that allows data to be consulted and crossed. The multicloud architecture implies one Prometheus server per cloud domain, all registered in the Plane Services Registry & Discovery of the XR Service E2E Conducting Plane to update its configuration with each new service deployed or migrated. The communication between Thanos and the Prometheus servers outside its cloud domain is done by sidecars, they are Thanos components deployed along with Prometheus instances outside the main cluster that allow them to advertise data sources to the central data storage element, Thanos. This will be the data source from which the visualization tool, Grafana, open-source and widely used, will be drawn, which meets the complexity reduction requirements of CHARITY's monitoring architecture. The basic operation of Grafana with Prometheus creates a visualization instance with all the data of the elements monitored by that Prometheus server[29]. The challenge here, with Thanos as data center, is to create a user framework that collects data independent of each use case without exposing the others.

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 must be added to the monitoring that makes dynamic the service monitoring, based on a static tool like Prometheus. The so-called monitoring agent will work with the orchestrator and adapt the configuration of the Prometheus servers deployed in each cloud. The monitoring agent will use a template repository to automatize all the changes in the monitoring system. Prevention will have a personalized character through a Prometheus Alertmanager configurable by the UC owners. This will be developed through the user framework that will show the individualized alerts of each case and will offer the configuration of alerts over the elements of the own application.

[29] https://grafana.com/grafana/dashboards/12937

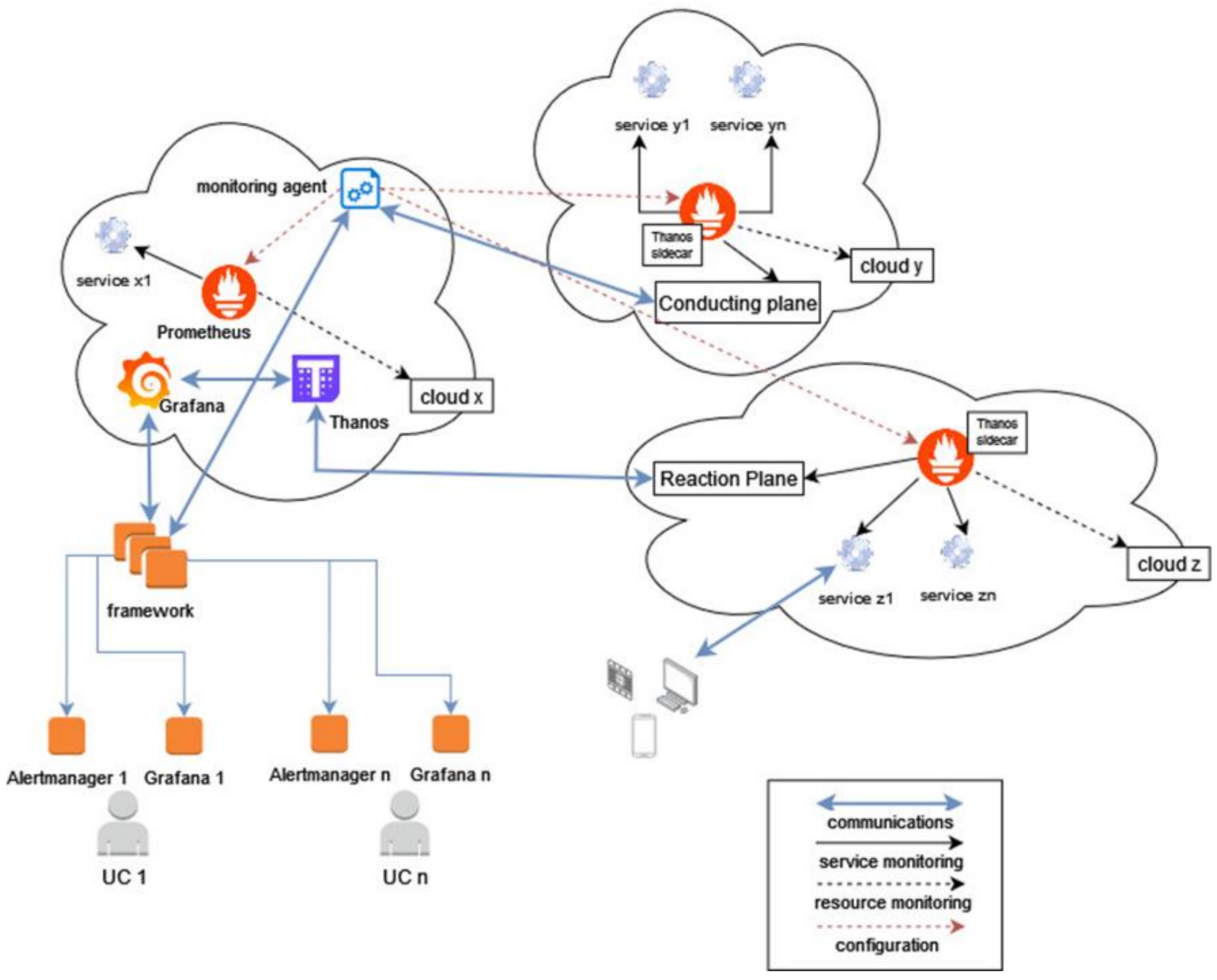


Figure 27: CHARITY monitoring architecture.

5.2 Computation utilization prediction

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 [53]. Some of these frameworks are based on the use of ARIMA models, such as the ARIMA-DEC [54]. which is VM provisioning technique based on load prediction. In [55], 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 [56], an LSTM-based model is used in order to predict future CPU utilization. Furthermore, in [57], 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 [58], 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. [59] 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 28 shows the architecture of the proposed double tower neural network.

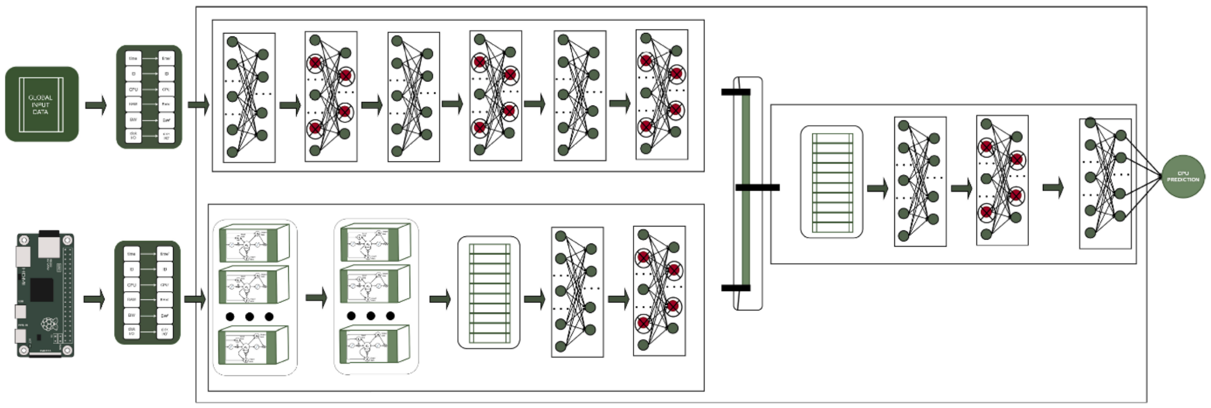


Figure 28: Architecture of the proposed Double Tower Neural Network [59].

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 [59].

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 4.

Table 4: Experimental results of the proposed Intelligent Autoscaling method [59]

	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 Autoscaller 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 4, 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 4 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 5, 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 5: 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 5. 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 5, 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.

5.3 Network state prediction

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 [60] 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) [61] 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 [62]. 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 centers have been used successfully for single-service traffic prediction and interactive network traffic prediction [63].

Many data transfer, storage and processing services include short- and long-range time dependencies, making the multi-step prediction a prominent solution [64]. Multi-step prediction using RNN with iterated prediction over many time steps has been applied for IoT traffic time-series prediction [65]. 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 29 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.

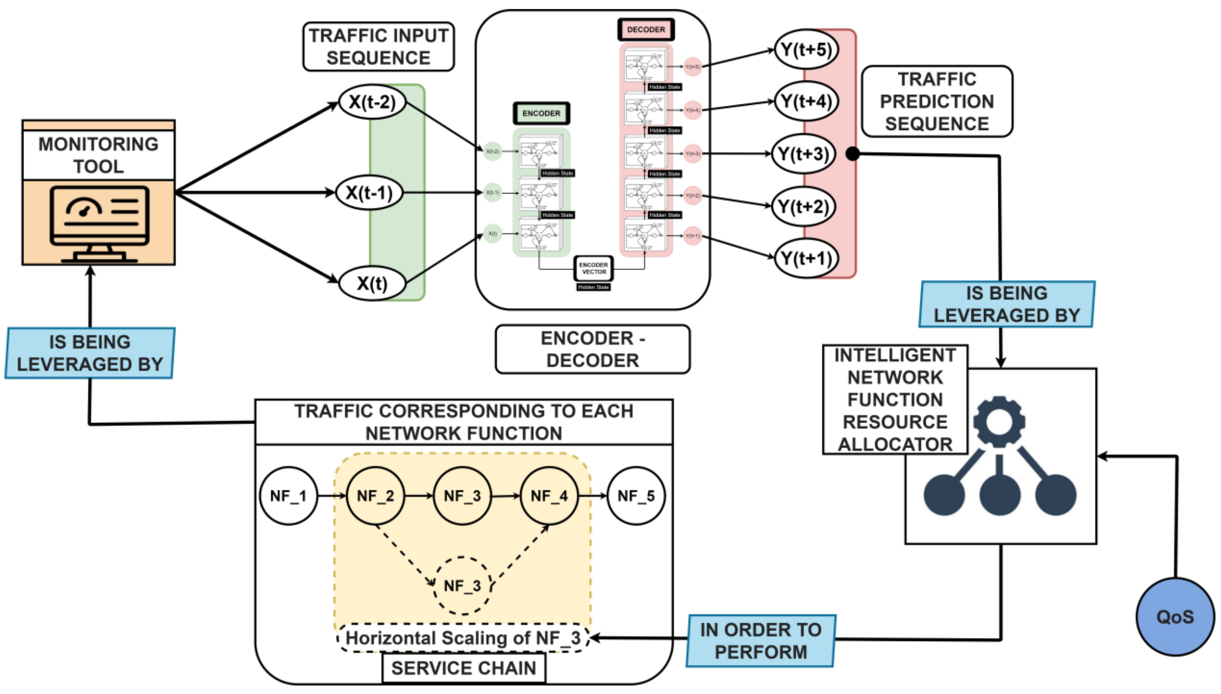


Figure 29: Leveraging Service Traffic Prediction for Horizontal Scaling of Network Functions [66].

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 6, Table 7 and Table 8, 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 6: Request number [66]

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 7: Transmitted data [66]

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 8: Session duration [66]

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 [66]. 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.

6 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.

6.1 Service migration triggers

In this section, the triggers of service migrations are discussed, it summarizes Addad et al.'s work [67]. 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 30 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 [68]. 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.

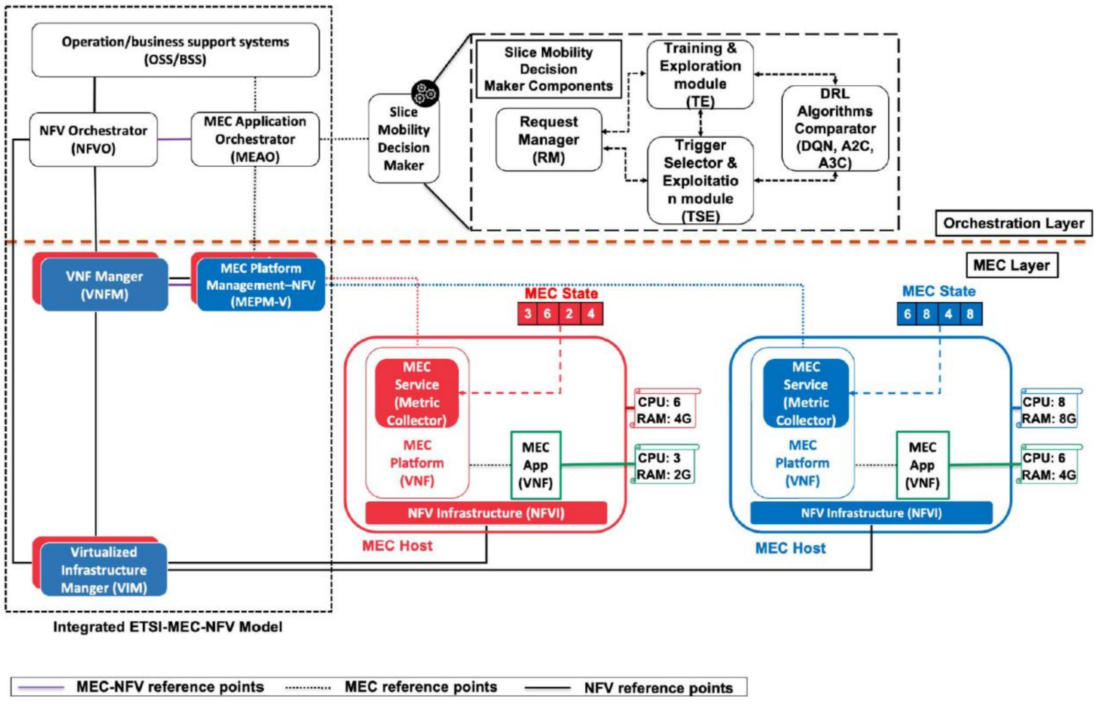

Operation/business support systems (OSS/BSS)
NFV Orchestrator (NFVO)
MEC Application Orchestrator (MEAO)
Slice Mobility Decision Maker
Slice Mobility Decision Maker Components
Training & Exploration module (TE)
Request Manager (RM)
Trigger Selector & Exploitation module (TSE)
DRL Algorithms Comparator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

Figure 30: Architecture of smart triggers selection for service migration [67].

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 31, 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.

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, …).

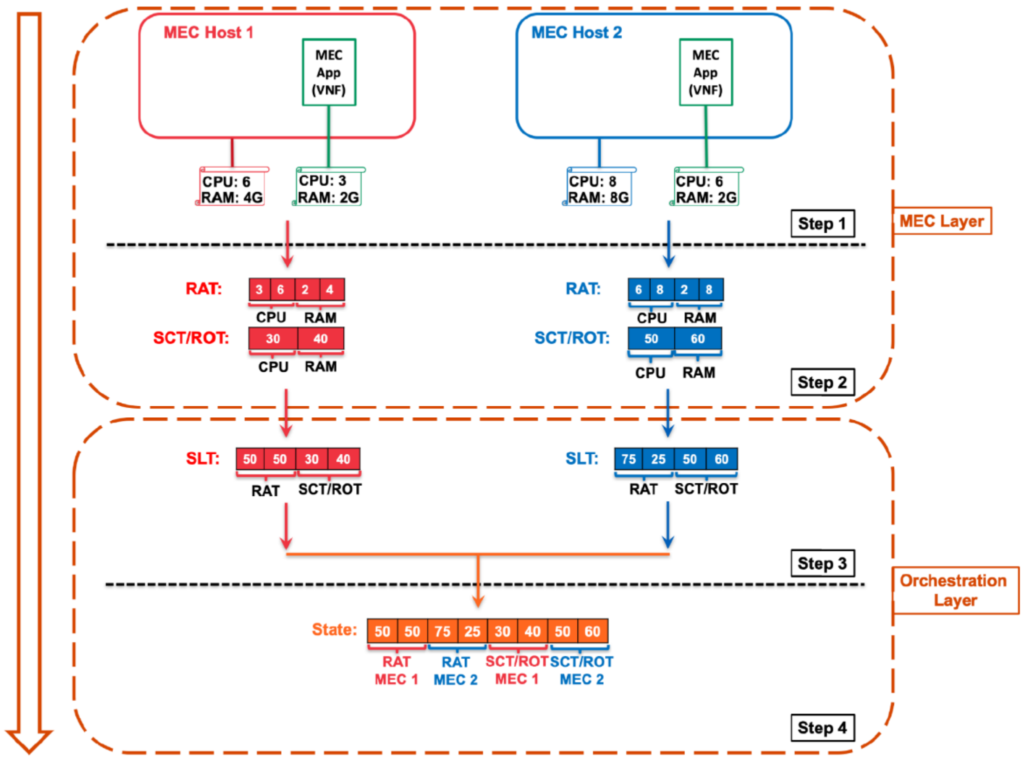


Figure 31: Triggers for RL agent [67].

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 32 shows the result of two agents, the first is based on DQN and the second on A2C. In Figure 32, 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.

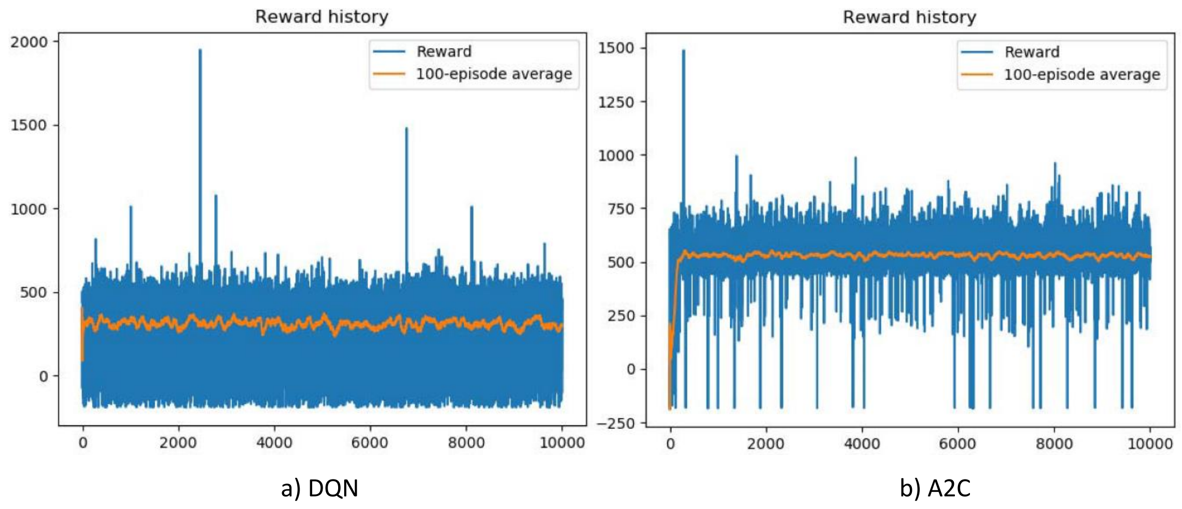


Figure 32: Reward history of SMDM agents [67].

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 others can be also leveraged for the service migration such as DDPG, TRPO, or PPO. The next step is to investigate other models and also tailor the triggers specifically towards XR services.

6.2 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 [69] that was done in this vein. A system is proposed to host the RL agents. The proposed system, depicted in Figure 33, 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.

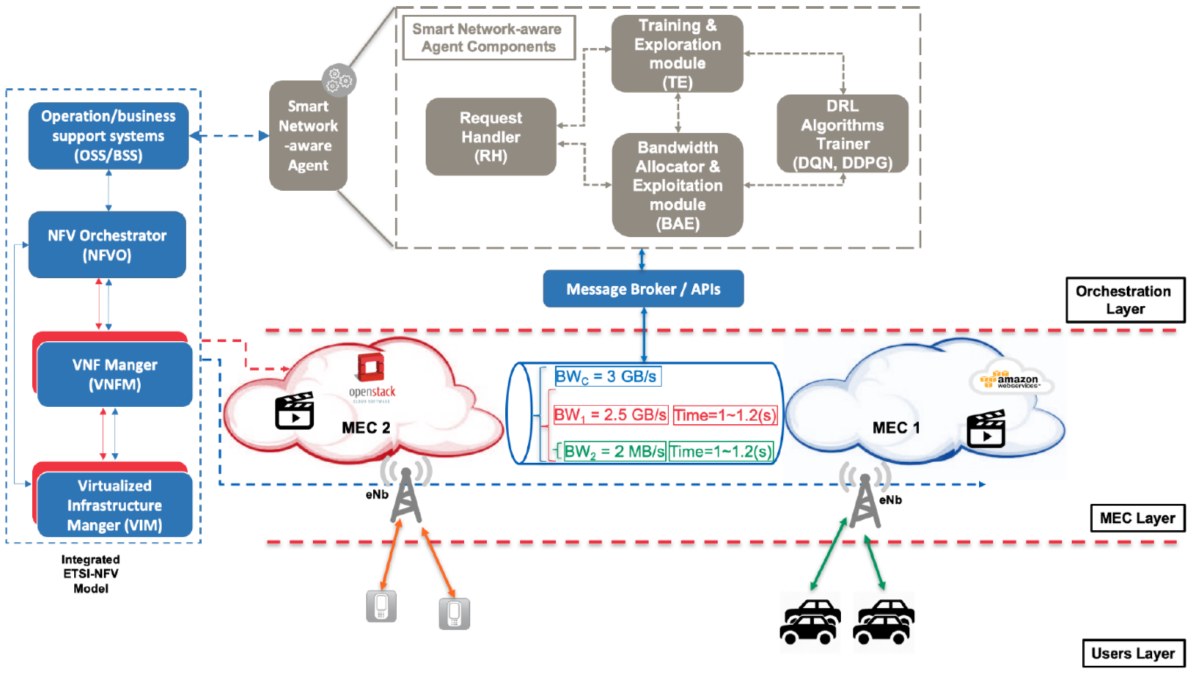

Smart Network-aware Agent Components
Training & Exploration module (TE)
Request Handler (RH)
DRL Algorithms Trainer (DQN, DDPG)
Bandwidth Allocator & Exploitation module (BAE)
Smart Network -aware Agent
Operation/business support systems (OSS/BSS)
NFV Orchestrator (NFVO)
VNF Manger (VNFM)
Virtualized Infrastructure Manger (VIM)
Integrated ETSI-NFV Model
Message Broker / APIs
Orchestration Layer
openstack
MEC 2
BW_C = 3 GB/s
BW_1 = 2.5 GB/s
Time=1~1.2(s)
BW_2 = 2 MB/s
Time=1~1.2(s)
amazon webservices
MEC 1
eNb
eNb
MEC Layer
Users Layer

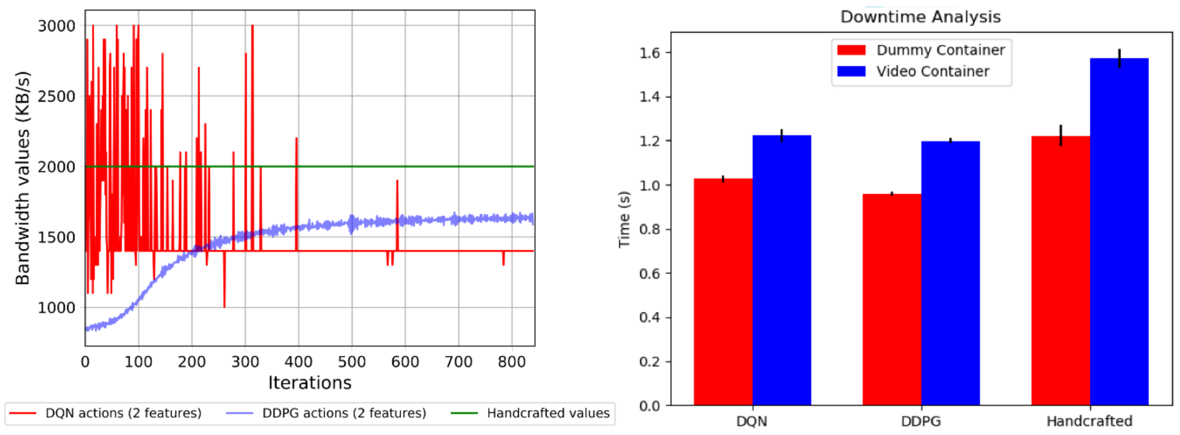
Figure 33: Architecture of a network aware service migration orchestrator [69].

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 [70]. 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 34-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 34-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 34: Performance evaluation [69].

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 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.

7.1 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.

7.1.1 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 [71], 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 [71], the authors proposed a taxonomy with five main categories: *Data Protection, Input Protection, Output Protection, User Interfaces and Device Protection* (Figure 35), 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.

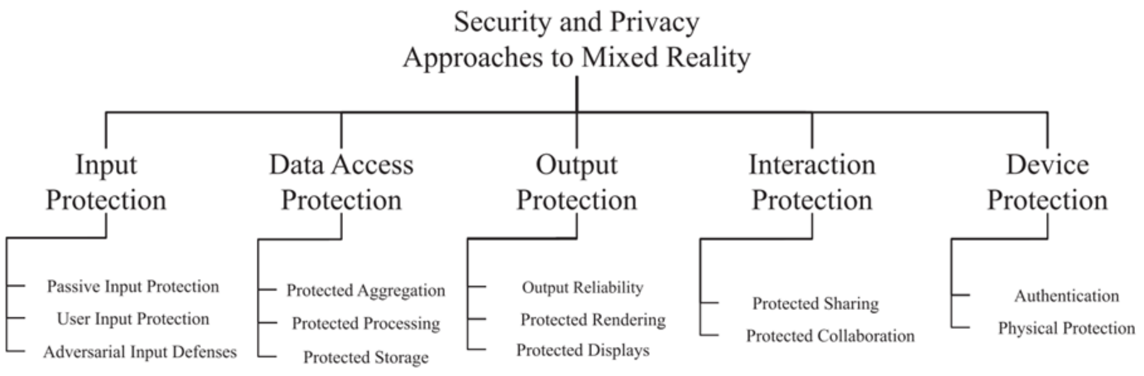


Figure 35: Mapping security and Privacy approaches to XR applications [71].

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.

7.1.2 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 [72] 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.

7.1.3 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 [73].

In such environments, no default trust should occur. Instead, adopting the principle of least privilege (POLP) [74] 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 no 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) [75].

According to NIST, the Zero Trust architecture must follow the basic zero trust principles [76]: 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 [76].

Figure 36 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).

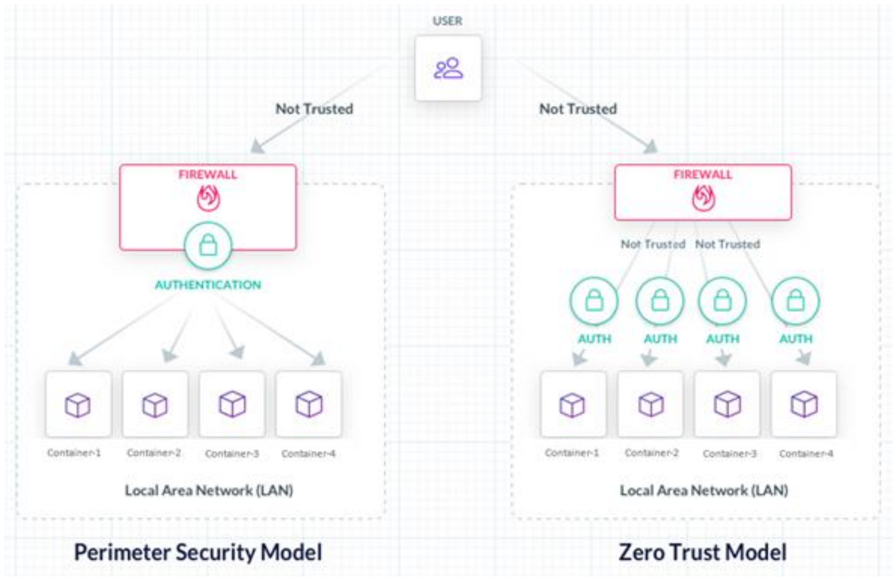


Figure 36: Comparison between Perimeter Security Model and Zero Trust Model, adapted from[30].

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”.

7.1.4 Service Mesh

Service mesh is an increasingly widely used solution to address the challenges of complex and exponential relationships between components [77]. 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.* [51] 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 [77]. 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 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.

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

- **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.
- E**ncryption, 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 [78]. As shown in Figure 37, 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.

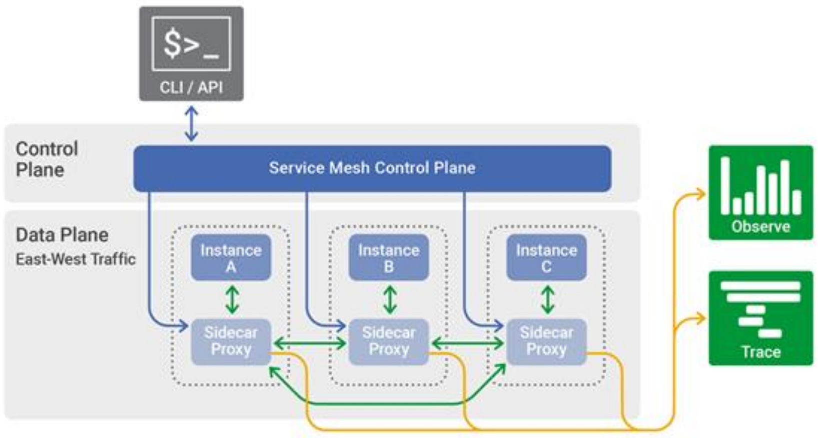


Figure 37: Data and Control planes of a Service Mesh[31].

Although this approach has many benefits, they require further investigation. Chandramouli *et al.* [79] 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 [80] discussed this problem for detecting web-based attacks on micro-services using Kubernetes and Istio. Similarly, in [81], 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).

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

Two essential widely-used tools that enable the service mesh concept are Istio and Envoy proxies. Istio[32] 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[33]. 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.

7.1.5 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.* [82] 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)[34] 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.

7.1.6 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 [83, 84]. For XR applications, 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 [85].

[32] https://istio.io/

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

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

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.

7.2 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 7.1.1, 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[35]: 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 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

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

to improve security against threats. Then, tools such as Kube-bench[36] 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[37], 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[38] or Falco[39] 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[40] or Falco[41] 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 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[42], 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[43]) 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

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

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

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

[39] https://falco.org/

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

[41] https://falco.org/

[42] https://checkmarx.com/

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

strings or exposed private keys). Likewise, tools such as WhiteSource Bolt[44] or Black Duck[45] are pivotal in searching for known vulnerabilities.

The following 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.

7.2.1 Autonomic and Cognitive Security Management Framework

Modern 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 38. 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.

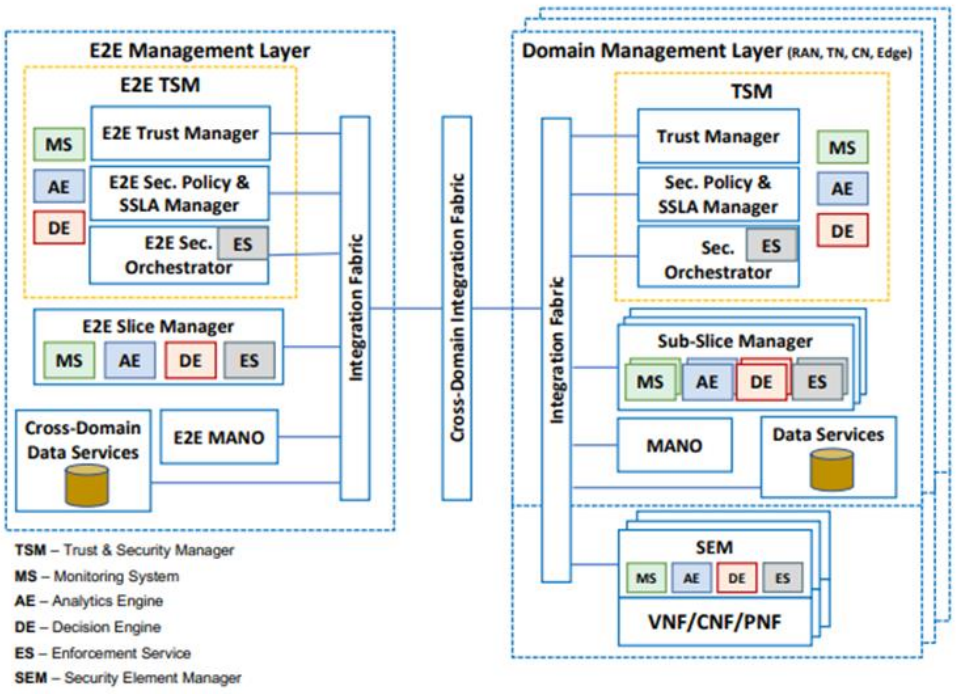


Figure 38: Architecture of envisioned Autonomic and Cognitive Security Management framework [86].

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-

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

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

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 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).

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[46] 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.

In the context of management and orchestration platforms, Open Network Automation Platform (ONAP)[47] and Open Source MANO (OSM)[48] 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.

[46] https://linkerd.io/

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

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

To provide services in AI and analytics, Platform for Network Data Analytics (PNDA)[49] and Acumos AI Platform[50] 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 39 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., 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.

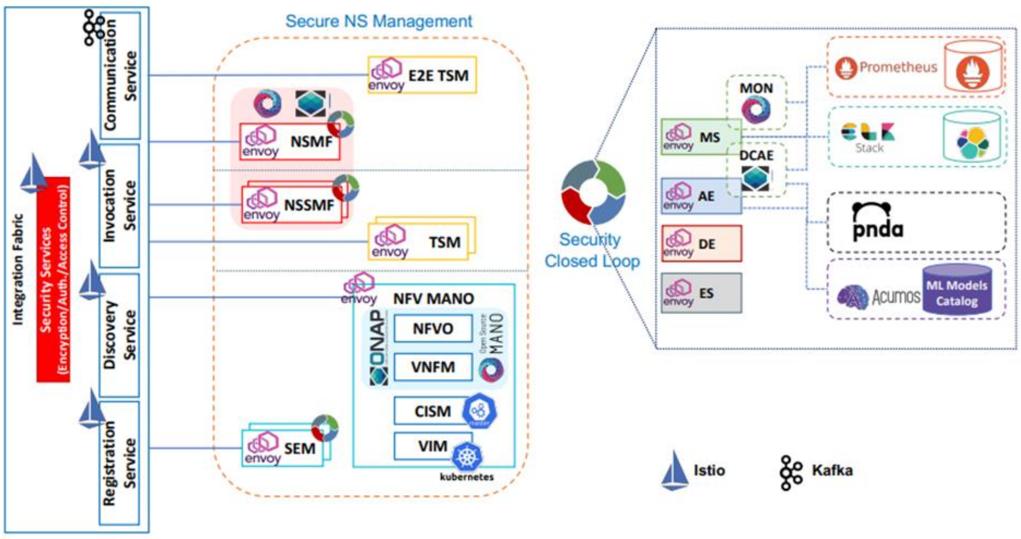


Figure 39: Architecture of the framework with the enablers and tools [86].

7.2.2 Cloud Native Anomaly Detection and Mitigation

As discussed before in section 7.2, 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

[49] http://pnda.io/

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

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 40 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.

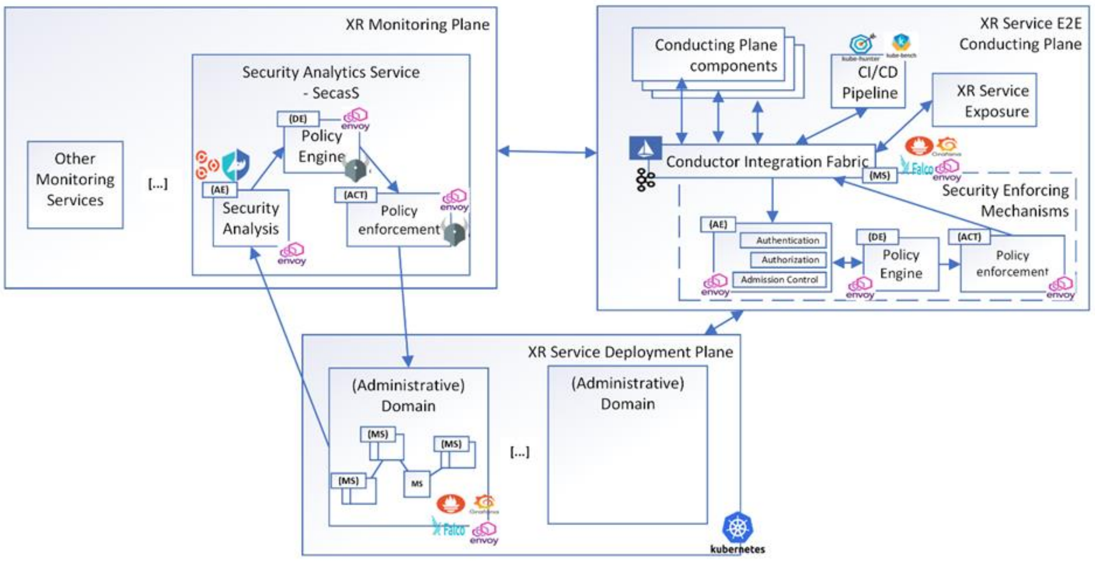


Figure 40: Mapping open-source enablers for network security in CHARITY architecture.

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[51] 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

[51] https://www.nfstream.org/

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 41 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.

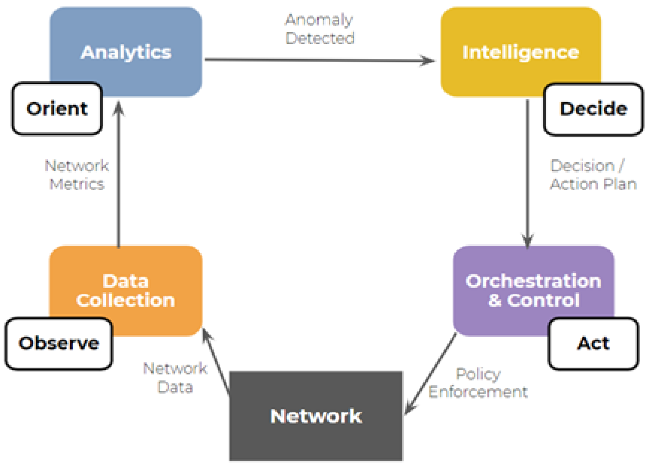


Figure 41: Network anomaly detection and mitigation functions mapped into OODA loop and closed-loop pattern.

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, 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 42. 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).

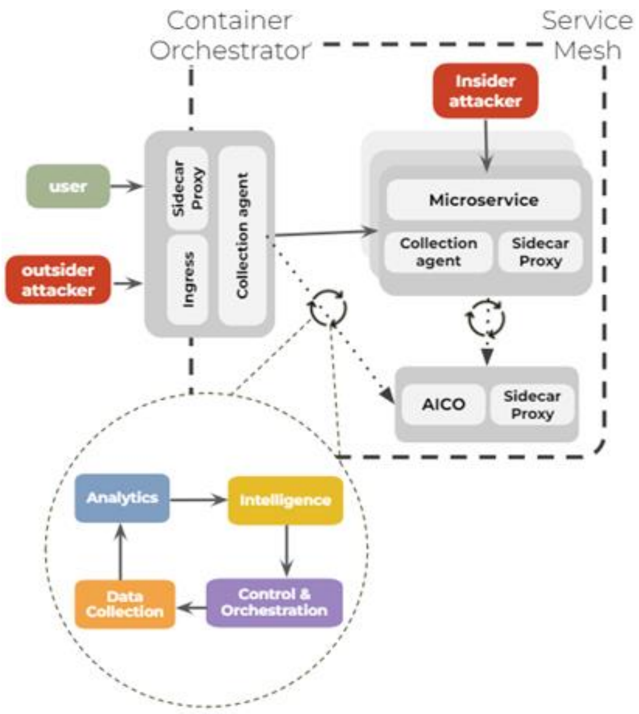


Figure 42: Reference architecture of the proposed framework.

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 43 depicts the practical use case scenario.

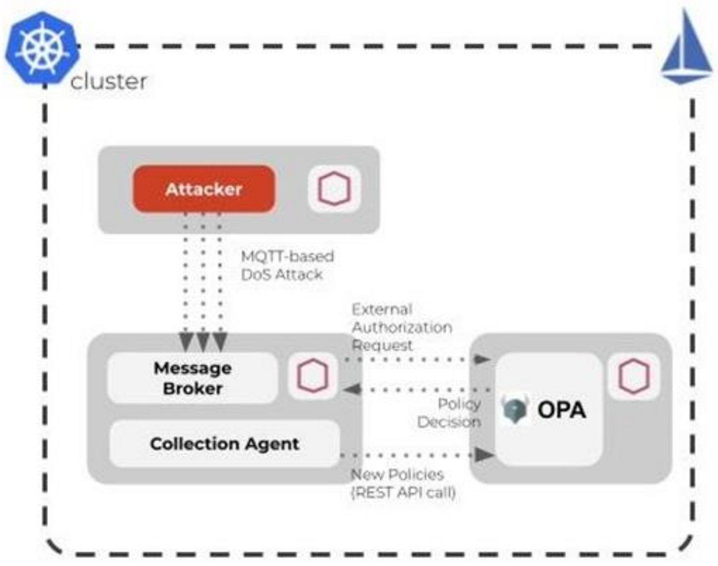


Figure 43: Experimental Use Case scenario.

A DoS attack against the message broker was generated to evaluate the architecture's behaviour and reaction. Figure 44 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.

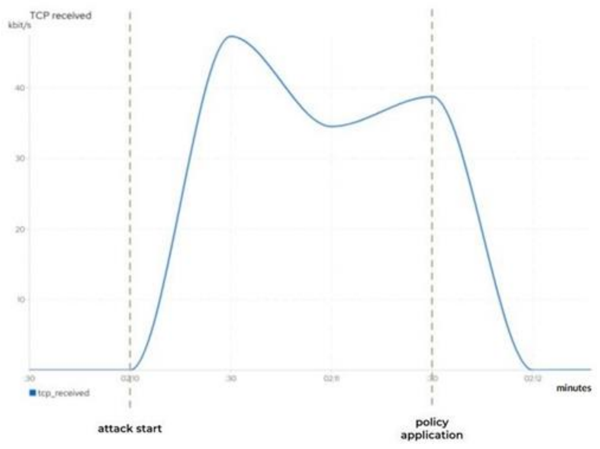


Figure 44: Evolution of TCP connections registered in the message broker.

Using the discussed open-source enablers made it possible to conclude that threats can be effectively stopped with the proposed approach.

7.3 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.

7.3.1 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.

7.3.2 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. [87], 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 [87], Trassare et al.'s solution [88] and LinkBait [89]. 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.

7.4 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 8). 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. Their selection will be made at a later stage, based on aspects like license type, capabilities, open project activity and maturity, documentation, and possibility to be extended. At the moment, we can outline the main areas that are a candidate 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, if provided, might very likely be made 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 (see section 8).

- Static container image security testing:

The container images of application microservice components are statically scanned, searching for known vulnerabilities, typically by parsing through image packages or other dependencies.

Automated static application testing combined with container testing techniques allow the detection of a broad range of different vulnerabilities. They can be tightly integrated with the basic DevOps pipeline or put aside as an additional set of tools. A decision on this point will be taken at a later stage of the project.

Regarding the security aspects for the Application Management Framework, a centralized Authentication and Authorization mechanism based on the OpenID-connect[52] protocol (an identity layer on top of the OAuth 2.0[53] 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.

[52] https://openid.net/connect/

[53] https://oauth.net/2/

8 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 will be 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 will be 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 45 depicts the CHARITY high level architecture, in which the AMF framework is highlighted.

Interaction with Network Service Orchestrator will be implemented in a loosely coupled way. At the moment, the candidate mechanisms are message queue publish/subscribe model (e.g., exploiting Apache Kafka) or REST APIs.

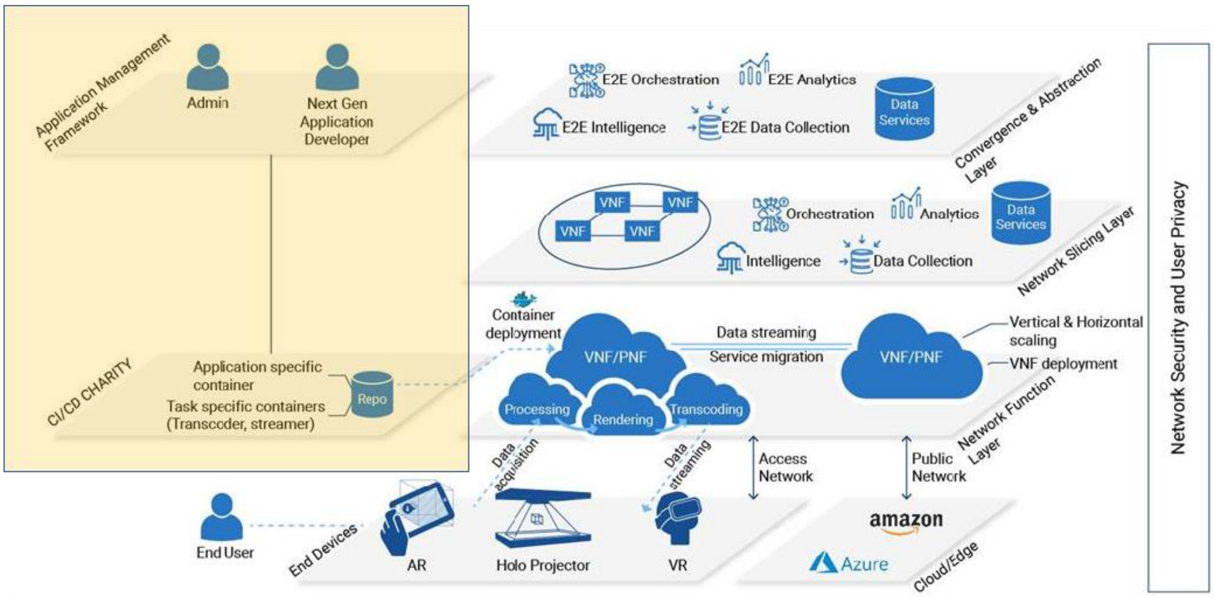


Figure 45: CHARITY High Level Architecture (as initially envisioned in the description of work).

The full functional definition of the AMF is still in progress and might have evolutions in the coming phase. We outline the current vision of 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 artifacts), in a repository where they can univocally be registered with an abstract description of the artifact 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 artifact is available and tagged in the artifact repository.

Whenever possible, each described artifact (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 artifact, the owner can request to deploy, renew the deployment with a new version, and to decommission the artifact. 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 46 represents the above-mentioned interactions allowed to the XR Developer for the definition and management of the developed artifacts.

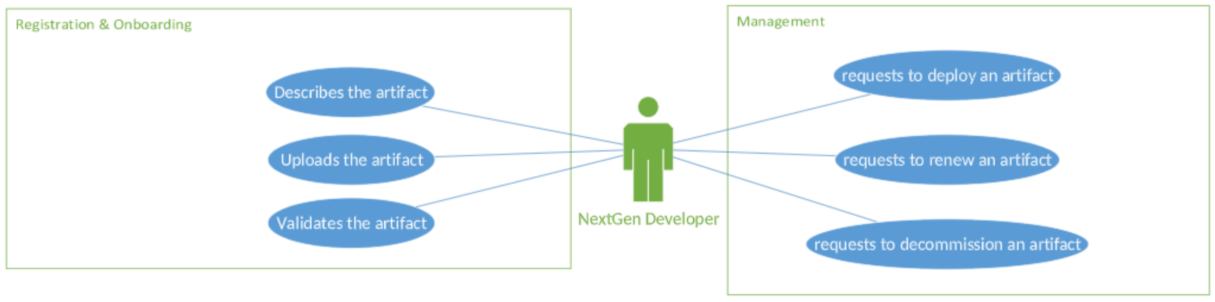

Registration & Onboarding
Describes the artifact
Uploads the artifact
Validates the artifact
NextGen Developer
Management
requests to deploy an artifact
requests to renew an artifact
requests to decommission an artifact

Figure 46: Developers' Activities.

After the NextGen Application Developer decides to deploy/decommission an artifact, the Application Management Framework alerts orchestration components that an artifact is ready to be deployed-renewed/decommissioned. Mechanisms will be investigated to pass run-time deployment parameters not included in the abstract application model e.g., geo-location parameters. The interface between AMF and Network Service Orchestrator is expected to be either based on a message queue mechanism, e.g., Apache Kafka or based on REST API calls. Figure 47 depicts the high-level deployment sequence.

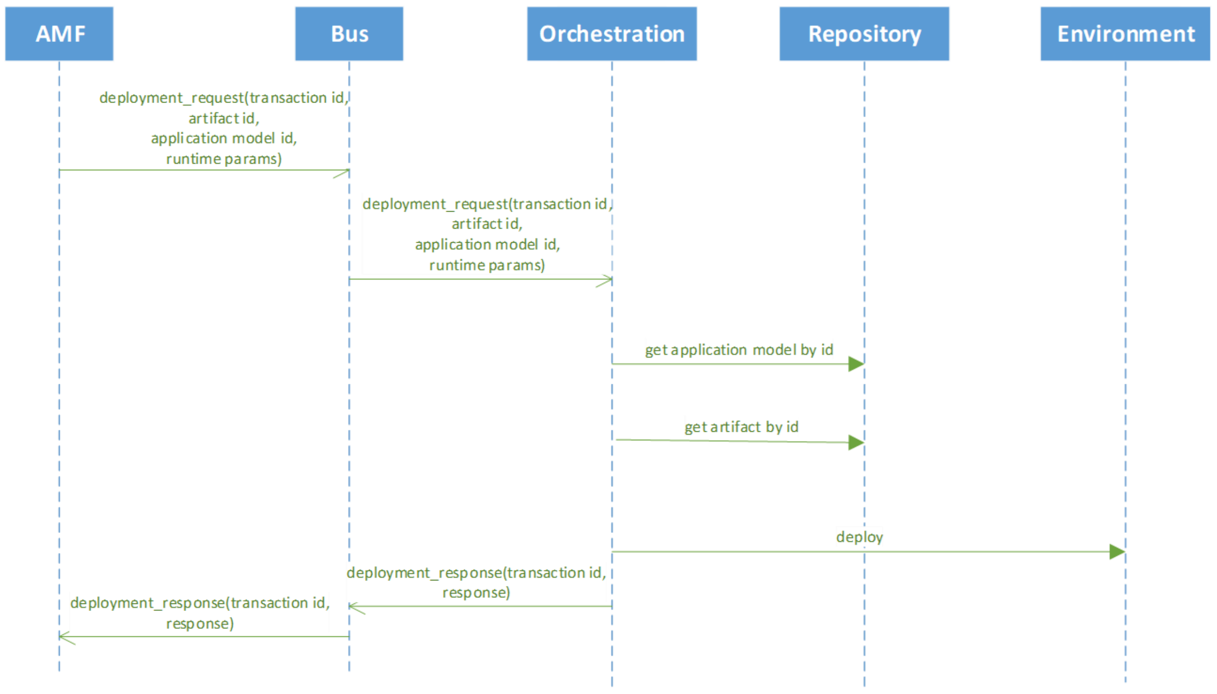

AMF
Bus
Orchestration
Repository
Environment
deployment_request(transaction id, artifact id, application model id, runtime params)
deployment_request(transaction id, artifact id, application model id, runtime params)
get application model by id
get artifact by id
deploy
deployment_response(transaction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

Figure 47: 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 artifacts -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 artifacts 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 in a NSL catalogue (XR Service Blueprint Templates Repository) that can be accessed by the orchestration layer (see Figure 48).

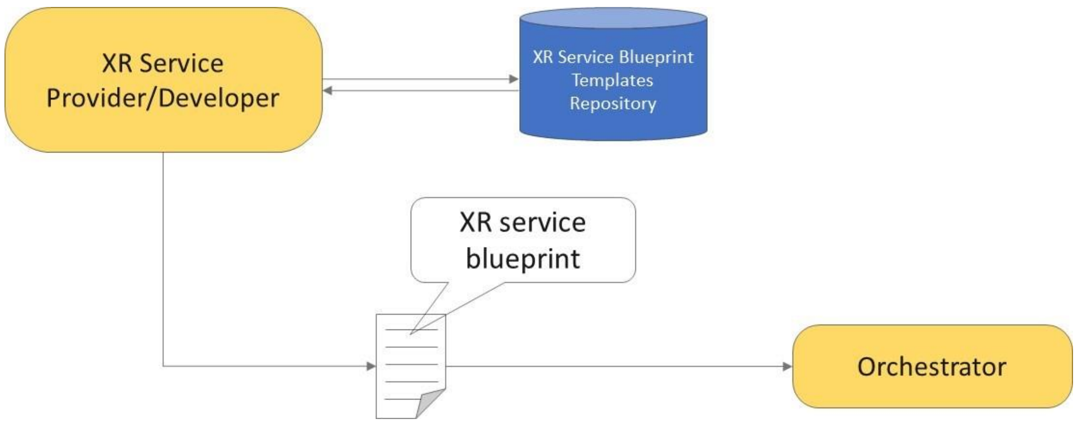

XR Service Provider/Developer
XR Service Blueprint Templates Repository
XR service blueprint
Orchestrator

Figure 48: AMF and network blueprints.

Blueprints do not contain runtime objects or artifacts; 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.

Regarding the components that will build up the CI/CD pipeline, they will be selected preferably among available, widely recognized open tools (e.g., Jenkins and Anchore). The exact specification of the pipeline elements will be nailed down once the detailed design of the pipeline steps will be completed.

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 sub-section 5.1.4 on Monitoring architecture.

8.1 XR Application Management Framework Architecture

The AMF relies on the architectural components depicted in Figure 49 and described in the following:

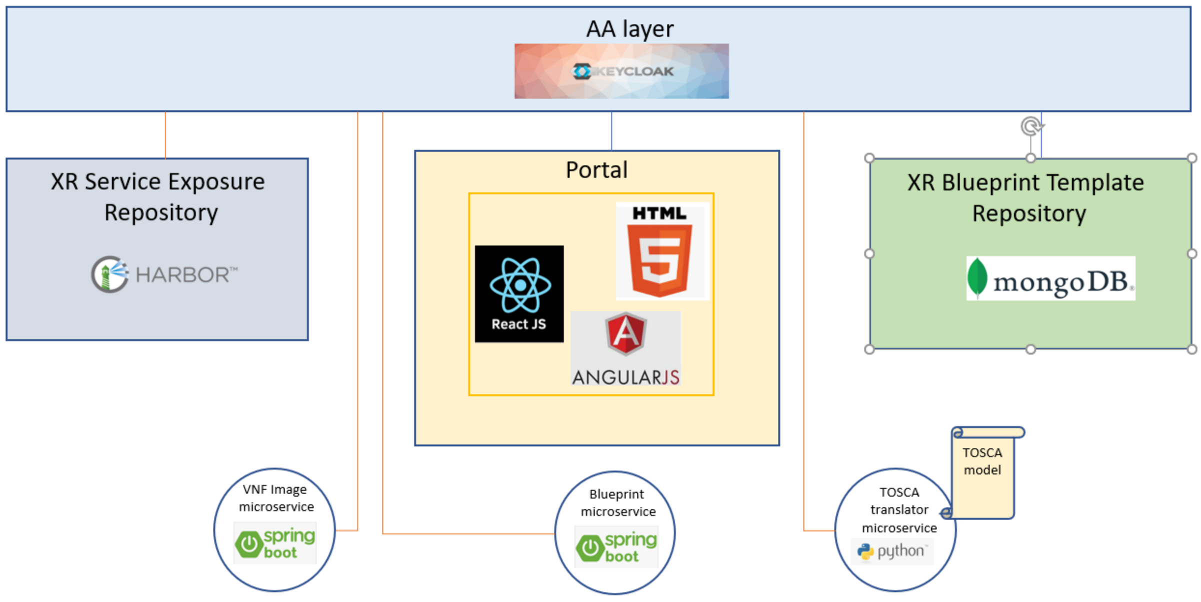


Figure 49: Architectural components of the XR Application Management Framework.

- 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 artifacts 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. Currently, 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 planned, dedicated to the translation at runtime or a upon request of a blueprint template into the TOSCA representation expected by the orchestration layer.
- 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.

8.2 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[54] 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 50, 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.

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

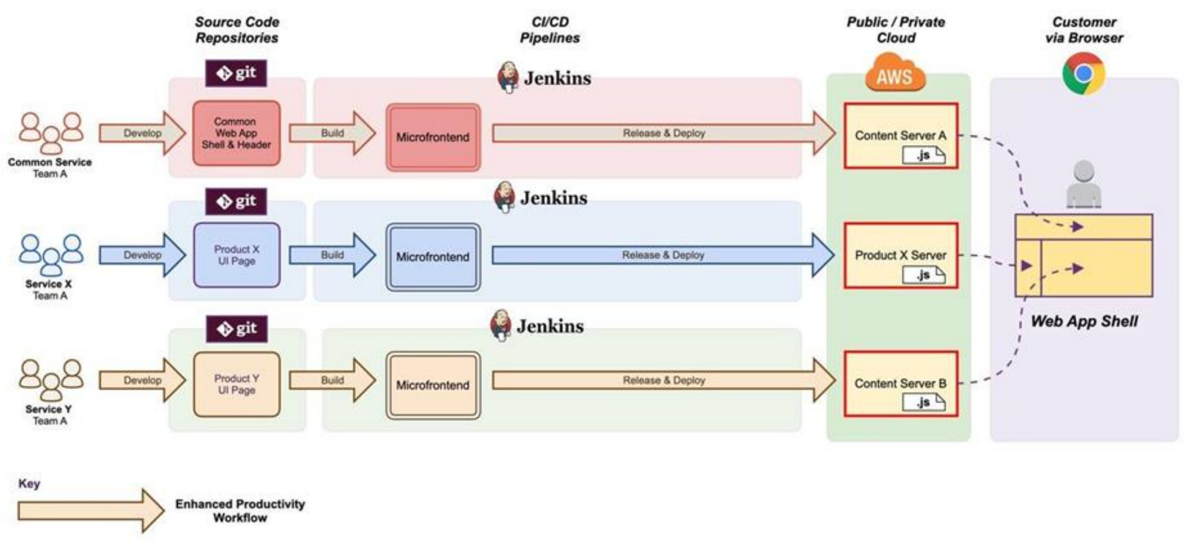


Figure 50: How a Micro frontend architecture enables independent end to end development workflows55.

To implement the CHARITY Micro Frontend architecture, a set of tools and technologies based on Html and JavaScript have been selected (Figure 51), namely:

- AngularJS 12 for the visualization layer
- Webpack 5, a static module bundler for modern JavaScript applications, specifically the Webpack5 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.

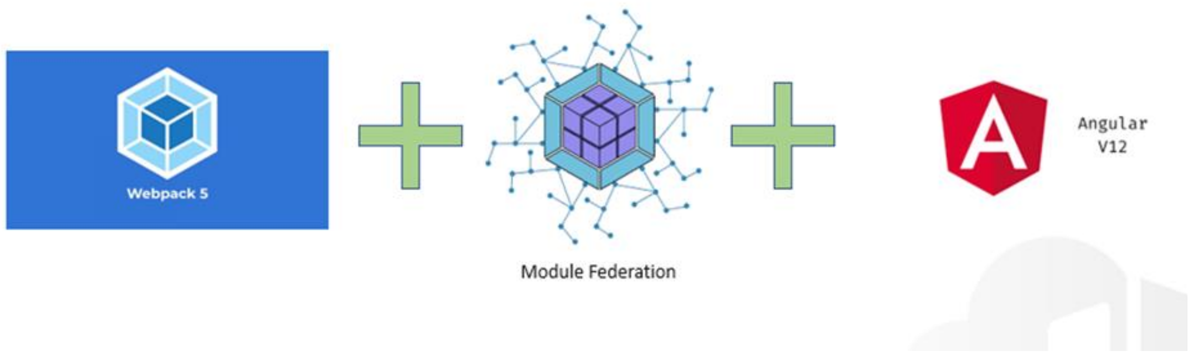


Figure 51: Technologies allowing Micro Frontends in CHARITY.

In the following we show, with the help of screenshots, the AMF Portal functionalities implemented so far.

The entry point to the platform is the Login page (Figure 52). 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 (sub-section 7.4) 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

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

token is self-contained, meaning that it contains, among many pieces of information, also declarations about the roles of the callers.

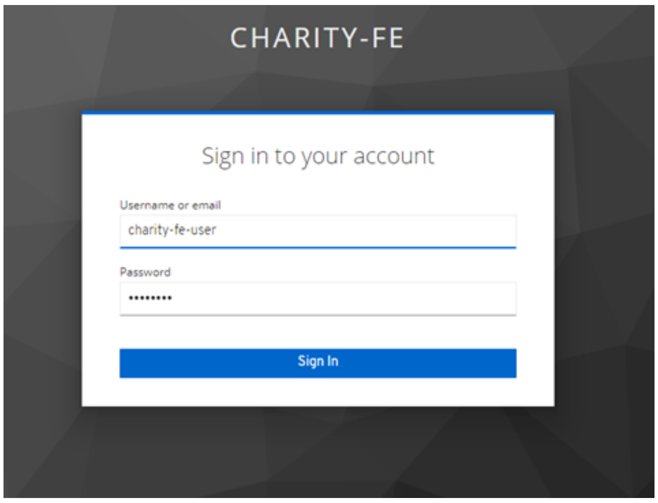


Figure 52: Login page mediated from the openid-connect authentication provider.

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 53).

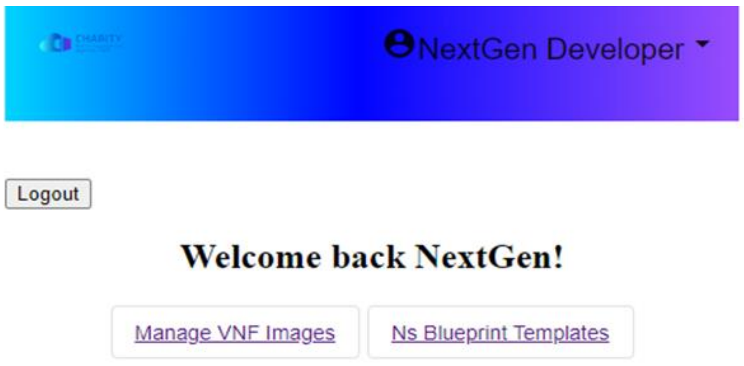
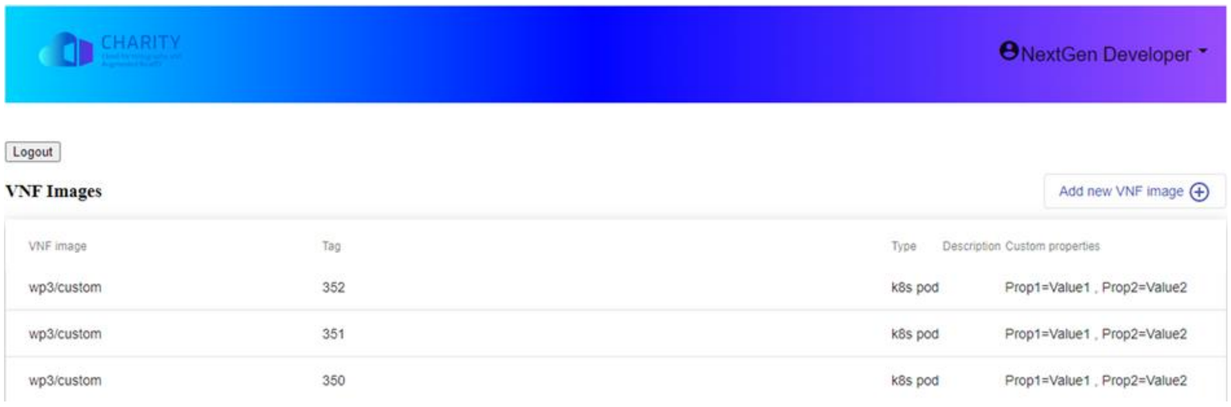


Figure 53: Landing page for an XR Developer.

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

- List (Figure 54) the VNF images available in the system (private to the developer or public for their organization).

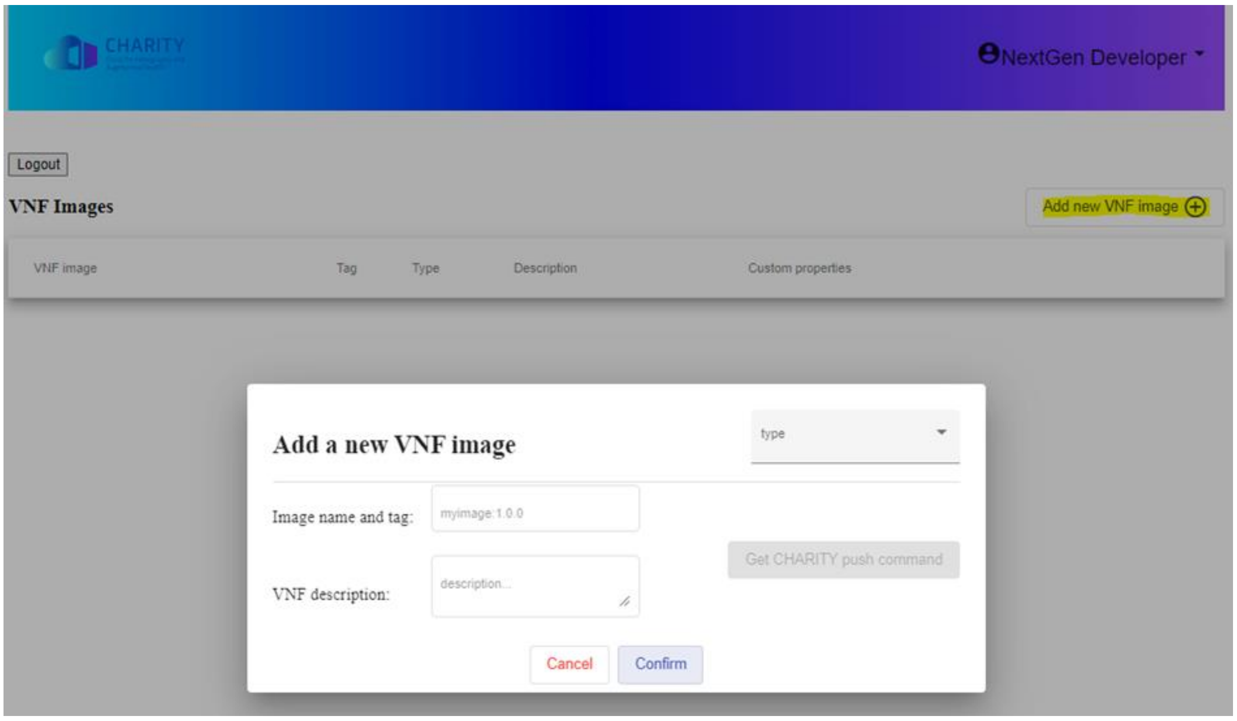


VNF Images

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 54: List of VNF images available.

- Upload (Figure 55) 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.



Add a new VNF image

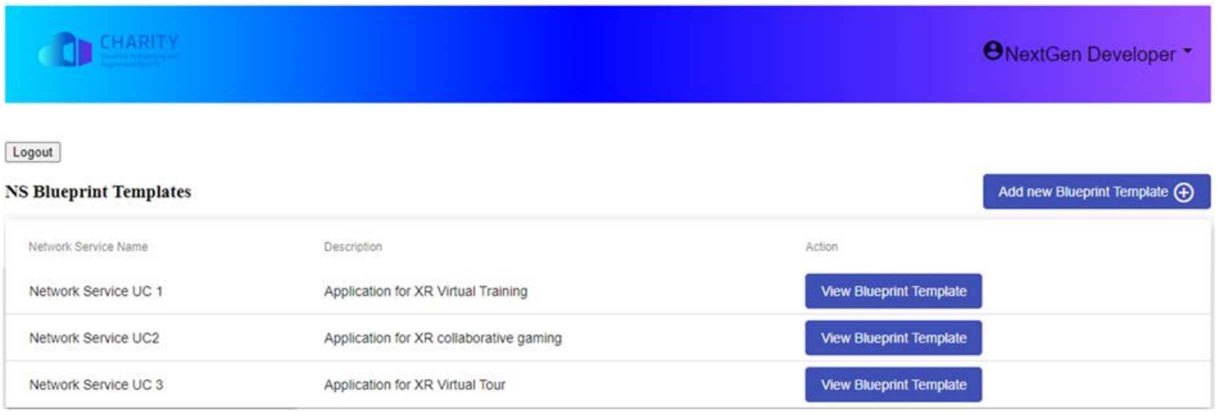
Image name and tag:	myimage:1.0.0
VNF description:	description...

Figure 55: 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 56) 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

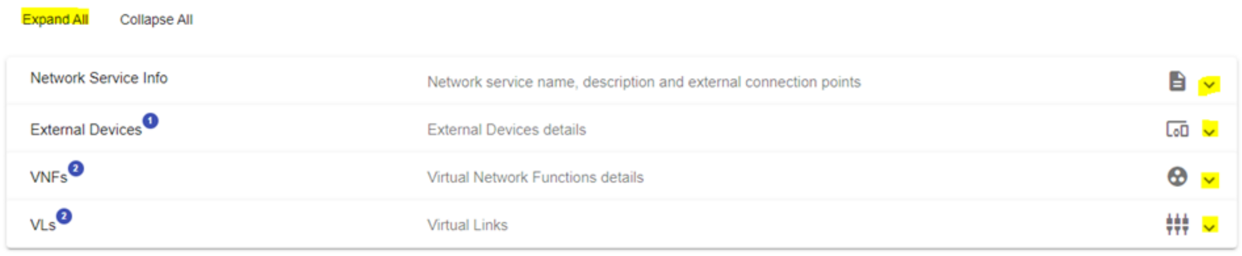
Network Service Name	Description	Action
Network Service UC 1	Application for XR Virtual Training	View Blueprint Template
Network Service UC2	Application for XR collaborative gaming	View Blueprint Template
Network Service UC 3	Application for XR Virtual Tour	View Blueprint Template

Figure 56: 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.

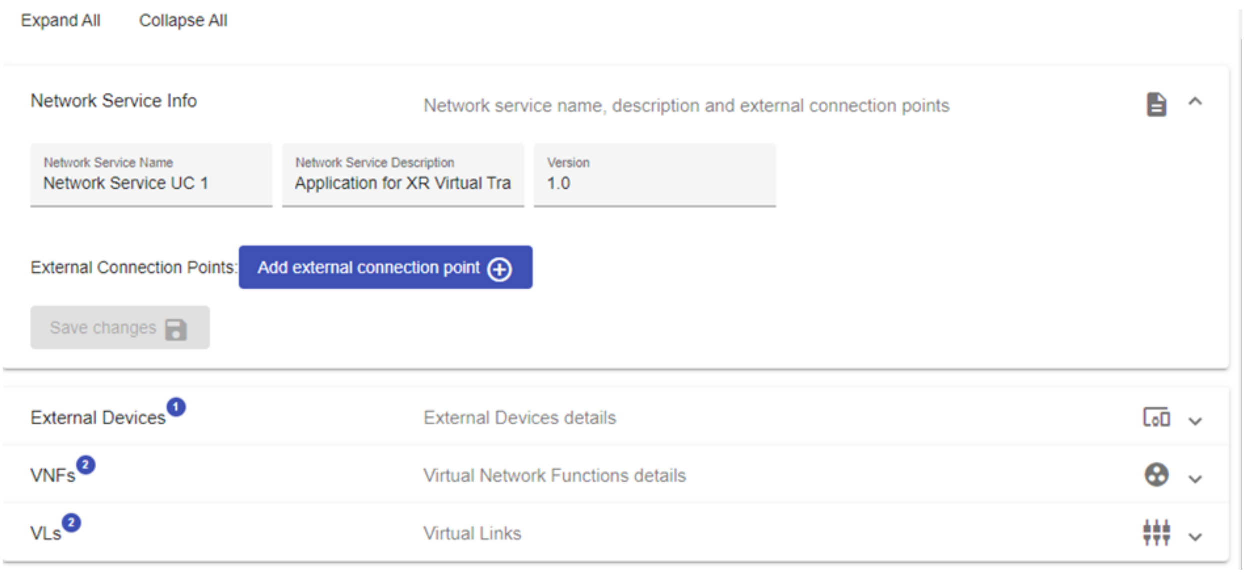
Figure 57 depicts the "summary" view that the user gets when selecting a Blueprint Template from the list. The GUI shows only top-level components, and for each of them, the number of defined elements (blue numbered icon) and an arrow for expanding into the details. At the same time, it is also possible 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 57: Summary view ("Collapsed") of a Blueprint Template definition.

By expanding the "Network Service Info" panel (Figure 58), the user can edit general information such as the service name, a description, and a version number. They also can define connection points used for describing communication ports.



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

Network Service Name: Network Service UC 1
Network Service Description: Application for XR Virtual Tra
Version: 1.0

External Devices 1	External Devices details
VNFs 2	Virtual Network Functions details
VLs 2	Virtual Links

Figure 58: Editing of the NS general information.

The next panel (Figure 59) 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.

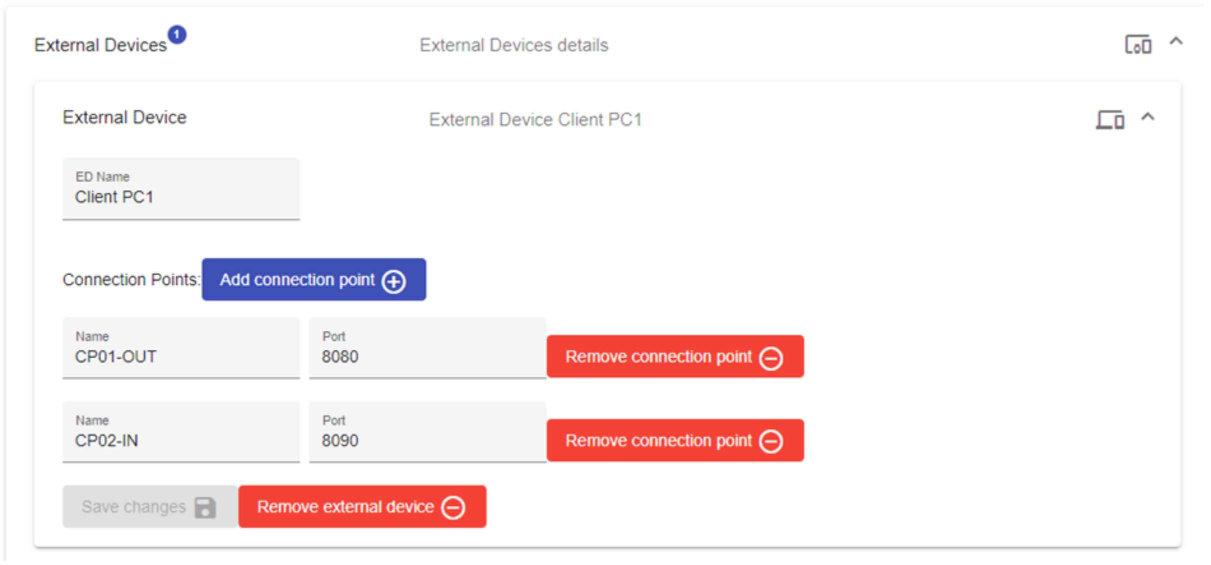
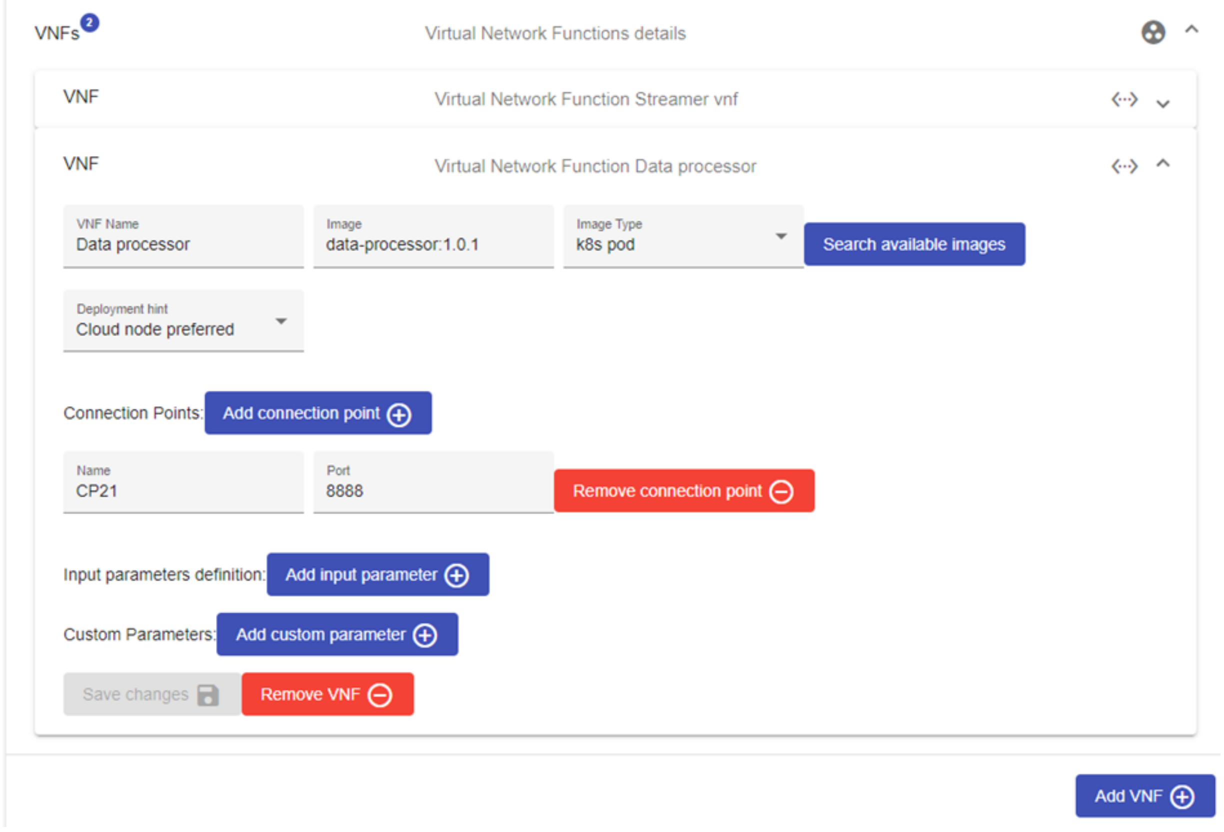


Figure 59: Editing of the NS “External devices” Section.

The “VNFs” panel (Figure 60) 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.


VNFs 2 Virtual Network Functions details

VNF Virtual Network Function Streamer vnf

VNF Virtual Network Function Data processor

VNF Name: Data processor
Image: data-processor:1.0.1
Image Type: k8s pod

Deployment hint: Cloud node preferred

Connection Points:

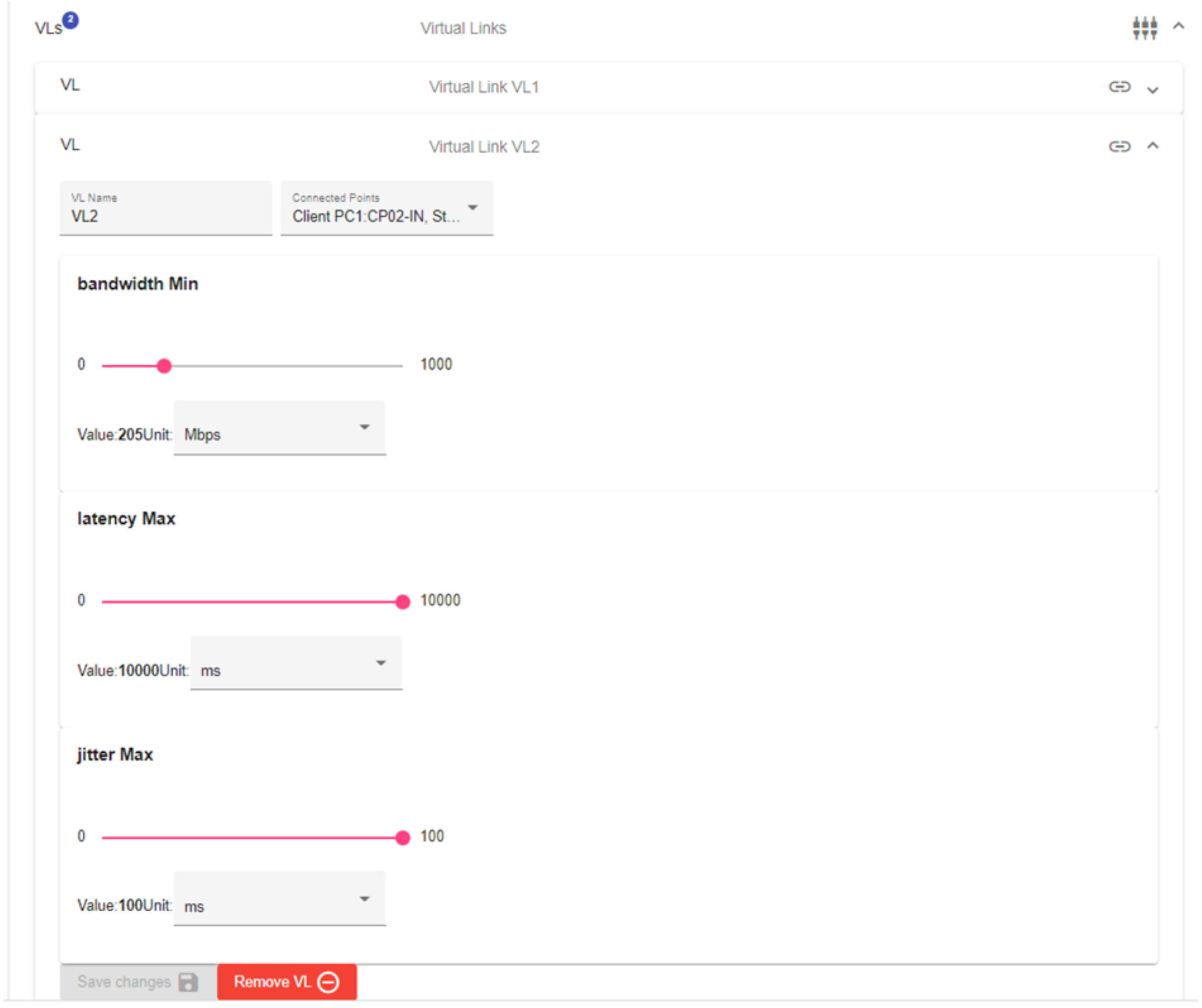
Name: CP21
Port: 8888

Input parameters definition:

Custom Parameters:

Figure 60: Editing of the VNFs section.

The “Virtual Links” panel (Figure 61) 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.



VL Name: VL2
Connected Points: Client PC1:CP02-IN, St...

bandwidth Min

0 — 1000

Value:205 Unit: Mbps

latency Max

0 — 10000

Value:10000 Unit: ms

jitter Max

0 — 100

Value:100 Unit: ms

Figure 61: Editing of the Virtual Links section.

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 62: 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.

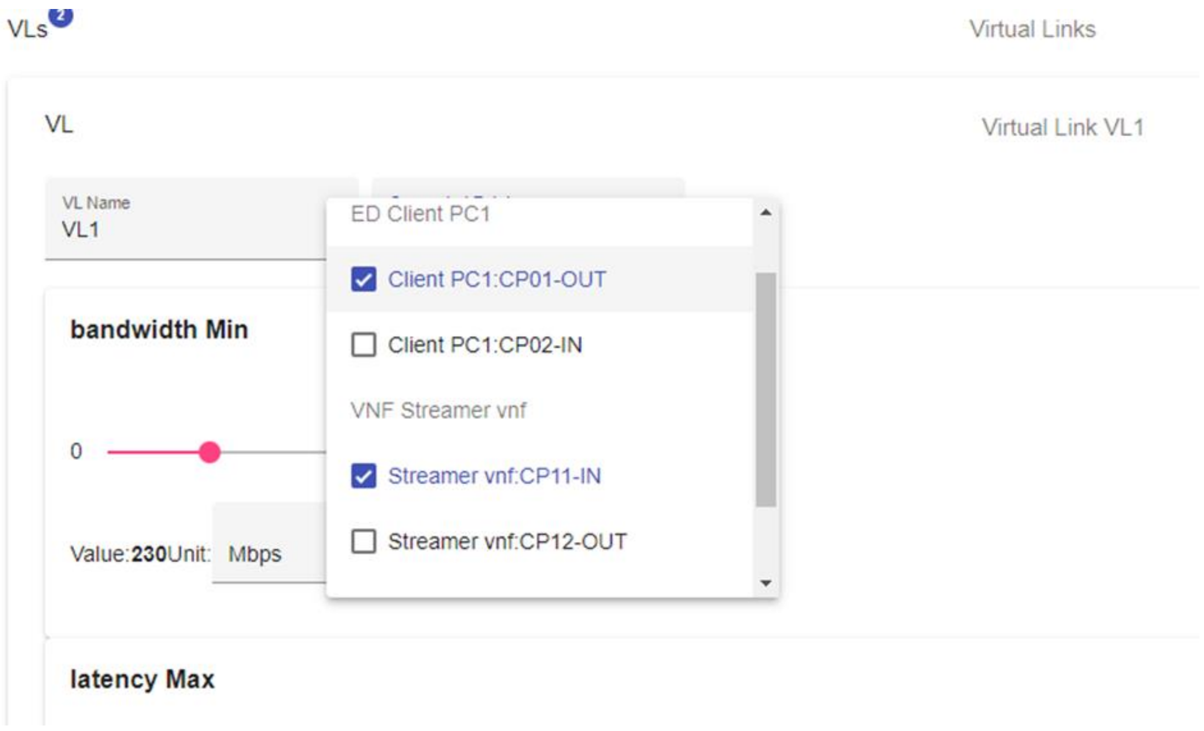


Figure 62: Detail of the multi-select drop down list for specifying Virtual Link connectivity.

8.3 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.

As mentioned in sub-section 8.1.2, 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. In addition to container images, it might be necessary to provide access to VNF represented by VM images, that would be executed on runtime nodes similarly to container, leveraging the Kubevirt.io[56] plugin technology for Kubernetes.

To support this use case, the XR Service Enabler Repository will also provide access to VM images and the underlying technology for this enablement is currently under evaluation. Options are:

- VM images stored in a file system, and made available via a REST API provided by the XR service Enabler Repository.
- VM images stored in an object storage, and made available either via a REST API provided by the XR service Enabler Repository or via the native API of the object storage itself.
- VM images properly packaged and wrapped into a container image representation[57], and made available in a transparent way as if they were container images.

[56] https://kubevirt.io/

[57] https://kubevirt.io/2020/Customizing-images-for-containerized-vms.html

The decision on the most suitable option is out of scope for this document and depends on outcomes of other work packages.

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.

8.4 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 (sub-section 8.6).

8.5 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.

8.5.1 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.

8.5.2 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.

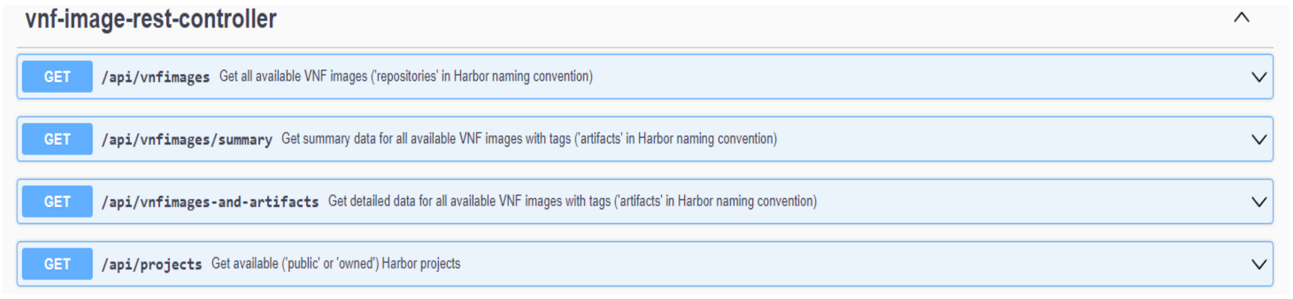
8.5.3 VNF Image microservice

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 (sub-section 8.3). The core of a VNF Image is the container or VM image 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 or VMs in the CHARITY Cloud or Edge nodes.

Figure 63 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 63: Sample list of REST 'GET' operations provided by the VNF image microservice.

In the next implementation cycles, additional features are going to be added, such as the enrichment of VNF Image information with additional metadata and the storage of such representation into a private MongoDB instance.

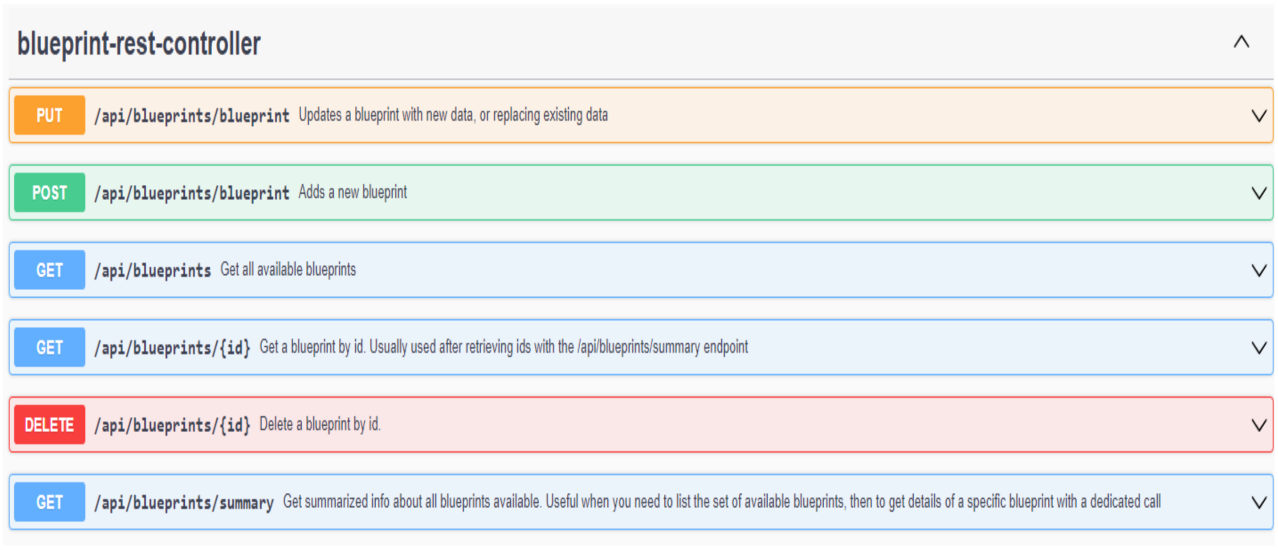
8.5.4 Blueprint microservice

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 64 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.


blueprint-rest-controller

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

Figure 64: Endpoints of the Blueprint microservice REST API.

8.6 TOSCA model for blueprint templates

As introduced in sub-section 8.1, the TOSCA Model allows to define an application Blueprint Template through a TOSCA-compliant representation[58].

The Model has been defined taking into account the capabilities of the TOSCA specification, focusing in particular on the Tosca Simple Profile sub specification[59].

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

"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 specification 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.

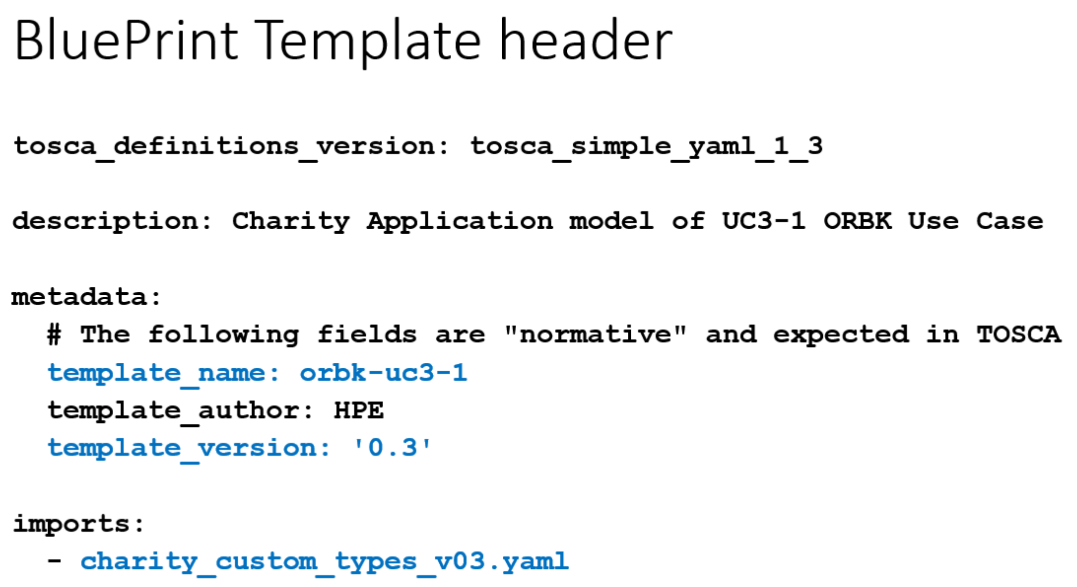
Leveraging the support for the YAML representation of the TOSCA specification, we defined such custom types in a "charity_custom_types.yaml" document, to be imported in the TOSCA model according to the specification rules. The contents of the custom type definition are documented in Appendix A.1.

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

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

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

Figure 65 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.



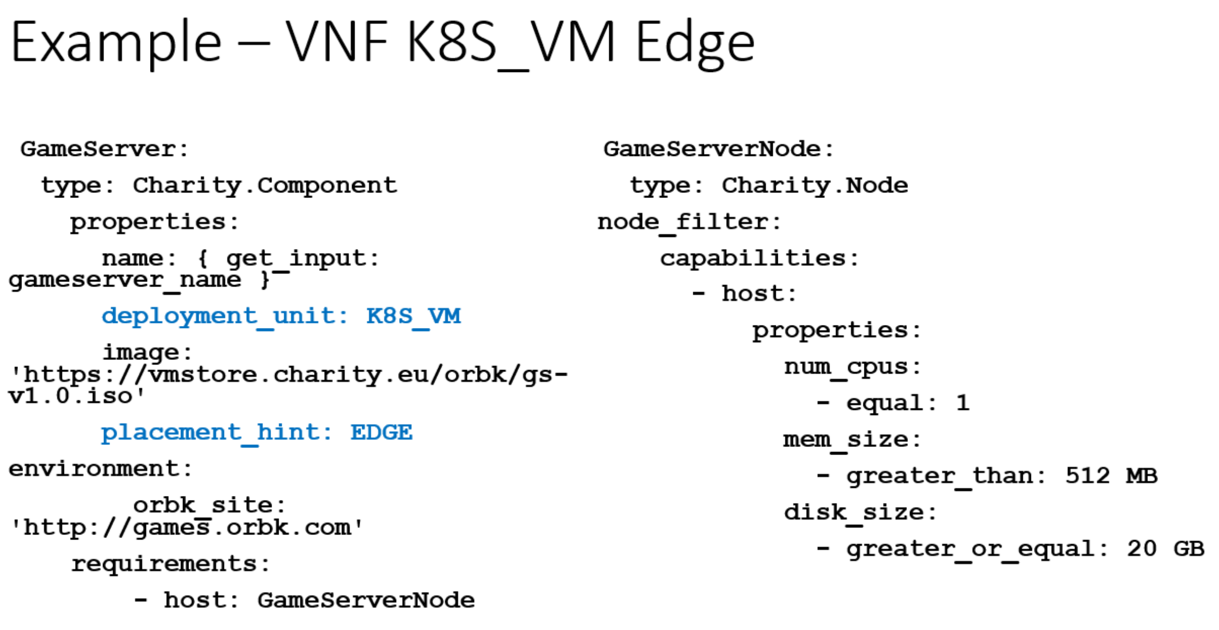
BluePrint Template header

```
tosca_definitions_version: tosca_simple_yaml_1_3

description: Charity Application model of UC3-1 ORBK Use Case

metadata:
  # The following fields are "normative" and expected in TOSCA
  template_name: orbk-uc3-1
  template_author: HPE
  template_version: '0.3'

imports:
  - charity_custom_types_v03.yaml
```

Figure 65: Sample TOSCA Blueprint Template header section.

Figure 66 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, the type of image (in this case a virtual machine), some deployments preferences, and the requirements in terms of runtime node.

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



Example – VNF K8S_VM Edge

```
GameServer:
  type: Charity.Component
    properties:
      name: { get_input:
gameserver_name }
      deployment_unit: K8S_VM
      image:
'https://vmstore.charity.eu/orbk/gs-
v1.0.iso'
      placement_hint: EDGE
environment:
        orbk_site:
'http://games.orbk.com'
    requirements:
        - host: GameServerNode
```

```
GameServerNode:
  type: Charity.Node
node_filter:
    capabilities:
      - host:
          properties:
            num_cpus:
              - equal: 1
            mem_size:
              - greater_than: 512 MB
            disk_size:
              - greater_or_equal: 20 GB
```

Figure 66: Sample representation of a CHARITY Component and a CHARITY Node.

Figure 67 shows an example similar to the previous one, but specifying a container image as VNF image type.

```
MeshMerge:
  type: Charity.Component
  properties:
    name: meshmerge
    deployment_unit: K8S_POD
    image:
'http://harbor.charity.eu/orbk/m
m:v1.0'
    placement_hint: EDGE
  requirements:
    - host: MeshMergeNode
```

```
MeshMergeNode:
  type: Charity.Node
  node_filter:
    capabilities:
      - host:
          properties:
            num_cpus:
              - equal: 4
            mem_size:
              - greater_than: 2 GB
            disk_size:
              - greater_or_equal: 50 GB
```

Figure 67: Another sample representation of a CHARITY Component and a CHARITY Node.

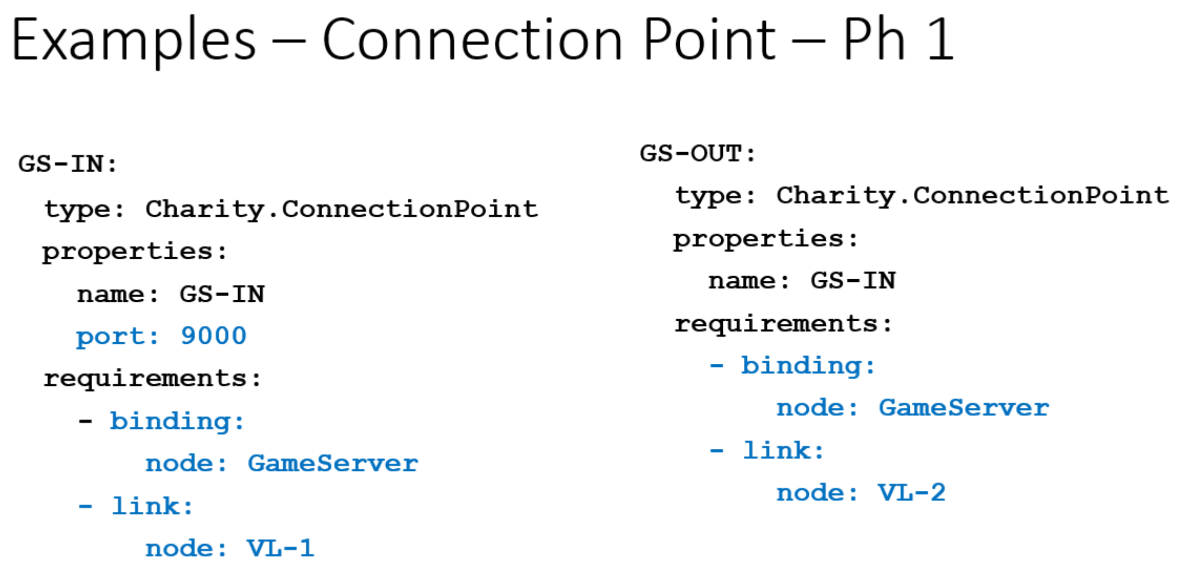
Figure 68 shows how to represent an "external device". This kind of representation may be required to model the interaction with an external system that has to pass some parameters (in the example the location) to the application deployed by CHARITY.

External Device (not deployed by Charity)

```
UserDevice:
  type: Charity.Component
    properties:
      name: userdevice
      deployment_unit: EXTERNAL
      geolocation: { get_input: location }
```

Figure 68: Representation of an external device.

Figure 69 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.



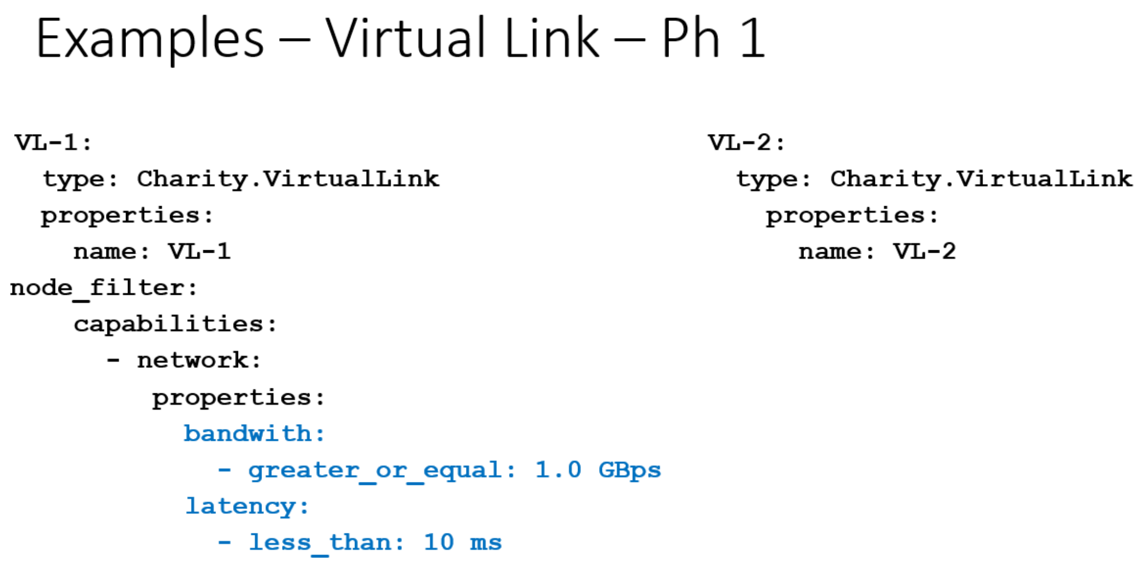
Examples – Connection Point – Ph 1

```
GS-IN:
  type: Charity.ConnectionPoint
  properties:
    name: GS-IN
    port: 9000
  requirements:
    - binding:
        node: GameServer
    - link:
        node: VL-1
```

```
GS-OUT:
  type: Charity.ConnectionPoint
  properties:
    name: GS-IN
  requirements:
    - binding:
        node: GameServer
    - link:
        node: VL-2
```

Figure 69: Representation of connection points.

Figure 70 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.



Examples – Virtual Link – Ph 1

```
VL-1:
  type: Charity.VirtualLink
  properties:
    name: VL-1
node_filter:
    capabilities:
      - network:
          properties:
            bandwith:
              - greater_or_equal: 1.0 GBps
            latency:
              - less_than: 10 ms
```

```
VL-2:
  type: Charity.VirtualLink
    properties:
      name: VL-2
```

Figure 70: Representation of a virtual link with QoS requirements.

8.7 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. At the moment of writing this document, the translator is under design phase and the implementation will follow accordingly. The design principles behind this component are:

- It will be a dedicated microservice, specific for the translation from the JSON representation of a Blueprint Template into a TOSCA YAML representation.
- The translation will 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 will provide a REST API to the Web GUI layer to perform translation requests.

9 Conclusions

This document is the first of the two deliverables of WP2. It shows the advancement of this work package after half of the project duration has passed. It has discussed four core aspect of the CHARITY platform. These are: i) the orchestration framework; ii) the monitoring framework; iii) the security of the platform and XR services; iv) the interface between the CHARITY platform and XR service providers.

For the orchestration framework, the architecture of this framework along with some algorithms such as service scheduling, dynamic routing, and service migration have been introduced. This framework is the heart of the CHARITY platform. Its main focus is twofold. On one side, it abstracts edge and cloud resources into one continuum of resources to be seamlessly consumed by XR services. It does so by logically and dynamically aggregating many Edge Miniclouds (EMs) into clusters. On the other side, it manages the lifecycle of XR applications. This is done by carefully placing the components of an XR application so the KPIs would be met. The KPIs are continuously monitored to preserve the QoE by judiciously adapting the XR service to the condition of the platform. This can be done by dynamically changing the path of data streams, relocating some components, or even allowing the XR service itself to alter its own behaviour concertedly with the orchestrator.

The second aspect, which is about the monitoring framework and computation and networking utilization prediction mechanisms, is the main building block that would enable an effective orchestration framework. Indeed, given the distributed nature of the CHARITY platform, using a centralised monitoring server is not a viable option. Therefore, the monitoring framework should be designed in such a way that would ensure the monitored data to be distributed over many locations while maintaining the overall observability for the orchestration framework. The architecture of the monitoring framework was presented along with the tools that would be used to implement it. While for the prediction mechanisms, many were proposed and evaluated for both of computation and networking predictions. These mechanisms are based on novel neural network models that use continuous time-series data to offer precise predictions. For computation prediction, predicting CPU usage was investigated, which also should be extensible to offer memory prediction. While network prediction mechanisms concentrate on the load exerted, which consist on bandwidth usage prediction or users' requests / sessions prediction.

A sizeable amount of effort was directed to study the privacy and security implications of running XR services onto Edge/Cloud resources. Thus, an architecture based on service-mesh and Open Policy Agent (OPA) is proposed to address these challenges. This architecture heavily leverages the notion of closed control loops to offer autonomous security capabilities. Some security mechanisms for cloud-native environments were also been proposed. One mechanism was proposed to secure containers by limiting them to the minimum set of capabilities that would ensure a correct behaviour. Another mechanism concerns the obfuscation of the networking infrastructure. This mechanism is aimed to limit the insight an attacker might get by scanning the network while preserving subnet information for debugging purposes. Other mechanisms were proposed to offer Authentication and Authorization to XR providers, perform static application and container image security testing, etc.

Finally, the last aspect is about the application management framework, where a frontend application has been developed to permit XR service enablers to connect to the CHARITY platform and deploy their services. The AMF is implementing many major components of the CHARITY platform architecture such as XR Service Enabler repository, XR Service Blueprint Template Repository and XR service exposure component.

References

[1] 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.

[2] W. Tärneberg, A. Mehta, E. Wadbro, J. Tordsson, J. Eker, M. Kihl and E. Elmroth, “Dynamic application placement in the Mobile Cloud Network,” *Future Generation Computer Systems,* vol. 70, pp. 163-177, 2017.

[3] T. Bahreini and D. Grosu, “Efficient placement of multi-component applications in edge computing systems,” in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, New York, USA, 2017.

[4] G. Castellano, F. Esposito and F. Risso, “A Distributed Orchestration Algorithm for Edge Computing Resources with Guarantees,” in *IEEE Conference on Computer Communications*, 2019.

[5] L. Baresi, D. F. Mendonça, M. Garriga, S. Guinea and G. Quattrocchi, “A Unified Model for the Mobile-Edge-Cloud Continuum,” *ACM Transactions Internet Technology,* vol. 19, no. 2, pp. 1-29, 2019.

[6] X. Q. Pham, T. D. Nguyen, V. Nguyen and E. N. Huh, “Utility-Centric Service Provisioning in Multi-Access Edge Computing,” *Applied Sciences,* vol. 9, no. 18, 2019.

[7] S. Wang, M. Zafer and K. K. Leung, “Online Placement of Multi-Component Applications in Edge Computing Environments,” *IEEE Access,* vol. 5, pp. 2514-2533, 2017.

[8] H. Li, N. Chen, B. Liang and C. Liu, “RPBG: Intelligent Orchestration Strategy of Heterogeneous Docker Cluster Based on Graph Theory,” in *IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2019.

[9] G. F. Anastasi, E. Carlini, M. Coppola and P. Dazz, “QoS-aware genetic Cloud Brokering,” *Future Generation Computer Systems,* vol. 75, pp. 1-13, 2017.

[10] O. Abdel-Wahab, A. Mourad, H. Otrok and T. Taleb, “Federated Machine Learning: Survey, Multi-Level Classification, Desirable Criteria and Future Directions in Communication and Networking Systems,” *IEEE Communications Surveys & Tutorials,* vol. 23, no. 2, pp. 1342 - 1397, 2021.

[11] D. Kimovski, H. Ijaz, N. Saurabh and R. Prodan, “Adaptive Nature-Inspired Fog Architecture,” in *IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, Washington DC, DC, USA, 2018.

[12] 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.

[13] X. Foukas, G. Patounas, A. Elmokashfi and M. K. Marina, “Network Slicing in 5G: Survey and Challenges,” *IEEE Communication Magazine,* vol. 55, no. 5, p. 94–100, 2017.

[14] S. Global, “2021 RESEARCH PLAN - Cloud Price Index,” 451research S&P Global, 2021.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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," *IEEE IoT Journal,* under review.

[23] 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.

[24] P. Goransson and C. Black, Software Defined Networks: A Comprehensive Approach, Morgan Kaufman, 2014.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] L. Siamak, F. Pakzad and M. Portmann, "SCOR: software-defined constrained optimal routing platform for SDN," *arXiv preprint arXiv:1607.03243,* 2016.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] T. P. Lillicrap and e. al., “Continuous control with deep reinforcement learning”. Patent U.S. Patent 0 024 643 A1, 2 Feb 2017.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] J. Prados-Garzon and T. Taleb, “Asynchronous Time-Sensitive Networking for 5G Backhauling,” *IEEE Network,* vol. 35, no. 2, pp. 144-151, 2021.

[45] 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.

[46] 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].

[47] 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.

[48] 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.

[49] 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].

[50] 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].

[51] 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.

[52] 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].

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] F. Pilka and M. Oravec, “Multi-step ahead prediction using neural networks,” in *Proceedings ELMAR-2011*, Zadar, Croatia, 2011.

[65] 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.

[66] 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.

[67] 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.

[68] D. Sabella and e. al., “Developing Software for Multi-Access Edge Computing,” ETSI White Paper No. 20, Sophia Antipolis, France, 2019.

[69] 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.

[70] R. A. Addad and e. al., “Towards studying Service Function Chain Migration Patterns in 5G Networks and beyond,” in *IEEE Global Communications Conference (GLOBECOM)*, Waikoloa, HI, USA, 2019.

[71] 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.

[72] ETSI, “Zero-touch network and Service Management (ZSM); General Security Aspects,” ETSI, Sophia Antipolis, France, 2021.

[73] 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.

[74] 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.

[75] 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.

[76] S. Rose, O. Borchert, S. Mitchell and S. Connelly, “NIST Special Publication 800-207 - Zero Trust Architecture,” NIST, USA, 2020.

[77] 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.

[78] 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].

[79] R. Chandramouli and Z. Butcher, “NIST Special Publication 800-204A - Building Secure Microservices-based Applications Using Service-Mesh Architecture,” NIST, USA, 2020.

[80] E. Harlicaj, “Anomaly Detection of Web-Based Attacks in Microservices,” Aalto University, Espoo, Finland, 2021.

[81] 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.

[82] 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.

[83] 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.

[84] 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.

[85] 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.

[86] C. Benzaid, T. Taleb and J. Song, “AI-based Autonomic & Scalable Security Management Architecture for Secure Network Slicing in B5G,” *IEEE Network,* pp. 1-9, (to appear).

[87] R. Meier, P. Tsankov, V. Lenders, L. Vanbever and M. Vechev, “NetHide: Secure and Practical Network Topology Obfuscation,” in *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD, USA, 2018.

[88] S. T. Trassare, R. Beverly and D. Alderson, “A Technique for Network Topology Deception,” in *MILCOM 2013 - 2013 IEEE Military Communications Conference*, San Diego, CA, USA, 2013.

[89] Q. Wang, F. Xiao, M. Zhou, Z. Wang, Q. Li and Z. Li, *Linkbait: Active Link Obfuscation to Thwart,* arXiv, 2017.

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]