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

D3.1: Energy, data and computational-efficient mechanisms supporting dynamically adaptive and network-aware services (preliminary)

Version: v1.0

Deliverable type	R (Document, report)
Dissemination level	PU (Public)
Due date	31/12/2022
Submission date	22/12/2022
Lead editor	Massimiliano Corsini (CNR)
Authors	Antonios Makris (HUA), Konstantinos Tserpes (HUA), Theodoros Theodoropoulos (HUA), Michael McElligott (CAI), Tom Loven (PLEXUS), Laura Sande (PLEXUS), Yago Gonzalez Rozas (PLEXUS), Antonis Protopsaltis (ORAMA), Manos Kamarianakis (ORAMA), Enrico Zschau (SRT), Federico Ponchio (CNR), Pedro Sá (ONE), Joao Rodrigues (DOTES)
Reviewers	Luis Rosa, Patrizio Dazzi
Work package, Task	WP3
Keywords	XR enablers, storage system, software dynamic adaptation, rendering algorithms, data compression

Abstract

WP3 is the work package devoted to the research and development of strategies, mechanisms, and algorithms, for the efficient exploitation of available network and computational resources to enable sophisticated XR applications. Several aspects are investigated, like the intelligent management of data storage and access, and innovative strategies to adapt the Quality of Experience of the running application according to the available resources. Regarding the advancement of XR technologies, we investigated techniques to obtain more complex realistic VR simulation, technical solutions for rendering adaptation, novel algorithms for 3D point cloud compression and for the next-gen multi-user AR gaming experience, and we proposed solutions for the editing and streaming of immersive 360 video.

Document revision history

Version	Date	Description of change	List of contributor(s)
v0.1	04/04/22	Initial skeleton and first contributions	Massimiliano Corsini (CNR), Antonios Makris (HUA), Konstantinos Tserpes (HUA), Theodoros Theodoropoulos (HUA), Michael McElligott (CAI), Tom Loven (PLEXUS), Laura Sande (PLEXUS), Yago Gonzalez Rozas (PLEXUS), Antonis Protopsaltis (ORAMA), Manos Kamarianakis (ORAMA), Enrico Zschau (SRT), Federico Ponchio (CNR), Pedro Sá (ONE)
v0.2	20/06/22	Version for the internal reviewers	Luis Rosa (ONE), Patrizio Dazzi (CNR)
v0.3	29/06/22	Editorial check, sent to PO	Anja Köhler, Uwe Herzog (EURES)
v0.4	15/10/22	Updating of the draft	Massimiliano Corsini (CNR), Antonios Makris (HUA), Konstantinos Tserpes (HUA), Theodoros Theodoropoulos (HUA), Michael McElligott (CAI), Tom Loven (PLEXUS), Laura Sande (PLEXUS), Yago Gonzalez Rozas (PLEXUS), Antonis Protopsaltis (ORAMA), Manos Kamarianakis (ORAMA), Enrico Zschau (SRT), Federico Ponchio (CNR), Pedro Sá (ONE), Joao Rodrigues (DOTES)
v0.5	02/12/22	Version for the internal reviewers	Massimiliano Corsini (CNR)
V0.6	12/12/22	Version for the PO	Luis Rosa (ONE), Massimiliano Corsini (CNR)
V0.9	14/12/22	Final editorial check	Uwe Herzog (EURES)
v1.0	22/12/22	Adding heading 5.1, re-design of section 2 tables, refinement in sect. 5.4	Michael McElligott (CAI), Antonis Protopsaltis (ORAMA), Uwe Herzog (EURES)

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

Acknowledgement

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.

Executive Summary

The research and development activities in the WP3 will drive the advancement of complex and highly demanding (in terms of computation and/or bandwidth resources) XR applications. The systems and the algorithms delivered by WP3 will be integrated (WP4) into the CHARITY platform and in some of the Use Cases (UCs) of the CHARITY project. These ad-hoc technologies and novel algorithms regard different aspects of advanced XR applications.

A flexible monitoring framework which fills the needs of the different UCs of CHARITY is under development. This monitoring framework is based on the open-source Prometheus technology. According to the metrics identified, different types of exporters are under development to enable the different UCs to monitor their metrics of interest. The monitoring framework interacts with the orchestration system of the CHARITY platform.

AI-based technologies are under development to be included in the CHARITY Edge Storage (CHES). The CHES is a data management system for the intelligent management of data storage and data access. This system takes into account the high degree of heterogeneity that characterises the computational resources considered in the CHARITY project and it is lightweight so that it can also be used on edge devices with limited capabilities such as a Raspberry Pi. Preliminary experimental results, in terms of achieving the planned KPI, are really encouraging. CHES will be released as an open-source software under GPL 3 license.

XR applications are demanding in terms of computational and network resources, and the environmental circumstances may become sub-optimal during their running, for example, due to a reduction of bandwidth. In many of these cases, it is convenient to modify the behaviour of the running application so that the application itself adapts to the available resources instead of re-routing or deploying it. In CHARITY, a variant of the MAPE-K Loop [7] approach, based on micro-services, is proposed to perform an adaptation of XR applications at runtime. This novel solution has been carefully designed and some preliminary studies related to the flight simulator of the Collins Aerospace (UC3-2 Manned-Unmanned Operations Trainer Application) have been conducted.

Virtual Reality applications often require high realism in rendering and physical simulation. The UC2-1 VR Medical Training Application of CHARITY is one of these types of virtual reality applications. This VR UC is currently being optimized by exploiting multi-threading to make the rendering and the physics part even more efficient. Preliminary results are encouraging, for what concerns the physics simulation, while multi-threaded rendering in Unity has shown some limits of applicability.

The immersive applications, to reach high-quality levels of experience, require ultra-low latency and large bandwidth resources. To improve immersiveness of applications, we begin to integrate into some selected UCs, an adaptive rendering algorithm to reduce the rendering computation, and hence, the overall latency of the application.

Another important aspect of immersive applications is 360-degree video. The UC2-2 VR Tour Creator Application of CHARITY regards the advancement of a platform for the creation of virtual tours based on 360° video. In the next we describe the new features under and technological advancement under development of the Cyango Cloud Studio, that is the platform of UC2-2.

Specific data services to satisfy the needs of XR applications like the UC1-3 Holo Assistant and the UC3-1 Collaborative AR Gaming are also under development. Respectively, a point cloud codec allows the transmission from the cloud to the edge (the holographic display) of a huge amount of 3D points, and a geometry processing algorithm guarantees the continuous update between the real and the virtual gaming environment (called Mesh Merger). The Mesh Merger is in the first stage of development, while a first working prototype of the PC encoder/decoder will be available in short time.

In this deliverable, the research and the technical work related of the activities mentioned above is described and some experimental results are reported.

Table of Contents

List of Figures

List of Tables

Abbreviations

AR	Augmented Reality
CCU	Concurrent Users
CHES	CHARITY Edge Storage
COS	Container Orchestration Information
CPU	Central Processing Unit
CRD	Custom Resource Definition
CSI	Container Storage Information
DLF	(Kubernetes) Dynamic Lifecycle Framework
DLSP	Dynamic Software Product Line
DoW	Description of Work
GPU	Graphics Processing Unit
HMD	Head Mounted Display
ISP	Internet Service Provider
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
PC	Point Cloud
PVC	Persistent Volume Claim
QoE	Quality of Experience
QoS	Quality of Service
SPLE	Software Production Line Engineering
TCP	Transmission Control Protocol
WP	Work Package
UC	Use Case
UDP	User Datagram Protocol
VM	Virtual Machine
VR	Virtual Reality
XR	Extended Reality

1 Introduction

WP3 is devoted to the development of strategies, mechanisms, and algorithms that support both parts of the CHARITY Framework (described in the Deliverable D1.3 and D4.1) and the Use Cases (UCs) (described in D1.2 and D1.3). The UCs support is provided by developing specific data services that are used by some of the XR applications shown in the ambit 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. For example, the 3D point cloud encoder/decoder can be used by any XR application which needs to transfer a huge amount of 3D data points.

The R&D work conducted is presented in the following way: first, a brief introduction of the different activities is given, together with their mapping with the Tasks of the WP3. The relationships between the activities conducted and the rest of the project (other WPs/tasks) are also described. After this introduction, the R&D activities are described in detail in the subsequent sections. The activities description is organized by topic and does not follow the task subdivision.

1.1 Activities in a nutshell

The activities described in the next sections are:

- Monitoring framework
- CHARITY Edge Storage (CHES)
- Resource-aware Adaptation Mechanisms
- Transforming the flight simulator UC to a cloud-native XR application
- Rendering and physics simulation optimization for realistic VR applications
- Adaptive rendering for high QoE
- Point Cloud Encoding/Decoding
- Virtual tours through 360 video streaming platform
- Mesh Merger

The *monitoring* of the available network and computational resources plays a fundamental role for their assignment according to specific requests, i.e. for the orchestration, and for the applications performance management. Regarding performance, sometime, the applications should adapt their behaviour during their execution to guarantee a target QoE or reduce it in case of loss of resources. The monitoring is based on the open-source Prometheus framework. Such technology is configured and integrated to satisfy the CHARITY requirements. The monitoring activity is conducted in the ambit of the Task 3.1, the task committed to the efficient exploitation of computing resources, and in the ambit of the Task 3.3, that is about the dynamic adaptation mechanisms of the applications. The approach followed for monitoring and the architecture of the monitoring system is detailed in Section 2.

The *CHARITY Edge Storage (CHES)* is a solution for the optimized edge storage services to the CHARITY framework and its hosted applications. The goals of the CHES are ambitious; it should work on hardware with limited resources (e.g. a Raspberry Pi), and, at the same time, should provide reliable, robust, and fast access to the information. It is based on Lightweight Kubernetes (K3s), MinIO and Prometheus technologies. The CHES is developed in the ambit of the Task 3.2. The current status of the development is detailed in Section 3.

The *Resource-aware Adaptation Mechanisms* are designed following the MAPE K-Loop [7] approach. It consists in adapting the running applications according to the available resources by acting on applications' keypoints (e.g. changing the frame rate, changing the resolution). Such adaptation can be achieved by dynamically modifying the configuration of the application. In CHARITY, a variant of the standard MAPE K-Loop approach is proposed; a Mesh Service is used for the re-routing of micro-services to perform the application adaptation. This idea is explained in Section 4. This is the main activity of Task 3.3.

In CHARITY, we are also modifying the architecture of some UCs such that these XR applications become cloud-native. The case of UC3-1 Manned-unmanned Operations Training Application is particularly complex and requires a lot of effort. Such technical effort has been described in Section 5.1.

Many VR applications require both high-quality rendering and accurate physical simulation to provide a realistic virtual environment. One of the UC of CHARITY, UC2-1, regards VR simulation for medical training. The idea is to improve the performance of this VR medical simulation platform by employing *multi-threading to speed up rendering and physics simulation* (as detailed in Sections 5.2 and 5.3). The multi-threading exploitation of rendering and physics for the realistic simulation of virtual environments is part of the activities for the efficient exploitation of computational resources (Task 3.1).

VR immersive applications, to be comfortable, satisfying, and convincing, require low latency and high bandwidth. In CHARITY, we aim to integrate in two UCs, the UC2-1 VR Medical Training Simulator and the UC3-1 Manned-unmanned Operations Training Application, an adaptive rendering algorithm to reduce the computational burden and, consequently, the motion-to-photon latency. This activity is described in Section 5.4 .

The *Point Cloud Encoder/Decoder* is the main component of the UC1-3 Holo Assistant. The Holo Assistant must efficiently transmit a huge amount of 3D data from the cloud to the edge (the holographic display). This UC is described in detail in D1.2. The current status of the development of this innovative PC encoder/decoder is given in Section 5.5. This activity is conducted in the ambit of the Task 3.4, devoted to the development of an adaptive rendering algorithm and data compression/decompression for high demanding rendering applications.

Another activity of the Task 3.4 is the development of a *virtual tour platform* (UC2-2 VR Tour Creator Application) to create interactive VR experiences. This platform, called *Cyango*[2], supports 360 videos, panoramas, 3D models, standard images and videos and basic 3D meshes. The progress of the Cyango platform is described in Section 5.4.

The *Mesh Merger* is a data service built on a geometry processing algorithm which runs on the server to enable the UC3-1 Collaborative Game. This algorithm integrates the different pieces of geometry of the environment so that the game players can interact with a virtual environment that is continuously updated with the real one. For example, if a chair inside a room is moved during the game, and one gamer acquires this change through her smartphone, the Mesh Merger integrates this environment change in the virtual environment. The Mesh Merger data service is described in section 5.5. This activity is also conducted in the context of Task 3.4.

1.2 Relationships between the CHARITY Framework and the WP3 Tasks

An overview of the CHARITY architecture with the WPs / Tasks is given in Figure 1. It is composed of three planes: i) the Domain Specific XR Service Monitoring and Reaction Plane, ii) the XR Service E2E Conducting Plane, and the iii) XR Service Deployment Plane.

[2] https://www.cyango.com

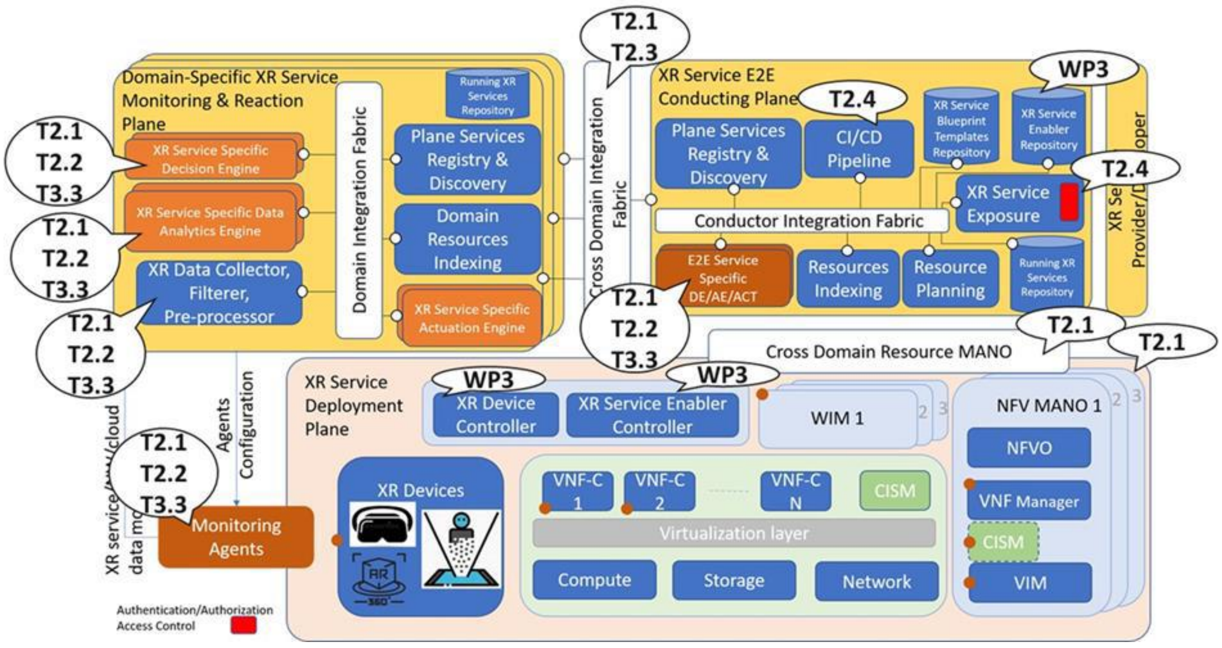

T2.1
T2.3
Domain-Specific XR Service Monitoring & Reaction Plane
Running XR Services Repository
T2.1 T2.2 T3.3
XR Service Specific Decision Engine
Plane Services Registry & Discovery
Domain Integration Fabric
XR Service Specific Data Analytics Engine
Domain Resources Indexing
XR Data Collector, Filterer, Pre-processor
XR Service Specific Actuation Engine
Cross Domain Integration Fabric
XR Service E2E Conducting Plane
T2.4
CI/CD Pipeline
XR Service Blueprint Templates Repository
XR Service Enabler Repository
WP3
XR Service Exposure
Conductor Integration Fabric
E2E Service Specific DE/AE/ACT
Resources Indexing
Resource Planning
Running XR Services Repository
XR Service Provider/Developer
T2.1 T2.2 T3.3
Cross Domain Resource MANO
XR Service Deployment Plane
XR Device Controller
XR Service Enabler Controller
WIM 1
NFV MANO 1
NFVO
VNF Manager
CISM
VIM
XR Devices
VNF-C 1
VNF-C 2
VNF-C N
Virtualization layer
Compute
Storage
Network
Agents Configuration
Monitoring Agents
XR service/app/cloud data mo
Authentication/Authorization Access Control

Figure 1: CHARITY Architecture and project WPs/Tasks mapping.

The *XR Service Deployment Plane* consists of the infrastructure where the XR services run. It thus hosts the different Virtual Network Functions (VNFs) that compose the different XR services. The main responsibility of this plane is to manage the computational, network, and storage resources of the infrastructure. Two of the main components of the XR Service Deployment Plane are the *XR Device Controller* and the *XR Service Enabler Controller*. The XR Device Controller is in charge to control the XR devices. This allows to separate the data plane from the control plane. Similarly, the XR Service Enabler Controller is in charge of control specific XR services instead of devices. The XR Device Controller and the XR Service Enabler Controller are developed as part of the research activities of the WP3.

The *Domain-specific Monitoring and Reaction Plane* is responsible for monitoring the service inside a technological or administrative domain. It keeps track of the resource usage and of the XR services running in the domain, it makes decisions according to the monitored data, and carried out the actuation which are specific to each running XR service, without resorting to the E2E conducting plane.

The *XR Service E2E Conducting Plane* is responsible for creating the different sub-slices inside each domain and for monitoring the E2E KPIs of the XR services. It is also responsible for the lifecycle management of the XR services, and it can shift services between the different domains when necessary.

A detailed description of the different components of the CHARITY architecture can be found in the Deliverable D1.3.

The WP3 tasks are connected to the CHARITY architecture as described in the following (see Figure 1, Table 1 and Table 2):

- **Task 3.1 Efficient exploitation of CPUs, GPUs and FPGAs on edge devices.** This task is focused on providing efficient solutions for exploiting computational resources to support the project needs. The main activities are related to the monitoring and the resource indexing, as well as technological and algorithmic solutions for enabling the exploitation of the different and heterogeneous computational resources belonging to CHARITY. The monitoring framework is strictly connected with the Task 2.1 and 2.2 and also with the Task 3.3.

- **Task 3.2 Efficient storage and caching for AR, VR and Holographic applications.** In the ambit of this task several components for the realization of a distributed edge storage framework spread across heterogeneous edge and cloud nodes, with intelligent data management, high quality performance (QoE), and high-security levels are under development. These components, parts of the XR Service Deployment Plane, are: the CHarity Edge Storage (CHES) which is a distributed hybrid storage component and the CHES Registry component that realizes a localized Docker registry in order to support the faster application deploying and limit the network flooding caused by large image downloads during deployment. In addition, two mechanisms will be investigated, the chaRity OnLine prOactive cachIng (ROLOI) mechanism which is an online proactive caching scheme based on deep neural network models, and the PRoActive ComponenT Image plaCement in edge computing Environments (PRACTICE) which ensures that application images are delivered within a given amount of time to any resource composing the federation of edge devices.

- **Task 3.3 Network and infrastructure awareness for efficient exploitation of resources:** is to explore the Dynamic Software Production Line (DSPL) paradigm to adapt XR services dynamically and automatically to network and environment changes. Task 3.3 will also design and develop specific Monitoring, Analytics, Decision and Actuation Engines for both domain and cross-domain levels. This work is related to the realization of the XR Service Specific Analytics Engine, the XR Service Specific Decision Engine, and the XR Service Specific Actuation Engine components, which are parts of the Domain-specific Monitoring and Reaction Plane as well as of the XR Service E2E Conducting Plane.

- **Task 3.4 Adaptive rendering and contextualized data compression / decompression:** The R&D activities conducted in this task are related to the development of the algorithms that will integrated in data services for XR applications such as the *Point Cloud Encoder/Decoder (PC E/D)*, used by the UC 1-3 Holo Assistant, or the *Mesh Merger*, developed to support the UC3-1 Collaborative Gaming.

To make this document self-containing and more readable, we report below two tables adapted from D4.1. Table 1 contains the name of the component of the CHARITY Framework together with the name of the tasks related to its development. and Table 2, which contains the name of the algorithms/mechanisms that is at the base of some specific plane/component, and the task within it is studied and developed.

Table 1: CHARITY Component List

Component Name	Architectural Layer	Tasks
Monitoring Agents	Monitoring & Reaction Plane	T3.1, T3.3
XR Service Specific Analytics Engine	Monitoring & Reaction Plane	T2.1, T2.2, T3.3
XR Service Specific Decision Engine	Monitoring & Reaction Plane	T2.1, T2.2, T3.3
XR Service Specific Actuation Engine	Monitoring & Reaction Plane	T2.1, T2.2, T3.3
Running XR Services Repository	Monitoring & Reaction Plane	T2.1, WP3, WP4
Plane Services Registry & Discovery	Monitoring & Reaction Plane	T2.1, WP3, WP4
E2E Service Specific DE/AE/ACT	XR Service E2E Conducting Plane	T2.1, T2.2, T3.3
XR Service Enabler Repository	XR Service E2E Conducting Plane	T2.4, WP3
Running XR Services Repository	XR Service E2E Conducting Plane	T2.1, WP3, WP4
Resource Planning	XR Service E2E Conducting Plane	T3.1
Resource Indexing	XR Service E2E Conducting Plane	T3.1
XR Device Controller	XR Service Deployment Plane	WP3
XR Service Enabler Controller	XR Service Deployment Plane	WP3

Table 2: CHARITY proposed mechanisms and algorithms

Component Name	Component Description	Architectural Layer	Tasks
Prometheus and Monitoring agents	Resource monitoring tool Agent to facilitate VNF monitoring	Monitoring Agents / Monitoring & Reaction Plane	T2.2, T3.1, T3.3
Adaptative Network Traffic Mechanism	Mechanism to dynamically route network traffic accordingly to infrastructure conditions	DE/AE/ACT Monitoring & Reaction Plane / E2E Conducting Plane	T3.3
XR Service Enabler Repository	Repositories for container images, VM images and metadata	XR Service Enabler Repository / XR Service E2E Conducting Plane	T2.4, WP3
CHES (CHARITY Edge Storage)	A distributed hybrid storage component spread across heterogeneous edge and cloud nodes with intelligent decisions on data placement, data caching and considerations on performance (QoE) and security	XR Service Deployment Plane	T3.2
ROLOI (chaRity OnLine prOactive cachIng)	Online proactive caching scheme based on deep neural network models to predict time-series content requests and update edge caching accordingly	XR Service Deployment Plane	T3.2
PRACTICE (PRoActive ComponenT Image plaCement in edge computing Environments)	A component that ensures that application images are delivered within a given amount of time to any resource composing the federation of edge devices	XR Service Deployment Plane	T3.2
3D Point cloud encoder/decoder	Data service component to compress/decompress point cloud for efficient transmission	XR Service Deployment Plane	T3.4
Decentralised storage / network performance	Measuring the performance of DHT-based decentralised storage platforms such as IPFS and pub-sub based federation networks.	XR Service Deployment Plane	T3.2

For the complete list of components and algorithms/mechanisms, refer to the Appendix A, B, and C of the Deliverable D4.1. The corresponding tables of D4.1 report also provide additional information for each component/algorithm, like the name of the partners involved in the development.

Note that Table 2 reports the algorithms/mechanisms that can be mapped on the CHARITY architecture. The Mesh Merger, depending on its final implementation, could be integrated into the game server of the UC3-1 or become an additional component.

2 Monitoring

2.1 Monitoring approach

2.1.1 Monitoring agent

Prometheus, Grafana and Thanos are the open-source tools at the base of the CHARITY's monitoring platform: monitoring, alerting, data storage and visualization. However, a monitoring element is needed to achieve a multi-cloud platform focused on prevention and reactivity. This monitoring agent converts individual static configurations into a dynamic platform and needs to be reporting quickly to the XR Data Collector and communicating with the orchestrator. The following table provides an overview of these communications.

Table 3: Monitoring agent communications

Source	Destiny	Description
Orchestrator	Monitoring agent	Change configuration, alerting
Monitoring agent	Prometheus server	Change configuration, alerting
Monitoring agent	Thanos	Change configuration
Orchestrator	Monitoring agent	Monitor new element
Monitoring agent	Prometheus server	Monitor new element
Monitoring agent	Thanos	Collect data from new element
Orchestrator	Monitoring agent	Stop monitoring an element but don't delete Thanos stored data
Monitoring agent	Prometheus server	Stop monitoring an element
Orchestrator	Monitoring agent	Stop monitoring an element and delete Thanos stored data
Monitoring agent	Thanos	Stop monitoring an element
XR Data Collector	Monitoring agent	Request Data
Monitoring agent	XR Data Collector	Send data

The monitoring agent communicates through HTTP to receive configuration update orders based on service migration and to send stored performance data to the Reaction Plane to predict the immediate needs of CHARITY elements and use case elements. The types of orders will be encoded by an integer numeric value and will include the parameters necessary to carry out that request: IP of the Kubernetes service that represents the element, element id, metric and/or metric value. Based on the data, the monitoring agent will generate the necessary commands according to the language set by the target elements and launch the request to the specific element.

2.1.2 Monitoring architecture

The monitoring architecture, presented in Deliverable D2.1 (Figure 2), responds to the preliminary requirements of XR applications to be developed on a multi-cloud platform, reduce complexity, focusing on prevention and reactivity in ecosystems with heterogeneity of technologies. To translate these formal needs into functional values, it is necessary to identify the elements of the architecture of each UC, the links between them, and the needs of each of the developing partners.

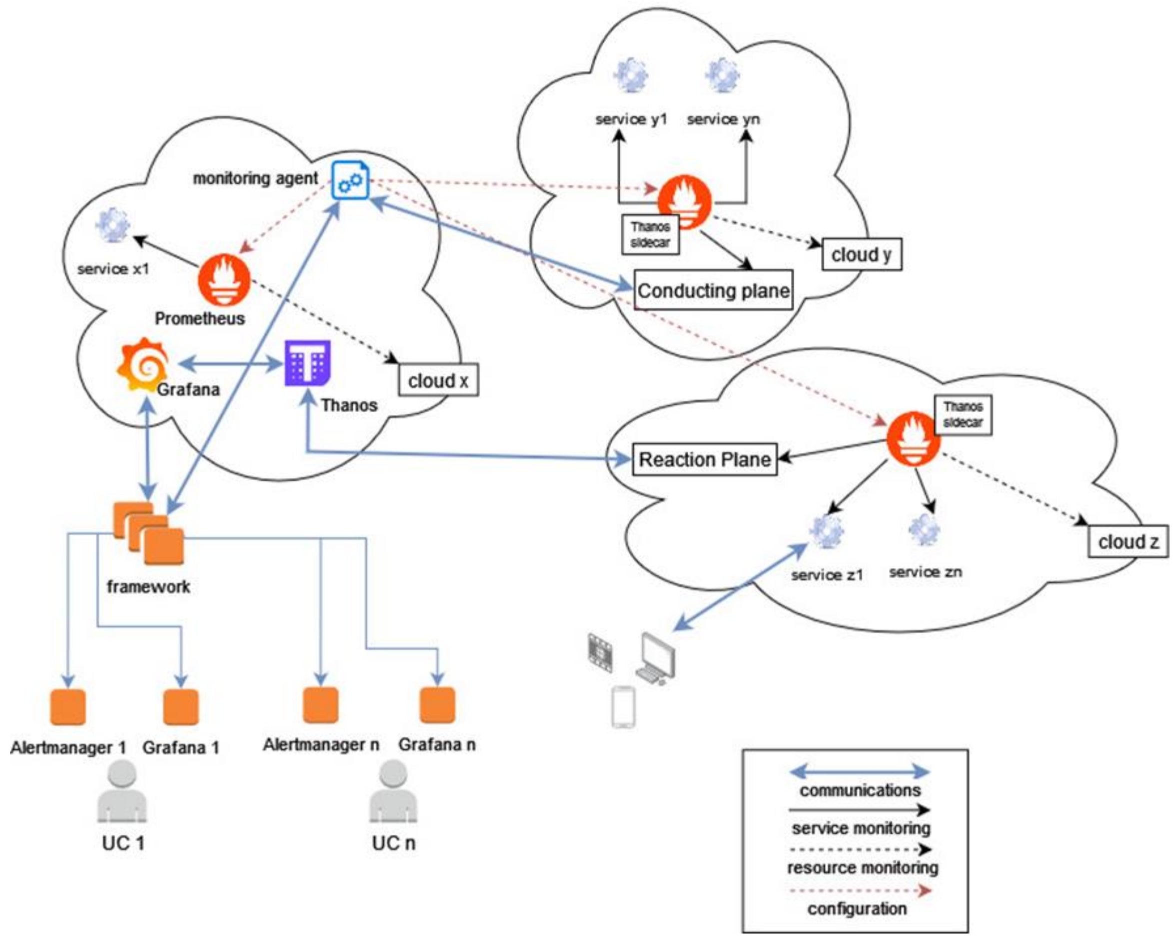


Figure 2: Monitoring Architecture defined in D2.1

The analysis of each UC and the KPIs collected in deliverable D1.3 allowed to define the set of values to be monitored according to the needs of the UC owners. These metrics and their formats will be consulted and processed in the different CHARITY architecture planes. Hence, it is necessary to establish common values from the outset to advance in the development of other CHARITY elements that depend on the monitoring system, as seen in Table 4.

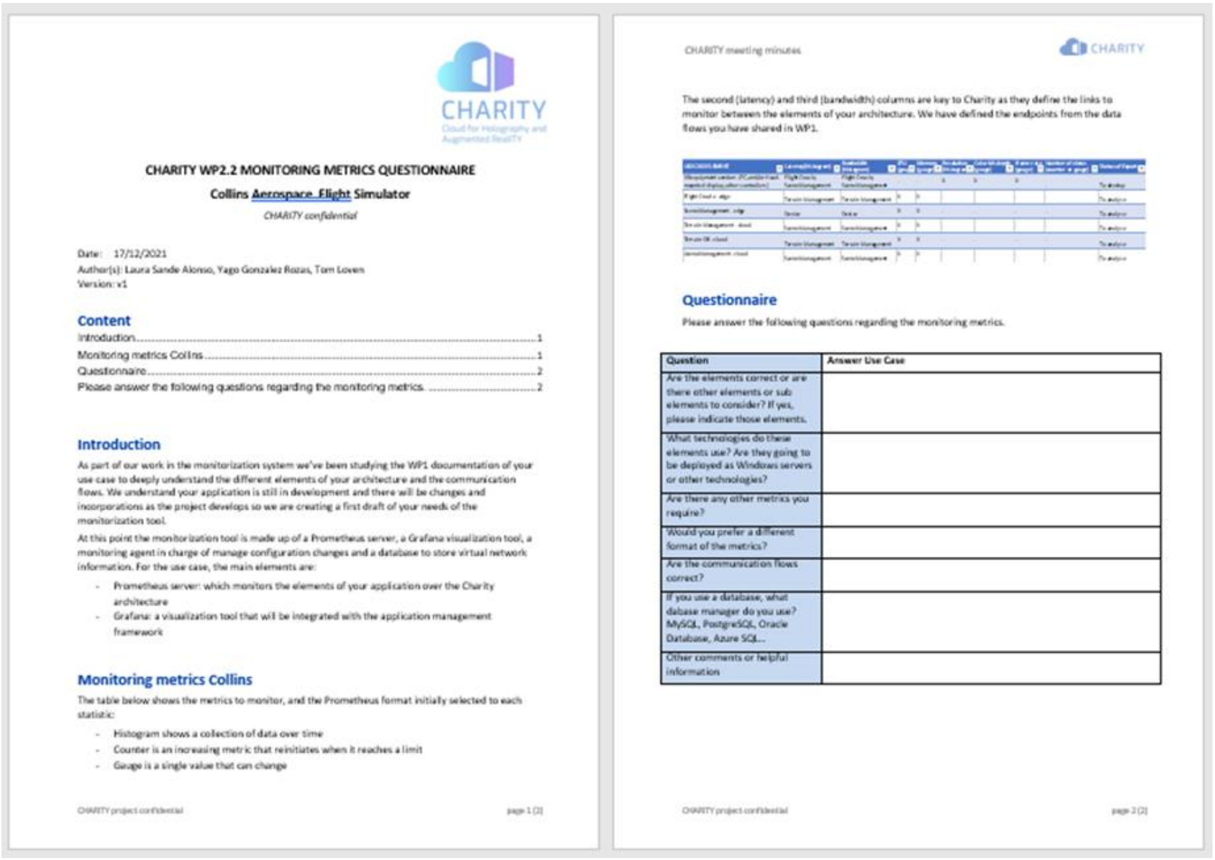
Table 4: Metric definition

METRICS	DEFINITION	OUTPUT NAME	OUTPUT UNITS	FORMAT	EXAMPLE
Latency	Time it takes for a request to reach the destination and return, including the operation time of the destination to respond to the request	latency	miliseconds -ms	three decimals	125,123
RTT	Round trip time. Time it takes for a request to reach the destination and return. It doesn't include the operation time of the destination to respond to the request	rtt	miliseconds -ms	three decimals	125,123
Bandwidth	Maximum capacity that can be transmitted over a link	bw	Mbps	three decimals	1000.000
CPU	Percentage of used CPU	cpu	percentage	positive integer	50
GPU	Percentage of used GPU	gpu	percentage	positive integer	50
Memory	Percentage of used memory	memory	percentage	positive integer	50
Resolution	Number of pixels a screen is capable of displaying	resolution	megapixel	three decimals	4,096
Color bit depth	Number of bits needed to represent the color of a pixel	colorbitdepth	bits per pixel	positive integer	24
Frame-rate	Frequency at which a device displays images	framerate	frames per second - fps	positive integer	240
Petitions per second	Number of requests per second	petitionspersecond	requests per second	positive integer	1000

A preliminary collection of the monitoring requirements of the use cases, from the performance of each of the elements to the performance of the links between the different elements. To open communications with each UC and find out their preliminary needs, ad-hoc surveys were created. These surveys contained a table of metrics that affected the case and a series of questions with which to delve into the types of data they need and the technologies of the elements they are developing (see an example in Figure 3).

The results of these surveys allowed to convert the requirements into a list of values to be monitored, with already defined formats. This allows also to design the exporters that will expose the data collected by the Prometheus server, the core of the monitoring system. The Prometheus server pulls metrics from elements monitored through the HTTP endpoint each one uses to communicate. To expose these metrics, the elements use *exporters*, which collect the monitoring information, convert it to the format used by Prometheus and expose it to the outside.

The extensive use of Prometheus implies the existence of a community that maintains numerous exporters developed by third parties, which are already identified in the tables in the following sections focused on each use case. However, XR applications involve the appearance of new elements that require the development of custom exporters, for which Prometheus offers detailed documentation and compatibility with the most common programming languages. Therefore, the collection of information, made through a questionnaire (as in Figure 3), needs to be made prior to the development of the monitoring system is a key step for the following phases, since it allows efforts to be focused on understanding the elements, their languages and the need or not to develop custom exporters, that can be similar between different use cases.


CHARITY
Cloud for Holography and Augmented RealiTY

CHARITY WP2.2 MONITORING METRICS QUESTIONNAIRE

Collins Aerospace Flight Simulator

CHARITY confidential

Date: 17/12/2021
Author(s): Laura Sande Alonso, Yago Gonzalez Rozas, Tom Loven
Version: v1

Content

Introduction 1
Monitoring metrics Collins 1
Questionnaire 2
Please answer the following questions regarding the monitoring metrics. 2

Introduction

As part of our work in the monitorization system we've been studying the WP1 documentation of your use case to deeply understand the different elements of your architecture and the communication flows. We understand your application is still in development and there will be changes and incorporations as the project develops so we are creating a first draft of your needs of the monitorization tool.

At this point the monitorization tool is made up of a Prometheus server, a Grafana visualization tool, a monitoring agent in charge of manage configuration changes and a database to store virtual network information. For the use case, the main elements are:

- Prometheus server: which monitors the elements of your application over the Charity architecture
- Grafana: a visualization tool that will be integrated with the application management framework

Monitoring metrics Collins

The table below shows the metrics to monitor, and the Prometheus format initially selected to each statistic:

- Histogram shows a collection of data over time
- Counter is an increasing metric that reinitiates when it reaches a limit
- Gauge is a single value that can change

CHARITY project confidential page 1 (2)

CHARITY meeting minutes CHARITY

The second (latency) and third (bandwidth) columns are key to Charity as they define the links to monitor between the elements of your architecture. We have defined the endpoints from the data flows you have shared in WP1.

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	Flight Dynamics SceneManagement	Flight Dynamics SceneManagement			x	x	x		To analyse
[illegible]	Terrain Management	Terrain Management	x	x					To analyse
[illegible]	Device	Device	x	x					To analyse
[illegible]	SceneManagement	SceneManagement	x	x					To analyse
[illegible]	Terrain Management	Terrain Management	x	x					To analyse
[illegible]	SceneManagement	SceneManagement	x	x					To analyse

Questionnaire

Please answer the following questions regarding the monitoring metrics.

Question	Answer Use Case
Are the elements correct or are there other elements or sub elements to consider? If yes, please indicate those elements.	
What technologies do these elements use? Are they going to be deployed as Windows servers or other technologies?	
Are there any other metrics you require?	
Would you prefer a different format of the metrics?	
Are the communication flows correct?	
If you use a database, what dabase manager do you use? MySQL, PostgreSQL, Oracle Database, Azure SQL...	
Other comments or helpful information	

CHARITY project confidential page 2 (2)

Figure 3: Questionnaire for Collins Aerospace

Prometheus allows the use of four metric formats, two of them for individual values and the other two for storing a set of values during a certain period. *Counter* is an integer value that is incremented by one or reset to zero, while *gauge* allows the numeric value to increment and decrement. *Histogram* allows to collect values between certain margins over a period of time to later perform statistical analysis. The use of *summary* is similar to that of histogram, with the difference that it does not require a bucket definition, so it allows obtaining frequencies of more adjusted values than those of a histogram. In the questionnaires made to the UCs, these four possibilities were offered for the values to be monitored and they were asked to choose the formats according to the needs of each of their elements. In the following tables they will be defined as Counter -C-, Gauge -G-, Histogram -H- and Summary -S-.

The information discussed in this section serves as background for the following sections, which include the tables with the monitoring needs of both use cases and CHARITY own architecture, compiling metrics, formats and the existence of exporters already developed that expose the data.

2.2 UC1-1 - UC1-2: HOLO 3D - Holographic concert and holographic meeting

The preliminary level of development of the use cases is different for each of the applications, so not all of their microservices are already defined or implemented. In the case of the holographic systems for concerts and meetings devised by Holo3D, we find key metrics related to image quality, as well as the performance of communications to eliminate delays and offer a real-time experience.

Table 5: UC1-1 and UC1-2 - elements and metrics

USE CASE ELEMENT	Links	Latency	RTT	Band-width	CPU	GPU	Memory	Reso-lution	Color bit depth	Frame rate	HTTP Endpoint Exposer	Exporter	Endpoint Status
Musician (PC with camera, microphone)	Charity edge	HG	HG	HG	-	-	-	H	G	G	CUSTOM	-	To develop
Client (person watching the hologram on a holographic display)	Charity edge	HG	HG	HG	-	-	-	H	G	G	CUSTOM	-	To develop
Windows server	X	HG	HG	HG	G	-	G	-	-	-	EXPORTER	Windows server	Ready
Speaker (PC with camera and microphone)	Charity edge	HG	HG	HG	-	-	-	H	G	G	CUSTOM	-	To develop
Client (person watching the hologram on a holographic display)	Charity edge	HG	HG	HG	-	-	-	H	G	G	CUSTOM	-	To develop
Windows server	X	HG	HG	HG	G	-	G	-	-	-	EXPORTER	Windows server	Ready

2.3 UC1-3: SRT - Holographic assistant

The elements of the SRT holographic assistant are developed on Windows servers, which already have existing exporters to expose data in Prometheus format. The only custom exporter to create is the one that involves the end user of the application. The creation of this type of exporter is common to all use cases, since Prometheus cannot monitor screens, virtual reality headsets or cockpits. Its monitoring will be carried out on another element of the architecture of the use case that communicates with these final elements.

Table 6: UC1-3 - elements and metrics

USE CASE ELEMENT	Links	Latency	RTT	Band-width	CPU	GPU	Memory	Reso-lution	Color bit depth	Frame rate	HTTP Endpoint Exposer	Exporter	Endpoint Status
PC with holographic 3D device and Eyetracker	Charity edge	HG	HG	HG	-	-	-	H	G	G	CUSTOM	-	To develop
Windows server	X	HG	HG	HG	G	-	G	-	-	-	EXPORTER	Windows server	Ready
SRT_SW_CLIENT	SRT_SW_CONTENT	HG	HG	HG	G	-	G	H	G	G	EXPORTER	Windows server	Ready
SRT_SW_CONTENT	SRT_SW_BEHAVIOUR, SRT_SW_PCGEN	HG	HG	HG	G	G	G	-	-	-	EXPORTER	Windows server	Ready
SRT_SW_PCGEN	CHARITY_SW_PCENC	HG	HG	HG	G	G	G	-	-	-	EXPORTER	Windows server	Ready
CHARITY_SW_PCENC	SRT_SW_CLIENT	HG	HG	HG	G	G	G	-	-	-	EXPORTER	Windows server	Ready
SRT_SW_BEHAVIOUR	Google API	HG	HG	HG	G	-	G	-	-	-	EXPORTER	Windows server	Ready

2.4 UC2-1: ORAMA - Medical training

Surgical learning through extended reality implies high synchronization between all the participants in the session to accurately simulate the collaborative work that takes place in an operating room between the end users of the application and the responses of the virtual elements to collisions with users.

Table 7: UC2-1 - elements and metrics

USE CASE ELEMENT	Links	Latency	RTT	Band-width	CPU	GPU	Memory	Reso-lution	Color bit depth	Frame rate	HTTP Endpoint Exposer	Exporter	Endpoint Status
VR equipment vendors (PC, and/or Head-mounted display, other controllers)	Charity edge	HG	HG	HG	-	-	-	H	G	G	CUSTOM	-	To develop
Signalling Service	LSpart_1	HG	HG	HG	G	G	G	-	-	-	CUSTOM	-	To develop
LSpart_1	LSpart_2	HG	HG	HG	G	G	G	-	-	-	EXPORTER	Windows server	Ready
LSpart_2	LSpart_1	HG	HG	HG	G	G	G	-	-	-	EXPORTER	Windows server	Ready
Relay server	LSpart_1	HG	HG	HG	G	-	G	-	-	-	EXPORTER	Windows server	Ready
Lspart_1 - Controller interface	?	HG	HG	HG							EXPORTER	Windows server	Ready

2.5 UC2-2: DOTES - Virtual tours

Virtual tour applications are widely popular; however, they are still far from providing a realistic immersive user experience as they don’t focus on the limitations that the network imposes on application performance. To achieve the quality of the image and interaction with the scenarios that DOTES plans with its application, it’s essential that the communication speeds of the network and the processing of the elements adjust to their maximum performance.

Table 8: UC2-2 - elements and metrics

USE CASE ELEMENT	Links	Latency	RTT	Band-width	CPU	GPU	Memory	Reso-lution	Color bit depth	Frame rate	HTTP Endpoint Exposer	Exporter	Endpoint Status
Cyango Story - front-end	Charity edge	HG	HG	HG	-	G	-	H	G	G	CUSTOM	-	To develop
Cyango Cloud Editor - front-end	Charity edge	HG	HG	HG	-	G	-	H	G	G	CUSTOM	-	To develop
Charity media converter	Cyango API	HG	HG	HG	G	G	G	-	-	-	CUSTOM	-	To develop
Cyango API	3D engine	HG	HG	HG	G	-	G	-	-	-	CUSTOM	-	To develop
File Hosting	3D engine	HG	HG	HG	G	-	G	-	-	-	CUSTOM	-	To develop
3D engine	Cyango front-end	HG	HG	HG	G	G	G	-	-	-	CUSTOM	-	To develop
Image Engine		HG	HG	HG		G					CUSTOM	-	To develop
Video Engine (replace by charity media converter)		HG	HG	HG		G					CUSTOM	-	To develop
Livestream service		HG	HG	HG		G					CUSTOM	-	To develop
Database - Mongo DB		HG	HG	HG		-					EXPORTER	mongoDB	Ready
Transcribe Service		HG	HG	HG		-					CUSTOM	-	To develop

2.6 UC3-1: ORBK - Mixed reality

The extended reality application devised by ORBK focuses on the user's interaction with the virtualized scenario, the virtual objects introduced and that all this happens with the minimum delay between all the users of that application session. The mesh collider that is being developed in the CHARITY project is key to the performance between real image and virtualized elements, and to achieve the KPIs of the next generation XR applications, a proactive adaptive architecture like CHARITY is needed.

Table 9: UC3-1 - elements and metrics

USE CASE ELEMENT	Links	Latency	RTT	Band-width	CPU	GPU	Memory	Reso-lution	Color bit depth	Frame rate	HTTP Endpoint Exposer	Exporter	Endpoint Status
Game client	Game Server	HG	HG	HG	-	-	-	H	G	G	CUSTOM	-	To develop
Game Server	Game client, Mesh collider, Game Servers Status DB	HG	HG	HG	G	-	G	-	-	-	CUSTOM	-	To develop
Game Servers Status DB	Game Server	HG	HG	HG	G	-	G	-	-	-	EXPORTER	Cloud Watch	Ready
Charity Mesh collider	Game Server	HG	HG	HG	G	G	G	-	-	-	CUSTOM	-	To develop
Mesh Merging Service by CNR	Game Server	HG	HG	HG	-	G	-	-	-	-	CUSTOM	-	To develop

2.7 UC3-2: Collins Aerospace (CAI) - Flight simulator

To date, the simulation of high-speed scenarios has been limiting in terms of collaborative applications due to the difficulties of synchronization between users and the performance of the different microservices in charge of predicting the images to be displayed in the participants. In the case of Collins Aerospace, an edge architecture is proposed to reduce interaction times and control over the requests received by each element to show the highest image quality to always maintain synchronization between its users.

Table 10: UC3-2 - elements and metrics

USE CASE ELEMENT	Links	Latency	RTT	Band-width	CPU	GPU	Memory	Reso-lution	Color bit depth	Frame rate	HTTP Endpoint Exposer	Exporter	Endpoint Status
Cockpit (flight stick, thrustor, pedals)	Flight Oracle, Scene Management	HG	HG	HG	-	-	-	H	G	G	CUSTOM	-	To develop
Flight Oracle - edge	Terrain Management	HG	HG	HG	G	-	G	-	-	-	CUSTOM	-	To develop
Scene Management - edge	Device	HG	HG	HG	G	G	G	-	-	-	CUSTOM	-	To develop
Terrain Management - cloud	Scene Management	HG	HG	HG	G	G	G	-	-	-	CUSTOM	-	To develop
Terrain DB - cloud	Terrain Management, Image Generator	HG	HG	HG	G	-	G	-	-	-	EXPORTER	PostgreSQL	Ready
Arena Management - cloud	Scene Management	HG	HG	HG	G	-	G	-	-	-	CUSTOM	-	To develop
Flight Dynamics	Flight Oracle, Cockpit, View Builder	HG	HG	HG	G	G	G	-	-	-	CUSTOM	-	To develop
Image Generator	Flight Oracle, Terrain DB	HG	HG	HG	G	G	G	-	-	-	CUSTOM	-	To develop
Frame Caché	View Builder, Resolution Upscaler, Image Generator	HG	HG	HG	G	G	G	-	-	-	EXPORTER	Redis	Ready
View Builder	Flight Dynamics, Frame caché, WARP, web RTC Client	HG	HG	HG	G	G	G	-	-	-	CUSTOM	-	To develop
WARP	View Builder	HG	HG	HG	G	G	G	-	-	-	CUSTOM	-	To develop
web RTC Client	PC-HMD	HG	HG	HG	G	G	G	-	-	-	CUSTOM	-	To develop
PC, HMD	webRTC	HG	HG	HG	G	-	G	H	-	G	CUSTOM	-	To develop
Resolution Upscaler	Frame Caché	HG	HG	HG	G	G	G	-	-	-	CUSTOM	-	To develop

2.8 Relationship with the CHARITY Architecture

The adaptive scheme devised by CHARITY implies the migration of elements to offer maximum performance in services and applications, something that does not only affect the deployed applications but also the elements of the CHARITY architecture that work with them and resources available in the different cloud domains. The monitoring is a key part between dynamically adaptive network-aware services and efficient exploitation of resources being in continuous communication with the Monitoring & Reaction Plane. This plane is in charge of making proactive decisions, avoid delays and improve the quality of the next generation XR applications. CHARITY architecture performance is defined by use cases extreme KPIs, therefore, the latencies and bandwidths required between use case microservices must be directly proportional in the elements of the CHARITY architecture to ensure adaptability to the performance required by the applications. The continuous monitoring of elements and the network that connects them is what allows us to anticipate performance failures that will affect the gaming experience, so monitoring, prediction and migration are the key cycle in the CHARITY project after the initial deployment of the applications. This is only possible with the continuous performance analysis of all the components and the available resources for the modification of the deployed architecture in order to achieve the best possible performance.

The elements of the CHARITY architecture that are under development will require the creation of custom exporters adapted to the work of the developers, since they are new services that are not based on existing solutions. These exporters will be developed in the languages of the services themselves and will be in charge of adapting formats to the metrics managed by Prometheus and offering an exposure point that will be consulted by the Prometheus monitoring server, as listed in Table 11 and Table 12. The initiation of the development of these exporters is planned for the coming period.

Table 11: Architecture - elements and metrics monitoring and reaction plane

USE CASE/PARTNER	PLANE	USE CASE ELEMENT	Latency /latency ms	RTT /rtt ms	Bandwidth /bw Mbps	CPU /cpu %	GPU /gpu %	Memory /memory %	Resolution /resolution Megapixel	Petitions per second /petitionspersecond requests per second	HTTP Endpoint Exposer	Other partners incolved
PLEXUS	Monitoring & Reaction Plane	Monitoring Agents	HG	HG	HG	G	-	G	-	HG	CUSTOM	TID
??	Monitoring & Reaction Plane	XR Data Collector, Filterer, pre-processor	HG	HG	HG	G	-	G	-	HG	CUSTOM	??
??	Monitoring & Reaction Plane	XR Service Specific Analytics Engine	HG	HG	HG	G	-	G	-	HG	CUSTOM	TID, ICT-FI, HUA, ONE, CAI, XR Providers
??	Monitoring & Reaction Plane	XR Service Specific Decision Engine	HG	HG	HG	G	-	G	-	HG	CUSTOM	TID, ICT-FI, HUA, ONE, CAI, XR Providers
??	Monitoring & Reaction Plane	XR Service Specific Actuation Engine	HG	HG	HG	G	-	G	-	HG	CUSTOM	TID, ICT-FI, HUA,
CS	Monitoring & Reaction Plane	Running XR Services Repository	HG	HG	HG	G	-	G	-	HG	CUSTOM	CNR HUA
CS	Monitoring & Reaction Plane	Plane Services Registry & Discovery	HG	HG	HG	G	-	G	-	HG	CUSTOM	-
PLEXUS	Monitoring & Reaction Plane	Domain Resource Indexing	HG	HG	HG	G	-	G	-	HG	CUSTOM	-
ONE/ICT-FI	Monitoring & Reaction Plane	Domain Integration Fabric	HG	HG	HG	G	-	G	-	HG	CUSTOM	-

Table 12: Architecture - elements and metrics other planes

USE CASE/PARTNER	PLANE	USE CASE ELEMENT	Latency /latency ms	RTT /rtt ms	Bandwidth /bw Mbps	CPU /cpu %	GPU /gpu %	Memory /memory %	Resolution /resolution Megapixel	Petitions per second /petitionspersecond requests per secon	HTTP Endpoint Exposer	Other partners incolved
CNR	XR Service Deployment Plane	XR Device Controller	HG	HG	HG	G	-	G	-	HG	CUSTOM	XR Providers
CNR	XR Service Deployment Plane	XR Service Enabler Controller	HG	HG	HG	G	-	G	-	HG	CUSTOM	XR Providers
HUA	XR Service Deployment Plane	Storage Management Services	HG	HG	HG	G	-	G	-	HG	CUSTOM	ICT-FI, HPE, ORAMA,
CS	XR Service Deployment Plane	NFV MANO	HG	de	HG	G	-	G	-	HG	CUSTOM	ONE ICT-FI
CS	XR Service Deployment Plane	Cross Domain Resource Mano	HG	HG	HG	G	-	G	-	HG	CUSTOM	ONE ICT-FI
HUA	XR Service Deployment Plane	WIM	HG	HG	HG	G	-	G	-	HG	CUSTOM	T2.2 partners
ONE/ICT-FI	XR Service E2E Conducting Plane	Conductor Integration Fabric	HG	HG	HG	G	-	G	-	HG	CUSTOM	-
??	XR Service E2E Conducting Plane	E2E Service Specific DE/AE/ACT	HG	HG	HG	G	-	G	-	HG	CUSTOM	TID?,ICT-FI, HUA,
CS	XR Service E2E Conducting Plane	Plane Services Registry & Discovery	HG	HG	HG	G	-	G	-	HG	CUSTOM	-
HPE	XR Service E2E Conducting Plane	CI/CD Pipeline	HG	HG	HG	G	-	G	-	HG	CUSTOM	TID PLEXUS
HPE	XR Service E2E Conducting Plane	XR Service Template Blueprint Repository	HG	HG	HG	G	-	G	-	HG	CUSTOM	CNR, CS,
HPE	XR Service E2E Conducting Plane	XR Service Enabler Repository	HG	HG	HG	G	-	G	-	HG	CUSTOM	CNR
HPE	XR Service E2E Conducting Plane	XR Service Exposure	HG	HG	HG	G	-	G	-	HG	CUSTOM	-
CS	XR Service E2E Conducting Plane	Running XR Services Repository	HG	HG	HG	G	-	G	-	HG	CUSTOM	CNR HUA
PLEXUS	XR Service E2E Conducting Plane	Resource Planning	HG	HG	HG	G	-	G	-	HG	CUSTOM	CNR TID
PLEXUS	XR Service E2E Conducting Plane	Resource Indexing	HG	HG	HG	G	-	G	-	HG	CUSTOM	CNR TID

3 CHARITY Edge Storage (CHES)

3.1 Component descriptions

The CHARITY Edge Storage Component (CHES) is responsible for providing optimized edge storage services to the CHARITY framework and its hosted applications. These services include data storage, retrieval and migration tasks, security and privacy protection capabilities, QoS and QoE violation prevention and mitigation as well as other data-related services that serve the runtime requirements of CHARITY. In detail, the edge storage component has to provide a reliable, fast, stable and secure shared storage engine, accessible by all devices and users in an edge-cloud. Furthermore, it needs to be extremely lightweight since it is created for edge devices with extremely limited resources, like Raspberry Pies or other micro-computer devices.

Edge nodes generally have limited computation, storage, network, or power resources. The distributed, dynamic and heterogeneous environment in the edge and the diverse application's requirements pose several challenges. The edge storage component needs to overcome some inherent edge challenges like:

- Coordination of unreliable devices and network
- Hardware and software incompatibilities that arise due to the plethora of different devices
- Mobility of the devices and the users (in some Use Cases)
- Integration of different data storage formats and data types
- Limited resources of the edge devices
- Security and privacy concerns
- QoE insurance

CHES component is based on the Kubernetes (K3s)[3], MinIO[4] and Prometheus[5] technologies, combining and optimizing them in order to better serve the needs of CHARITY. Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. More specifically, a lightweight Kubernetes distribution built for IoT & edge computing is used, called K3s. K3s is a highly available, certified Kubernetes distribution designed for production workloads in unattended, resource-constrained, remote locations or inside IoT appliances. As a storage solution, an open-source framework created by IBM is utilized, called MinIO. MinIO is an inherently decentralized and highly scalable Peer-to-Peer solution, allowing us to deploy it freely on usable nodes. It is designed to be cloud native and can run as lightweight containers managed by external orchestration services such as Kubernetes. It supports a hierarchical structure in order to form federations of clusters and it has been proven as a valid candidate for an edge data storage system[1]. MinIO writes data and metadata together as objects, eliminating the need for a metadata database. In addition, MinIO performs all functions (erasure code, bitrot check, encryption) as inline, strictly consistent operations. The result is that MinIO is exceptionally resilient. Moreover, it 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. Unlike other object storage solutions that are built for archival use cases only, the MinIO platform is designed to deliver the high-performance object storage that is required by modern big data applications. In addition, MinIO provides both a web-based GUI and an AWS S3 compatible API library. The Kubernetes Dataset Lifecycle Framework provided by IBM's Datashim[6] is employed on top of MinIO, allowing the edge storage component to be used as a file system folder, which is useful for applications that we

[3] https://k3s.io/

[4] https://min.io/

[5] https://prometheus.io/

[6] https://datashim.io/

cannot or do not want to integrate with the Restful API of MinIO. A detailed description of the Kubernetes Dataset Lifecycle Framework is provided in Section 3.1.1. Finally, Prometheus is responsible for collecting monitoring data about the real-time performance of the nodes and the component as a whole to analyze the behaviour of different applications and optimize the cluster architecture, the options, and the data distribution.

Additionally, a Localized Docker Registry (LDR) is provided using CHES as its file storage backend, in order to move application images closer to the edge and limit network traffic and image download times. CHES Localized Docker Registry hosts the Docker images and employs Kubernetes containerization in order to provide its services, creating a new pod in the CHES namespace that is able to connect to the Minio storage backend. In addition, CHES registry creates a set of secrets that allows the secure communication between the registry and its clients using the HTTPS protocol and a basic authentication scheme.

3.1.1 Kubernetes Dataset Lifecycle Framework

Hybrid edge/cloud environment is rapidly becoming the new trend for organizations seeking the perfect mix of scalability, performance and security. As a result, it is now common for an organization to rely on a mix of on-premises data centers (private cloud), and cloud/edge solutions from different providers to store and manage their data. Nevertheless, many obstacles arise when applications have to access the data. On the one hand, developers need to know the exact location of the data and, on the other hand, manage the correct credentials to access the specified data-sources holding their data. In addition, access to cloud/edge storage is often completely transparent from the cloud management standpoint and it is difficult for infrastructure administrators to monitor which containers have access to which cloud storage solution. Even if containerized components and micro-services are widely promoted as the appropriate solution for efficiently deploying and managing storage over a hybrid edge/cloud infrastructure, containerization makes it more difficult for the workloads to access the shared file systems. Currently, there are no established resource types to represent the concept of data-source on Kubernetes. As more and more applications are running on Kubernetes for batch processing, end users are burdened with configuring and optimizing the data access [2].

To tackle the aforementioned issues, the Dataset Lifecycle Framework (DLF) is employed, which is an open-source project that enables transparent and automated access for containerized applications to data-sources. DLF enables users to access remote data-sources via a mount-point within their containerized workloads and it is aimed to improve usability, security and performance, providing a higher level of abstraction for dynamic provisioning of storage for the users' applications. By integrating DLF on Kubernetes pipelines, it is able to mount object stores as Persistent Volume Claims (PVCs), which are pieces of storage in the cluster, and present them to pipelines as a POSIX-like file system. In addition, DLF makes use of Kubernetes access control and secret so that pipelines do not need to be run with escalated privilege or to handle secret keys, thus making the platform more secure.

In more technical detail, DLF orchestrates the provisioning of PVCs required for each data-source, which users can refer to their pods (the smallest deployable unit in Kubernetes), allowing them to focus on the actual workload development rather than configuring/mounting/tuning the data access.
DLF is designed to be cloud-agnostic and due to Container Storage Interface (CSI)[7], it is highly extensible to support various data-sources. CSI is a standard for exposing arbitrary block and file storage systems to containerized workloads on Container Orchestration Systems (COS) like Kubernetes. With the adoption of COS, the Kubernetes volume layer becomes truly extensible. Using CSI, third-party storage providers are able to write and deploy plugins exposing new storage systems in Kubernetes without interacting or changing the core Kubernetes code. This provides Kubernetes users more options for storage and makes the system more secure and reliable. On the infrastructure side, DLF also enables cluster administrators to easily monitor, control, and audit data access.

[7] https://kubernetes-csi.github.io/docs/

DLF introduces the Dataset as a Custom Resource Definition (CRD)[8], which is a pointer to existing S3 or NFS data-sources. A Dataset object is a reference to a storage provided by a cloud-based storage solution, potentially populated with pre-existing data. In other words, each Dataset is a pointer to an existing remote data source and is materialized as a PVC. The Dataset is a declarative construct that abstracts access information and provides a single reference for data in Kubernetes. Users only need to include this reference in their deployments to make the data available in pods, either through the file system or through environment variables [3].

Figure 4 illustrates an example configuration of a Dataset CRD for data stored in COS. The mandatory fields are the *bucket*, *endpoint*, *accessKeyID*, and *secretAccessKey*. The *bucket* entry creates a one-to-one mapping relationship between a Dataset object and a bucket in the COS. The *accessKeyID* and *secretAccessKey* fields refer to the credentials used to access this specific bucket.
DLF is completely agnostic to where/how a specific Dataset is stored, as long as the *endpoint* is accessible by the nodes within the Kubernetes cluster, in which the framework is deployed.

```
apiVersion: com.ie.ibm.hpsys/v1alpha1
kind: Dataset
metadata:
  name: example-dataset
spec:
  local:
    type: "COS"
    accessKeyID: "{ACCESS_KEY_ID}"
    secretAccessKey: "{SECRET_ACCESS_KEY}"
    endpoint: "{S3_SERVICE_URL}"
    bucket: "{BUCKET_NAME}"
    readonly: "false"
```

Figure 4: An example of mounting a PVC created by the DLF integration

Creating a CRD is just the first step to add custom logic in the Kubernetes cluster. The next step is to create a component that has embedded the domain-specific application logic for the CRD. Essentially, a service provider needs to develop and install a component which reacts to the various events which are part of the lifecycle of a CRD and implements the desired functionality.

DLF utilizes the Operator-SDK, an open-source component of the Operator Framework[9], which provides the necessary tooling and automation in the development of these components in an effective, automated, and scalable way. Operator-SDK is utilized to create the Dataset Operator in DLF. Its main functionality is to react to the creation (or the deletion) of a new Dataset and materialize the specific object. Specifically, when a Dataset gets created, the software stack invokes the necessary Kubernetes CSI plugin and creates a PVC that provides a file system view of the bucket in the COS.

Figure 5 demonstrates in an abstract view, the Dataset Lifecycle Framework with the various components employed in an example of a two-node K3s cluster.

[8] https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions

[9] https://operatorframework.io/

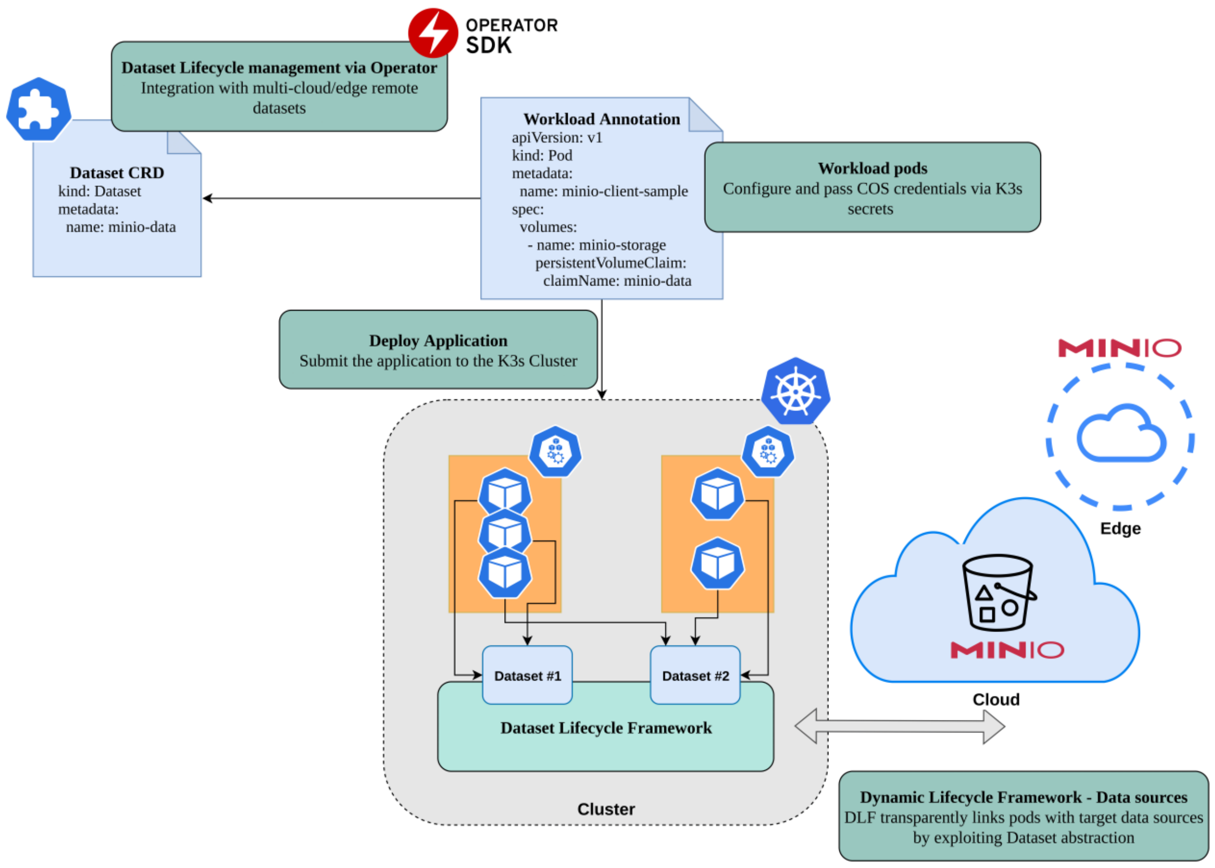

OPERATOR SDK
Dataset Lifecycle management via Operator
Integration with multi-cloud/edge remote datasets
Dataset CRD
kind: Dataset
metadata:
name: minio-data
Workload Annotation
apiVersion: v1
kind: Pod
metadata:
name: minio-client-sample
spec:
volumes:
- name: minio-storage
persistentVolumeClaim:
claimName: minio-data
Workload pods
Configure and pass COS credentials via K3s secrets
Deploy Application
Submit the application to the K3s Cluster
MINIO
Edge
MINIO
Cloud
Dataset #1
Dataset #2
Dataset Lifecycle Framework
Cluster
Dynamic Lifecycle Framework - Data sources
DLF transparently links pods with target data sources by exploiting Dataset abstraction

Figure 5: Dataset Lifecycle Framework (DLF)

3.2 Package information

3.2.1 CHES Storage

CHES is a package including Kubernetes deployment files in YAML format, installation scripts in bash script format and a configuration file in JSON format that contains all options needed to configure the component.

All files of the package are available on the official CHARITY GitLab page[10] and can be obtained with the following command:

```
$ git clone https://gitlab.charity-project.eu/hua/edgestoragecomponent.git
```

In detail, we have one YAML file called chesDeployment.yaml which is the Kubernetes deployment file for the storage server (master). This file will install all necessary services, authentication keys, roles and images on the Kubernetes cluster, reading information from the configuration file (.conf). It will use the Kubernetes architecture, deploying most services on the Kubernetes master. Of course, the actual MinIO instances that store the data will be deployed on the nodes having the label "ches-worker" set to "true". The second yaml file is called chesClientDeployment.yaml and it will allow nodes to use CHES as a file system folder by mounting the PVC that is connected to the CHES storage service.

The bash scripts are again two, chesInstalldeploy.sh that configures and deploys the chesDeployment.yaml on the Kubernetes master, and chesClientDeploy.sh that configures and deploys the chesClientDeployment.yaml on the client nodes. These scripts are just applying the options selected in the configuration file to the YAML files and then run the necessary commands to deploy

[10] https://gitlab.charity-project.eu/hua/edgestoragecomponent

the YAML files on the Kubernetes cluster. There is a third bash script called InstallScript.sh which is configuring and deploying the chesDeployment.yaml file in a single K3s cluster node installation, without requiring any additional configuration steps.
Finally, a yaml file called dlf_kube.yaml is used for the deployment of the Dataset Lifecycle Framework and a bash script named Undeployches.sh which undeploys the CHES containers and jobs. A complete list of the files included is presented in Table 13.

Table 13: List of package files for Edge storage component

Filename	Description
`chesDeployment.yaml`	Kubernetes deployment file for CHES master
`chesClientDeployment.yaml`	Kubernetes deployment file for CHES client(s)
`chesInstalldeploy.sh`	Bash script for deploying the CHES servers
`chesClientDeploy.sh`	Bash script for deploying the CHES client(s)
`InstallScript.sh`	Bash script for deploying the CHES servers on single node clusters
`configuration_file.conf`	JSON file containing the configuration options for CHES
`dlf_kube.yaml`	Kubernetes deployment file for the Dataset Lifecycle Framework
`Undeployches.sh`	Bash script for undeploying the CHES containers and jobs

3.2.1.1 Kubernetes Dashboard

Along with the CHES component, the Kubernetes dashboard is provided, which is a web-based Kubernetes user interface. In general, Kubernetes dashboard is used to deploy containerized applications to a Kubernetes cluster, troubleshoot the containerized applications, and manage the cluster resources. In addition, the dashboard can get an overview of applications running on a cluster, as well as for creating or modifying individual Kubernetes resources (such as Deployments, Jobs, DaemonSets, etc). Dashboard also provides information on the state of Kubernetes resources in the cluster and on any errors that may have occurred. The associated files are located in the same repository with CHES.

In detail, the installation of Kubernetes dashboard includes four files, two deployment yaml files and two bash scripts. A bash script named InstallDashboard.sh is used for deploying the Kubernetes dashboard in a K3s cluster. A complete list of the files included, is presented Table 14.

Table 14: List of files included in the Kubernetes Dashboard

Filename	Description
`InstallDashboard.sh`	Bash script for deploying the Kubernetes Dashboard
`recommended.yaml`	Kubernetes deployment file for Kubernetes dashboard
`dashboard_account_roles.yaml`	Kubernetes deployment file for creating a minimal RBAC configuration, i.e. a Service Account and a ClusterRoleBinding
`UndeployDash.sh`	Bash script for undeploying the Kubernetes Dashboard

3.2.2 CHES Localized Docker Registry

CHES Localized Docker Registry is the second sub-component that realizes a localized registry in order to support the faster application deploying and limit the network flooding caused by large image downloads during deployment. This functionality acts as a proactive caching mechanism by optimizing the download delays and the network traffic. The port of the CHES LDR as well as its credentials are pre-configured using the generalized configuration file that is packed with the edge storage solution. Figure 6 illustrates the CHES LDR sub-component. The associated files are separated into a different folder, in order to separate them by functionality, make documentation and maintenance easier and decouple their installation process.

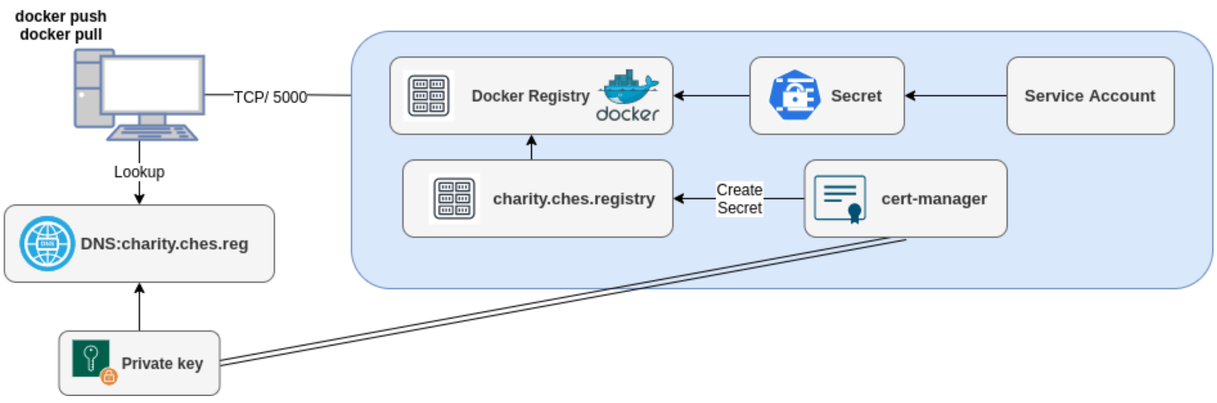


Figure 6: CHES Localized Docker Registry

CHES Localized Docker Registry can be downloaded by running the command:

```
$ git clone https://gitlab.charity-project.eu/hua/edgestoragecomponent.git
```

In detail, the installation of CHES LDR includes six files, four deployment yaml files and two bash scripts. The yaml files are deploying all the necessary containers and jobs that need to be executed to setup and configure the registry, in order to be functional and accessible by other containers hosted in the same K3s cluster. A complete list of the files included is presented in Table 15 .

Table 15: List of files included in the CHES Registry repository

Filename	Description
`add_certs.yaml`	Kubernetes deployment for a daemon job that adds the appropriate SSL certificates to new containers
`add_to_hosts.yaml`	Kubernetes deployment for a daemon job that adds the appropriate configurations to the hosts files of new containers
`deployment.yaml`	Kubernetes deployment for the Docker registry container
`registry_setup.sh`	Bash script for deploying the CHES LDR containers and jobs
`registry_uninstall.sh`	Bash script for undeploying the CHES LDR containers and jobs
`test_deploy.yaml`	Kubernetes deployment for a test container that loads a docker image from the deployed CHES LDR

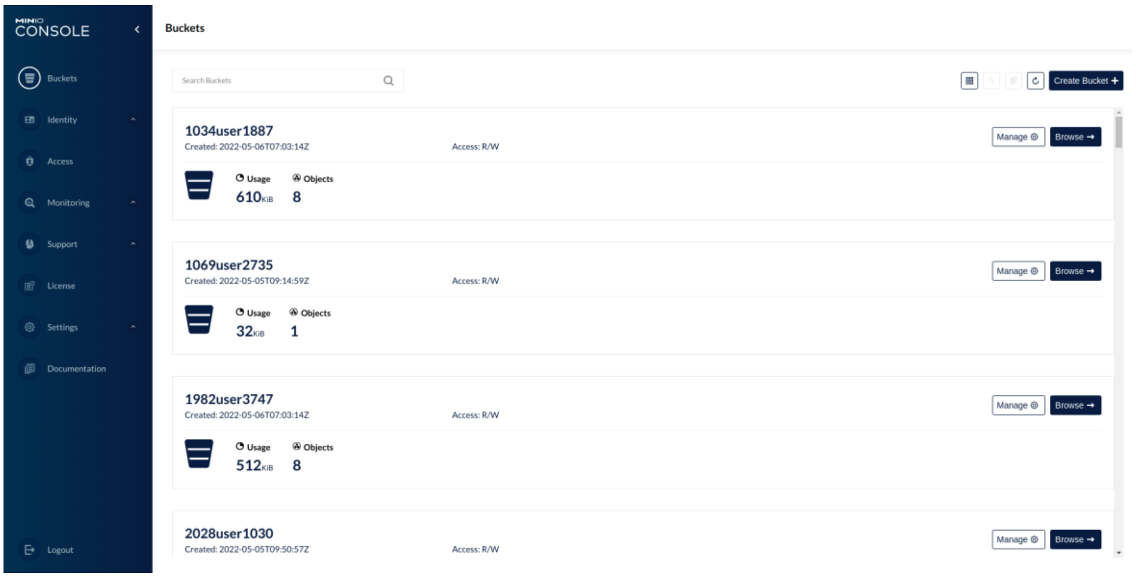
3.2.3 Semi-automated Deployment and off-loading

In the context of the presented solution, a set of bash and yaml scripts have been developed that handle all the configuration, installation and deployment processes that need to be contacted before and after the MinIO workers are deployed. These configurations include firewall rules, DNS settings, package installations and security checks that take into account the setup environment, the architecture and resources of the physical machines and the software involved. These tasks enable the semi-automatic deployment of the edge storage solution, forming complex pipelines that in most other cases are performed manually by a system administrator. This ensures that scaling can be performed seamlessly on each cluster, regardless of the underlying physical machines that act as nodes. In addition, off-loading of data can be achieved by "ordering" more instances of the MinIO worker to be deployed on more nodes and adding them in the same MinIO cluster in real-time.

3.3 User Manual

3.3.1 CHES Storage

We have three ways to utilize CHES, the first way is through the MinIO Web GUI which is clearly described in detail on the official MinIO documentation[11]. A sample MinIO storage deployment can be seen in Figure 7.



1034user1887
Created: 2022-05-06T07:03:14Z Access: R/W
Usage 610KiB Objects 8

1069user2735
Created: 2022-05-05T09:14:59Z Access: R/W
Usage 32KiB Objects 1

1982user3747
Created: 2022-05-06T07:03:14Z Access: R/W
Usage 512KiB Objects 8

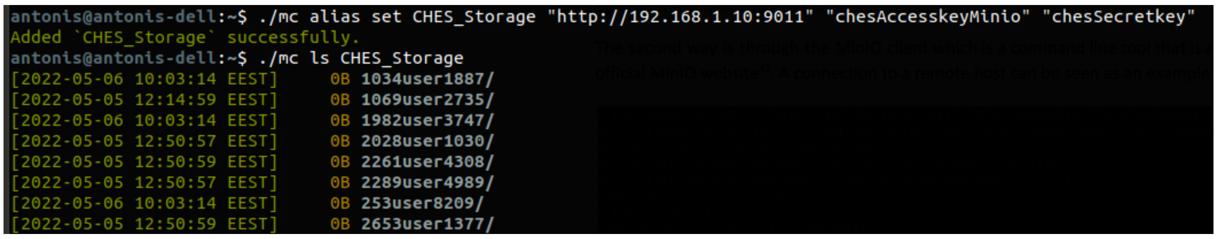
2028user1030
Created: 2022-05-05T09:50:57Z Access: R/W

Figure 7: The MinIO web-based interface

The second way is through the MinIO client which is a command line tool that is also documented in detail on the official MinIO website[12]. A connection to a remote host can be seen as an example in Figure 8.

[11] https://docs.min.io/docs/minio-quickstart-guide.html

[12] https://docs.min.io/docs/minio-client-complete-guide.html

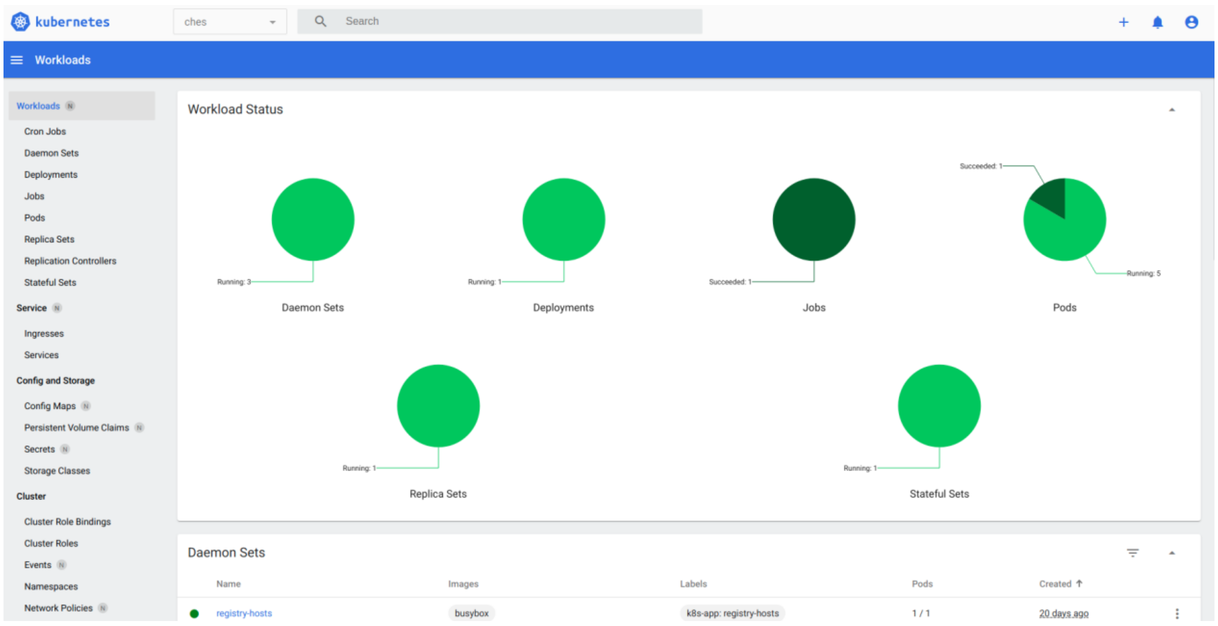


```
antonis@antonis-dell:~$ ./mc alias set CHES_Storage "http://192.168.1.10:9011" "chesAccesskeyMinio" "chesSecretkey"
Added `CHES_Storage` successfully.
antonis@antonis-dell:~$ ./mc ls CHES_Storage
[2022-05-06 10:03:14 EEST]     0B 1034user1887/
[2022-05-05 12:14:59 EEST]     0B 1069user2735/
[2022-05-06 10:03:14 EEST]     0B 1982user3747/
[2022-05-05 12:50:57 EEST]     0B 2028user1030/
[2022-05-05 12:50:59 EEST]     0B 2261user4308/
[2022-05-05 12:50:57 EEST]     0B 2289user4989/
[2022-05-06 10:03:14 EEST]     0B 253user8209/
[2022-05-05 12:50:59 EEST]     0B 2653user1377/
```

Figure 8: An example connection with the command line MinIO client

Additionally, using the integrated Datashim's DLF, CHES can be accessed through the K3s deployment files by mounting the PVC it creates as a system volume. Detailed reference of the usage of PVCs can be found in the Kubernetes API documentation[13]. An example of the deployment is illustrated in Figure 4.

Moreover, the Kubernetes dashboard which is a web-based Kubernetes user interface is illustrated in Figure 9.

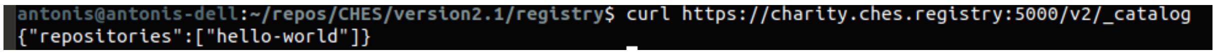


Name	Images	Labels	Pods	Created ↑
● registry-hosts	busybox	k8s-app: registry-hosts	1 / 1	20 days ago

Figure 9: Kubernetes Dashboard

3.3.2 CHES Registry

CHES LDR can be accessed through the Docker Registry APIs. These APIs are described in the official Docker documentation[14]. An example of the catalog API, which lists the available repositories, is illustrated in Figure 10. Catalog API is the simplest of the APIs provided, displaying a list of the available images pushed in a registry. In our case it is hosting an example *hello-world* image.



```
antonis@antonis-dell:~/repos/CHES/version2.1/registry$ curl https://charity.ches.registry:5000/v2/_catalog
{"repositories":["hello-world"]}
```

Figure 10: Example of the catalog API for CHES LDR hosted in a local K3s cluster

[13] https://kubernetes.io/docs/concepts/storage/persistent-volumes/

[14] https://docs.docker.com/registry/spec/api/

3.4 Licensing

This component, including all originally created source files, scripts and other resources is going to be published as free software under the terms of the GNU General Public License version 3 or later, as published by the Free Software Foundation.

MinIO is provided under GNU Affero General Public License version 3 which enables us to use it as an open-source component providing that we also use a GNU public License.

Prometheus, Datashim and K3s are protected under Apache License which gives us full usability of their open-source components.

3.5 Results obtained in relation to the objectives (KPIs)

The work conducted in Task 3.2 aims in achieving the objectives along with the requirements and targeted KPIs. More specifically, the KPIs that will be met from Objective 2 (*Provide holistic support for the orchestration of advanced media solutions*) are:

- **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*" [28] has already been submitted which presents the proposed new edge storage solution (CHES).
- **KPI-2.3 Edge storage hit rate: higher than 70%**
  - The native "disk cache" feature of MinIO is investigated. 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.
  - An online proactive caching scheme based on deep recurrent neural network models is investigated to predict time-series content requests and update edge caching accordingly.
- **KPI-2.4 Blockchain for edge storage transaction rate: more than 4 transactions per second**
  - A blockchain database, namely BigchainDB[15] is being explored. 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). One design goal of BigchainDB is the ability to process a large number of transactions each second. Each BigchainDB instance is a virtual concept consisting of three parts: i) a MongoDB database, ii) a BigchainDB server and iii) a Tendermint communication node which uses a Byzantine Fault Tolerant middleware for networking and consensus. Preliminary results demonstrated that MinIO is able to achieve a higher transaction rate (*4.3*) compared to BigchainDB (*3.2*) for a specific class of experiments. The performance evaluation was executed through Locust[16], an open-source load testing framework that enables the definition of user behaviour and supports running load tests distributed over multiple machines

[15] https://www.bigchaindb.com/

[16] https://locust.io/

and simulates millions of simultaneous user requests. Overall, the experimental results demonstrated that MinIO presents the best performance in both read and write operations. To further evaluate the storage systems, we also measured the RAM usage, the CPU usage, the disk latency and the disk IO time for a single user's request and for all users' request. Again, MinIO achieved the best performance. 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. However, we will investigate further the blockchain capabilities for increasing the edge storage transaction rate.

3.6 Relation to research questions

There are a number of research questions regarding the edge storage, which are actively being researched at the moment. These questions include the intelligent data placement in computing networks, the pro-active and intelligent caching of data, the minimization of resource waste and the maximization of resource efficiency and the harmonization of IoT network diversity. The present research work and the designed component provides solutions to most of these open research questions by providing a complete edge storage solution that takes into account the present issues in IoT edge networks and the vast number of data transactions that continuously happen between them.

Pro-active and intelligent caching of data are two questions that also trouble the academic community and the industry for a very long time. It concerns the replication or migration of data before they are needed to have them ready for usage when they are finally needed. This minimizes the wait time of operations since the I/O and network operations, which usually take much more time to be completed than processing does, are performed before they are needed. In order to achieve that, an edge storage system needs to be able to predict the need for a specific data packet early enough to be able to complete the data operations before the need arises. Modern approaches are using machine learning in order to profile the applications and the users of a system, extracting patterns of behaviour that hint at the future data operations. The presented solution is using Kubernetes as an orchestrator, which enables us to define certain node affinity and node selection rules that aid the selection of storage workers and the placement of the data inside an edge cluster. The affinity rules are relaxed rules that are instructing Kubernetes to prefer nodes that are meeting most of the affinity rules specified. On the other hand, selection rules are strict and instruct Kubernetes to deploy the storage workers on nodes that fulfill all of the selection rules. These rules can be dynamically set either by a network administrator or by an automated mechanism such as an intelligent agent or a machine learning model that can estimate the most efficient placement of storage workers.

Harmonization of IoT network diversity concerns the definition of a uniform way of handling the various IoT devices that can be a part of an edge cluster. An IoT edge network is like a living organism. The parts that comprise it can change at any given time either because they do not wish to be part of the network anymore, due to hardware or software malfunction, scaling out and in operations or for any other reason that removes or adds new devices over the device-edge-cloud continuum. The presented solution is using K3s as an orchestrator which is compatible with most devices that run windows or unix based operating systems. This enables the administrators to create generalized deployment scripts that handle the deployment, configuration, un-deployment and re-deployment of the storage workers. These generalized scripts are highly configurable and can be edited in real time by higher level scripts and automated mechanisms adding more layers of intelligence and automation to these deployment and configuration processes. In addition, DLF provides a uniform way of accessing the data, using the local file system of each device, eliminating the need of customized solutions for each new device that becomes a member of the device-edge-cloud continuum.

3.7 Evaluation of CHES

The CHARITY Edge Storage Component aims at improving the Quality of Experience (QoE) of the end-users by migrating data “close” to them, thus reducing data transfers delays and network utilization. To evaluate the effectiveness of the storage component, a number of resource utilization and Quality of Service (QoS) metrics are collected using the Prometheus system. The data are collected on the edge, by Prometheus agents running on edge nodes that handle the data storage. These data are stored in the Prometheus database of each edge cluster. More specifically, the data are collected at regular intervals of 5 minutes throughout the functional period of the component, i.e. for the whole duration that the edge storage component is active and waiting for serving data requests.

The evaluation metrics employed are divided into two categories:

- Resource consumption: CPU available (total, used), RAM available (total, used), HDD available (total, used), Network available (total, used)
- Performance: Throughput, Data request response time, and Network time

The resource consumption metrics of the first category are all being passively collected by the Prometheus agents placed on storage nodes. The performance metrics of the second category on the other hand, require a client-side approach so they are actively collected only during benchmarks and tests.

The evaluation is conducted using two CHES deployments, one in a local and one in a remote edge cluster. The behaviour of CHES is evaluated using a collection of small to medium binary files ranging from 15KB to 10MB. All these files are forming the evaluation dataset that is stored in various MinIO buckets, created and managed by CHES in the local and remote edge cluster. These buckets are then mounted onto new pods, using the DLF, and these new pods are taking the role of clients, sending data requests to the CHES and recording performance metrics for these requests.

Figure 11 illustrates the percentage change of various resource utilization metrics -CPU Usage, Memory Usage, Available Memory, Disk Write Latency, Disk IO time- during intense data transactions and during normal functionality of the node.

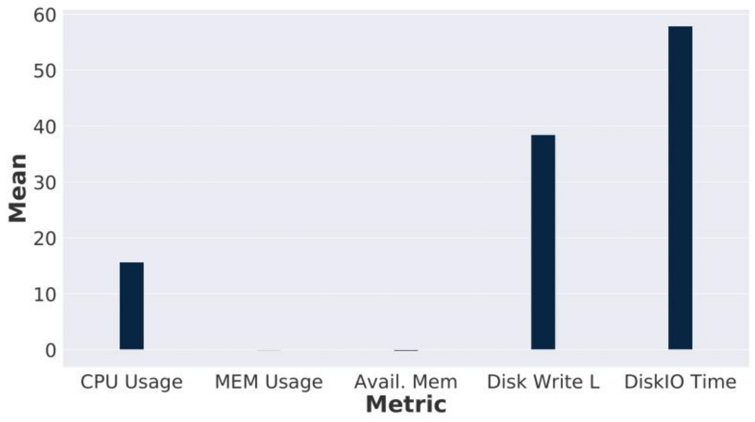


Figure 11: Percentage change of various resource utilization metric

As the results suggest, CHES is not overusing the RAM of the node, although it is slightly increasing the usage of the CPU and the disk operations, as expected. This proves that CHES is lightweight enough to be deployed on most edge devices. More specifically, the RAM related metrics are near to zero, meaning almost no change, the CPU metric is slightly increased while the disk metrics are increased by a larger degree, proving intense I/O activity.

Client-side metrics collected to assess the impact of CHES on QoE, are presenting a clearer picture of how CHES improves the response times of various data requests. Figure 12 and Figure 13 demonstrate the comparison between read, write and delete operations for the local and the remote CHES respectively.

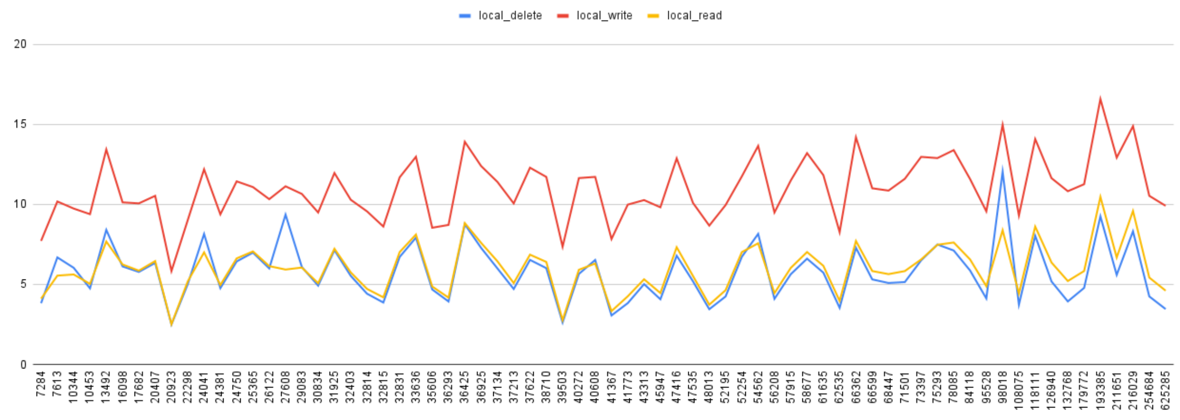


Figure 12: Read, Write and Delete operation response times in milliseconds for the local CHES deployment

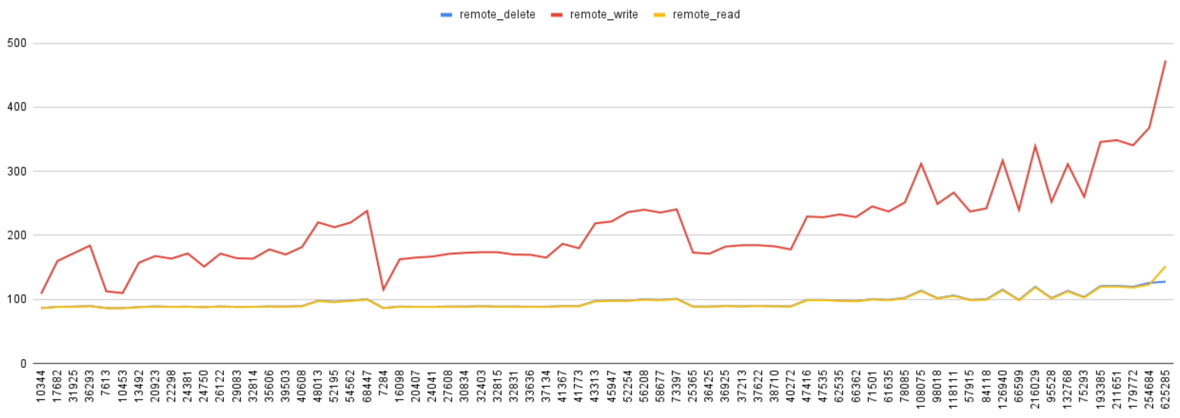


Figure 13: Read, Write and Delete operation response times in milliseconds for the remote CHES deployment

Due to the object store nature of MinIO, it can be observed that write operations are more time consuming compared to read and delete operations. On the other hand, read and write operations do not differ much compared to each other, the only difference is the network delay for the final file transfer, which is pretty small taking into account that present evaluation tests were conducted using file transfers of multiple small to medium files.

The comparison between the different operations are similar but at a different scale; for the local CHES, response times vary between 3 to 17 ms while for the remote CHES, response times vary between 84 to 450 ms. This is becoming more obvious when putting the response times into direct comparison, as illustrated in Figure 14. The request response time for the local CHES is under 20 ms for all file operations which is significantly lower than the remote CHES. In summary, all data operations were significantly enhanced during runtime when the data storage was placed near the edge devices.

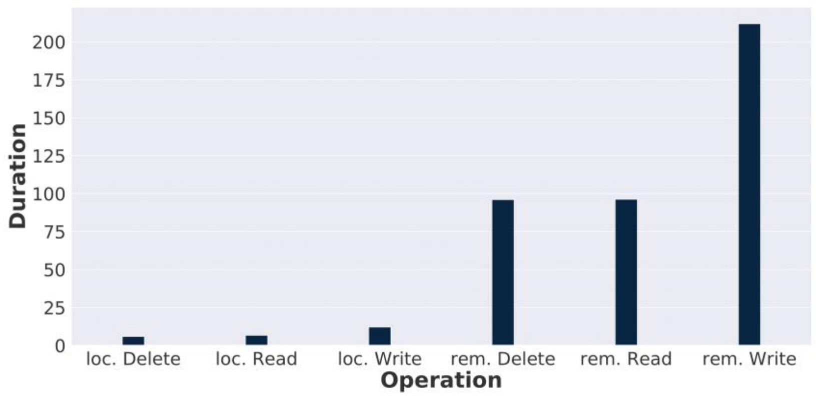


Figure 14: Comparison of response times for various operations for the remote and local CHES deployments

In conclusion, the above experiments prove two things: a) the lightweight nature of the edge storage component, making it a perfect fit for edge device deployments and b) the great reduction in data request response times, which on some edge use cases is a necessity for their basic functionality.

Detailed results can be found at the scientific conference paper entitled "*Towards a Distributed Storage Framework for Edge Computing Infrastructures*" [29] presented at the 2nd Workshop on Flexible Resource and Application Management on the Edge (FRAME 2022).

4 Resource-aware Adaptation Mechanisms

While the cloud offers extreme scaling opportunities through the dynamic allocation of physical resources to meet demand, it comes at a cost. Apart from the design challenges of engineering an elastically scalable architecture, the financial costs of cloud resources require careful monitoring. Although an application may be able to physically scale to meet demand, it may not be able to do so economically - unconstrained growth leads to unconstrained costs and if the returns do not exceed the investment, then cost can serve as a scalability brake. Edge computing resources such as those increasingly offered through metropolitan points of presence by hyperscalers and the forthcoming rollout of hyper-local edge infrastructure throughout 5G radio networks, offer new architectural options for domains such as real-time media streaming which require the flexibility of the cloud with the low latency typically associated with locally dedicated hardware. In comparison to traditional cloud deployment, edge resources are far scarcer requiring a more measured approach to scalability as there may simply be insufficient physical resources available in proximity to the user for optimal operation.

Across the cloud and edge, software engineers will increasingly find themselves challenged with designing software that needs to scale and dynamically adapt its tactics to suit the computational and network resources currently available within the environment in which it operates. With a Service Based Architecture approach increasingly favoured in modern architectures, there is a growing challenge with respect to how we equip services with sufficient adaptability to adjust their operation in line with the ebb and flow of physical resources available, and affordable, in their local environment.

4.1 Dynamic Software Adaptation

Software should be designed for change so that maintenance and reuse efforts can be minimised. Designing for variability has the significant advantage of enabling architects and engineers to delay key decisions until late in the development cycle or even until run time through site configuration. The longer we can accommodate a delayed decision, the more information we may have to hand when having to make the decision as requirements are adjusted in line with customer needs and environmental realities. These delayed design decisions are known as variability points [6] and the successful integration and curation of variability points has been the subject of intensive research for decades [7]. Variability points serve a key role in the design and construction of software product lines in which organizations seek to reassemble collections of reusable components into distinct members of a product family through leveraging a wide array of architectural, engineering and run-time variability point strategies ranging from abstract, interchangeable design stereotypes to run-time command line parameters [8].

While there is much active research into Software Product Line Engineering (SPLE) to attain development and deployment reuse efficiencies at industrial scale [9], the approach necessitates a highly planned, rigorous, and disciplined approach to variability management throughout the software design and implementation phases. It facilitates the reuse of software across multiple products in the same family by carefully designing variability points that can be leveraged during the software build and deployment process. An extension of this approach, known as the Dynamic Software Product Line (DSPL) paradigm, merges SPLE with techniques to adapt software at runtime to produce a collection of variability points that may be manipulated through configuration or runtime binding to alter the behaviour of deployed software [7].

Configurability lies at the heart of modern software development and it is rare for software to be developed to such a narrow purpose and exact set of parameter values that no deployment configuration is required. Indeed, configurability is desirable as it can improve the versatility of software and often enable functional behaviour or adaptation to environmental setups that were not envisioned at the time of initial software deployment. While some software is equipped with runtime dynamic configurability and zero downtime, the vast majority of software at least supports static configurability. This could be facilitated through environment variables, command-line parameters or configuration held in a file or repository of some form. Such configurability essentially exposes a collection of variability points which can be manipulated to affect the behaviour of the software and the principle is the same irrespective of whether the software was developed in-house, open-source or closed-source acquired from a third party.

The number and nature of variability points exposed will vary from one application to the next and can range from debugging trace activation to port numbers and timeout values, from sampling rates to thread numbers. In fact, the very configurability of software often results in a software configuration space explosion [10] that causes challenges for the testability of software (Linux has well over 10,000 configurable features [11]). In the hands of a knowledgeable user however, configuration is a powerful tool to adapt and tune software to its environment and user needs.

4.1.1 A structure for adaptation

In [12], the authors put forth a vision of autonomic computing in which software systems could self-manage according to specific goals. Each component would be designed as an autonomic element which would manage its own internal behaviour and relationships with other autonomic elements through integration of an autonomic manager in each element. This manager would take responsibility for monitoring the operation of the element and its interactions and adjust the operation of the element as required (e.g. enable/disable features).

The autonomic manager comprises of what has come to be known as a MAPE-K loop – Monitor, Analyze, Plan and Execute (see Figure 15) according to available Knowledge. In DSPL, the autonomic manager becomes the adaptation manager as shown in the figure below.

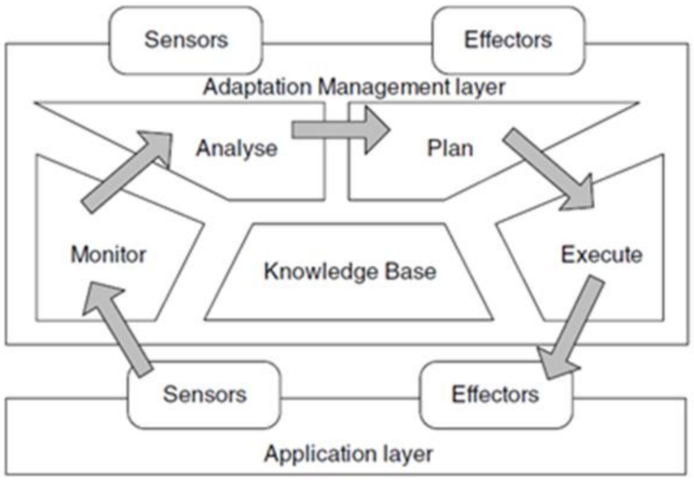


Figure 15: MAPE-K Loop [7]

The Monitoring step is concerned with capturing data regarding the properties which will drive the adaptation choices. The Analysis step examines the monitored data and performs any necessary pre-processing before making it available to the Planning step which decides, if adaptation is required, which variant of the system is more suited to the current conditions. Once the variant has been identified then the Execution step performs the adaptation.

4.1.2 Context Monitoring & Analysis

Applications and their environment need to be monitored to observe when the operation of software needs to be adapted. To record the properties being monitored, the adaptation manager can maintain flat context variables [13] or a more sophisticated hierarchical ontology [14] maintained as a dynamically updated property set that can be undergo analysis using pre-defined rules or queries to check for conditions that would warrant an adaptation.

4.1.3 Planning

“Our claim is that a major reason for the lack of context-aware, adaptive mobile applications is the inherent complexity of building them. Not only need the developers understand the main functionality of the application and how this can be provided on a mobile device, but they also have to conceive different application variants, specify how applications are linked to the execution context variables, and consider which variant should be activated under which context conditions. This complexity may easily appear like an insurmountable barrier to the developer” [13].

As mentioned previously, the potential system variant explosion arising from variability points in a software application can overwhelm the testing efforts. If left to an adaptation manager to explore unbridled, automated manipulation of variability points at runtime can lead to operational profiles that were not tested or foreseen by the developers. In the field of DSPL, the approach of static goal evolution involves an approach in which a software system has a fixed adaptation policy and system variants [7]. In the event the system needs to adapt to a new goal (operate at a reduced media streaming resolution for example), then the system is stopped, modified and restarted. Verification of such systems is greatly simplified as the state space is highly constrained. This suggests a model in which the Planning step of the MAPE-K loop can collapse to the selection of a particular variant in response to a given goal.

4.1.4 Execution

To initiate adaptation, it is required to reconfigure the software using some form of runtime reconfiguration mechanism. How this may be accomplished naturally depends on the design and capabilities of the software. Approaches based on capabilities of the software architecture range from dynamic aspect weaving essentially rewiring the software assembly on the fly [15] to service re-routing in a service-oriented architecture. In [11], the authors examined self-adaptation within a micro-service architecture for a media streaming platform in which they proposed leveraging the rollout functionality available in the Kubernetes platform which can perform rolling upgrades of a given micro-service without service interruption.

4.2 Challenges

In CHARITY we seek to enable the self-adaption of software systems to significant fluctuations in the resource availability within the execution environment. Based on an analysis of the state of the art and considering the particular needs of CHARITY, we identify a number of challenges.

4.2.1 Avoid design time intrusions

We seek to avoid prescriptive, opinionated approaches which step into the architecture and design of such systems requiring particular scaffolding and algorithms to be integrated. We adopt this position for a number of reasons. Firstly, most software is legacy software and seeking developers to modify this software retrospectively creates a significant barrier to adoption. Secondly, updates to the adaptation design and capabilities places an onus on developers to integrate these changes into their software resulting, over time, in version mismatches and requiring constant vigilance to maintain backwards compatibility. Thirdly, not all the components and services employed in a given software system are modifiable. They may be commercial or otherwise unavailable for modification and, even when the source is available, it may have been written by a third party (e.g. open source) and difficult to modify without subsequent upgrade and maintenance concerns.

4.2.2 Prevent platform instability

CHARITY seeks to support a micro-service architecture which can involve chains of services working together. When performing an adaptation, we need to be careful that the integrity of the chain is maintained and that all services that need to be made aware of an adaptation *are* made aware – regardless of where they are deployed (device, edge, cloud).

4.2.3 Accommodate user-level adaptation

CHARITY also aims to support media streaming software services that operate at scale. At any given point in time, there will be a mix of users using a particular service that may necessitate different priorities. For example, in a flight training simulation system, users co-operating in a key team operational training exercise may take priority over individual users experimenting with the controls of the flight simulator. Alternatively, we may want to maintain a high Quality of Experience (QoE) for existing users but lower the QoE for new users entering a resource-stressed environment. Supporting this model of operation will require that CHARITY supports a multi-tenant architecture where applications can simultaneously operate in different modes and priorities.

4.2.4 Transparency & Tractability

It is imperative that system adaptations are predictable and visible to avoid instability or loss of confidence.

4.3 Adaptation Infrastructure

As discussed previously, variability points are used in Software Product Line (SPL) engineering to delay decisions until such point as we are better informed as to how software needs to adapt to its use and environment. Run-time adaptation through manipulation of variability points at run-time is used in Dynamic SPL (DSPL). In CHARITY we propose to implement a DSPL model which utilizes existing variability points in a software application to provision a collection of service editions in which each service has a number of differently configured copies of itself deployed. In this model, the function of any given service edition does not change during its lifetime (i.e. the software itself is not expected to self-adapt) but rather different configurations of it are selected according to the environmental circumstances. This model is depicted below in Figure 16 in which we show three services – each with multiple editions – that exchange information to operate an overall software application.

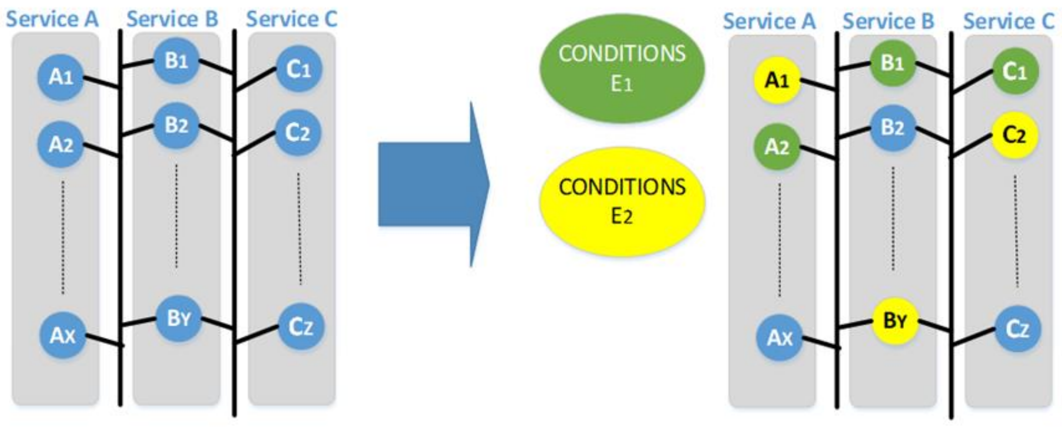


Figure 16: Service Editions used to satisfy different environment conditions

In effect, we propose to use static goal evolution [7] in which we constrain the variability state space to explicitly configured variants and thus prevent the application from entering into unforeseen (and untested) states. There are a range of challenges involved here:

- How do we enable multiple editions of a single service to operate alongside each other.
- How do we decide which editions to use under given circumstances and wire these together into a safe and coherent service chain.
- How do we route traffic between services without them needing to be made aware of multiple editions.
- How do we monitor the environment.

As we will discuss in the following sections, we propose an evolution of the MAPE-K loop introduced previously for runtime adaptation in DSPL. In CHARITY we propose to position a Service Mesh between the Adaptation Management and Application Layers as shown below in Figure 17.

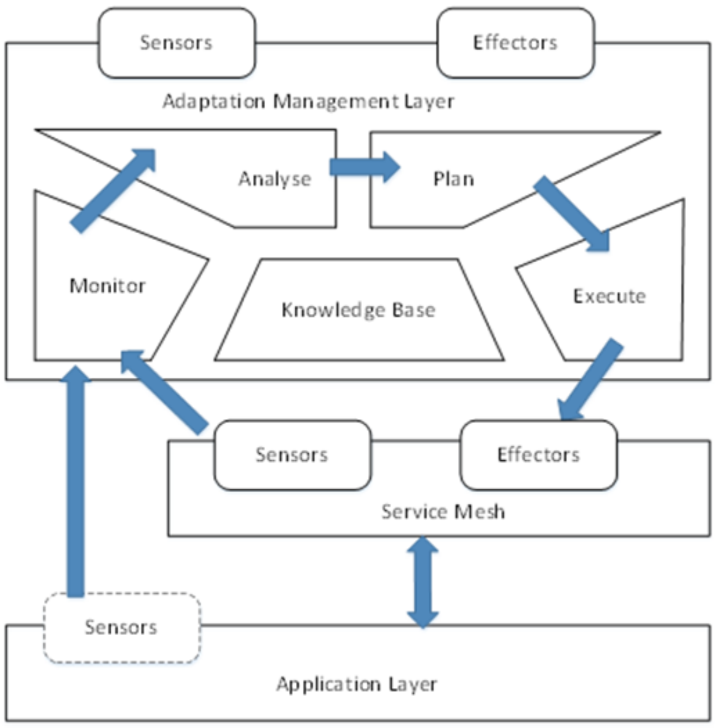


Figure 17: MAPEK-K look modified to include a service mesh for monitoring and execution

Based on the previously identified research and technical challenges, the following sections outline how we plan to meet these in CHARITY.

4.3.1 Configuration Containment

One of the fundamentally transformative benefits of Docker containers for software development has been the ability to create separate self-contained environments for experimentation and production. On a single host, we can deploy multiple containers hosting applications that, if run collectively outside the container confines on a single node, would come into conflict with each other – for example, conflicting version requirements of common software packages; conflicting requests to use the same ports, environment variables or journal files. Containers allow us to run multiple copies of the same application side by side without coming into conflict. This ability to contain the application's environment to just that application allow us to painlessly run multiple copies of the same application side-by-side with different configurations. Configurability through feature flags and configuration options at application launch is a widely used technique in software development to offer a variety of deployment variations to suit the needs of the given environment (whether business or operational) [4]. Docker containers enable us to leverage the power of this configurability in a production environment.

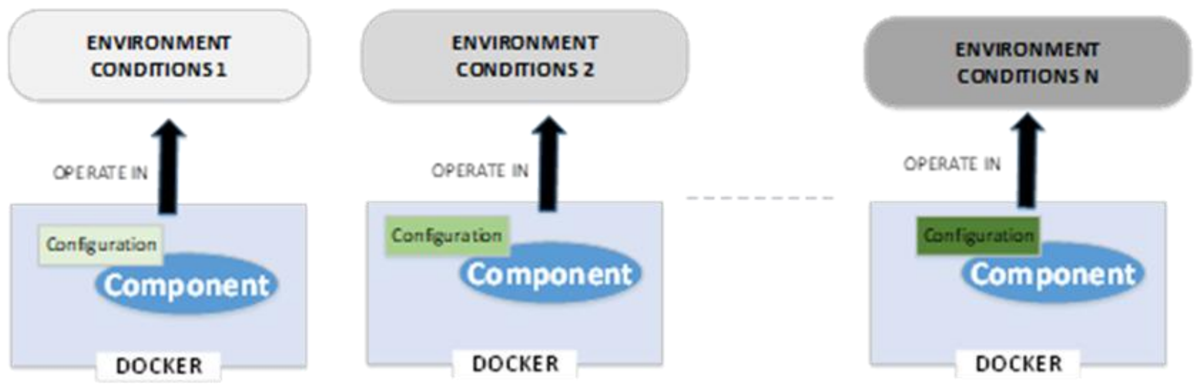


Figure 18: Run differently configured copies of a single application simultaneously

Given a particular set of environmental conditions (e.g., GPU availability, network latency, user request profile) then we may find that a change to the configuration of a given component to alter its mode of operation (e.g., disable feature, reduce sampling rate) may produce a more stable application that operates more in tune with the environment in which it finds itself.

While some software is equipped with runtime dynamic configurability and zero downtime, most will support static configurability through environment variables, command-line parameters or configuration held in a file or repository of some form. Such configurability essentially exposes a collection of variability points which can be manipulated to affect the behaviour of the software - irrespective of whether the software was developed in-house, open-source or closed-source acquired from a third party. By leveraging the environment isolation properties of containers, we can launch multiple instances of a service in different configurations. As we shall see, coupled with the ability of Kubernetes to orchestrate the launch of groups of services, containers bestow a powerful ability to seamlessly replace whole subsets of a service-based application to deliver a coherent application variant – involving multiple individual service variants working in concert - in a safe and predictable manner.

4.3.2 Service Routing

With more focused and cohesive segmentation of responsibilities into separate services, service-based architectures rely extensively on inter-service communication to collectively perform their work. In Microservice-based architectures, the mechanics of enabling services to communicate with each other robustly requires careful and detailed design and planning. Apart from peer discovery, there are significant challenges involved in establishing and monitoring communication links. Transferring control from one process to another – irrespective of the distance between them – requires coordination in the event of link failure. We must facilitate failover between multiple copies of services and indeed decide on the efficient distribution of traffic when multiple peers are available to accept it.

The Service Mesh concept [5] seeks to offer an overlay onto existing microservice architectures to take over many of the common operational and infrastructure responsibilities that would otherwise have to be engineered into the services themselves. Of particular interest within the context of application adaptation is inter-service communication and whether we can leverage a service mesh to deliver a frictionless means of dynamically routing traffic between peers such that we can swap a given service for a differently configured variant and update the routing so that other peers do not experience any collateral effects.

The service mesh is manifested as a collection of proxy processes that sit between services. Each individual service in the application is deployed with a sidecar network proxy which operates as a mediator for all inter-service communication on the service's behalf. Services do not require any knowledge or modification to operate with the sidecars which simply position themselves as highly efficient communication mediators.

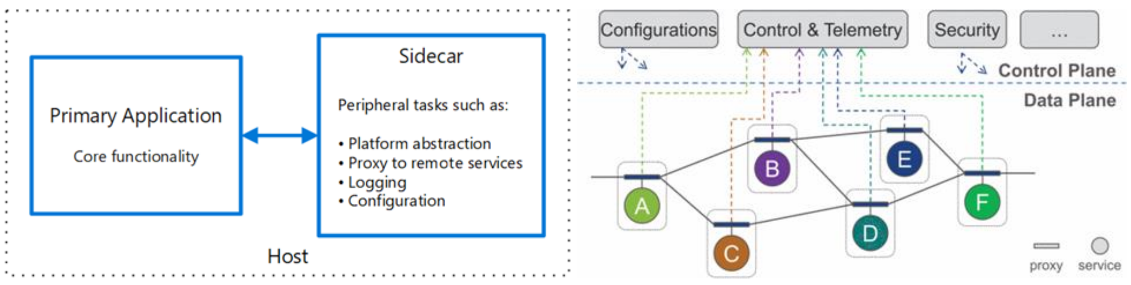


Figure 19: The Sidecar Pattern [22] and an architectural overview of its use in the Service Mesh [5]

The Service Mesh [5] offers a powerful and unintrusive mechanism to introduce flexible and dynamic route configurability amongst a group of communicating services and we intend to leverage it for the runtime adaptation of service-based applications.

4.3.3 Application Quality Modes

Consider an application comprised of three microservices as shown below. The services deliver a response or perform a particular action in accordance with a request. We refer to the sequence of services involved in delivering on this response as a Service Chain.

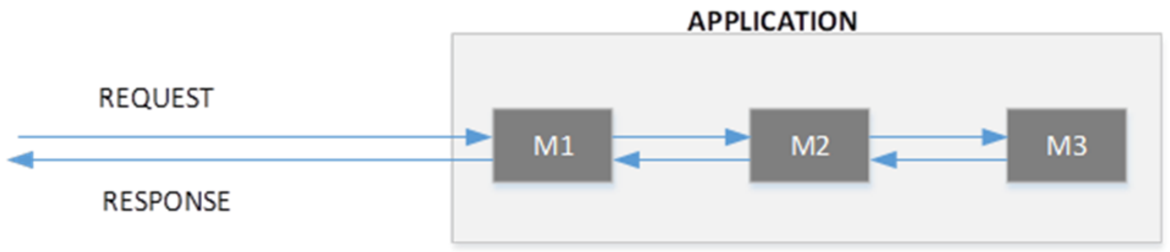


Figure 20: Simplified Application with Microservice Architecture

For an interactive XR streaming application the Quality of Experience is typically measured according to the response or action performed in response to the triggering request on several dimensions.

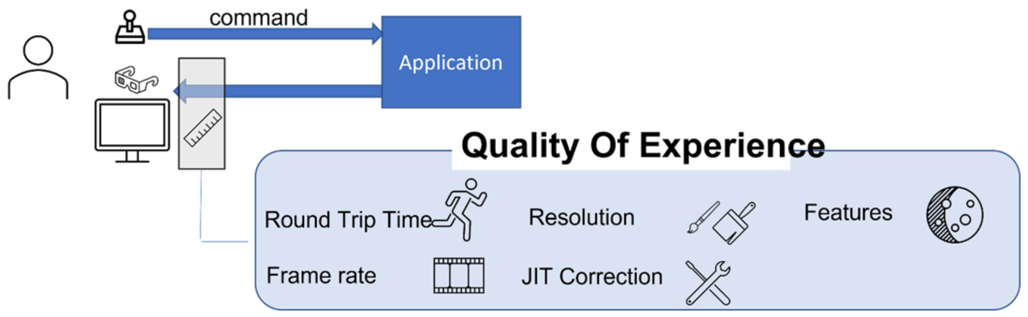


Figure 21: XR Application Quality of Experience is often multi-faceted

The Round-Trip Time (motion-to-photon, glass-to-glass) captures how long it takes for the application to deliver updated imagery in response to the triggering user interaction. The Frame Rate captures how many frames per second the application is delivering to the user device. Resolution captures how many pixels per frame are being rendered. Just in Time (JIT) Correction is the term we will assign to processing carried out on the generated media stream to try and compensate for insufficient frame rates, delays or insufficient resolution. Such processing generally involves algorithmic guess work to repair incomplete media streams on the fly through pixel or frame rate upscaling. Features typically involve visual flourishes such as sophisticated weather effects, reflections and shadows but could also include some AI-driven augmentation such as object recognition and framing to assist the end user.

In an ideal world, we may want a sub-20ms RTT, 90 FPS, 4K resolution, no need for JIT correction and full feature set enabled. In an ideal world we have unlimited resources. The application provider knows that resources are not unlimited and that networks get congested. We propose to offer the application provider the facility to specify configurations of their application that would offer acceptable, but less than ideal, Quality of Experience specifications. The objective is to allow the application to remain operational in resource contested environments. To explore this concept, we present the application provider with the facility to specify three modes of target QoE – High, Medium, Low – representing the different levels of QoE we want to be able to deliver. We will term these QModes. While the objective of QModes is to capture different levels of physical resource consumption by the application running in a virtualized environment, what constitutes a given QMode only makes sense within the context of a particular application. QModes may be differentiated for example, by the set of rendered features (e.g., accurate weather effects, reflections, shadows), by the number of simultaneously active users, the resolution and/or frame rate delivered to the HMD, or even the placement and operation of service components across the device-edge-cloud. For a given application deployed on our platform, it's QMode values map to distinct deployment configurations of the application.

In the figure below we see three different configurations of an application and the introduction of a logical switch that can chose which deployment configuration to route traffic to.

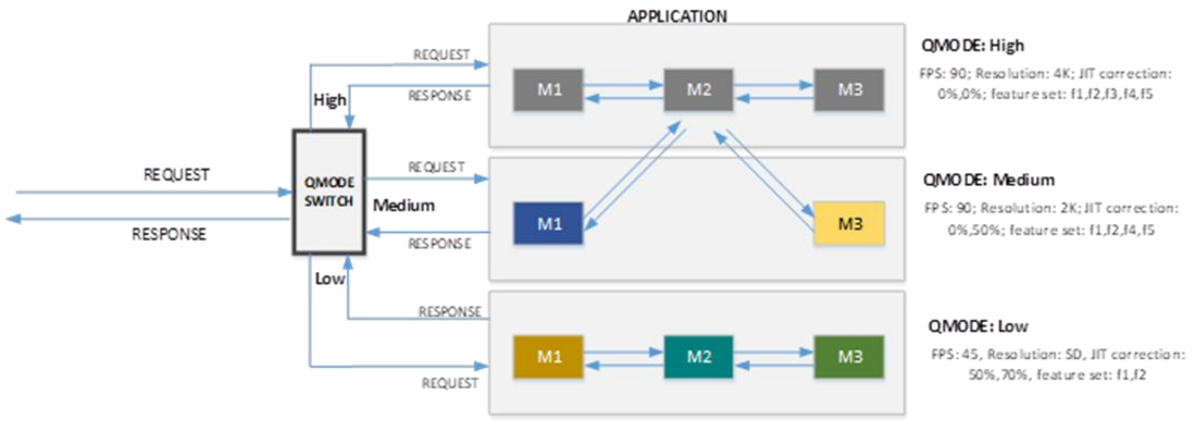

APPLICATION
REQUEST
RESPONSE
High
QMODE SWITCH
Medium
Low
M1
M2
M3
QMODE: High
FPS: 90; Resolution: 4K; JIT correction: 0%,0%; feature set: f1,f2,f3,f4,f5
QMODE: Medium
FPS: 90; Resolution: 2K; JIT correction: 0%,50%; feature set: f1,f2,f4,f5
QMODE: Low
FPS: 45, Resolution: SD, JIT correction: 50%,70%, feature set: f1,f2

Figure 22: Logical QMode Switch and how it could be employed to divert traffic between different service configurations

In the above model, we can see that a single service instance may be involved in multiple chains. We will return to this later.

A multi-user application will likely be operating in multiple QModes simultaneously. We view QMode as being tied to a particular traffic characteristic. Different users may be assigned different QModes according to their circumstances (e.g., SLA, device capabilities, local network congestion levels, etc.)

Conceptually, a QMode enables network routing in a similar fashion to a VLAN in that it allows us to segment and route traffic according to a tag.

The choice of QMode to perform at can depend on a variety of factors. Application providers may elect to differentiate based on class of device (is it capable of high resolution, does it support frame interpolation[17], etc.), speed of network, availability of edge resources, user contract, number of local active users, etc.[18]. To be able to make this choice, however, requires that we gather and monitor this information in a centralized monitoring framework.

[17] For example, SteamVR Motion Smoothing or Oculus Asynchronous Space and Time warping

[18] It is quite possible that three QModes would not be sufficient to capture the complexity of conditions and granularity of configuration options available to a given application provider. We have restricted ourselves to three modes to simplify concept evaluation and development.

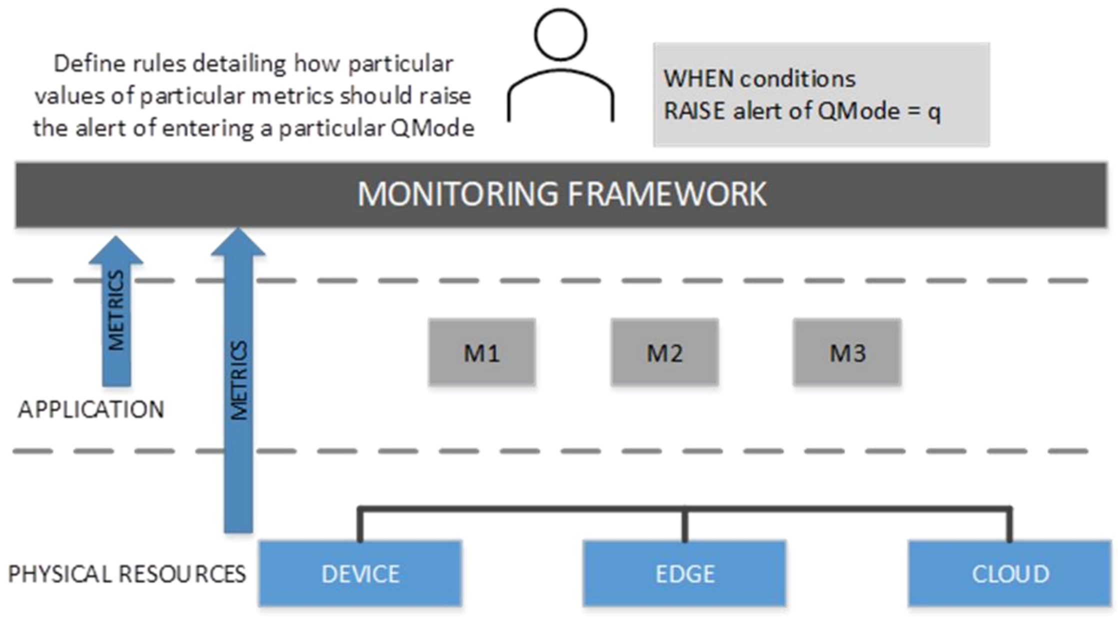


Figure 23: Monitor for conditions that warrant changes to QMode

4.3.4 Monitoring & Analysis

Adaptation requires context. The drivers for adaptation can vary according to the business and resource environment but in general, applications must adapt to resource availability. An XR application distributed across device, edge and cloud resources can depend on a delicate, geographically dispersed, web of resources. Monitoring every leveraged resource individually, seeking to detect bottlenecks and deficiencies, can overwhelm our decision making. Without intimate knowledge of an application's resourcing windows and inbuilt compensation mechanisms[19], we may elevate disparate resource stresses (such as link delays, GPU overload, database response times) to high priority problems that require countermeasures while, in fact, the application is still able to operate as a whole and deliver an acceptable quality of experience to the end user. A more sensible approach would appear to be initiating action in response to a small number of high-level red flags that holistically capture underlying problems rather than monitoring a multitude of low-level warning indicators.

The ultimate purpose of any application is to perform its work and deliver acceptable performance and experience to the end user. If the application is delivering an acceptable Quality of Experience (QoE), then we could deem the application to be performing adequately and not in need of adaptation.

[19] It may transpire, for example, that a well-resourced database equipped with advanced SSD disks can compensate for an underperforming cache relying on overly stressed RAM. Such trade-offs and compensations are generally particular to each distinct application. In addition, application providers generally dimension some latitude into their resource requirement specifications to accommodate leg room and usage peaks that may not always be used. An over-eager adaptation mechanism may seek to fix a problem that does not need fixing.

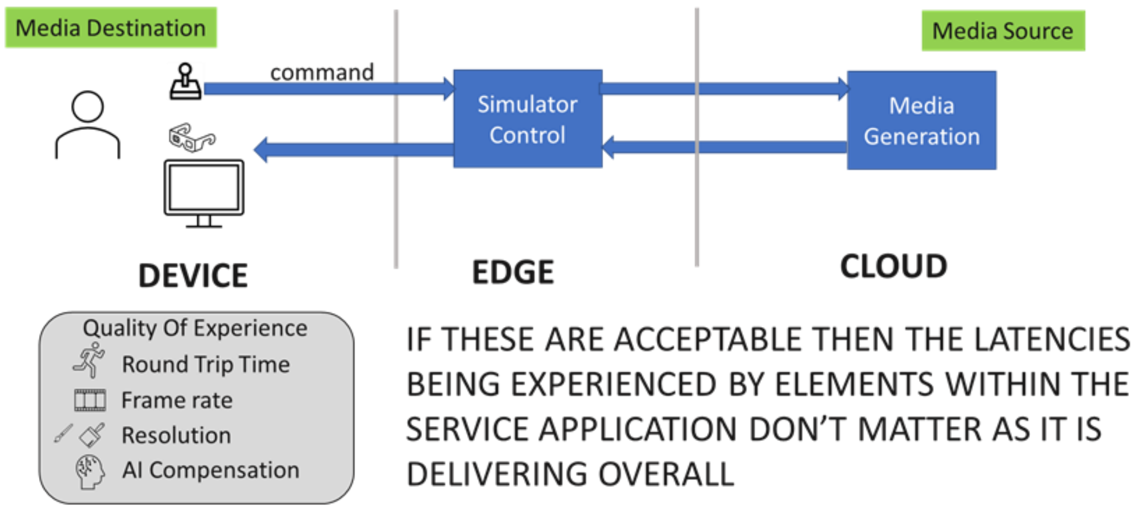


Figure 24: Monitoring High level indicators reduces decision complexity

In Figure 24 we see representative XR metrics we can monitor for conditions that capture the overall fitness for purpose of the application:

- Round Trip Time: the length of time between a user action and its reflection on the visual experience
- Frame Rate: How many frames per second we are delivering to the user device
- Resolution: the pixel depth of the frames we are delivering
- AI Compensation: Rate of interpolation/extrapolation we need to do locally to 'fix' sub-standard resolution or frame rate being delivered from the visual renderer. This may arise if a remote visual renderer generates lower quality media streams to reduce bandwidth needs from the cloud while it is up-scaled at the edge or on the device.

By monitoring these metrics, we can assess the application's fitness. Requesting the application provider to specify meaningful thresholds and operating windows for these metrics is reasonable – unlike requesting them to specify a combination of hardware resource availability deviations that could expose a problem.

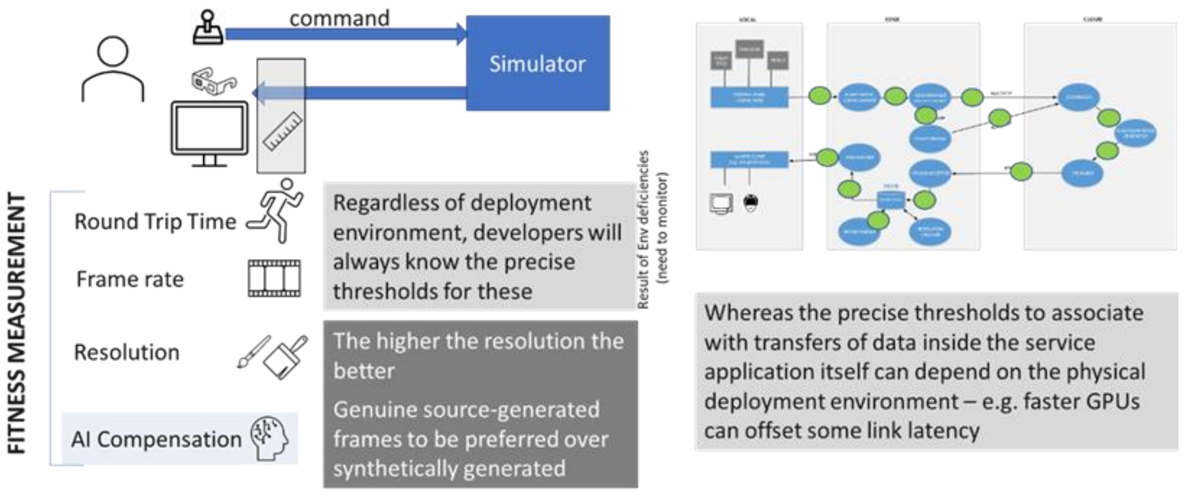

command
Simulator
FITNESS MEASUREMENT
Round Trip Time
Frame rate
Resolution
AI Compensation
Regardless of deployment environment, developers will always know the precise thresholds for these
The higher the resolution the better
Genuine source-generated frames to be preferred over synthetically generated
Result of Env deficiencies (need to monitor)
Whereas the precise thresholds to associate with transfers of data inside the service application itself can depend on the physical deployment environment – e.g. faster GPUs can offset some link latency

Figure 25: Monitoring the manifested user experience is more tractable and efficient

While unacceptable levels of application fitness may highlight a problem, high-level indicators cannot inform us what the cause of it is. They inform us *when* to investigate a manifest problem rather than necessitating constant low-level monitoring and analysis to ascertain if we can deduce a problem. Root-cause investigation requires examination of far more detailed and lower-level metrics (such as individual service performance, particular link latencies or bandwidth shortcomings, and queueing backlogs) as gathered by the CHARITY monitoring platform. The driver for this level of analysis is that applications may be adapted differently depending on the root cause of the problem. For example, a deficiency in the response time from a cloud-based service to an edge node may require different adaptation than experiencing resource stresses on the edge node itself. We seek to enable application providers to fully leverage the adaptation avenues they have available to them within their application design.

This requires us to be able to retrieve metrics relevant to the application under investigation – an application that may be operating across multiple nodes over the device-edge-cloud continuum.

In Figure 26, we see the role of monitoring in application adaptation.

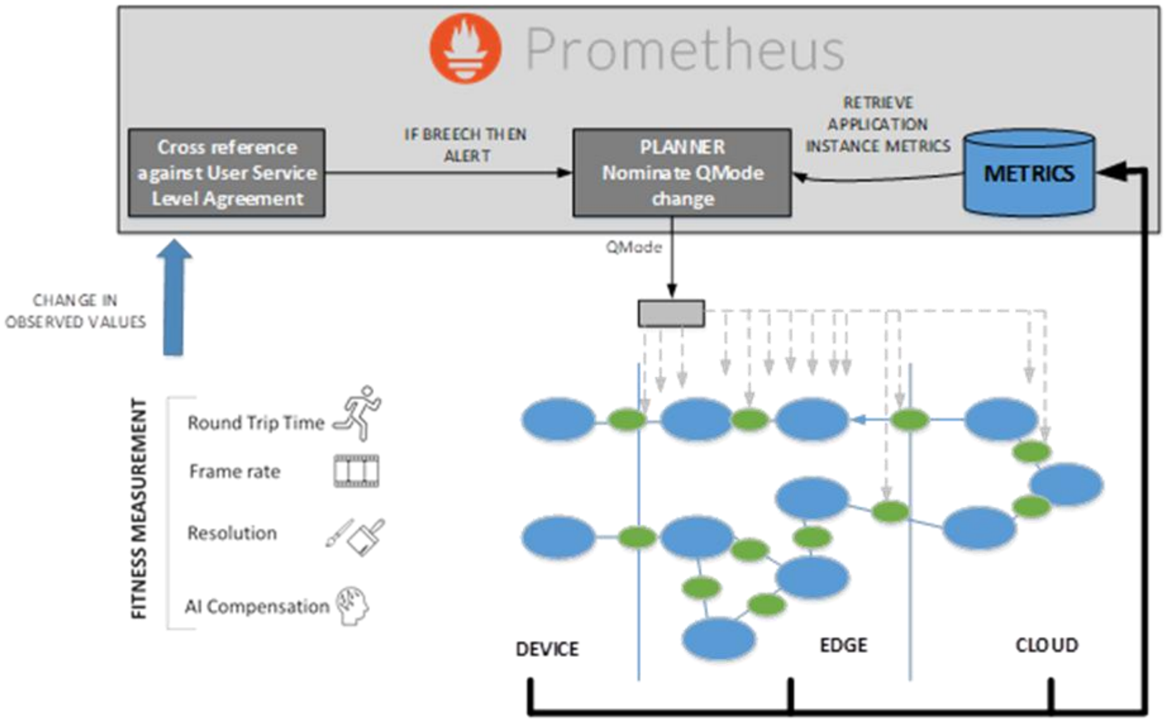

Prometheus
Cross reference against User Service Level Agreement
IF BREECH THEN ALERT
PLANNER Nominate QMode change
RETRIEVE APPLICATION INSTANCE METRICS
METRICS
QMode
CHANGE IN OBSERVED VALUES
FITNESS MEASUREMENT
Round Trip Time
Frame rate
Resolution
AI Compensation
DEVICE
EDGE
CLOUD

Figure 26: Monitoring high level metrics while supporting interrogation of low-level for adaptation

We propose to use the Prometheus and its Alert Manager to trigger examinations of lower-level metrics when SLA-breaking conditions are observed with higher-level *Fitness Measurements*. The actual mechanics of how QMode updates are relayed to the Service Mesh is still under investigation and is a topic we will return to when discussing Early Investigative Work later in this section.

4.3.5 Planning & Execution

When analysis of ongoing monitoring reveals the occurrence of conditions warranting a QMode change then an alert is raised and relayed to the Prometheus Alert Manager. The Alert Manager in turn publishes the alert.

The logical QMode Switch we referred to earlier routes to a particular application configuration based on the current value for the QMode associated with the application. Applied at the global system level, this would have a sledgehammer effect. We need to be lighter handed and enable QMode changes to apply to a subset of user sessions.

4.3.5.1 User-Level routing

To support user/session level granularity then the switch needs awareness about the user associated with a given request.

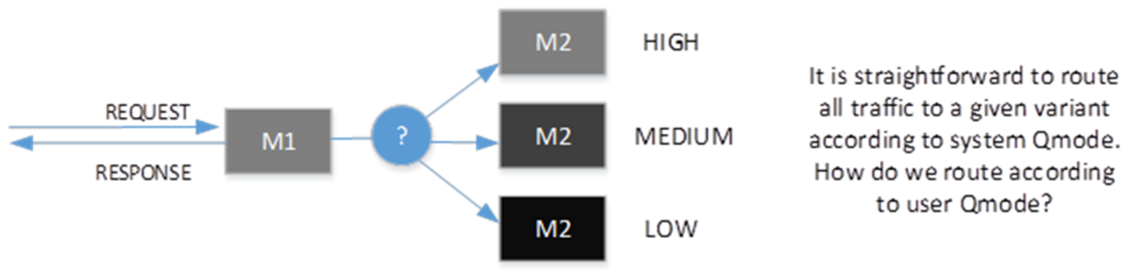

REQUEST
RESPONSE
M1
?
M2
HIGH
M2
MEDIUM
M2
LOW
It is straightforward to route all traffic to a given variant according to system Qmode. How do we route according to user Qmode?

Figure 27: QMode Routing

The proposed solution lies in associating a QMode tag with each user session and having a particular application configuration to be employed for a given QMode tag. Adapting an application fundamentally entails instantiating a variant of the application, having it run side by side with the original while it prepares itself to accept traffic, and then switching live traffic to the variant so that we can retire the original. Below we depict a snapshot in time when it has been decided to swap the user to a lower-resource-consuming variation of the application and are ready to switch the traffic over.

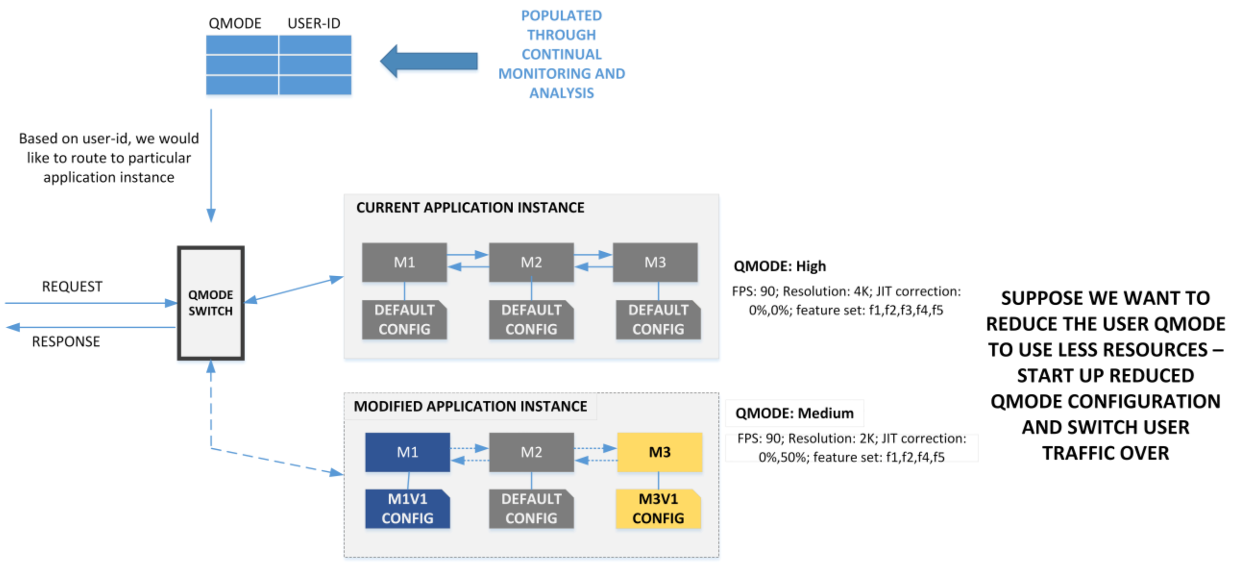

QMODE
USER-ID
POPULATED THROUGH CONTINUAL MONITORING AND ANALYSIS
Based on user-id, we would like to route to particular application instance
REQUEST
RESPONSE
QMODE SWITCH
CURRENT APPLICATION INSTANCE
M1
M2
M3
DEFAULT CONFIG
DEFAULT CONFIG
DEFAULT CONFIG
QMODE: High
FPS: 90; Resolution: 4K; JIT correction: 0%,0%; feature set: f1,f2,f3,f4,f5
MODIFIED APPLICATION INSTANCE
M1
M2
M3
M1V1 CONFIG
DEFAULT CONFIG
M3V1 CONFIG
QMODE: Medium
FPS: 90; Resolution: 2K; JIT correction: 0%,50%; feature set: f1,f2,f4,f5
SUPPOSE WE WANT TO REDUCE THE USER QMODE TO USE LESS RESOURCES – START UP REDUCED QMODE CONFIGURATION AND SWITCH USER TRAFFIC OVER

Figure 28: Application about to switch over to application variation that consumes less resources

Note in the scenario depicted above that not all services in the application are reconfigured. We can see that M2 is unchanged. There should be no need to start another copy of M2 and we should keep using the already running M2. The configurations of M1 and M3 have changed however and new instances of these services will be started and introduced.

4.3.5.2 QMode Switching

As introduced earlier when discussing Service Routing, we propose to use a Service Mesh for managing unobtrusive intra-application routing changes. Each service is deployed into a Kubernetes Pod along with a sidecar proxy that mediates all network traffic. Pods are tagged with the QModes that they support. In Figure 28 we depicted how our application instance was modified through reconfiguration of Services M1 (moving from Default Config to M1V1 Config) and M3 (moving from default to M3V1 config). Table 16 summarizes the necessary reconfiguration required within our application to target differing QModes.

This Service	Should be using this Configuration	When we seek to offer this QMode
M1	DEFAULT	HIGH
	M1V1	MEDIUM
M2	DEFAULT	HIGH
		MEDIUM
M3	DEFAULT	HIGH
	M3V1	MEDIUM

Table 16: Configuration changes mapping to QMode targets

In Figure 29 the corresponding Kubernetes Pod layout in which we capture the point in time at which Pods have been launched to support both QModes and the sidecar proxies within each Pod select the next Pod to route based on the current value of QMode associated with the client and the tags attached to the Pods.

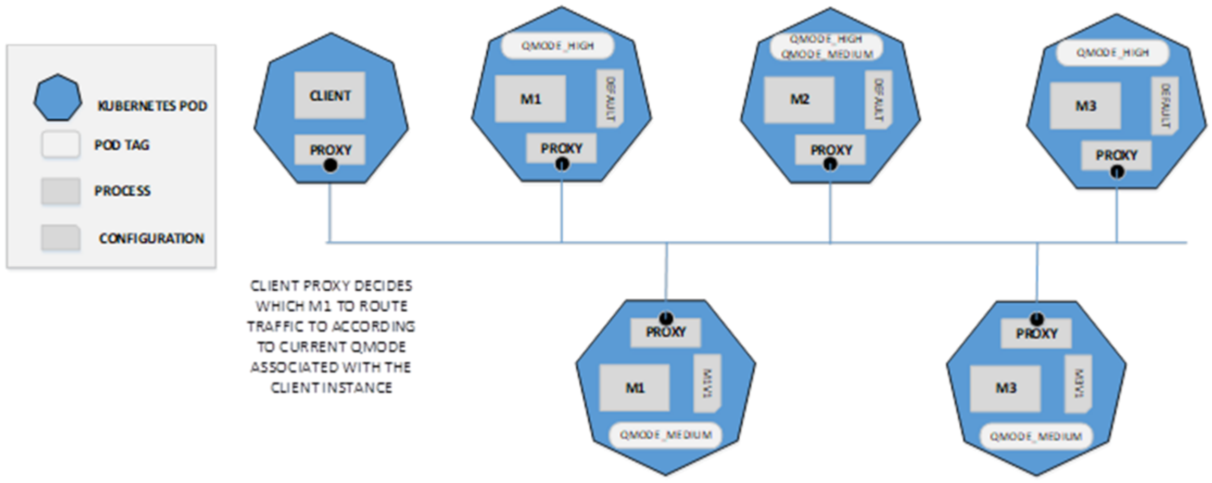


Figure 29: Kubernetes Pods manage service variations and are threaded together through tagging and sidecar proxies

4.4 Early Investigative Work

4.4.1 Service Mesh Routing

The sidecar proxies involved in a service chain need to have a common view of the current QMode value.

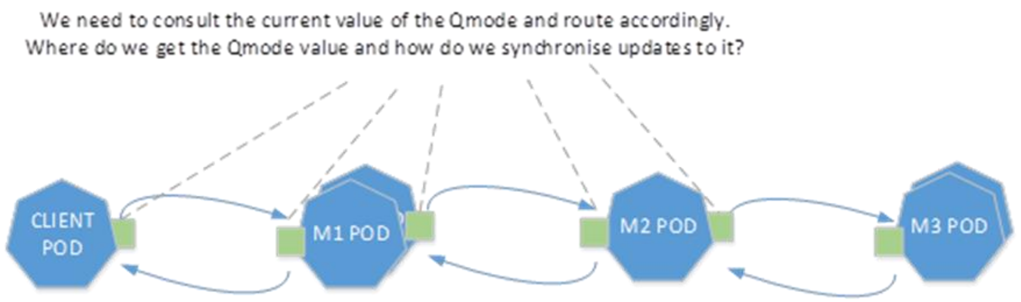

We need to consult the current value of the Qmode and route accordingly.
Where do we get the Qmode value and how do we synchronise updates to it?
CLIENT POD
M1 POD
M2 POD
M3 POD

Figure 30: QMode synchronization and propagation

Initial approaches involved injecting the tag at the client ingress point and then piggybacking this tag along the service chain. While promising in a purely cloud native scenario in which all inter-service communication occurs using http, it became increasingly clear that the approach would prove lacking when having to deal with a wider range of communication protocols and payload formats. Subsequent approaches moved away from the piggyback approach to one in which the sidecars could reference (or be updated by) an external service or repository from which it could obtain the current QMode value.

Various strategies for injecting the QMode value have been considered:

1. Istio header-based routing
2. Envoy header-to-metadata filter
3. Custom Envoy filters
4. External Routing Logic
5. WASM Plugins

4.4.1.1 Istio Solution

This approach utilizes Header-based routing, which entails defining routing rules to be activated depending on specific field values in a request's header. Several Istio resources can be used to implement such a solution. A **VirtualService** is applied to route traffic to different subsets that match a value in the **quality** tag in the request header. A **DestinationRule** is used to define a subset for each configuration/version of each application.

With an intuitive configuration scheme, this solution enjoys the benefit of being straightforward for providers to deploy applications. This solution is also manageable in large-scale scenarios as scaling doesn't require any kind of canary deployments.

This approach was evaluated with a rudimentary scenario in a Kubernetes environment, which contains:

- A web server (**httpbin**)
    - **httpbin-high** – version optimized for high resource availability
    - **httpbin-low** – version optimized for low resource availability
- A web client (**sleep**)

httpbin is deployed as a single service but with two different instances, one for each configuration. Each instance is configured to have a *quality label* in the metadata field. This label will later be used by Istio's resource to define the routing rules. The **sleep** application will be deployed with no special configuration. Subsequently, a **VirtualService** is configured to define routes to each version of the **httpbin**, based on a **quality** field set in the request's header. The **VirtualService** will match the preset routing rules to the value in the **quality** tag, and thus route traffic to a defined subset. This subset is defined in the **DestinationRule**, which declares two subsets for the **httpbin**service: **high** and **low**. To each subset, a label is assigned, which is the same label assigned to each **httpbin** configuration. Therefore, this collection of resources allows for traffic to be routed to the ***high*** version of the **httpbin** service when the request is tagged with the ***high-***quality value and the same goes for the ***low*** version.

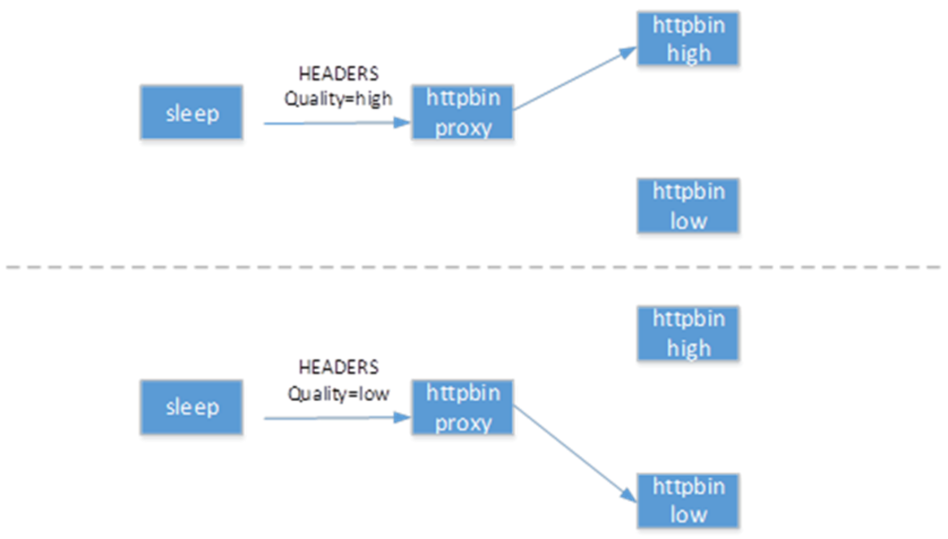


Figure 31: Header-based routing

As seen in the figure above, both versions of the same application co-exist, but traffic is routed to each of them based on the value of the **quality** tag in the request's header. This is configured in the **VirtualService** and **DestinationRule** resources. This solution is not restricted to only two routes or two versions of a single application and is extensible to any number.

This form of routing algorithm can be coupled to a more intelligent system that is responsible for performing health checks, hence approximating a load balancer's behaviour. Since these routing rules are unchangeable, traffic tagged with the **low** value must always be routed to the **low** version of an application, load balancing does not need to be considered in this implementation. The intelligent load balancing component is the one tasked with injecting the **quality** header, that will later be used for header-based routing.

Initial experimentations included the creation of dummy services, that act as a front for real services and enable the routing traffic according to tag value. The following scenario was implemented:

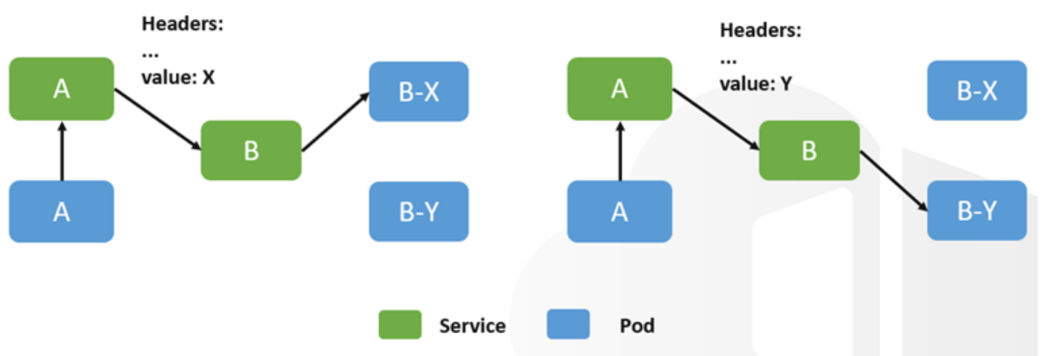
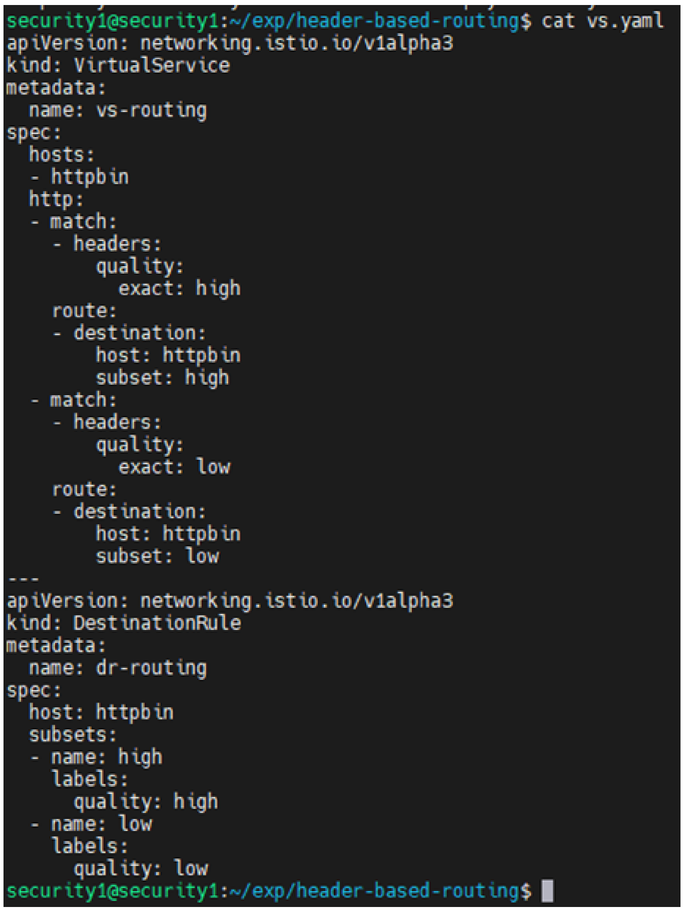


Figure 32: Scenario implemented with Istio header tagging

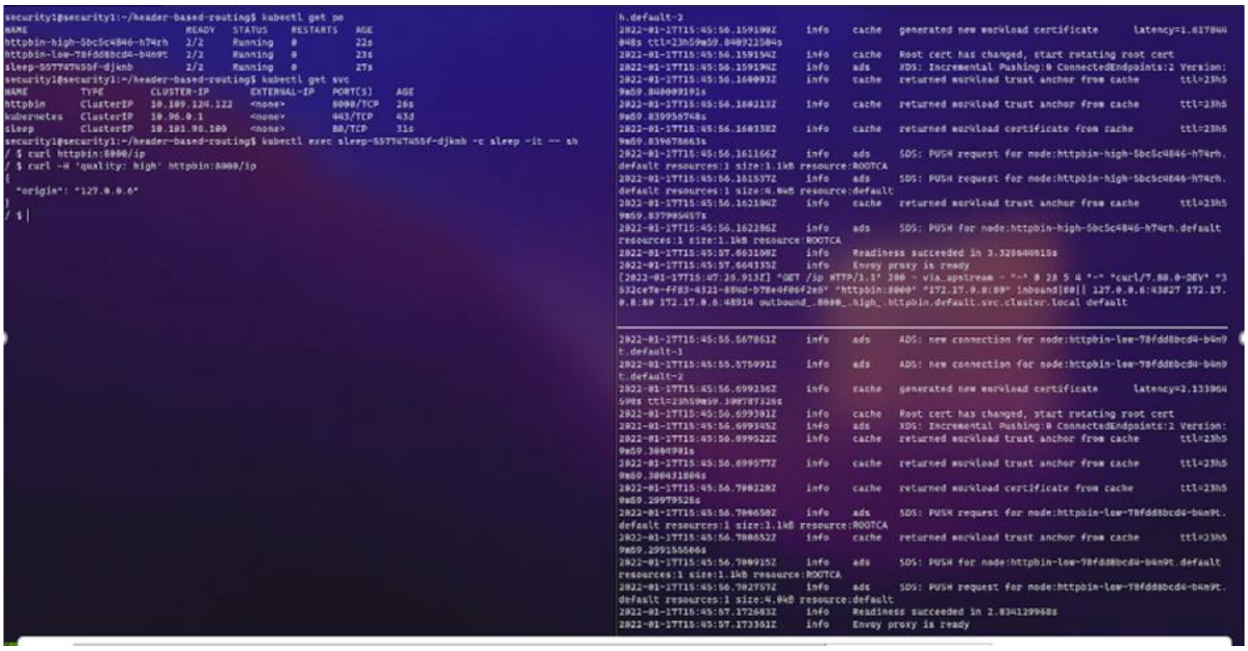
In this type of architecture, service B will be able to route both pods B-X and B-Y, just depending on the value of the tag/header. Each is configured for different types of conditions (high and low resource levels), and the value on the header of the request performed by service is used by a **VirtualService** to route to subset high -> B-X or to subset low -> B-Y.



```
security1@security1:~/exp/header-based-routing$ cat vs.yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: vs-routing
spec:
  hosts:
  - httpbin
  http:
  - match:
    - headers:
        quality:
          exact: high
    route:
    - destination:
        host: httpbin
        subset: high
  - match:
    - headers:
        quality:
          exact: low
    route:
    - destination:
        host: httpbin
        subset: low
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: dr-routing
spec:
  host: httpbin
  subsets:
  - name: high
    labels:
      quality: high
  - name: low
    labels:
      quality: low
security1@security1:~/exp/header-based-routing$ 
```

Figure 33: Configuration of routing logic

Upon applying this configuration, we can see that traffic does indeed get routed according to the tag value as captured below in Figure 34.

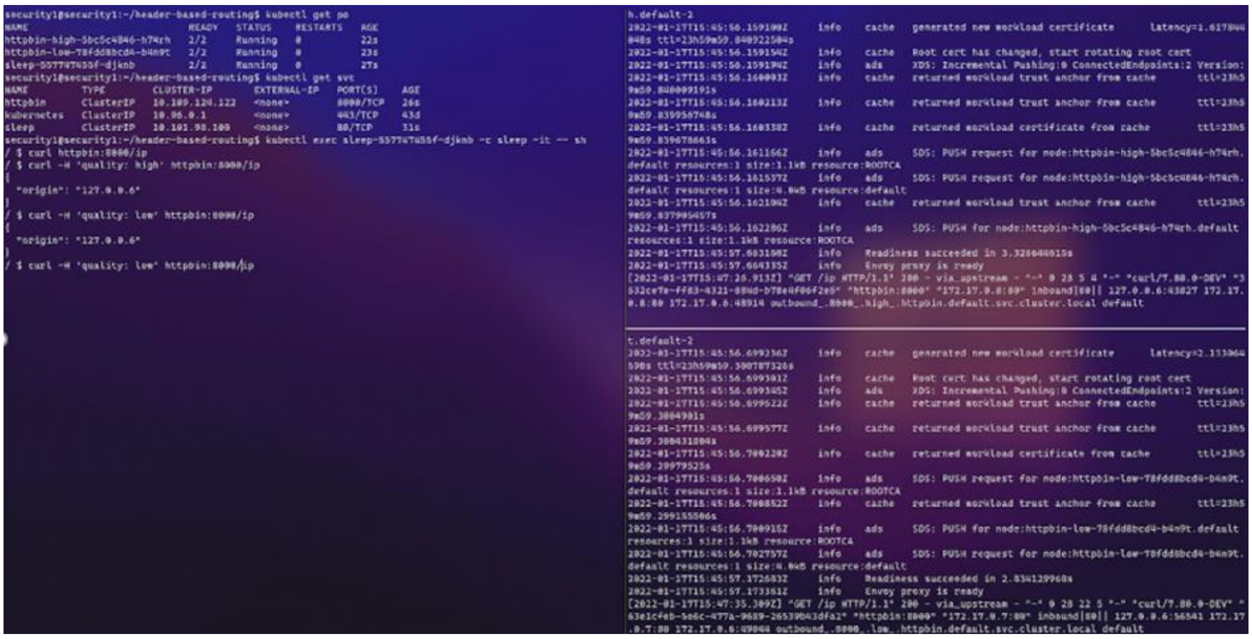


```
security1@security1:~/header-based-routing$ kubectl get po
NAME                              READY   STATUS    RESTARTS   AGE
httpbin-high-5bc5c4846-h74rh      2/2     Running   0          22s
httpbin-low-78fdd8bcd4-b4n9t      2/2     Running   0          23s
sleep-557747455f-djknb            2/2     Running   0          27s
security1@security1:~/header-based-routing$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
httpbin      ClusterIP   10.109.124.122   <none>        8000/TCP   26s
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP    43d
sleep        ClusterIP   10.101.98.100    <none>        80/TCP     31s
security1@security1:~/header-based-routing$ kubectl exec sleep-557747455f-djknb -c sleep -it -- sh
/ $ curl httpbin:8000/ip
/ $ curl -H 'quality: high' httpbin:8000/ip
{
  "origin": "127.0.0.6"
}
/ $ |
```

```
h.default-2
2022-01-17T15:45:56.159100Z     info    cache   generated new workload certificate      latency=1.617044
048s ttl=23h59m59.840922504s
2022-01-17T15:45:56.159154Z     info    cache   Root cert has changed, start rotating root cert
2022-01-17T15:45:56.159194Z     info    ads     XDS: Incremental Pushing:0 ConnectedEndpoints:2 Version:
2022-01-17T15:45:56.160093Z     info    cache   returned workload trust anchor from cache       ttl=23h5
9m59.840009191s
2022-01-17T15:45:56.160213Z     info    cache   returned workload trust anchor from cache       ttl=23h5
9m59.839956748s
2022-01-17T15:45:56.160338Z     info    cache   returned workload certificate from cache        ttl=23h5
9m59.839678663s
2022-01-17T15:45:56.161166Z     info    ads     SDS: PUSH request for node:httpbin-high-5bc5c4846-h74rh.
default resources:1 size:1.1kB resource:ROOTCA
2022-01-17T15:45:56.161537Z     info    ads     SDS: PUSH request for node:httpbin-high-5bc5c4846-h74rh.
default resources:1 size:4.0kB resource:default
2022-01-17T15:45:56.162104Z     info    cache   returned workload trust anchor from cache       ttl=23h5
9m59.837905457s
2022-01-17T15:45:56.162286Z     info    ads     SDS: PUSH for node:httpbin-high-5bc5c4846-h74rh.default
resources:1 size:1.1kB resource:ROOTCA
2022-01-17T15:45:57.663108Z     info    Readiness succeeded in 3.328644015s
2022-01-17T15:45:57.664335Z     info    Envoy proxy is ready
[2022-01-17T15:47:26.913Z] "GET /ip HTTP/1.1" 200 - via_upstream - "-" 0 28 5 4 "-" "curl/7.80.0-DEV" "3
532ce7e-ff83-4321-80ad-b78e0f06f2e6" "httpbin:8000" "172.17.0.8:80" inbound|80|| 127.0.0.6:43827 172.17.
0.8:80 172.17.0.6:48914 outbound_.8000_.high_.httpbin.default.svc.cluster.local default

2022-01-17T15:45:55.567861Z     info    ads     ADS: new connection for node:httpbin-low-78fdd8bcd4-b4n9
t.default-1
2022-01-17T15:45:55.575991Z     info    ads     ADS: new connection for node:httpbin-low-78fdd8bcd4-b4n9
t.default-2
2022-01-17T15:45:56.699236Z     info    cache   generated new workload certificate      latency=2.133064
598s ttl=23h59m59.300787326s
2022-01-17T15:45:56.699301Z     info    cache   Root cert has changed, start rotating root cert
2022-01-17T15:45:56.699345Z     info    ads     XDS: Incremental Pushing:0 ConnectedEndpoints:2 Version:
2022-01-17T15:45:56.699522Z     info    cache   returned workload trust anchor from cache       ttl=23h5
9m59.30049901s
2022-01-17T15:45:56.699577Z     info    cache   returned workload trust anchor from cache       ttl=23h5
9m59.300431804s
2022-01-17T15:45:56.700220Z     info    cache   returned workload certificate from cache        ttl=23h5
9m59.29979525s
2022-01-17T15:45:56.700650Z     info    ads     SDS: PUSH request for node:httpbin-low-78fdd8bcd4-b4n9t.
default resources:1 size:1.1kB resource:ROOTCA
2022-01-17T15:45:56.700852Z     info    cache   returned workload trust anchor from cache       ttl=23h5
9m59.299155506s
2022-01-17T15:45:56.700915Z     info    ads     SDS: PUSH for node:httpbin-low-78fdd8bcd4-b4n9t.default
resources:1 size:1.1kB resource:ROOTCA
2022-01-17T15:45:56.702757Z     info    ads     SDS: PUSH request for node:httpbin-low-78fdd8bcd4-b4n9t.
default resources:1 size:4.0kB resource:default
2022-01-17T15:45:57.172683Z     info    Readiness succeeded in 2.834129968s
2022-01-17T15:45:57.173361Z     info    Envoy proxy is ready
```

Figure 34: Communication with the httpbin service - high version of the app



```
security1@security1:~/header-based-routing$ kubectl get po
NAME                              READY   STATUS    RESTARTS   AGE
httpbin-high-5bc5c4846-h74rh      2/2     Running   0          22s
httpbin-low-78fdd8bcd4-b4n9t      2/2     Running   0          23s
sleep-557747455f-djknb            2/2     Running   0          27s
security1@security1:~/header-based-routing$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
httpbin      ClusterIP   10.109.124.122   <none>        8000/TCP   26s
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP    43d
sleep        ClusterIP   10.101.98.109    <none>        80/TCP     31s
security1@security1:~/header-based-routing$ kubectl exec sleep-557747455f-djknb -c sleep -it -- sh
/ $ curl httpbin:8000/ip
/ $ curl -H 'quality: high' httpbin:8000/ip
{
  "origin": "127.0.0.6"
}
/ $ curl -H 'quality: low' httpbin:8000/ip
{
  "origin": "127.0.0.6"
}
/ $ curl -H 'quality: low' httpbin:8000/ip
```

```
h.default-2
2022-01-17T15:45:56.159100Z     info    cache   generated new workload certificate      latency=1.617844
048s ttl=23h59m59.840922504s
2022-01-17T15:45:56.159154Z     info    cache   Root cert has changed, start rotating root cert
2022-01-17T15:45:56.159194Z     info    ads     XDS: Incremental Pushing:0 ConnectedEndpoints:2 Version:
2022-01-17T15:45:56.160093Z     info    cache   returned workload trust anchor from cache       ttl=23h5
9m59.840009191s
2022-01-17T15:45:56.160213Z     info    cache   returned workload trust anchor from cache       ttl=23h5
9m59.839956748s
2022-01-17T15:45:56.160338Z     info    cache   returned workload certificate from cache        ttl=23h5
9m59.839678663s
2022-01-17T15:45:56.161166Z     info    ads     SDS: PUSH request for node:httpbin-high-5bc5c4846-h74rh.
default resources:1 size:1.1kB resource:ROOTCA
2022-01-17T15:45:56.161537Z     info    ads     SDS: PUSH request for node:httpbin-high-5bc5c4846-h74rh.
default resources:1 size:4.0kB resource:default
2022-01-17T15:45:56.162100Z     info    cache   returned workload trust anchor from cache       ttl=23h5
9m59.837905457s
2022-01-17T15:45:56.162286Z     info    ads     SDS: PUSH for node:httpbin-high-5bc5c4846-h74rh.default
resources:1 size:1.1kB resource:ROOTCA
2022-01-17T15:45:57.663100Z     info    Readiness succeeded in 3.326044015s
2022-01-17T15:45:57.664335Z     info    Envoy proxy is ready
[2022-01-17T15:47:26.913Z] "GET /ip HTTP/1.1" 200 - via_upstream - "-" 0 28 5 4 "-" "curl/7.80.0-DEV" "3
532ce7e-ff83-4321-884d-b78e4f06f2e5" "httpbin:8000" "172.17.0.8:80" inbound|80|| 127.0.0.6:43827 172.17.
0.8:80 172.17.0.6:48914 outbound_.8000_.high_.httpbin.default.svc.cluster.local default

t.default-2
2022-01-17T15:45:56.699236Z     info    cache   generated new workload certificate      latency=2.133064
590s ttl=23h59m59.300787326s
2022-01-17T15:45:56.699301Z     info    cache   Root cert has changed, start rotating root cert
2022-01-17T15:45:56.699345Z     info    ads     XDS: Incremental Pushing:0 ConnectedEndpoints:2 Version:
2022-01-17T15:45:56.699522Z     info    cache   returned workload trust anchor from cache       ttl=23h5
9m59.3004981s
2022-01-17T15:45:56.699577Z     info    cache   returned workload trust anchor from cache       ttl=23h5
9m59.300421804s
2022-01-17T15:45:56.700220Z     info    cache   returned workload certificate from cache        ttl=23h5
9m59.299795525s
2022-01-17T15:45:56.700650Z     info    ads     SDS: PUSH request for node:httpbin-low-78fdd8bcd4-b4n9t.
default resources:1 size:1.1kB resource:ROOTCA
2022-01-17T15:45:56.700852Z     info    cache   returned workload trust anchor from cache       ttl=23h5
9m59.299155506s
2022-01-17T15:45:56.700915Z     info    ads     SDS: PUSH for node:httpbin-low-78fdd8bcd4-b4n9t.default
resources:1 size:1.1kB resource:ROOTCA
2022-01-17T15:45:56.702757Z     info    ads     SDS: PUSH request for node:httpbin-low-78fdd8bcd4-b4n9t.
default resources:1 size:4.0kB resource:default
2022-01-17T15:45:57.172683Z     info    Readiness succeeded in 2.834129960s
2022-01-17T15:45:57.173361Z     info    Envoy proxy is ready
[2022-01-17T15:47:35.309Z] "GET /ip HTTP/1.1" 200 - via_upstream - "-" 0 28 22 5 "-" "curl/7.80.0-DEV" "
63e1cfeb-5e6c-477a-9689-26539b43dfa2" "httpbin:8000" "172.17.0.7:80" inbound|80|| 127.0.0.6:56541 172.17
.0.7:80 172.17.0.6:49844 outbound_.8000_.low_.httpbin.default.svc.cluster.local default
```

Figure 35: Communication with the httpbin service - low version of the app

Figure 34 and Figure 35 show the results of the experiment. As you can see, when inserting the tag "quality: low", the traffic is forwarded to the low version of the app, and when inserting the tag "quality: high", traffic is forwarded to the high version.

4.4.1.2 Standard Envoy Filters

Early investigations explored using standard Envoy filters used to perform Load Balancing. Istio's EnvoyFilter resources allow one to modify the configuration of the Istio proxy sidecars (Envoy). We can use Envoy's Header to Metadata filter, which enables us to extract values from HTTP headers and attach them to a request's dynamic metadata, which in turn can be used for matching endpoints.

To accomplish this, we inject a header "quality" attribute into incoming HTTP requests. When traversing proxies, traffic will be routed to hosts whose associated metadata matches the value of the quality header, which is extracted and attached as request metadata. Hosts will be manually configured with specific metadata, in a sense, with the key-value pair: <quality, value>.

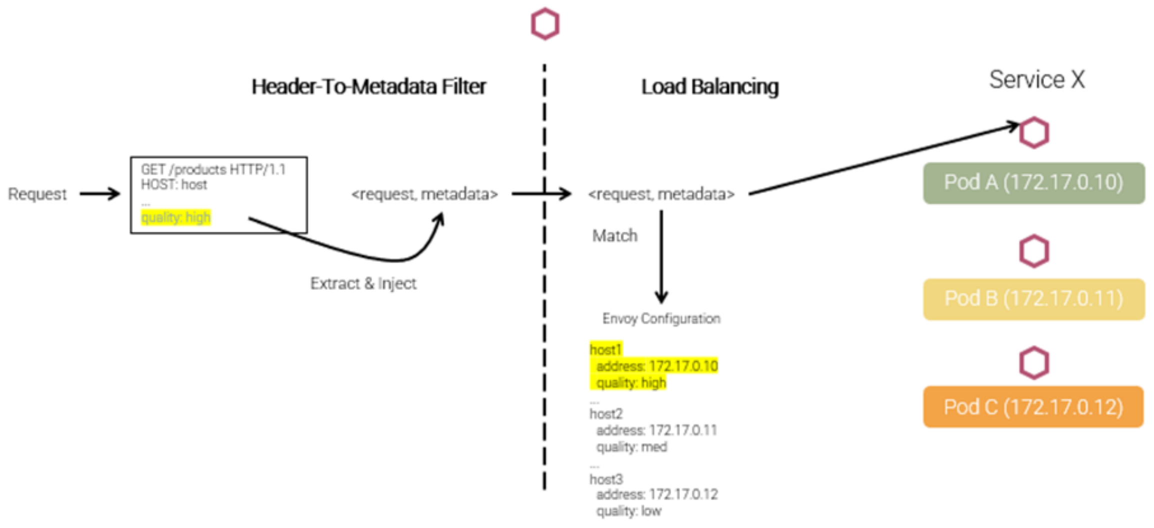


Figure 36: Leveraging Envoy filters and Load Balancing functionality

A disadvantage of this approach is that it will not support dynamic traffic distribution schemes. It requires defining a set of routing rules or load balancing decisions, and therefore, configuration is hardcoded and static. In a sense, it's only dynamic because traffic is not known until the tag is injected. If a particular application modifies the number of supported configurations, a new host must be configured along with its metadata. This further adds complexity and suffers from scalability limitations.

If we also consider that routes can be set dynamically (not chosen from a preconfigured list), we must designate that task to a component of the framework. This requires support of the scenario in which hosted applications interact with the mesh configuration, which is not only increasingly complex, but also raises clear security concerns.

This approach considers HTTP traffic only, not addressing the several other communication protocols commonly used within applications. Furthermore, we must not forget that Envoy may not provide configuration APIs for a range of protocols. As such, additional techniques and filters are being researched to assure compatibility with every protocol and to be used in conjunction with the filter Header-To-Metadata.

4.4.1.3 Custom Envoy Filters

This approach follows the same principle as the previous one but using custom, user-defined filters. This entails developing a filter that performs tag extraction in our specific, tailored fashion. This suffers from the same disadvantages as the standard Envoy filters discussed in the previous section but does, however, remedy one of its problems - the lack of a protocol-agnostic approach. With this approach, the custom filter can be developed in such a way that the tag extraction is performed differently with respect to the communication protocol. In this manner, we would only need this one filter for every situation, and not need to concern ourselves about compatibility issues and working with multiple features for the same task. We could avoid over-reliance on Envoy's out-of-the-box capabilities.

Integrating a custom filter requires close integration into the Envoy codebase and this is problematic. We would be burdened with maintaining a close watch on future Envoy releases to ensure that our own filter remains compatible and functional.

4.4.1.4 External Routing Logic

An external routing logic approach consists of delegating load balancing decisions. This approach takes inspiration in the way OPA[20] is built, in which policy enforcement decisions are transferred from Envoy (and Istio) to an external entity. An external component could therefore be tasked with extracting a tag from ongoing traffic and possibly making a routing decision with respect to an internal list of hosts and respective metadata. This decision would be relayed back to the proxy and load balancing performed.

This approach is summarized below and is still at an early stage of analysis. A number of variations are being considered.

[20] Open Policy Agent – opensource framework for managing policies in a cloud native architecture – openpolicyagent.org

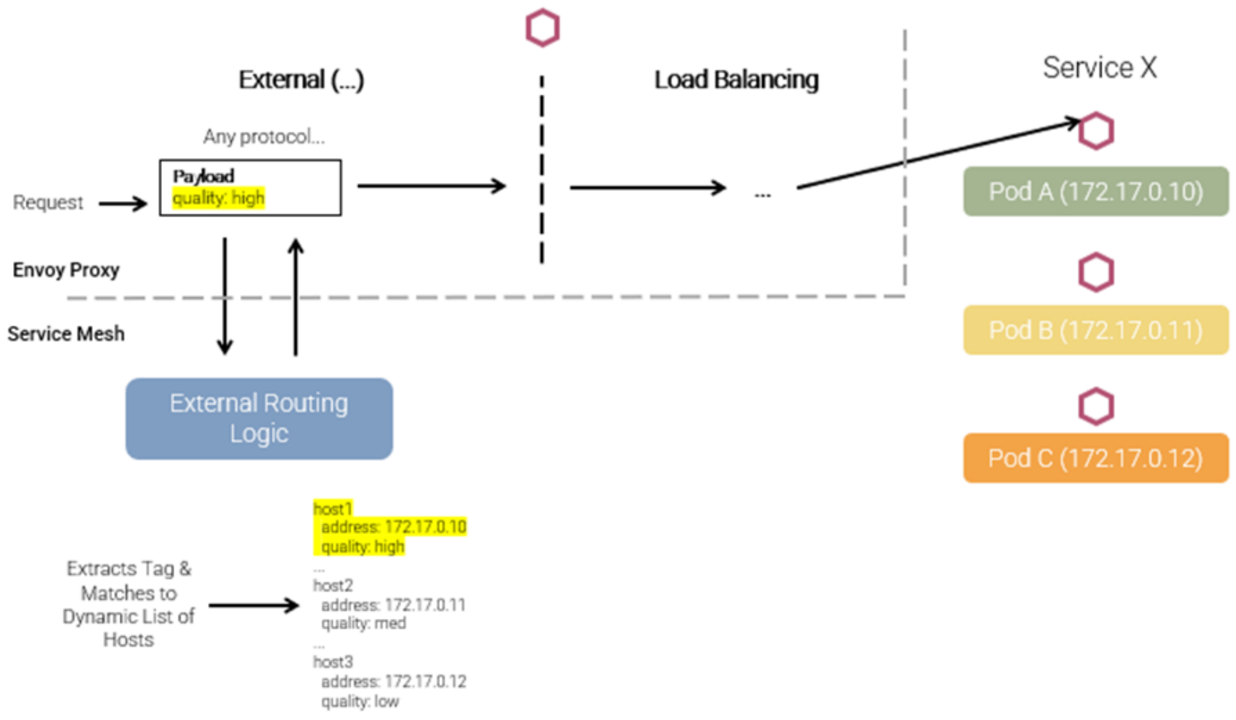


Figure 37: Delegating decisions from proxies to external service

4.4.1.5 WASM Plugins

This approach consists of utilizing a Web Assembly plugin to perform the routing logic. WASM plugins allow us to extend Istio proxies' (Envoy) capabilities. The core idea is to implement a plugin to extract the tag from ongoing traffic and in some way perform load balancing upon the extracted tag. Furthermore, it can also be considered as the underlying technology of the mechanism for injecting quality-regarding tags into the traffic payload. Initially, we started by implementing a WASM plugin for intercepting traffic and displaying information about it, and are currently investigating new iterations to this component.

This approach is very similar to the custom envoy filter, although they differ in a very important aspect - dependencies. Because the WASM plugin does not depend on the Envoy source code, it's completely independent and its development lifecycle will not interfere with Envoy's release lifecycle. We also don't have to worry about obscure configuration aspects and are properly decoupling responsibilities and concerns.

4.4.1.6 Comments on current progress

Much of our thinking has been formed around the principle of injecting some form of metadata into the client request and propagate this metadata through the service chain. This is conceptually rather clean as we avoid every proxy having to lookup some external source to check the QMode and we are guaranteed all services in a call chain for a given request see the same QMode setting.

An alternative approach is also under consideration in which sidecar proxies act as QMode change consumers to a single Source of Truth associated with the user session a given application is operating in. In this model, proxies would subscribe to QMode changes. We avoid the propagation of metadata and, potentially, examination of protocol traffic leading to a more flexible and protocol agnostic approach. This approach entails some additional complexities (e.g., filters run on demand instead of constantly, where do we persist QMode value to avoid constant consults to the Source of Truth for every message, how does this work in a distributed topology).

Recent investigations carried out with the approaches of external routing logic and WASM plugins provide a highly promising avenue to pursue this model and work is continuing in this direction.

4.4.2 Adaptation Tactics

To tease out the kind of adaptivity space that may be available to an unmodified application we examined a use case in depth – the Collins Aerospace Flight Simulator (UC3-2 Manned-Unmanned Operations Trainer Application). We sought to discover whether purely configurational changes could be identified that would equip us with tactics that we could bring into play to deal with resource deficiencies observed from resource monitoring (see Figure 38).

We can observe how the same tactic can be used in multiple scenarios. Tactics 1 and 2, for example, can be brought into play if we need to reduce bandwidth needs between the edge and cloud or free up compute resources on the cloud.

It became clear during analysis that changing the configuration of one service regularly requires changes to others to compensate or adapt to the new execution landscape. We cannot just lower the resolution generated on the cloud in isolation as the end user would experience a catastrophic drop in their Quality of Experience. We must simultaneously enable resolution upscaling on the edge on compensate. We see this need to deal with collateral effects of changes to how a single service operates repeated elsewhere.

Naturally, not all applications can lower their resolution on the cloud and have the necessary allowances in their design to compensate through upscaling elsewhere. Indeed, we expect other applications to have opportunities not offered by the flight simulator and we expect further insights into adaptability tactics as we continue our investigations with other use cases.

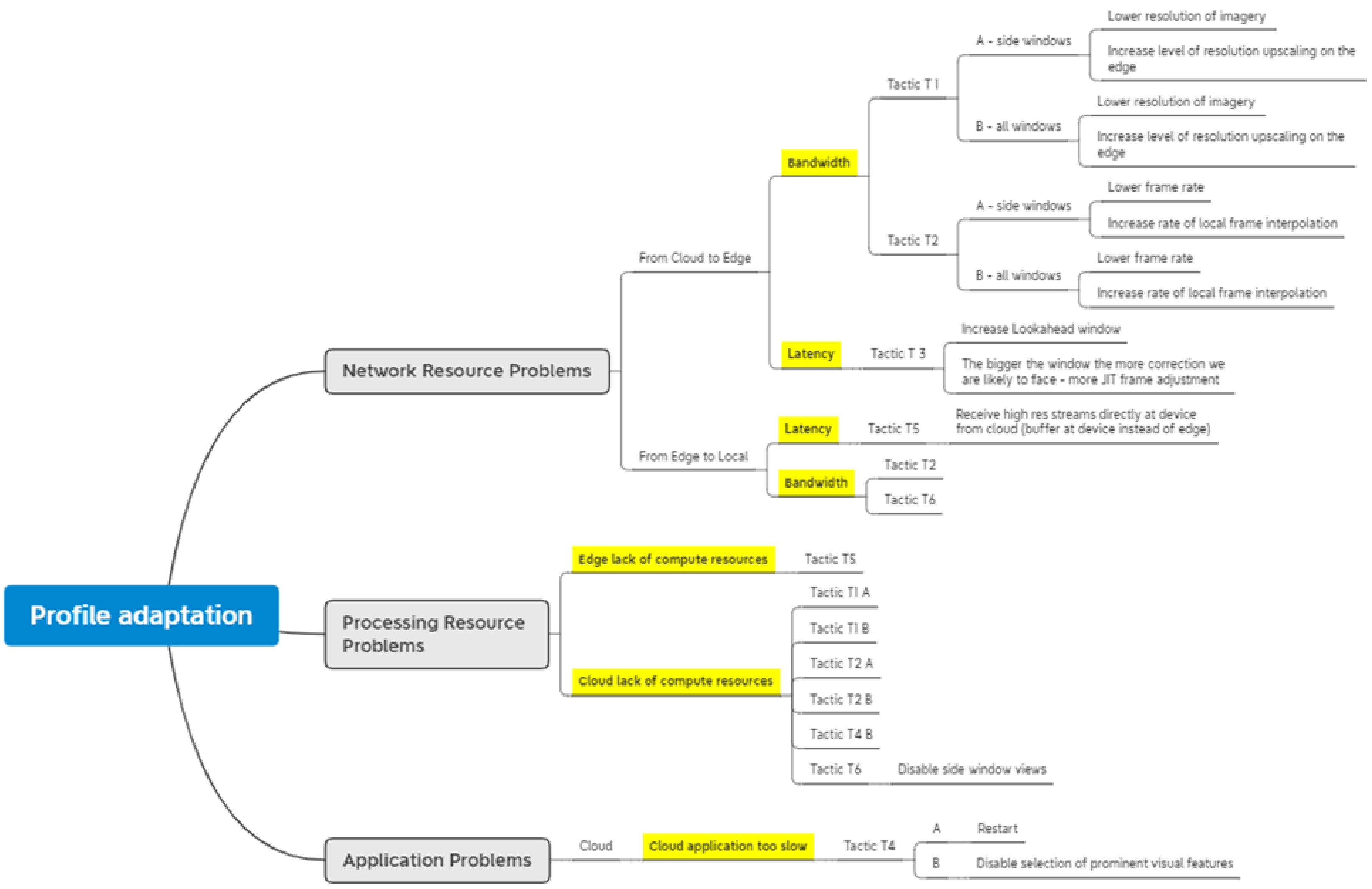

Profile adaptation
Network Resource Problems
From Cloud to Edge
Bandwidth
Tactic T 1
A - side windows
Lower resolution of imagery
Increase level of resolution upscaling on the edge
B - all windows
Lower resolution of imagery
Increase level of resolution upscaling on the edge
Tactic T2
A - side windows
Lower frame rate
Increase rate of local frame interpolation
B - all windows
Lower frame rate
Increase rate of local frame interpolation
Latency
Tactic T 3
Increase Lookahead window
The bigger the window the more correction we are likely to face - more JIT frame adjustment
From Edge to Local
Latency
Tactic T5
Receive high res streams directly at device from cloud (buffer at device instead of edge)
Bandwidth
Tactic T2
Tactic T6
Processing Resource Problems
Edge lack of compute resources
Tactic T5
Cloud lack of compute resources
Tactic T1 A
Tactic T1 B
Tactic T2 A
Tactic T2 B
Tactic T4 B
Tactic T6
Disable side window views
Application Problems
Cloud
Cloud application too slow
Tactic T4
A
Restart
B
Disable selection of prominent visual features

Figure 38: Configurability options to deliver adaptability tactics

5 Enabling XR technologies under development

5.1 Migrating from on-premise to on-cloud

For Collins Aerospace, the vision of CHARITY to enable highly efficient network slices spanning the domains of Cloud providers, Edge infrastructure and local resources inspired a radical re-imagining of what could be achieved in terms of real-time, interactive XR streaming on the cloud. The traditional approach to flight simulators has been to deploy sufficient compute and storage resources alongside 2D fixed screens to deliver on the stringent quality demands of a certification-grade simulator. Scaling up or down is essentially constrained to vertical scaling in which we use more powerful or less powerful hardware as the deployment dictates. In Figure 39 below, we depict three sample deployment configurations

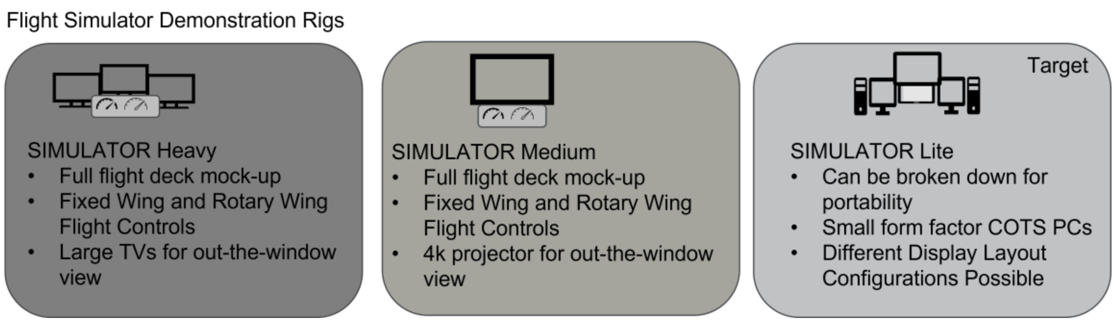

Flight Simulator Demonstration Rigs
SIMULATOR Heavy
• Full flight deck mock-up
• Fixed Wing and Rotary Wing Flight Controls
• Large TVs for out-the-window view
SIMULATOR Medium
• Full flight deck mock-up
• Fixed Wing and Rotary Wing Flight Controls
• 4k projector for out-the-window view
Target
SIMULATOR Lite
• Can be broken down for portability
• Small form factor COTS PCs
• Different Display Layout Configurations Possible

Figure 39: Some deployment models for the existing flight simulator

As presented in Figure 40, the traditional approach is somewhat monolithic in terms of deployment flexibility. Multiple flight simulators co-located on the same site have no interaction or resource sharing and each operates independently on its own dedicated hardware.

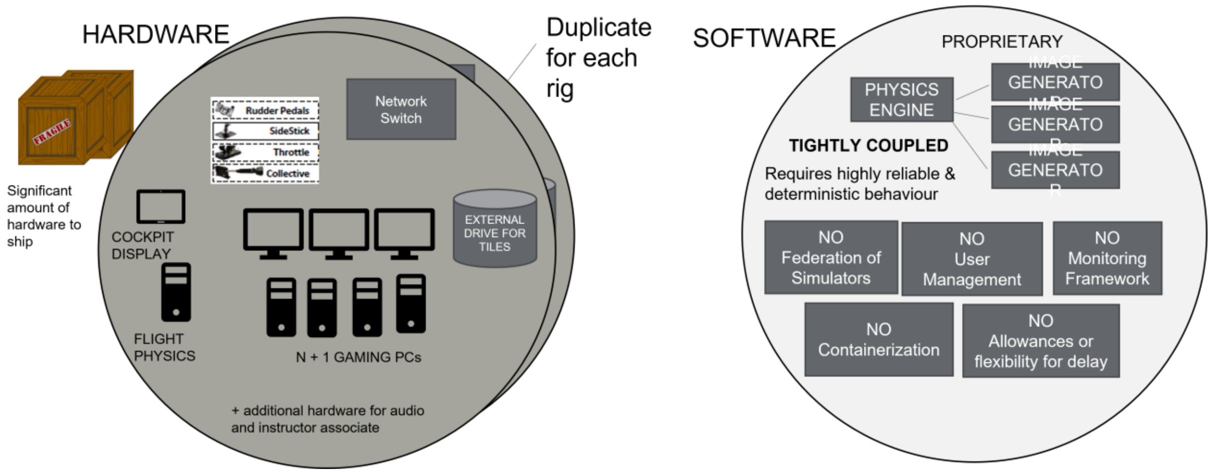


Figure 40: Existing deployment options revolve around a monolithic approach

The current deployment model presents a variety of challenges as outlined below in Table 17 .

Table 17: Challenges presented by the traditional deployment model

Challenges	
Each user requires their own full rig – dimension site hardware up front for max number of simultaneous users	MS Windows focused
No sharing of resources between rigs	Strict latency demands

Software updates are problematic – especially tiles database which is very large	Specialized scenery generator coupled with flight dynamics
Hardware updates are problematic	Difficult to scale
Sense of immersion with low-end rig is poor	Licensing complicates experimentation on third party edge/cloud
No centralized monitoring (across users)	

At the outset, these considerations drove our decision to rethink the flight simulator architecture, to work in a distributed manner with the ability to leverage the CHARITY platform. We envisaged clear benefits that a redesign should bring as outlined below in Table 18.

Table 18: Target benefits from redesign

Benefits	
Greatly reduced local hardware footprint	User & session management, simulator federation
Edge and cloud resources shared between simulators	Monitoring framework integration
Tiles database and rendering engines can be updated on the cloud	Improved versatility through Microservices with Docker containers
Hardware upgrades simplified	Caching with lookahead rendering to manage delays
Improved sense of immersion	Pluggable scenery generator -> flightgear
Improved Scalability	Headless remote rendering for remote computation and local display
Pluggable upscaling	Customizable latency compensation tactics available

5.1.1 The Latency Challenge

Operating a commercial Flight Simulator requires speed and consistency. The turnaround budgets are tight. In deploying to the cloud, we take an already demanding problem that is currently addressed using dedicated local hardware and network resources and exacerbate it by distributing resources across large distances as summarized below in Figure 41.

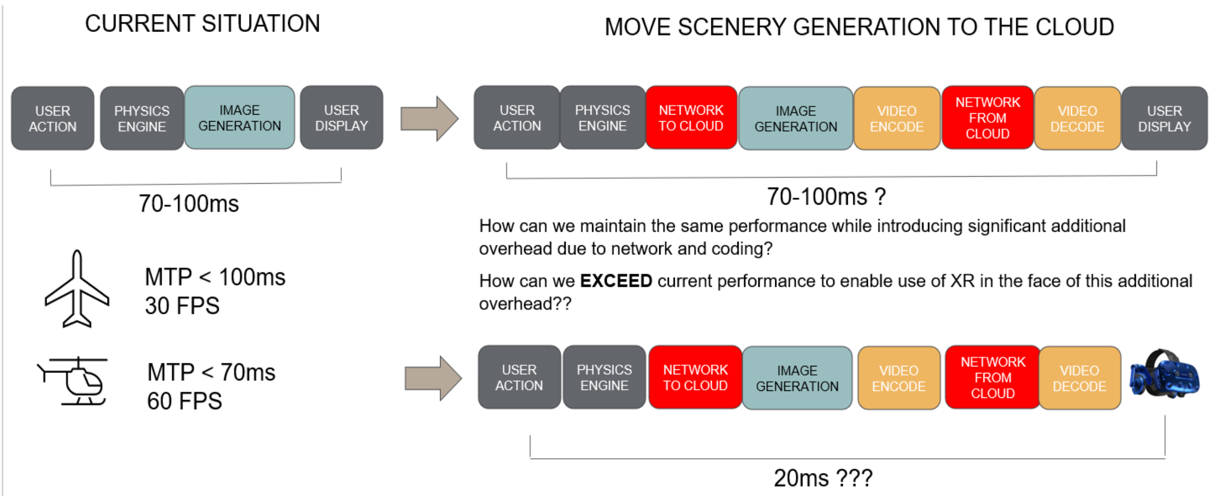

CURRENT SITUATION
MOVE SCENERY GENERATION TO THE CLOUD
USER ACTION
PHYSICS ENGINE
IMAGE GENERATION
USER DISPLAY
NETWORK TO CLOUD
VIDEO ENCODE
NETWORK FROM CLOUD
VIDEO DECODE
70-100ms
70-100ms ?
How can we maintain the same performance while introducing significant additional overhead due to network and coding?
How can we EXCEED current performance to enable use of XR in the face of this additional overhead??
MTP < 100ms
30 FPS
MTP < 70ms
60 FPS
20ms ???

Figure 41: Motion To Photon budgets become even more demanding with XR and the cloud

In 2018, Collins conducted internal experiments assessing the viability of cloud hosted flight simulation [23]. The findings revealed significant challenges that needed to be overcome with respect to network latency and jitter:

- Network Latency is a significant obstacle "*Transport delays vary widely based on network topology, provider, virtual private network, user-to-cloud distance, and other factors*"
- Sporadic variations in rendering times can result in stalls "*cloud-based computing model will require stringent provisioning of shared resources to provide the kind of performance and determinism guarantees users expect*"

The experiments were predicated on the display of scenery on two dimensional monitors – not XR headsets which have far more demanding latency budgets. It was clear from the early stages of the CHARITY project that we were facing significant challenges that could be alleviated but not solved entirely by the CHARITY platform alone. The physics of distance needed to be tackled.

5.1.2 Tackling XR Latency

A key observation about latency budgets in XR is that there are different types – rotational and translational as shown below in Figure 42. The charts on the right portray how latency demands are dependent on the nature of the user activity [25] and we superimposed the position that scenery generation for a flight simulator would occupy.

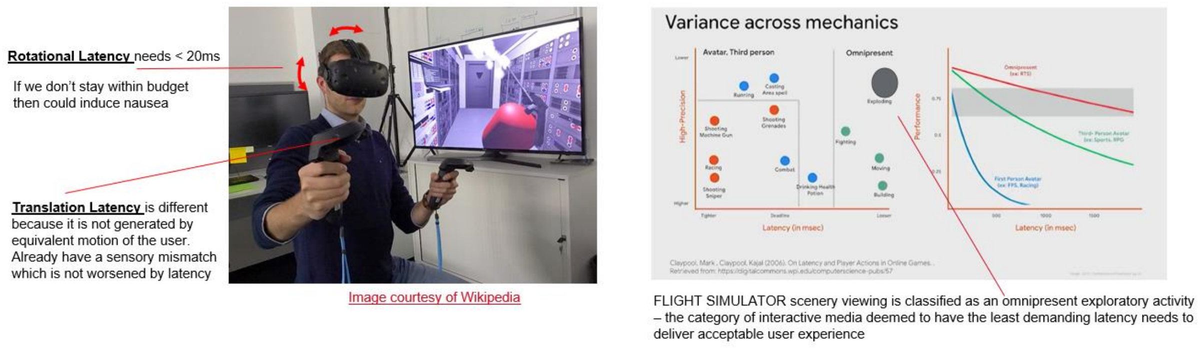


Figure 42: The latency budget available depends on the activity

Updates caused by the user rotating their head need to be very fast (< 20ms) to prevent nausea for a significant proportion of the population. However, in [24] the authors note that translation motion delays of 100-200ms are "non-trivial to notice". For the flight simulator scenario, we have a user that sits within a virtual cabin and is able to look out the window at synthetically generated scenery. If the user turns their head then the local view inside the cabin needs to update quickly. The outside view only changes with the movement of the simulated aircraft itself (which alters course slowly in response to user actions). We propose to leverage this dichotomy to move the generation of synthetic scenery seen through the cabin windows to the cloud while keeping the rendering of the cabin itself local.

5.1.2.1 Prediction to extend the latency budget

If we detach the world outside a simulated aircraft cabin from the world inside then an additional opportunity presents itself to further extend our latency budget. As pointed out previously, the out-the-window view updates in accordance with movement of the aircraft. Aircraft possess nothing like the rapid freedom of movement of a human pilot. Its position within the seconds ahead should be predictable with a high degree of accuracy. This presents the opportunity to render what we need ahead of time on the cloud and cache it locally to enable what Google have referred to as Negative Latency [27] – a variation of which they employed in the Google Stadia platform.

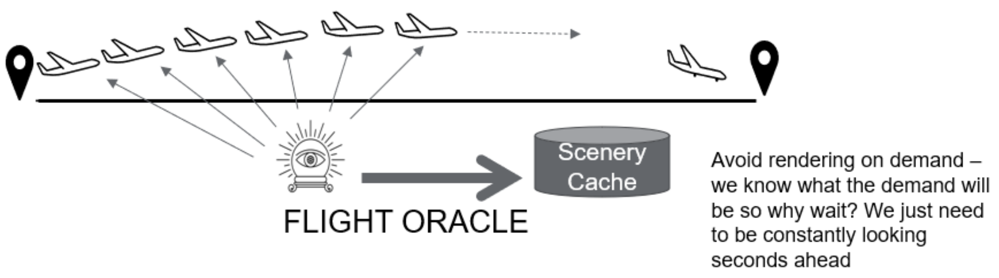

Scenery Cache
FLIGHT ORACLE
Avoid rendering on demand – we know what the demand will be so why wait? We just need to be constantly looking seconds ahead

Figure 43: Movement of an aircraft can be predicted to enable pre-rendering if scenery ahead of time

By caching at the edge, our goal is to detach the cloud from the stringent motion-to-photon loop to reduce the latency and jitter that would otherwise be experienced with cloud rendering in the real-time chain.

5.1.2.2 The Frame Rate Challenge

As witnessed by the steadily increasing refresh rates of XR headwear, high frame rates are seen as an essential component of an acceptable XR user experience. Regardless of what is deemed to be an acceptable rate of frames per second – 30, 60, 90, 120 – we assume that the originator of the media stream must generate that rate. If we want to attain 90FPS with flight scenery, then must we render 90FPS in the cloud and ship back to the nearest cache? As with modern televisions, frame interpolation has become standard functionality in XR headsets. The manufacturers of such headset want to avoid inconsistent or below-par frame rates emanating from media sources to result in compromised experience for the user who may attribute blame to the headset itself. XR headsets need a consistent frame rate. If they don't get it, then they use predictions cached locally on the headset to backfill any missing frames. We observe functionality termed Asynchronous Timewarp and Spacewarp [26] in the Oculus headsets and Motion Smoothing in SteamVR headsets.

Instead of the XR experience imposing more stringent quality demands than conventional 2D monitors, we propose to explore using the stabilization technology built into XR headsets to our advantage. It gives us the option of generating a lower frame rate on the cloud when resourcing pressures preclude us from either rendering the required frame rate due to computational resource stresses or from delivering the required frame rate to the edge due to bandwidth stresses.

5.1.2.3 The Pixel Resolution Challenge

As consumer XR headsets evolve to target 4K or 8K resolutions, there is a growing imperative on the part of media stream producers to render higher and higher resolution imagery. This has significant repercussions for bandwidth as 4k resolution requires an order of magnitude more bandwidth than High Definition (approx. 15Mbps versus 1.5Mbps). As with frame rate, we assume that the originator of the media stream must generate the required resolution. Modern TVs and games consoles need to deliver a high-resolution viewing experience even when the source of frames is of low resolution. To accomplish this, they employ upscaling algorithms to 'fill out' the missing pixels. We propose to integrate a resolution upscaling component into our streaming pipeline to cater for scenarios in which we cannot render high resolution imagery on the cloud for reasons of resource availability (compute or network bandwidth). Lower resolution frames will be received at the Edge and upscaled as appropriate.

5.1.3 Towards Cloud Native

We began our journey with a monolithic platform that was not amenable to distributed deployment and execution. We proceeded to redesign the platform and move towards a cloud native architecture. As can be seen below in Figure 44, we decomposed the platform into self-contained microservices.

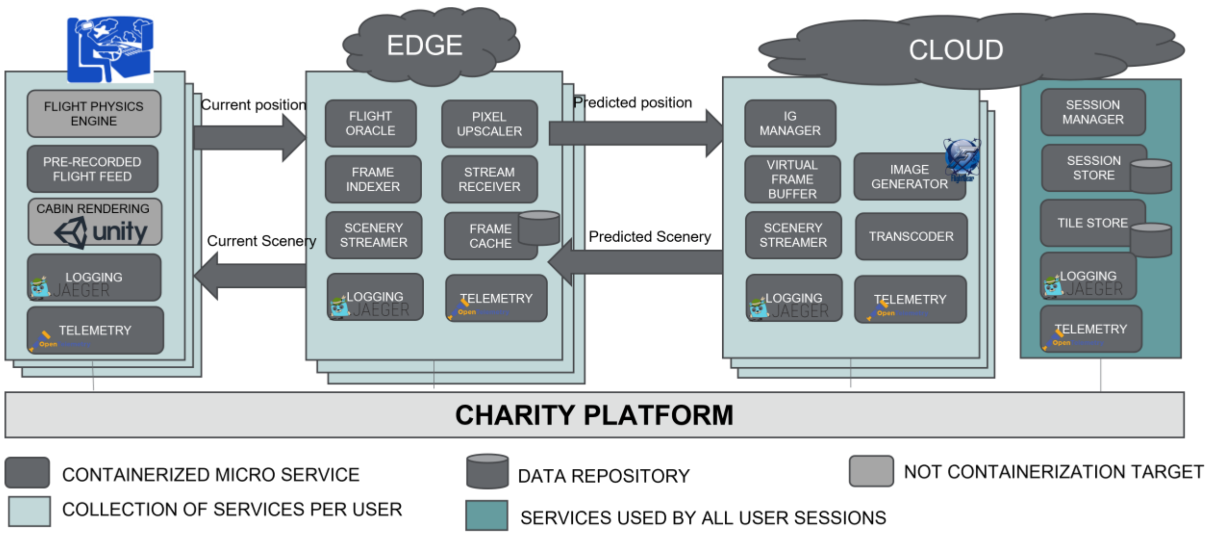


Figure 44: Flight Simulator redesigned as cloud native

The new architecture better reflects modern application design and gives us the opportunity to leverage core features of the CHARITY platform that would have been difficult and far more restricted with the original design such as the CHARITY service mesh for application adaptation, monitoring and alerting, dynamic deployment and orchestration. Crucially, it brings options and mechanisms to explore distributed deployment across the edge and cloud.

5.2 Dissection of the Unity3D Physics engine

ORAMA's commercial gamified multi-user VR medical training platform is built using the MAGES SDK on top of the Unity3D game engine. Exploiting Unity3D's network layer, the MAGES SDK handles and synchronises in-game interactions, deformable object transformations and physics simulation by broadcasting transformation values over the network. Under the hood, as part of the MAGES SDK, the custom Geometric Algebra interpolation engine is utilised for efficient network transmission and local interpolation of in-between positions/rotations for each end-device (HMD). The architectural design of ORAMA's training applications involves a single application component, installed and run on untethered HMDs, that employ local processes for storage, rendering, and physics deformations.

An experimental architecture, based on MAGES SDK, allows the transition to an Edge-Cloud application, upscaling to collaborative cloud VR training applications specially formulated for untethered HMDs. The goal of this R&D version of ORAMA's training application is to optimize the status of the cooperative mode in terms of lower latency, higher performance on average network conditions, and, ultimately, higher number of CCUs. This new approach, realized through computation offloading of the entire ORAMA's application in edge-cloud resources, requires interactions and data exchanges between the different modules placed on device, and services on Edge-Cloud.

ORAMA is currently designing and developing the required technologies and solutions to support its advanced media applications by exploiting methods and techniques for the dissection of the Unity physics simulation engine as a separate VM microservice that will run on the Edge-Cloud. Methods and techniques regarding multi-threaded rendering and physics in Unity are also being investigated.

5.2.1 Dissection of Physics Simulation Engine

Currently, a typical Unity3D game engine pipeline involves simultaneous execution of CPU physics-related calculations along with GPU calculations related to the rendering of the scene.

In this section, we provide an overview of how a dissection of the physics and the scene-rendering pipeline can be achieved. Although a distributed application architecture usually decreases running times, an unoptimized dissection may lead to increased latency, since there are numerous inter-calls between the physics engine and the renderer. In the case of a desktop-VR local network system setup, the dissection is feasible and almost straightforward. However, in the case of a mobile-VR edge-cloud setup the physics engine dissection is rather challenging. ORAMA is currently investigating methods and techniques that will assist such a potential dissection and allow the physics simulation engine to be run as a separate edge-cloud, possibly containerized, microservice.

5.2.2 Methodology – Notation

The dissected Unity3D pipeline involves two, bidirectionally communicating, components:

- The Host (Game Logic and Graphics rendering), and
- The Physics Server (Physics simulations).

The Host includes the entire Unity3D pipeline, along with its own, local, physics engine. Main goal of the dissected Unity3D pipeline is to allow any GameObject on the scene to be fully simulated by the dissected Physics Server and not by the Host's local physics engine.
For the reader's convenience, we define below some terms used throughout the dissection overview.

- **Graphics Object:** A Game Object component, with no physics-related scripts and data residing within the Host. Any Game Object may be converted into a Graphics Object by detecting and removing all physics-related parameters (colliders, rigid bodies, etc.) attached to it. The removed parameters are stored temporarily in order to be sent to the Physics Server in the form of a Physics Object (see below).
- **Physics Object:** A Game Object component, responsible for storing all physics parameters. It has attached a Rigid Body script, a Collider script, or a combination of the two. When initialised, however, it generates physics components based on this data and is responsible for updating these components whenever a change occurs. The data is not editable in the Physics Server, only by the host, since the Host side is responsible for manipulating physics parameters.
- **Remote Game Object:** Since the above two components use different ways of storing their data (Physics or Graphics Objects) we need a way for them to communicate. The **Remote** Game Object, a data structure containing all shared data between Physics and Graphics objects (Transform, Collision Events), is used to forward Graphics Objects updates to the respective Physics Objects, or vice versa.
- **Graphics/Physics Client:** These services are listeners responsible for all communications and orchestration on either the Host or Physics Server side. They apply all incoming changes on all Graphics/Physics Objects and sends all outgoing changes in the form of Remote Game Objects.

5.2.3 Methodology - Overview

5.2.3.1 Communication of the two Components

The main two components, i.e., the Host and the physics server, communicate via network using TCP and UDP connections, exchanging messages in JSON format. The TCP connection is used for the exchange of more critical information, such as the creation of an object or the modification of a critical parameter. The UDP connection is used to transmit real-time data, like the object transformations or time-critical data, such as collision events. For the convenience of the reader, we shall refer to these communication methods as *Reliable* (TCP) and *Unreliable* (UDP). Of course, these transmission protocols can be substituted for a more domain specific networking solution.

5.2.3.2 Splitting a Game Object into a Graphics and a Physics Object

After a successful connection, the **Host's** game objects are split into **Graphics objects**, which remain in the **Host**, and **Physics Objects**, which are created in the **Physics Server**. This is accomplished by transferring the physics attributes, such as Colliders and Rigidbodies, from each of the Host's game objects, to the Physics server, that creates Physics objects with the same parameters. The Physics Server retains no knowledge regarding the Host's scene, the game loop or the behaviours, and is only for simulating the physics of all Game Objects in the scene.

During gameplay, the transformations of the Host's Graphics Object are synchronized with the respective Physics Server Physics Object. The Game Object's transformations can either be controlled entirely by the Physics Server, or by the Host. In the latter case, the Physics simulation continues and the controlled Physics Object interacts with the rest of the Physics Objects as expected.

5.2.4 Implementation

5.2.4.1 Initial Setup

With the successful initiation of the Physics Server, the Physics Client starts listening for messages on specific ports. At this stage, any Host can connect to it and provide Game Objects for physics calculations.

After the Host's successful initialization, its Graphics Client scans all Game Objects in the scene for Physics objects and converts them into Graphics Objects. The Physics components, attached to those Game Objects, are subsequently collected and sent to the Physics Server as Remote Game Objects. The Physics Server then converts and instantiates them as Physics Objects.

5.2.4.2 Game Object creation after Initialization

A new Game Object, spawned in the Host, is initially attached to all of its Physics components, and subsequently it goes through the same conversion process, to Graphics and Physics Object, described in the initialization subsection.

5.2.4.3 Simulation and Gameplay

After the Graphics Objects are successfully copied from the Host to the Physics Server, their transformations will be synchronised using unreliable transport. Depending on the developer's choice, the transformations of a Game Object can be either controlled by the Host or the Physics Server. In both cases, the Physics simulation is always running, and no components are deactivated. When a Graphics Object is translated by the Host, the corresponding Physics Object is also translated using Physics calculations and not direct transformation changes so that the simulation is accurate and ensuring that no undesirable object clipping occurs. Besides simply changing the transformations, the Host can change all parameters of the Graphics Object which, depending on the type of data, are to be sent to the Physics Server using a reliable or unreliable transmission method. In that respect, time-sensitive parameters are usually sent using the unreliable connection; those that must definitely reach the Physics Server (such as Gravity) are instead sent using a reliable connection.

5.2.4.4 Collision Detection

When collisions happen, the Physics Objects provide the Graphics Objects with the identifier of the collided Game Object. The well-known Unity3D events *OnCollisionEnter*, *OnCollisionStay* and *OnCollisionExit* are accessible via the Graphics Objects and can be subscribed to and used as expected on the Host side.

5.2.5 Lab Testing

The testing process was performed in three stages: a) Initial testing, b) Synthetic testing and c) In-vivo testing.

5.2.5.1 Initial testing

The dissected pipeline was deployed and tested on two separate machines connected to the same network, one via Ethernet and one via Wi-Fi. Main objective of the tests was to measure the subjective and objective performance of the actual system based on different network quality. The initial testing scenarios involved one or multiple Physics objects, defined as GameObjects, with 1 Rigidbody and 1 BoxCollider component. The conducted tests also involved the compression of the exchanged data between the two network components. For compression, the GNU GZip, which is based on the Deflate algorithm, is used. Following is the list of the used testing scenarios:

1. Stress testing on the number of objects able to be synchronised using no compression.
2. Stress testing on the number of objects able to be synchronised using compression.
3. Random transformation of 1 Physics object using no compression.
4. Random Transformation of 1 Physics object using compression.
5. Random Transformation of 10 Physics objects using no compression.
6. Random Transformation of 10 Physics objects using compression.

Stress testing with no compression, showed that the system is able to synchronise the simultaneous transformations of at most 28 Physics objects. Beyond that, the internal network buffer is overflown. Overcoming this limitation without reducing the packet size is not currently feasible.

On the other hand, when data compression is used, the system is able to synchronize over 480 simultaneously transformed Physics objects. The system's performance started degrading for cases with over 200 Physics objects and, in cases with over 480 Physics objects, it was almost unusable. The internal network buffer did not overflow in any case.

Table 19 provides the results of tests 3-6.

Table 19: Testing scenario – results

		1 object compressed	**1 object uncompressed**	**10 objects compressed**	**10 objects uncompressed**
TCP	**Average Packet Size** (bytes)	1114	2850	1705	16105
	Maximum Packet Size (bytes)	1288	4240	2256	23946
	Minimum Packet Size (bytes)	912	2062	912	2062
	Average Packet Delay (ms)	0.0988	0.0981	0.1003	0.0734
	Packet Loss (%)	0	0	0	0
UDP	**Average Packet Size** (bytes)	1175	2252	2643	22364
	Maximum Packet Size (bytes)	1204	2264	2740	22435
	Minimum Packet Size (bytes)	1132	2240	2248	22299
	Average Packet Delay (ms)	0.0842	0.0842	0.0504	0.0363
	Packet Loss (%)	0.15	**0.15**	0.26	0.23

Based on above results, we can safely conclude that compression reduces the size of the packets such that more simultaneously Remote Game Objects are allowed. This reinforces our findings, regarding the limit of simultaneous physics objects and the effectiveness of compression.

The network usage, when synchronizing 10 simultaneously transforming virtual Objects, without compression, was at an average of 22.4KB/s. This metric is further improved when using compression, averaging at 2.6KB/s.

5.2.5.2 Synthetic testing

To test the dissected Unity3D pipeline against network conditions and evaluate the visual impact on the gameplay, we performed a synthetic testing process, where the dissected pipeline testing is conducted in emulated network conditions. These testing results and the subjective system's performance will be used as reference for the actual integration testing in the selected testbed. To eliminate any additional overhead in the emulated network conditions, that would tamper our results, we deployed the two servers on the same machine. For the generation of synthetic network conditions, we emulated the major network factors that mostly affect the visual performance and QoE in real-time VR applications: a) packet loss and b) latency.

Packet Loss

Packet loss occurs in unstable network conditions, causing unsuccessful packet delivery. During testing, packet loss of over 20% resulted in choppy movement in VR. In such cases, the visual and subjective performance of the system depended on the object transformation rate. When the interacted object is moved slowly by the user within the Virtual Environment, the visualized transformation result is satisfying, even at higher package losses. When the virtual objects are transformed fast and abruptly, the visualized transformation is noticeably jittery.

Most GameObjects in UC2-1 are interactive, representing surgical tools, whose normal use does not involve very high transformation speeds. As such, a maximum packet loss of at most 50% is visually acceptable. This value is the upper limit, as in real situations many other factors may impact network conditions, that would affect negatively the system's QoE. In any case, a packet loss of less than 20% would provide a stable QoE. Table 20 provides a summary of the visual feedback results against the used synthetic packet loss.

Table 20: QoE vs Packet loss

Packet Loss	Visual Feedback
5%	Good QoE.
20%	Intermediate QoE when the user moves virtual objects fast and abruptly; marginal good QoE during a normal gameplay.
50%	Intermediate QoE during normal gameplay, but acceptable for short periods of time; for longer periods of time QoE can get annoying or distracting.
80%	Bad QoE; the training scenario is unplayable. Objects feel like they teleport randomly instead of moving in the scene.

Latency

Network latency is defined as the time that takes a packet to be transmitted to the target client, often causing a significant delay projecting the visual output of the system. Increased latency in distributed VR systems, like the one in UC2-1, may cause bad synchronization in the VR HMD, between user hand/head actions and the respective image projection. This delay is not constant, causing packets to be possibly received out of order, resulting in a jittering visual effect. Introduced latency during testing sessions, caused a significant visual impact even at smaller latency values. Table 21 provides a summary of the visual feedback results against the used synthetic latency.

Table 21: QoE vs Latency

Latency	Visual feedback
50ms	Intermediate QoE; Slightly noticeable jittering - when user moves virtual objects slowly, intermediate jittering can be more obvious.
100ms	Bad QoE; Jittering very noticeable - the object jitters a lot and feels unstable.
200ms	Unplayable and very annoying to use.

5.2.5.3 In-Vivo testing

In order to prepare the system for testbed deployment, we conducted tests over the internet with a remote computer.

Initially, the system could not be deployed due to the large number of UDP packets exchanged by the two servers, causing the ISP's firewall to block all outgoing and incoming connections. This was solved by sending packets less frequently compared to a local network setup. As such, many packets were not transmitted, introducing additional perceived latency and packet loss to the system.

The packet send-rate was initially set to 100ms, which is the maximum playable latency determined during synthetic testing, producing similar visual results, without generating anymore the jitter caused by out-of-order packets. Although, this slightly improved QoE caused less distraction to the user, the virtual objects movements lagged behind significantly, producing extremely jittery movements. The firewall issue and its resolution, kept the aggregated perceived latency permanently over 20ms. After thorough experimentation, we determined that the ideal send-rate is 50ms, which provided the most stable experience with average QoE and without overloading the connection. Table 22 summarizes the data gathered using the UDP connection and its stability.

Table 22: UDP connection results

Packet Loss	Average Latency	Average Packet Size
6.5%	237ms	12452 bytes

The large network latency recorded in the in-Vivo testing is the main reason for the significant delay in user's actions. Packet size was relatively small, with room for improvement using compression, however it was well within reason, as network usage remained under 10Mb/s.

Due to the unstable network conditions, in-Vivo testing of the entire medical training session was impossible. The TCP connection often dropped and the current implementation does not anticipate failed TCP connections. Further in-Vivo testing will be conducted after the respective modifications are applied.

5.2.6 QoE Subjective remarks

During gameplay, whenever the network quality falls below a threshold, there were two issues that were especially noticeable. First, when directly interacting with objects in VR, the network latency causes the objects to feel "squishy", since the user's hand that pushes the object would initially penetrate inside the object and after some milliseconds the object would react to the push and move away. Secondly, when there is a high amount of packet loss, some objects tend to "flicker" between two positions. This issue is not very common as it is not experienced every time network conditions deteriorate. The first issue, however, is rather common, but not distracting from the gameplay in a severe way.

5.2.7 How Compression affects performance

We can measure the impact of the compression algorithm on the system's performance by utilising the Unity3D profiler. The scenario to measure the system's performance, included the random translation of 10 Physics Objects.

When compression is not used, the generation of a frame takes approximately 20ms to be executed, sometimes spiking up to 33ms. On the other hand, when compression is used, a frame takes an average of 21ms to be generated, with some higher spikes at 34ms. These spikes occur since sending/receiving packets happens asynchronously. We conclude that both compressing and decompressing packets adds around 1ms to frame rendering times.

When more Physics Objects were used, the compression/decompression had a linear performance impact (20 objects added 2ms to no compression, 30 objects added 3ms).

The compression method used is not a domain specific solution for game engines so, in a more realistic scenario where one would make use of third-party solutions (such as Oodle Networking), it is expected that the average frame-time will not be severely impacted.

5.2.8 Conclusions - Future Work

The work in this section has shown that the dissection of the Unity3D pipeline is feasible, yet dependent on the network characteristics between the Host and the Physics server. The conducted tests helped the derivation of the network latency and packet loss thresholds, below which we can achieve a pleasant QoE to the VR medical training application. These thresholds should not be exceeded by the provided testbed network.

Although docker containers outperform VMs in the case of space and processing overhead, they are rather immature in graphics acceleration processes. In this case, the use of VMs is far more advantageous since they have highly optimized graphics drivers and kvm passthrough support. Additionally, docker containers have limited graphics drivers support, since only experimental versions (for all vendors) for Linux are currently available. As such, the porting of the Host, that exploits GPU resources for rendering, into docker container is a rather error prone procedure.

On the other hand, the Physics engine, exploits CPU resources for the physics computations. In the next period, we will work towards porting the Physics engine into docker container. Additionally, to improve the computational latency of the Physics server, we will investigate optimizing the physics computations in Unity3D.

5.3 Investigating Multi-threaded rendering in the Unity3D game engine

Multi-threading exploits a CPU's capability of processing many threads concurrently across many cores. A multi-threading program always starts in one main thread, which subsequently creates new threads that run in parallel. Upon completion, these threads usually synchronise their results with the main thread.

The generation of more concurrent threads than the available CPU cores, leads to a concurrent sharing of CPU resources among the threads, which causes frequent, resource-intensive, context switching. As such, the multi-threading approach always suits cases with a few long-life tasks. Game Engine pipelines mostly deal with many short-life unrelated tasks that execute at once. Multi-threading in such systems often results with a large number of short-life threads that challenge the CPU's and operating system's processing capacity, due to frequent creation and destruction of threads for short-lived tasks. The employment of a pool of threads, often mitigates this issue, increasing performance and avoiding latency in execution. However, even this solution does not always prevent a large number of concurrent active threads.

Multi-threaded programming faces high risks for race conditions which often produce significant challenges. A *race condition* occurs when the output of one task depends on the timing of another process outside of its control. This issue may be a source of crashes, deadlocks, incorrect output, and generally non-deterministic behaviour that produce non accurate rendering or simulations. As the cause of these problems depends on timing, the recreation of the issue could happen on rare occasions, making debugging a cumbersome process. Debugging tools, such as breakpoints and logging, often change the timing of individual threads, causing the problem to falsely disappear.

In the frame of taking advantage of the edge-cloud resources parallel processing, methods and techniques for parallel/multi-threading Rendering and Physics in Unity3D will be explored. Unity3D supports multi-threaded math calculations and, in this regard, we will seek to exploit parallelization techniques for various sub-tasks, such as the skinning algorithms. Furthermore, Unity3D supports a limited form of multi-threaded rendering by utilising specific graphics API implementations or through the utilisation of Graphics Jobs System.

5.3.1 Single-threaded Rendering

Unity3D mainly features a single client occupying the main thread with the execution of the high-level rendering commands. The client also owns the real graphics device GfxDevice and performs the actual rendering through the underlying graphics API (GCMD) on the main thread.

5.3.2 Unity3D Multi-threading Built-in System

Multithreaded rendering in Unity, provided its graphics API permits it, is implemented as a single client, single worker thread. This works by taking advantage of the abstract GfxDevice interface in Unity3D. The different graphics API implementations, such as Vulkan, Metal and GLES, inherit from the GfxDevice.

When this system is enabled, rendering calculations are performed on a separate thread, called the *RenderThread*, while the rest of calculations are performed on the main game thread, namely the *MainThread*.

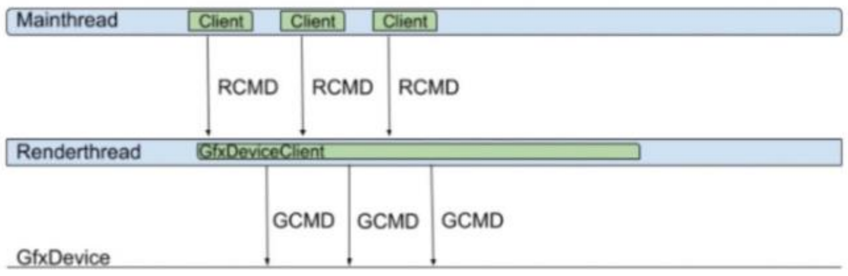


Figure 45: Unity3D multi-threading Built-in System

5.3.3 Graphics Jobs System

The Unity3D *Jobs* system is not the traditional kind of multi-threading system as it manages multi-threaded code by creating jobs instead of threads. In that frame, a game is split into small units of work where each is responsible for one specific task. These units of work are called *jobs*. The Graphics Job system manages a group of worker threads across multiple cores. It usually has one worker thread per logical CPU core, to avoid context switching. Some cores may also be reserved for the operating system or other dedicated applications. As the job system enqueues the generated jobs in the job queue, the Worker threads take items from the job queue and execute them.

A job may receive parameters and operate on data in a similar way to a method call. As such they can be self-contained, or they can depend on other jobs to complete before they can run. Once scheduled, it cannot be interrupted. In complex systems, such as those required for game development, it is unlikely that a job is self-contained. All jobs are usually dependent on other jobs as they prepare data for them. The Graphics Job system supports dependencies across jobs, as it is responsible for managing them, ensuring job execution in the appropriate order. The Unity3D C# Job System is able to detect all race conditions, protecting the programmer from potential bugs.

Writing multithreaded code can provide high-performance benefits, such as significant gains in frame rate. Using the Burst compiler with C# jobs gives you improved quality, which also results in substantial reduction of battery consumption on mobile devices.

Graphics Job System integrates Unity's native job system. As such, User-written code and Unity3D engine code share the same Worker threads, avoiding the creation of more threads than CPU cores, which would cause contention for CPU resources.

Using the Job system, multiple native command generation threads take advantage of the graphics APIs that support recording graphics commands (GCMD) in a native format on multiple threads. It is implemented as multiple clients, no worker thread. This removes the performance impact of writing and reading commands in a custom format before submitting them to the API.

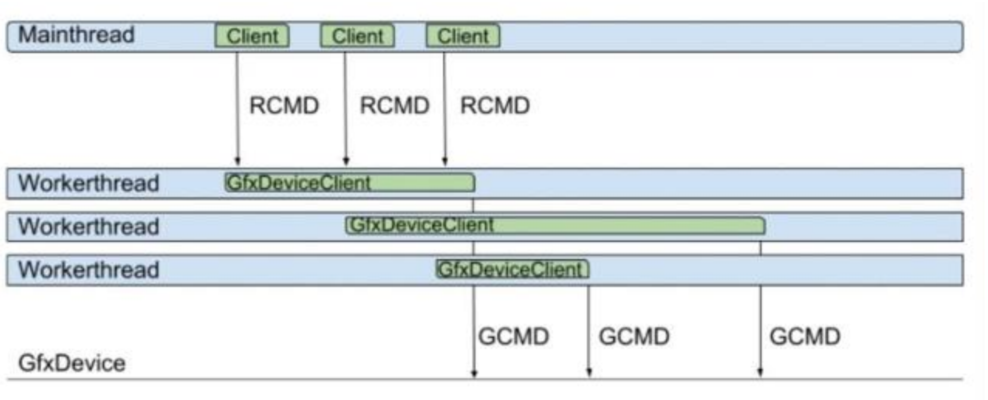


Figure 46: Graphics Jobs System

Note: Currently, Graphics Jobs do not have a RenderThread to schedule jobs, causing a small amount of overhead on the main thread for scheduling.

5.3.4 Vulkan Graphics API

By enabling Graphics Jobs and the use of the Vulkan graphics API for Windows on Unity3D, we tested the potential increase of performance of Unity3D for our VR offloaded solution. In most cases, the positive performance impact was minimal:

Table 23: Potential increase of performance of Unity3D using Vulkan graphics API for Windows

	Direct3D-11	Vulkan
Average Frame rate	45.47 fps	46.16 fps

As an additional remark, Vulkan on Unity3D proved to be more unstable than Direct3D11; in some cases, performance dropped significantly to ~30 fps when Vulcan was enabled.

5.3.5 Conclusions

In our research we noticed that, regarding multi-threaded rendering for 3D applications, one rendering thread is used, while many other work threads can parallelize other jobs such as physics, logic, AI, etc. To the best of our knowledge, there is no other multi-threaded rendering solution or any other alternative solution within Unity.

5.4 Adaptive rendering algorithms for low latency immersive applications

Virtual Reality (VR) applications have gaining importance and interest over the last few years in various fields, like as manufacturing, training, entertainment, and so on. Moreover, modern wireless lightweight powerful Head Mounted Display (HMD), reach a high level of maturity and provides a more immersive experience. Despite these, a high-quality level of experience is still challenging to have when using HMD, also modern ones, because ultra-low latency (<20 ms) and high-bandwidth are required for a comfortable, satisfying, and convincing immersive experience [30].

Several solutions have been developed by the VR research community to achieve this goal. A lot of effort has been spent to reduce the computational burden related to the rendering. In fact, rendering for immersive devices require at least the rendering from two different viewpoints, and in some cases to generate a 360-degree panoramic image/video to stream accordingly to the position and orientation of the gaze of the user. The computation of the rendering can be alleviated in different ways. Some approaches exploit the fact that the best visual acuity is around the fovea, and exploit eye tracking to optimize the rendering, obtaining the so-called *foveat rendering*. Many other solutions exploit how the Human Visual Systems (HVS) works to reduce the quality of the rendering ensuring the same visual perceptual quality. For example, in [31], modify the standard primitive rasterization considering some perceptual effects to make the rasterization pipeline more efficient for HMD. Some other approaches take into account that distant object does not require to be rendered with different disparities to be perceived correctly. For example, in [32], a of mix stereoscopic and standard rendering is used to generate the images to display, according to the fact that disparities are reduced for distant objects. The experiments conducted demonstrate that this simple solution can give a satisfying experience in many cases. Other approaches work by super-sampling the temporal line, so they create/interpolate new frames in-between other ones to reduce the total number of images to generate. The state-of-the-art of this type is ExtraNet [33], a deep learning network capable to double the speed of the frame generation by extrapolating the new frame for the previous ones. The new frame is generated by minimizing the visual artefacts that typically happen in view-dependent parts of the images (e.g. specular reflection).

Recently, with the main goal of obtaining the VR experience for mobile devices, solutions that takes advantage of computing the rendering at the edge are explored [34]. In this case, the total end-to-end latency is given by the time to transmit sensor data from HMD to the edge computing node, plus the time to render (and encode) the views on the edge node, plus the time to transmit rendered images/video from the edge computing node to HMD, and time to (decode and) display the view on the HMD. The encoding and decoding phases are optional and depends on the specific application. In this setting, different strategies can be used to optimize the rendering, caching, and streaming of the different views.

FlashBack [35], is a VR system which pre-renders all possible views on a 3D grid of suitable size, and delivers frames according to the position and orientation of the viewers. Obviously, this is not optimal from a caching point of view. In [36], a parallel rendering and streaming mechanisms is adopted. Streaming latency is reduced by re-using rendering parts that remain the same during the interaction. Long-Short Term Memory (LSTM) ([37], [38]) model and Recurrent Neural Networks (RNN) ([39], [40]) are used to estimate the head/body movements. The prediction of these movements is useful to optimize the view generation, reducing the overall computational and improving performance.

In CHARITY, we aim to develop and integrate in one or more UCs an adaptive rendering solution to obtain high-quality low-latency VR applications. The UCs under investigation are the UC2-1 VR Medical Training Simulator (ORAMA) and the flight simulator, i.e. the UC3-2 Manned-Unmanned Operations Trainer Application (CAI). The current activity has regarded the study of the state of the art to identify the method/technique that can be easily integrated in the architectures of the UCs just mentioned.

In the next period we will study in detail the adoption of a frame extrapolation/interpolation method, and the various correlations with UC2-1 and UC3-2 architectures, as its characteristics imply a promising solution. After that, the implementation will start immediately. In this context a goal is to keep the integration efforts as small as possible yet without compromising the quality of the results to be achieved.

5.5 Point Cloud Encoding / Decoding

5.5.1 UC1-3 Holo Assistant

The CHARITY UC Holo Assistant (Figure 47) adopts the physical principles "diffraction and interference of light" to enable real 3D holography, based on sophisticated custom optical components and algorithms. This lays the foundation for showing a butler-like avatar in 3D space on a holographic 3D display with true depth and true eye focus - for your eyes it is like natural viewing. The butler shall react to natural language and assists by providing information gathered from the cloud or the internet. Beside the 3D holographic presentation, this use case enables a lot of challenging services and new technology to be developed and implemented in the CHARITY cloud.

The use case is focusing on a cloud-based application rendering a virtual holographic 3D assistant including additional information and transferring / streaming the content to a local client system in a format compatible to interference-based holography. On the client system, the content is computed into a real-time 3D hologram and is presented on a holographic 3D display from SeeReal Technologies (SRT). By using eye-tracking, the observer always sees the correct perspective of the holographic assistant 3D scene. The hologram enables natural viewing for correct eye focusing and convergence to experience true depth and natural viewing. So, the well-known accommodation convergence conflict known from classic 3D stereo does not apply here.

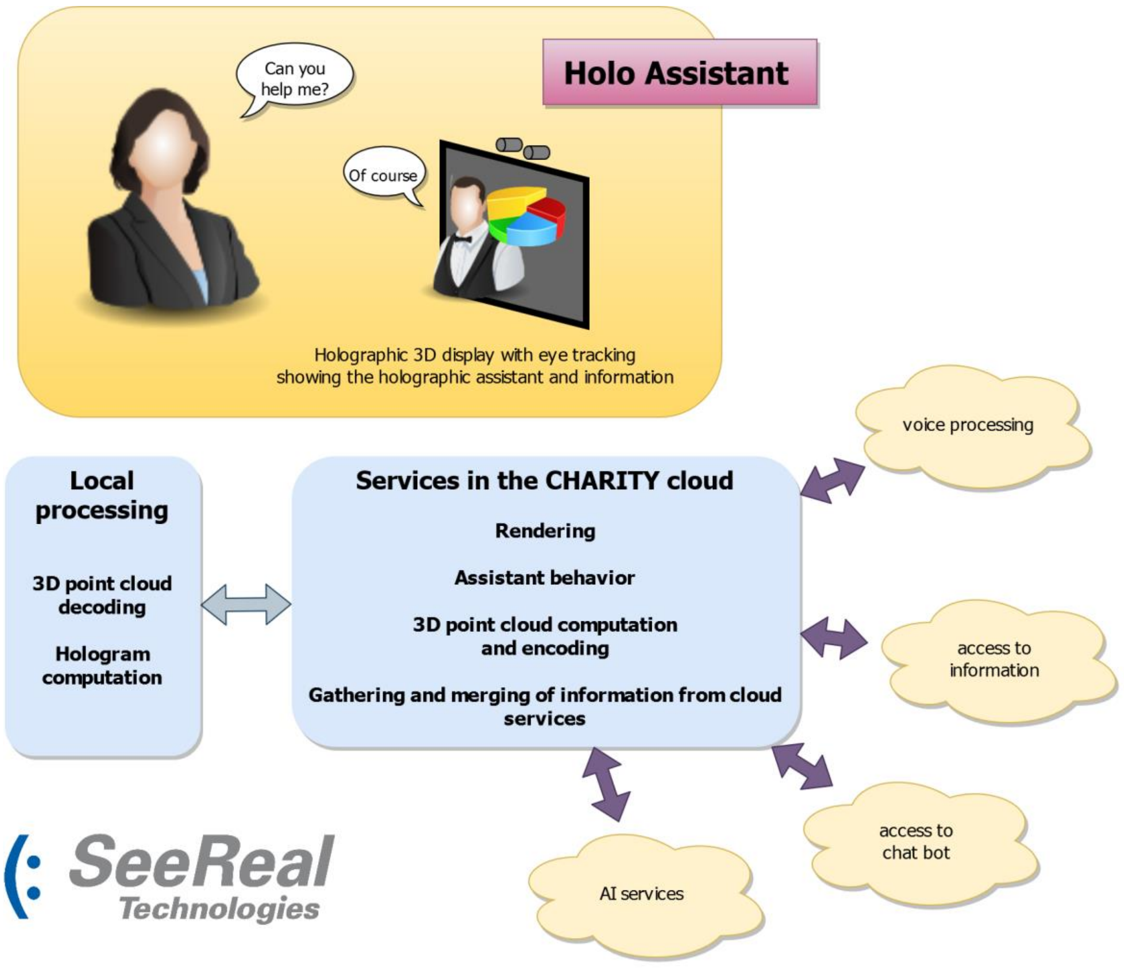

Holo Assistant
Can you help me?
Of course
Holographic 3D display with eye tracking showing the holographic assistant and information
Local processing
3D point cloud decoding
Hologram computation
Services in the CHARITY cloud
Rendering
Assistant behavior
3D point cloud computation and encoding
Gathering and merging of information from cloud services
voice processing
access to information
access to chat bot
AI services
SeeReal Technologies

Figure 47: The Holo Assistant User Case

The preferred format is a Point Cloud (PC) based format which provides following advantages over an image-based data format like 2D+depth:

- Multiple views can be encoded acting as a cache on client side - if many views are available, there is no missing data when data coming from the cloud is delayed
- The format becomes more effective over image based 2D+depth the more different views are encoded
- Multiple points on one "line of sight" allow for looking around an object within one and the same data set, but also enable transparency effects
- compared to typical point cloud formats, here is targeted to only include certain views or a certain view range, thus the point cloud scene must not be viewed correctly from all sides

The use case thus requires the following modules to be developed:

- Point cloud (PC) generation module (which is dependent on the rendering engine)
- PC compression
- Data transfer of compressed PC data
- PC decompression

We underline that the R&D activity in CHARITY regards the aspects just mentioned, and no other aspects involved in the fruition of the holographic content, such as the interaction modalities between the user and the avatar or the design of the user interface.

5.5.2 First point cloud encoder/decoder (PC E/D) design considerations

The overall process that we have to take into account for the development of the PC E/D is the following:

Step 1: We need to generate a point cloud from a generic 3D scene created with a game engine like Unity 3D. This point cloud contains all the 3D scene points to be seen from different views - at least two for the two eyes of the observer looking at the holographic 3D display. The generation could be based for instance on rendering multiple views of the Unity 3D scene, but the point cloud can be generated also in other ways. An advantage of this method is that it relies on a very generic approach, easy to be applied to any 3D content and any 3D engine.

Step 2: the 3D point cloud needs to be compressed. As stated in the previous section, additional algorithms and heuristics like detecting changes from frame to frame can be applied in order to reduce the amount of data to transfer. Network quality adaption is also done here, reacting to indicators and control mechanisms from the CHARITY Cloud. For example, the resolution of the 3D point cloud could be adapted dynamically and/or the number of encoded views could be reduced.

Step 3: the data is transferred over the network. Some feedback about network quality is provided by the receiving client to the CHARITY cloud. The receiving client decompresses the received point cloud data and applies it to the existing data model - i.e. applies scene point changes for the case that only changes in the 3D point cloud have been transmitted. In the last step, depending from the actual observer’s eye location at the holographic 3D display, the views needed to generate the hologram are extracted from the local 3D point cloud and the hologram is computed and presented to the observer.

To start, we need to define a data format suitable for data compression / decompression algorithm. The idea is to use a volumetric format, i.e. a voxel, and store in each cell of the voxel a 3D points plus additional information such as:

- Location in space → defined by position in the grid
- Color + optional alpha + material tag to define transparency behaviour
- material tag could be something like: fog/smoke, clear glass, distorting glass, coloured glass
- *Viewability* - definition from where the point or a certain list of points can be seen → certain eye boxes in space are needed to be defined

- If no eye boxes are defined, we assume this is not a reduced PC and could not be seen from all sides, in this case no viewability attributes are provided.

For the overall PC we need:

- Eye boxes / ranges for which this PC is valid → in 3D space we define the PC cuboid's location and size + multiple eye boxes
- Resolution in X/Y/Z → number of voxels / definition of the 3D grid
- Information about globally contained attributes → alpha and / or material tags, viewability information.

Figure 48 explains what is meant with eye boxes and 3D point viewability. Certain 3D points would be seen only from certain eye boxes while most points are visible from all eye boxes.

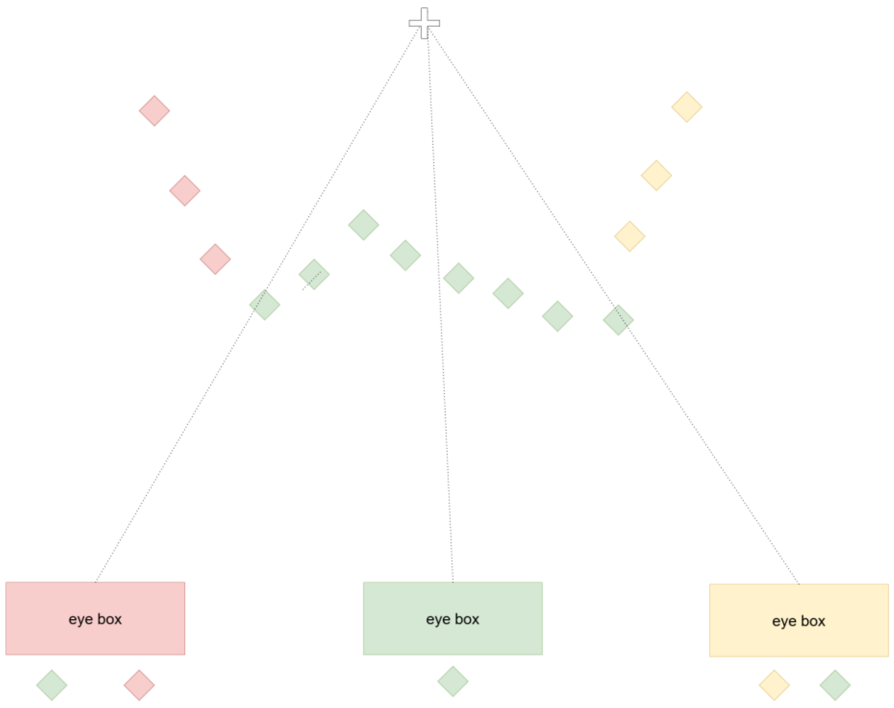


Figure 48: Relationship between the eye boxes and visibility of the 3D points

Regarding the existing standards for point clouds, we analysed the recently published MPEG Point Cloud Compression (MPEG PCC) standard [21].

From the viewpoint of official standardization, good progress was made by the MPEG Point Cloud Compression project (MPEG PCC). It was initiated in about 2014. A call for proposals in 2017 resulted in a first draft of the standard at the end of 2018. Until today the standard is under development and there is an actively maintained reference implementation. Basically, the standard proposes two types of 3D point cloud compression - video based (V-PCC ISO/IEC 23090-5) and geometry based (G-PCC ISO/IEC 23090-9).

Figure 49: Example data sets used for comparing V-PCC (image taken from paper in Ref MPCC-1).

The V-PCC variant uses classic image-based processing (color + depth + occupancy maps). By applying common image-based compression methods (HEVC in the reference implementation), quite good compression rates can be achieved. The method is based on projection of the 3D source scene or point cloud on multiple 2D maps from different perspectives. These projections or patches are then mapped into the frame - the "atlas" - to be encoded / decoded by means of video compression. Here multiple maps are generated, attribute maps (can be RGB color but also something else), depth maps (representing the distance from the according perspective) and an occupancy map (representing valid pixels). Within a (lossless encoded) meta data channel, information about how to reconstruct these patches back into the 3D point cloud are provided within the multiplexed data stream. Within the process of generating the patches and atlas, some improvements on the data are done, i.e. detection and removal of duplicate 3D points or improvement of quality esp. on the regions between patches (seams). As a result, very good compression rates are achieved. The MPEG PCC research group defined some reference data sets (see Figure 49), where the rates and quality of different algorithm versions and parameter variants could be measured and compared. For example, a scene with 100k points @30fps corresponds to 360Mbit/s uncompressed data rate. With V-PCC a compression to about 1 MBit/s can be achieved using version TMC2v8.0 while achieving good quality.

The G-PCC variant is based on compressing the 3D points directly one by one. Here the 3D points structure (point locations) is encoded lossless by using an octree approach (divide a cube into 8 cubes iteratively until we are at point level – noting down if there is something inside the cube or not – represented with 8 bit per cube). For encoding point attributes (i.e. RGB color), three compression methods have been developed. These methods basically make use of similarity / redundancy between colors down the octree graph. The algorithm also allows for different level of details - usable e.g. to adapt for variations in available data rate or to adapt for current detail requirement in rendering process. Currently the algorithm does not use temporal compression approaches, that would enable lower data rates in situations where the 3D scene does not change much from frame to frame – as compared to MPEG video compression where this approach is employed and is extremely effective. However, some work in this direction may be done for the next version of the standard.

Preliminary analysis

For G-PCC some of the above data sets have been compared. For example, in a scene with 100k points at 10 fps, corresponding to 110 MBit/s uncompressed data rate, a compressed rate down to about 24 MBit/s could be achieved with good quality.

Further tests are required, but from this preliminary analysis, we can conclude that the V-PCC encoding time is too much high for our target requirements, while G-PCC approach would be a better starting point. Anyway, G-PCC has no support for taking into account visibility of the 3D points. Hence, the key steps of the development are:

- to exploit the visibility information to reduce the amount of data required by the edge device the viewpoint information can be used also to make the generation of views more efficient
- to do more accurate tests about the performance of the G-PCC, since many parameters can be tuned
- to evaluate if the compression scheme investigated in section 5.5.3 (the ones considered promising) are a valid alternative to the MPEG standards due to their light computational complexity

5.5.3 PC generation module and first prototype of the algorithm

Before a point cloud can be compressed, it needs to be generated. Typically, one creates a point cloud from a static 3D scene/3D model, which then can be watched from different angles at different level of details. In this case the point cloud is often directly generated from triangles or 3D-mesh.

In the context of the UC1-3 Holo Assistant a different approach was chosen. The main goal is to convert the visual output of any 3D-application with any content including animations, complex materials and lighting into a video based, streamable 3D point cloud. The advantage is that such a point cloud enables to generate the required views from certain directions locally at the end user device, while the actual 3D-content is managed and rendered somewhere else, e.g. in the cloud. This has following advantages: first the certain views required by the output device, e.g. a holographic 3D display, are generated with very low delay independent from actual network performance. Secondly, the end user device could be something like a thin client, thus it needs only to output the required views and does not need to render high fidelity 3D-content. This is comparable to actual 2D based game streaming services commercially available. These approaches cannot be used for XR or holographic devices where the observer needs different views dependent from his own (head-) location (cf. VR/AR headsets or holographic 3D devices with head- or eye-tracking).

Thus, in this case, the point cloud is generated from GPU renderings of the 3D scene in Unity 3D from different viewpoints (one RGB and depth image per view, see Figure 50) and then merged into a single or multiple point clouds. Compared to typical point cloud data sets where the data provides information from all watching directions, full details are in this case visible only from certain angular ranges. These limited valid viewing ranges or zones are generated from the different provided views mentioned above. This concept has the advantage to dramatically reducing overall amount of required 3D points in the point cloud to enable more efficient compression and frame by frame-based transfer of point cloud-based video. Frame by frame-based point cloud data will also enable the opportunity to make use of differences between point cloud frames, so for quite static 3D scenes with limited changes from frame to frame, a lower number of changing 3D points is to be expected so this can be used for efficient compression and transfer of video-based point cloud data.

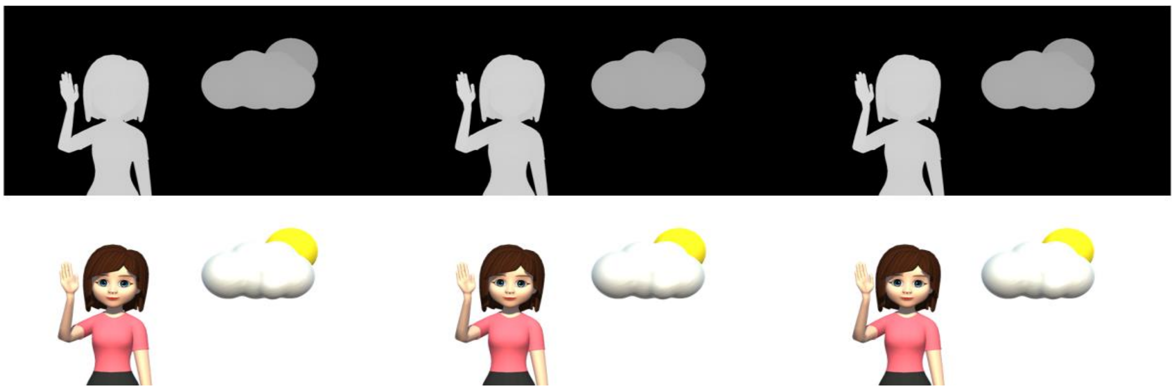


Figure 50: Example of three slightly different views (depth + RGB data). These views can be merged together to form the point cloud

Nevertheless, this step is costly simply due to the large number of views and 3D points to be processed. One approach is to use an octree technique to reduce the number of 3D points by applying hierarchical 3D rasterization however octree implementation suffers for the number of lookups and poor memory coherence.

The point cloud resulting from merging depth maps from slightly different views, can be more efficiently represented as a 2D depth map with colour information and additional points whenever a jump in depth occurs: (see Figure 51).

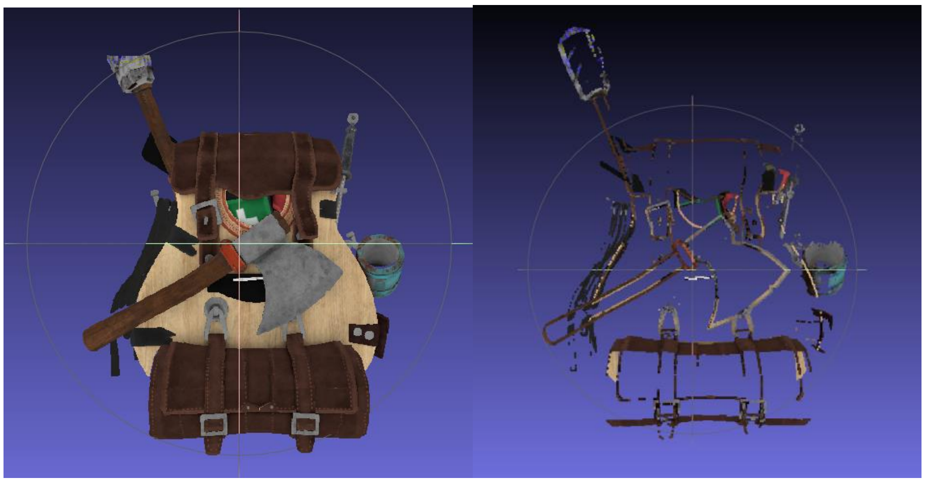


Figure 51: (Left) Depth+RGB, central view (Right) Hidden points revealed through the others views

For these reasons, a more suitable data structure is a 2D grid where each element is the start of a linked list (usually containing zero or one element). A similar structure is a hash map with linked lists to resolve conflicts, but in our case an 'identity' hash already guarantees very few collisions, optimal memory coherence and O(1) access time.

To extract this merged mesh from the several depth maps+RGB generated from different viewpoints, we can start from one of the views (V0) depth map, then for each pixel of the following views (V1) we un-project in world space and re-project in V0 space, compare the depth to determine if the point is already present and if not add it to the merged point cloud (see Figure 52).

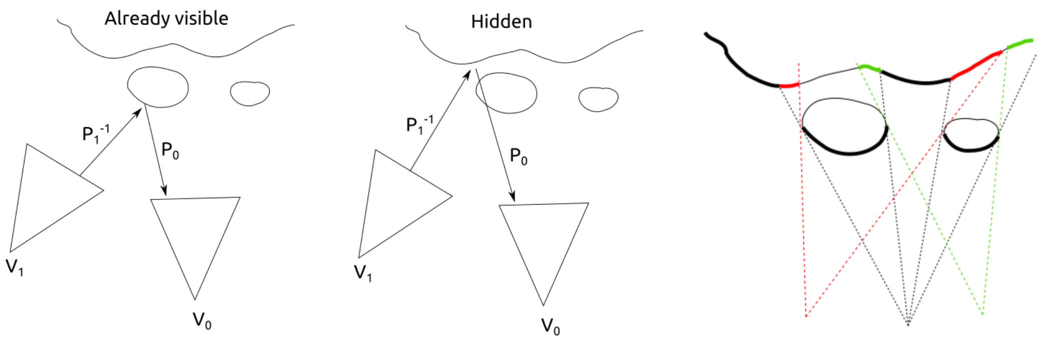


Figure 52: Depth map can be used to find the hidden points using projection between different views. In green and red the points revealed by this operation.

We can process each depth map line by line and exploit the extreme memory coherence.

We can move the bulk of these computations in GPU: Instead of saving depth map and RGB, we project each point in a common final voxel space saving x y and z as additional attributes. While data size increases, we save matrix multiplication per pixel in CPU.

We tested this strategy and we can process 25 views 800x600 in CPU in 30ms, including rendering and transfer of the views depth maps from GPU to CPU for a simple dataset.

An additional optimization is to use the first depth-map as a texture and the following renderings can directly compare each pixel with the corresponding (projected and in the first view space) pixel in the texture and write the 'hidden' pixels only if the depth does not match.

Finally, the few 'hidden' pixels can be directly written to an array or compacted in a second pass, to minimize the amount of data transferred back to the CPU.

The final data to be compressed consists of a depth+rgb map, where depth is quantized accordingly to the precision needed by the hologram projector, and a small array of xyz+rgb points.

Figure 53 shows an example of the merged point cloud visualized from a different, invalid perspective, so one can see the missing data. In addition, the result of rendering the point cloud from a valid angle is shown.

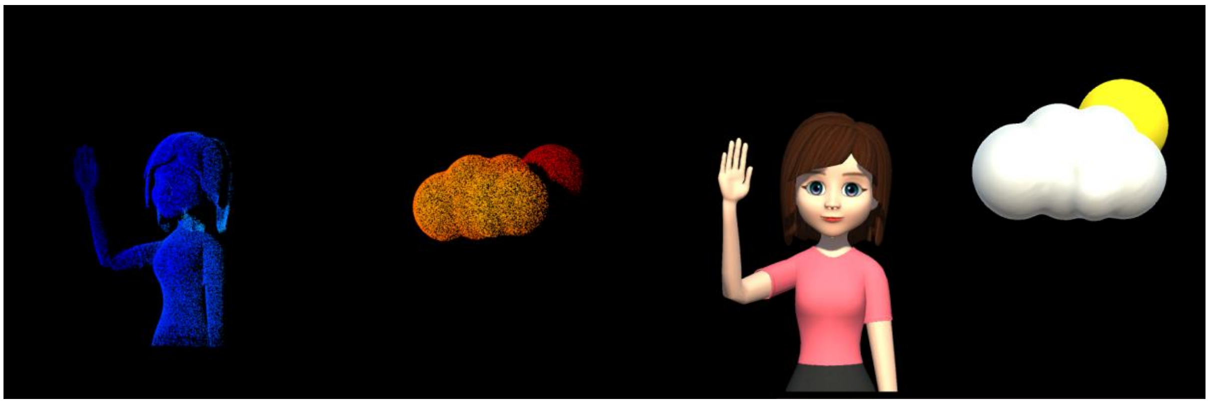


Figure 53: (Left) Example of a 3D Point Cloud visualization; (Right) Rendered final result after reconstructing into an image

5.5.4 Point cloud compression – first evaluations

Data compression trades CPU computation and latency for reduced network bandwidth usage: the effectiveness of a compression algorithm depends both on compression ratio which determines the bandwidth reduction and on compression and decompression speed. However, long computational time might negate the bandwidth advantage.

Compression and decompression speed of an algorithm have always played a crucial role in determining its success, where good compression performances are especially difficult to obtain. Historically, in Computer Graphics, geometry compression algorithm competition has been focused mainly if not almost exclusively on compression ratio, and consequently widely used compression algorithm has become available only very recently when good performances combined with fast decompression have become possible, (Draco [15], Corto [18], Potree [19]) especially on the Web where the limited performance of JavaScript prevented a solution for a long time, while at the same time, bandwidth limitations made the problem more pressing.

We performed an initial evaluation of the performances of the available open-source libraries on a sample point cloud containing 20K points with colour information, weighting 570KB in raw binary format. All tests were performed using the same attribute and position quantization and a single thread processing. Results are reported in Table 24.

Table 24: Evaluation geometry compression algorithm

Algorithm	Compression time in seconds	Compressed size
Quantization	< 0.001s	140KB
gzip -1	0.004s	100KB
gzip: -7	0.018s	90KB
Corto	0.005s	71KB
Dracol	0.030s	71KB (missing colours!)
Tmc13	0.138s	53K

All geometry compression algorithms perform some form of quantization on the vertex position and attributes. Due to the limited size of the dataset, drastic quantization can be performed on the positions (from 32 bits to 11 bits per coordinate) at a negligible cost in quality. Larger datasets can be easily cut into blocks so the numbers from this experiment remain significant.

As a comparison we tested a zip library (actually zlib), a general-purpose compression algorithm. The low compression ratio is mainly due to the fact that it cannot exploit the geometric coherence of the point cloud. Due to the relatively low compression ratio, there is a small difference in compression ratio when changing the dictionary length of the algorithm. On the other hand, large dictionaries become a large penalty in decompression time (4 times here) mostly due to the fact that the dictionary will not fit in the L2 cache generating many cache misses. Other entropy compression algorithms (LZ4 for example) have been tested, with much faster compression timings but worse compression ratio.

Corto [18] adopts a very simple Morton-code based geometry compression with a difference encoder for the attributes (colours in this case).

Tunstall [20] (which is basically a reverse Huffman) is used as an entropy coder due to its extreme speed in decompression while still being fast enough in compression and having compression ratio similar to Huffman. Corto is able to encode five million vertices per second, while decoding at around 25M vertices per second. Adopting Huffman instead would probably reverse the speeds. Other entropy coders could be used and offer different trade-off between speed and compression ratio.

Draco [15] adopts a similar approach based on differences combined with arithmetic entropy coding. Surprisingly the compression ratio is worse while colour information has not been encoded (command line software does not support it). Unsurprisingly, due to more sophisticate entropy coding, the compression timing is 5 times worse. Draco offers much better trade-off for meshes.

Tmc13 [21] offers the best compression ratio (1:10), at the cost of a long processing time 0.14s, 142K triangles per second. This software offers a very large set of parameters to be tuned, coupled with a lack of a decent documentation or guidance. We tested a (very) large number of configurations with mixed results. We are confident that marginally better results can be obtained, the picture is not going to change substantially.

For each compression algorithm, speed and compression ratio defines a bandwidth above which it makes no sense to compress as it would take more time to compress/decompress the data than to send raw, quantized (11 per coordinate 8 per colour channel, for small datasets, in total 58 bits) data.

Tmc13 becomes useful when the network bandwidth is smaller than (58/8)*142K/s ~ 1MB/s, while Corto keeps being competitive up to 58*5M/8 = 36M/s. For bandwidth lower than ~1.3MB/s higher compression ratio of Tmc13 allows to better make use the limited bandwidth.

Since it is relatively easy to perform point data compression in parallel, adding computational power allows Tmc13 to remain competitive with higher available bandwidth.

The compression algorithm could be very easily swapped for a different one at any time in the streaming depending on bandwidth or CPU limitations, and the most promising algorithms to adopt for geometric compression, according to these preliminary investigations are Corto, Tmc13, and also the simple quantization is competitive.

In the case of limited bandwidth, a more aggressive compression strategy is used on video codecs (h264 for example) where the depth+rgb maps can be treated as a video stream, and the array of ‘hidden’ points compressed as before. The main point is that this codec can take into account differences between consecutive frames and drastically cut the bandwidth needed.

Preliminary tests on the backpack dataset shows 20ms are needed for the h264 compression strategy, enough for 30fps, but the ability to only encode differences shows the savings in bandwidth are very promising (10x on a few tests, but obviously depends on the dataset, camera movement, animations etc.).
Tweaking the compression parameters allows also to control the trade-off between computational cost and compression ratio.

5.5.5 Conclusions

From the set of tests we concluded that the most promising approach is to use a 2D map + collision data structure, process the different views directly in GPU and use H264 for the 2D map and the extra, conflicting points encode the differences between successive points using a simple entropy coding (being too sparse to really take advantage of octree based point cloud compression algorithms).

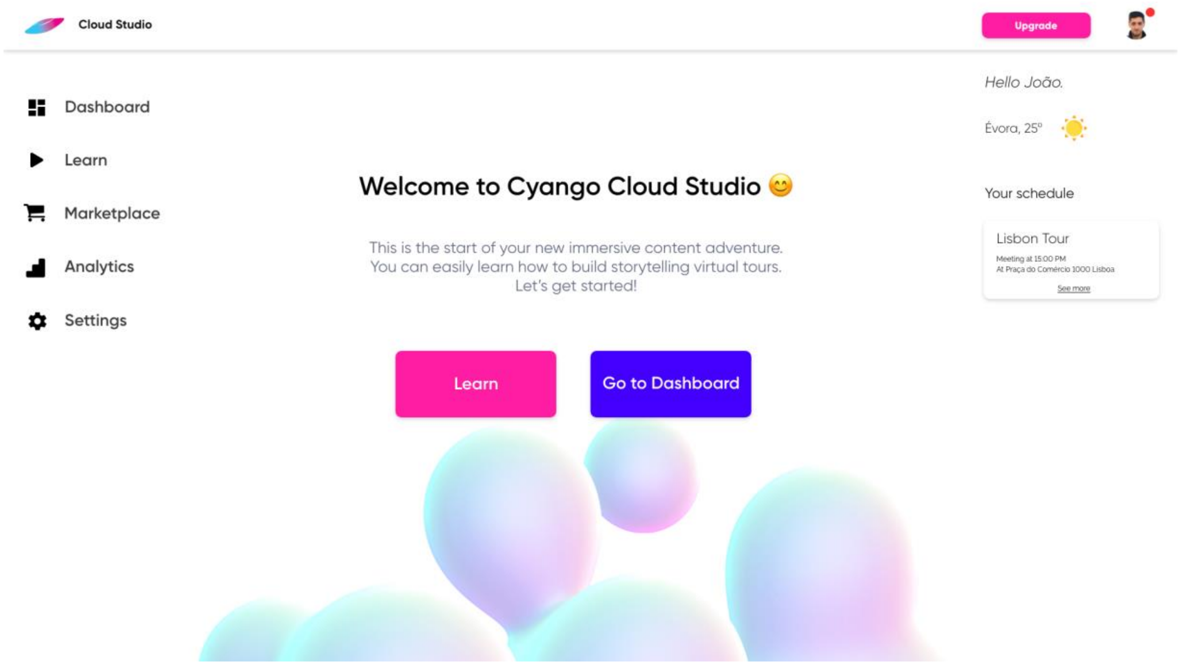
The implementation of the first prototype is ongoing. We expect to have the working version of the PC E/D available for the end of January 2023.

5.6 Video streaming and platform development

Cyango Cloud Studio is the name of one of the software platforms related to the UC2-2 VR Tour Creator Application in CHARITY. Cyango Cloud Studio allows any user to create virtual experiences that can be used for all industry verticals, especially education and tourism. It is a web-based platform and the main goal is to host the micro services inside CHARITY cloud to provide a better performance and better user experience. There are many problems that CHARITY allows us to address, mainly related to the livestreaming, and to the editing in real time of media content.

Cyango Cloud Studio development is progressing. We have been focusing in many features and making sure the user interface and user experience are according to the feedback we gathered via meetings, calls and demos at events showing our software.

One important progress was the migration of the 3D Web engine framework from Aframe[21] to the more modern and compatible framework with React.js, which is called React-Three-Fiber[22]. This change of framework required an extensive code re-factoring, as its logic was different from Aframe. Some months have been spent for the migration. This guarantee better for the scalability of the use case. Basically, we built the already made features again like placing 360 videos on VR, interactive hotspots and other UX features. Additionally, with this new framework we can now have a smooth coding always inside React environment. We also re-designed the UI of the platform using Adobe XD, and in the following months we will start implementing this new design.



Welcome to Cyango Cloud Studio 😊

This is the start of your new immersive content adventure.
You can easily learn how to build storytelling virtual tours.
Let's get started!

Hello João.

Évora, 25º

Your schedule

Lisbon Tour
Meeting at 15:00 PM
At Praça do Comércio 1000 Lisboa
See more

Figure 54: New design of Cloud Studio (screenshot 1)

[21] https://aframe.io

[22] https://github.com/pmndrs/react-three-fiber

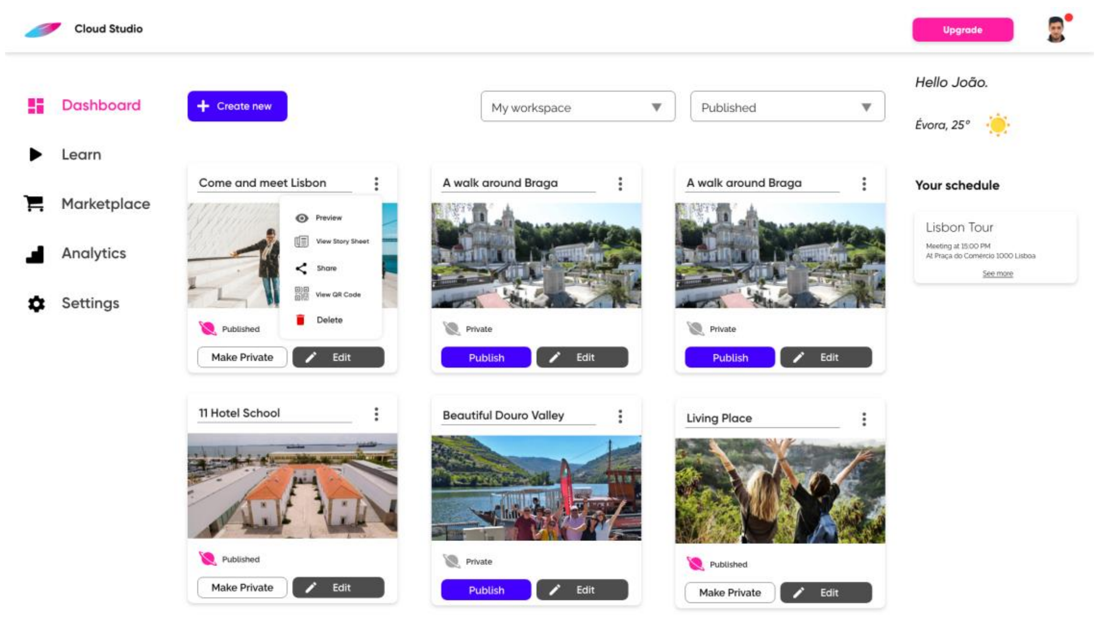


Figure 55: New design of Cloud Studio (screenshot 2)

The images above (Figure 54 and Figure 55) are some screens of the whole platform redesign. We also made important research about how to achieve a important feature the users requested, which is the online video editor, that allows the user to edit the video and audio. This video editor tool will be based on the FFMPEG+WASM[23], a pure web assembly port of FFMPEG, that allows to edit video, audio and stream inside the browser. We also designed a screen of how this video editor tool would be like in Cyango Cloud Studio, shown in Figure 56.

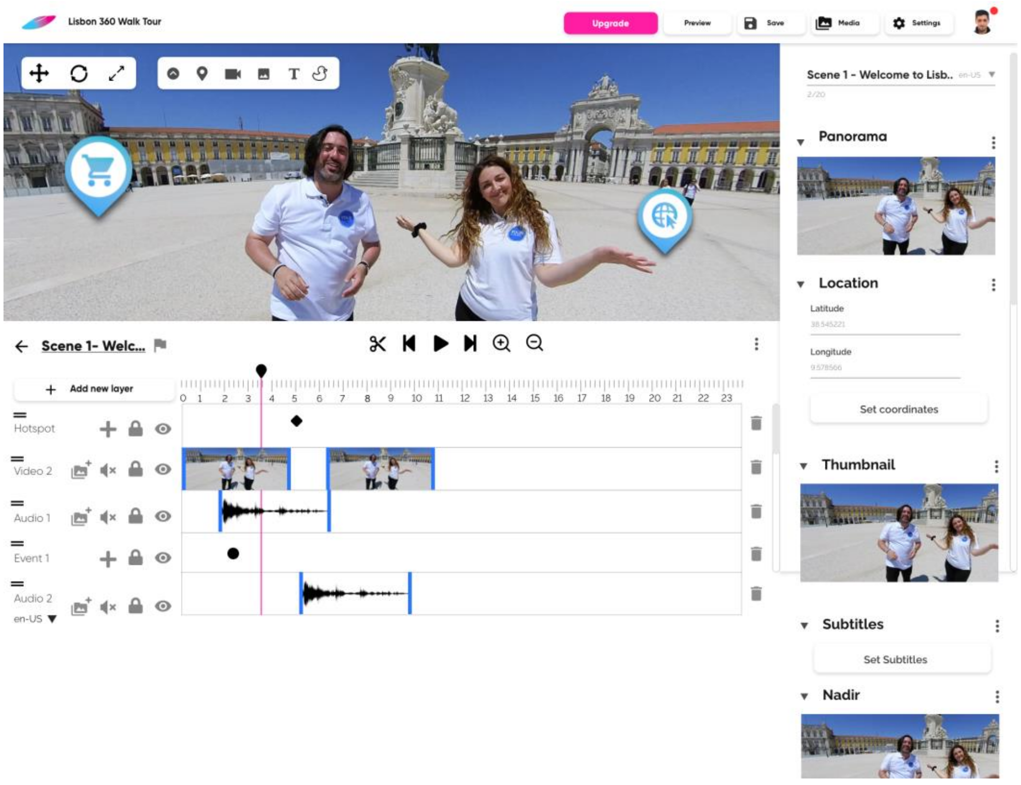


Figure 56: Video editor tool

[23] https://github.com/ffmpegwasm/ffmpeg.wasm

DOTES implemented also the first livestreaming feature that is under optimization.

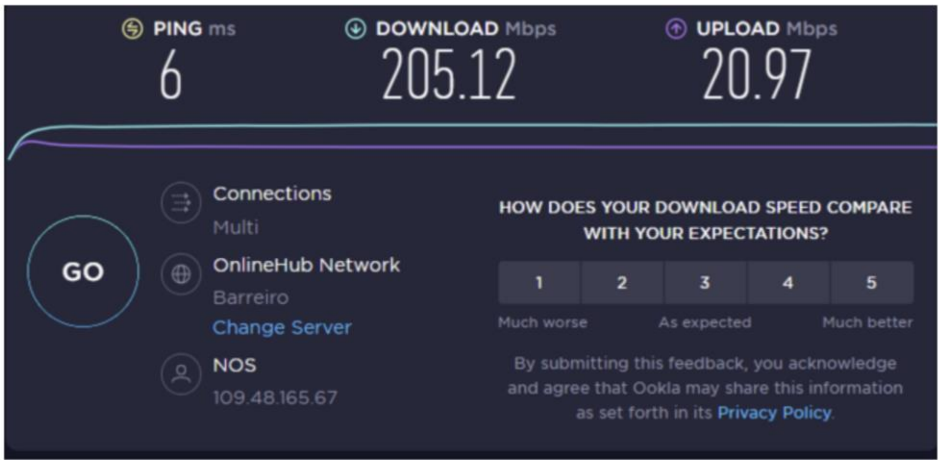
Regarding the performance, a series of livestreaming tests using a high quality 360 camera and a server of the company has been conducted. These tests allow to understand what factors are preventing a good user experience.

The setup used in such tests were: 360 camera streaming in Lisbon, Portugal and the consumer user located in Évora, Portugal. The 360 camera was streaming to a service inside a docker container hosted in our Synology NAS 918+[24] . This container is built on:

- Nginx 1.17.5 (compiled from source)
- Nginx-rtmp-module 1.2.1 (compiled from source)
- FFmpeg 4.2.1 (compiled from source)

and allows to stream to a RTMP url using a server public IP address, and then the front-end app consumes the url called https://live.cyango.com . This url points to the docker container in a server of the DOTES. This docker container receives a video stream from the 360° camera via RTMP and then uses ffmpeg to convert the video in real-time to the HLS format so we can consume it on the front-end.

The network parameters of each endpoint are the following:


PING ms 6
DOWNLOAD Mbps 205.12
UPLOAD Mbps 20.97

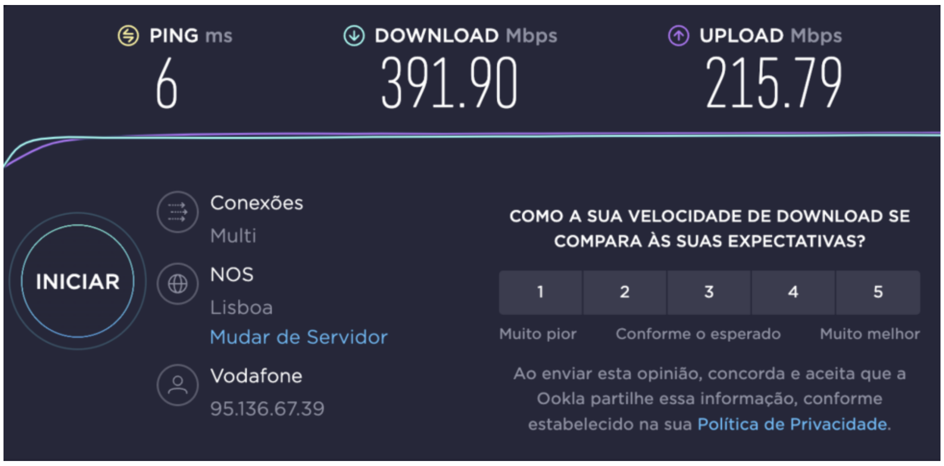
Connections
Multi
OnlineHub Network
Barreiro
Change Server
NOS
109.48.165.67

HOW DOES YOUR DOWNLOAD SPEED COMPARE WITH YOUR EXPECTATIONS?

1 2 3 4 5
Much worse · As expected · Much better

By submitting this feedback, you acknowledge and agree that Ookla may share this information as set forth in its Privacy Policy.

Figure 57: Camera end network settings


PING ms 6
DOWNLOAD Mbps 391.90
UPLOAD Mbps 215.79

Conexões
Multi
NOS
Lisboa
Mudar de Servidor
Vodafone
95.136.67.39

COMO A SUA VELOCIDADE DE DOWNLOAD SE COMPARA ÀS SUAS EXPECTATIVAS?

1 2 3 4 5
Muito pior · Conforme o esperado · Muito melhor

Ao enviar esta opinião, concorda e aceita que a Ookla partilhe essa informação, conforme estabelecido na sua Política de Privacidade.

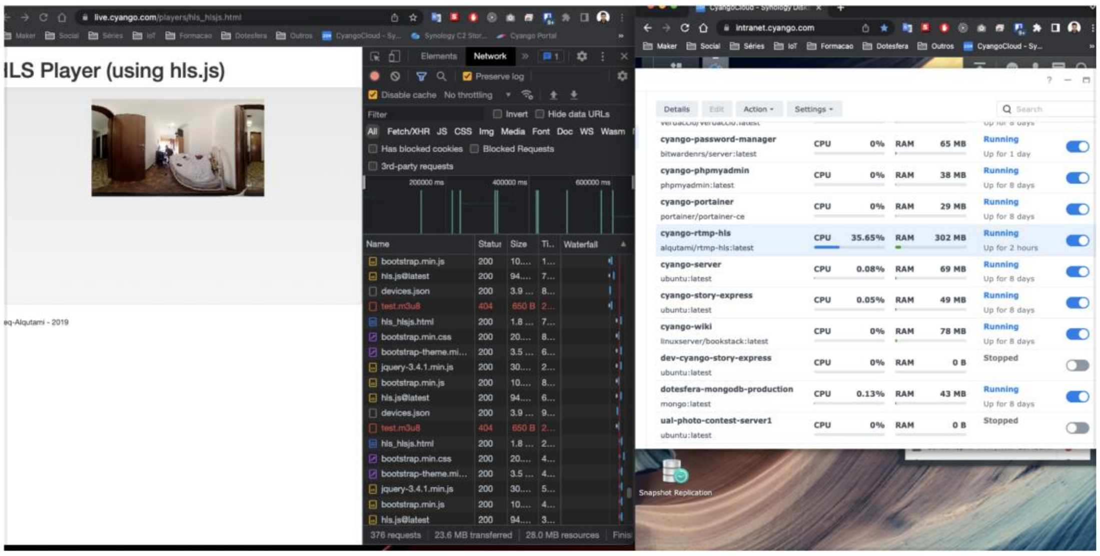
Figure 58: End user network settings

We did some tests with different parameters as detailed below. These tests were conducted to understand the performance of the algorithm and protocols we are using in Cyango Cloud Studio**.**

[24] https://www.storagereview.com/review/synology-diskstation-ds918-review

Test 1

In the first test the camera was streaming video at 4k 3840x2160 with a bit rate of 10MB/s. In this first test we experienced an high number of video stops during playing, approximately 10 times per 20 seconds of streaming. An accurate perceptual measure of this problem is under evaluation but the streaming quality has shown to be clearly insufficient.



Name	Status	Size	Ti...	Waterfall
bootstrap.min.js	200	10....	1...	
hls.js@latest	200	94....	7...	
devices.json	200	3.9 ...	8...	
test.m3u8	404	650 B	2...	
hls_hlsjs.html	200	1.8 ...	7...	
bootstrap.min.css	200	20....	8...	
bootstrap-theme.mi...	200	3.5 ...	6...	
jquery-3.4.1.min.js	200	30....	2...	
bootstrap.min.js	200	10....	8...	
hls.js@latest	200	94....	6...	
devices.json	200	3.9 ...	9...	
test.m3u8	404	650 B	2...	
hls_hlsjs.html	200	1.8 ...	2...	
bootstrap.min.css	200	20....	4...	
bootstrap-theme.mi...	200	3.5 ...	4...	
jquery-3.4.1.min.js	200	30....	5...	
bootstrap.min.js	200	10....	4...	
hls.js@latest	200	94....	3...	

376 requests | 23.6 MB transferred | 28.0 MB resources | Finis

cyango-password-manager bitwardenrs/server:latest	CPU	0%	RAM	65 MB	Running Up for 1 day	●
cyango-phpmyadmin phpmyadmin:latest	CPU	0%	RAM	38 MB	Running Up for 8 days	●
cyango-portainer portainer/portainer-ce	CPU	0%	RAM	29 MB	Running Up for 8 days	●
cyango-rtmp-hls alqutami/rtmp-hls:latest	CPU	35.65%	RAM	302 MB	Running Up for 2 hours	●
cyango-server ubuntu:latest	CPU	0.08%	RAM	69 MB	Running Up for 8 days	●
cyango-story-express ubuntu:latest	CPU	0.05%	RAM	49 MB	Running Up for 8 days	●
cyango-wiki linuxserver/bookstack:latest	CPU	0%	RAM	78 MB	Running Up for 8 days	●
dev-cyango-story-express ubuntu:latest	CPU	0%	RAM	0 B	Stopped	○
dotesfera-mongodb-production mongo:latest	CPU	0.13%	RAM	43 MB	Running Up for 8 days	●
ual-photo-contest-server1 ubuntu:latest	CPU	0%	RAM	0 B	Stopped	○

Figure 59: Screenshot of the livestream test

Test 2

In this test we lowered the camera settings to 1440P 2560x1440 with a bit rate of 10MB/s, and still experienced video stops similar to test 1.

Test 3

We lowered the camera settings to 1080P 1920X1080 with a bit rate of 10MB/s, and still experiencing the same as test 1 and test 2.

Test 4

In this test we used the camera settings as 960P 1920x960 with a bit rate of 5MB/s. And in this test the video plays without stops, but we noticed about a 3 minutes delay. We could confirm this delay, because we had a phone call between the two DOTES collaborators confirming the delay.

Test 5

In this test we lowered the camera settings to 720P 1440x720 and a bit rate of 5MB/s. In this case the video plays without stops and with a delay of about 45 seconds, using the same process as test 4.

From these preliminary tests, we conclude that the server we used is the major factor of the delay, because it does not have good hardware resources to quickly transcode the video coming from the stream to HLS. In the next, we exploit resources made available by CHARITY partners to make additional tests. Also, some tweaks could be done on the algorithm approach. In the next tests iteration, we will research about Low latency HLS[25] to assess the latency reduction using this protocol.

[25] https://developer.apple.com/documentation/http_live_streaming/enabling_low-latency_http_live_streaming_hls

5.7 Mesh Merger

Initially, UC3-1 Collaborative Gaming Application, required *Mesh Collider* service. This service is based on the point cloud data gathered on the mobile devices through RGB cameras and it creates a set of well-defined polygons to allow a clean and clear interaction with the environment. We found out, in the course of the research, that in many cases, 3D points reconstructed through RGB cameras cannot reach high quality to obtain such clean geometry in in many cases (as shown in Figure 60). Main issues when scanning using this method are:

- Noise/phantom data: point generated in random locations not connected to the environment features.
- No point generated at flat surfaces: flat surfaces was treated as empty space. This happens also for other featureless surfaces.
- Low precision: sometime low precision of feature points localisation.

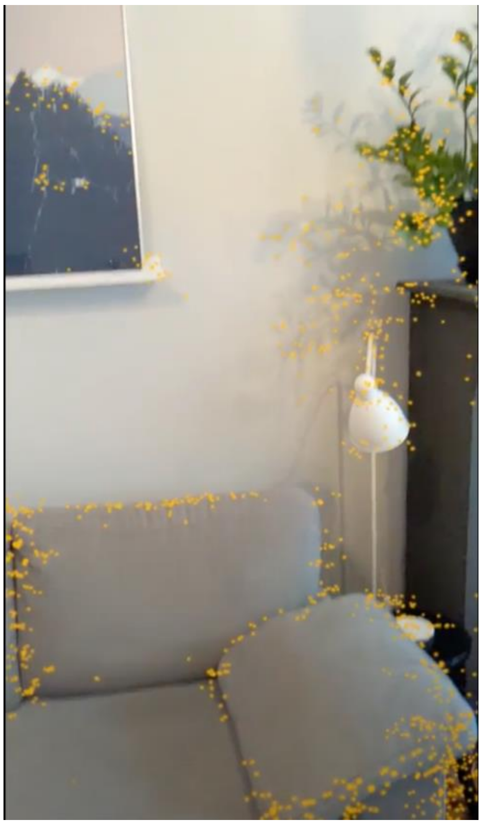


Figure 60: Environment scanning using RGB method on Android device

To overcome the aforementioned problems, and to take into account that future mobile devices will be more and more equipped with 3D sensors, we will move on this type of smartphone. LiDAR is nowadays available on selected Apple devices (iPhones and iPads Pro), for this reason it has been decided to test it along with ARKit available for Apple devices. The benefit of using ARKit is that it offers additional functionalities like Mesh Collider (see Figure 61). The results of some reconstruction tests were very promising: the quality of scanned data is very high and mesh colliders generated automatically by ARKit have good geometric properties: continuity (no gaps) and simplicity (low number of triangles) (see Figure 61).

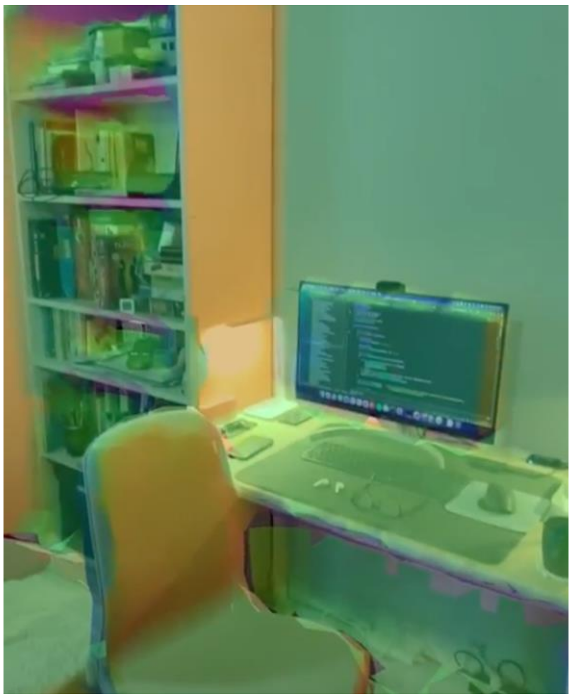


Figure 61: Environment scanning using LiDAR with instant mesh collider building

Since ARKit provides proper mesh collider generation functionality, it has been decided to switch the focus on *merging* mesh colliders coming from different acquisition devices. The idea is to merge mesh colliders that are scanned and built in the same game session (and physical location) by the gamers. This functionality will significantly enrich the immersion of all participants of the game.

Each participant equipped with smartphone with LiDAR scans a fragment of the environment, ARKit builds a mesh collider from the scanned data and all mesh colliders are sent to the *Mesh Merger* service (see Figure 62) developed in the ambit of the Task 3.4. This service merges all mesh colliders into one common mesh collider. The merged version of the collider is then sent back to all devices. This way all participants will be able to interact with a continuously update version of the mixed environment. The Mesh Merger is in its first stage of the development and, at the moment, we do not have preliminary results to show.

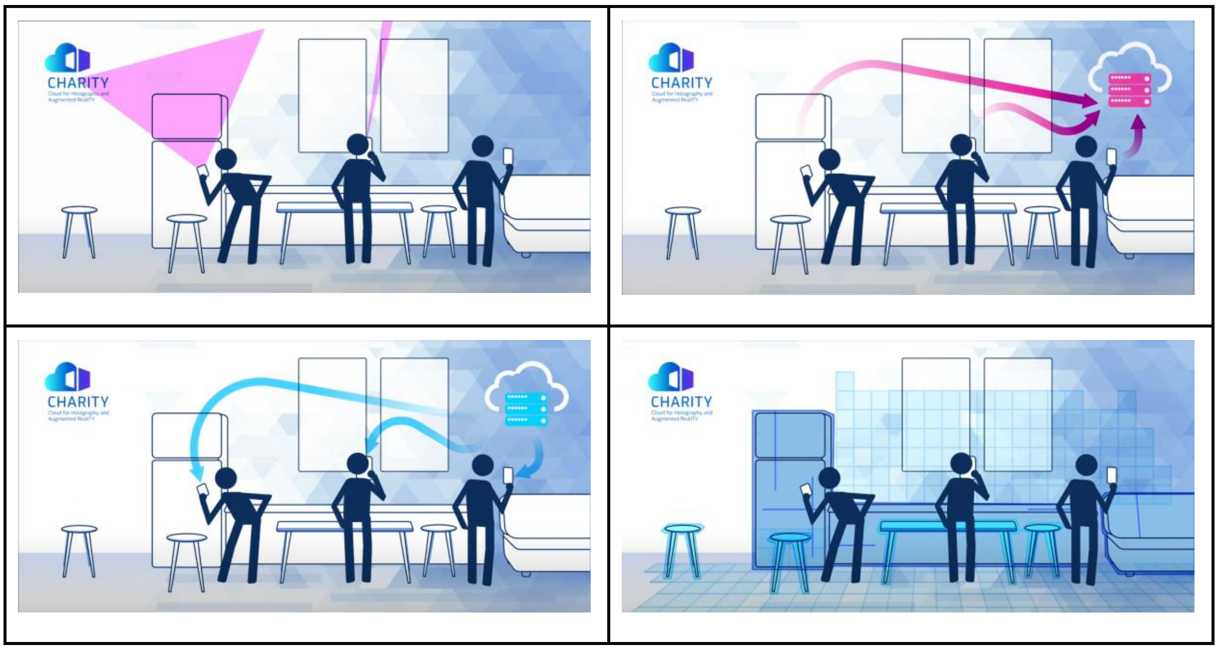


Figure 62: Merging mesh colliders

6 Conclusions

The description of the research and development work conducted in the WP3 and the results reported in this deliverable, demonstrate the big effort put in developing innovative solutions for XR applications, both from a scientific and a technological point of view. In fact, we do not only propose and design new technical solutions, but we also develop novel algorithms. Regarding the prototypes, even if some implementation activities are experiencing some delays, in general the direction is well established, and the prototypes of some components are not far to be achieved. The monitoring framework and the Mesh Service for applications adaptation have been carefully designed, spending a lot of work to fulfil the needs and the requirements of the CHARITY project, and it will be ready in a short time (the monitoring framework, in particular). The CHES is going to be released as open-source software soon. Some of the XR-enabling technologies that we are developing, like the adaptive rendering solutions and the Mesh Merger are still in the first implementation stage but, reasonably, the first prototypes will be available in about 3-4 months. The first working prototype of the Point Cloud E/D is planned to be released before the end of January 2023.

References

[1] Baresi, Luciano, and Danilo Filgueira Mendonça. "Towards a serverless platform for edge computing." 2019 IEEE International Conference on Fog Computing (ICFC). IEEE, 2019.

[2] Gkoufas, Yiannis, and David Yu Yuan. "Dataset Lifecycle Framework and its applications in Bioinformatics." *arXiv e-prints* (2021): arXiv-2103.

[3] Koutsovasilis, Panos, et al. "A Holistic Approach to Data Access for Cloud-Native Analytics and Machine Learning." *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021.

[4] Jens Meinicke, Chu-Pan Wong, Bogdan Vasilescu, and Christian Kästner. 2020. Exploring differences and commonalities between feature flags and configuration options. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice* (ICSE-SEIP '20). Association for Computing Machinery, New York, NY, USA, 233–242

[5] W. Li, Y. Lemieux, J. Gao, Z. Zhao and Y. Han, "Service Mesh: Challenges, State of the Art, and Future Research Opportunities," 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), 2019, pp. 122-1225.

[6] Svahnberg, M., Gurp, J., Bosch, J. On the Notion of Variability in Software Product Lines. Blekinge Institute of Technology Research Report No 02/01. (2001)

[7] Bashari, M., Bagheri, E., Weichang Du, W. Dynamic Software Product Line Engineering: A Reference Framework. International Journal of Software Engineering and Knowledge Engineering, Vol. 27, No. 2 (2017) 191–234

[8] Raatikainen, M., Tiihonen, J., Männistö, T. Software product lines and variability modeling: A tertiary study,J Systems and Software, Volume 149, Pages 485-510 (2019)

[9] Berger, T., Steghöfer, JP., Ziadi, T. et al. The state of adoption and the challenges of systematic variability management in industry. Empir Software Eng 25, 1755–1797 (2020).

[10] Reisner, E., Song, C., Ma, K., Foster, J., Porter, A. Using symbolic evaluation to understand behavior in configurable software systems. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (2010)

[11] Mendonca, N., Jamshidi, P., Garlan, D., Pahl, C., Developing Self-Adaptive Microservice Systems: Challenges and Directions. IEEE Software, vol. 38, no. 02, pp. 70-79, 2021.

[12] J. O. Kephart and D. M. Chess, "The vision of autonomic computing. *Computer*, vol. 36, no. 1, pp. 41-50, Jan. 2003

[13] J. Floch1 et al. Playing MUSIC - Building context-aware and self-adaptive mobile applications. *Software: Practice and Experience*. 43. 359-388. (2013)

[14] G. Alfrez, V. Pelechano, R. Mazo, C. Salinesi and D. Diaz, Dynamic adaptation of service compositions with variability models, *J. Syst. Softw*. 91 (2014) 24–47.

[15] R. Andrade, M. Ribeiro, H. Rebêlo, P. Borba, V. Gasiunas and L. Satabin, Assessing idioms for a flexible feature binding time, Comput. J. 59(1) (2015) 1–32.

[16] Google 2017. Draco: 3D data compression. https://google.github.io/draco/. Accessed: 2022-05-11.

[17] Max Limper, Stefan Wagner, Christian Stein, Yvonne Jung, and André Stork.2013. Fast Delivery of 3D Web Content: A Case Study. In Proceedings of the 18th International Conference on 3D Web Technology (San Sebastian, Spain) (Web3D ’13). Association for Computing Machinery, New York, NY, USA, 11–17. https://doi.org/10.1145/2466533.2466536

[18] Federico Ponchio and Matteo Dellepiane. 2016. Multiresolution and fast decompression for optimal web-based rendering. Graphical Models 88 (2016), 1 – 11.

https://doi.org/10.1016/j.gmod.2016.09.002

[19] Markus Schütz. 2016. Potree: Rendering Large Point Clouds in Web Browsers. Ph. D. Dissertation.

[20] Tunstall, Brian Parker (September 1967). Synthesis of noiseless compression codes. Georgia Institute of Technology.

[21] H. Liu, H. Yuan, Q. Liu, J. Hou and J. Liu, "A Comprehensive Study and Comparison of Core Technologies for MPEG 3-D Point Cloud Compression," in IEEE Transactions on Broadcasting, vol. 66, no. 3, pp. 701-717, Sept. 2020, doi: 10.1109/TBC.2019.2957652.

[22] https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar, accessed 20-06-2022.

[23] Jeanette Ling, Rockwell Collins. Understanding Cloud-Based Visual System Architectures. Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2018.

[24] Teemu Kämäräinen, Matti Siekkinen, Jukka Eerikäinen, and Antti Ylä-Jääski. CloudVR: Cloud Accelerated Interactive Mobile Virtual Reality. Proceedings of the 26th ACM international conference on Multimedia (MM '18). Association for Computing Machinery, New York, NY, USA, 1181–1189.

[25] Mark Claypool, Kajal Claypool. Latency and Player Actions in Online Games. Communications of the ACM, November 2006, Vol. 49 No. 11, Pages 40-45

[26] Waveren, J., The Asynchronous Time Warp for Virtual Reality on Consumer Hardware, 22nd ACM Conference on Virtual Reality Software and Technology, 2016

[27] Nicholson, N. “Exploring ‘Negative Latency’”, December 2019, https://nolannicholson.com/2019/12/16/exploring-negative-latency.html

[28] Makris A, Kontopoulos I, Psomakelis E, Xyalis SN, Theodoropoulos T, Tserpes K. Performance Analysis of Storage Systems in Edge Computing Infrastructures. Applied Sciences. 2022; 12(17):8923. https://doi.org/10.3390/app12178923

[29] Antonios Makris, Evangelos Psomakelis, Theodoros Theodoropoulos, and Konstantinos Tserpes. 2022. Towards a Distributed Storage Framework for Edge Computing Infrastructures. In Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge (FRAME '22). Association for Computing Machinery, New York, NY, USA, 9–14. https://doi.org/10.1145/3526059.3533617

[30] X.Hou,Y.Lu,andS.Dey,“WirelessVR/ARwithedge/cloudcomputing,” in Proc. Int. Conf. Comput. Commun. Netw., 2017, pp. 1–8.

[31] Sebastian Friston, Tobias Ritschel, and Anthony Steed. 2019. Perceptual rasterization for head-mounted display image synthesis. ACM Trans. Graph. 38, 4, Article 97 (August 2019), 14 pages. https://doi.org/10.1145/3306346.3323033

[32] L. Fink, N. Hensel, D. Markov-Vetter, C. Weber, O. Staadt and M. Stamminger, "Hybrid Mono-Stereo Rendering in Virtual Reality," *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2019, pp. 88-96, doi: 10.1109/VR.2019.8798283.

[33] Jie Guo, Xihao Fu, Liqiang Lin, Hengjun Ma, Yanwen Guo, Shiqiu Liu, and Ling-Qi Yan. 2021. ExtraNet: Real-time Extrapolated Rendering for Low-latency Temporal Super-sampling. ACM Trans. Graph. 40, 6, Article 278 (December 2021), 16 pages. https://doi.org/10.1145/3478513.3480531

[34] X. Hou and S. Dey, "Motion Prediction and Pre-Rendering at the Edge to Enable Ultra-Low Latency Mobile 6DoF Experiences," in *IEEE Open Journal of the Communications Society*, vol. 1, pp. 1674-1690, 2020, doi: 10.1109/OJCOMS.2020.3032608.

[35] K. Boos, D. Chu, and E. Cuervo, “Flashback: Immersive virtual reality on mobile devices via rendering memoization,” in *Proc. Int. Conf. Mobile Syst. Appl. Services*, 2016, pp. 291–304.

[36] L. Liu *et al.*, “Cutting the cord: Designing a high-quality untethered VR system with low latency

remote rendering," in *Proc. Int. Conf.* Mobile Syst. Appl. Services, 2018, pp. 68–80.

[37] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human trajectory prediction in crowded spaces," in Proc. Conf. Comput. Vis. Pattern Recognit., 2016, pp. 961–971.

[38] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social GAN: Socially acceptable trajectories with generative adversarial networks," in Proc. Conf. Comput. Vis. Pattern Recognit., 2018, pp. 2255–2264.

[39] J. Martinez, M. J. Black, and J. Romero, "On human motion prediction using recurrent neural networks," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4674-4683.

[40] J. Butepage, M. J. Black, D. Kragic, and H. Kjellstrom, "Deep representation learning for human motion prediction and classification," in Proc. Conf. Comput. Vis. Pattern Recognit., 2017, pp. 1591–1599.

[end of document]