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

D4.6: Evaluation, Validation and Showcasing Outcomes (final)

Version: v1.0

Deliverable type	R (Document, report)
Dissemination level	PU (Public)
Due date	30/06/2024 (Latest Amendment)
Submission date	
Lead editor	Antonis Protopsaltis (ORAMA)
Authors	Antonis Protopsaltis (ORAMA), George Kokiadis (ORAMA), Antonios Makris (HUA), Theodoros Theodoropoulos (HUA), Konstantinos Tserpes (HUA), Joao Rodrigues (DOTES), Mike McElligott (UTRC), Massimiliano Corsini (CNR), Massimo Coppola (CNR), Peter Gray (CS), Luis Rosa (ONE), Luis Ferreira (ONE), Luis Cordeiro (ONE), Diogo Fevereiro (ONE); Tarik Taleb (ICT-FI), Nora Taleb (ICT-FI), Hao Yu (ICT-FI), Qize Guo (ICT-FI), Yan Chen (ICT-FI), Tarik Zakaria Benmerar (ICT-FI), Giovanni Guliani (HPE), Laura Sande (PLEXUS), Yago González (PLEXUS), Alex Roibu (HOLO3D)
Reviewers	Giovanni Guliani (HPE), Antonios Makris (HUA)
Work package, Task	WP4, T4.3
Keywords	Evaluation, use cases

Abstract

This document reports on the outcomes of the evaluation, validation and showcasing activities. It reports on the set of experiments, technological setups and validations, to provide feedback to technological Work Packages (WPs) and on the impact the evaluation and validation had on the development process. A combination of different subtopics with related metrics is exploited, derived primarily from log data to assess individual functionalities of the platform components/services as well as Use Case related aspects and verify the functional aspects of their intended operation. This document constitutes the final version of the validation and evaluation.

Document revision history

Version	Date	Description of change	List of contributor(s)
v0.1	23/04/24	Initial ToC	ORAMA
v0.2	25/05/24	Initial document content	ORAMA
v0.3	20/05/24	Partial Content editing	all
v0.4	05/06/24	Partial Content editing	all
v0.5	22/06/24	Partial Content editing	all
V0.6	25/06/2024	Final Content editing	ONE, HPE, CNR, ORAMA
V0.7	25/06/2024	Final parse editing	ORAMA
V1.0	27/06/2024	Reviewers comments addressed	ORAMA, HPE, CNR, HOLO3D, CS, PLEXUS, TID, UTRC, ORBK, ONE

Disclaimer

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

The deliverable delves into the comprehensive process of validating and evaluating the CHARITY prototype, a crucial step in ensuring its functionality and efficacy. It underscores the significance of assessing not only the individual components and services but also their integration within the broader context of the project's use cases. This evaluation takes place across diverse testbed environments provided by partners, enabling controlled tests and data collection essential for thorough analysis.

To gauge the success of the project's ambitions and methodologies, a multifaceted approach is employed, leveraging various metrics primarily derived from log data. These metrics serve as quantitative indicators, shedding light on the technical characteristics and operational functionalities of the CHARITY platform. By scrutinizing these metrics, valuable insights into the platform's performance and its alignment with project objectives is gained.

The document delineates the validation and evaluation process into distinct phases, with this report representing the initial stage. It focuses on reporting the outcomes of the set of experiments, technological setups, and validation procedures. Moreover, it provides valuable feedback to the technological work packages, informing future development endeavours.

A critical aspect of the evaluation involves defining and addressing specific subtopics linked to project requirements, both functional and non-functional. Each subtopic is meticulously examined, with appropriate metrics defined to measure its performance. Experimental procedures are outlined, detailing the methodology, tools, and instruments utilized to gather relevant data during functional tests. Validation of the platform is achieved through KPI assessment, and a lessons learnt discussion is also conducted. Additionally, project showcasing is presented through pictures from specific events.

Furthermore, the document emphasizes the necessity of defining testbed characteristics and capacities to facilitate the deployment of CHARITY components. This involves compiling detailed descriptions of the testbed infrastructure, including production cloud resources and open-source cloud stacks. Such information is crucial for ensuring seamless integration and optimal performance across diverse environments.

Overall, the deliverable serves as a comprehensive guide to the validation and evaluation process, providing invaluable insights into the progress and performance of the CHARITY prototype.

Table of Contents

List of Figures

List of Tables

Abbreviations

5G-PPP	5G Infrastructure Public Private Partnership
AAA	Authentication, Authorization, Accounting
AAE	Adversarial Autoencoder
AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
CCU	Concurrent User
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DASH	Dynamic Adaptive Streaming over HTTP
EDSR	Enhanced Deep Super Resolution
ESPCN	Efficient Sub-Pixel Convolutional Neural Network
ESRGAN	Enhanced Super-Resolution Generative Adversarial Network
ETSI	European Telecommunication Standard Institute
FSRCNN	Fast Super-Resolution Convolutional Neural Network
GPU	Graphics Processing Unit
GSM	Game Server Manager
HD	High Definition
HDD	Hard Disk Drive
HLO	High Level Orchestrator
HLS	Http Live Streaming
HMD	Head-Mounted Display
HPC	High Performance Computing
IPFT	Intelligent proactive Fault Tolerance
JSON	Javascript Object Notation
LAPSRN	Laplacian Pyramid Super-Resolution
LHLS	Low Latency Http Live Streaming
LLO	Low Level Orchestrator
LSTM	Long Short-Term Memory
MILP	Mixed-Integer Linear Programming
PLY	Polygon File Format
RIFE	Real-Time Intermediate Flow Estimation
QoE	Quality of Experience
QoS	Quality of Service
RAM	Read Access Memory
REST-API	Representational State Transfer Application Programming Interface
RFT	Reactive Fault Tolerance
RTMP	Real-Time Messaging Protocol

RTSP	Real-Time Streaming Protocol
RTT	Round Trip Time
SRGAN	Super Resolution Generative Adversarial Network
SSD	Solid State Drive
UC	Use Case
UDP	User Datagram Protocol
VDI	Virtual Desktop Infrastructure
VPN	Virtual Private Network
vGPU	Virtual Graphics Processing Unit
VM	Virtual Machine
VR	Virtual Reality
XR	Extended Reality
ZSM	Zero Touch Management

ki>

1 Introduction

1.1 Scope, Motivation and Objectives

The CHARITY approach is based on a two-stage prototyping and evaluation cycle which focuses on researching, designing, implementing and evaluating a cloud-native framework to be able to use specific mechanisms to support the deployment and life-cycle management of a set of Cloud-based XR Services (e.g., distributed holographic, AR and VR applications) and improving the overall applications' performance and user experience.

The deliverable focuses on the validation and evaluation of components and services of the CHARITY prototype as well as the Use Cases in different testbed environments. The testbed environments are provided by the partners and allow the execution of controlled tests and collecting the required measurements for the assessment. The validation and evaluation aim to provide insight to whether the project ambitions and approaches provide tangible outcomes to the use cases of the projects. A combination of metrics is exploited, derived primarily from log data to draw conclusions on the platform's technical characteristics and functionalities.

This document constitutes the final of two versions of the validation and evaluation, reporting on the final set of experiments, technological setups, and validation. It aims to provide feedback to technological WPs and on the impact the evaluation and validation had on the development process.

1.2 Methodology

We mainly devised the content presented in this deliverable based on the following approaches:

- Via regular communication between CHARITY partners using suitable communication tools (e.g., cross WPs meetings, offline communication through e-mail or Slack messages)
- Sharing examples to guide partners on drafting their evaluation subtopics and providing different rounds of feedback to support a common approach and alignment between the different subtopics handled
- Each partner responsible for an evaluation subtopic has conducted the required functional tests and experiments to collect and analyse relevant data. For the evaluation subtopics related to the use cases reference to relevant KPIs was provided.

1.3 Structure of the document

Section 2 reports on the preparatory activities for evaluation with reference to the evaluation subtopic definition. The subtopics were linked to the requirements devised in D1.2 and categorized according to platform components, services and use cases. The section further reports on the testbeds and resources that will be used for experimentation and validation.

Section 3 details the evaluation results of the platform and for specific enablers hosted in the platform.

Section 4 details the use case evaluation results exploiting the CHARITY platform.

Section 5 summarizes the validation and lessons learnt.

Section 6 presents the showcasing of CHARITY platform through use case applications.

2 Preparatory activities for evaluation

2.1 Evaluation subtopics definition

In order to assess individual functionalities of the platform components/services as well as Use Case related aspects, and verify the functional aspects of their intended operation, a number of related subtopics was provided. The subtopics were linked to the requirements devised in D1.2 and the non-functional requirements addressing different attributes of the system. For each subtopic, appropriate metrics were defined and the necessary tools have been utilized by each partner to gather the relevant evaluation data during the functional tests (i.e., log data).

2.1.1 Platform

As reported in D4.2, the integrated components are evaluated in the scope of T4.3. The efforts of this task are reflected in two document versions: D4.4, which describes the evaluation and experimentations of the first phase of the integration of the isolated components, and D4.6, which contains the experimentations and evaluations conducted on the final version of the integrated components. The integration activities conducted to arrive to the integrated version of the platform components is described in detail in D4.5.

The CHARITY Platform is responsible for the deployment, monitoring, adaptation, and release of resources of the XR applications inside the platform. The general "XR Application Journey" of an XR Application stored in the CHARITY platform is described in detail in the D4.5, together with the CHARITY project pilots prepared to test it. The platform components have been deployed in a dedicated cluster specifically created at CloudSigma premises in Zurich, and resources for XR applications have been onboarded from three datacentres: "c01" from CloudSigma in Zurich (Switzerland), "e01" from CloudSigma in Sofia (Bulgaria) and "e02" from OneSource in Coimbra (Portugal), as depicted in the following picture (taken from a screenshot of a demo support dashboard).

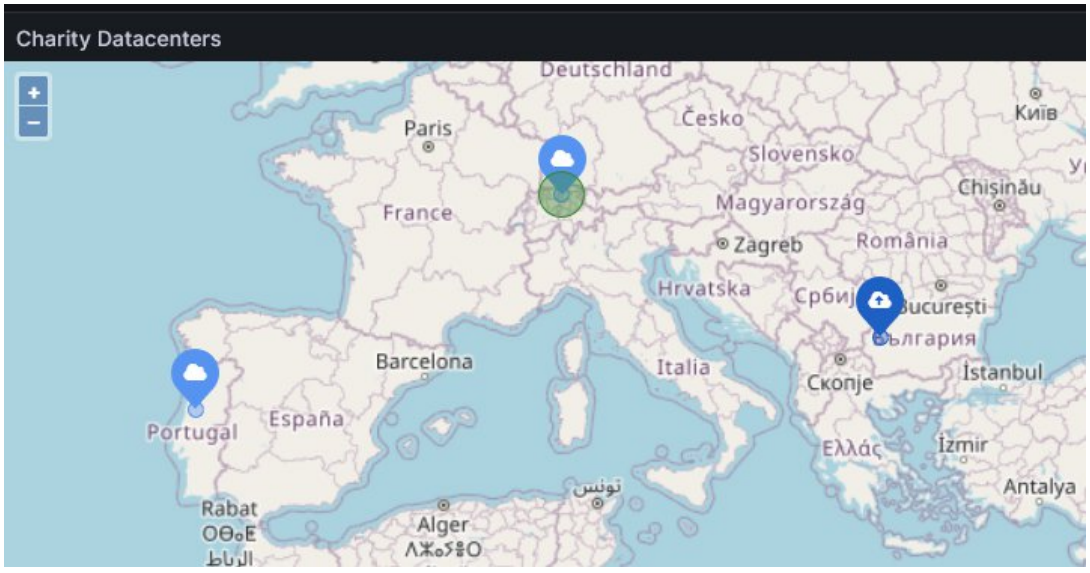


Figure 1 - Location of CHARITY platform datacentres

Besides the AMF web interface, two Grafana enabled dashboards have been developed/customized to help troubleshooting and demonstrating CHARITY Platform functionalities.

The first dashboard has been developed to show, for each datacentre, the total and free resources (e.g., CPUs , GPUs, flavour sizes), the available clusters and their occupancy in terms of CPU and

memory load and, the deployed XR applications. Figure 2 provides the overview of the “c01” datacentre.

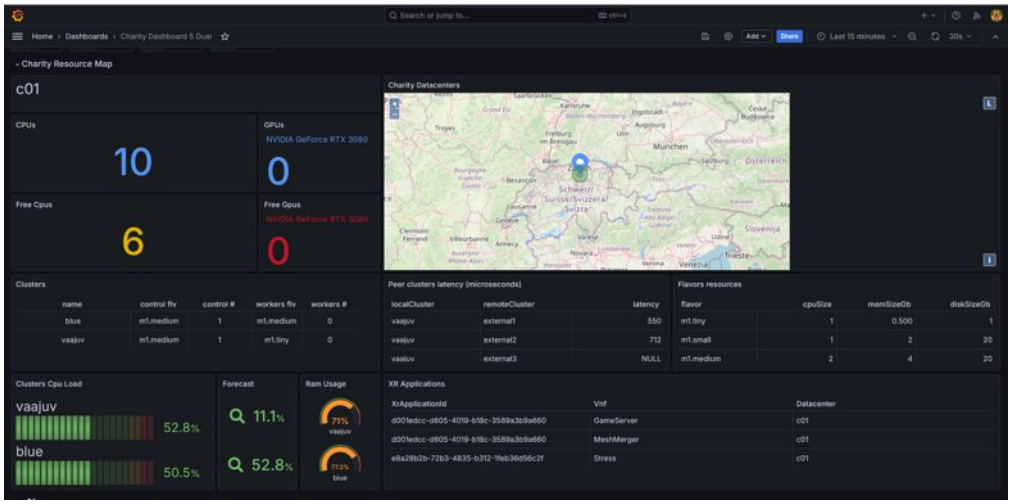


Figure 2 - Dashboard: Overview of "c01" datacentre

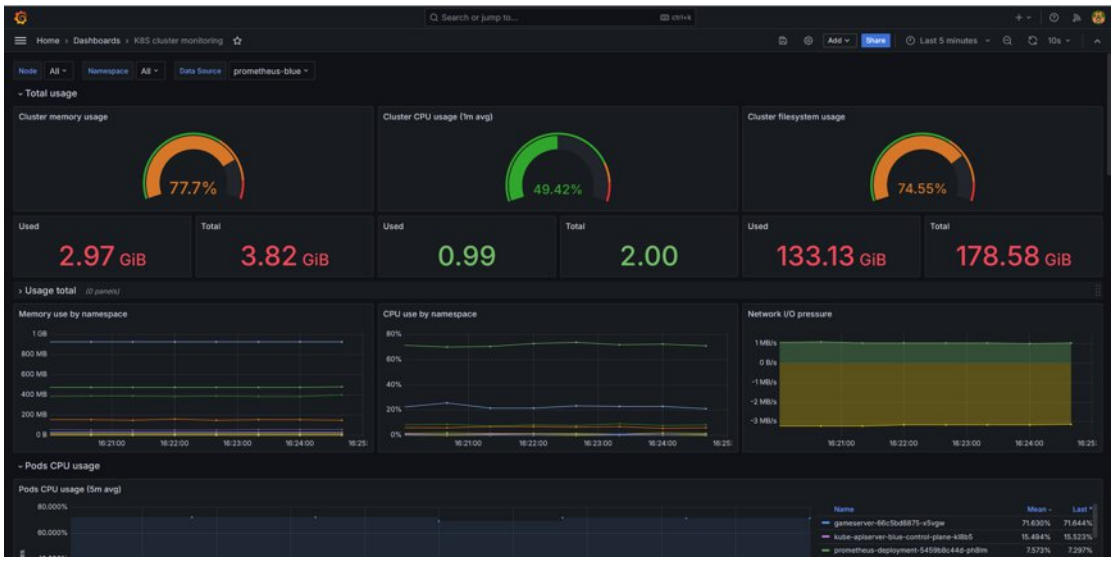


Figure 3 - Dashboard: cluster resource usage

The second dashboard (cf. Figure 3) is an open-source Kubernetes dashboard that has been customized to select CHARITY clusters and directly retrieve metrics used by the Monitoring manager.

Figure 3 presents the displayed information regarding the detailed resource usage by the “blue” cluster inside the “c01” datacentre as follows: the top row gauges show cluster memory, CPU and filesystem usage, while the graphs in the lower part depict the last values of various detailed metrics aggregated in several grouping modes.

2.1.2 XR service Enablers

As reported in D3.2 specific data services were developed as part of the project and are exploited by a subset of XR applications of the CHARITY project. Even if these data services are targeted to the use cases of CHARITY, it is envisioned by the partners of the consortium that such services can be used/adopted by other XR applications with similar needs, beyond the ones involved in the project itself. As part of this deliverable the following XR services have been evaluated:

- The Mesh Merger service which employs geometry processing algorithms to build virtual environment for AR applications, that enables the UC3-1 Collaborative Game.

- The Point cloud encoder/decoder service, that is the main component of the UC1-3 Holo Assistant and supports the efficient transmission of a huge amount of 3D data from the cloud to the edge (the holographic display).

2.1.3 Use Cases

All the application Use Cases and the developed components following the integration and experimentation plan reported in Section 5 of D4.2. The evaluation is reported in this deliverable, except for the Use Case UC1-3 Holographic Assistant, that left the consortium at the end of the Q4-2023.

2.2 Procedures and metrics definition

For each of the listed subtopics, proper experimental procedures and metrics are defined. The main items related to the evaluation subtopics definition were focused on providing information on a) the evaluation scope of each subtopic, b) the related requirements, c) the relevant platform application components, d) the measurement points for collecting and storing the data, e) the instruments and tools exploited, f) the methodology and procedure and g) the metrics to analyse the results. *Table 1* provides a summary of the subtopics that are handled in this deliverable, along with a reference to which category they belong (platform, XR services, Use Cases) and the metric used in the evaluation. Detailed descriptions and results of the individual subtopics are reported in Section 3.

Table 1. Summary of subtopics

Subtopic description	**Category**	**Metric/Evaluation**
CHARITY platform	Platform	Usability and functional metrics
Point Cloud encoding/decoding (PC E/D)	XR services	Number of views, number of 3D points, FPS, KPIs
Mesh Merger	XR services	Data transmission time, processing time, KPIs
Holographic Concert and holographic meetings	Use Case	Several metrics, KPIs
Realistic simulation in VR medical training	Use Case	Several metrics, KPIs
Virtual Experiences Builder for the web	Use Case	Several metrics, KPIs
Mobile multiplayer game utilising AR technology	Use Case	Latency, RTT between Game Client and Game Server, and between Game Server and Mesh Merger, KPIs
Cloud Native Flight Simulator	Use Case	Several metrics, KPIs

2.3 Testbeds and resources

This section describes the general testbed characteristics and capacity of the combined testbed infrastructure consisting of CloudSigma's production cloud and supplementary testbed deployments. We detail each operator's general characteristics, resource capacity, account creation and access criteria. We also describe the ongoing technical support and maintenance required to ensure continuous operation throughout the project.

The main objective is to support the integration of the components developed in WP2 and WP3, according to the technical requirements of the CHARITY architecture and provide the underlying infrastructure to deliver a working proof-of-concept for validation and demonstration

To support the deployment of CHARITY components on the combined testbed infrastructure, we must first define testbed descriptions with the characteristics and capacity available per provider/operator. This information is collected using *Table 2* as a template,. As stated, the CHARITY testbed combines production cloud resources (e.g., CloudSigma) with private clouds based on widely used open-source cloud stacks (e.g., Openstack).

Table 2. Testbed Template

Short description	
General configuration	
Hypervisor	
IaaS stack/version	
VM Monitoring	
Access methods	
Connectivity	
Cloud interface	
Provisioning	
Integration/drivers	
Networking	
Compute capacity (available for project use)	
CPU (Ghz)	
RAM (GB)	
Number of VMs	
Storage capacity (available for project use)	
SSD (GB)	
HDD (GB)	
Image format	
Networking	
Max internal network bandwidth per VM (Gb)	
Max external network bandwidth per VM (Gb)	
Max inter-VM latency (ms)	
Total cloud external network bandwidth (Gb)	

2.3.1 CloudSigma Testbed Characteristics and Capacity

CloudSigma has provided testbed infrastructure comprising a testing environment in Sofia, Bulgaria, and three production cloud environments in Geneva and Zurich, Switzerland and Boden, Sweden. During the project, CloudSigma provided accounts to project partners along with the required

computing resources and provided expertise and support on configuration and infrastructure optimisation. CloudSigma tested and validated a number of high-performance GPUs in their testlab in Sofia using the project use case specifications, including NVIDIA TESLA V100 and A100 Tensor Core GPUs. Eventually, CloudSigma was able to integrate and expose the NVIDIA RTX A6000 into one of their production cloud locations in Boden, Sweden for testing only in Q3 2023. The Sofia testlab retained the NVIDIA A100's. The NVIDIA RTX A100's are optimised for data analytics workloads and applications like VDI, high-performance computing (HPC), and AI/Deep learning. However, it is essential to note that some advanced settings are not yet exposed via the CloudSigma Web Interface.

During the validation phase, CloudSigma explored both passthrough and vGPU (virtualised GPU) for VM allocation, with passthrough typically being preferred for workloads that require dedicated access to the GPU.Passthrough is typically preferred for workloads that require maximum GPU performance, such as high-performance computing or deep learning applications, but require dedicated access to the GPU. At the same time, vGPU is more suitable for scenarios where GPU resources need to be shared among multiple VMs or containers, such as VDI or multi-tenant environments, offering a balance between performance and resource consolidation. While CloudSigma has successfully tested both options, only passthrough is enabled at the time of writing, meaning project partners can only attach one GPU per VM. A ClusterAPI (CAPI) CloudSigma provider was developed that facilitates the management, provisioning, and lifecycle of Kubernetes clusters across different infrastructure environments. The CloudSigma provider implementation for Cluster API provides a declarative API and tooling to simplify the management and lifecycle of Kubernetes clusters.

Table 3. CloudSigma Testbed Characteristics

CloudSigma Cloud Locations: Geneva, Switzerland (GVA) and Boden, Sweden (LLA)	
Short description	Test environment in Sofia, Bulgaria. Production environment in Geneva (GVA), Zurich (ZRH) and Boden (GVA). The platform combines a proprietary stack with open-source technologies to provide a utility approach to IaaS provisioning. The platform offers a high level of control and flexibility in the provision of computational power, RAM, storage, and networking.
General configuration	
Hypervisor	KVM
IaaS stack/version	Proprietary CloudSigma stack
VM Monitoring	Intra-VM testing tools, at the discretion of the VM owner, NewRelic third-party integration
Access methods	API via HTTPS
Connectivity	Internet, VPN, Secure Remote User Access, Direct private patch to local switch
Cloud interface	WebApp, API
Provisioning	API, API middleware, WebApp, Python library (Pycloudsigma).
Integration/drivers	Ansible, CloudInit, Apache Libcloud, JClouds, Fog, Abiquo Hybrid Cloud, pycloudsigma Library, Terraform, Cluster API
Networking	API, WebApp
Compute capacity (available for project use)	
CPU (Ghz)	400Ghz
RAM (GB)	400GB
vGPU (instance spec.)	Multiple:

	• A6000 (specializing in graphics computations) commissioned in Boden testbed on November 2023 and decommissioned since December 2023. • A100 (specializing in Machine learning computations) in Sofia testbed
Number of VMs	Unlimited
Storage capacity (available for project use)	
NVMe SSD (GB)	5000
HDD (GB)	N/A
Image format	RAW
Networking	
Max internal network bandwidth per VM (Gb)	20
Max external network bandwidth per VM (Gb)	10
Max inter-VM latency (ms)	1

2.3.2 TID Testbed Characteristics and Capacity

Leveraging TID expertise, a second testbed was deployed at TID premises (Valladolid) (c.f. *Table 4*) for internal experimentation. This testbed is also an important milestone that helps test the developed technologies with huge computational resource requirements in a customized infrastructure with a three GPU support, and 3 small clusters without GPUs.

TID also installed a less powerful workstation in TID offices (Barcelona) to test on-site components with extremely low latency requirements. TID has installed NVIDIA RTX 3090 graphics cards in both locations. One VM is supported. NVIDIA RTX 3090 GPU is ideal for data analytics workloads and applications like VDI, HPC, and AI/Deep learning. Being a testing platform enables easy testing of various GPU settings to evaluate the Adaptive Scheduling workload algorithm.

Table 4. TID Testbed Characteristics

TID Cluster Location (Valladolid)	
Short description	Test IaaS workstation in Valladolid (Zrh). The workstation uses open-source technologies to provide a utility approach to IaaS provisioning. The workstation offers a large computational resource with GPU support.
General configuration	
Hypervisor	-
IaaS stack/version	OpenNebula
VM Monitoring	-
Access methods	ssh
Connectivity	Internet, VPN, Secure Remote User Access
Cloud interface	-
Provisioning	API, Python library
Integration/drivers	Flexible
Networking	N/A

Compute capacity (available for project use)	
CPU (Ghz)	100Ghz
RAM (GB)	126GB
vGPU (instance spec.)	3x Titan RTX 3900
Number of VMs	Currently up to 1 VM
Storage capacity (available for project use)	
SSD (GB)	500
HDD (GB)	9Tb
Image format	RAW
Networking	
Max internal network bandwidth per VM (Gb)	N/A
Max external network bandwidth per VM (Gb)	N/A
Max inter-VM latency (ms)	N/A

Table 5. *TID Testbed Characteristics without GPU support*

TID 2 Cluster Location (Peñuelas)	
Short description	Test IaaS cluster uses open-source technologies to provide a utility approach to IaaS provisioning. The workstation offers a medium computation without GPU support
General configuration	
Hypervisor	-
IaaS stack/version	OpenNebula
VM Monitoring	-
Access methods	ssh
Connectivity	Internet, VPN, Secure Remote User Access
Cloud interface	-
Provisioning	API, Python library
Integration/drivers	Flexible
Networking	N/A
Compute capacity (available for project use)	
CPU (Ghz)	100Ghz
RAM (GB)	32GB
vGPU (instance spec.)	N/A
Number of VMs	Currently up to 1 VM
Storage capacity (available for project use)	
SSD (GB)	50
HDD (GB)	1Tb

Image format	RAW
Networking	
Max internal network bandwidth per VM (Gb)	N/A
Max external network bandwidth per VM (Gb)	N/A
Max inter-VM latency (ms)	N/A

2.3.3 OneSource Testbed Characteristics and Capacity

Leveraging the expertise of OneSource, a third testbed has already been identified at OneSource premises to support further the integration and experimentation of the CHARITY framework. This testbed features a robust 3-node Kubernetes cluster (refer to *Table 6*), vital in facilitating the testing and evaluation of distributed scenarios encompassing hybrid edge-cloud domains. Additionally, it enables the exploration of multi-cluster deployments utilizing cutting-edge overlay networking technologies, such as Liqo. Besides the 3-node cluster, an OpenStack provider is also hosted within the testbed, which is leveraged by Cluster API, integrated in the low-level orchestrator component of the CHARITY framework. Also, considering the newly added support for deployment and management of on-premises Kubernetes clusters within the low-level orchestrator, OneSource's datacentre became usable by the Low-level orchestrator (LLO) as an additional datacentre, where Cloud-Native applications can be deployed.

Furthermore, it offers an opportunity to assess further the performance and effectiveness of the Low-level orchestrator within the CHARITY framework, while also being an asset for integrating and showcasing the framework with the project UCs, each exploiting respective features.

Table 6. OneSource Testbed Characteristics

OneSource Coimbra, Portugal	
Short description	Testbed is located in OneSource's datacenter. The testbed comprises a 3-node Kubernetes cluster for CHARITY project experimentation and validation, along with an OpenStack provider integrated with Cluster API and the datacentre itself as an additional provider leveraged by the LLO.
General configuration	
Hypervisor	VMWare ESXi
IaaS stack/version	N/A
VM Monitoring	N/A
Access methods	Kubectl
Connectivity	VPN
Cloud interface	-
Provisioning	N/A
Integration/drivers	N/A
Networking	N/A
Compute capacity (available for project use)	
CPU (Ghz)	100Ghz

RAM (GB)	32
vGPU (instance spec.)	0
Number of VMs	3
Storage capacity (available for project use)	
SSD (GB)	150
HDD (GB)	N/A
Image format	-
Networking	
Max internal network bandwidth per VM (Gb)	-
Max external network bandwidth per VM (Gb)	-
Max inter-VM latency (ms)	-

2.3.4 ORAMA Testbed Characteristics and Capacity

Utilizing ORAMA's expertise and infrastructure, a fourth testbed has been established at the ORAMA lab to enhance the integration and experimentation of the CHARITY framework. This lab is equipped with Windows machines, each featuring a dedicated public IP, a powerful CPU and memory, and an Nvidia GPU for real-time interactive VR rendering. These machines can host network applications and software that complement the CHARITY infrastructure at other testbeds. They are essential for testing and evaluating distributed scenarios, including hybrid edge-cloud setups, thereby allowing further assessment of the CHARITY framework's performance and effectiveness.

Table 7. ORAMA lab characteristics

ORamaVR, Heraklion, Crete, Greece	
Short description	Testbed is located in ORAMA's lab. The testbed comprises a set of supplementary machines with public IPs.
General configuration	
Hypervisor	N/A
IaaS stack/version	N/A
VM Monitoring	N/A
Access methods	Internet, AnyDesk
Connectivity	Internet
Cloud interface	N/A
Provisioning	N/A
Integration/drivers	N/A
Networking	Ethernet
Compute capacity (available for project use)	
CPU (Ghz)	148Ghz
RAM (GB)	144GB

vGPU (instance spec.)	Multiple (1x RTX 3060, 2x GTX 1070, 1x RTX 2070S)
Number of VMs	4
Storage capacity (available for project use)	
SSD (GB)	200GB
HDD (GB)	1 TB
Image format	N/A
Networking	
Max internal network bandwidth per VM (Gb)	0.5
Max external network bandwidth per VM (Gb)	0.5
Max inter-VM latency (ms)	1

3 Evaluation and results

The following sections present the evaluations and the results of the test activities performed on the CHARITY platform as a whole, and separately for each CHARITY enabler.

3.1 E2E CHARITY Orchestration Workflow

This section describes the scenarios combining the integrated CHARITY framework components devised for evaluating the E2E CHARITY Orchestration workflow as a whole. These scenarios intend to recreate the main reference usage scenarios of the CHARITY platform for demonstrating the CHARITY deployment options and the life cycle management of Cloud-Native applications. These range from the user interaction with the CHARITY dashboard to the allocation and deployment of resources (i.e., applications, clusters) in the multi-domain infrastructure. They also include the monitoring and forecasting capabilities of the CHARITY platform. Moreover, this section focuses on component interactions rather than their detailed description, which can be found in deliverables D2.2, D3.2 and D4.5. For the sake of readability, Figure 4 depicts the high-level diagram of the CHARITY components. A brief description of components is also provided.

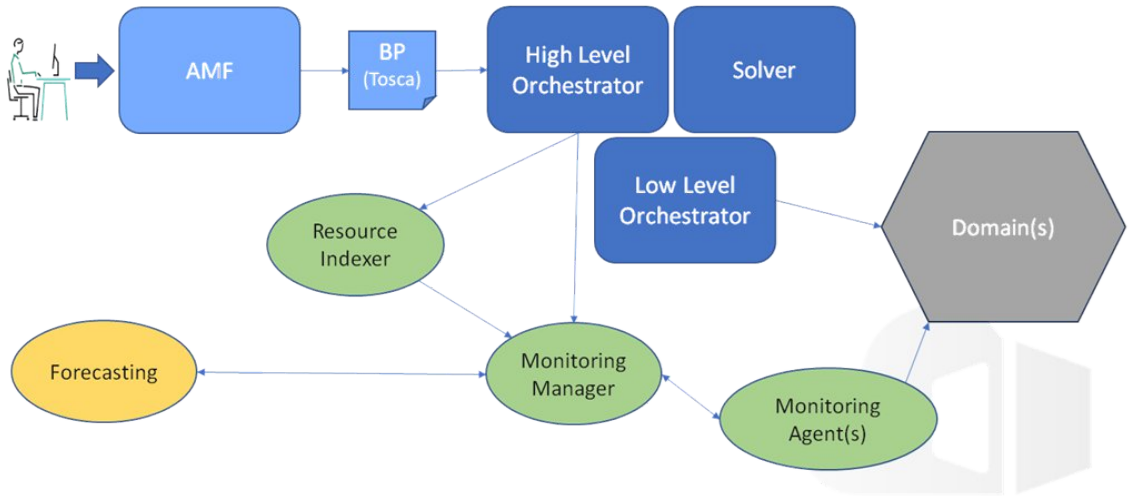


Figure 4 - High level view of CHARITY platform components

Application Management Framework (AMF): The external user interface for XR developers and application managers is provided by the CHARITY Application Management Framework. To use any feature offered by CHARITY, end users need to authenticate themselves through the AMF login process; then they can upload to the internal registry their XR application container images and have them automatically scanned for security vulnerabilities. XR developers are guided through an easy-to-use Web GUI to create XR Blueprints for their applications. Finally, these Blueprints can be deployed to the platform and XR application managers can track their execution state and operate un-deployment and re-deployment actions.

AMF also provides a REST API for all management operations, so that they can be automated by external controller applications.

High Level Orchestrator (HLO): The High-Level Orchestrator module (HLO) performs the high-level phase of resource allocation in the platform, choosing the most suitable deployment topology for a specific XR-application initial deployment request, as well as deciding dynamic re-orchestration for XR applications requesting specific constraints in terms of QoS. HLO abstracts resources in groups at the domain level (e.g., Data centres and Edge clusters).

The HLO architecture interacts with the AMF receiving the user input and with the LLO to perform concrete deployment over computing, storage and computing resources. The HLO is the core of the

MAPE loops performing dynamic orchestration of application and services in the CHARITY platform, thus it receives input data from the MON, the RES and FOR modules. The collected information allows on the one hand resource allocation, and on the other hand to detect the current and forecasted status of in-use and available resources.

The HLO leverages a plug-in interface to potentially select, for each platform instance, the most suitable subcomponent version to perform the solution search and optimization for initial deployment as well as for dynamic re-deployments. The Complex Scenario Optimal Solver Plug-in (CSOC) is based on a Mixed Integer Linear programming model of the deployment problem instance and employing state of the art optimization technology to solve said MILP instance to perform multi-objective and hierarchical optimization of application deployment and dynamic management. The simple Solver Plug-in (SIS) version employs a simpler, linear heuristic algorithm to quickly achieve results for simple topologies, thus speeding up the integration and evaluation activities, demonstrating the end-to-end deployment workflow and leaving the way open to a future integration with more sophisticated, full-powered optimal solvers, like the CSOC.

Low-level Orchestrator (LLO): LLO is responsible for processing the data provided by the user through the AMF and abstracting that input to Kubernetes Custom Resources (e.g., clusters, deployments, services, ingresses) in the available domains, following additional information from decisions provided by the High-Level orchestrator (HLO). Furthermore, the LLO orchestrator is also responsible for orchestrating and managing the deployed Cloud-Native applications and the infrastructure where the applications are hosted, as it is the only component which interacts directly with the infrastructure.

Monitoring Manager (MON): The Monitoring Manager manages, collects and serves monitoring data to other components of the platform. This data is used to take deployment decisions, feed forecasting models, notify predictions of performance failure and provide the developer a permanent picture of the status of the components deployed in the CHARITY Platform. The Prometheus server monitors the clusters and all the components deployed in it, while the Monitoring Manager analyses and transforms the raw data to avoid the complexity of PromQL, the Prometheus query language, and the mathematic calculations behind the metrics used. The collected data is available to the rest of the platform components through a REST API.

Resource Indexing (RES): The Resource Indexing gathers data from all the clusters to provide the HLO, the most accurate performance representation of the available resources distributed across the different datacenters. This component communicates with the Prometheus servers deployed in each cluster to check CPU, memory and storage metrics; these data are provided by cAdvisor, a tool that analyses running containers performance. The performance picture completes with the analysis of the different links dynamically established between clusters according to the architectures of the use case scenarios. The Resource Indexer leverages additional cluster network metrics such as latency and bandwidth provided by Liqo, a tool which is integrated within the LLO, complementing the monitoring of the platform.

Forecasting Manager (FOR): The Forecasting Manager is in charge of providing accurate multi-step predictions regarding various specified metrics. This component communicates with the Monitoring Manager. For each specified metric, the Forecasting Manager entails a designated forecasting model that is based on a novel Deep Learning architecture that has been developed within the frame of the CHARITY project and that has been extensively presented in prior deliverables and corresponding publications. The XR applications whose metrics shall be subjected to the forecasting process are included in a dedicated Forecasting List. Based on the requests performed by the Monitoring Manager, the Forecasting Manager is capable of adding / removing elements to / from the Forecasting List and constructing multi-step predictions regarding a single or multiple elements of the Forecasting List.

3.1.1 Description, procedure, metrics

Table 8. Description of evaluation subtopic -Platform services for XR applications

Subtopic Title: CHARITY *Platform services for XR applications*	**Partners:** HPE - CNR
Short description and evaluation scope: The evaluation of the platform's features supporting the development of XR application blueprints and the deployment of XR applications with adaptive behaviors to guarantee QoS has been conducted using various test applications under different topology load conditions across three scenarios 1. Deployment of a simple XR application Blueprint 2. Deployment of multi-cluster distributed applications 3. Adaptation of XR application deployment for best QoS (OODA loop)	
Components involved: All CHARITY platform component are involved in these 3 scenarios	
Where are data collected and stored – measurement points: The functional aspects are evaluated using the AMF Web GUI support as well as external custom dashboards (see Grafana screenshots in 2.1.1). Moreover, direct access to the clusters of the datacentres is provided through Kubernetes config files generated by the platform and displayed to XR developers by AMF interface, so that traditional Kubernetes troubleshooting techniques can be used to check CHARITY platform component actions directly at the lower levels. Typically, this kind of inspection does not require any storage of the data. The Monitoring component of the CHARITY platform collects infrastructure metrics from the clusters and XR applications: the data is collected by Prometheus and Thanos and stored inside the management cluster of CHARITY platform together with the forecasted data (based on the monitoring ones) that get stored inside a local private database.	
When are data collected? The platform automatically collects only infrastructure metrics of the clusters and XR applications. Custom metrics internal to XR applications are collected only of the XR application explicitly exports them to CHARITY platform using the custom metrics REST API.	
Instruments/tools: Prometheus and Thanos are the main Open-Source software used by the Monitoring system, together with a local private database for the forecasted metric values Grafana is used to create additional custom dashboard to support troubleshooting and demos AMF Web GUI provides a rich set of information to track XR applications deployments K8s command line tools (kubectl) to inspect cluster resources and XR applications pod logs	
Methodology/Procedure: A series of tests will be performed on the platform in a default state, which request a fixed set of applications, with pseudo-randomized key request parameters and application execution order, to allow for significant and repeatable experiments. The execution order will cause different load condition on the platform and trigger adaptation behaviours.	
Metrics to analyse the results The different scenarios will be analysed to validate 1. Correctness of placement decisions 2. Network connectivity between XR application Blueprint internal and external VNFs 3. Adaptation actions to guarantee QoS	

3.1.2 Experimentation Scenarios

This section documents the experimentation scenarios where the metrics stated in the previous section will be collected and evaluated in the next section, while also showcasing the features provided by the CHARITY Framework. The scenarios described in this section highlight the more technical features of the Framework, using generic Cloud-Native applications, showcasing the gentle learning curve for software developers when using the CHARITY platform. The user-related features will be highlighted further in this document, in the sections dedicated to each project use-case.

3.1.2.1 Scenario 1 – Deployment of a Simple XR Application

This section describes the scenario of deploying a simple XR application using the CHARITY platform. The goals of this scenario are the following:

- E2E Integration of CHARITY framework components
- Deployment of a Single application on a Single Domain (cluster)
- Validate the CHARITY orchestration features, such as placement decisions, allocation of resources, monitoring and access to the application.

Figure 5 depicts the scenario and the components involved, including the AMF, HLO, LLO, MON/RES and FOR. The tests were conducted on multiple datacentre testbeds (cf. Section 2.1.1) Mainly two Kubernetes clusters were used as part of the tests: namely a management cluster, for hosting the CHARITY platform components, and a workload cluster, for the deployment of XR applications. For the latter we leveraged the LLO capabilities and integration with ClusterAPI/Openstack to replicate a cloud computing infrastructure. The tests consisted in replicating the steps of a XR application deployment from its specification until the service is running and observe the behaviour. The validation occurred under three conditions: the observation of all components logs, the monitoring of application through the CHARITY dashboards, and the application reachability from the outside.

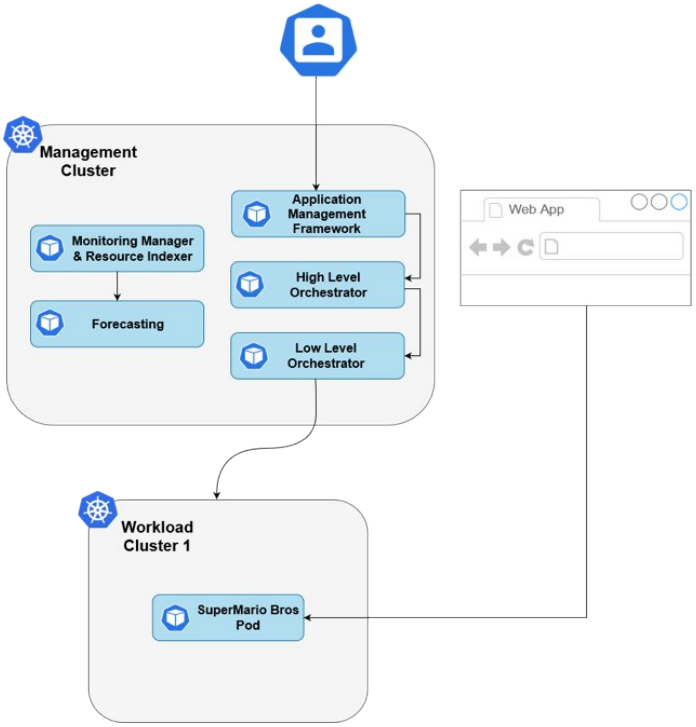


Figure 5 - Single-Cluster Cloud-Native Application Deployment

Next, we describe the test execution details (and the application deployments). Through the AMF, a blueprint was created by defining the requirements of a reference application, i.e. “Super Mario Bros” XR application. This application was chosen as a lightweight web-based single microservice application which can run on a single Kubernetes cluster without additional specific requirements. The application details were introduced through the AMF web graphical interface. At deployment decisions time, the details are converted by AMF to TOSCA and sent to the HLO. The HLO used this information - facilitated by the Solver plugin - to decide where the application should be deployed. For this purpose, HLO leverages the data provided by the MON/RES to determine the optimal deployment decision (case 1). If the Solver can’t find suitable resources in the clusters of the various domains, HLO can request LLO the creation of a new cluster and use it to perform the deployment (case 2). Both cases were tested (cf. Figure 6). The placement decision and application details get merged inside TOSCA model and are then forwarded to the LLO. The latter translated TOSCA application topology specifications and HLO placement decisions to Kubernetes environments. LLO's role included the optional step of

bootstrapping a Kubernetes cluster, installing ancillary packages and dependencies, creating Kubernetes Custom Resources (e.g., deployments, services, ingresses) and their enforcement into the actual cluster. Lastly, the LLO returns to the AMF the corresponding application accesses (i.e., URLs and Kubeconfig), enabling the user to access the deployed application's components. The application deployment's success was observed through the Monitoring Dashboards, including the visualisation of the application components, infrastructure, and CHARITY statistics.

Figure 6 - Kubernetes Cluster Bootstrapping

The conducted experiment validated the aforementioned goals, demonstrating how CHARITY platform versatility can be used to orchestrate and manage containerized Cloud-Native applications. The further sections describe additional platform features and the performed UC validation. Its relevant to highlight, this scenario and the E2E CHARITY orchestration capabilities were showcased in project showcasing activities, including the last EUCnC & 6G Summit 2024. Such capabilities were also published in various scientific venues during the course of the project, namely *"Cross-Cluster Networking to Support Extended Reality Services"*, submitted to the IEEE Network Magazine, *"Intelligent Multi-Domain Edge Orchestration for Highly Distributed Immersive Services: An Immersive Virtual Touring Use Case"* submitted to 2023 IEEE International Conference on Edge Computing and Communications (EDGE) and *"Towards Establishing Intelligent Multi-Domain Edge Orchestration for Highly Distributed Immersive Services: A Virtual Touring Use Case"* (extended version) submitted to Cluster Computing - The Journal of Networks, Software Tools and Applications, in which similar scenarios were devised and tested, further validate the goals, component integration and features described.

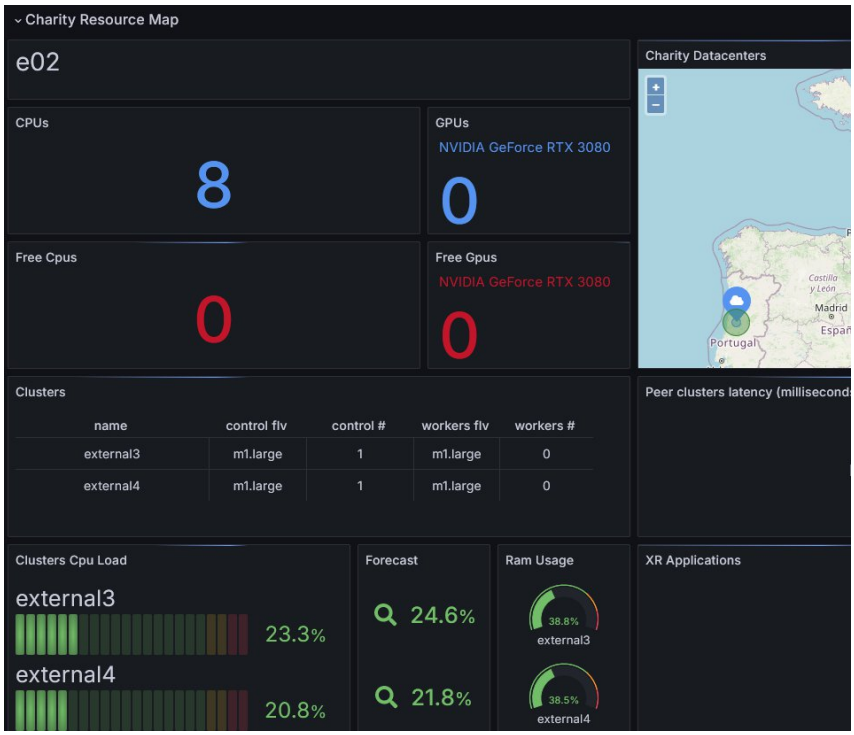


Figure 7 - Resource status at 'e02' datacentre

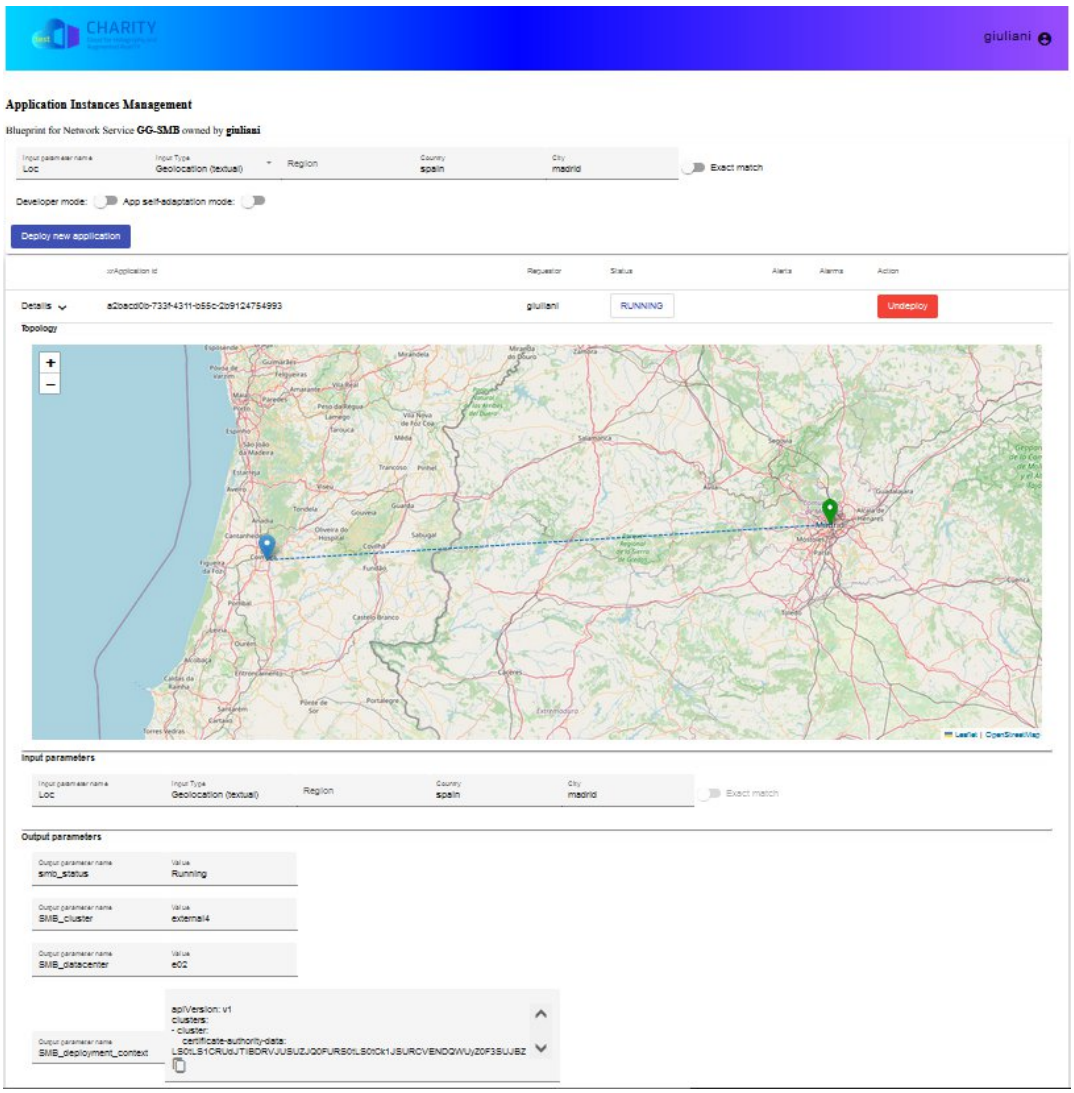


Figure 8 - AMF editor view of deployed XR application

The following screenshots illustrate deployment decisions for the Simple XR application Blueprint being deployed for a gamer located in Madrid (Spain): the selected datacentre will be 'e02' in Coimbra (Portugal), in one of the available clusters (external3 or external4). Figure 7 shows the Grafana dashboard view of the resources in that domain.

The AMF dashboard, once the XR application has been deployed, returns the status information about the target datacentre and cluster, together with K8s configuration file to be used to troubleshoot the pod containing XR application. Figure 8 is a screenshot of this information from AMF editor.: the green marker is for the location of the client, the blue one is where the XR application has been deployed (the closest datacenter, i.e. 'e02' in Coimbra).

At the end this scenario demonstrated the capabilities of the CHARITY platform for deploying an XR application by selecting the optimal placement in the managed domains, and the setup of monitoring/forecasting infrastructure, without requiring any domain specific skills to the XR developers.

3.1.2.2 **Scenario 2 – Multi-Cluster Distributed Application**

This section describes the scenario of a Cloud-Native application distributed across two different clusters.

The goals of this scenario are to validate the following:

- E2E Integration CHARITY component integration and deployment of applications (similar to the previous)
- Bootstrapping of a multi-domain environment (composed of multiple Kubernetes clusters dynamically peered)

These experiments were conducted in the same testbed of the previous scenario. The same management cluster was used, while two work clusters dynamically created upon request (cf. Figure 9). Next, we describe the test execution details (and the application deployments). Similar to Scenario 1, the application blueprint was created in AMF. In this case, the UC2.3 VR Tour Creator (cf. Section 4.3) was used. This UC is composed of 5 components: Cyango-Backend, Cyango-database, Cyango-worker, Cyango-cloud-editor and Cyango-story-express which were all specified in AMF. The specification introduced through the AMF web graphical interface were similarly converted to a TOSCA specification and sent to the HLO. Again, the HLO used this information to decide where the application should be deployed. Nevertheless, in this case we recreated a distributed application deployment where different application components were split across the two workload clusters to guarantee the right level of computing resources availability and required connectivity across them. Hence, upon the request of HLO, the LLO leveraged Liqo's peering and offloading features (cf. Figure 11 - Multi-Cluster Application Deployment) to create a dynamic VPN tunnel (cf. Figure 10). This ensures that regardless of the location of application components (across the two clusters), they communicate with each other. Last, as stated in Scenario 1, the LLO returns to the AMF the corresponding application accesses (i.e., URLs and Kubeconfig), granting the user access to the deployed application's components. The application deployment's success was observed through the Monitoring Dashboards, including the visualisation of the application components, infrastructure, and CHARITY statistics, similar to the the previous scenario.

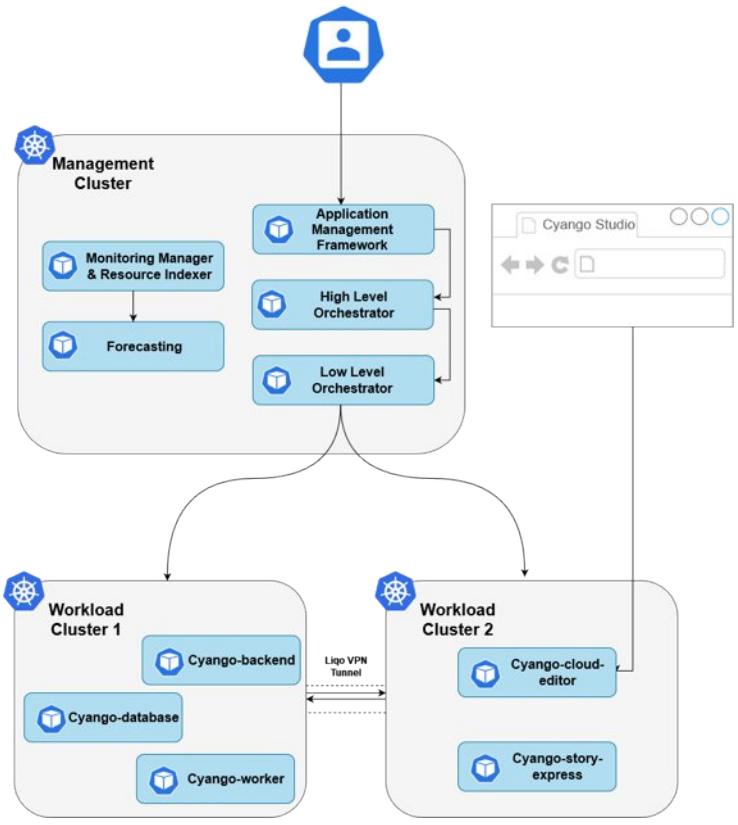


Figure 9 - Multi-Cluster Cloud-Native Application Deployment

The conducted experiment validated the aforementioned goals, demonstrating the CHARITY platform's capabilities in orchestrating and managing multi-cluster environments dynamically. The

further sections describe additional platform features and the performed UC validation. It is also important to highlight that this scenario, features and goals described in this section were also showcased in the events and papers (cf. Figure 12), mentioned in Scenario 1, previously explained in this document.

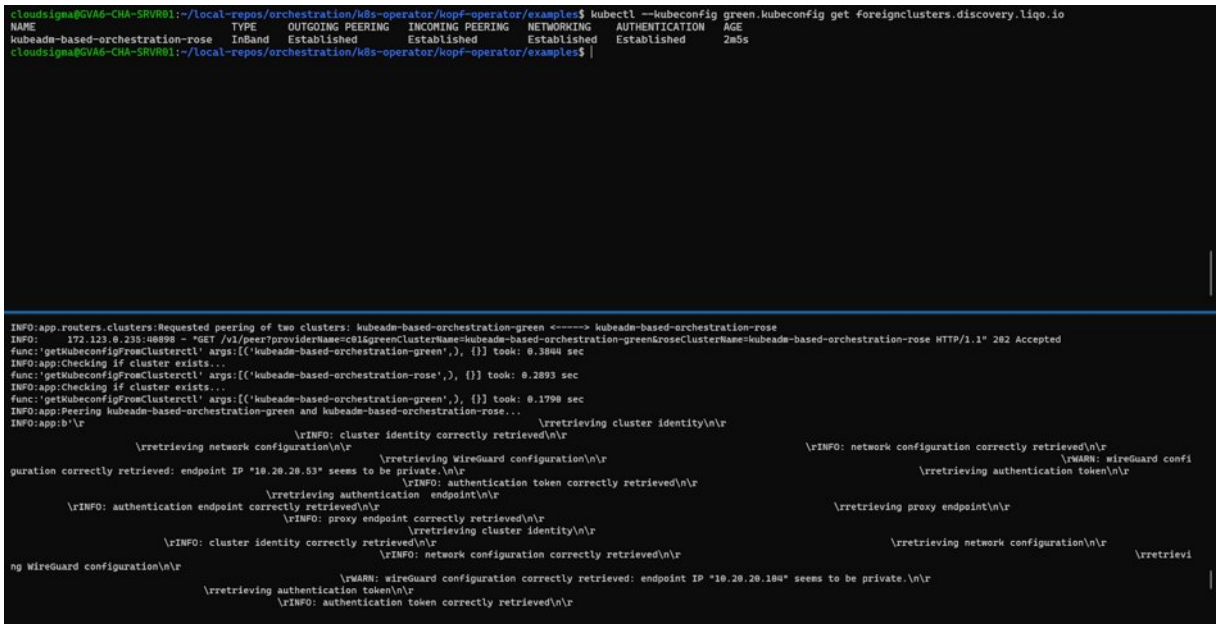


Figure 10 - Cluster Successful Peering

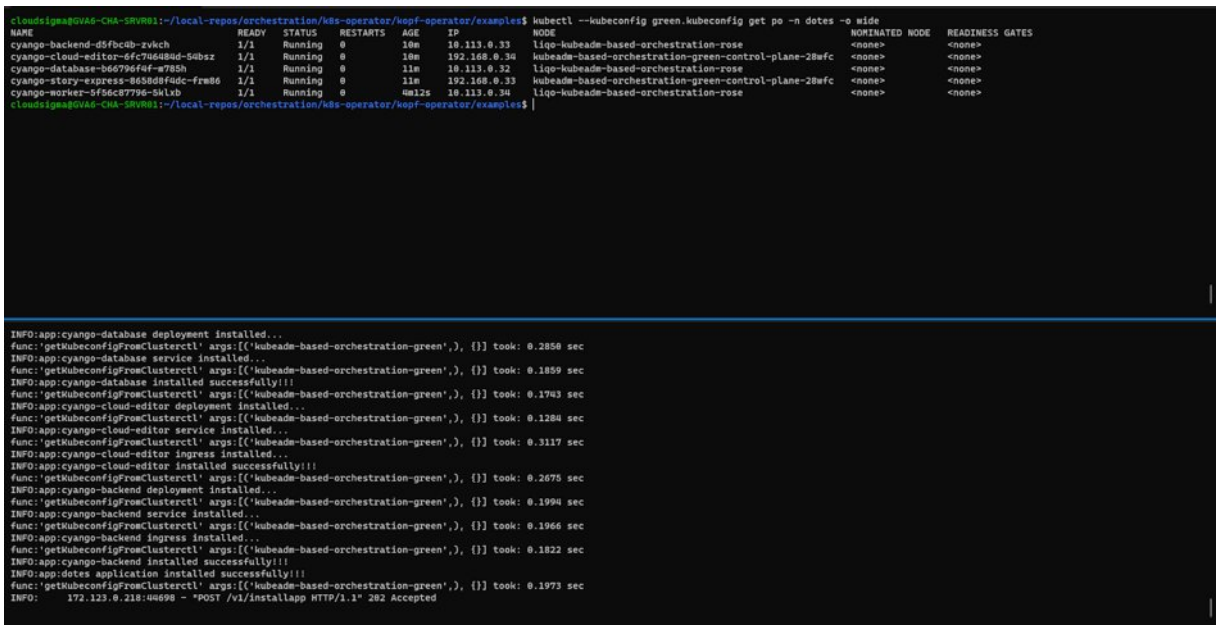


Figure 11 - Multi-Cluster Application Deployment

Figure 12 - EUCnC & 6G Summit 2024 CHARITY Platform Showcase

This scenario demonstrates how the CHARITY platform helps developers to deploy their application in a complex multi-domain environment, allowing them to ignore the final topology details, through CHARITY's transparent network setup.

3.1.2.3 **Scenario 3 – Adaptation of XR application deployment for best QoS (OODA loop)**

This last scenario capitalizes on the previous ones: the XR application has already been deployed by CHARITY platform, and monitoring/forecasting has been setup taking into account the infrastructure metrics of the XR application affecting the QoS (let's assume as an example the CPU load).

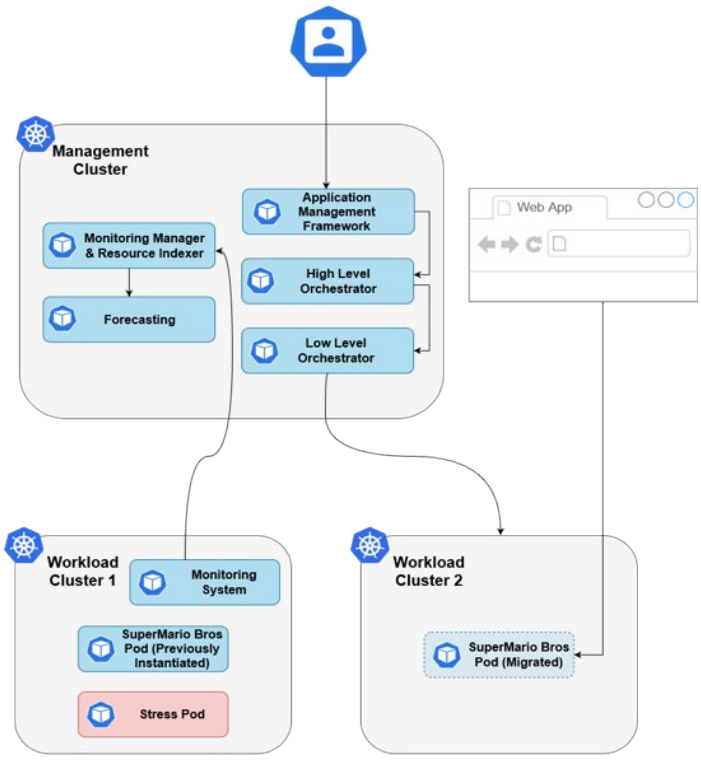


Figure 13 - Cloud-Native Application Live Migration

Inside the cluster where the XR application has been placed by CHARITY, we also deploy another application running the typical Linux stress tests for CPU. When the CPU load of the cluster grows

above attention limits, FOR and MON notify the Alert and Alarm conditions to HLO, that analyse the impact on affected XR applications, and might decide to re-deploy the CPU-QoS sensitive application to another less loaded cluster, so that it will be able to provide the requested QoE. Figure 14 shows the high CPU load of both clusters of ‘e02’ datacentre.

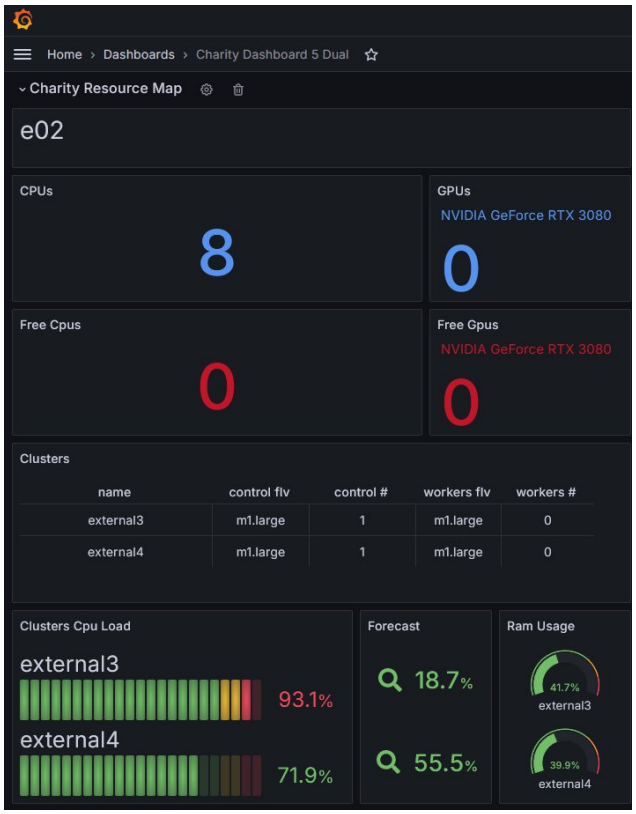


Figure 14 - High CPU load on 'e02' datacentre clusters

At this point MON and FOR notify HLO that sets alerts and alarms for the impacted XR applications, as displayed in Figure 15.

HLO decides to re-deploy the XR application and invokes Solver to find the new optimal placement. Once the redeployment has been completed by LLO, AMF editor will display the new deployment decisions, as displayed in Figure 16. Specifically, the “closest” datacentre with enough free resources is ‘c01’ located in Zurich inside cluster “vaajuv” (‘e02’ in Coimbra is closer, but its CPUs are overloaded). The strange name of this cluster derives from the fact that this has been created automatically by CHARITY platform during a previous deployment that did not find enough resources in the existing clusters but allocated a new one using datacentre free CPUs.

The simple XR application used in the scenario does not use cloud-native storage support (like CHARITY CHES for instance), therefore once migrated the client sessions restart from scratch. This would not have happened if the sessions state was stored in CHES.

This scenario shows an example of the OODA loops realized by CHARITY platform: MON and RES “Observe” the status of the infrastructure and notify HLO in case of significative events. HLO analyses the impact of the event on the running XR application that specified QoS requirements, thus it “Orients” the platform re-deployments. The HLO Solver “Decides” how to perform the re-placements of the XR applications, and finally the LLO “Acts” by actually re-deploying (migrating) the XR applications in different domains.

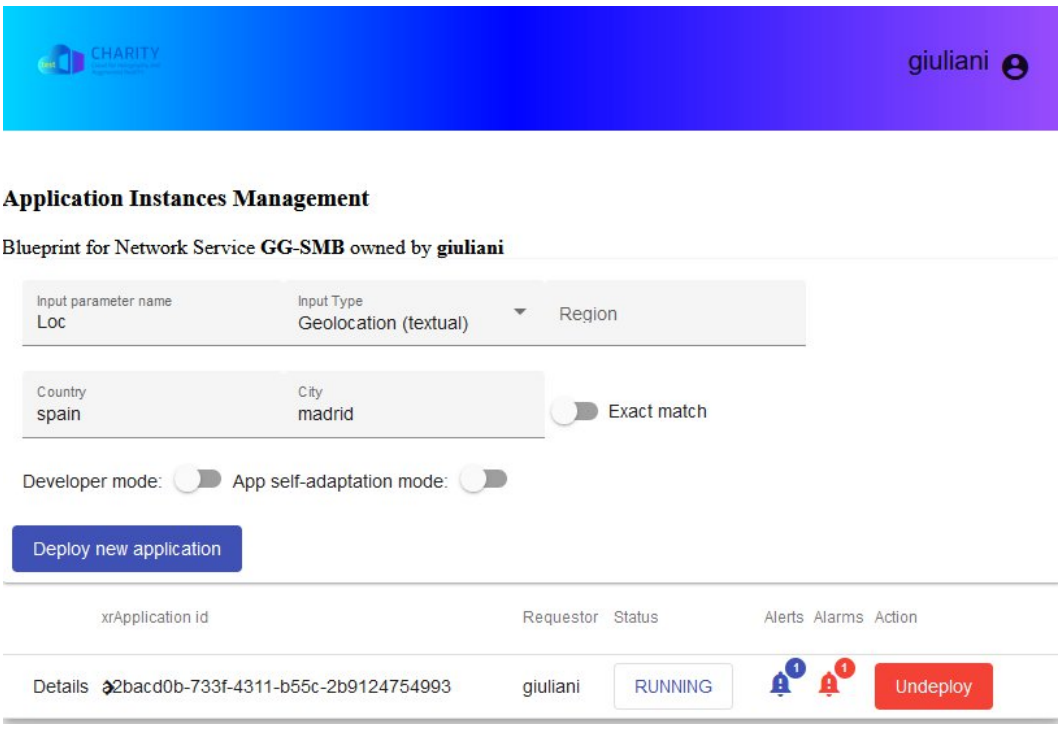


Figure 15 - AMF editor displaying alerts and alarms

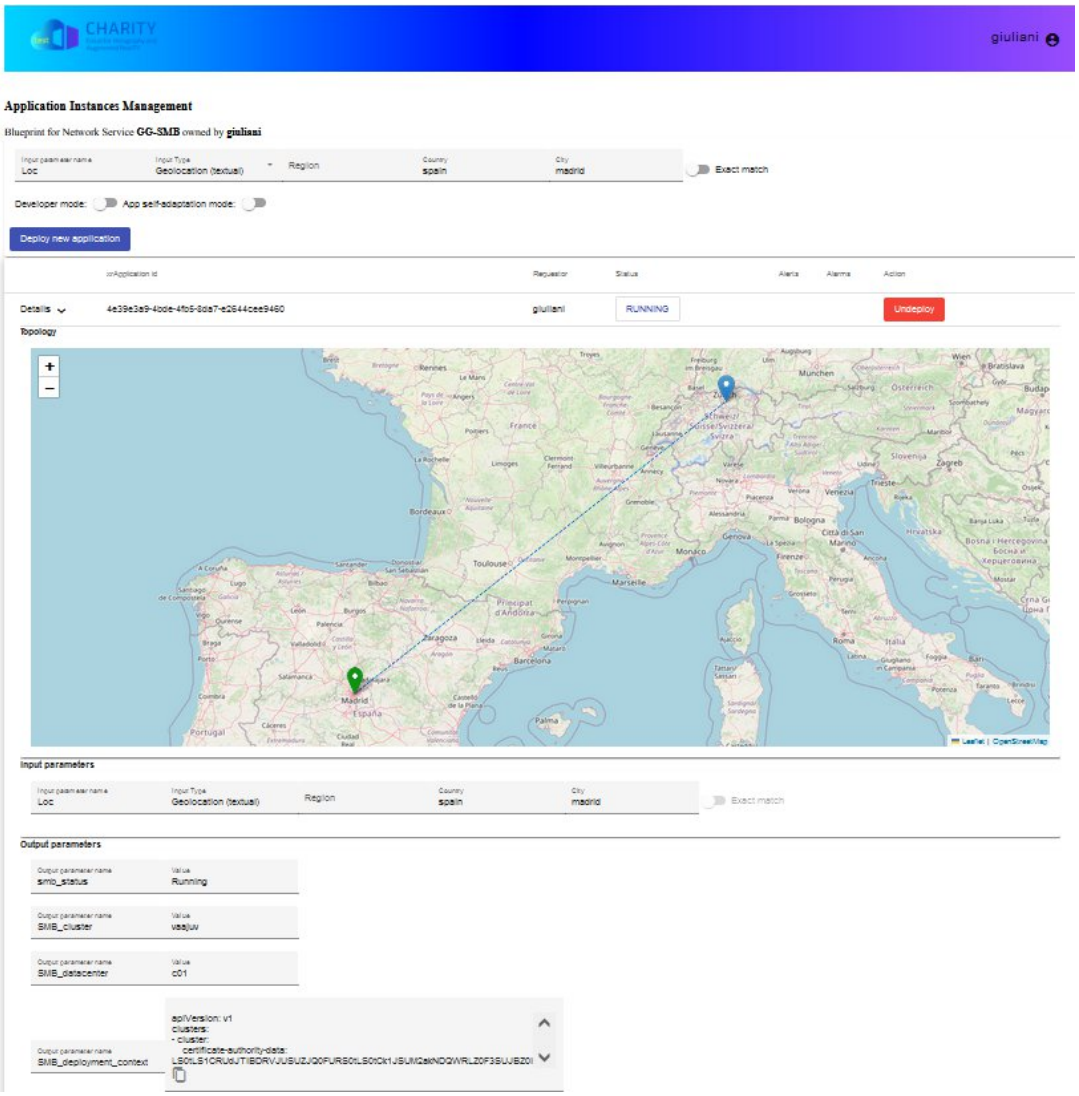


Figure 16 - Placement after re-deployment

3.1.3 Evaluation tests, data collection and analysis

The following section outlines the evaluation of the features tested by the previously described scenarios.

1. Simple XR application deployment:
    a. Simple to use AMF Editor to create Blueprints
    b. Intuitive AMF interface for XR application management
    c. Successfully tested deployment placement with multiple diversified workload and network topologies
    d. Useful information returned to XR developer to access XR applications endpoints
        i. If the domain does not have public addresses for K8s cluster ingress, a VPN needs to be setup to access deployed resources
        ii. If the domain does not have a dynamic integration with DNS, K8s cluster ingress IP address needs to be manually inserted into client HOST file
    e. Useful information returned to XR developers to troubleshoot XR application resources in K8s
        i. effective kubectl configuration support, but it required K8s skills
2. Multi-cluster Distributed XR application
    a. Simple creation of complex Blueprints using AMF editor
        i. simple topology graph provides a clear summary
    b. Success tests with different XR applications resource requirements (e.g. CPUs, GPUs, etc) that force certain components to powerful enough domains
    c. Transparent network connectivity among XR applications VNFs allow programmer to interconnect them in a simple way
        i. same internal resource endpoints independently on their placements in different domains
3. Adaptation of XR application deployment for best QoS/QoE
    a. Easy declaration of relevant metrics for QoS/QoE in AMF editor
    b. Automatic setup of monitoring, forecasting and alarming systems
    c. Automated XR application life cycle management provided by CHARITY platform
        i. Optional support for custom metrics and self-adaptive management
    d. Example of OODA loop

3.2 Point Cloud Encoding/Decoding

3.2.1 Description, procedure, metrics

Table 9. Description of evaluation subtopic - Point Cloud Encoding/Decoding service

Subtopic Title: Point Cloud Encoding/Decoding service	**Partners:** CNR
Short description and evaluation scope: The Point Cloud Encoding/Decoding (PC E/D) component is used for the fast compression/decompression of point clouds. The main intended use is to transmit a huge amount of coloured 3D points. This may be useful in application context like the ones where a device/display receives coloured 3D points generated on a edge/cloud.	
Related requirements: No particular requirements. GPU is needed to speed up the performance.	
Components involved: Point Cloud Encoding/Decoding component.	
Where are data collected and stored – measurement points:	

The data used has been created specifically for test purposes. The test data has been provided by the SRT. It consists in an animated 3D scene where a person (the Assistant) talks about weather conditions. Another scene where a people is inside a room has been also created and used to test such component (see *Figure 17*). These synthetic scenes are generated in real-time using Unity. The measurements have been conducted with the component in isolation and using the SRT prototype.
When are data collected? The test 3D scenes have been created during the Q1-Q2 2023 period. Before this period other data has been used to conduct preliminary tests on 3D points encoding and to test the component in isolation. This dataset is described in Deliverable 1.7.
Instruments/tools: C++, ffmpeg, GPU shaders
Methodology/Procedure: *in which way data are collected* The PC E/D component is integrated in the UC1-3 Holo Assistant as a software library. The scenes generated by the Unity rendering engine is represented as a set of RGBD images taken from different viewpoints. Since camera calibration is known for each RGBD image, each pixel represents a 3D point with colour. This representation of the point cloud permits to the system, taking into account the viewpoint of the user, to transmit a set of RGBD images around such viewpoints to the holographic display. The holographic display splats the coloured points creating a 3D virtual scene that appears real. The PC E/D is a view-dependent compression algorithm that is specifically designed to compress and decompress efficiently this type of RGBD images.
Metrics to analyse the results Number of 3D points (i.e. resolution of the RGBD images), number of views (i.e. number of RGBD images to compress), Frame-Per-Seconds (FPS).

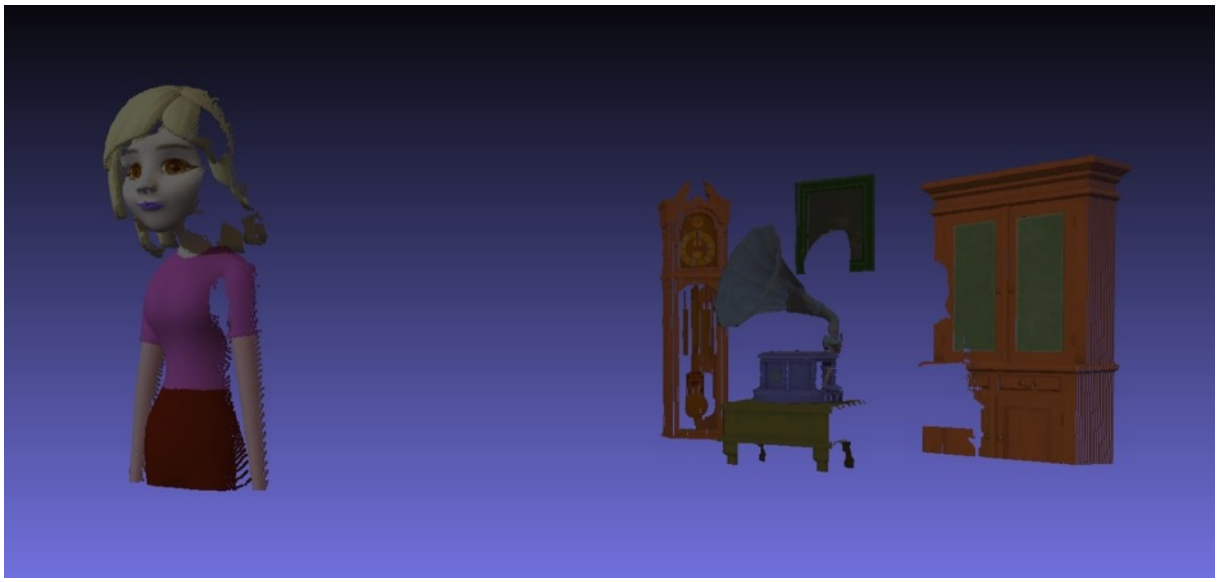


Figure 17 - A test scene reconstructed from 8 RGBD views.

3.2.2 Experimentation Scenarios

Essentially, we have two experimentation scenarios. The first scenario was used during the component's development and for its performance optimization, involving testing the component in isolation. This phase of experimentation utilizes the dataset described in Deliverable 1.7.

The second scenario is the test of the component inside the SRT use case (UC1-3 Holo Assistant). In this second case the PC E/D is used to transmit the 3D-coloured points representing a bunch of views related to the viewpoint of the observer from the Unity engine, that generate them, to the Holographic display. For a schematization of the UC, see Section 2 of the Deliverable 1.2.

3.2.3 Evaluation tests, data collection and analysis

The algorithm at the base of the PC E/D component, and the design motivations, are described in detail in the Section 5.6 of the Deliverable 3.2. This component has been tested separately and in the pipeline of the UC1-3. The overall frame rate measured after the integration for the test scene is around 5 fps. Further optimization of the streaming parameters of the video parts (that is achieved using ffmpeg) obtain a considerable gain of performance, reaching 15-20 fps. This frame rate is measured for the transmission of 8 viewpoints of resolution 1280 x 752. This number of views is sufficient to permit to the user of the holographic display slightly changes of viewpoint without the need to transmit other data (more details about this point can be found in the D3.2). This performance reported are related to the C++ version. The GPU version is obviously more performing, but it has not been tested inside the UC1-3 as this requires a tight integration in the image generation pipeline of the UC. Such integration would allow to save computations and memory transfers between the application and the component. Tests of the GPU version of the component in isolation show promising results. The results indicate that achieving over 30-40 fps with the GPU-integrated version appears feasible.

3.2.4 KPIs assessment

Regarding the general objectives of the CHARITY project, this component has been developed in the ambit of the *Objective #4 – Develop highly interactive and collaborative services and applications*, and it satisfies the *KPI-4.3 Specialized data services support: streaming*, *rendering*, *compression*, *caching and encoding*. The performance obtained by the CPU version, particularly the version with the optimized ffmpeg parameters, satisfies the speed performance required by the Holographic Assistant application to reach an acceptableQoE. The GPU version is more performing, ensuring high levels of QoE even with high-resolution images.

3.3 Mesh Merger

3.3.1 Description, procedure, metrics

Table 10. Description of evaluation subtopic - Mesh Merger service

Subtopic Title: Mesh Merger service	**Partners:** CNR
Short description and evaluation scope: The Mesh Merger service is a XR data service to assemble together pieces of geometry of an indoor environment to set up a corresponding virtual environment for AR applications. The same service, with slightly modifications, can be used to update an existing virtual environment according to the changes of the real environment. The pieces of geometry are assembled in a mesh called *mesh collider*, since it is used to resolve collisions enabling the interaction of the virtual objects with the real environments.	
Related requirements: No particular requirement.	
Components involved: Mesh Merger, Game Server (UC3-1)	
Where are data collected and stored – measurement points: The data about indoor environment are collected on-the-fly through a test application (Game Client) developed by the ORBK which allows to scan a part of the environment using a smartphone equipped with a Lidar. Different mesh colliders of different indoor environments have been created.	
When are data collected? The Mesh Merger has been tested with different acquired single mesh colliders during the Q2-Q3 2023 period. In this last period the Mesh Merger has been modified to be integrated in the CHARITY platform. At this point, it works together with the Game Server, which request to the service to assemble the pieces of geometry acquired at the begin and during the game. The current version works with data collected on-the-fly by the gamers' devices, i.e. smartphone equipped with Lidar camera.	

Instruments/tools: C++, C#, ARKit[2], Unity AR Foundation, TEASER++[3], OpenVDB[4], Node.js
Methodology/Procedure: The methodology for the test procedure is as follows: It has been evaluated the time to transmit the pieces of geometry, i.e. the single mesh colliders, provided by the ARKit on the smartphone device equipped with the Lidar, the processing time for the alignment, and the processing time for the fusion of the aligned mesh to create the final mesh collider. First, this evaluation has considered the component with the data acquired by an application developed by the ORBK. Then, the Mesh Merger component has been turned into a service, based on a REST-API. This service can receive requests of fuse a new single mesh collider into the current mesh collider of the indoor environment or create a new environment for another game. The processing time and the transmission time have been evaluated also in the server version, which exploits Node.js to manage the HTTP requests.
Metrics to analyse the results Data transmission time, processing time, quality visual inspection.

3.3.2 Experimentation Scenarios

A first set of experiments have been conducted was adopted during the development and the initial steps of the integration. In this case, the Game Server directly sent a set of acquired meshes to the Mesh Merger and retrieve the results. In a second round of experiments, the requests are one at a time, i.e. one mesh at a time is aligned and fused to obtain the final mesh collider for the virtual environment, and more than one Game Server can use the same instance of the Mesh Merger. The Mesh Merger processes the requests in an asynchronous way and each Game Server identifies itself by a unique id. This allows two type of communications mode between the Mesh Merger and the Game Server, schematized in Figure 18 and Figure 19.

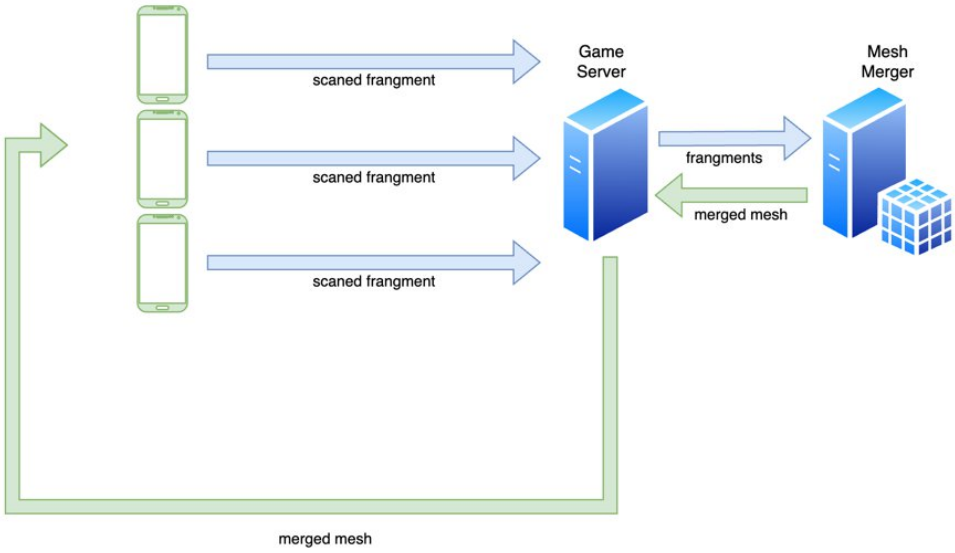


Figure 18 - Each Game Client can send a fragment of scanned environment and trough Game Server it is sent to the Mesh Merger. Game Server is responsible for setting up a merging session with the Mesh Merger Service, sending all the fragments, and after receiving merged mesh distributing it back to all Game Clients connected to given game session.

[2] https://developer.apple.com/augmented-reality/arkit/

[3] https://github.com/MIT-SPARK/TEASER-plusplus

[4] https://www.openvdb.org

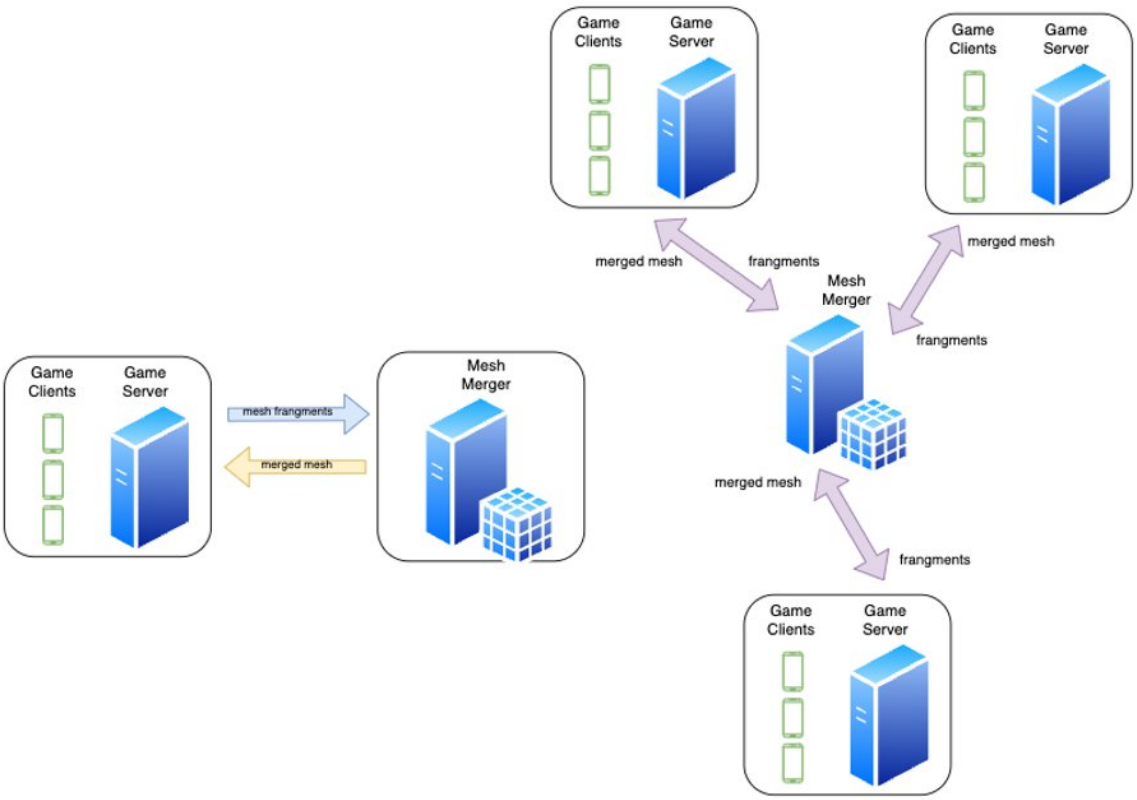


Figure 19 - Each Game Server can open its own mesh-merging session with Mesh Merger Service. It is realised by assigning to each session its unique ID and using it every time Game Server is requesting merge operation. This way, each instance of deployed Mesh Merger Service is able to serve single or multiple Game Servers.

3.3.3 Evaluation tests, data collection and analysis

The tests conducted are based on real data acquired through an ad hoc application developed by ORBK based on the ARKit. The Mesh Merger service implemented is based on Node.js and follows a REST-API paradigm. The registration and fusion algorithm are based on two open-source codes, the TEASER++, for the alignment, and the OpenVDB library, for the fusion of the aligned pieces of geometry into the Mesh Collider, respectively. More details about the alignment and the fusion algorithm can be found in the Section 5.9 of the Deliverable 3.2.

The Mesh Merger service is currently available on the CHARITY platform and can be deployed alongside the ORBK Game Server to meet the needs of UC3-1 Collaborative Gaming Application. In this setup, the Game Server communicates with the Mesh Merger to establish the mesh collider for the game environment. For this AR game, the tests conducted demonstrated that the processing time is sufficiently fast to provide to the gamers a high QoE, (i.e., less than 2 seconds are necessary to download and process a new acquisition into the Mesh Collider). In particular, the transmission time is made efficient by using a binary version of a JSON containing a PLY format of the mesh. Even if this data format is not compact, the number of triangles of a single mesh collider, that is in the order of 100K-200K triangles, is such that it is sufficient for the purpose of an interactive experience and it is easy to manage. The processing is asynchronous, so that multiple users can scan different parts of the indoor environment and set up the game quickly. Even if this service works tightly with the Game Server, it is a general mesh processing data service and it can be used by any graphics application that needs to register and fuse mesh together. If a higher resolution of the merged mesh is required, the service can be further optimized by using entropy encoding of the geometry to maintain reasonable

performance. This simple solution makes it useful not only by other AR application but also for other types of graphics/geometry processing applications.

3.3.4 KPIs assessment

Regarding the general objectives of the CHARITY project, this component has been developed according to the *Objective #4 – Develop highly interactive and collaborative services and applications*, and it fulfils the *KPI-4.3: Specialized data service support: streaming, rendering, compression, caching, and encoding*.

3.4 CHARITY Edge Storage (CHES)

3.4.1 Description, procedure, metrics

Table 11: Description of evaluation subtopic - CHARITY Edge Storage component (CHES) & CHES Registry sub-component

Subtopic Title: CHARITY Edge Storage component (CHES) & CHES Registry sub-component	**Partners:** HUA, DOTES
Short description and evaluation scope: CHES component aims to provide a hybrid distributed cloud/edge storage framework spread across heterogeneous edge and cloud nodes with considerations on performance (QoS), emphasizing on the resolution of the problem of data distribution and offloading based on CHARITY application's requirements. CHES Registry aims to provide a localized Docker registry using CHES as its storage backend. It combines the official Docker registry image with Kubernetes orchestration, CHES object storage backend, and a set of automated deployment and configuration scripts in order to store and distribute container images closer to the edge.	
Related requirements: No particular requirements.	
Components involved: CHARITY Edge Storage component and CHES Registry sub-component	
Where are data collected and stored – measurement points: CHES component • The data originating from DOTES UC (UC2-2 VR Tour Creator) are acquired and stored within a designated bucket in MinIO managed by CHES. This bucket serves as the repository for the collected data. CHES Registry sub-component • The data collected and stored for evaluation primarily comprise the 10GB-sized LSPart1 VM image utilized in the UC2-1 VR Medical Training (ORAMA). The VM image is stored within a bucket in MinIO managed by CHES.	
When are data collected? The data was collected during the 2nd quarter of 2024.	
Instruments/tools: MinIO, MinIO client (mc), Prometheus, Kubernetes Dataset Lifecycle Framework provided by IBM's Datashim, Python3, shell, Docker	
Methodology/Procedure: CHES component • To assess the component's performance and effectiveness, several metrics are collected utilizing the Prometheus system. The performance evaluation was performed through Locust, an open-source load-testing framework that enables the definition of user behaviour and supports running load tests distributed over multiple machines. CHES Registry sub-component • The effectiveness of the sub-component is evaluated through the latency involved in fetching the VM image from both a remote and the local registry.	
Metrics to analyse the results	

- Cache hit ratio
- Transaction rate
- Latency

3.4.2 Experimentation Scenarios

Three distinct experimental scenarios were devised to evaluate CHES and CHES Registry. The first two scenarios aimed to measure the cache performance and the transaction rate of CHES, while the third scenario focused on assessing the feasibility and efficiency of the CHES Registry, specifically in terms of latency.

3.4.3 Evaluation tests, data collection and analysis

Assessment of CHES cache performance

The caching performance of CHES was assessed with the utilization of Locust. For the purposes of the experiment, 20 users were configured to execute distributed query requests (read operations). Metric data were collected using Prometheus agents running on the node responsible for data storage. Specifically, these metric data was collected at 2-minute intervals throughout the operational span of the component, i.e. for the whole duration that CHES was active and ready to serve data requests. The performance of MinIO's native "disk cache" feature was evaluated using a collection of small to medium binary files ranging from 5MB to 99MB. These files, originated by DOTES UC (UC2-2 VR Tour Creator), comprised the evaluation dataset stored in a MinIO bucket managed by CHES.

The most important metric for assessing the cache performance of CHES is the Cache Hit Ratio, defined as follows:

Hit Ratio = #cache hits / (#cache hits + #cache misses)

A cache hit denotes the successful retrieval of content from the cache instead of the original storage. Conversely, a cache miss indicates the absence of the requested data in the cache memory, prompting a query to the origin storage. Following a cache miss, the request is redirected to the origin storage, and upon retrieval, the content is transferred to the user and, if feasible, cached for future access. The metrics for cache hits and cache misses were derived from Prometheus, collected at a two-minute interval. Specifically, *minio_cache_hits_total* and *minio_cache_missed_total* were utilized to quantify cache hits and cache misses, respectively.
The results indicate a cache hit ratio of 93%: Hit Ratio = 6124 / (6124 + 412) = 0,93 = 93%

Analysis of CHES transaction rate performance

The transaction rate in a storage system refers to the capacity at which the system can handle read and write operations, typically measured in transactions per second (TPS) or requests per second (RPS). It indicates the system's ability to process data access requests efficiently and quickly, reflecting its overall performance and responsiveness.

The evaluation of CHES's transaction rate was performed using Locust, yielding an average result of 6.1 RPS, as illustrated in Figure 20. In alignment with the preceding experiment, a configuration of 20 users was established to execute distributed query requests (read and write operations) over the data provided by DOTES UC (UC2-2 VR Tour Creator) and stored within a bucket in MinIO managed by CHES.

Figure 20 - Average RPS - CHES

Additionally, a blockchain database, namely BigchainDB was explored as an alternative solution. More specifically, BigchainDB supports both blockchain (decentralization, immutability, and owner-controlled assets) and database properties (high transaction rate, low latency, indexing, and structured data querying). Experimental results demonstrated that CHES is able to achieve a higher RPS compared to BigchainDB (3.9) for a specific class of experiments.

Evaluating CHES Registry sub-component

The seamless delivery of XR applications on resource-constrained edge devices, poses unique challenges due to limited network bandwidth, latency constraints, and intermittent connectivity. Additionally, the size of XR application images is often significant, and downloading these images from remote repositories can put a burden on the limited network bandwidth and introduce significant latency. CHES Registry sub-component serves as a crucial component, addressing the need to bring application images closer to the edge while minimizing network traffic and image download durations.

The feasibility and efficiency of the CHES Registry are evaluated through the examination of one specific use case scenario: UC2-1 VR Medical Training. During the pilot evaluations of the VR medical training application, retrieving the 10GB-sized LSPart1 VM image from a remote repository led to considerable network congestion, causing delays in image download and concurrent network operations. This issue was addressed by pre-positioning the VM image within the CHES Registry on the same edge node before initiating a new VR session request. This change, which involved deploying the new VM from a local repository rather than a remote one, significantly reduced deployment times. In the tests without CHES Registry pre-loading, the application took over 10 minutes to deploy, and in some cases, even up to 20 minutes. With CHES Registry pre-loading, deployment times dropped to 1-2 minutes. These results indicate that CHES Registry achieved deployment times up to 10 times faster than raw Kubernetes deployment.

Overall, the evaluation reveals a significant reduction in application deployment time, indicating the positive impact of the proposed solution.

3.4.4 KPIs assessment

Regarding the general objectives of the CHARITY project, this component has been developed in the ambit of the Objective #2 - Provide holistic support for the orchestration of advanced media solutions. More specifically, the KPIs that are satisfied are the following:

- KPI-2.2 Storage formats: at least one (block, file, object)
    - As already mentioned, as a storage solution, an open-source framework created by IBM is utilized, called MinIO. This framework uses object storage over block storage, so it is in fact a combination of the two systems, preserving the lightweight distributed nature of block storage while providing the plethora of metadata and easy usage of the object storage.
        - Extensive research has been conducted in the field of storage solutions in edge computing infrastructures. A scientific journal entitled “A Lightweight

Storage Framework for Edge Computing Infrastructures/EdgePersist" has been published in Software Impacts (Elsevier) presenting the proposed edge storage solution.

- KPI-2.3 Edge storage hit rate: higher than 70%
  - The native "disk cache" feature of MinIO has been utilized. Disk caching feature refers to the use of caching disks to store content closer to the tenants allowing users to have the following: i) object to be delivered with the best possible performance and ii) dramatic improvements for time to first byte for any object. Experimental results revealed a hit ratio exceeding 93%.
- KPI-2.4 Blockchain for edge storage transaction rate: more than 4 transactions per second
  - A blockchain database, namely BigchainDB was explored as an alternative solution. Experimental results demonstrated that MinIO is able to achieve a higher transaction rate (6.1) compared to BigchainDB (3.9) for a specific class of experiments.
    - A scientific journal in the context of performance of storage systems in edge computing infrastructures entitled "Performance Analysis of Storage Systems in Edge Computing Infrastructures" has been published in Applied Sciences (MDPI) to the Special Issue Cloud, Fog and Edge Computing in the IoT and Industry Systems.
  - In addition, we conducted extensive experiments within a distributed computing environment, utilizing a configuration consisting of four nodes, and once again, we observed consistent outcomes. Specifically, MinIO demonstrated a superior transaction rate in comparison to BigchainDB and also achieved a better performance in both read and write operations. This reaffirms the robustness and efficiency of MinIO across varied deployment scenarios, further underscoring its potential as a high-performance data storage solution.
    - A scientific conference paper entitled "A Study on the Performance of Distributed Storage Systems in Edge Computing Environments" has been accepted to the 15th IEEE International Conference on JointCloud Computing (IEEE JCC 2024), showcasing the aforementioned results.

3.5 CHARITY Adaptive Scheduling of Edge Tasks (ASET)

3.5.1 Description, procedure, metrics

Table 12. CHARITY Adaptive Scheduling component

Subtopic Title: CHARITY Adaptive Scheduling component	**Partners:** TID
Short description and evaluation scope: Adaptive scheduling (ASET) component focuses on the problem of scheduling inference queries that have to be allocated to DL models/resources available in the edge-cloud network at short time-scales (i.e., few milliseconds) with considerations on performance (QoS) and security, emphasizing on offloading workloads depending on CHARITY application's requirements. ASET components is based on Kubernetes, Kafka, Prometheus, and Reinforcement Learning technologies. ASET aims at selecting the best policy from a set of several policies in a realistic network settings and workloads of a large European ISP. Some policies enable real-time applications in realistic settings, but a dynamic scheduling policy is required to adapt to different network conditions, topologies, and workloads. Our results show the dynamic policy automatically adapts to conditions and effectively improves performance over baselines for edge-enabled deployments.	
Related requirements: No particular requirements.	
Components involved: CHARITY Adaptive Scheduling component	
Where are data collected and stored – measurement points:	

Client generator has been created to simulate incoming app users following a Poisson distribution. This generator runs on average lambda clients per minute querying the scheduler of a given geographical area (antenna). Once spawned, each client requests for processing a stream featuring randomized characteristics in terms of frame rate, required end-to-end latency, required model accuracy, frame sizes, stream duration. To capture realistic queries characteristics, we modelled metrics of generated streams according to the reference edge applications in Table 10.
When are data collected? The data was collected during the 1st semester of 2024.
Instruments/tools: Prometheus, Kubernetes, Pytorch, Kafka, Python3, shell, Docker, Object Detection algorithms
Methodology/Procedure: *in which way data are collected* A series of tests have been performed on the platform with different settings in order to compare the static policies over a distributed pool of edge resources. The comparison is based on success, failure and rejection rates.
Metrics to analyse the results Latency, success queries, rejection queries and failure queries.

Table 13. Characteristics of reference applications

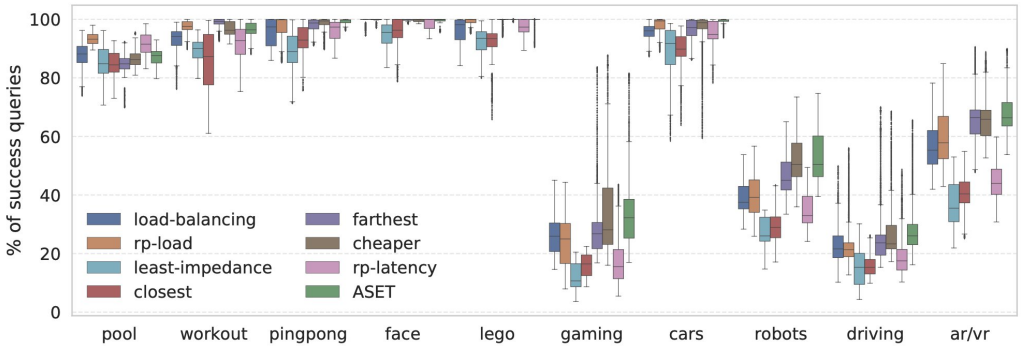
Edge app	Tolerated delay	Frame rate	Streams duration	Required accuracy
Pool	95 ms	5 FPS	5-10 s	10 mAP
Workout Assistant	300 ms	2 FPS	90 s	10 mAP
Ping-pong	150 ms	15-20 FPS	20-40 s	15 mAP
Face Assistant	370 ms	5 FPS	1-5 s	30 mAP
Lego/Draw/Sandwich	600 ms	10-15 FPS	60 s	25 mAP
Gaming	20-30 ms	25 FPS	10-30 m	35 mAP
Connected Cars	150 ms	10-15 FPS	15-30 m	40 mAP
Tele-Robots	25-35 ms	10 FPS	5 m	40 mAP
Remote-driving	20-30 ms	20 FPS	15-30 m	50 mAP
Interactive AR/VR	30-50 ms	25 FPS	30-60 s	35 mAP

3.5.2 Experimentation Scenarios

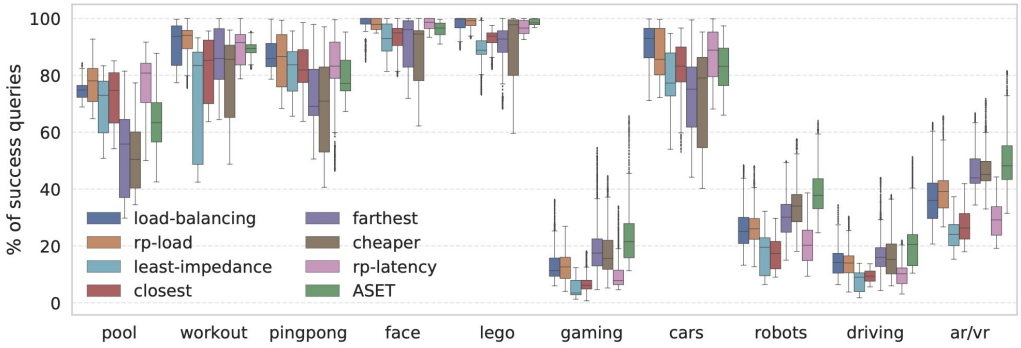
Two distinct experimental scenarios were devised to evaluate ASET. These scenarios aimed to measure the success, failure and rejection rates and the feasibility of the ASET by changing the network topology, cloud and edge.

3.5.3 Evaluation tests, data collection and analysis

Initially, we compared the performance of the baseline policies, e.g., closest, farthest, load balancing, least impedance, random, rp-latency, rp-load and cheaper, distinguishing results for different applications described above. As a performance metric we consider the percentage of queries that are successfully processed by the system satisfying the application QoS requirements. Figure 18 shows results of multiple runs with lambda = 60, suggesting that there is no one-size-fits-all policy, as various applications may benefit differently from each policy. Varying the rate of stream requests on the antenna may further increase the uncertainty of relying on a single policy.

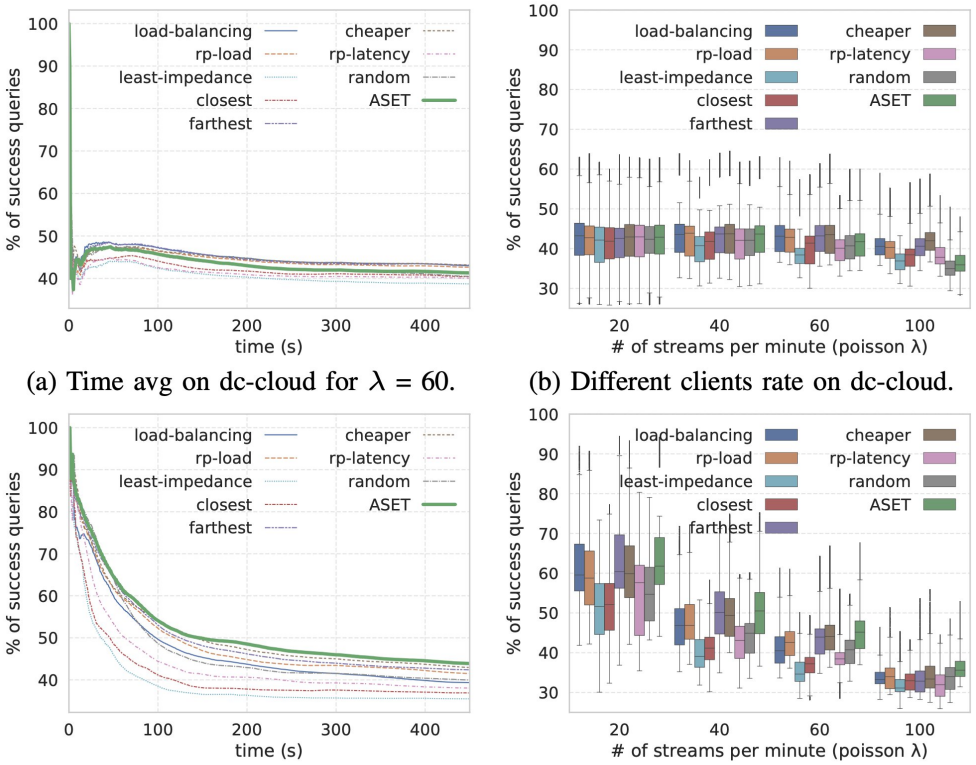


(a) Average values for clients rate $\lambda = 60$



(b) Average values for episodes with dynamic clients rates.

Figure 21 - Success percentage for different apps on the full-edge topology.



(a) Time avg on dc-cloud for $\lambda = 60$.

(b) Different clients rate on dc-cloud.

(c) Time avg on co-dc-cloud for $\lambda = 60$.

(d) Different clients rate on co-dc-cloud.

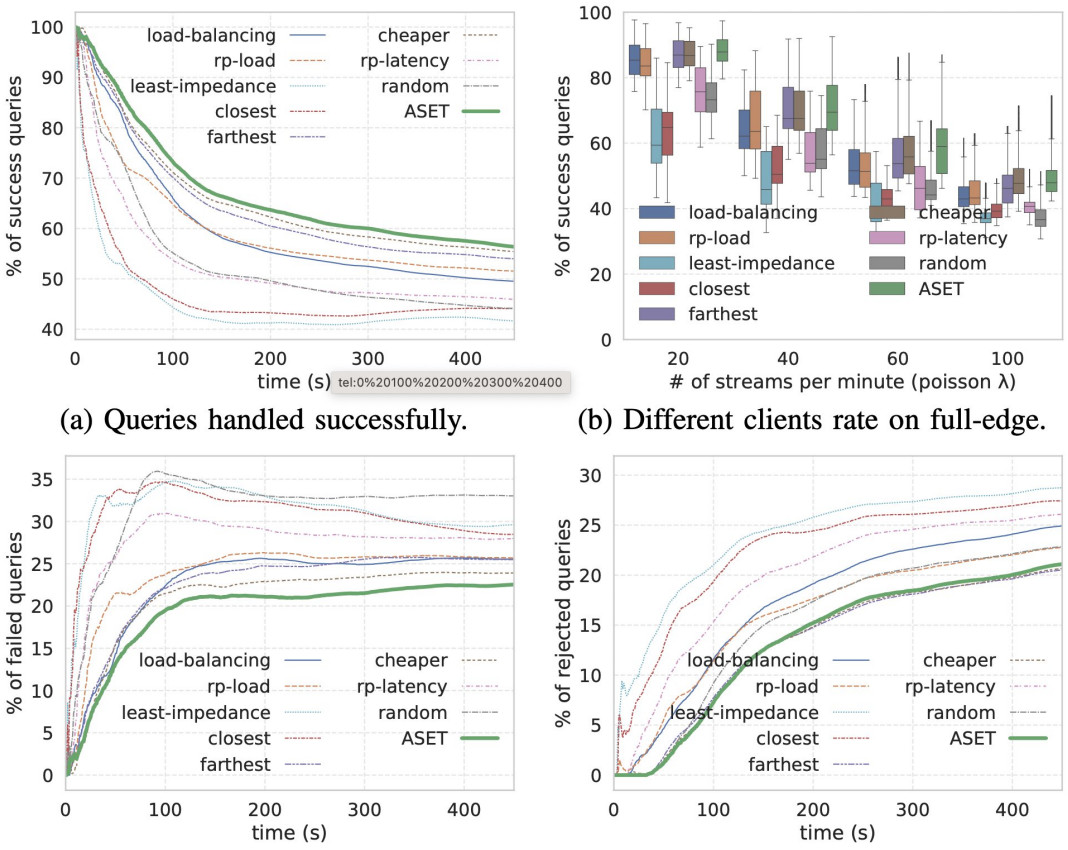
Figure 22 - Performance of ASET compared with static policies for (ab) the dc- cloud topology and (cd) the co-dc-cloud topology.

Cloud deployment

First, we focus on testing the performance when all the available resources are located in a few centralized clusters. Static policies have small differences in performance and a dynamic approach has little room for improvement. The results for the dc-cloud topology, shown in Figure 19, indicate that, for this topology, ASET does not improve over static policies, and it even performs worse for higher lambdas. However, moving some resources to Central Offices (co-dc-cloud topology) makes a huge difference. In general, all the policies achieve a higher success ratio on this configuration, as they can exploit the additional lower latency spots, and the higher level of distribution gives to ASET a certain margin of improvement.

Edge deployment

As shown in Figure 20, the benefits of using a dynamic scheduling approach become more concrete in a full-edge topology, where resources are better distributed on multiple smaller clusters in different locations. In fact, the dynamic approach of ASET is able to achieve a constant improvement over any static policy, with a higher success ratio over time while maintaining the same rejection rate as the best static-policy. ASET effectively reduces the number of queries that are handled violating one or more QoS requirements.



(a) Queries handled successfully. (b) Different clients rate on full-edge.

(c) Queries delivered with QoS violations. (d) Queries rejected for lack of resources.

Figure 23 - Performance of ASET compared with static policies for the full-edge topology. (a) (c) and (d) show averages of multiple runs with λ = 60.

3.5.4 KPIs assessment

Regarding the general objectives of the CHARITY project, this component has been developed in the ambit of the Objective #2 - Provide holistic support for the orchestration of advanced media solutions focusing on distributing jobs on edge device architectures KPI-2.1.

4 Use case Evaluation and results

4.1 UC1-1 Holographic Concert and UC1-2 Holographic meetings

4.1.1 Description, procedure, metrics

Table 14. Description of evaluation subtopic - Holographic Concert and Holographic meetings

Subtopic Title: Holographic Concert and Holographic meetings	**Partners: HOLO3D**
Short description and evaluation scope: Our plan is to measure latency and data rate. Since the number of consumers and devices is finite and quite low, the latency is expected to be related to the internet connection, rather than anything else. We need to conduct testing to determine the maximum acceptable latency that is acceptable while not degrading the QoE (video quality and synchronization).	
Related requirements: F_UC1_01: CHARITY provides Cloud server with resources necessary to achieve KPIs. F_UC1_02: CHARITY provides cloud-based software to receive, decompress and render / modify the content in the cloud in real time. F_UC1_03: CHARITY software renders in real time several types of pre-set video modes and resolutions, for several types of Holographic Displays. F_UC1_04: APPLICATION PROVIDER provides speaker PC, video camera, lights, black background, secondary screen. F_UC1_05: APPLICATION PROVIDER provides speaker PC with software to retrieve the raw, 2D video from the video camera and send it to the Cloud server. F_UC1_06: APPLICATION PROVIDER provides client PC, Holographic Display, webcam, mic for 2-way communication with the Speaker PC. F_UC1_07: APPLICATION PROVIDER provides client PC with software to send live video/sound stream to the Cloud server. F_UC1_08: APPLICATION PROVIDER provides client PC with software to receive the scrambled, 3D adapted video from the Cloud Server and send it to the Holographic Display. F_UC1_09: APPLICATION PROVIDER provides client PC with software to synchronize with the other connected client PCs. F_UC1_10: APPLICATION PROVIDER, the software F_UC2_08 can choose to retrieve a different type of scrambled, 3D adapted stream from the Cloud server according to the connected Holographic Display. F_UC1_11: APPLICATION PROVIDER provides speaker PC with software to convert the shared content (jpg, pdf, doc, ppt, mp4) to the same type of raw,2d video as in F_UC1_05 (Holographic meetings scenario). NF_UC1_01: The video resolution should be > than full HD (1920x1080) @ 30 fps. NF_UC1_02: Average latency between receiving the raw, 2D video stream from Speaker PC and rendering it for the specific Holo Display resolution and format required by the Client PC<= 30-600 Seconds NF_UC1_03: Average latency between receiving the raw, 2D video stream from Speaker PC and rendering it for the specific Holographic Display resolution and format required by the Client PC<=1000ms (second scenario).	
Components involved: Cyango-media-server	
Where are data collected and stored – measurement points: Data is directly computed in the Speaker and Client PCs. We do not intend store any data.	
When are data collected? Several sessions of 60 minutes each.	

Instruments/tools: Full hd/4k cameras that support RTMP streaming,
Methodology/Procedure: Data is directly computed in the Speaker and Client PCs.
Metrics to analyse the results • Available Incoming Bitrate • Available Outgoing Bitrate • Bytes Discarded On Send • Bytes Received • Bytes Sent • Current Round-Trip Time • Total Round Trip Time

4.1.2 Experimentation scenarios

The video livestreaming scenario involves a speaker streaming from a streaming device (such as a camera or webcam) to the cyango-media-server component. This component initiates data processing and optimization to deliver real-time streaming video and high-quality audio.

This scenario involves cyango-media-server, where the goal is to achieve an average latency < 20 ms.

The same applies to the client PC connected to the holographic display, essentially performing the same function in reverse.

4.1.3 Evaluation tests, data collection and analysis

Initial tests: Video Streaming over wired local network

Our first tests were relevant to both UC1-1 and UC1-2 use cases.

Our initial tests were conducted using TCP. As anticipated, the stability was satisfactory, but latency was high.

Over a 1gb wired connection, we observed a 5000-7000 ms delay between the Musician and the Client PCs when streaming a 1280x720 video at 25 fps with approximately 3500 kbps. The latency was mostly induced by the local video manipulation component that vastly depends on the computer performance.

We utilized the most challenging template for the Dreamoc Diamond, a four-sided holographic device

We then transitioned to UDP for local streaming. We made some video and error handling optimizations, which slightly improved latency to 3000-4000 ms for the same 1280x720 video stream at 25 fps and ~3500 kbps. However, we encountered another issue: the sound was no longer synchronized with the video stream. We used the same video manipulation template designed for the most challenging four-sided holographic device.

The results were not conclusive, as performance largely depended on the hardware configuration of both the Musician's and Client PC. The observed latency was much higher (up to tenfold) than expected for a local streaming solution, leading us to conclude that a cloud-based solution with significantly enhanced computing power was necessary.

Video Streaming – Cloud Server

We successfully integrated our local streaming app with the CHARITY Edge Cyango-media-server. Initially, we tested various local and web streaming protocols, but most yielded unsatisfactory results due to high latency. As a result, we opted to implement the WebRTC protocol.

The streaming occurred during two sessions of 2 hours each, and the following results were recorded.

For the sake of consistency, we tried to use the same video settings for the stream:

- Device: Microsoft LifeCam HD-3000
- Resolution: HD (1280x720)
- Bitrate (kbps): 2500-3500
- Frame rate: 25
- Video codec: H264

The findings indicate that latency has been successfully reduced to below 1000 ms, with synchronized audio and video. These results are derived from the analysis of raw, unedited videos, given that the development of the cloud video manipulation component is ongoing.

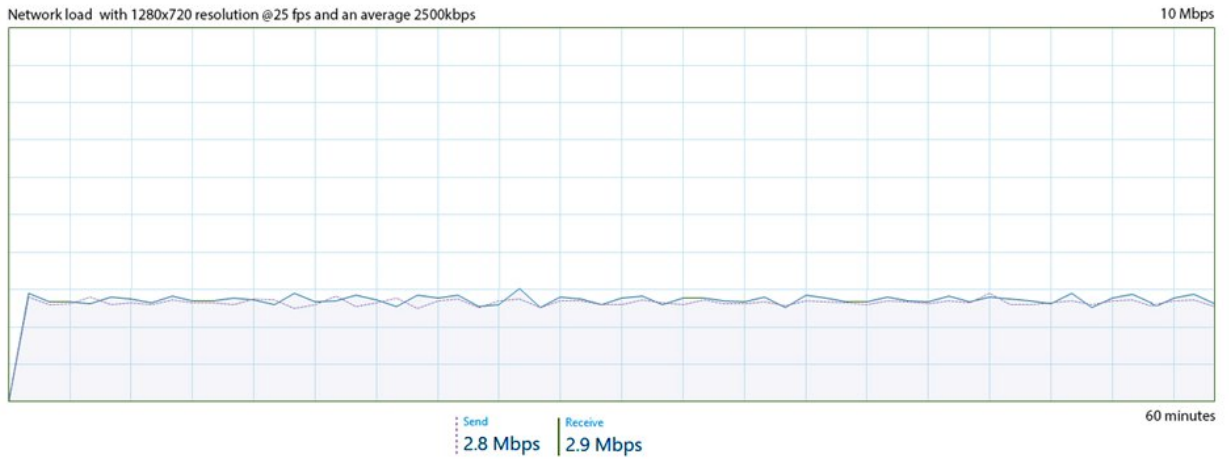


Figure 24 - Network load with 1280x720 resolution @25fps and an average 2500kbps

Network load shows a stable average of 2.8 Mbps sent and 2.9 Mbps received over a 60-minute session.

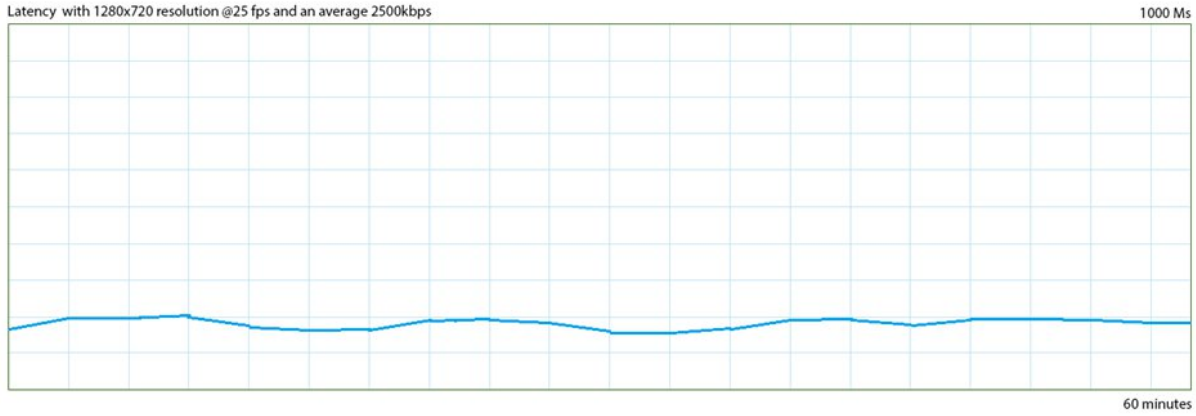


Figure 25 - Latency with 1280x720 resolution @25fps and an average 2500kbps

The latency graph indicates a relatively stable latency of 150-200 ms on average over a 60-minute session. While this raw, unaltered video stream exhibits latency levels that are insufficient for the Holographic Concert Use Case, ongoing testing will involve the cloud video manipulation component. Upon its completion, subsequent tests will assess any additional latency introduced by video manipulation.

We have integrated the video editing tool for the large Holographic Device (Dreamoc Diamond) and have also incorporated sound. Our findings indicate that the impact on performance is negligible. Despite testing with longer-distance connections (specifically, between Bucharest-Istanbul and Bucharest-Munich), we have maintained latency under 1000 ms.

4.1.4 KPIs assessment

KPI-UC-1.1: Average latency < 20 ms.

Description: This KPI measures the time delay from the transmission of input data to the reception of 3D-point cloud data. Ensuring low latency is crucial.

The average latency currently exceeds 20 ms. It remains uncertain whether this meets the requirements for the Holo Concert use case; further tests are necessary, particularly after the completion of the video manipulation and synchronization components.

KPI-UC-1.2: Decrease in bandwidth by 50%

This KPI focuses on reducing the amount of data transmitted over the network by half. This is crucial for optimizing network resources and reducing costs.

This has not yet been achieved, but the following implementation strategies will be attempted:

- Data Compression: Implement advanced data compression techniques to reduce the size of the transmitted data.
- Efficient Data Encoding: Use efficient encoding schemes that reduce the data size without significant loss of quality.
- Selective Data Transmission: Transmit only essential data needed for the task, possibly using data filtering or aggregation methods.
- Caching: Use caching mechanisms to store frequently accessed data locally, reducing the need to transmit data repeatedly.

KPI-UC-1.3: Frame rate of the holographic visualization >= 30Hz

Description: This KPI ensures that the holographic visualization operates at a minimum of 30 frames per second (FPS), providing a smooth and seamless visual experience. This has been achieved, further improvements can be obtained through the following Implementation Strategies:

- Optimize Rendering Pipeline: Enhance the rendering pipeline to ensure efficient processing of visual data, maintaining high frame rates.
- High-Performance Graphics Hardware: Use high-performance GPUs to handle intensive rendering tasks.
- Efficient Resource Management: Allocate resources effectively to ensure consistent frame rates, avoiding bottlenecks in the rendering process.
- Parallel Processing: Utilize parallel processing techniques to distribute the rendering workload across multiple processors or cores.

KPI-UC-1.4: Data services required (raw data streaming, rendering, compression, caching, encoding) =>5.

Description: This KPI indicates that at least five different data services are necessary for the use case, ensuring a comprehensive and robust data processing workflow.

- The tests were made with five different data services: i.e.raw data streaming, rendering, compression, caching, encoding.

4.1.5 Benefits from the use of the Platform/Component

The CHARITY platform provides three key benefits to UC1-1 and UC1-2:

- Cost-efficient video processing capabilities— by optimizing how and where the video editing is performed and by removing the dependency on local third party software;
- Scalability and flexibility — CHARITY platform can quickly scale resources up or down based on demand, providing flexibility to handle varying workloads. The local software that was used before needed a separate licence for every instance run simultaneously.
- Reliability and failover elastic mechanisms for ensuring high availability and minimizing downtime. The initial local software did not offer any reduncy nor failover mechanisms.

4.2 UC2-1 VR Medical Training

4.2.1 Description, procedure, metrics

Table 15. Description of evaluation subtopic - Realistic simulation in VR medical training

Subtopic Title: Realistic simulation in VR medical training	**Partners: ORAMA**
Short description and evaluation scope: ORAMA plans to use the metrics of latency, data rate and number of users in order to determine the maximum latency that is supported in relation to the number of concurrent users in a VR session. The end-to-end latency derives from three factors: processing in the edge /cloud resources, transmission over the network and processing on the HMD. End-to-end latency includes the rendering and the streaming latency, HMD Decoding time and Time to latch frame as well as the jitter and refers to the time since a user movement is registered by the system and for the corresponding image to be displayed on the headset's screen. In addition, we aim to approximate both the data cost and the latency cost (network related) each additional user adds to a VR session.	
Related requirements: **F_UC2_01:** APPLICATION DEVELOPER: Use the mirror networking service or similar for matchmaking, creation of session and selection of an already existing session (IP, location, userid master) photon. **F_UC2_03:** APPLICATION DEVELOPER: Session management through a relay server or message broker in the cloud. **F_UC2_07:** APPLICATION DEVELOPER: The application component running on the HMD should be aware of the connected resources (app instance on edge) where part of the application has been offloaded. **F_UC2_08:** APPLICATION DEVELOPER: The application running on the HMD should be able to connect via standardized protocols to the resources (app instance on edge) where part of the application has been offloaded. **F_UC2_11:** APPLICATION DEVELOPER: Support continuous streaming of produced frames, that combines two images (one per eye) per user, from the edge resource node to the HMD. **F_UC2_13:** APPLICATION DEVELOPER: The resource discovery mechanism of CHARITY should offload part of the application functionality from the HMD to nearby edge resource considering lowest average latency. **F_UC2_17:** APPLICATION DEVELOPER: Establish communication of the HMD and the Remote Service (RS) in the cloud/edge when launching the app on the HMD. **NF_UC2_01:** USER, APPLICATION DEVELOPER: Round trip time (RTT) latency <15ms. **NF_UC2_04:** USER, APPLICATION DEVELOPER: Connectivity from user HMD device <10 ms. **NF_UC2_10:** APPLICATION DEVELOPER: Receive error messages on potential problems with existing resources, continue the VR app by communicating with another newly discovered resource (discovery and placement). **NF_UC2_17:** USER, APPLICATION DEVELOPER: Performed actions from all users must be synchronized to the output rendered image of each individual user's HMD with lowest average latency.	
Components involved: Activation Proxy, LSPart1, HMDApp, Photon	
Where are data collected and stored – measurement points:	

Network statistics regarding the streaming of rendered images is measured on the LSPart1 and is stored on the HMD. User login information, and User analytics is stored on the LSPart1 and sent to the Microsoft Azure Cloud.
When are data collected? Each session of tests lasts approximately 5 minutes.
Instruments/tools: Mobile HMD, LSPart1, LSPart2
Methodology/Procedure: Data is captured and timestamped on the HMD from the streamer client and stored temporarily on the HMD. At specific time intervals, every 40 secs, all captured metrics are transmitted to the Analytics Engine through the AMF. User input can greatly vary in each VR session. We aim to scale to over 50 users in the experiments.
Metrics to analyse the results • Frames Per Second: The frames per second rendered by the machine • Frame Delivery Time: Time from the creation of the frame to the time it is displayed on the HMD • Bandwidth Utilization (kbps): Throughput used by the streaming application • Bandwidth Utilization (%): Estimated throughput usage based on available throughput estimates • Round Trip Delay (ms): Network Latency from the Streaming Machine to the HMD • Jitter (us): Variance of Network Latency • Packets Received (total): Total packets sent to the HMD • Packets Lost (cumulative): Total packets lost by the HMD • Packets Dropped (cumulative): Total packets dropped due to high latency • Packets Lost (Percentage): Percentage of packets lost

4.2.2 Experimentation scenarios

A number of testing sessions were conducted, in which experiments were incrementally staged to reach a large number of CCUs, exploiting both real users with available HMDs and simulated users via the exploitation of bots. Simulated Bot users, spawned by a script developed by ORAMA, behave as a real HMD user and generate the same overhead on the network bandwidth and on the CPU load of the LSPart2 server, reaching the target of 50 CCUs. The experiments in this deliverable aimed to separately assess the Use Case components and the orchestration of the CHARITY platform. On one hand, the focus was on evaluating the deployment of the developed components within one of the project's testbeds. On the other hand, the goal was to assess the metrics and KPIs before further leveraging the orchestration functionalities of the CHARITY platform. The HMDs for the experiment were provided from ORAMA and the participants were members from the ORAMA team located in Greece. The LSPart1 and LSPart2 components were deployed in CHARITY platform, specifically at the Sofia Testbed provided by OneSource. For each LSPart1, an XR Application Activator Blueprint with a docker image is created and acts as a proxy, which activates an always-on Machine, which runs the LSPart1 software, in the ORAMA Lab. In addition, another Machine was exploited for simulating up to 55 users, depending on the scenario, via bots. The LSPart2 was deployed in a separate Blueprint as a docker image. Each participant's HMD was connected to a separate machine, where the LSPart1 was running. The HMD would connect to the LSPart1 proxy created in the CHARITY Platform and would get assigned on a physical machine where it would then connect. The machines were connected using Ethernet. The HMD was connected through 5Ghz Wi-Fi. The following tests were conducted:

Test 1 - 1 HMD User and 20 Bots

The test was conducted via the deployment of 1 container for the XR Activation Proxy (Blueprint with Docker image), 1 machine (Windows 10 with RTX 3060 GPU) for the LSPart1 and 1 container for the LSPart2 (Blueprint with Docker image), and 1 machine (Windows 10) to run the 22 Bots.

Test2 - 2 HMD Users and 30 bots

The test was conducted via the deployment of 2 containers for the XR Activation Proxy (Blueprint with Docker image), 2 machines (Windows 10 with GTX 1060 GPU) for the LSPart1 and 1 container for the LSPart2 (Blueprint with Docker image), and 1 machine (Windows 10) to run the 33 Bots.

Test3 - 3 HMD Users and 50 bots

The test was conducted via the deployment of 3 containers for the XR Activation Proxy (Blueprint with Docker image), 3 machines (3 Windows 10, with 1x RTX 3060 GPU and 2x GTX 1070 GPUs) for the LSPart1 components, 1 container for the LSPart2 (Blueprint with Docker image), and 1 machine (Windows 10) to run the 55 Bots.

4.2.3 Evaluation Tests, data collection and analysis

Test 1 - 1 HMD User and 20 Bots

In the first test, the distributed VR pipeline was evaluated with one HMD user and 20 bots. The system maintained a steady frame rate of 75 fps, which is suitable for interactive VR simulations, ensuring a smooth and immersive experience. Packet loss was minimal, indicating reliable network performance. The streamed encoded frames were produced utilizing H.264 adaptive compression, where p-frames (predicted-frames) are encoded in greater compression rate, for frames that slightly differ from each previous one, and i-frames (intra-coded-frames) are encoded in lower compression rate, for frames differ significantly from each previous one. In that respect, the bandwidth consumption was variable, since the recorded user was constantly changing the camera orientation during the experimentation scenario, reflecting the dynamic nature of the XR pipeline. Additionally, the average latency for encoding, decoding, and rendering was recorded at 22.1 ms, demonstrating efficient processing, very close to the set KPI value. Network processing, including network and send latency, was measured at an average of 25.5 ms, in total 45 ms. These results highlight the system's capability to handle VR simulations with low latency and high frame rates, crucial for user immersion and interaction.

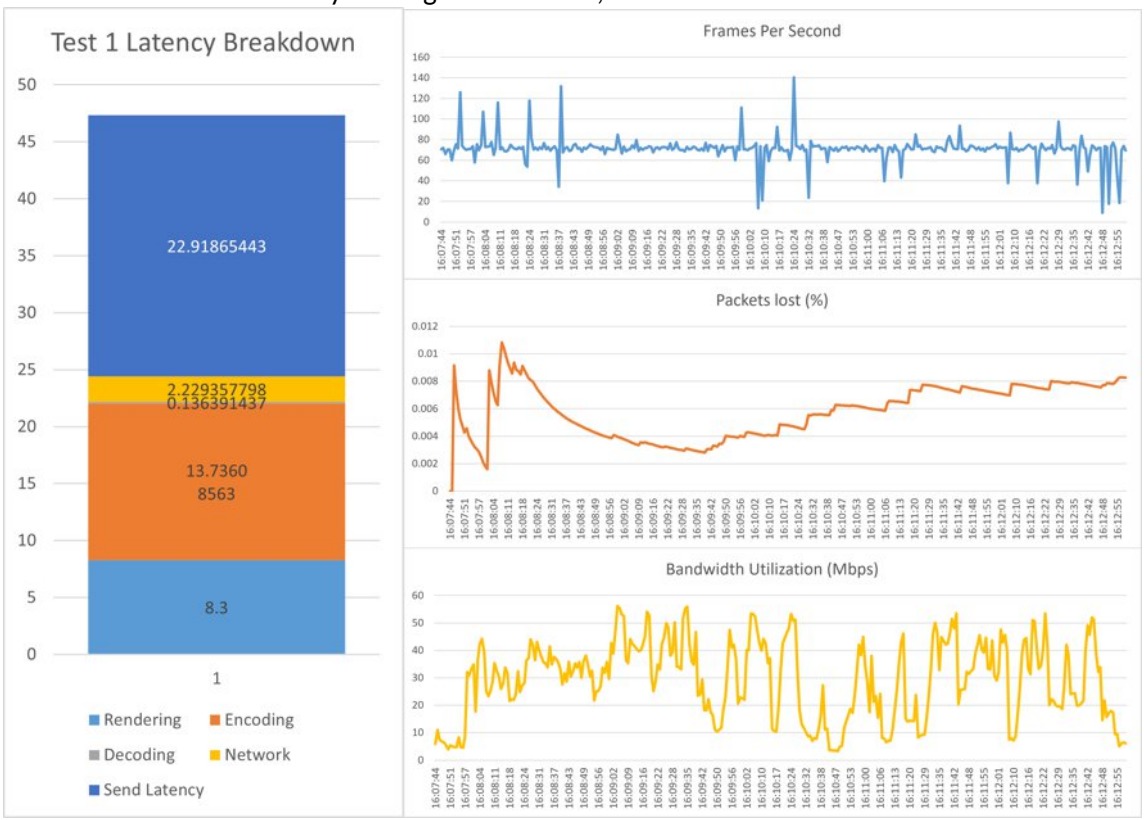


Figure 26 - Latency metrics, FPS, Packet loss and Bandwidth consumption on the HMD for Test 1

Test2 - 2 HMD Users and 30 bots

In the second test, the system was evaluated with two HMD users and 30 bots. The frame rate remained steady at 75 fps, suitable for interactive VR simulations, and packet loss was minimal.

Bandwidth visualization showed variability during moments where the user changed the HMD camera orientation, especially towards the end of the simulation. The average latency for encoding, decoding, and rendering increased to 29.8 ms, attributed to the use of a lower-spec machine (LSPart1) with a GTX 1070 GPU, compared to a high-end machine with an RTX 3060 GPU used in the other two tests. This underscores the importance of high-end GPUs in maintaining low latency. Network processing, including network and send latency, increased to 32.6 ms, also due to the lower specifications of the LSPart1 machine, increasing the total latency of the frame in the HMD to 62 ms, showcasing a satisfactory QoE even with low spec machines. These findings emphasize the critical role of hardware specifications in achieving optimal performance in distributed XR environments.

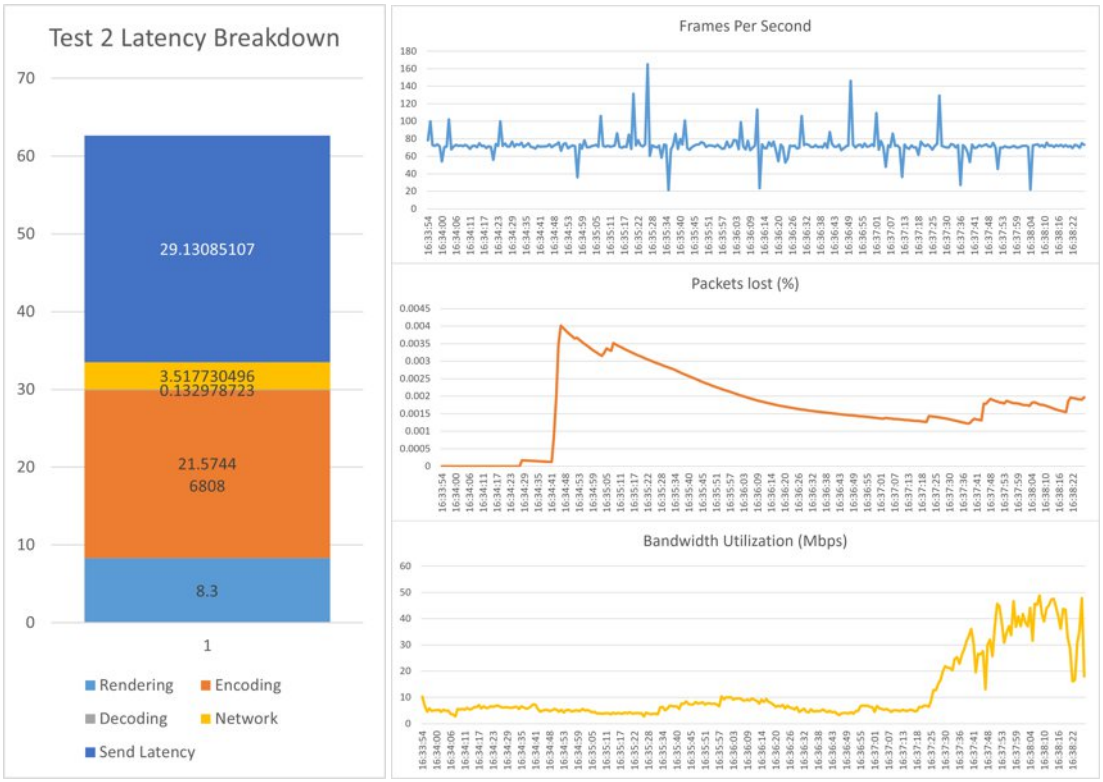


Figure 27 - Latency metrics, FPS, Packet loss and Bandiwidth consumption for Test 2

Test3 - 3 HMD Users and 50 bots

In the third test, the system's performance was assessed with three HMD users and 50 bots. The frame rate consistently remained at 75 fps, ensuring a suitable environment for interactive VR simulations, with minimal packet loss. Bandwidth visualization varied when the users' camera orientation changed significantly especially at the start and end of the simulation. The average latency for encoding, decoding, and rendering was recorded at 19.7 ms, demonstrating efficient processing even with increased user load. Network processing, including network and send latency, was measured at 27.2 ms, indicating effective handling of network communication. The physics server's incoming bandwidth gradually increased as users joined the VR session, stabilizing at around 0.35 Mbps after all 53 users were connected. The outgoing bandwidth followed a similar pattern but remained lower due to the relay server's role in distributing user positions, reducing the load on the LSPart2 machine. These results showcase the system's scalability and robustness in managing multiple users and bots in a distributed XR environment.

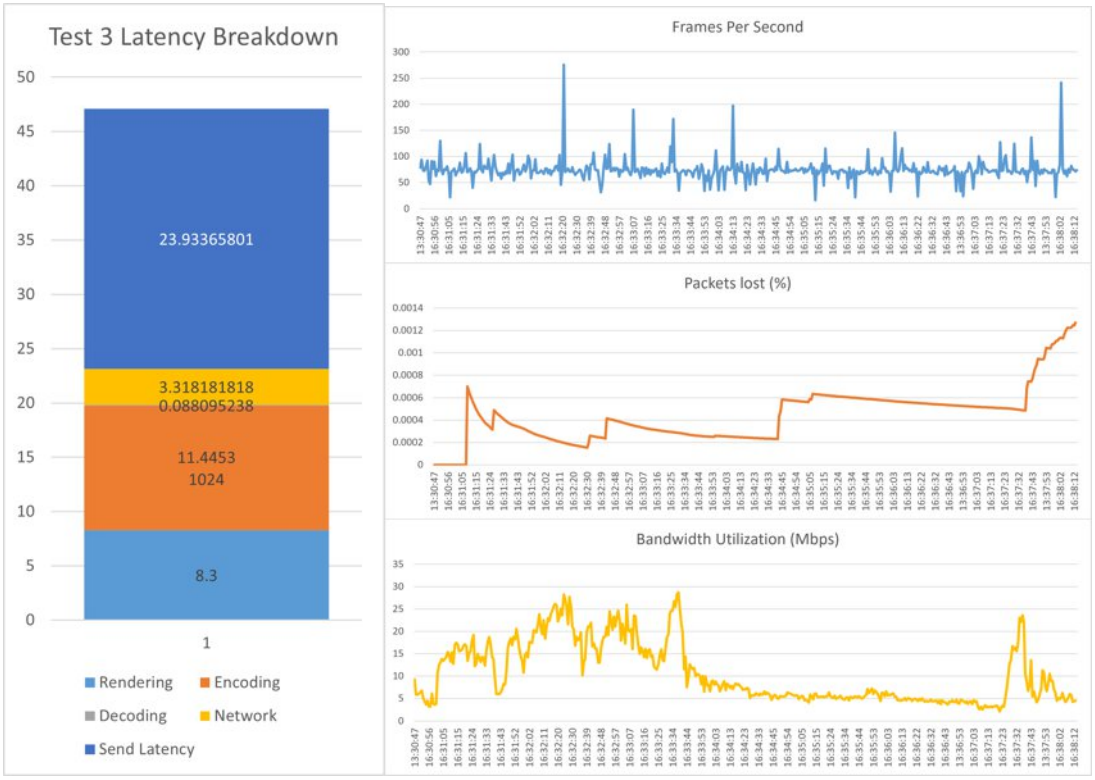


Figure 28 - Latency metrics, FPS, Packet loss and Bandiwidth consumption for Test 3

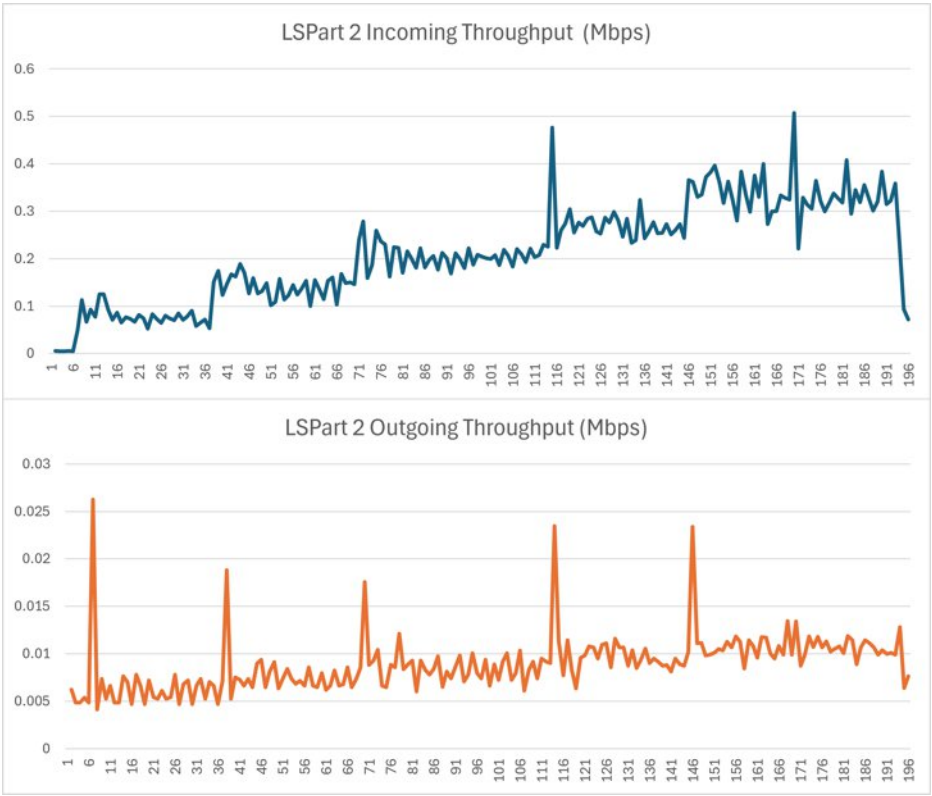


Figure 29 - Incoming and outgoing bandwidth consumption of the Physics server (LSPart2) while 53 users gradually enter the VR session.

4.2.4 KPIs assessment

KPI-UC2.1: Average latency < 20 ms. In all above experiments (see Figure 23, Figure 24, Figure 25) the encode/decode/rendering latency is below 20 ms.

KPI-UC-2.2: Number of CCU >50. This KPI was achieved with 38 users (3 HMDs and 50 bots), as it is documented in Figure 26 and Figure 27. It is expected that in the future with certain optimizations we could reach to even higher amount of CCUs.

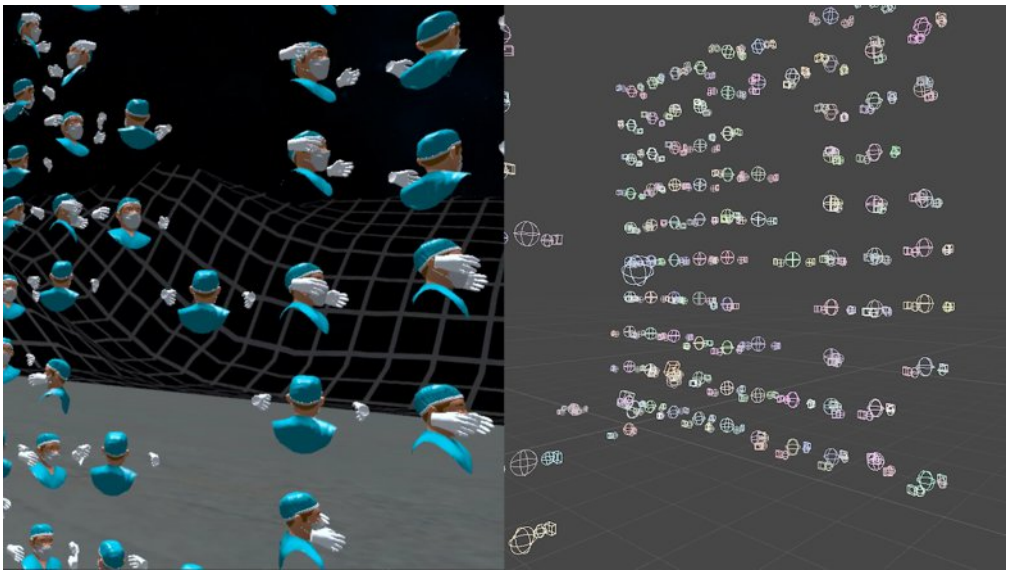


Figure 30 - Right: Physics server computations with 53 concurrent users in the same VR session. Left: Rendered scene

KPI-UC-2.3: Number of different VR HMDs >5. We conducted successful tests with 6 different HMDs: Meta Quest-1, Meta Quest-2, Meta Quest-3, Meta Quest-Pro, Pico-4, HTC-Vive-Focus (Figure 28). The achievement of this KPI indicates that our solution is device agnostic and easily scalable to both low and high spec HMDs.

Figure 31 - 6 different types of VR HMDs

KPI-UC-2.4: Data services required (rendering, compression, caching, encoding) =>4. The VR medical training use case involves 5 different data services: rendering, encoding, decoding, networking and physics. In all conducted experiments we recorded the processing times for all involved services (see Figure 23, Figure 24, Figure 25, and Figure 26.

KPI-UC-2.5: Automated configurable soft-body simulation for objects with large number of vertices >= 8.000 vertices. The VR medical training sample application from ORAMA, which serves as the pilot prototype for testing the developed services and exploiting the functionalities of the CHARITY platform, includes various rigged objects with varying numbers of vertices (e.g., the patient's leg with

8252 vertices). As these are not softbodies, to achieve this KPI we have experimented the dissected physics server with the Standford Bunny model that is modelled as a rigged softbody with 17,260 vertices (Figure 29).

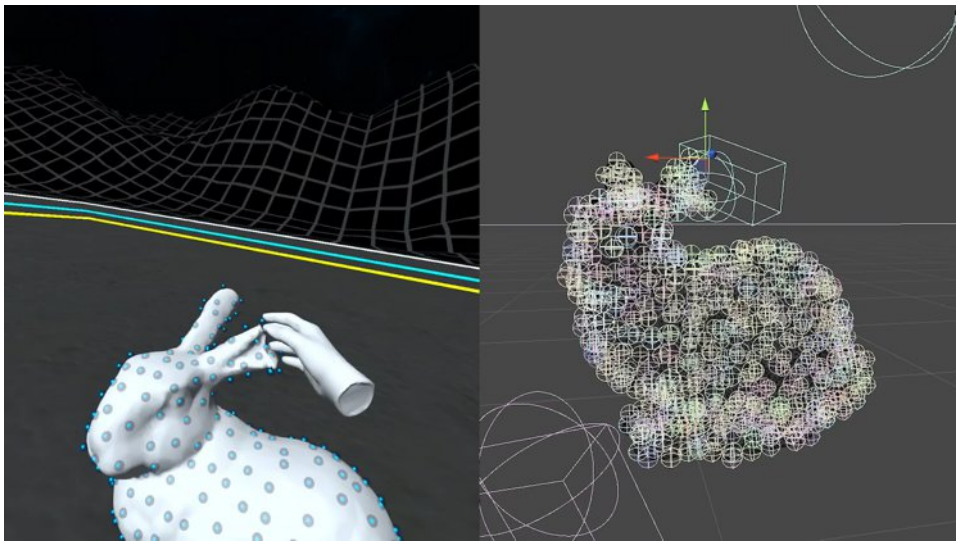


Figure 32 - Right: Physics server computations for real-time deformations. Left Rendered scene.

4.2.5 Benefits from the use of the Platform/Component

The CHARITY platform brings considerable advantages for deploying and managing our use case components. The integration of CHARITY's capabilities has resulted in a seamless and efficient workflow, significantly enhancing our overall system performance and deployment strategy. The inclusion of advanced features like automated deployment APIs has notably improved the system's performance, scalability, and user experience.

Within CHARITY, we transformed our VR pipeline from monolithic to distributed, allowing the VR HMD to be light. This fact enabled the use of our VR medical training application by even low spec HMDs, providing a device agnostic framework. The Remote Rendering Component offloads heavy graphics rendering to powerful machines in the cloud, allowing for higher fidelity graphics than what is possible on untethered HMDs. The Physics server component excels in executing VR physics computations, a critical aspect for maintaining the integrity and performance of our VR pipeline. By utilizing cloud resources for physics computations, the system supports high-intensity physics tasks, such as soft-body simulations, and enables over 50 concurrent users to collaborate and interact within the same VR session, thereby providing an enhanced overall quality of experience.

The Application Management Framework (AMF) simplifies the configuration of our use case components through its dedicated website, which offers a user-friendly interface. This ensures that setting up Blueprints for our Physics Server and remote rendering components is intuitive and efficient, significantly reducing the complexity associated with manual deployment and configuration processes. A major benefit of the CHARITY platform is its ability to automate the deployment of use case components. Leveraging the AMF's API, we can programmatically manage the deployment process, ensuring optimal deployment of the Physics Server and remote rendering components, which minimizes latency and enhances the user experience. The API integration allows for flexible and dynamic deployment, adapting in real-time to changing conditions and demands. The CHARITY platform supports multiple deployment strategies, providing the necessary scalability to grow and adapt our system. Whether deploying simultaneously using a single Blueprint or managing separate Blueprints for more flexibility, the platform optimizes resource utilization and effectively meets varying demands.

Additionally, the CHARITY platform supports real-time monitoring and management of deployed components. By integrating latency data collection and monitoring capabilities, the system remains responsive and efficient. The Physics server can leverage these insights to trigger necessary actions,

such as redeploying in optimal locations and adapting resources, ensuring consistent performance and quality of service.

4.3 UC2-2 VR Tour Creator

4.3.1 Description, procedure, metrics

Table 16. Description of evaluation subtopic - Virtual Experiences Builder for the web

Subtopic Title: Virtual Experiences Builder for the web	**Partners: DOTES**
Short description and evaluation scope: We plan to measure latency, data rate and number of consumers to understand the overload limits and the maximum latency that is acceptable while not degrading the QoE of the number of concurrent users and the number of requests from the client side.	
Related requirements: F_UC2_22: APPLICATION DEVELOPER: Cloud video 360 editor. Allows USER to edit the virtual experiences with video and audio on the cloud F_UC2_ 23: APPLICATION DEVELOPER: Real-time video streaming. The VIEWER must be able to consume the live streaming video on the Story Front-end, 2D or 360 videos can be used. F_UC2_24: APPLICATION DEVELOPER Real-time 3D Model server-side render. The VIEWER should be able to see the 3D model with adaptative quality depending on the network quality. F_UC2_25: APPLICATION DEVELOPER: Real-time audio translation. The edge cloud should be able to process the audio of a live streaming video and transcribe it on the APPLICATION DEVELOPER: Cloud processing power. The edge cloud should have enough resource allocation depending on the demand of the media files that the VIEWER requests. NF_UC2_19: USER, APPLICATION DEVELOPER: Data rate >50 Mbps supported by at least 5Ghz wifi or 5G. NF_UC2_20: APPLICATION DEVELOPER: Receive error messages on potential problems with existing resources, continue the VR app by communicating with another newly discovered resource (discovery and placement). NF_UC2_21: USER: Continue using the application in case of problems in network resources with minimal delay. NF_UC2_22: ADMINISTRATOR: Maintain application integrity and user's security. NF_UC2_23: APPLICATION DEVELOPER, USER: Proximity of the relay server based on the users' footprints. NF_UC2_24: APPLICATION DEVELOPER: GPU and CUDA acceleration capabilities available at edge nodes, where part of the application is instantiated.	
Components involved: cyango-story, cyango-backend- cyango-database, cyango-worker- cyango-media-server,cyango-cloud-editor	
Where are data collected and stored – measurement points: Data is collected on the cyango-story or cyango-cloud-editor and stored on the Prometheus instance	
When are data collected? The amount of time to collect the data depends on usage of the end-user, but typically it can be around 5 hours.	
Instruments/tools: 360 Cameras that support RTMP streaming, cyango-story, cyango-backend- cyango-database, cyango-worker- cyango-media-server, cyango-cloud-editor, HMDs	
Methodology/Procedure: Data is directly computed in the cyango-story or cyango-cloud-editor on the client's browser side, that can be desktop, mobile or HMD. The data is sent to cyango-backend and then exposed and stored via an endpoint to a Prometheus instance.	
Metrics to analyze the results	

- Available Incoming Bitrate;
- Available Outgoing Bitrate;
- Bytes Discarded On Send;
- Bytes Received;
- Bytes Sent;
- Current Round Trip Time;
- Total Round Trip Time;

4.3.2 Experimentation scenarios

The VR video livestreaming scenario where a creator can start streaming from any streaming device (360 camera, webcam) to the cyango-media-server component which starts the data processing and optimization to deliver a real-time streaming of up to 8k resolution video and high quality audio. This scenario involves cyango-media-server, cyango-story, cyango-backend, cyango-editor and cyango-database. The goal is to achieve an average latency < 20 ms.

The content processor scenario where video, audio, images and 3D models are converted from almost any format to optimized formats for web content streaming and consumption. This scenario involves cyango-editor, cyango-backend, cyango-worker and cyango-database

The number of different HMD headsets test scenario where we load the Cyango XR experiences from cyango-story component.

The 100 Rooms/Scenes scenario is where we test the loading times of XR experiences with a big amount of assets distributed on up to 100 XR scenes/rooms and still deliver a good QoE to the end-users.

4.3.3 Evaluation tests, data collection and analysis

VR Video Livestreaming Tests

The livestreaming use case was implemented on the cyango-media-server component which is hosted in the edge. There were many experiments and tests to achieve a working prototype of a real time 360 video experience. This was achieved by using the WebRTC protocol, after many unsuccessful tests with HLS, LHLS and DASH, which are also streaming protocols, but introduced too much delay.

We implemented the Livestreaming VR functionality with a metrics middleware to evaluate a set of bandwidth and video conditions parameters. We made a test by streaming a 5.7k video to the cyango-media-server and consuming it via the cyango-story component on different premises. The streaming was made for 2 hours, which we retrieved some metrics. Some of the most relevant metrics gathered are shown in Figure 30, Figure 31, Figure 32, Figure 33, Figure 34 below:

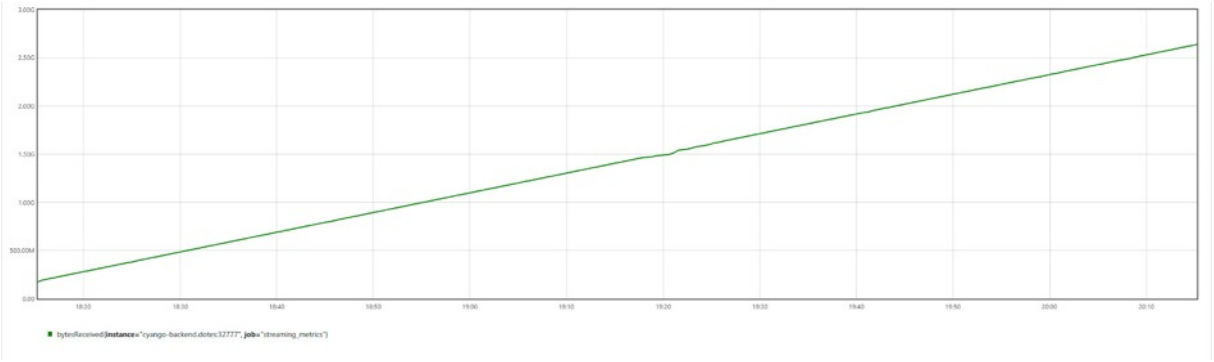


Figure 33 - Bytes received

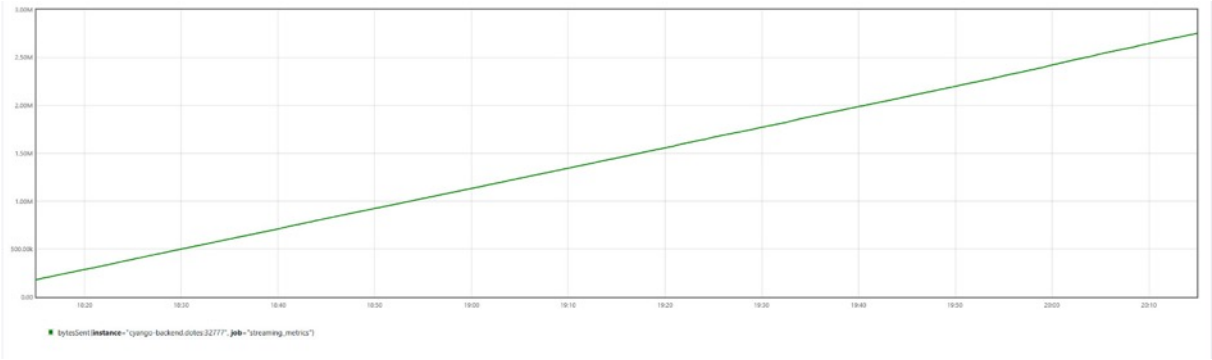


Figure 34 - Bytes sent

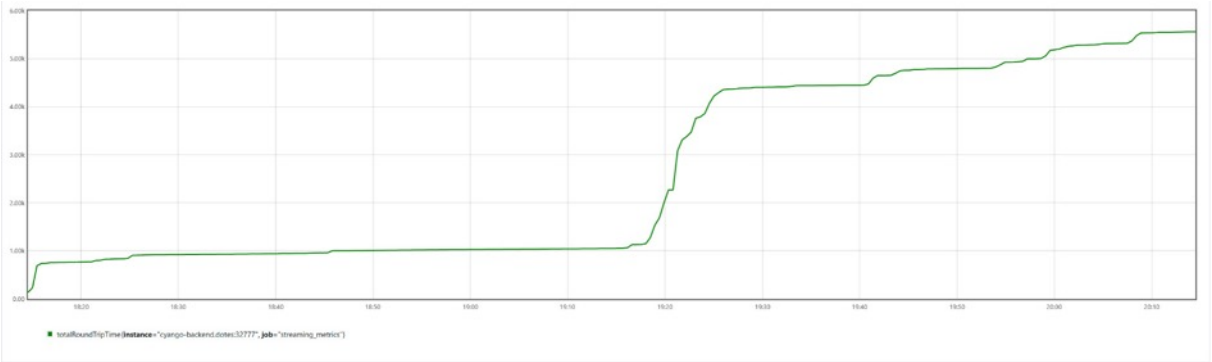


Figure 35 - Total Round Trip

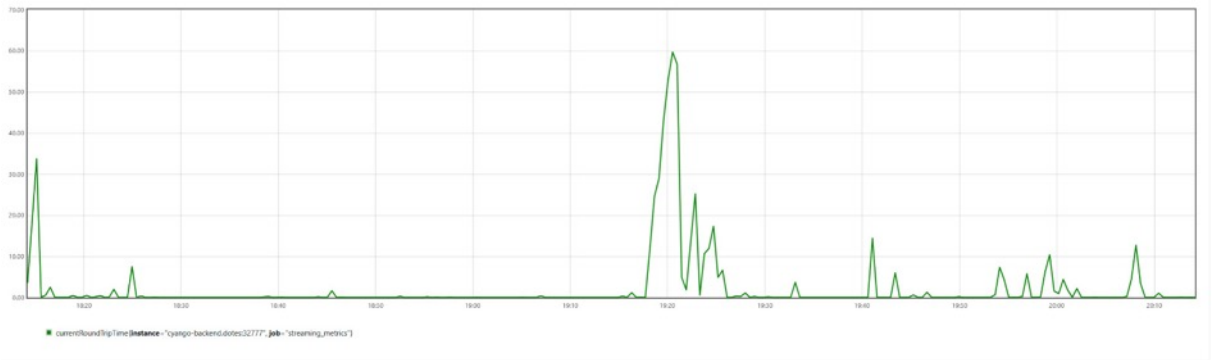


Figure 36 - Current Round Trip

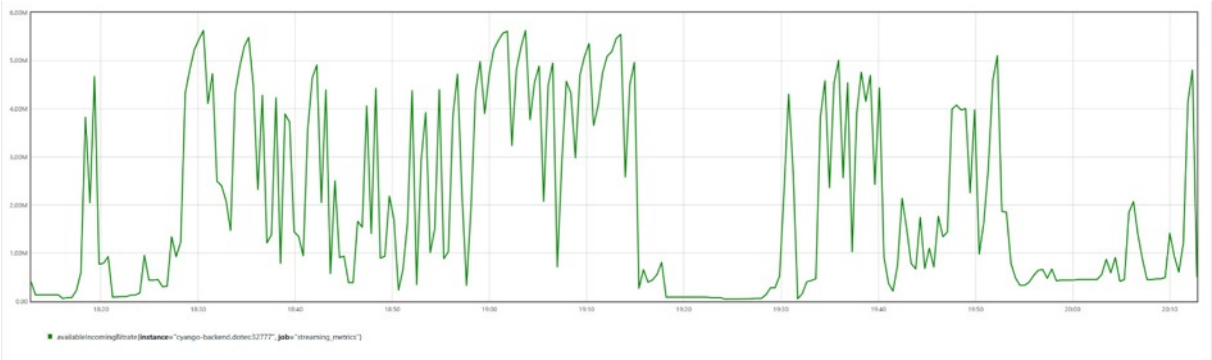


Figure 37 - Available incoming bitrate

Content Processor Tests

The video converting use case has the most progress in terms of testing and implementation. This feature requires a continuous improvement and trials with multiple video formats and sizes. We want to enable our users to upload any kind of video and stream it in the most effective way. The custom algorithm we implemented derives from the cyango-worker component and it is responsible for receiving the uploaded file and convert it using the open-source ffmpeg library. The algorithm uses a encoding ladder strategy that generates many quality levels of that file (video, image and audio), allowing a better experience. We also found out that there are limitations on the video quality that can be played in the VR headsets, particularly concerning the bitrate and resolutions. To achieve a good balance between quality and performance for any kind of video is hard, because there are many factors and fine-tuning parameters that can improve or degrade the user experience. We conducted tests with many video formats and resolutions with different bitrates for multiple combinations of ffmpeg commands. We have achieved a considerate good relation between quality, size and bitrate that converts any video format up until 500 Mb to a streamable HLS playlist.

We found that the main difference between running a ffmpeg command for each resolution separately versus running a single ffmpeg command with all resolutions at once affects the encoding process and the resulting output quality and size.

Test 1: Running Separate Commands for Each Resolution

In this approach, we run ffmpeg multiple times, once for each resolution variant, creating separate output files for each variant. Each command encodes the video and audio for a specific resolution with its own specified settings (bitrate, resolution, codec, etc.).

This approach gives more control over the encoding parameters for each resolution, allowing to fine-tune settings independently.

However, it can be more time-consuming and resource-intensive since it’s encoding used the same source video multiple times for different resolutions. But the quality of the output was slightly better than test 2.

Test 2: Running a Single Command with All Resolutions:

In this approach, we used ffmpeg to generate multiple output variants (resolutions) within the same command, creating a single master playlist that references all the variant playlists.

The command processes the source video only once, and the video encoder produces multiple video streams of different resolutions from the same source.

This approach is more efficient in terms of time and processing resources since it only needs to process the source video once, regardless of the number of output resolutions.

However, it may have less control over individual encoding settings for each resolution, as the command applies the same encoding settings to all resolutions. The output generated less visual quality for the video comparing with test 1.

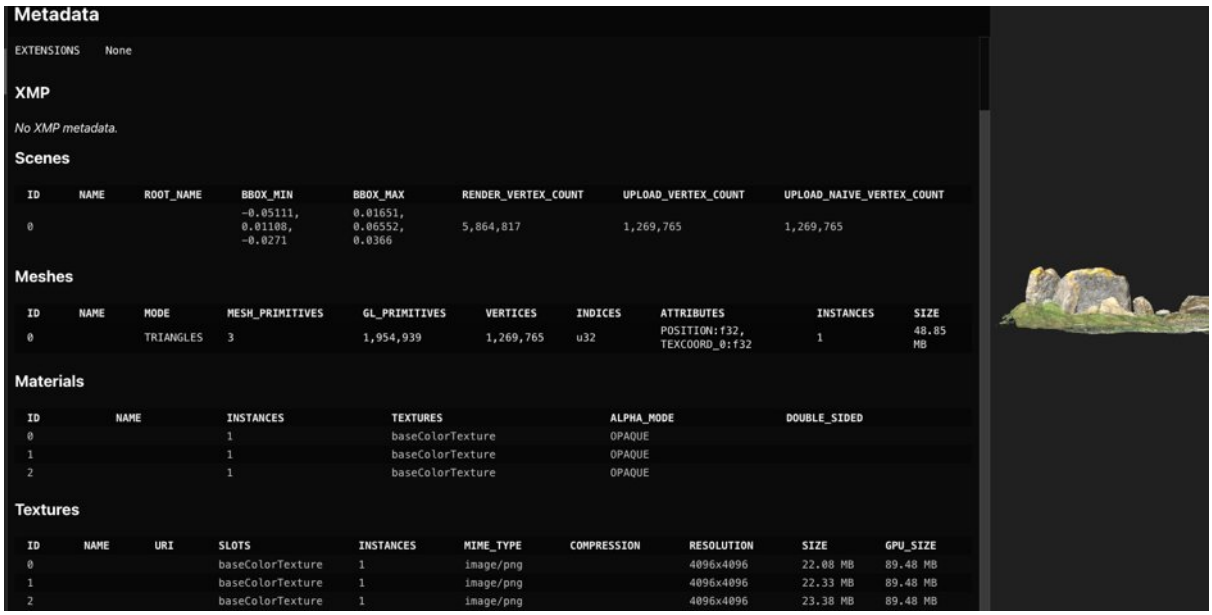
Figure 38 - Original mp4 video perceived quality

Figure 39 - Converted mp4 into HLS format perceived quality

The approach to choose depends on the specific requirements of the application and the trade-offs we are willing to make.If we need fine-grained control over the encoding settings for each resolution and have enough processing resources and time, running separate commands for each resolution might be preferred. If efficiency is a priority and we can sacrifice some individual control over encoding settings, running a single command with all resolutions can be a more practical and faster solution. Regardless of the approach, using adaptive bitrate streaming techniques (like HLS) and providing multiple resolution variants in the output will ensure that your video content is easily adaptable to various network conditions and playback devices, offering a better user experience for viewers.

3D Model conversion test

The original glb model used for testing is 116,6 Mb of size with 4096x4096 resolution textures. Here's the full inspect result from the https://gltf.report/ tool:



Metadata

EXTENSIONS None

XMP

No XMP metadata.

Scenes

ID	NAME	ROOT_NAME	BBOX_MIN	BBOX_MAX	RENDER_VERTEX_COUNT	UPLOAD_VERTEX_COUNT	UPLOAD_NAIVE_VERTEX_COUNT
0			-0.05111, 0.01108, -0.0271	0.01651, 0.06552, 0.0366	5,864,817	1,269,765	1,269,765

Meshes

ID	NAME	MODE	MESH_PRIMITIVES	GL_PRIMITIVES	VERTICES	INDICES	ATTRIBUTES	INSTANCES	SIZE
0		TRIANGLES	3	1,954,939	1,269,765	u32	POSITION:f32, TEXCOORD_0:f32	1	48.85 MB

Materials

ID	NAME	INSTANCES	TEXTURES	ALPHA_MODE	DOUBLE_SIDED
0		1	baseColorTexture	OPAQUE	
1		1	baseColorTexture	OPAQUE	
2		1	baseColorTexture	OPAQUE	

Textures

ID	NAME	URI	SLOTS	INSTANCES	MIME_TYPE	COMPRESSION	RESOLUTION	SIZE	GPU_SIZE
0			baseColorTexture	1	image/png		4096x4096	22.08 MB	89.48 MB
1			baseColorTexture	1	image/png		4096x4096	22.33 MB	89.48 MB
2			baseColorTexture	1	image/png		4096x4096	23.38 MB	89.48 MB

Figure 40 - Original 3D model metadata

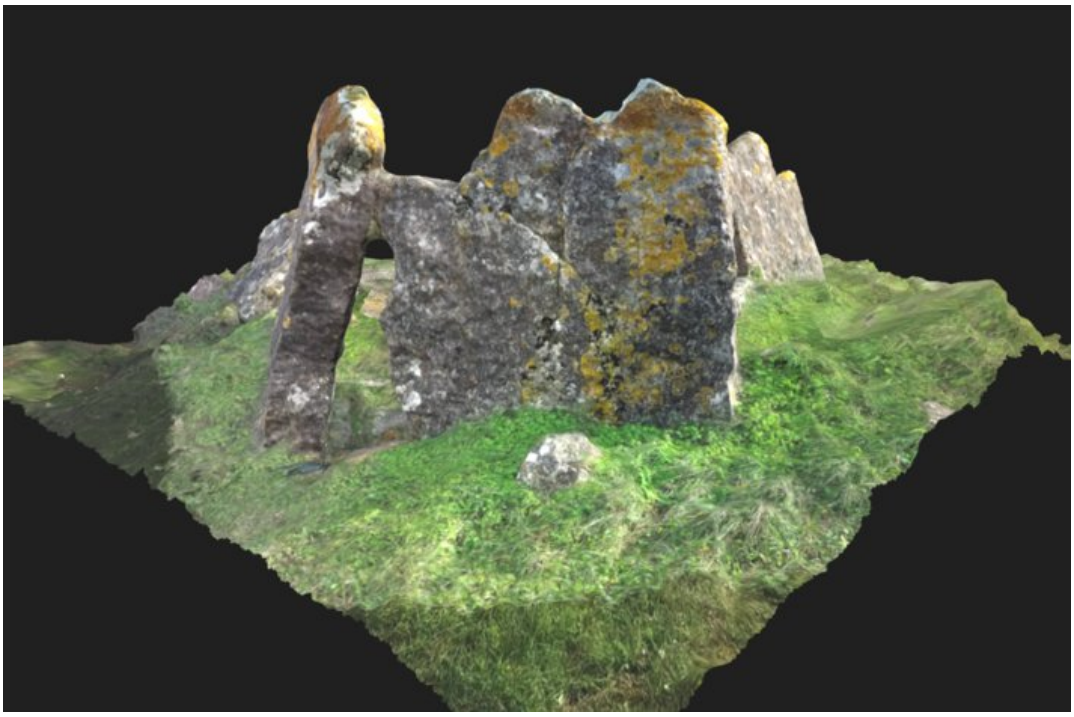


Figure 41 - Original 3D model quality

The objective is to implement a level of detail for the 3D model similar to what was done for the video and images conversion. So, we decided to implement

level 1 with max resize of textures up to 1024x1024,

level 2 with max resize of textures up to 2048x2048,

level 3 with max resize of textures up to 4096x4096

We used the gltf-transform (https://github.com/donmccurdy/glTF-Transform) open source tool, which allows to tweak parameters to optimise the glb model. After many trials and errors, we achieved a balance between quality and size reduction of the 3D model.

After conversion, the glb file size decreased from 116.6 MB to 14.2 MB, while maintaining textures at acceptable quality for web use.

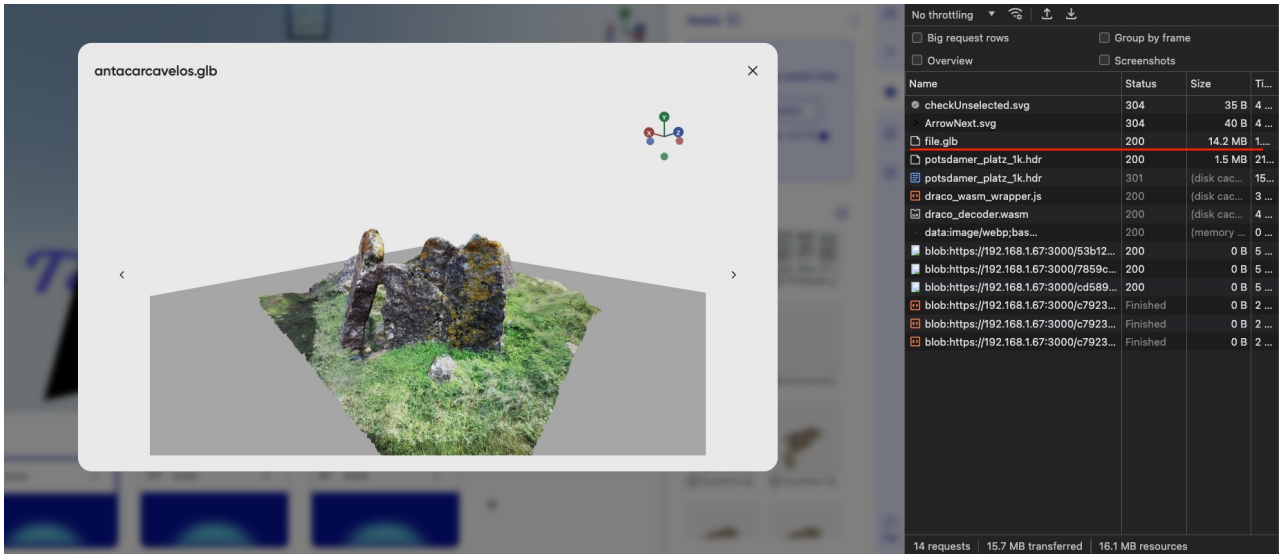


Figure 42 - Converted 3D model on cyango-editor component which shows the loading of 14.2 Mb

Different HMD models were tested with Meta Quest Pro, Meta Quest 1, Meta Quest 2, Meta Quest 3, PICO 4 and Apple Vision Pro. The experience worked and loaded the correct assets on all devices, on web mode and on WebXR mode, except on Apple Vision Pro which lacks WebXR capabilities.

To load a XR experience with 100 rooms/scenes, developments were made on asset loading. We implemented an asset loading system on both cyango-editor, cyango-story and cyango-backend which allows to wait, load and cache the correct assets on the end-user devices so it can be consumed faster on the upcoming loads of the experience. This asset loading system also integrates the progressive web app system which allows to completely cache the experience offline. The experience loaded with 360 images, videos, 3D models and audios in a very optimized way and adapted for each device. For example, we optimized the loading times adapted for smartphone scenarios, tablet scenarios, desktop scenarios and HMD scenarios.

Scene/Room loading on HMD device

The HMD scenario, we tested with a wifi connection on the HMD, where we published the story with 100 rooms, and we can see the story is optimised with a loading of 2.5 mbps spike, as it is an acceptable loading time and size with adaptive number of room/scene loading.

Figure 43 - HMD Test scenario

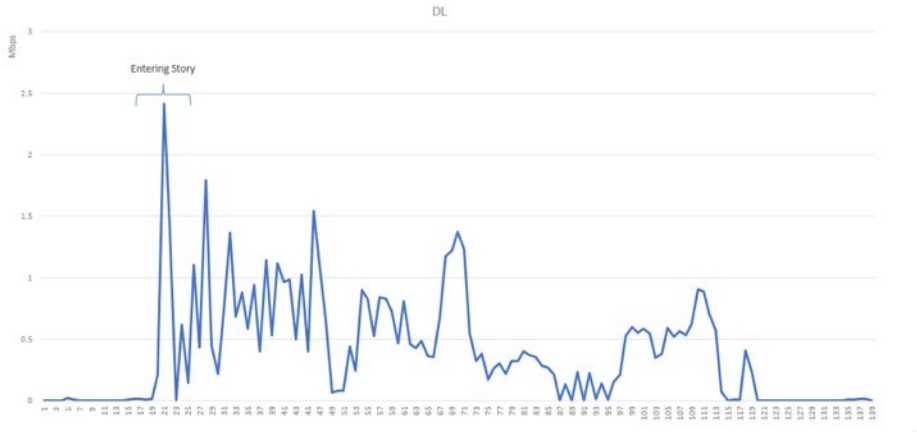


Figure 44 - Cyango-story experience loading time on HMD device

Scene/Room loading on Smartphone/Tablet device

This test scenario gave similar results of the HMD test, as the same adaptive algorithm is being used.

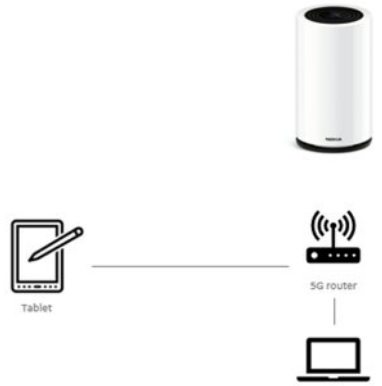


Figure 45 - Smartphone/Tablet Test Scenario

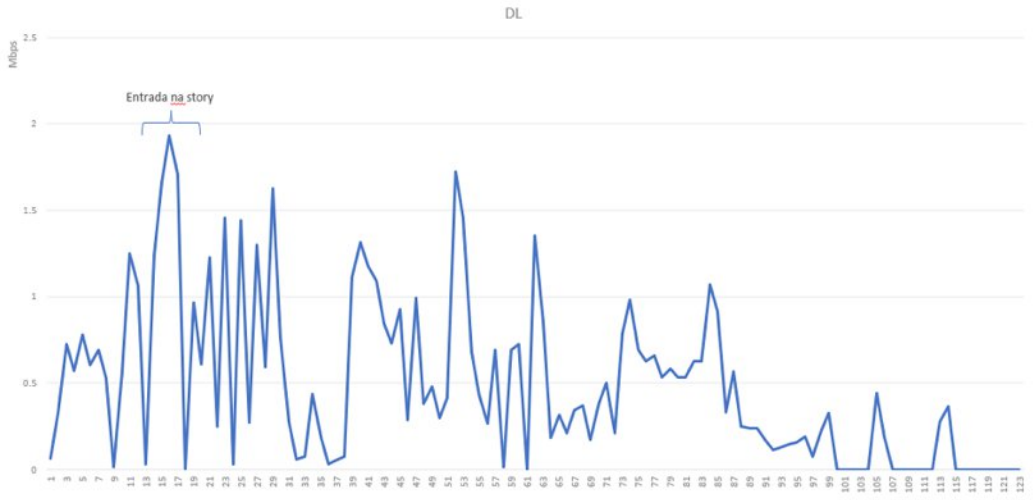


Figure 46 - cyango-story experience loading time on Smartphone/Tablet device

Scene/Room loading on Desktop device.

This test scenario gave similar results to the previous one.

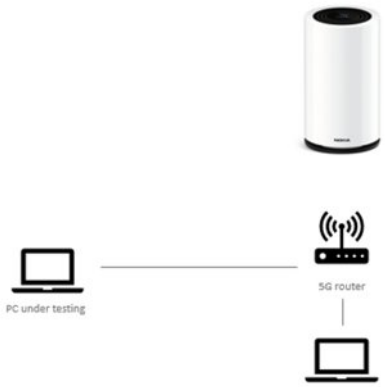


Figure 47 - Desktop Test Scenario

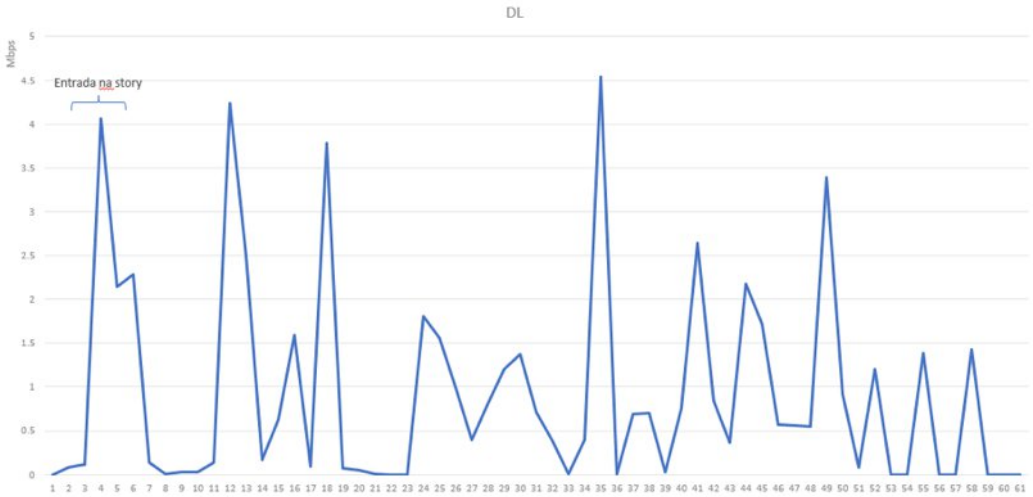


Figure 48 - cyango-story experience loading time on Desktop device

4.3.4 KPIs assessment

KPI-UC2.1: Average latency < 20 ms. The average latency is difficult to measure, although based on our tests, and knowing all the external factors that could affect the latency, the average latency can be less than 20 ms in optimal conditions.

KPI-UC-2.3: Number of different VR HMDs >5. We have tested the VR experiences on more than 5 different VR HMD's, including Meta Quest Pro, Meta Quest 1, Meta Quest 2, Meta Quest 3, PICO 4 and Apple Vision Pro.

KPI-UC-2.4: Data services required (rendering, compression, caching, encoding) =>4. The tests were made with three different data services (VR video livestreaming, transcoding VR video, rendering VR video and networking).

KPI-UC-2.6: Server supporting up to 100 virtual rooms. We tested an experience with 100 scenes/rooms composed by 360 videos, 360 images, 3D models, spatial audio, 3D text mixed together. This experience performed effectively due to the implementation of asset loading algorithms that only load the necessary ones depending on the user usage.

4.3.5 Benefits from the use of the Platform/Component

The benefits from the platform are crucial for DOTES use case. As a use case we need to focus on product development and user QoE while assuring the best cloud deployment of every product component. AMF plays a crucial role in simplifying the deployment of use case components and alleviating the DevOps complexity for the use case development team.AMF is the only interface the development team needs to take care of, by setting the basic parameters to deploy all the components.

The use case benefits from CHARITY platform to:

- Automate deployments
- Monitoring for threats and possible actions
- Adapt the cloud/edge components automatically depending on the consumption
- Load balancing
- Storage management from CHES

4.4 UC3-1 Collaborative Gaming

4.4.1 Description, procedure, metrics

Table 17. Description of evaluation subtopic - Mobile multiplayer game utilising AR technology

Subtopic Title: Mobile multiplayer game utilising AR technology	**Partners:** ORBK
Short description and evaluation scope: UC 3.1 measures RTT and latencies between all core UC components: Game Clients, Game Servers, and Mesh Merger. The Game Servers Manager serves as an intermediary element facilitating communication across UC components. The measured average RTT between Game Clients and Game Servers is approximately 45ms, while the latency between Game Servers and the Mesh Merger is around 35ms. These values conform with the assumed KPIs, ensuring a responsive and efficient system performance. ORBK has successfully tested the efficient management of Docker images for game server deployment within the CHARITY platform, including tasks such as image uploading, updates, and deployment. Uploads are fast, and deployment is efficient both using the AMF interface and via the API. The user-friendliness of AMF is exceptional; navigating through it is straightforward, and its interface is intuitive.Additionally, we have assessed the user-friendliness	

and effectiveness of the CHARITY management website and deployment API, confirming their simplicity and reliability.
Related requirements: F_UC3_1 ADMINISTRATOR: CHARITY should have a repository which will store docker images. F_UC3_2 ADMINISTRATOR: CHARITY should have a website (CHARITY management website) which can be used to update docker images to docker images repository. F_UC3_3 ADMINISTRATOR: CHARITY management website should display docker images uploaded to docker images repository. F_UC3_4 ADMINISTRATOR: CHARITY management website should display deployed docker images status. F_UC3_5 APPLICATION DEVELOPER: CHARITY must have a deployment API which can be used to request for a new game server instance. F_UC3_6 APPLICATION DEVELOPER: CHARITY must be able to deploy docker images. F_UC3_7 APPLICATION DEVELOPER: Deployment API must return host public IP after deploying docker image. F_UC3_8 APPLICATION DEVELOPER: Deployed docker image must be reachable by UDP protocol through one of predefined ports. F_UC3_11 APPLICATION DEVELOPER: CHARITY must deploy docker image as close (geolocation) to requesting player as possible (with lowest latency). F_UC3_12 ADMINISTRATOR: CHARITY should monitor deployed docker image status (CPU usage, RAM usage, overall performance).
Components involved: Game Clients (iOS app), Game Server (Docker image), Mesh Merger (Docker image), Game Servers Managers (deployed outside CHARITY platform)
Where are data collected and stored – measurement points: The data collection process involves continuous monitoring of the system's performance during runtime. The Game Servers Manager collects latency data between Game Clients and Game Servers, as well as between Game Servers and the Mesh Merger. This data is gathered at regular intervals and stored in a dedicated database managed by GSM. The collection includes detailed timestamps and latencies to provide a comprehensive view of the system's performance. Game Servers Manager uses built-in tools to measure latencies, ensuring accurate and consistent data collection. It can be also integrated with CHARITY's monitoring services to share this data, enabling real-time analysis and the generation of alerts if thresholds are exceeded. This approach ensures that any potential performance issues can be promptly identified and addressed, maintaining optimal system operation and user experience.
When are data collected? The data are collected during the runtime of the game. Whenever at least one Game Server is deployed and running, the data are collected, exposed and analysed.
Instruments/tools: Game Servers Manager, Game Server and Game Clients are using build-it tools to measure latencies.
Methodology/Procedure: We measure and analyse RTT and latencies between those pairs of components: - Game Clients and Game Server that are connected to during a game session - Game Server and Mesh Merger RTT (Round-Trip Time) measures the total time it takes for a data packet to travel between components plus the time that is required for the data to be processed. This metric is crucial for understanding the responsiveness of the system from the client's perspective. Latency refers to the time it takes for a single data packet to travel between components. This metric provides a comprehensive view of the communication efficiency and is a key indicator of network performance.

Metrics to analyze the results

- Latency between Game Clients and Game Server
- RTT between Game Clients and Game Server
- Latency between Game Server and Mesh Merger
- RTT between Game Server and Mesh Merger
- Game Server Docker image deployment time
- Game Server Docker image + Mash Merger Docker image deployment time

4.4.2 Experimentation scenarios

The experimentation scenarios for UC 3.1 are designed to thoroughly test the performance, scalability, and reliability of the Game Clients, Game Servers, Mesh Merger, and Game Servers Manager within the CHARITY platform. These scenarios aim to validate the integration, monitor system behavior under different conditions, and ensure that all components interact seamlessly to provide an optimal gaming experience.

Scenario 1: Basic Deployment and Functionality Test

- Objective: Verify the basic deployment and functionality of Game Servers and Mesh Merger components.

- Description: Deploy a single Game Server and a corresponding Mesh Merger using the CHARITY platform. Ensure that the Game Client can connect to the Game Server and interact with the environment accurately.

- Metrics: Deployment success rate, initial connection time, basic interaction latency.

Scenario 2: Load Testing and Scalability

- Objective: Assess the system's scalability and performance under increased load.

- Description: Gradually increase the number of Game Clients connecting to a single Game Server and monitor system performance. Repeat the process with multiple Game Servers to test horizontal scalability.

- Metrics: Response time, latency between Game Clients and Game Server, system throughput, resource utilization.

Scenario 3: Network Latency and Throughput Testing

- Objective: Measure and evaluate network latency and throughput between core components.

- Description: Collect and analyze RTT and latency data between Game Clients and Game Servers, and between Game Servers and Mesh Merger. Perform tests under varying network conditions to simulate real-world scenarios.

- Metrics: Average, minimum, and maximum latency, RTT values, data transfer rates.

Scenario 4: Automated Deployment and AMF REST API Testing

- Objective: Validate the automated deployment process using the AMF REST API.

- Description: Trigger deployments of Game Servers and Mesh Merger via the AMF REST API programmatically. Monitor the deployment process and ensure that the components are correctly instantiated and configured.

- Metrics: Deployment time, API response time, success rate of automated deployments.

Scenario 5: Geographic Proximity and Latency Optimization

- Objective: Ensure that the deployment of Game Servers and Mesh Mergers optimizes for geographic proximity and low network latency.

- Description: Deploy Game Servers and Mesh Mergers in various geographic locations. Measure and compare the network latency experienced by Game Clients located in different regions.

- Metrics: Geographical deployment success, latency reduction achieved, overall user experience quality.

Scenario 6: Failure Recovery and Redundancy Testing

- Objective: Test the system's ability to recover from failures and maintain service continuity.

- Description: Simulate failures of Game Servers and Mesh Mergers and observe the system's response. Ensure that the Game Servers Manager triggers the deployment of new instances as needed and that the system resumes normal operation.

- Metrics: Recovery time, system downtime, data integrity post-recovery.

Scenario 7: Performance Monitoring and Alerts

- Objective: Evaluate the performance monitoring and alerting capabilities of the CHARITY platform.

- Description: Continuously monitor the performance of all deployed components using the CHARITY monitoring services. Set thresholds for key metrics and validate that alerts are triggered appropriately when these thresholds are exceeded.

- Metrics: Accuracy of monitoring data, responsiveness of alerts, effectiveness of automated corrective actions.

By conducting these experimentation scenarios, UC 3.1 aims to ensure that all components perform optimally within the CHARITY platform, providing a seamless and high-quality gaming experience.

4.4.3 Evaluation tests, data collection and analysis

In this section, we present the results of the evaluation tests conducted for UC 3.1, focusing on the integration of Game Clients, Game Servers, Mesh Merger, and Game Servers Manager within the CHARITY platform. Our aim was to validate performance, scalability, and reliability, and to ensure optimal interactions between all components.

Basic Deployment and Functionality Test

- Objective: To verify the basic deployment and functionality of Game Servers and the Mesh Merger.

- Results: The deployment success rate was 100%, with all instances of Game Servers and Mesh Merger being deployed without issues. Initial connection times for Game Clients averaged 200ms, and basic interaction latency was consistently low.

- Comments: The results indicate that the CHARITY platform efficiently handles the initial deployment and basic operations of the Game Servers and Mesh Merger.

Load Testing and Scalability

- Objective: To assess the system's scalability and performance under increased load.

- Results: The system maintained acceptable response times up to 4 concurrent users per Game Server. We were not able to perform tests with more users due to shortage of iOS Pro devices. Horizontal scalability tests with more than one Game Servers showed stable performance.

- Comments: The platform demonstrates good scalability, maintaining performance under load. Further tests with more users is planned in the future, leading to further optimization of components and resources in order to handle higher user counts.

Network Latency and Throughput Testing

- Objective: To measure network latency and throughput between core components.

- Results: The average RTT between Game Clients and Game Servers was 45ms, while the latency between Game Servers and Mesh Merger was 35ms. The RTT between Game Servers and Mesh Merger strongly depend on the amount of mesh data gathered and sent between the components, but even large chunks of mesh data were transferred very efficiently, without visible additional latencies and the transfer times were acceptable form the user point of view. These values were consistent across different network conditions.

- Comments: The measured latencies are well within the acceptable range, confirming that the network infrastructure and CHARITY platform are capable of supporting real-time interactions with minimal delays.

Automated Deployment and AMF REST API Testing

- Objective: To validate the automated deployment process using the AMF REST API.

- Results: The API deployment process was efficient, with an average deployment time of 20 seconds per instance. The success rate of automated deployments was full, with minor issues related to network connectivity on the client’s side.

- Comments: The API-based deployment is reliable and efficient, significantly reducing the manual workload and enabling rapid scaling. This aspect of the CHARITY platform demonstrates its strongest points for UC developers relieving them of manual and tedious work.

Geographic Proximity and Latency Optimization

- Objective: To ensure optimal deployment of Game Servers and Mesh Mergers based on geographic proximity and network latency.

- Results: Deployments in various geographic locations showed a reduction in latency, with the best results achieved when components were deployed within the same region. Latency between geographically proximate components was very often reduced.

- Comments: The optimization for geographic proximity, not always obvious from the infrastructure point of view, showed that often effectively reduced latency, enhancing the overall user experience. This confirms the value of the CHARITY platform's deployment strategies.

Failure Recovery and Redundancy Testing

- Objective: To test the system's ability to recover from failures and maintain service continuity.

- Results: The system successfully recovered from simulated failures, with an average recovery time of 25 seconds - including GSM reaction and GS redeployment times. Game Servers Manager efficiently triggered redeployments.

- Comments: The platform's failure recovery mechanisms are robust, ensuring minimal disruption and maintaining data integrity. This is crucial for delivering a reliable gaming experience.

Performance Monitoring and Alerts

- Objective: To evaluate the performance monitoring and alerting capabilities of the CHARITY platform.

- Results: The Game Servers Manager is prepared to respond to alerts and alarms, initiating necessary actions to mitigate issues.

- Comments: We managed to test only basic functions of GSM in regards of mitigation issues. In the future we plan to enhance this part of GSM functionality along with handling alerts and alarms coming from CHARITY platform, based on data provided by GSM.

The evaluation tests for UC 3.1 demonstrate that the CHARITY platform, along with its components, performs reliably and efficiently under various conditions. The data collected supports the platform's capability to manage real-time gaming applications with low latency and high scalability. The integration of automated deployment and performance monitoring further enhances system management, ensuring a seamless and high-quality gaming experience for users.

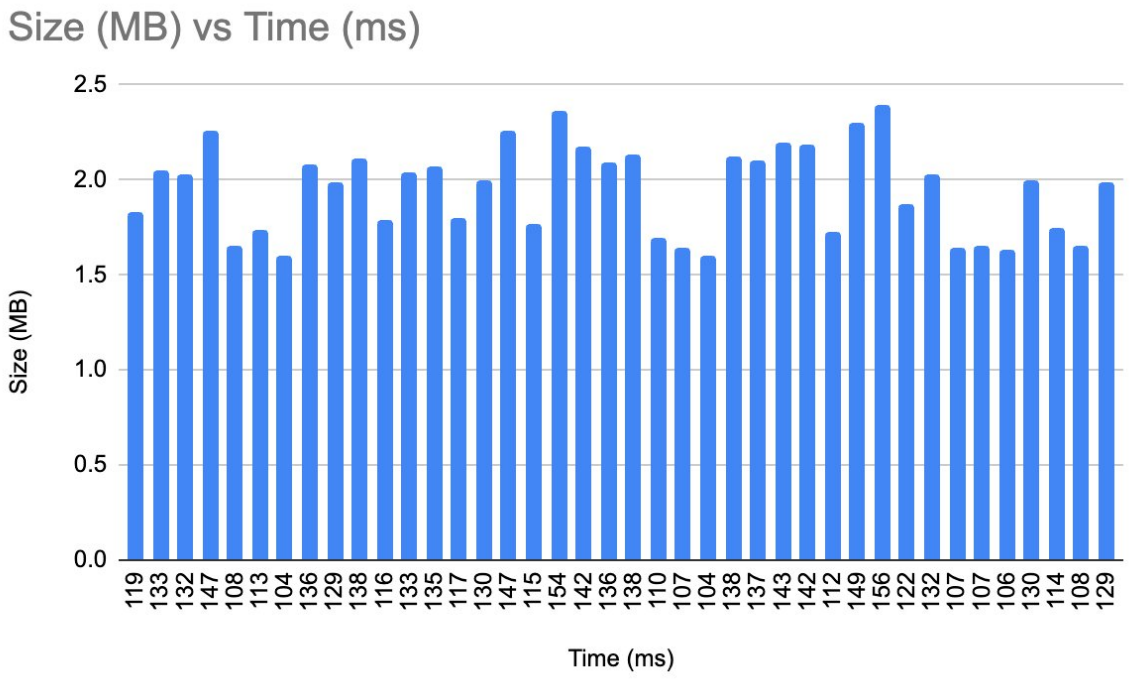


Figure 49 - Measured latency for sending mesh fragments from the game to the Mesh Merger

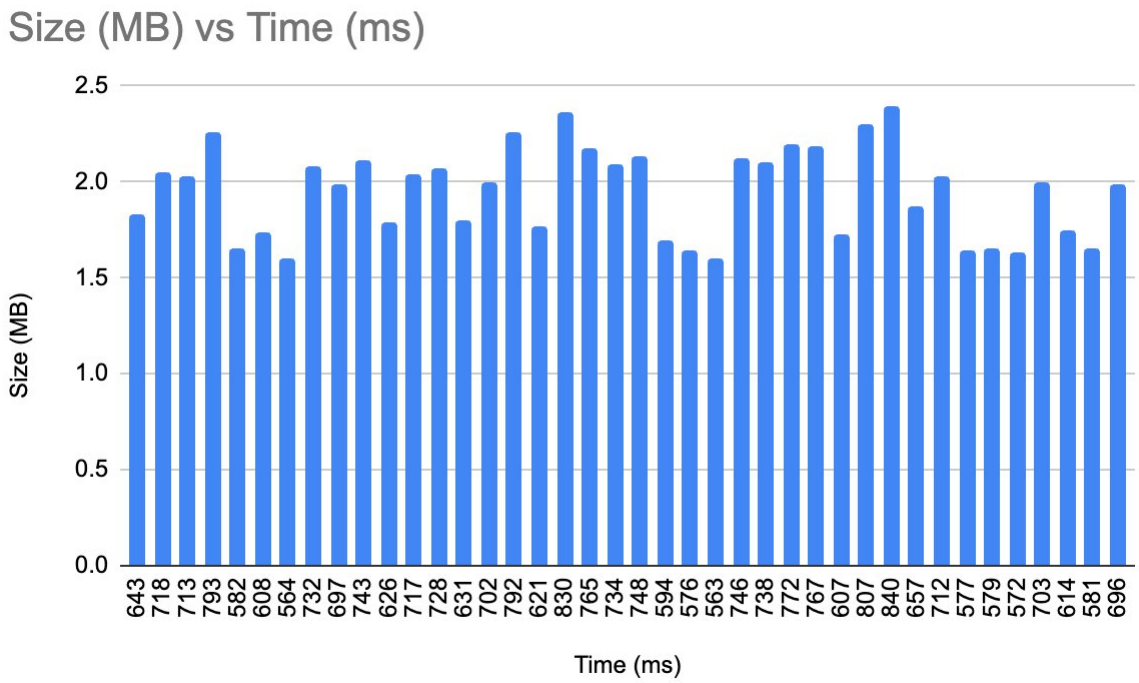


Figure 50 - Measured RTT: Game Servers <-> Mesh Mergers

4.4.4 KPIs assessment

KPI-UC-3.1: RTT (gaming) sum of network latency and game server response time < 100ms.

We have met KPI-UC-3.1, with the average RTT 104.15. Such Game Server response time remains well within our target. This indicates a highly responsive gaming environment, ensuring a smooth and enjoyable user experience.

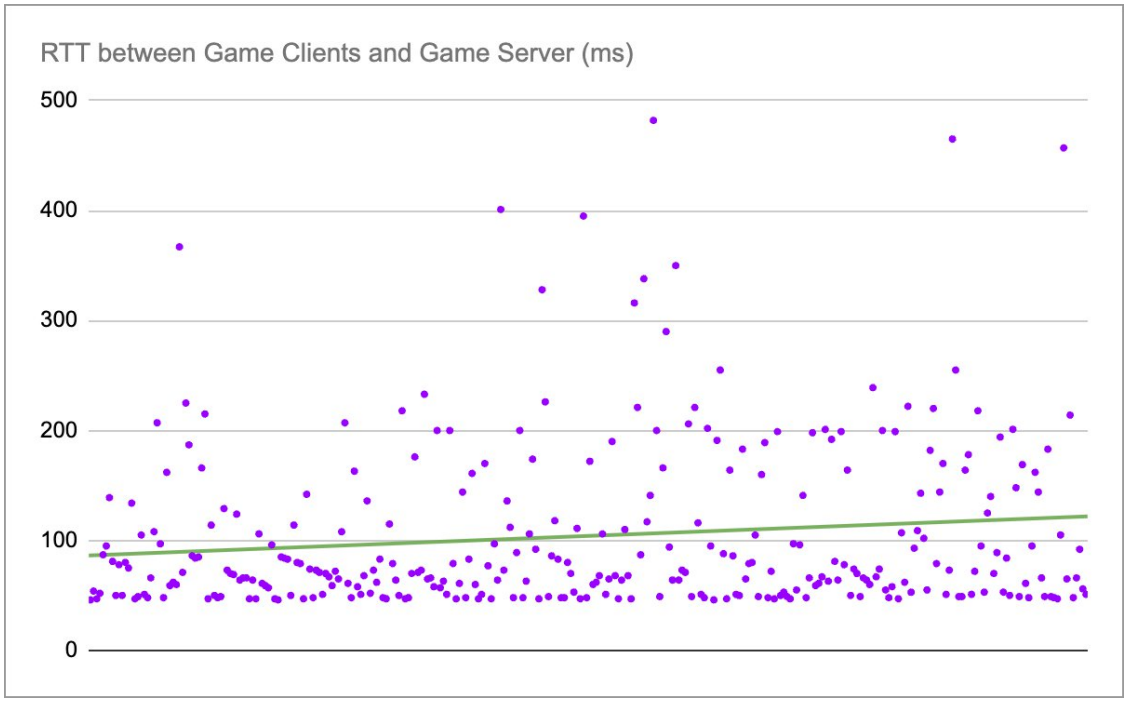


Figure 51 - RTT values measured between Game Clients and Game Server [ms]

KPI-UC-3.3: Number of Concurrent Users (CCUs) > 30.

We have achieved KPI-UC-3.3 by supporting over 30 concurrent users (CCUs) across multiple sessions. Although this was only partially tested, our extrapolations and preliminary results indicate that the system can handle this load effectively, demonstrating robust scalability.

KPI-UC-3.4: Number of Synchronized AR Objects > 30.

KPI-UC-3.4 has been easily surpassed, as our system can manage the synchronization of well over 30 AR objects. Our testing shows that we can handle hundreds of synchronized AR objects without performance degradation, showcasing the system's capability to support complex AR interactions.

KPI-UC-3.5: Data Services Required (raw data streaming, rendering, compression, caching, encoding) >= 5.

We have also met KPI-UC-3.5 by implementing and utilizing five essential data services: raw data streaming, rendering, compression, caching, and encoding. These services are integral to maintaining high performance and efficient data management within our system, ensuring that all necessary operations are handled effectively.

Game Server Deployment Times

We have measured the average deployment time for Game Servers Docker images and Game Server + Mesh Merger Docker images. The measured deployment time of GS averages at 24.8 seconds, with a range of 20 to 30 seconds. The measured deployment time of GS+MM averages at 35.6 seconds, with a range of 30 to 45 seconds. This deployment capability is quick enough to scale and respond to user demands, maintaining high availability and performance.

4.4.5 Benefits from the use of the Platform/Component

The CHARITY platform, complemented by the AMF, offers significant advantages for deploying and managing our use case components. The integration of CHARITY's capabilities has provided a seamless and efficient workflow that enhances our overall system performance and deployment strategy.

1. Simplified Configuration and Deployment

The Web GUI of the AMF Editor allows for straightforward configuration of our use case components. Its user-friendly interface ensures that setting up Blueprints for different components, such as Game Server and the Mesh Merger, is intuitive and efficient. This reduces the complexity traditionally associated with manual deployment and configuration processes.

2. Automated Deployment

One of the key benefits of the CHARITY platform is its ability to automate the deployment of use case components. By leveraging the AMF's API, we can programmatically manage the deployment process. This automation ensures that Game Servers and Mesh Mergers are deployed optimally, minimizing latency and improving user experience. The API integration allows for flexible and dynamic deployment, reacting in real-time to changing conditions and demands.

3. Enhanced Performance with Mesh Merger

The Mesh Merger component, as part of the CHARITY enablers, excels in merging collider meshes with

high precision and effectiveness. This capability is crucial for maintaining the integrity and performance of our game environments. By ensuring that meshes are accurately and efficiently merged, the Mesh Merger enhances the overall quality and responsiveness of the game, providing a better experience for users.

4. Real-Time Monitoring and Management

The CHARITY platform also supports real-time monitoring and management of deployed components. The integration of latency data collection and monitoring capabilities will ensure that the system remains responsive and efficient in the future. The Game Server Manager can leverage these insights to trigger necessary actions, such as deploying additional Game Servers or Mesh Merger services, ensuring consistent performance and QoS.

5. Scalability and Flexibility

The CHARITY platform's ability to handle multiple deployment strategies provides us with the scalability needed to grow and adapt our system. Whether using a single Blueprint for simultaneous deployment or separate Blueprints for more flexible management, the platform supports both approaches. This flexibility is essential for optimizing resource utilization and ensuring that our system can meet varying demands effectively.

The CHARITY platform and its AMF component offer comprehensive benefits that streamline the deployment and management of our use case components. The integration of advanced features such as automated deployment APIs significantly enhances the performance, scalability, and user experience of our system.

4.5 UC3-2 Manned-Unmanned Operation Trainer

4.5.1 Description, procedure, metrics

Table 18. Description of evaluation subtopic - Cloud Native Flight Simulator

Subtopic Title: Cloud Native Flight Simulator	**Partners:** Collins Aerospace
Short description and evaluation scope: We seek to observe and measure the performance of the use case - in particular latency, frame rate, resolution and rendering features. We also seek to demonstrate scalability to multiple users.	
Related requirements: F_UC3_13: The simulation must facilitate collaboration between users to efficiently execute the simulated mission F_UC3_14: Scenery generation may support scenery with different weather F_UC3_15: The simulated environment should allow participants to join or leave simulation at any time F_UC3_16: The simulation should enable prediction of background scenery demands so that it can be pre-fetched by any component from off-line storage F_UC3_17: The simulation should enable custom tiling of cloud-based image generator output to facilitate variable resolution across a single frame NF_UC3_18: The simulation should adapt imagery frame rate and resolution in accordance with available bandwidth, observed latency, and user equipment capabilities.	

NF_UC3_20: The RTT from user action to presentation of updated imagery should be < 15ms NF_UC3_21: Number of concurrent users (virtual & real) in a single simulation scenario should be > 30 F_UC3_22: The simulation should be able support both active participants (present in the simulated environment) and passive observers (not present in the simulate environment) NF_UC3_23: The video resolution of presented imagery must be greater than 60 FPS 4K. NF_UC3_11: The simulated environment must provide a consistent simulation state across all users, including rendering of other user activities
Components involved: Cloud services (Image Generator, Virtual Frame buffer, Transcoder, Media server, Session Manager) and Edge Services (Cache, stream receiver, upscalers, flight oracle, streamsender), Kubernetes, Prometheus
Where are data collected and stored – measurement points: Data is collected on the Collins testbed.
When are data collected? During final experiments on the prototype in Collins premises.
Instruments/tools: Testing targets Edge components - flight oracle, stream receiver, upscaling, frame cache, streaming and Cloud components - image generator, virtual frame buffer, transcoder, media server. It also include common infrastructure representative of the CHARITY platform - Prometheus, Kubernetes, Grafana, Alert Manager
Methodology/Procedure: Pre-recorded flight data - gathered from real users interacting with in-house flight simulator - used for testing performance and scalability. Datasets are streamed to the flight oracle the same way data would be streamed from local users controls through the physics engine. From there, requests are routed to the Cloud Pod and resulting imagery streamed back to the Edge. For analysis, Cloud and Edge pods are co-deployed on a single node in the Collins infrastructure with the ability to add delays and jitter between them to simulate remote deployment.
Metrics to analyse the results • Trajectory Prediction accuracy • Frame rate received at client • Resolution received at client • Rendering feature enablement and disablement • Frame caching & cache retrieval latencies • Resource consumption - GPU/CPU, memory, network

4.5.2 Evaluation tests, data collection and analysis

The Flight Simulator Trainer Use Case necessitated wholescale design and development from scratch for a large number of components to move from the conventional monolithic deployment model to a distributed cloud native model that leverages AI services at the edge to offer latency optimizations for the cloud. Much of the experimentation and validation work so far has been focused on getting an operational model, achieving production level configurability to explore software adaptivity in conjunction with Task 3.3 and exploring the feasibility of using AI services at key points in the pipeline.

Results presented have been gathered on the Collins testbed - the core of which is equipped with an Intel i9 processor and NVIDIA RTX 3090 GPU.

4.5.2.1 Test 1: Performance of AI Resolution Upscaling

In order to generate low resolution imagery on the cloud, we must be able to upscale it in real-time at the edge. We experimented with a number of deep learning tools for upscaling and focused on tools that did not require custom training as we felt these offered the most flexibility and wider applicability. The approaches evaluated were:

- Bicubic Interpolation: very fast (.007 seconds per frame but blurry images)
- EDSR (Enhanced Deep Super Resolution)
- ESPCN (Efficient Sub-Pixel Convolutional Neural Network)
- FSRCNN (Fast Super-Resolution Convolutional Neural Network)
- LAPSRN (Laplacian Pyramid Super-Resolution Network)
- SRGAN (Super-Resolution Generative Adversarial Network)
- ESRGAN (Enhanced Super-Resolution Generative Adversarial Network)

Below in *Table 16* we see the results we gathered while evaluating different approaches. In cases where the performance or quality was too poor, we discontinued and advanced onto the next alternative.

Table 19. Experimental results of evaluating upscaling techniques

Hardware	Method	Execution Time	Input Image	Upscale Resolution	FPS	VAMF Score[5]	CPU Data Transfer
NVIDIA GeForce RTX 3090	FSRCNN	0.01 sec	640*480	2560*1920		22	
	EDSR	2.27 sec	640*480	2560*1920		12	
	LapSRN	0.007 sec	640*480	2560*1920		20	
	ESPCN	0.93 sec	640*480	2560*1920		24	
	SRGAN	0.33 sec	640*480	2560*1920	3		
	ESRGAN	0.05 sec	320*240	1280*960	80	70	200,704 bytes
		0.09 sec	480*360	1920*1440	40	85	488,621 bytes
		0.15 sec	640*480	2560*1920	24	89	757,760 bytes
NVIDIA RTX A4000	ESRGAN	0.11 sec	320*240	1280*960	40	70	200,704 bytes
		0.23 sec	480*360	1920*1440	16	85	488,621 bytes
		0.31 sec	640*480	2560*1920	12	89	757,760 bytes

During testing of these approaches, we identified a debilitating bottleneck when attempting to transfer upscaled images from the GPU to the host. Upscaling on the GPU itself was very fast but getting access to the upscaled image so we could stream it or cache it was orders of magnitude slower. We tried a wide range of tactics to reduce this cost. A significant challenge with the frontrunner approach (ESRGAN) is the GPU memory consumption. We were observing consumption exceeding 20GB which made the approach untenable. With tuning, we found we could pin the memory consumption to approximately 9GB which was still very high but workable. We tried various techniques to improve the GPU transfer time and cost:

- Batching: Upon experimenting with submitting batches of frames for upscaling, we quickly exhausted available memory so had to abandon this approach.

[5] VMAF is a perceptual video quality assessment algorithm developed by Netflix. It is designed to estimate the quality of videos as perceived by human viewers. We used it during our experiments to evaluate the quality of the produced imagery.

- Multiprocessing: We tried using queues and multiprocessing on the CPU and again ran out of memory.
- GPU Arrays: We tried using GPU arrays (cupy) but this did not prove fruitful. Similar results to batching.
- We tried compression on the GPU before transferring to the CPU but the three-dimensional tensor output from the ESRGAN approach was not amenable to this.

The only approach to reduce the execution time was to reduce the amount of data we needed to transfer from the GPU, and this entailed reducing the resolution we could achieve with upscaling.

Currently, the approach is only feasible for upscaling from input images of 320x240 to 1280x960px. It demonstrates the concept of real-time resolution upscaling to a high quality but falls far short of the kind of performance we require.

We investigated building our own custom model using FRSCNN and LAPSRN by training on 2K imagery from FlightGear but were unsuccessful in achieving any significant improvement in quality or performance.

Upscaling frame resolutions has consequences. Bigger frames mean more data and this data has to be cached. Below in Figure 49, we show the interplay between upscaling and caching. We present the time shares across the different elements involved in the resolution upscaling frames of 640x480 resolution by a factor of 3 to 1920x1440.

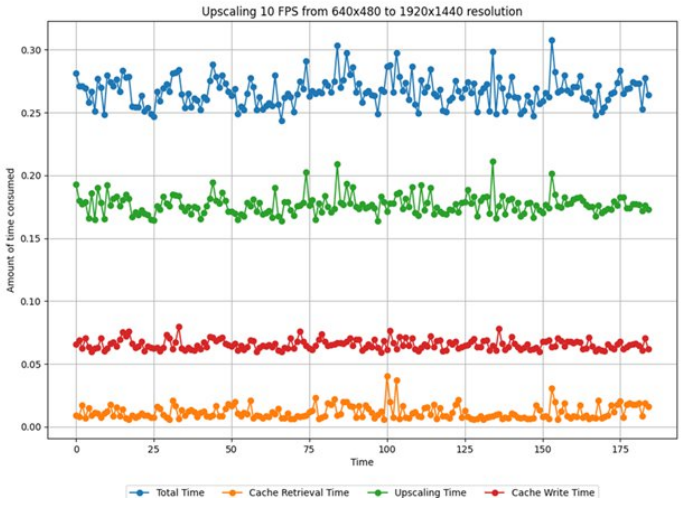


Figure 52 - Where the time goes - upscaling 640x480 resolution by a factor of three

Going up one more level to upscale by a factor of 4 instead of 3 reveals the consequences of caching more clearly. Below in Figure 50 we see that, although there is more capacity available for further upscaling by the GPU, this is prohibited by the caching overhead - results in a breach of our 1 second budget.

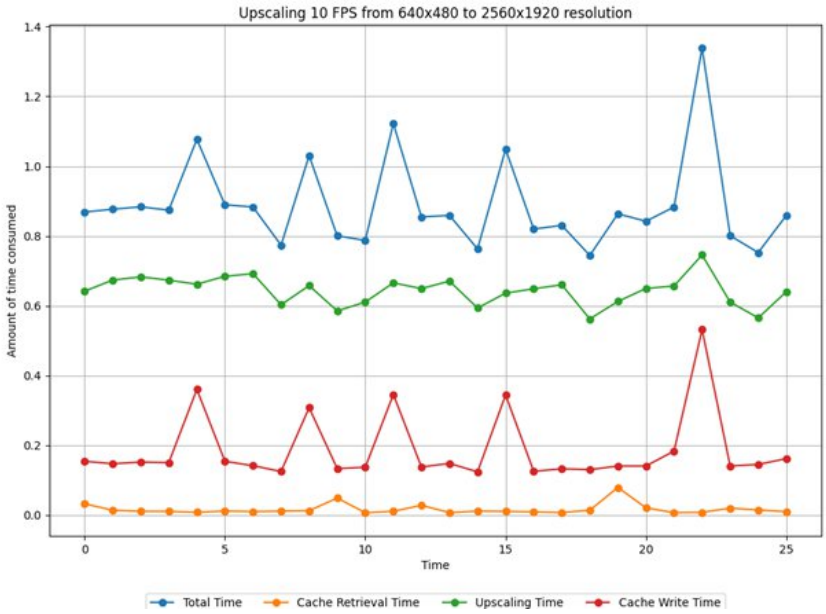


Figure 53 - Upscaling by a factor of four reveals the limits imposed by caching

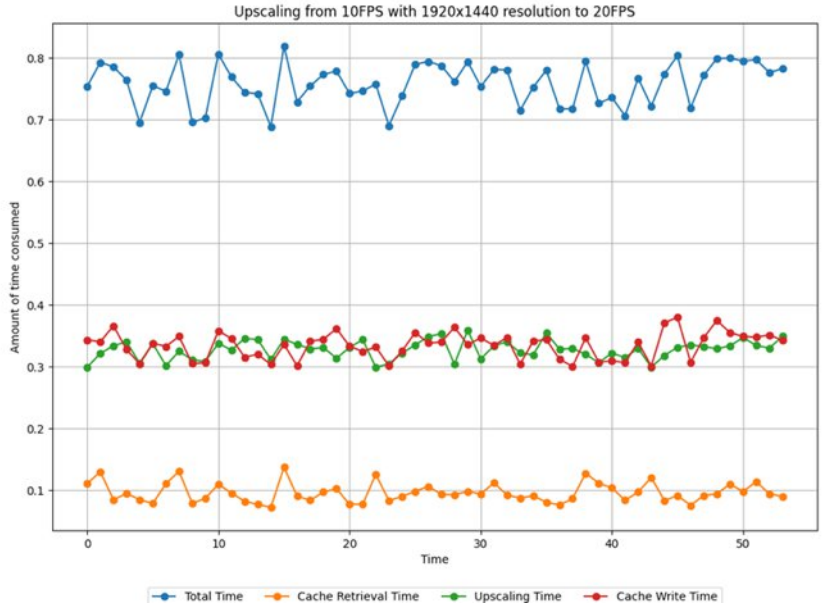


Figure 54 - Where the time goes - upscaling from 10fps to 20fps with 1920x1440 resolution

4.5.2.2 Test 2: Performance of AI Frame Rate Upscaling

Although modern XR headsets come equipped with FPS upscaling, we set out to investigate doing frame rate upscaling on the Edge. This offers a means independent of headset choice to ease experimentation while additionally enabling us to investigate the latest developments in frame interpolation that may not have made their way into commercial headsets.

We investigated two approaches:

- Lucas-Kanade Optical Flow: estimates the motion vectors (displacements) of image features between two frames and uses these to guide the generation of new frames

- RIFE (Real-Time Intermediate Flow Estimation): Deep learning technique using Convolutional Neural Networks. Estimates optical flow in both directions. Also seeks preserve temporal consistency and is designed to work in real-time.

From the beginning RIFE produced clearly superior results and we observed the ability to upscale from 10FPS to 40FPS across a range of resolutions with sub-second performance.

Similar to the caching brake imposed on resolution upscaling we discussed in the previous section, we see a similar phenomenon in frame rate upscaling as captured below in Figure 51.

We see there is plenty of room left in the tank in terms of pure upscaling effort, but caching is limiting further growth.

4.5.2.3 Test 3: Performance of AI Trajectory Prediction

We adopted an LSTM approach for trajectory prediction and trained a model using a small number of recorded flight trajectories and tested with an unseen trajectory. The position of an aircraft is captured by a set of values for latitude, longitude, heading, altitude, pitch and roll. Of these figures, we would expect a fast-moving commercial aircraft to experience most change on the geographical coordinates – latitude and longitude – and this has been borne out with our predictions which demonstrate prediction errors on these vectors. Our results can be viewed below in *Figure 52*.

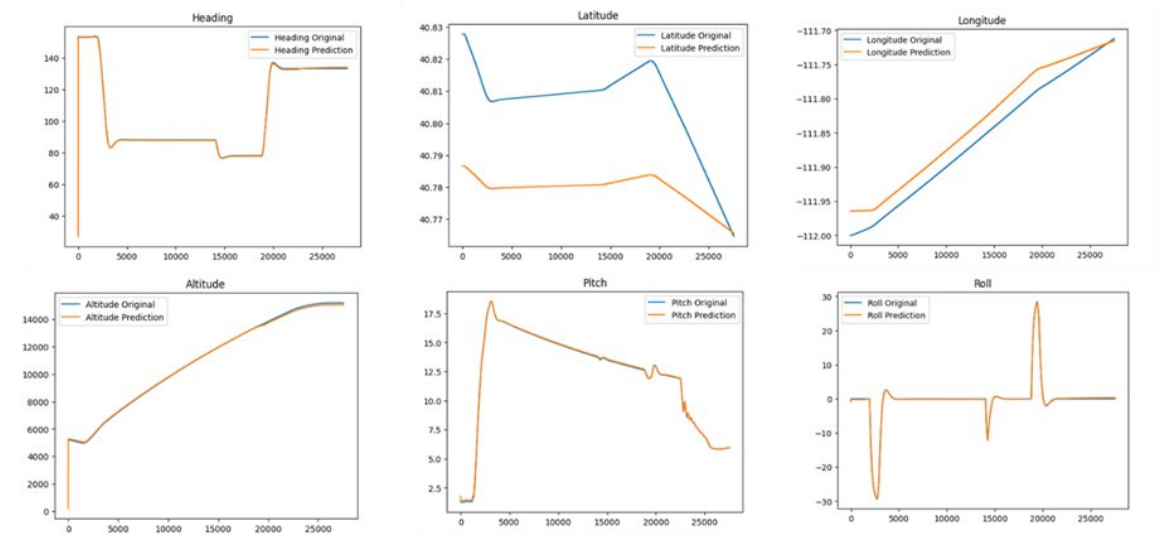


Figure 55 - Predicted trajectory versus observed trajectory

While we observed deviations between the predicted positions and observed positions, we believe we have achieved sufficient accuracy to demonstrate the prediction concept.

4.5.2.4 Test 4: Software Adaptation

Adaptivity was a key target for the Flight Simulator Use Case from the outset. We sought to facilitate configurable rendering sophistication such that we could alter the weather effects and enable or disable advanced rendering features such as shadows and reflections. Additionally, we sought to support configurable frame rate and resolution. To facilitate integration with the Dynamic Software Adaptation model in Task 3.3, the use case needed to support a coherent and joined-up configurability

model that would orchestrate configuration per user across multiple services[6] even when we do not have access to the source code of those services.

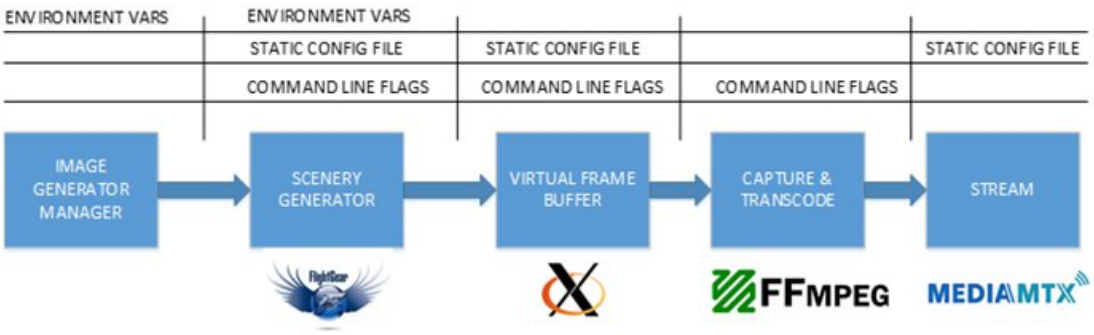


Figure 56 - Diversity of configuration channels for remote rendering components

After containerizing all services with Docker, we first implemented a co-ordinated configuration scheme using Docker Compose to centralize configuration for groups of containers and then evolved this to use Kubernetes ConfigMaps. We deployed Prometheus, custom exporters, and its Alert Manager to facilitate the configuration of environmental triggers (such as GPU busy-ness) that could initiate a Kubernetes rolling update – the launching of alternatively configured pods to seamlessly take over the operation of existing pods and in effect offer dynamically adapted software without session interruption.

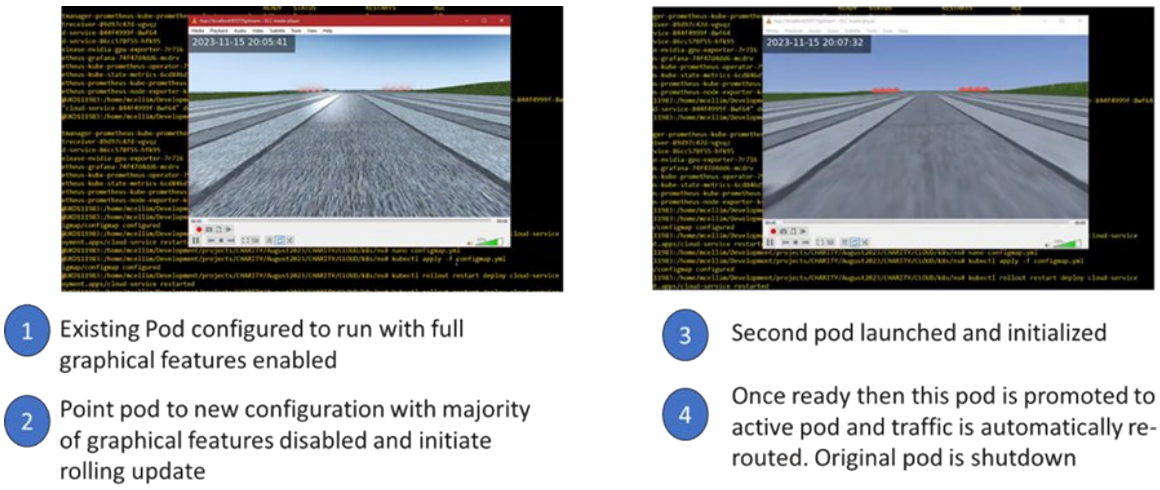


Figure 57 - Dynamic Software Adaptation using rolling updates for the Collins use case

We were able to successfully validate the model experimentally with maintaining a user session while handing over from one rendering pod to another. We added Kubernetes readiness probes to delay handover of the media streams. These probes seek to ensure that the newly launched scenery rendering pipeline is up and fully operational before switching traffic over to it. Although the probes helped us to narrow the gap, we could not successfully capture the exact point that the scenery generator was operational and streaming. This results in occasions where the user's session is briefly interrupted with a splash screen before their session is resumed. We don't however, see this as a general limitation of the design.

[6] To support a configurable frame rate, for example, requires the FlightGear Image Generator to be instructed to operate at a particular frame rate generation cadence; for this frame rate to be supported by the virtual frame buffer; for the ffmpeg transcoder to acquire frames at this rate and stream them to the RTSP server at this rate; and for the reader of the resulting video stream exposed by the RTSP server to be aware of the incoming frame rate target to operate correctly. We cannot require this to involve multiple configuration settings in different services as this would be too error prone.

We ran various experiments to observe the variations in resource consumption of the flight simulator Cloud Pods under different configurations and results are shown below in *Table 17*. The flight simulator can be run with just a single window (showing the scenery straight ahead) or multiple windows for left and right views[7]. Low QModes signify operation with disabled advanced graphical features (smoke, shadows, etc.) while high QModes signify operations with all features enabled.

The bandwidth reflects the amount of data being sent from the transcoder to the streamer.

Table 20. Resource usage profiles across various configurations

QMode	GPU Memory usage (MiB)	GPU utilization	Frames Per Second (FPS)	Resolution	Bandwidth (MB/sec)
Low single window	105	2%	10	848x480	0.1
Low single window	120	4%	20	848x480	0.16
Low single window	122	13%	60	848x480	0.38
High single window	208	3%	10	848x480	0.51
High single window	208	6%	20	848x480	0.75
High single window	208	18%	60	848x480	1.2
Low multiple window	270	8%	10	848x640	0.3
Low multiple window	284	16%	20	848x640	0.35
Low multiple window	316	37%	60	848x640	0.5
High multiple windows	675	13%	10	848x640	1.35
High multiple windows	675	29%	20	848x640	1.75
High multiple windows	675	33%	60	848x640	2.4
High single window	268	6%	10	1920x1080	2.1
High single window	268	16%	20	1920x1080	2.6
High single window	268	19%	60	1920x1080	3.2

The experiments revealed the somewhat surprising effects of advanced graphical flourishes on the bandwidth requirements for video streams. We witnessed a five-fold increase in bandwidth between a stream with advanced graphical features turned on (High Single Window) versus the same stream with the features turned off (Low Single Window). The relationship between bandwidth and graphical effects can be explained by the amount of additional variance that graphical effects (such as rain, shadows, smoke, etc.) produce across video frames and thus increasing the amount of change from one frame to another – thus reducing the benefits of savings that can be achieved with video codecs.

Graphical flourishes are not easily recovered if not included at rendering time so while we can indeed reduce the resource footprint, we cannot do so without clearly visible degradation of service to the end user. Resolution and frame rate are aspects we can seek to degrade at the rendering source with the objective to recover them closer to the user. From the experimental results shown in *Table 17* we can observe the effect of going from Standard Definition (848x640) to High definition (1920x1080) resolution at 10 frames per second is an effective quadrupling of the bandwidth needs and doubling of the GPU utilization.

[7] We encountered a persistent problem with the display of the right-hand window with FlightGear that we have not yet succeeded in solving. The results with multiple windows only represent two windows.

4.5.2.4.1 Automated Adaptation Strategies

Demonstrating we can adapt through leveraging Kubernetes rolling updates still leaves the question on how to orchestrate such adaptations through monitoring. Different users have different needs depending on what they are doing and what individual priorities we associate with the users. An aircraft positioned on the runway exerts less scenery rendering pressure that one flying at 200km per hour. An aircraft flying above the clouds exerts less rendering pressure that one flying low above detailed terrain. A trainee focused on experimentation with the cockpit trying things out does not necessarily require the same scenery fidelity as one participating in a regulatory training session. We desire a scheme that dynamically adapts to the circumstances at hand, and this requires monitoring and reacting on an individual user-level basis.

A key strategy we identified in tackling this challenge was to deploy separate application instances, running in separate pods, per trainee. This would allow us to reduce the problem of monitoring all users to monitoring one user and then replicate the solution per user. This requires us to gather metrics per pod, raise alerts per pod, and react per pod. This scheme is outlined below in Figure 55:

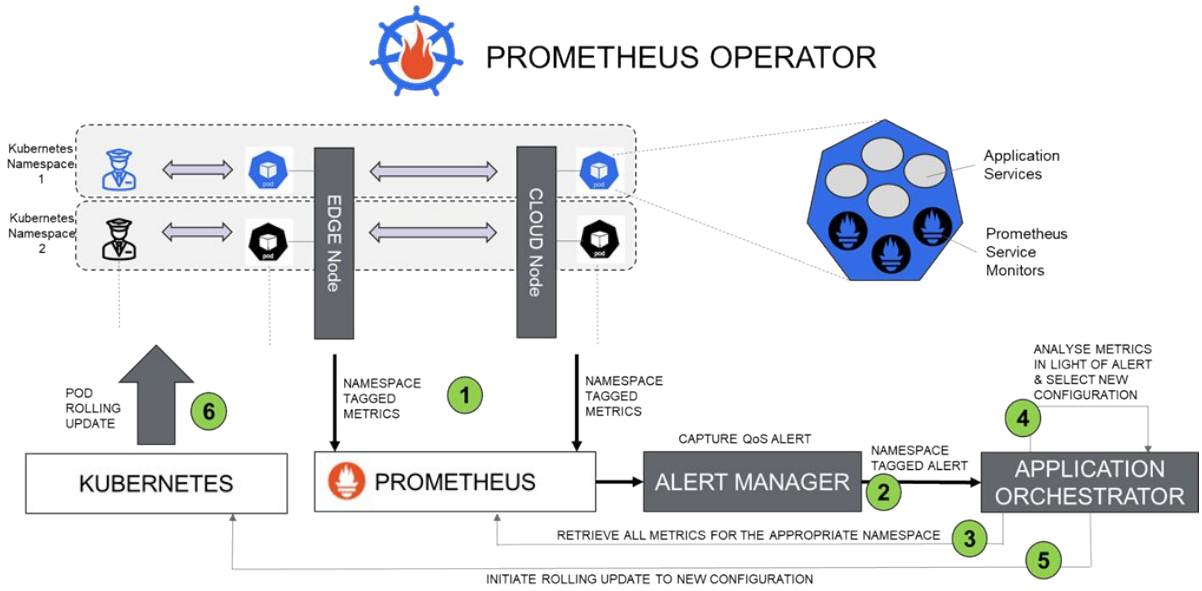


Figure 58 - Employing a Prometheus Operator and Kubernetes namespaces for segregated metrics

We employ a Prometheus operator to manage the deployment of service monitors inside our application pods. Each user is managed in a separate session with their own allocated edge and cloud pods. Each user's pods operate in a distinct Kubernetes namespace (essentially their session id). The essential scheme operates as follows:

1. Metrics reported to Prometheus from within pods are tagged with the namespace assigned to the user.
2. Alerts configured within the Alert Manager are raised within the context of a namespace and relayed to an Application Orchestrator (this would be developed and deployed by the application owner – it accepts alerts and decides on the appropriate course of action) via webhook, passing the name of the namespace in which the alert was raised as a query parameter.
3. The Application Orchestrator queries Prometheus using the namespace as a query filter to retrieve relevant metrics for that namespace.
4. The Application Orchestrator contains logic to select an appropriate course of action according to the alert raised – this boils down to selecting an appropriate configuration to move the pod to which will alleviate the difficulties revealed by the alert and the pod metrics.
5. Once a configuration option has been decided on (selection of one of a small number of pre-configured Kubernetes configmaps holding environment variable settings controlling the behaviour of the application at launch time), then the Application Orchestrator initiates a rolling update via the kubectl API to move the pod to the new configuration.

6. Kubernetes manages the rolling update and smoothly starts up a new pod, retires the original and swaps over using ingress and egress services in a service mesh arrangement.

We tested with a small number of different scenarios to validate the concept which we fell validate the design.

Scenario 1: A new user creates a session resulting in a user limit being reached (this can be managed without resorting to alerts as it is tracked by the application session manager – in the Collins case the Session Management functionality is included in the Application Orchestrator for simplicity). This limit results in the existing users being moved from full featured cloud rendering (with advanced rendering shaders enabled) to low quality cloud rendering and thus reducing resource usage by the cloud pod.

Scenario 2: An alert is raised for a user being served with a high Quality Of Service (high frame rate and resolution on the cloud) resulting in the cloud pod being moved to a low Quality of Service (low frame rate and resolution on the cloud, upscaled at the edge).

At the root of each scenario is the ability to invoke APIs on the Kubernetes API Gateway (kubectl) to change a pod configuration. The conditions under which we invoke these APIs may encompass any of a wide range of considerations – time of day, user priority, resource availability, user device, aircraft stage of flight (e.g. landing, take-off, cruising, etc.). The mechanics of adaptation is the same regardless of the conditions that drive it to occur.

In Figure 56 below, we present a screenshot of our Grafana dashboard where we bring attention to the change in resource usage as a result of initiating a live software adaptation from an application running with cloud pod rendering at 20fps, a resolution of 1280x1440px, and advanced shaders turn on. The pod is being moved to a configuration with a cloud pod rendering at 10fps, with a resolution of 640x480, and advanced shaders turned off. This scenery stream now needs to be upscaled on the edge in realtime and we can see the shift of resource usage this causes.

Figure 59 - Dynamic Adaptation in action – less cloud and more edge resources to upscale at the edge

The experiments revealed the successful operation of dynamic adaptation of a cloud pod rendering scenery.

4.5.2.5 Test 5: Assessing Edge Costs for Cloud Savings

Operating with reduced cloud resources means introducing additional compensating resources near the user if we want to try and maintain a similar Quality of Experience. If we operate at reduced frame rate and resolution at the remote rendering source on the cloud, then we sought to quantify what the cost of recovering this loss of fidelity could be. We conducted an experiment targeting 60 frames per second of Full High-Definition resolution. This required approximately 20% of our GPU, 1% of GPU

RAM and 2.6 MB/s. All advanced graphical features were turned on. This is summarised below in *Figure 57*.

Figure 60 - Generate high quality at rendering source

Reducing the frame rate at source to 15 and the resolution to Standard Definition reduced our bandwidth needs from the cloud by approximately 90% and the GPU consumed on the cloud by 75%. Clearly significant savings. However, attempts to recover this loss at the edge requires the use of additional physical resources. For the purposes of our experiment, the edge node is the same machine as used for the cloud, so this gives us a like-for-like comparison. The resolution upscaling model we are using is very demanding on GPU RAM, so we see very significant uptick in this area. The results are summarized below in *Figure 58*.

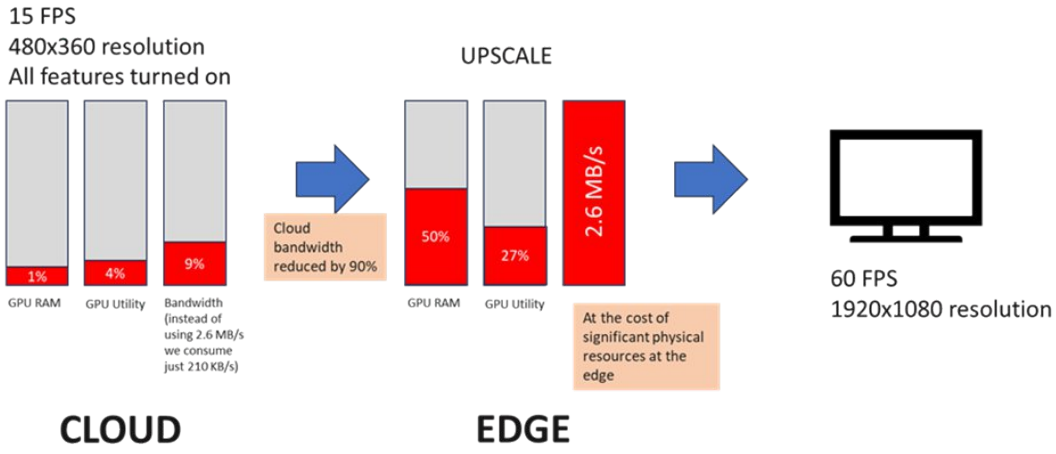


Figure 61 - Generate low quality on the cloud and seek to recover quality at the edge. Significant bandwidth reductions but also significantly increased resource usage overall

4.5.2.6 Test 6: Turnaround performance

The performance has been regulated throughout by how quickly we can move large amounts of data at the edge. Video codes excel at reducing the amount of data that needs to be shuttled across networks for streaming. To process a scenery stream at the edge to cache and upscale requires us to decode the stream into raw frames. We quickly find ourselves dealing with substantial volumes of data and hitting debilitating ceilings of performance.

There are three key pinch points – caching, communication and GPU data exchange.

4.5.2.6.1 Caching

We used the opensource in-memory key value store REDIS for our frame cache. This data store is renowned for its speed.

We experimented with various compression algorithms to minimise the space occupied by video frames – MsgPack, Blosc and Pickle[8]. MsgPack was quickly discounted as it consistently performed worse than the others as can be seen below in Figure 59.

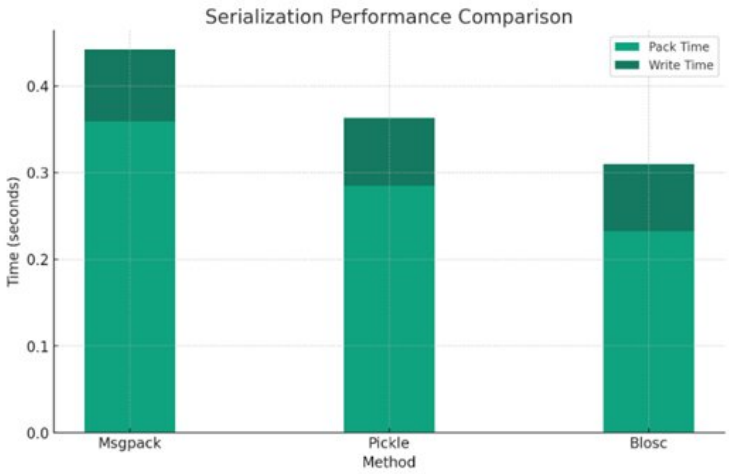


Figure 62 - Serialization performance for 20 frames per second at resolution of 1920x1440

Using Blosc as shown in Table 18 however, still left us with the need to pickle the resulting data anyway to store in Redis. Further experimentation with a newer version of Pickle (using Protocol 5), nudged it ahead of Blosc as seen in Table 19.

Table 21. Blosc Compression Results for 24 frames at a resolution of 1920x1440px

Compression Level	Typesize	Pack Time	Write Time
0	8	~0.320	~0.090
5	8	~0.370	~0.110
9	8	~0.420	~0.090
0	4	~0.300	~0.100

Table 22. Pickle Serialization Results (without Blosc) - 24 frames at a resolution of 1920x1440px

Method	Pack Time	Write Time
Pickle (Protocol 5)	~0.350	0.085

Pickle serialization with the highest protocol seems to be an effective choice for simplicity and performance, offering a straightforward implementation with competitive pack and write times.

Subsequent experiments showed that writing a compressed 4K video frame to cache takes 0.01 seconds (consuming around 25MB of RAM) and reading such a frame from cache takes 0.07 seconds. If we do nothing else but read such frames from a cache, we can operate no faster than 14 frames per second on the testbed. For comparison, a 1K video frame takes just 0.002 seconds to write and 0.006 to read meaning we could operate at around 175 frames per second. These results demonstrated that it is not feasible to achieve 4K resolution at 60 FPS cached and streamed from the edge. Experiments suggest the limit is closer to 2K than 4K.

[8] See msgpack.org, github.com/Blosc/c-blosc, docs.python.org/3/library/pickle.html

4.5.2.6.2 *Inter-container communication bandwidth*

Docker containers communicating on a private Docker network do so through the Docker network stack which comes with a cost. This is depicted below in Figure 61.

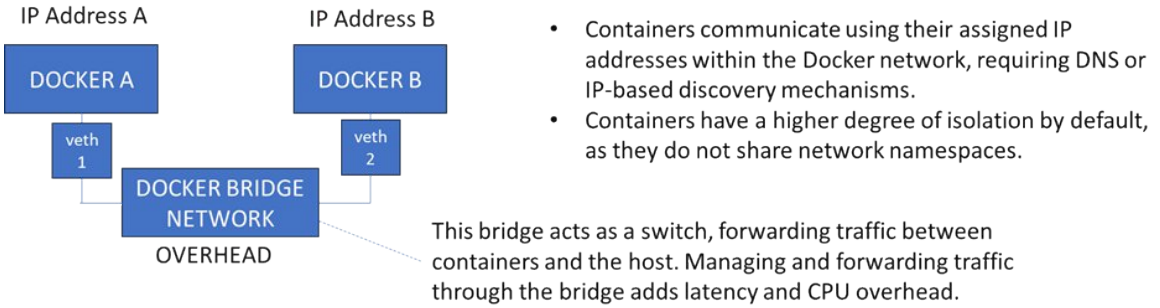


Figure 63: Communication between Docker containers does incur overhead and resources

With the REDIS Frame Cache container playing a pivotal role in the edge and experiencing large volumes of data transfer, we needed to assess this overhead. Using iPerf to run a 20 second stress test of the network between two containers revealed an average transfer speed of approximately 6.6GB/sec. This decreases as we increase the activity on the docker network. With five containers communicating with 5 others, for example then the effective bandwidth available drops to ~ 5.3GB/sec. This is important as it reveals some underlying limits for an EDGE host and the number of high bandwidth containers it can service simultaneously.

Using Kubernetes and pods we find an approach that scales better as there is less overhead as shown below in Figure 62.

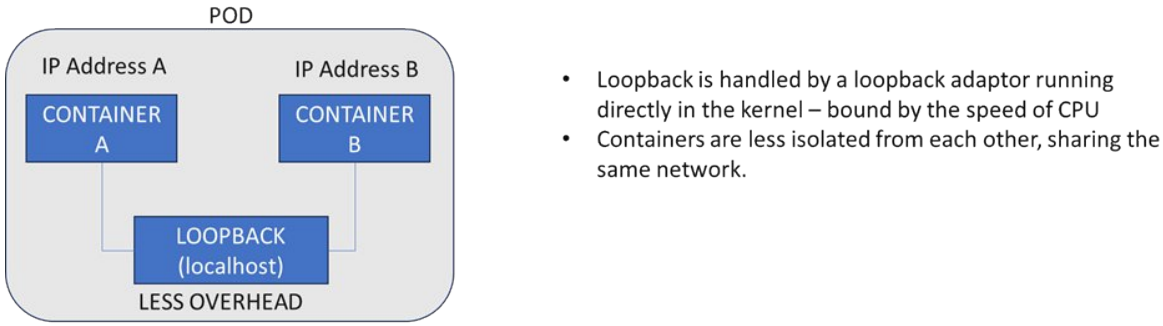


Figure 64 - Communication within a pod incurs less overhead

This was borne out with testing as captured in the graph below in Figure 63 using data gathered using iPerf between containers using a private Docker network versus a direct kernel loopback as employed in Kubernetes pods.

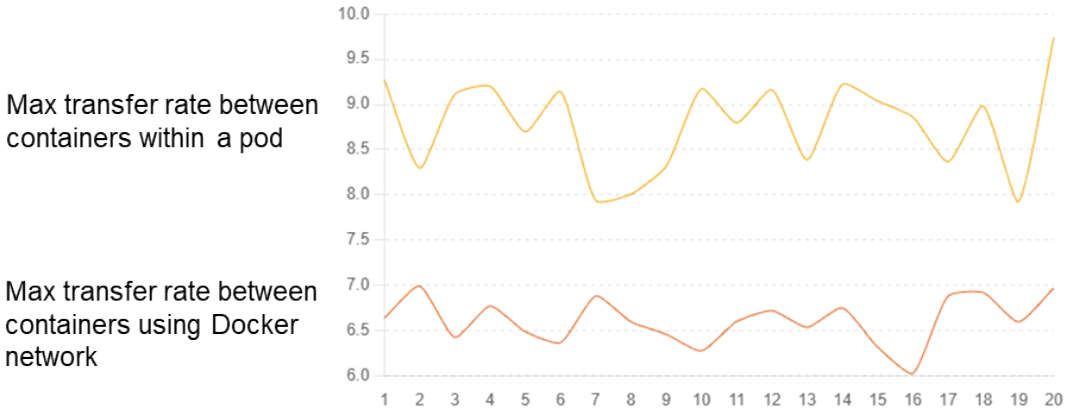


Figure 65 - Containers within a pod have higher bandwidth available than dockers

The bandwidth of inter-container communication jumped from 6.6GB/sec to 8.8GB/sec and multiple instances of inter-container communication scale better. In addition, the level of retransmissions in inter-container tests but not in pods suggests that the virtualized network path between containers may be subject to conditions causing packet drops or errors. These findings served as an additional incentive to move from standalone Docker containers to Kubernetes pods.

4.5.2.6.3 GPU to Host Transfer

GPUs are extremely fast but as we mentioned earlier during our discussion of upscaling in Test 1, there is a major bottleneck that often goes unmentioned. Transferring data from CPU memory to GPU memory and back can incur enormous overhead. This quickly comes to the fore when dealing with real-time operations on large video frames. Upscaling a frame from 1k to 4k can be done very quickly by a GPU but reading that 4K frame from the GPU to the host incurs more overhead than the upscaling – possibly an order of magnitude more. The problem is more pronounced with frame rate upscaling. Sending ten frames in and seeking to get 30 frames out drives the turnaround time from tens of milliseconds to hundreds.

We ran some benchmarks on our test-bed using a high-end RTX 3090 GPU to assess the GPU to host transfer limits. We see the results below in Table 20. Note that these figures *do not* include any actual processing effort on the GPU.

Table 23. Measuring the GPU to host transfer limits for transferring 10 frames in & 30 frames out

Frame resolution	Host -> GPU transfer	GPU -> Host transfer
1920x1440	0.041487 seconds	0.232328 seconds
2560x1920	0.096514 seconds	0.488443 seconds
3840 * 2160 (4K)	0.120750 seconds	0.815096 seconds

We can see from above that interpolating 4K frames hits physical limits around 30. In reality, this will be far less as we are not allowing for the actual frame interpolation processing effort itself.

The situation is further constrained. Since we also have to get the original frames from a frame cache and write the resulting frames back to the frame cache, we have far less time than one second available to upscale one second's worth of frames. To balance resolution quality and frame rate quality, we found the maximum stale throughput to be 20 fps at 1920x1440. The cloud rendering pod could drop to 10fps and 640x480 resolutions and we could still manage to upscale to the target rate in realtime. Findings are presented below in Table 19.

Table 24. The limits of Frame rate upscaling and caching

Input FPS	Upscaled FPS	Cache Retrieval Time	FPS Upscaling Time	Cache Write Time	Total time (needs to be < 1 second)
10	30	0.10181	0.7859	0.6397	1.5275
10	25	0.08519	0.5904	0.4940	1.1696
10	20	0.08441	0.4340	0.3737	0.8921

Between upscaling and caching, the highest stream we could deliver reliably during our experiments was 1920x1440px and 20FPS.

4.5.2.7 Test 7: Scalability

We have designed and developed the use case to deploy two Kubernetes Pods per user – one on the cloud and one on the edge. Apart from a shared monitoring infrastructure, there are practically no shared components between users. This model enables us to cleanly adapt the Quality of Experience per user according to their location, priority, and device characteristics. It also makes scaling very predictable. If a single user requires resources X then two users will require resources 2X and so on.

We deployed Cloud pods on an AWS EC2 instance with a Tesla T4 GPU and had no issues launching 12 parallel user sessions with resources to spare. This was with all rendering features enabled at moderate resolution (1920x1440) and frame rate (20 FPS)[9] - the objective was to validate the independent scalability of users. Below in Figure 64 we see a view of the parallel streams being displayed followed by a snapshot of GPU utilization in Figure 65.

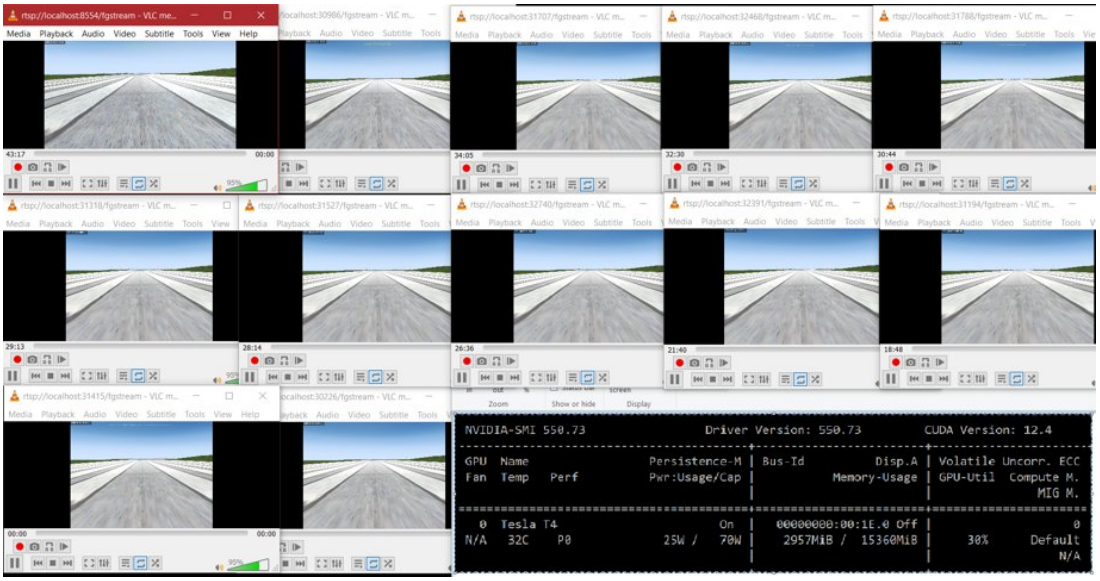


Figure 66: Parallel deployment of cloud pods on an AWS EC2 instance

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 550.73                 Driver Version: 550.73         CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       On  |   00000000:00:1E.0 Off |                    0 |
| N/A   32C    P0             25W /  70W |    2957MiB /  15360MiB |     30%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A     37712      G   ./usr/games/fgfs                  264MiB |
|    0   N/A  N/A    112942      G   ./usr/games/fgfs                  244MiB |
|    0   N/A  N/A    249198      G   ./usr/games/fgfs                  244MiB |
|    0   N/A  N/A    257177      G   ./usr/games/fgfs                  244MiB |
|    0   N/A  N/A    258119      G   ./usr/games/fgfs                  244MiB |
|    0   N/A  N/A    258249      G   ./usr/games/fgfs                  244MiB |
|    0   N/A  N/A    258453      G   ./usr/games/fgfs                  244MiB |
|    0   N/A  N/A    259077      G   ./usr/games/fgfs                  244MiB |
|    0   N/A  N/A    261228      G   ./usr/games/fgfs                  244MiB |
|    0   N/A  N/A    269000      G   ./usr/games/fgfs                  244MiB |
|    0   N/A  N/A    294407      G   ./usr/games/fgfs                  244MiB |
|    0   N/A  N/A    315830      G   ./usr/games/fgfs                  242MiB |
+-----------------------------------------------------------------------------+
```

Figure 67: Resources are not overburdened

[9] This resolution and frame rate alone would prove serviceable for flight training but higher targets would be expected for commercial trainers

The scalability of cloud rendering is demonstrated. In principle, the edge pods scale according to a similar model. As with the cloud pods they have no shared dependency so scaling is reasonably linear. We did not have sufficient resources on our testbed to scale as with the cloud pods but demonstrated that we could comfortably service three edge pods on the testbed upscaling with a frame rate of 20FPS and a resolution of 1920x1440.

4.5.2.8 Test 8 – XR Integration

We developed Augmented Reality and Virtual Reality prototypes that integrate the scenery generated from the use case prototype. Experiments showed that remote rendering of the scenery on the cloud significantly reduces the burden on the XR application as scenery frames do not need be re-rendered but can instead be quickly inserted into the XR experience using very lightweight texture mapping. Even in heavily resource-stressed XR environments, the scenery streaming was continuously stitched into the surroundings. In Figure 66 below, we see a Hololens Augmented Reality model in which we have streamed the front window generated and upscaled across cloud and edge pods deployed on our testbed into two separate floating screens.

Figure 68 - Hololens demonstration of a scenery stream generated on the testbed

Even though our frame source operated at 20 fps while Hololens operated at 60 fps, the resulting experience demonstrated the smooth accommodation of the generated scenery.

In Figure 67 below, we see a screenshot from the view of a basic VR cockpit with our generated scenery stream outside the windows. This used a Vive 2 headset.

Figure 69 - VR demonstration of a scenery stream generated on the testbed

What is particularly interesting about Figure 67 is that it captures a snapshot of the Frame Timing on a heavily resource-stressed machine - shown in the top left-hand corner. This headset operates at 90 frames per second[10]. The red in the top panel[11] of the Frame Timings signifies instances where the application could not generate frame instructions quickly enough to deliver to the GPU in order for the frame to be rendered. When such events happen, the GPU has to guess itself – it has to interpolate a frame and keep things running at a steady 90 frames per second. This is known as motion smoothing. Even in the face of intensive motion smoothing, the scenery is displayed without judder.

4.5.3 KPIs assessment

The current assessment of how the Flight Simulator Use Case satisfies the original requirements identified is presented below in *Table 22*.

Table 25. Requirements status for Flight Simulator Use Case

Requirement	Description	Comment	Status
F_UC3_13	The simulation must facilitate collaboration between users to efficiently execute the simulated mission	This is a feature of the Flightgear scenery generator itself - they offer a shared server model in which each generator considers the location of other players when rendering scenery.	Implemented
F_UC3_14	Scenery generation may support scenery with different weather	Implemented and validated.	Implemented
F_UC3_15	The simulated environment should allow participants to join	Infrastructure support multiple parallel users. Flightgear supports	Implemented

[10] The image was captured from a recording of the VR screen using a mobile device - this particular machine that the headset was tethered to for this test was too poorly resourced to enable screen recording during the test.

[11] On a sufficiently resourced machine, such red peaks would be practically non-existent

	or leave simulation at any time	common participants in a shared airspace.	
F_UC3_16	The simulation should enable prediction of background scenery demands so that it can be pre-fetched by any component from off-line storage	Implemented and validated.	Implemented
F_UC3_17	The simulation should enable custom tiling of cloud-based image generator output to facilitate variable resolution across a single frame	Validated and implemented. Scenery segmented into side and front windows with separate streams.	Implemented
NF_UC3_18	The simulation should adapt imagery frame rate and resolution in accordance with available bandwidth, observed latency, and user equipment capabilities	Validated and implemented. Have demonstrated the capability to switch between different frame rates and resolutions through configuration with selection driven by metrics.	Implemented
NF_UC3_20	The RTT from user action to presentation of updated imagery should be < 15ms	This was not successfully validated. The opensource RTSP streaming service we used implements internal buffering that adds delay to the presentation of the media stream. However, we did successfully buffer frames and feed them to the RTSP in realtime. It is important to point out however that we successfully demonstrated texture mapping of the scenery onto a VR cabin model that is streamed from the local device so we are operating within the headset budget as far as user experience is concerned.	Partially Implemented
NF_UC3_21	Number of concurrent users (virtual & real) in a single simulation scenario should be > 30	As users are serviced in independent pods that can be deployed and distributed according to resource availability with no shared resource, a 30 user scenario is attainable. We have demonstrated parallel users with no scalability bottleneck.	Implemented

F_UC3_22	The simulation should be able support both active participants (present in the simulated environment) and passive observers (not present in the simulate environment)	Implemented and validated. We can attach multiple viewers to a given RTP stream.	Implemented
NF_UC3_23	The video resolution of presented imagery must be greater than 60 FPS 4K.	This could not be achieved with edge upscaling. However, our design which separated scenery rendering from the cockpit, avoids the need for such a high frame rate. Our target was 30FPS to match commercial flight simulators but the best we could reliably achieve was 20-25 fps at a resolution of 1920x1440. For Mixed Reality, the Hololens device we tested with required a resolution of 1440x936 which was comfortably within range. If we forego edge upscaling and connect directly to cloud pods from user devices than we can comfortably reach 60FPS at 2K resolution.	Partially Implemented
NF_UC3_11	The simulated environment must provide a consistent simulation state across all users, including rendering of other user activities	This has been achieved through the segregation of different users into different pods that can be deployed independently.	Implemented

In terms of KPIs, the following were listed at the outset of the project.

- KPI-UC-3.2 RTT (aeronautical) – sum of network latency and game server response time < 15ms
- KPI-UC-3.3 Number of CCUs>30
- KPI-UC-3.5 Data services required (raw data streaming, rendering, compression, caching, encoding) >=5

The first two have already been discussed previously while the third regarding data services has been comfortably exceeded.

4.5.4 Benefits from the use of the Platform/Component

As previously discussed in Deliverable 4.5, the Collins Flight Simulator UC has not been integrated with the wider CHARITY platform for several practical and logistical reasons. Nevertheless, the design has

been driven and guided by the decisions, selections and strategies adopted in the overall CHARITY architecture. We chose a path that would seek to deliver CHARITY compatibility if not integration. In adopting Prometheus, Alerting and Grafana we mirror the metrics and monitoring stack. In shaping and redesigning our architecture to adhere to cloud native principles, we seek to ensure that our use case can be deployed and orchestrated by the CHARITY platform. In Figure 68 below, we identify the common technology stack elements we share with the overall CHARITY platform.

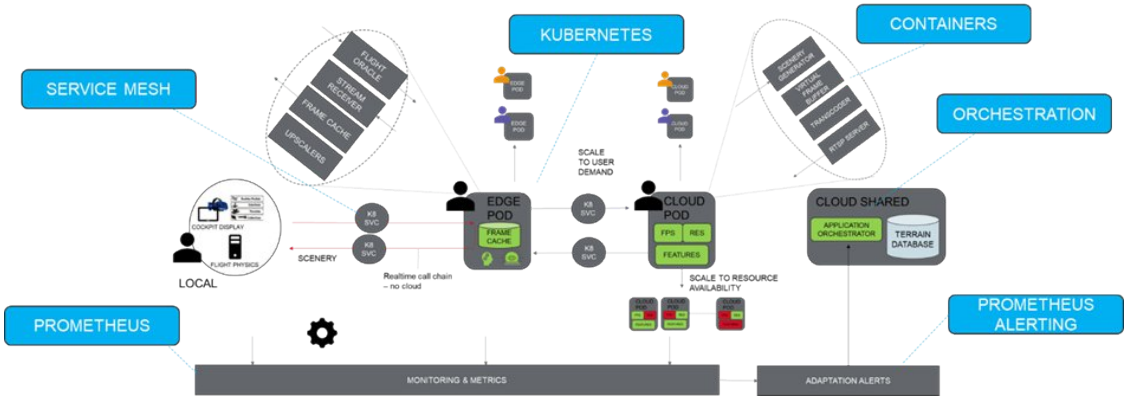


Figure 70 - Integration of Collins Use Case with CHARITY design patterns and technology stack

In the design of dynamic software adaptivity for Task 3.3, we proposed and implemented a model leveraging core elements of the CHARITY technology stack – Prometheus and Kubernetes – to formulate a design that could be adopted by cloud native applications and offer a seamless integration path. The Collins Use Case was integrated with this framework and used to demonstrate its validity and applicability.

While we do not currently enjoy the benefits of dynamic deploy-ability and re-deploy-ability offered by the CHARITY platform, we have made significant gains including :

- Demonstrated a transition from a challenging on-premise monolithic architecture to a distributed model
- Learned much about the pitfalls and pinch points inherent in the distribution of media streaming and dynamic upscaling
- Demonstrated how containerization, monitoring and the orchestration of containers within a multi-user distributed environment can be achieved in the context of media streaming applications.
- Demonstrated how an XR flight simulator could be decomposed with scenery rendering on the cloud for centralised asset and user management.
- Demonstrated how cloud native applications media streaming can be dynamically adapted at runtime – without the need for runtime APIs to be designed and provided by such applications.

5 Platform Validation & Lessons learnt

The overarching concept of CHARITY as a platform for facilitating the deployment of next-generation XR applications has proven to be a complex challenge. First and foremost, a diverse array of scenarios and use cases, each with distinct requirements, must be considered at various levels. Then, the journey from their representation to their realisation is a complex task. Whereas TOSCA provides a somehow standard format for representing application components and requirements, it remains to the interpretation of how to harness it for the specific goal of Cloud-Native XR application representation. Moreover, despite the benefits of TOSCA as a platform-agnostic format, translating them into actual running services is challenging. In CHARITY, we concentrated on cloud environments utilising Kubernetes, the de facto standard for microservice orchestration. However, this translation process is far from a straightforward engineering task, and Kubernetes alone does not provide an answer to that. Instead, as hypothesised at the beginning of the project and later proposed in CHARITY architecture, different components and enablers are required. AI, increasingly relevant in several domains, adds value to CHARITY's aim of autonomous orchestration. In CHARITY, we explored the role of AI in two contexts: the support for the decision of the location for XR application deployments and the prediction of their resource pattern during the runtime (further exploring the idea of dynamic reallocation when needed). Both proved to be successful. In CHARITY, we also considered the multi-domain aspect of the edge-to-cloud continuum and the notion of having application components spread across them. While that brings complexity, it also conveys flexibility and benefits in how the XR applications, orchestration and infrastructure are designed. Moreover, in CHARITY, we leveraged state-of-the-art Cloud-Native OSS – ClusterAPI and Liqo – to bridge such notions of the CHARITY overall orchestration solution. These two remain relevant and are expected to be pivotal in further research. Furthermore, the integration and evaluation tests were successful in general, and in addition we repeatedly received positive feedback in showcasing activities. For instance, at the last EuCNC & 6G Summit 2024, where the entire CHARITY concept and workflow were showcased, attendees highlighted the modularity and the integration of various OSS and AI components. Moreover, they expressed interest in the CHARITY framework as a tool for simplifying XR development and reducing the barriers to Cloud adoption for less familiar XR developers. Overall, the integrated evaluation and showcasing fulfil its purpose of demonstrating the workflow of deploying and managing Cloud-Native XR applications.

In the remainder of this section, we highlight the lessons learned for individual components and enablers of CHARITY, updated from the D4.4. The effectiveness of the **high-level orchestrator** has been proven through simulations and simple platforms. This includes analysing its support for an increasing number of virtual clusters and a more comprehensive range of deployed applications, encompassing both example blueprints and those from project use cases. The **Low-Level Orchestrator** tests underscore the capabilities of the Low-Level Orchestrator in dynamically managing Kubernetes clusters and deploying containerised applications, as well as its effectiveness in facilitating cross-cluster networking and supporting distributed services in multi-domain edge environments.

The **Monitoring framework** tests provided valuable insights into the performance and responsiveness of CHARITY's architecture components. Through rigorous evaluation of Resource Indexing, Monitoring Manager, and Monitoring Agents, several key findings emerge: a) Resource Indexing demonstrates its capability to offer real-time updates on cluster performance and response latency, crucial for maintaining system efficiency and reliability, b) The Monitoring Manager effectively handles requests and exhibits acceptable latencies, ensuring timely access to critical monitoring data such as metrics history and active alarms/alerts, c) Monitoring Agents, represented by Prometheus servers, prove reliable in gathering and delivering accurate application performance data, essential for informed decision-making and troubleshooting. Furthermore, the test scenarios—Stable Monitoring, Migration, and New Cluster—highlight the framework's adaptability to different architecture states, ensuring continuous monitoring and scalability. The experimental results for the **Forecasting Model** compare a proactive horizontal scaling approach with a conventional reactive method across latency-related

metrics, demonstrating the superiority of the proactive approach. Similarly, comparing an intelligent proactive approach (IPFT) and a conventional reactive method (RFT) across fault tolerance-related metrics reveals the proactive approach's dominance. These findings hold true even when assessing different task scheduling algorithms like Round-Robin, MinMin, and MaxMin. In summary, the proactive strategies consistently outperform reactive methods across all examined metrics, indicating their efficacy in enhancing system performance and fault tolerance.

The **Point Cloud encoder/decoder** component has undergone testing and integration into the UC1-3 pipeline (the CPU version). After optimising the video codec parameters, the component operates at an overall frame rate of approximately 15 fps, for the processing and the streaming of RGBD at a resolution of 1280 x 752, 8 viewpoints simultaneously. This number of views allows users of the holographic display to make slight viewpoint adjustments without additional data transmission. Preliminary tests of the GPU version in isolation show promising results, with anticipated frame rates of over 30-40 fps, even on more complex scenes. In terms of KPIs assessment, the component meets KPI-4.3, which concerns specialised data services support, including streaming, rendering, compression, caching, and encoding. The potential performance of the GPU version satisfies the speed requirements necessary for the Holographic Assistant application to ensure a good Quality of Experience (QoE). The main lesson learnt is that even small artefacts in the (lossy) transmission may cause visible errors due to the depth channel. This should be investigated better in future version of the encoder. Another important lesson learnt is that the GPU version, to obtain high frame rate, requires a tight integration with the rendering pipeline, this makes more difficult to generalize this solution to different XR applications. Anyway, the CPU version can be exploited easily for XR applications with similar requirements, i.e. set of 3D points created from close viewpoints.

The **Mesh Merger** service can be deployed by the CHARITY platform, and it has been tested by the UC3-1 Collaborative Gaming Application. Tests demonstrate sufficiently fast processing times, ensuring gamers a good QoE. Specifically, less than 2 seconds are required to download and process a new acquisition into the Mesh Collider. Efficient transmission is facilitated by employing a binary version of a JSON containing a mesh PLY format. Despite the format not being compact, the number of triangles per mesh collider is manageable and suitable for an interactive experience. In terms of KPI assessment, it fulfils KPI-4.3, which pertains to specialised data service support, including streaming, rendering, compression, caching, and encoding. The experiments show us that the idea is effective and that extending the current REST API and its functionalities may lead to an innovative XR data service that is useful for many different AR applications.

The **CHARITY Adaptive Scheduling of Edge Tasks** has not been integrated into any Use Case due to the high requirement of a centralised scheduler and the complexity of adapting the implementation of the application to work on distributed workloads. In contrast, isolated test demonstrate the viability of using reinforcement learning to distribute streaming workloads to improve the application requirements regarding latency, accuracy and QoS. Moreover, it fulfils the KPI-2.1 that provides holistic support for orchestrating advanced media solutions focusing on distributing jobs on edge device architectures.

For **UC1-1 and UC1-2,** initial tests were conducted for video streaming over a wired local network. With a 1GB wired connection, latency ranged from 5000-7000ms, primarily due to local video manipulation component performance dependency. However, the results were hardware configuration-dependent, with significantly higher latency than expected, prompting the need for a cloud-based solution with enhanced computing power. Transitioning to video streaming via a cloud server, initial attempts with various local and web streaming protocols yielded unsatisfactory results due to latency issues. After adopting the WebRTC protocol and conducting two 2-hour sessions, promising results were observed. Latency reduced to under 1000ms, and audio-video synchronisation was achieved. The latency graph showed a relatively stable average latency of 150-200 ms, which is still deemed insufficient for the Holographic Concert and Holographic meeting cases, pending completion and testing of the cloud video manipulation component. In terms of KPI assessment, KPI-UC1-2.1 aimed for an average latency of <20ms, which was not met. Further testing post-completion

of video manipulation and synchronisation components is necessary. However, KPI-UC-1-2:3, which pertains to required data services, was satisfied with tests conducted involving transcoding, rendering, and networking services.

For **UC1-3 (Holographic Assistant)** the previous phase of tests (performed before SRT had left the consortium) evaluated the overall ecosystem's performance and operation, revolving around a specific use scenario. The latency between providing eye-coordinates and rendering new views from the 3D point cloud consistently remains below 60 ms. The framerate of the streamed 3D point cloud currently stands at approximately 5 FPS, but with GPU optimisations, this can be increased to 30 FPS or more. The delay between sending and receiving 3D point cloud data is around 3-4 seconds, primarily due to compression and buffering processes. In terms of KPI assessment KPI-UC-1.1 ensures that the average latency between sending input data and receiving 3D point cloud data is ≤60ms, which is consistently achieved. KPI-UC-1.5 focuses on the latency in speech input and output, aiming for ≤2 seconds. Typically, reaction times are below 2 seconds, but may vary depending on the load on speech recognition services. KPI-UC-1.3 targets a holographic visualisation frame rate of ≥30Hz. While the computation load is typically below 50%, resulting in a frame rate above 30Hz for holographic visualisation, optimisations are required to increase the frame rate of the content from edge to client, currently limited to 5 Hz. Further optimisations, especially utilising GPU processing, are expected to achieve the desired frame rate easily.

The experimentation of **UC2-1 use case (VR medical training)** with the CHARITY platform for the VR medical training application yielded significant benefits and exposed certain challenges. The platform enhanced the overall system performance through automated deployment APIs, streamlined workflows, and improved scalability. Transitioning to a distributed VR pipeline allowed the use of low-spec HMDs, transforming our framework to device-agnostic, broadening accessibility. The Remote Rendering Component offloaded heavy graphics rendering to cloud machines, resulting in higher fidelity interactive graphics in VR. Utilising cloud resources for physics computations enabled high-intensity tasks and supported over 50 concurrent users, facilitating collaborative VR sessions. The Application Management Framework (AMF) simplified component configuration with a user-friendly interface, reducing deployment complexity. Additionally, real-time monitoring capabilities allowed for responsive system performance and proactive adjustments to maintain consistent quality of service.

However, several challenges were encountered during setup and deployment of UC2-1. The limited availability of GPUs on the platform's data centres and some limitations on cloud resources, such as CPU and RAM, hindered VR performance. This highlighted the need for extra resources and for an efficient resource management within the platform. Some data centres also lacked public IPs for machines, necessitating the use of VPNs, which added complexity and latency to the final VR pipeline. The absence of Windows support for VM deployment within Kubernetes required significant adjustments to setup plans. Debugging was a relatively difficult process, due to the lack of a way to view the output of the container in the AMF, instead of directly accessing the kubernetes cluster. The API provided for accessing AMF resources deviated from Unity's typical standards, necessitating some workarounds. Addressing these issues is crucial for optimising future deployments and fully leveraging the CHARITY platform's potential.

For the **UC2-2 use case (VR Tour Creator)** the platform validation was completed with a 100% cloud native migration. Along the project duration we learnt many valuable lessons and increased experience in the scope of cloud services and architecture. Before CHARITY the use case was not cloud native and suffered a almost complete refactor and created new components. We implemented concepts like containerization for Docker and Kubernetes, which we had to learn and optimise our use case for it. Many challenges were acknowledged: 360 video livestreaming, assets optimisation for the web, loading VR experiences on many different devices, user quality of experience and usability, and so on. Arriving the end of the project, we are proud our use case evolved so much.

For **UC3-1 (Collaborative Gaming)** the ongoing development and testing of gaming components within the CHARITY platform involve various elements, including Game Clients (iOS app), Game Server

(Docker image), Mesh Merger (Docker image), and Game Servers Managers. The Game Server Docker image has been configured and prepared for manual deployment within the project's infrastructure, enabling successful testing of connections between Game Clients and Game Servers. Work is underway to develop and prepare the Game Servers Manager for fully automatic deployment of Game Servers. Testing initially focused on CHARITY AMF API connectivity and functions, and later on further tests when all orchestration components became operational. Latency measurement tools are built into Game Servers Manager, Game Server, and Game Clients, covering response time computation during game runtime. In terms of KPI assessment: KPI-UC-3.1 targets a Round-Trip Time (RTT) for gaming, with the sum of network latency and game server response time aimed to be <100ms. Progress toward this KPI has been substantial, with the current latency well within the target, indicating a highly responsive gaming environment.

The **UC3-2** (Cloud Flight Simulator) use case entailed transitioning from a traditional monolithic to distributed cloud-native model. We designed and developed a working prototype that integrates smoothly into Mixed Reality and Virtual Reality experiences. We demonstrated the practicality and feasibility of cloud rendering by decoupling scenery generation, rendering it on the cloud ahead of time, and then stitching into XR experiences in real-time. The original challenge parameters we set were somewhat superseded by our research and design during CHARITY. We found that 90 frames per second for scenery generation was unnecessary due to the technique we employed to decouple the scenery and the cockpit. This allowed the headset device to deliver a consistent 90 fps cockpit experience while we could target 30 fps for the scenery.

Much of the experimentation focused dealing with jitter, configurable adaptivity, and leveraging AI services at the edge for latency optimisations. Seven tests (and sub-tests) were primarily conducted on the Collins testbed, equipped with an Intel i9 processor and NVIDIA RTX 3090 GPU. Some limited testing was also conducted on the AWS cloud. In terms of KPI assessment, for the RTT the sum of network latency and game server response time, the target of <15ms caused some difficulty. We could retrieve a frame from cache and stream to an RTSP server in this time but internal buffering in the ffmpeg encoder and streaming server made it practically impossible to measure the delivery time from cache to device. However, the origin of this tight deadline was driven by the original constraints of XR headset latency to avoid inducing nausea in the viewer. A delay of 15ms in terms of scenery frame delivery implies a frame rate of 80 frames per second – far beyond the 30 currently adopted in commercial flight simulators. The latency of XR world environment (such as VR cockpit) is managed by the XR device and our solution does not impinge on this.

For the Number of CCUs the requirement with over 30 Concurrent Connected Users (CCUs), while not demonstrated, has been shown to be perfectly feasible as the architecture scales linearly with resources. A dozen simultaneous cloud pods were deployed on a single AWS EC2 instance.

For the Data services it exceeds the requirement of >=5 data services required, demonstrating ample coverage in this aspect. Overall, the Flight Simulator Trainer Use Case has largely achieved the goals it set out to achieve with the exception of scenery frame rate and resolution. This may well have been achieved streaming directly from cloud pods and not using the edge if we had elected to go that path at the outset. However, this path offered little opportunity for exploration and innovation, for probing boundaries and advancing.

We demonstrated that real-time upscaling at the edge is feasible showing the ability to upscale a low fidelity video stream in real-time from 10fps at 640x480 resolution to 20fps at 1920x1440 resolution. We demonstrated that direct streaming from the cloud can deliver both high frames rates and resolution supporting 2K running at 60 frames per second. Perhaps most importantly, we demonstrated the feasibility of building a distributed cloud native flight simulator promising significant benefits over the traditional on-premises model with superior versatility, robustness and adaptivity. Such a model facilitates deployment with less maintenance efforts, less logistical expenses, less up-front investment in hardware.

To summarize, the CHARITY architecture and its enablers possess considerable research and development value, which is pertinent for advancing future endeavours.

6 Platform Showcasing

This chapter features a collection of images and screenshots illustrating the CHARITY use cases, providing a visual representation of the concepts and processes discussed in earlier chapters.

6.1 Holographic Meeting & Concert Showcasing

Figure 71 - Studio for speaker in the holographic meeting

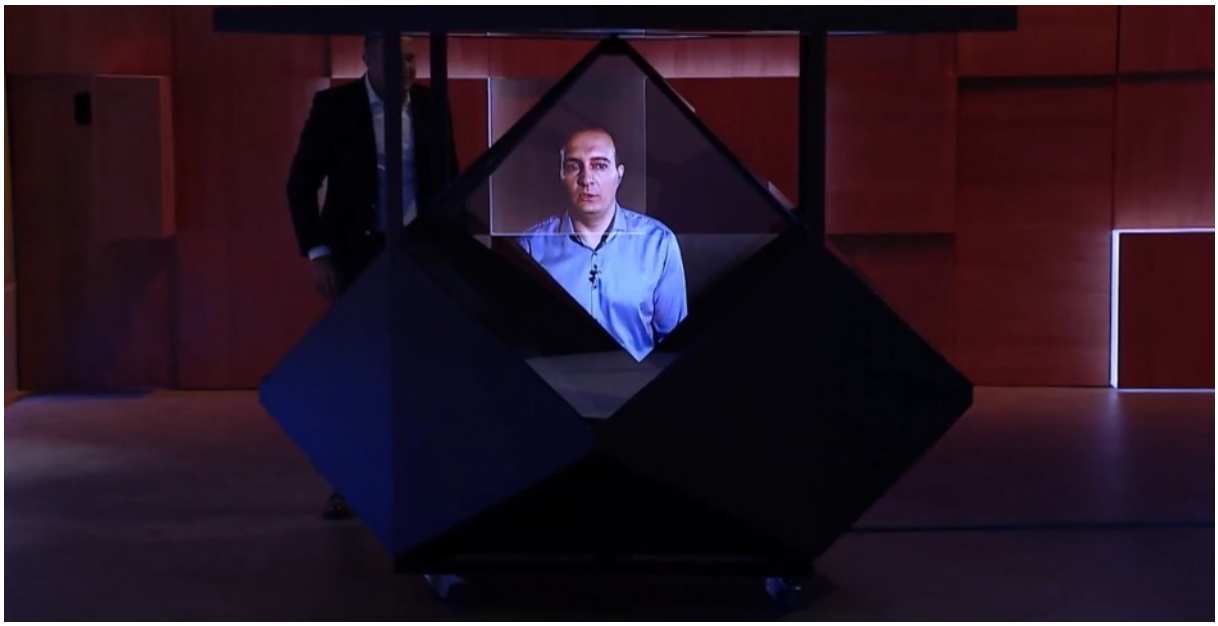


Figure 72 - Example of Speaker displayed on the Dreamoc Diamond Holographic device

Figure 73 - Example of Musician displayed on the Dreamoc Diamond Holographic device

6.2 VR Medical Training Showcasing

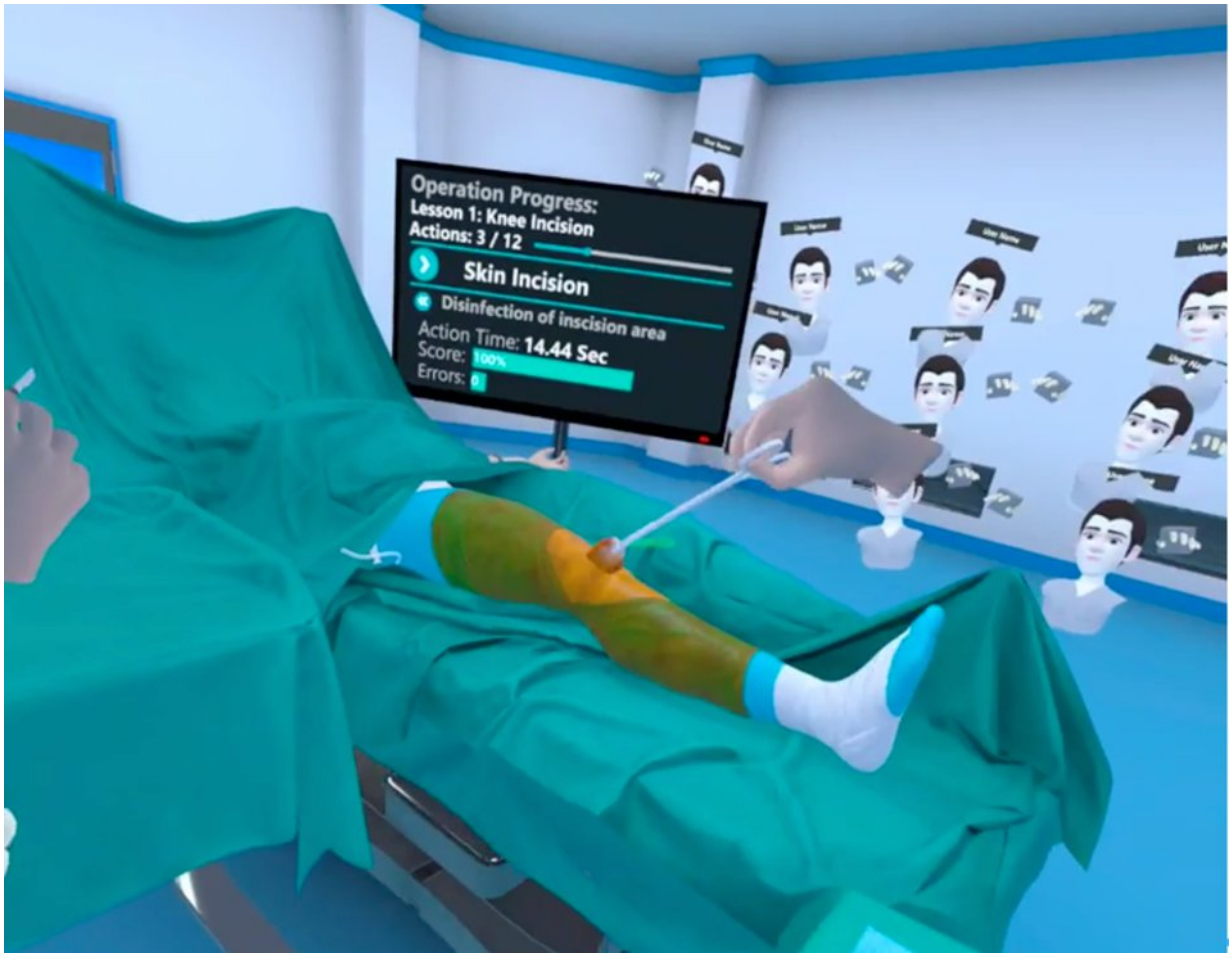


Figure 74 - 3 HMD users and 55 bots performing Knee surgery training

Figure 75 - VR Medical training showcased at EUCNC 2024

6.3 VR Tour Creator Showcasing

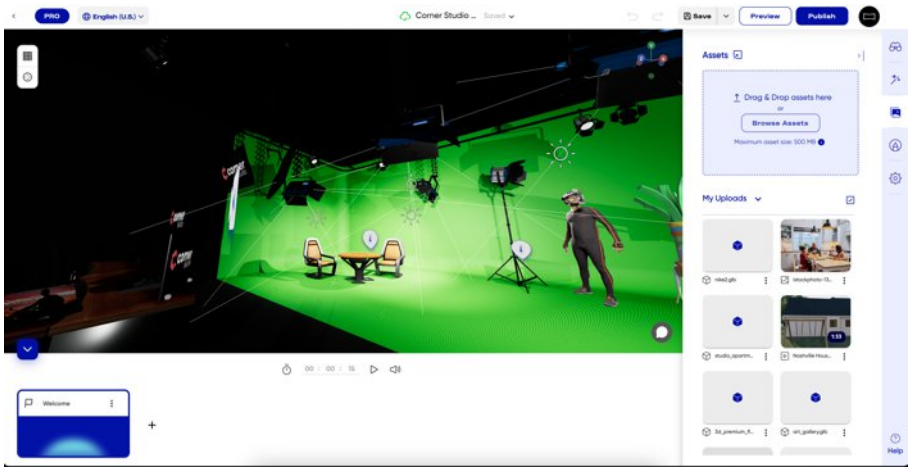


Figure 76 - XR Editor interface

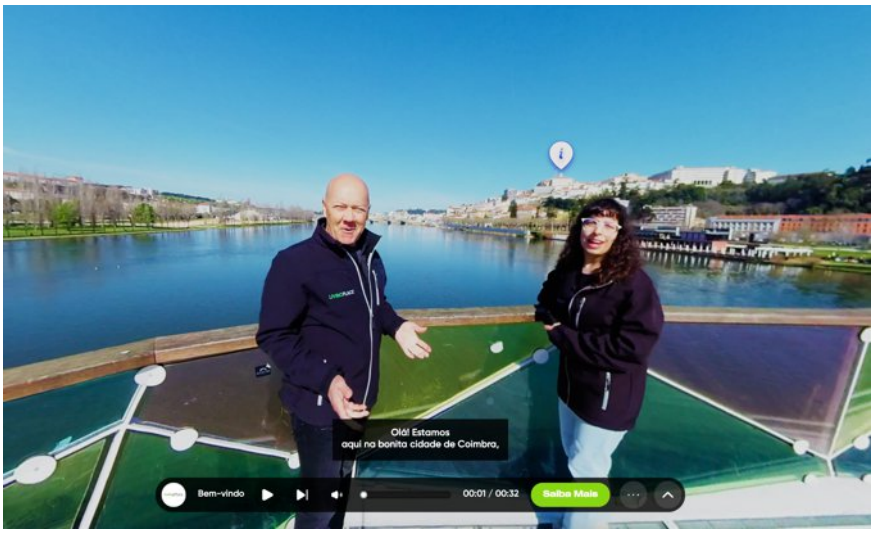


Figure 77 - cyango-story interface

Figure 78 - Portuguese TV use case showcase running on CHARITY

Figure 79 - AWE XR Lisbon Showcase

Figure 80 - Web Summit 2021 - Dotes Running on CHARITY showcase

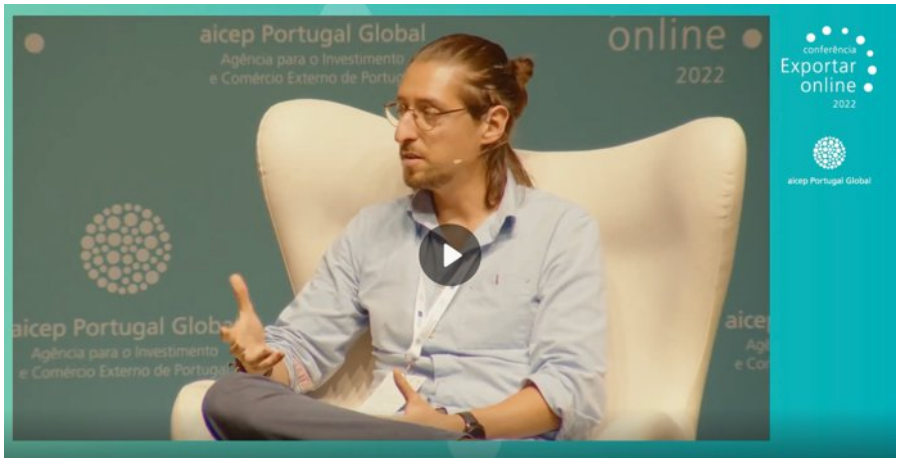


Figure 81 - Invited talk to present DOTES use case running on CHARITY

Figure 82 - XR Conference Showcase at University

Figure 83 - euCNC 2023

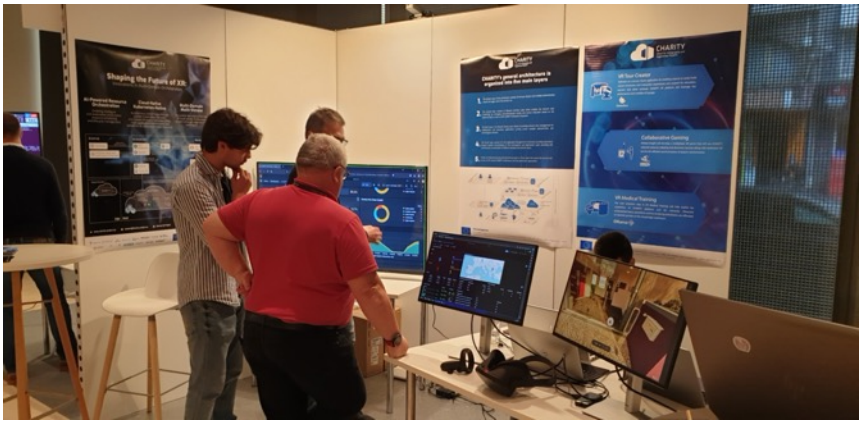


Figure 84 - euCNC 2024 Showcase

6.4 Collaborative Gaming Showcasing

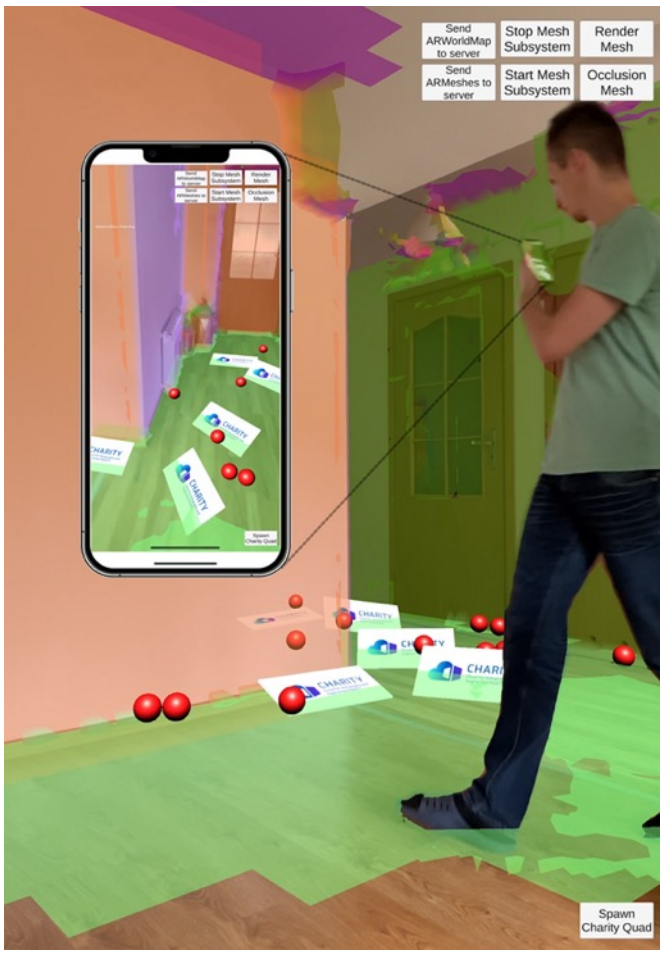


Figure 85 - Interaction with mixed reality in UC 3.1

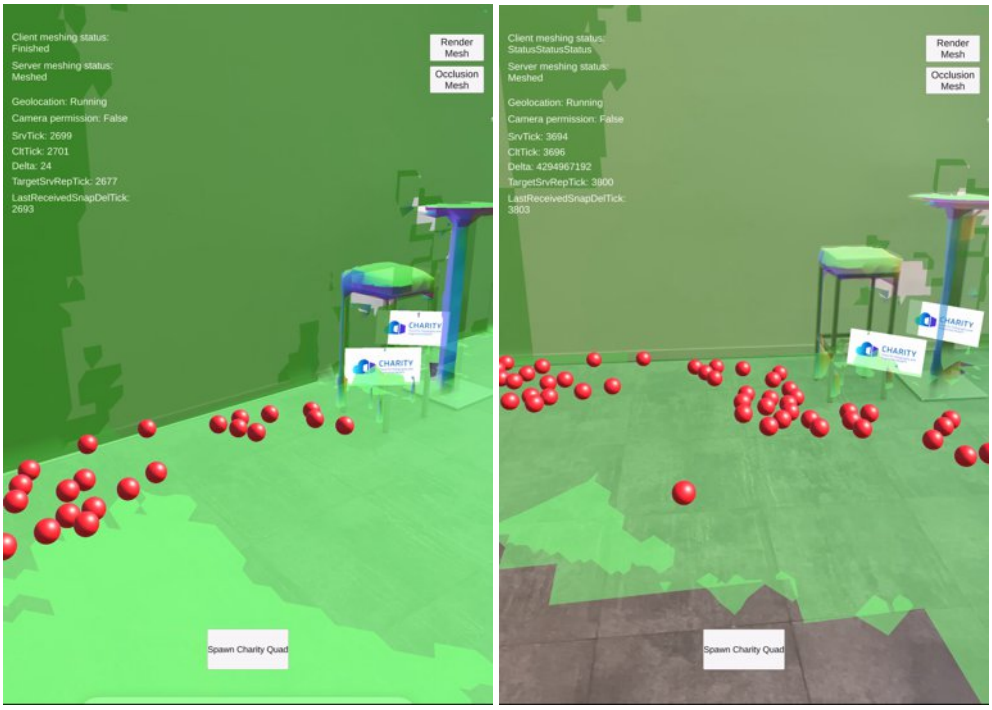


Figure 86 - Scanning the real space using 2 client devices

6.5 Manned-Unmanned Operation Trainer Showcasing

The Flight Simulator Use Case challenged Collins Aerospace to rethink, redesign and redevelop the traditional on-premises model for flight simulation. This, combined with adopting cloud native architecture principles to put forth a use case to apply the proposed solution for Task 3.3 (Dynamic Software Adaptation), proved to be a highly challenging task. The evolution of the current prototype has been monitored with interest within Collins Aerospace and drew inputs from a number of business unit groups across the company with a stand at numerous internal exhibition days held by the Collins Applied Research Centre in Ireland.

The separate research and development strands of work undertaken by UTRC in the use case began to coalesce and combine into a demonstrator in the final months of the project. One of the strands – that of real-time video upscaling at the edge using AI – was promoted at a national Irish conference on Artificial Intelligence while the move to Cloud Native was discussed in a public CHARITY webinar. Demonstration of integration with a commercial grade flight simulator has not been possible outside of the confines of Collins Aerospace due to the commercially sensitive nature of the flight simulation business.

Figure 87 - Promotion at national AI conference, in a public webinar, and during one of the research centre's internal open days

7 Conclusions

The document presents the outcomes of the final version of the validation and evaluation of components and services of the CHARITY platform prototype as well as the Use Cases in different testbeds environments. The experimentation involves the evaluation of individual subtopics which address core functionalities and aspects of the platform and are linked to the functional and non-functional requirements. The evaluation of each subtopic was undertaken by the partner responsible for developing the respective modules of the platform, services and use cases, the procedure and metrics were detailed upon which the analysis was conducted and reported in the present deliverable. Furthermore, based on the related KPIs the analysis comprised of different experiments. The outcomes of the evaluation are summarized as lessons learnt. In this final version of the deliverable use case UC1-3 Holographic Assistant was not evaluated, due to SRT leaving the consortium. All use cases were used to evaluate the deployment, monitoring and orchestration features of the CHARITY platform.

[end of document]