D3.1: Initial versions of XR enablers

Revision: V.1.0

Work package	WP3
Task	Task 3.1, Task 3.2, Task 3.3, Task 3.4, Task 3.5, Task 3.6
Due date	30/06/2024
Submission date	26/06/2024
Deliverable lead	VICOM
Version	1.0
Authors	Roberto Viola (VICOM), Inhar Yeregui (VICOM), Daniel Mejías (VICOM), Arne Erdmann (RAY), Chathura Sarathchandra (IDE), Mario Montagud (i2CAT), Isaac Fraile (i2CAT), Jaume Moragues (i2CAT), Aurora Ramos (CGE), Mariana Kyrova (MATSUKO), Michal Szakala (MATSUKO), Rafael Rosales (INTEL), Fernando Pargas (TID)
Reviewers	Sherif Adeshina Busari (IT), Valerio Frascolla (INT), Mohammed Al-Rawi (IT)
Abstract	The 6G-XR project is dedicated to building an advanced infrastructure for eXtended Reality (XR) services, including the uses cases of Augmented Reality (AR) exploiting network control plane and Virtual Reality (VR) based on network user plane. This deliverable D3.1 describes the initial versions of the developed XR Enablers, consisting in Multimedia Functions to enable the end-to-end multimedia pipeline for AR/VR services. These XR Enablers convers multi-sensor volumetric capture and reconstruction, cloud/edge XR processing, adaptive and low latency XR delivery, multi-modal synchronization, session management and media orchestration, and KPI monitoring system. The project aims at integrating these XR Enablers within the use cases to conduct trials and evaluate the Key Performance Indicators.
Keywords	5G/6G, Augmented Reality (AR), Virtual Reality (VR), Holographic Communications, User Plane, Control plane, Multimedia Functions, Media Synchronization

Co-funded by the European Union

WWW.6G-XR.EU

Grant Agreement No.: 101096838
Call: HORIZON-JU-SNS-2022

Topic: HORIZON-JU-SNS-2022-STREAM-C-01-01
Type of action: HORIZON-JU-RIA

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	20/11/2023	First version of the ToC for comments	VICOM
V0.2	20/05/2024	Full draft version	All partners
V0.3	31/05/2024	Version after external review	All partners
V0.4	12/06/2024	Technical manager review	IT
V0.5	21/06/2024	Final draft version	All partners
V1.0	26/06/2024	Final version	VICOM

DISCLAIMER

Project funded by

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Economic Affairs, Education and Research EAER
State Secretariat for Education, Research and Innovation SERI

The 6G-XR (*6G eXperimental Research infrastructure to enable next-generation XR services*) project has received funding from the **Smart Networks and Services Joint Undertaking (SNS JU)** under the European Union's **Horizon Europe research and innovation programme** under Grant Agreement No 101096838. This work has received funding from the **Swiss State Secretariat for Education, Research, and Innovation (SERI)**.

COPYRIGHT NOTICE

Project co-funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	**R**	
Dissemination Level		
PU	*Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page)*	✓
SEN	*Sensitive, limited under the conditions of the Grant Agreement*	
Classified R-UE/ EU-R	*EU RESTRICTED under the Commission Decision No2015/ 444*	
Classified C-UE/ EU-C	*EU CONFIDENTIAL under the Commission Decision No2015/ 444*	
Classified S-UE/ EU-S	*EU SECRET under the Commission Decision No2015/ 444*	

* R: Document, report (excluding the periodic and final reports)

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

DATA: Data sets, microdata, etc.

DMP: Data management plan

ETHICS: Deliverables related to ethics issues.

SECURITY: Deliverables related to security issues

OTHER: Software, technical diagram, algorithms, models, etc.

EXECUTIVE SUMMARY

This report is the first deliverable (D3.1) of Work Package 3 (WP3) – "XR Enablers" of the 6G-XR project. The purpose of D3.1 is to present the initial versions of the Virtual Network Functions (VNFs) for eXtended Reality (XR) that have been generated in WP3. These VNFs, called XR Enablers, will be developed further during the rest of the WP3 timeline.

The VNFs are meant to be integral part of the network user plane and control plane for enabling real-time holographic communications, including Virtual Reality (VR) and Augmented Reality (AR). Thus, they are necessary to enable the deployment of three of the five planned use cases (UCs) within the 6G-XR project (see D1.1 [1] for further details):

- UC1 - Resolution Adaptation or Quality on Demand
- UC2 - Routing to the Best Edge
- UC3 - Control Plane Optimization

The integration of the VNFs into the three UCs has been carried out at the 6G-XR South Node test facilities, namely 5Tonic[1] (Madrid, Spain) and 5GBarcelona[2] (Barcelona, Spain).

D3.1 offers a comprehensive description of the first versions of the XR Enablers, encompassing their requirements and the employed hardware and software for their development. These XR Enablers consist of multi-sensor volumetric capture and reconstruction, cloud/edge XR processing, adaptive and low latency XR delivery, multi-modal synchronization, session management and media orchestration. Enablers concerning infrastructure configuration and KPI monitoring system complete this report to provide a comprehensive overview of the XR Enablers integration and interoperability with the infrastructure.

[1] https://www.5tonic.org/

[2] https://5gbarcelona.org/

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

2D	2-dimensional
3D	3-dimensional
3GPP	3rd Generation Partnership Project
5G	Fifth Generation
5GC	5G Core
6DoF	6 Degrees of Freedom
AF	Application Function
API	Application Programming Interface
AR	Augmented Reality
AS	Application Server
CM	Clock Manager
ConM	Index / Connection Manager
CP	Control Plane
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DASH	Dynamic Adaptive Streaming over HTTP
DCSF	Data Channel Signalling Function
DMA	Direct Memory Access
DNS	Domain Name System
DPE	Device Packaging Entity
E2E	End-to-End
EC	European Commission
EU	European Union
FoV	Field of View
fps	Frames Per Second
GPS	Global Positioning System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ICE	Interactive Connectivity Establishment
IDE	Integrated Development Environment
IMS	IP Multimedia Subsystem
IMSDC	IMS Data Channel
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol
IPU	Image Processing Unit
KPI	Key Performance Indicator
LF-SDK	Light-Field SDK
LoD	Level of Detail
LTS	Long Term Support
MCU	Multipoint Control Unit
mDNS	multicast DNS
ML	Machine Learning
MLA	Microlens Array
MNO	Mobile Network Operator
MPD	Media Presentation Description
MU	Media Unit

NaaS	Network-as-a-Service
NBI	NorthBound Interface
NEF	Network Exposure Function
NTP	Network Time Protocol
NWDAF	Network Data Analytics Function
OS	Operating System
PC	Personal Computer
PCC	Policy and Charging Control
PCF	Policy Control Function
PDU	Protocol Data Unit
PTP	Precision Time Protocol
QER	QoS Enforcement Rule
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
RGBD	Red Green Blue Depth
RoI	Region of Interest
RTCP	Real-time Transport Control Protocol
RTP	Real-time Transport Protocol
SDK	Software Development Kit
SDP	Session Description Protocol
SfM	Structure from Motion
SFU	Selective Forward Unit
SM	Session Manager
SMF	Session Management Function
SNS	Smart Networks and Services
SRTP	Secure RTP
SUCI	Subscription Concealed Identifier
TCP	Transmission Control Protocol
TSN	Time-Sensitive Networking
UC	Use Case
UDP	User Datagram Protocol
UE	User Equipment
UHD	Ultra High Definition
UM	User Manager
UP	User Plane
UPF	User Plane Function
URSP	UE Route Selection Policy
VM	Virtual Machine
VNF	Virtual Network Function
VR	Virtual Reality
WebRTC	Web Real-Time Communication
WP	Work Package
XR	eXtended Reality
XRM	XR and Media Services

1 INTRODUCTION

The main purpose of D3.1 is to summarize the outcomes of all the tasks of Work Package 3 (WP3) "XR Enablers" during M4-M18 of the "6G eXperimental Research infrastructure to enable next-generation XR services" (6G-XR) project.

The objective of WP3 is to develop enablers for XR multimedia processing, including AR and VR. Thus, each task focuses on the implementation of different enablers or multimedia functions, addressing several aspects of the End-to-End (E2E) media pipeline:

1. Task 3.1 (T3.1) focuses on volumetric capture sensors and reconstruction.
2. Task 3.2 (T3.2) leverages edge computing capabilities to enable scalability of multimedia processing.
3. Task 3.3 (T3.3) employs heterogeneous protocols to enable the delivery of multimedia content across distributed users having different devices and interaction capabilities.
4. Task 3.4 (T3.4) provides clock and media synchronization capabilities to enable a coherent participation to immersive media communications among users.
5. Task 3.5 (T3.5) develops solutions for media session orchestration.
6. Task 3.6 (T3.6) enables the collection of Key Performance Indicators (KPIs) related with multimedia processing such that they can be used for evaluation or real-time actuation.

This document describes the initial implementation of the XR Enablers or multimedia functions of the E2E media pipeline. These functions will be developed further during the remaining months of the project and, finally, integrated within the computing infrastructures deployed by WP2 "Networking and Computing Enablers".

The solutions developed within WP3 will also be used to demonstrate three UCs related to VR user plane (UC1 and UC2 presented in D1.1 [1]) and AR control plane (UC3 presented in D1.1 [1]). These UCs are included in WP6 "Validation of Holographic and 3D Digital Twin Use cases".

1.1 OBJECTIVES OF THE DELIVERABLE

The objectives of D3.1 are to:

- Describe the E2E diagram of communications to be developed and integrated, as it is necessary to demonstrate three WP6 UCs (UC1 and UC2 focus on the VR user plane, while UC3 focuses on the AR control plane).
- Present the progress on the development of the multimedia functions required for the E2E media pipeline. For each of them, an overview of the function is included and, when applicable, the related requirements, and the initial hardware and software employed for the development.
- Present the components in charge of providing synchronization capabilities across the participants of a media session, as well as components for orchestrating the different functions of the E2E media pipeline.

- Describe the infrastructure configuration enablers and monitoring system employed by the XR Enablers to configure network resources and to collect KPIs from different multimedia functions of the E2E media pipeline, respectively.

1.2 STRUCTURE OF THE DELIVERABLE

The structure of D3.1 is as follows:

- Chapter 2 introduces the E2E diagram of communications that are related to WP6 UCs.
- Chapter 3 focuses on multi-camera capture system developed for generating volumetric video streams. Video reconstruction solution supported by the Edge is also described, as it helps to improve the quality of the volumetric video through the reconstruction and fusion of the multiple inputs.
- Chapter 4 describes the components to be deployed on the cloud/edge infrastructure to scale and provide wider access to XR services. These components include several media processing such as forwarding, mixing, transcoding and rendering of XR content.
- Chapter 5 describes the streaming protocols and video players employed to deliver the media to the end user's device. The device can run a native player with volumetric capture, 6 Degrees of Freedom (6DoF) interaction (i.e., movement through 3D space), and consumption capabilities, or a simplified web-based player where the volumetric capture is not available.
- Chapter 6 presents the clock and media synchronization mechanisms to be integrated to provide a synchronized XR experience among participants and to support the monitoring of multimedia performance.
- Chapter 7 describes the Holo Orchestrator for the VR user plane and the IP Multimedia Subsystem (IMS) session manager for the AR control plane.
- Chapter 8 presents the infrastructure configuration mechanisms to allocate the infrastructure resources and/or select the appropriate resources to run the multimedia processing components.
- Chapter 9 reports on the monitoring system employed to assess the performance metric from the E2E media pipeline.
- Finally, Chapter 10 summarizes the report.

1.3 TARGET AUDIENDE OF THE DELIVERABLE

This deliverable is a public report which targets the project consortium, stakeholders, academic and research organizations, EU commission services, and the general public.

2 END-TO-END DIAGRAM OF COMMUNICATIONS

XR services are typically developed by strategically concatenating building blocks or Application Functions (AFs) along the E2E media pipeline (i.e., from capture to presentation). In Mobile Network Operator (MNO) networks, these XR AFs can become part of either the Control Plane (CP) – thus overseeing session management and control functions – or the User Plane (UP) – thus being in charge of media processing, communications or delivery functions.

6G-XR is a project that provides, in its WP3, a set of XR Enablers or AFs, belonging to both the UP and CP. These enablers are deployed over distributed cloud continuum/edge computing infrastructures (described in WP2), making use of Radio Access Network (RAN) and core network resources (described WP4). The VR user plane and AR control plane are described in the following subsections.

2.1 VR USER PLANE

Figure 1 provides a high-level overview of an E2E multimedia pipeline that can be deployed to provide real-time multiuser holographic communication services in shared VR scenarios. This multimedia pipeline concatenates several XR Enablers for media processing over the UP. The key XR Enablers are briefly introduced as follows:

- Volumetric Content Capture and Reconstruction: these are UP AFs deployed at the User Equipment (UE) and/or in-cloud. They aim at integrating volumetric capture sensors and reconstruction capabilities. XR systems typically employ real-time capture sensors, using either single or multiple sensors, and which often allow for fine-tuning the resolution and granularity of the captured data, even during the lifetime of a session.
- Selective Forwarding Unit (SFU): it is a UP AF deployed as in-cloud/edge component. It is in charge of forwarding media streams from origin clients to the appropriate destination clients.
- Multipoint Control Unit (MCU): it is a UP AF deployed as in-cloud/edge component. In addition to forwarding media streams from origin clients to the adequate destination clients, it also performs other advanced features like streams mixing and transcoding, to provide a single and personalized stream to every target client.
- Remote Renderer: it is a UP AF deployed as in-cloud/edge component. It acts as a surrogate player on the cloud/edge and provides a rendered 2D/360º video stream to lightweight players.
- Native (full-fledged) or web (lightweight) players: these are UP AFs deployed at the User Equipment (UE). These are software components used by clients involved in media content creation and/or consumption, and to perform the required interactions.
- Holo Orchestrator: it is an UP AF meant for orchestration and deployed as in-cloud component. It is in charge of session management and interfacing the Edge Orchestration components.
- Monitoring System: it is a UP AF meant for monitoring and deployed as in-cloud component in charge of hosting relevant Quality of Service (QoS), resource usage and activity metrics (i.e., telemetry) that can be used to evaluate the performance of the session, and to perform advanced adaptations.

The streaming and communication protocols and pipelines, as well as the control protocols for exchanging metadata about session management with the Holo Orchestrator, are part of the E2E VR platform.

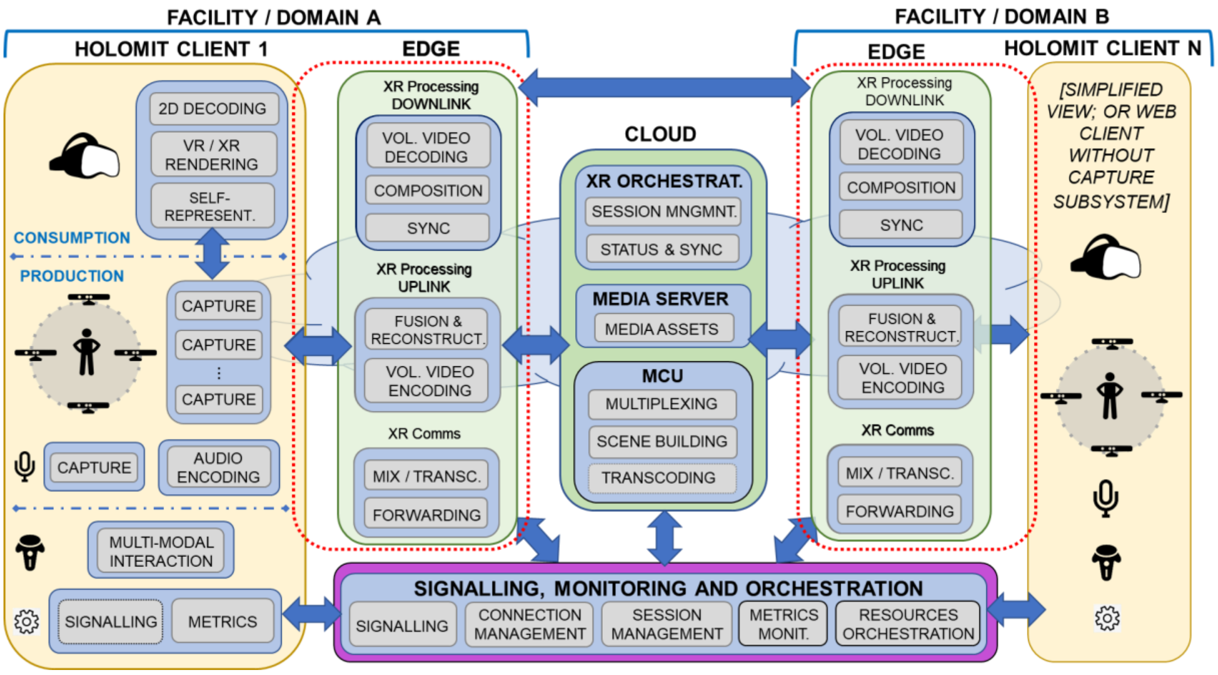


Figure 1. Components for VR User Plane.

2.2 AR CONTROL PLANE

Figure 2 provides a high-level overview of the holographic communication service based on the integration of an AR application with IMS, being part of network CP. It shows the IMS data channel architecture design with components and communication flows. Data channel components are distributed on IMS and public clouds. IMS cloud components ensure the proper signalling between the consumer User Equipment (UE) and the AR media servers. The signalling server, which resides in a public cloud, ensures the connectivity establishment between the data channel signalling function (DCSF), media server and the Agent UE using WebSocket messages.

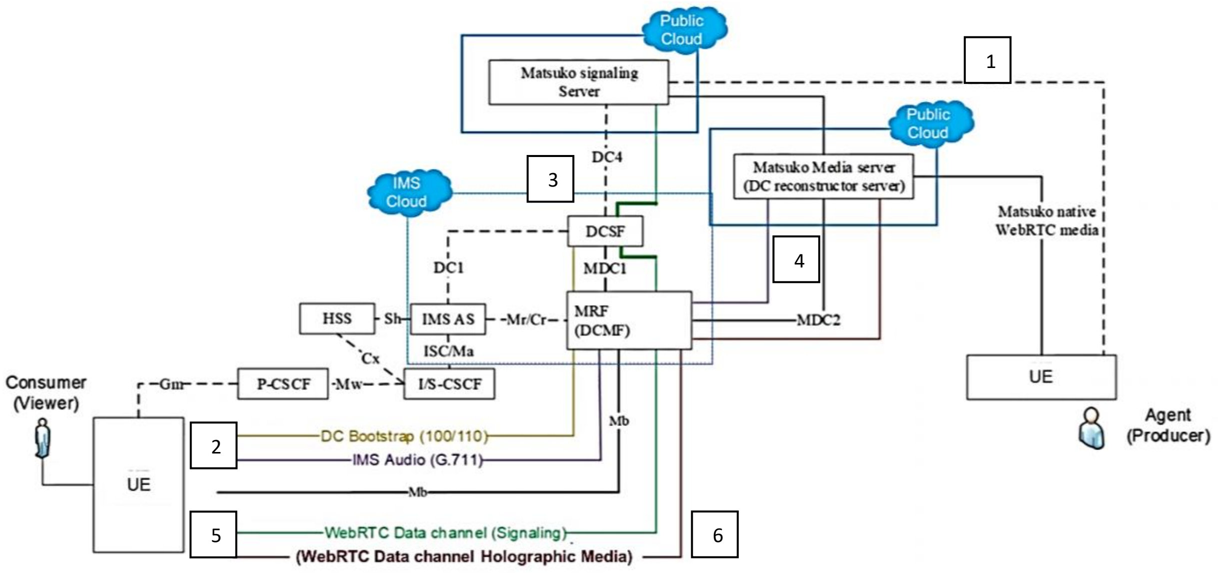


Figure 2. IMS data channel architecture design.

To exchange the holographic data through the UP, a series of preliminary steps need to be taken by the CP to carry out correct communication between the different network elements involved.

The information flow in Figure 2 is described in the following steps (1)-(6):

The first step is for the agent to register in the signalling server session manager to establish the start of the session (1) and wait for a consumer (viewer) user device to begin a classic call (IMS call) to the IMSDC (IMS Data Channel) service. Then, the device downloads the AR-enabled application from the IMS-AS that is represented in the viewer's phone dialler (2).

The signalling server receives then a holographic call registration request (3) and automatically obtains information from the different data channel reconstruction servers available to establish the connection with the one that allows the call to be made in the most optimal way in that specific moment (4).

Once connections are established, the signalling server sends a notification to the user agent of this connection to initiate the call and exchange Session Description Protocol (SDP) messages and the initial session properties (5). This way, the agent user is also able to access the established reconstruction server and change to the UP for the transmission of 2D/3D Web Real-Time Communication (WebRTC) video/data and WebRTC/IMS audio (6).

3 MULTI-SENSOR VOLUMETRIC RECONSTRUCTION

This section presents the first versions of the XR Enablers developed to capture volumetric video streams from a multi-camera setup. Furthermore, video reconstruction solutions are also presented as they are used to improve the quality of the captured streams. Operations such as reconstruction and fusion of several volumetric streams can be supported by edge processing capabilities, as presented in Figure 3.

Figure 3, the Volumetric Capturer (violet block) consists of four R32 light-field cameras (see Appendix A) and a 2x2x2 cubic meters capture area for holoportation. Four CXP-12 lanes transmit the four light-field streams to the edge (orange block) at a low latency of under 100ms, where they are converted into Red Green Blue Depth (RGBD) streams. These streams are then fused into a single volumetric stream, encoded, and sent for downstream processing (see Section 4).

The video capture and video reconstruction operations are described in the following subsections.

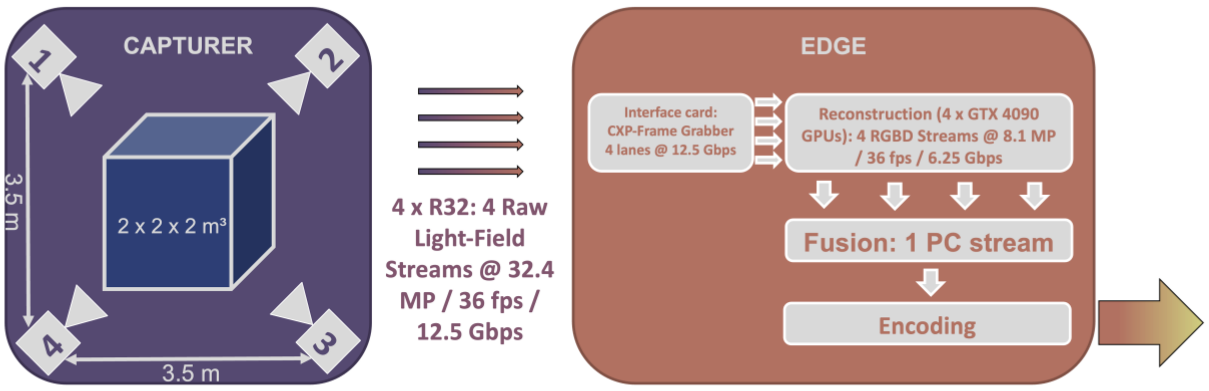

CAPTURER
1
2
3
4
2 x 2 x 2 m³
3.5 m
3.5 m
4 x R32: 4 Raw Light-Field Streams @ 32.4 MP / 36 fps / 12.5 Gbps
EDGE
Interface card: CXP-Frame Grabber 4 lanes @ 12.5 Gbps
Reconstruction (4 x GTX 4090 GPUs): 4 RGBD Streams @ 8.1 MP / 36 fps / 6.25 Gbps
Fusion: 1 PC stream
Encoding

Figure 3. Components of a Volumetric Capturer using Four Cameras and Edge Subsystems.

3.1 VIDEO CAPTURE

3.1.1 Overview of the component

The objective of the Volumetric Capturer is to create a sizable capture area, spanning at least 2 by 2 by 2 cubic meters, for high-fidelity holoportation. This setup is primarily targeted toward professional users who have the capability to dedicate sufficient space for holoportation activities and deploy specialized hardware accordingly. A typical use case scenario includes presentations or performances by lecturers or artists to an audience where interactive engagement is desired, and the fidelity of the experience is paramount.

Contemporary (active) multi-view stereo systems, such as the one presented by Google [2], can have up to 90 cameras and specialized illumination arrangements. This setup generates high-quality 3D captures but requires a significant amount of data, up to 520 Gbps. For now, such setups can pose significant challenges even in a professional context, particularly in terms of system size, setup time, expenses, and computational demands, and those are even more acute in scenarios requiring low latency, such as holoportation.

In contrast, the Volumetric Capturer incorporates four R32 light-field cameras developed as part of the 6G-XR project. Light-field cameras present a novel solution for holoportation by capturing both 2D and 3D data in a single image. This offers several practical advantages, including the ability to streamline setup processes with fewer cameras, thanks to the self-contained 3D information of each camera. Additionally, their compact nature facilitates easier transport, deployment, and reconfiguration of the setup for various capture areas. Moreover, light-field cameras exhibit increased tolerance to motion blur. Although high computational demands exist, especially in low-latency environments, today's hardware can manage these requirements.

The Volumetric Capturer uses CoaXPress (CXP)-12 to connect the R32 light-field cameras with the edge. CoaXPress is a high-speed serial communication standard designed for imaging applications over coaxial cables. It provides a robust, high-bandwidth, low-latency interface to connect cameras to processing units. Each R32 camera connects to the edge device through a single coaxial cable, allowing for cable lengths of up to 30 meters without affecting bandwidth. Alternatively, an extended cable length of up to 180 meters is achievable, although with reduced bandwidth. Although CoaXPress also supports fiber optic cables, removing all practical limitations to cable lengths, we decided to use copper CoaXPress cables as they deliver power to the cameras through a single cable, which also handles the data link, simplifying setup procedures and enhancing system flexibility.

6G-XR can fully leverage the CoaXPress high bandwidth and low latency characteristics to offload the intensive light-field processing tasks to the more powerful computational resources available at the edge, reducing overall system latency. This architecture ensures efficient data transfer and processing, optimizing the performance of the Volumetric Capturer for real-time applications such as holoportation.

The R32 cameras come with a custom lens mount that isolates the camera body from the lens thermally, making them more resilient to thermal fluctuations. It features a modified Basler OEM lens with a 24.5mm focal length, providing the R32 cameras with a 55.7° horizontal and 39.1° vertical Field of View (FoV).

The Volumetric Capture sensors have already been deployed at i2CAT facilities for their integration with the HoloMIT solution[3], by i2CAT.

The Volumetric Capturer will be interfaced with the Holo Orchestrator (Section 7) to perform Level of Detail (LoD) / rate adaptations based on indication from the in-cloud component (e.g., due to detected QoS drops, or congestion). This is planned for a later stage of the project.

3.1.2 Requirements

As mentioned in Section 3.1.1, the main goal of the Volumetric Capturer is to provide realistic and immersive holoportation experiences, a goal which requires specific features in the R32 cameras.

Each R32 camera is equipped with an Onsemi XGS image sensor, which has a resolution of 32.4 MP and can capture images at a maximum frame rate of 36 fps. However, during light-field processing, there is a natural loss of lateral resolution, which reduces the effective resolution to 8.1 MP. By merging multiple R32 outputs, reconstructions with resolutions greater than 4K (or surpassing Ultra High Definition (UHD) resolution in the first level of adjustments to the quality of experience) can be achieved. This excess resolution and bandwidth allow to prioritize various aspects and test their impact on immersion, such as high resolution at 30 fps vs medium resolution at 60 fps.

[3] https://i2cat.net/holoportation-technology/

The Onsemi XGS is an image sensor with a global shutter that simultaneously captures an entire frame. This eliminates the distortion and artifacts caused by conventional rolling shutters (scanning line by line), such as image skewing during the capture of fast-moving objects. This feature is particularly useful for allowing the natural movements of holoported users. Moreover, global shutter sensors are ideal for applications requiring precise synchronization and timing, making them perfect for capturing dynamic scenes with minimal motion blur with multiple cameras.

Light-field cameras gain depth-sensing capabilities by placing thousands of tiny lenses in front of the image sensor. For the R32, a **multi-focused plenoptic 2.0** Microlens Array (MLA) in a Galilean configuration with an aperture of f/1.8 was developed. The rationale for these design decisions is provided below:

- **Plenoptic 2.0**: it also known as focused plenoptic, represents the state-of-the-art approach to light-field imaging, offering minimal loss in lateral resolution. While it demands higher computational resources, it remains the preferred choice for achieving high-fidelity content.

- **Multi-focused**: it is a design aspect to extend the capabilities of plenoptic 2.0 by employing a MLA with multiple focal lengths, such as three in the case of 6G-XR, effectively expanding the Depth of Field of the cameras by a factor of 6. Although the Depth of Field expansion may not be advantageous in confined spaces, it can be traded for reduced motion blur, thanks to increased light sensitivity and faster shutter speeds, enabled by a wider aperture for the same Depth of Field compared to a 2D or conventional light-field camera.

Operating the Volumetric Capturer in confined spaces necessitates the use of wide-angle lenses. Through simulations and experiments, a Galilean setup, which focuses the lens of the camera behind the image sensor, has demonstrated superior compatibility and depth resolution when paired with wide-angle lenses compared to the more common Keplerian light-field configuration.

Ensuring a match between the working aperture of the camera lens and the MLA aperture is crucial in light-field imaging. The f/1.8 aperture chosen enables high light sensitivity, allowing for shutter speeds of under 5 ms that can still capture natural human movement with manageable motion blur. This wide aperture is made possible by the Depth of Field expansion provided by the multi-focus design. While a more open aperture is feasible, it would significantly limit compatibility with commercially available wide-angle lenses.

One challenge identified during initial experiments with the R32 cameras was the thermal output generated by the image sensor when operated at full bandwidth. The use of a conventional metal lens mount, paired with a metal lens, exacerbated this issue by acting as a heat sink for the image sensor, prolonging the time required for the camera to reach thermal equilibrium to as much as two hours. This extended startup time posed a significant problem, as the thermal expansion of the lens mounts caused shifts in the lens focus, rendering the calibration of the camera invalid. Although a hot and cold stage calibration method could potentially address this issue, neither would be applicable during the lengthy two-hour startup period, severely limiting the practical utility of the Volumetric Capturer.

To mitigate this issue, a software compensation method is proposed as described later, alongside the development of a thermally isolating lens mount. This solution significantly reduces the time required for the camera to reach thermal equilibrium to under 30 minutes. By implementing these measures, the adverse effects of thermal expansion on lens focus are mitigated, ensuring the validity of calibration, combined with the software compensation, even during the startup phase.

3.1.3 Volumetric sensors

To summarize, the volumetric sensors used for the Volumetric Capturer are four R32 light-field cameras, which are essential for achieving high-fidelity holoportation. These cameras are equipped with Onsemi XGS image sensors with 32.4 MP resolution, capable of capturing images at up to 36 frames per second. However, due to light-field processing, the effective resolution is reduced to 8.1 MP. The hardware setup uses CoaXPress for robust, high-speed data transmission between the cameras and edge processing units. The R32 cameras feature a unique lens mount design that thermally isolates the camera body from the lens. They are equipped with a modified Basler OEM lens that offers a 55.7° horizontal and 39.1° vertical FoV. Additionally, these cameras incorporate a multi-focused plenoptic 2.0 MLA in a Galilean configuration, optimizing depth of field and minimizing motion blur. This hardware configuration is designed to meet the demands of professional holoportation setups, providing the necessary resolution, frame rate, and depth sensing capabilities for immersive, high-quality 3D captures.

For further details, please refer to the R32 factsheet in Appendix A.

3.1.4 Initial software

The R32 light-field cameras use GeniCam to interface with the edge. GeniCam, short for Generic Interface for Cameras[4], is a standardized interface originally developed for machine vision cameras. It provides a unified way to control and access camera features across different platforms and interfaces. It simplifies integration, enabling easy configuration, image capture, and access to camera functionalities.

3.2 VIDEO RECONSTRUCTION

3.2.1 Overview of the component

This section outlines the hardware and software components deployed at the edge to interface with the Volumetric Capturer and extract 2D and 3D information from the raw light-fields. A CoaXPress frame grabber card is utilized on the hardware side, capable of capturing four streams and providing the required bandwidth between the Volumetric Capturer and the edge. Following the transmission of each of the four light-field streams to dedicated Graphics Processing Units (GPUs), the Image Processing Units (IPUs) convert these light-fields into RGBD streams. Integral to the accurate operation of each IPU are two key calibration components: intrinsic and extrinsic calibration.

- Intrinsic calibration pertains to the internal characteristics of the camera system, encompassing parameters such as focal length, lens distortion, and sensor alignment. By calibrating these intrinsic properties, the IPU can accurately interpret the captured light-fields, ensuring precise reconstruction of the scene geometry.

- Extrinsic calibration involves determining the spatial relationship between the individual cameras within the Volumetric Capturer setup. By aligning the coordinate systems of each camera, the IPU prepares the RGBD streams for fusion into one volumetric stream.

Both calibrations are currently provided through wizards in Raytrix's Light-Field Software Development Kit (LF-SDK). However, the implementation of a quicker, more user-friendly calibration based on the

[4] https://www.emva.org/standards-technology/genicam/

6G-XR’s Open Call 1 project ExCalibAR is ongoing. Level of Details (LoD) adjustments are implemented across this pipeline to account for limitations in bandwidth or availability of processing recourses.

The output from the Volumetric Reconstruction module is meant to be fed to the Volumetric Video Encoding components (described in T3.2) developed by i2CAT for an appropriate delivery over the network.

3.2.2 Requirements

The edge server is equipped with a CoaXPress frame grabber card capable of supporting four CXP-12 lanes, each providing a bandwidth of 12.5 Gbps. This setup ensures a high-bandwidth, low-latency interface with the Volumetric Capturer, facilitating the efficient transfer of light-field data. Utilizing Direct Memory Access (DMA), the frame grabber card enables low-latency transfer of light-field streams from the frame grabber to the GPU memory.

To meet the computational demands of processing light-field data in less than 100 ms, each R32 camera is assigned its own GPU running a single instance of an IPU. However, the system architecture is designed to accommodate an arbitrary number of IPUs running on an arbitrary number of GPUs. This flexibility allows an IPU to leverage computational resources from multiple GPUs or process multiple light-field streams simultaneously on a single GPU. This scalable architecture ensures efficient utilization of hardware resources.

The decision was made for the IPUs to convert the raw light-fields into RGBD streams rather than point clouds. RGBD streams offer more efficient resource utilization, particularly as they can be stored as textures directly on a GPU. Furthermore, RGBD streams provide deterministic memory requirements, a feature not necessarily guaranteed with point clouds, which can vary in the number of points. Additionally, using RGBD streams enables seamless interoperability with downstream processing by providing pointers rather than transferring the actual data, thereby significantly reducing latency and bandwidth demands. Moreover, established compression techniques can be readily applied to RGBD streams. Consequently, the conversion to point clouds is deferred until subsequent processing steps, where the benefits of the format outweigh its drawbacks, ensuring optimal utilization of resources throughout the processing pipeline.

The accurate rendering of RGBD streams by IPUs relies on precise intrinsic and extrinsic calibration. While the intrinsic calibration ensures the accurate reconstruction of the geometry of the scene, the extrinsic calibration determines the position of each camera within the setup and how they relate to each other. Raytrix's LF-SDK calibration wizard currently facilitates calibration by requiring users to capture a sequence of known and precisely manufactured calibration targets. However, recording adequate calibration data can present challenges, especially for users unfamiliar with camera calibration procedures for accurate measurement tasks. This gap is addressed by ExCalibAR, an Open Call 1 awarded project. ExCalibAR aims to deliver a quicker, more user-friendly, and less error-prone calibration method accessible to non-expert users without the need for precise calibration targets. The ExCalibAR project aims to extract intrinsic and extrinsic calibration parameters by utilizing Structure from Motion (SfM) while moving the camera to its final position, capturing a final image from each camera once they are all in their final position. It will enable a non-expert user to calibrate the system to an accuracy of 5 pixels within 10 minutes.

LoD adjustments are integrated into every processing pipeline stage. The R32 cameras offer flexibility by enabling the transmission of an arbitrary Region of Interest (RoI) or allowing pixel binning to macro pixels while ensuring that the IPUs maintain their calibration. Furthermore, IPUs can implement software-based RoIs to exclude already transmitted image areas from processing. The resolution at which the light-fields are rendered into RGBD streams can be tuned, allowing for adjustments to be made to both 2D and 3D data, even at different resolutions. Additionally, IPUs can be cloned to render the same input stream with varying parameters. Users have the option to adjust LoD settings through a user-friendly settings file or, for more granular adjustments, via a dedicated Application Programming Interface (API.

3.2.3 Initial hardware

The Euresys Coaxlink Quad CXP-12[5] serves as the frame grabber within the edge server, facilitating the transfer of light-field data from the Capturer. Complementing this, four Nvidia RTX 4090[6] GPUs, each equipped with 24GB of VRAM, are utilized to run the IPUs.

3.2.4 Initial software

An updated version of the Raytrix LF-SDK provides the IPUs and calibration wizards. The calibration wizards are set to be replaced by the results of ExCalibAR project. The LF-SDK currently runs on a Windows 10 or 11 host. ExCalibAR is currently under active development and has not yet reached the validation stage.

[5] https://www.euresys.com/en/Products/Frame-Grabbers/Coaxlink-series/Coaxlink-Quad-CXP-12-(1)nvidia

[6] https://www.nvidia.com/de-de/geforce/graphics-cards/40-series/

4 CLOUD/EDGE XR PROCESSING AND SCALABILITY

This section presents the current status (first version) of the XR Enablers, which are related to cloud/edge XR processing, and that can provide scalability and wider access to the XR services. The component modules or units are described in the following subsections.

4.1 SELECTIVE FORWARDING UNIT

4.1.1 Overview of the component

In traditional video conferencing services, Selective Forwarding Units (SFUs) are typically used for the exchange of multimedia information between the involved clients.

6G-XR has departed from a functional SFU (outcome of EU H2020 VR-Together project[7]), built on top of Node.js[8], that enables multi-user holographic communications [3]. In particular, the SFU acts as a UP AF that manages the exchange of volumetric video – and audio – streams from origin to destination clients via TCP WebSocket connections by using socket.io[9], as illustrated in Figure 4. A more detailed architecture of the SFU is provided in Figure 5.

Two main innovations are applied to that SFU component in 6G-XR. On the one hand, the SFU has been virtualized (both as a Docker and as a Helm Chart), thus becoming a VNF that can be dynamically instantiated over the cloud continuum (e.g., on a selected edge server), under request, via enablers from WP2 (Edge Orchestration, see D2.1 [4] for further details). On the other hand, novel interfaces with the Holo Orchestrator (Section 7.1) and a modular distribution architecture have been devised so that: (i) specific instances of SFUs can be selected for running sessions, based on specific criteria (e.g., deployment domains, edge resources); (ii) more than 1 SFU can be concurrently used to enhance the scalability of media sessions (Figure 6); and (iii) the SFU can also be interfaced to other in-cloud VNFs, like a Multipoint Control Unit (MCU) (Section 4.2) and a Remote Renderer (Section 4.3).

When adopting an SFU for a session with N clients, each client sends in uplink 1 media (audio + video) stream to the SFU and receives N-1 streams (the ones from all the rest clients) from the SFU. This also means that, in total, the SFU needs to send N*(N-1) streams in downlink, which can become a bottleneck. If more than 1 SFUs are adopted for a given session, then each client still sends its media streams to a unique SFU but may receive media streams from all the active SFUs. The SFU is just in charge of data forwarding, but does not perform any media processing tasks, like stream multiplexing and/or transcoding.

[7] https://vrtogether.eu/

[8] https://nodejs.org/

[9] https://socket.io/

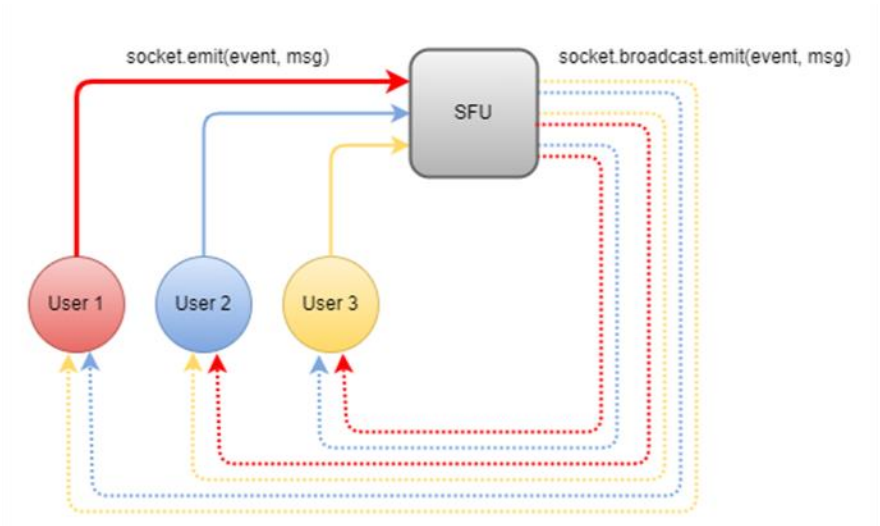


Figure 4. High-level communication architecture when adopting a Selective Forwarding Unit (SFU).

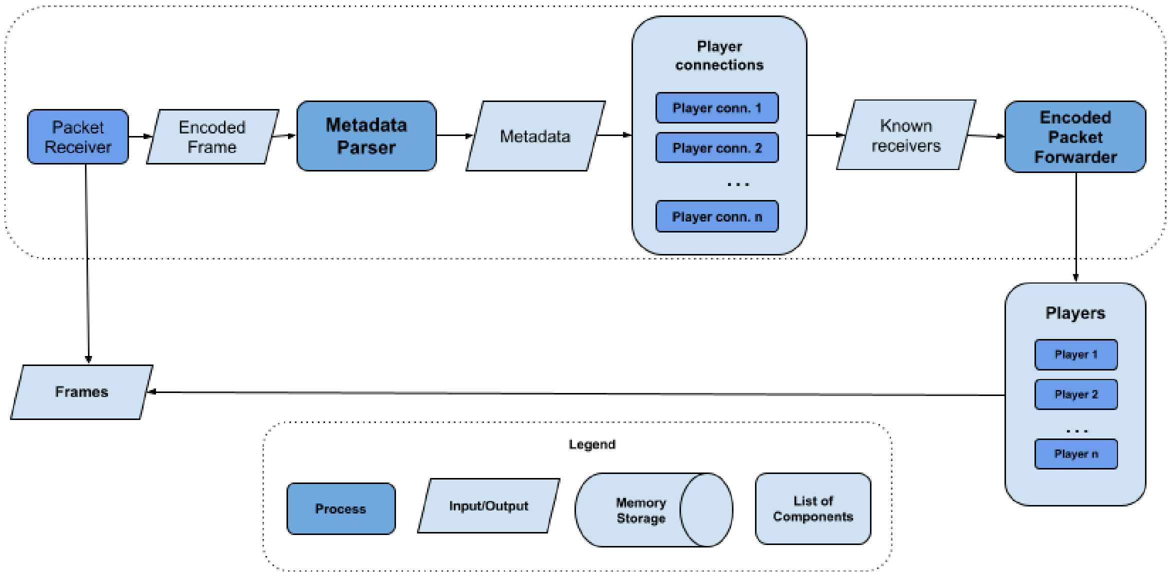


Figure 5. SFU Architecture.

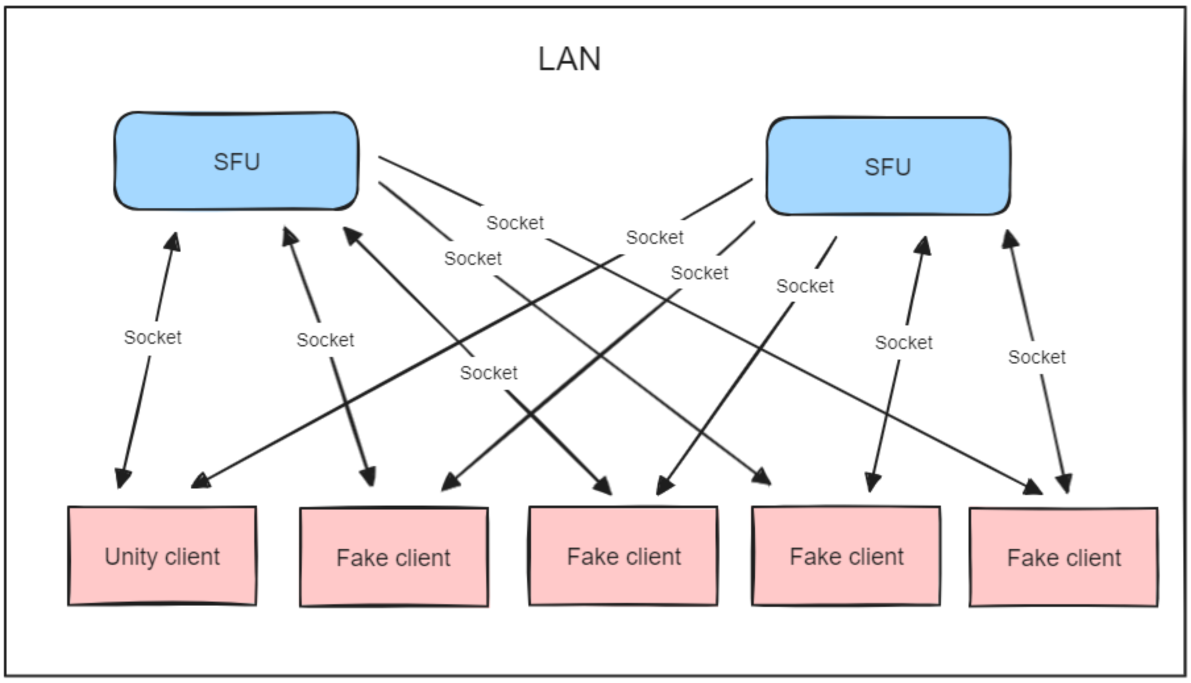


Figure 6. High-level scheme of a session with clients connecting to two different SFUs.

The SFU is composed of two independent modules/services:

1. **Media Manager**: It is responsible for forwarding the audio + video data between the involved clients.
2. **Events Manager**: It is responsible for forwarding relevant metadata between the involved clients, like their positions in the virtual space, or any other certain events that can be originated/triggered in the media session (e.g., interactions with the environment).

4.1.2 Requirements

The SFU is a lightweight in-cloud media function in the sense that it does not have to process data in real-time, but just to manage their forwarding to the appropriate clients.

In terms of KPIs, the SFU shall support at least 8 concurrent users per session, and multiple SFUs could be instantiated if such number needs to be increased (a dynamic decision maker to be implemented in the second phase of the project).

One important requirement though is that it needs to be deployed on a server and network with enough bandwidth capacity (>=1 Gbps), as it becomes a traffic hub, and its output traffic becomes multiplicative as the number of users increases.

Although technically an SFU could provide support for multiple concurrent sessions, its virtualization and modularization ease a new instantiation of the SFU per every new session being created.

4.1.3 Initial hardware

The SFU has been tested and run in a variety of Personal Computers (PCs) and Servers (both running Windows or Ubuntu), with no specific hardware requirements. It has been also successfully deployed on a server on the Microsoft Azure cloud computing platform with Standard DS1 v2 specs (1 vCPU, 3.5 GiB memory RAM).

4.1.4 Initial software

The SFU requires the installation of Node.js and socket.io, and it has been successfully installed and run in Windows 10 and Ubuntu 22.04 LTS machines (including Virtual Machines on Azure). It has been also virtualized as a docker container and helm chart, which eases its deployment on any machines, including those provided by hyper-scalers.

4.2 MULTIPOINT CONTROL UNIT

4.2.1 Overview of the component

In traditional video conferencing services, MCUs are typically used to lower the computation and bandwidth requirements at the client side, by performing stream multiplexing, transcoding, and composition functions on the cloud [5].

6G-XR has departed from a pioneer and fully functional Point Cloud MCU (outcome of EU H2020 VR-Together project[10]) to enable more lightweight and scalable multi-user holographic communication services [5]. In particular, the MCU acts as a UP AF that receives all volumetric video (i.e., Point Cloud) streams from a given session, multiplexes and fuses them, and then performs smart transcoding features with the goal of providing a single personalized output Point Cloud stream to each involved user in the session. That allows reducing the bandwidth and processing requirements on the client side, and thus contributes to higher scalability and interoperability [5]. A high-level architecture of the MCU is provided Figure 7, which includes the main components and modules of the MCU and the interactions among them. A legend below the architecture diagram has been added to facilitate the meaning of each block.

[10] https://vrtogether.eu/

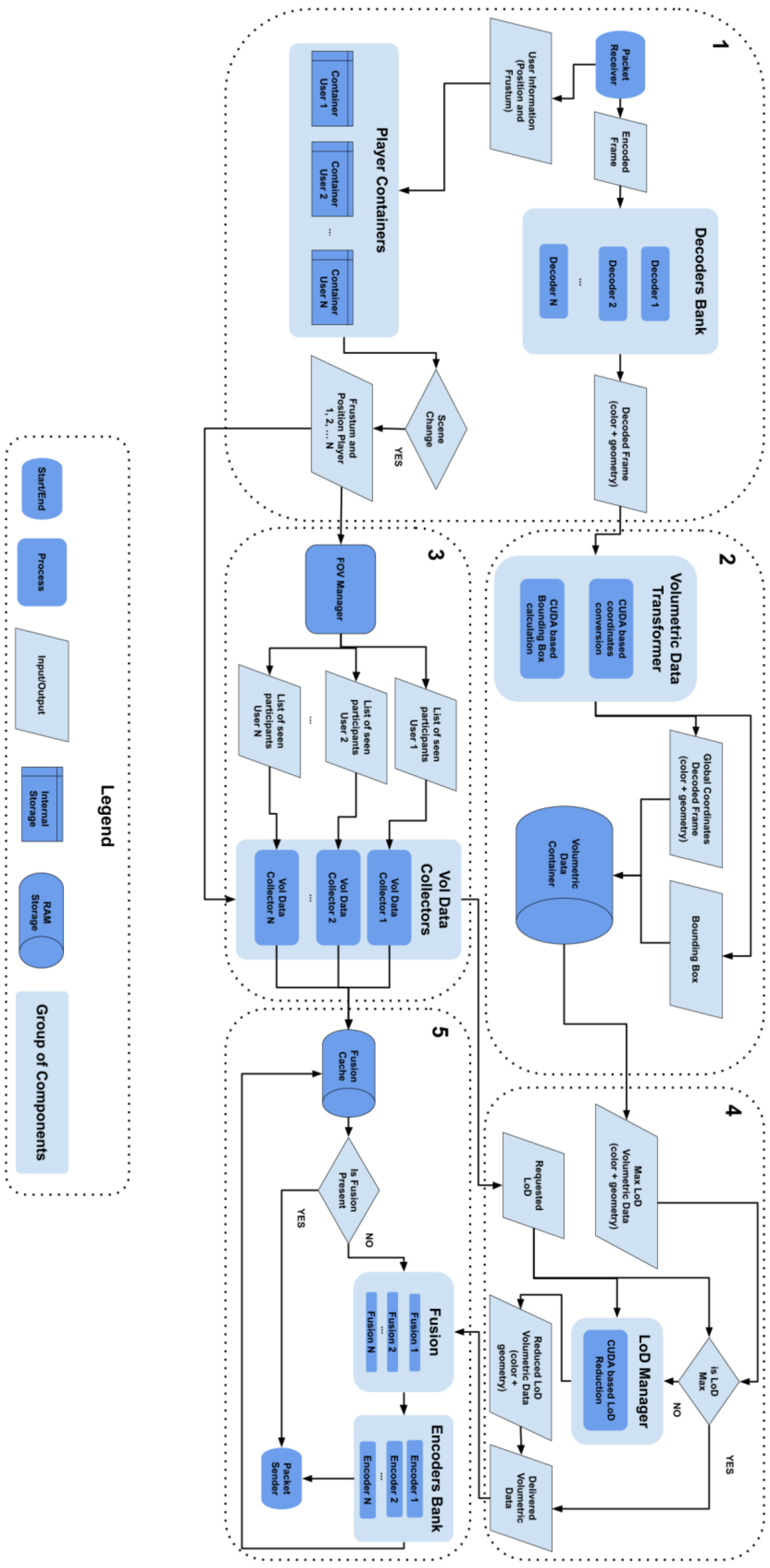


Figure 7. MCU architecture.

The architecture of the MCU is structured into five main blocks, briefly explained as follows (further details can be found in [5]).

1. **Block 1 - Reception and Decoding**

The first block of the diagram is dedicated to the reception and decoding components. The very first component, called ***Packet Receiver***, is in charge of establishing communications with the Holo Orchestrator. Once the MCU is added to the session, a Packet Receiver component, which is in charge of registering the actual participants of the holoportation session, keeps listening until one, or more, participants join. For each client (or participant) the MCU will instantiate a logic entity within a component called ***Player Containers***. Each logic entity (indicated as ***Container User 1, 2, …, N***), is used to store, for each user, the information needed by the MCU to perform its optimizations, such as user position and frustum (visible area coordinates). The *Player Containers* will provide an output to ***Block 3*** (explained afterwards in this section), every time a change of scene is detected (change of position or visible area). The last component of Block 1 is the ***Decoders Bank***, which is in charge of decoding the incoming volumetric video frames and it is designed following a producer/consumer multithreading scheme, allocating one CPU thread to every decoder. Every time a new input frame is received, the first available thread will be allocated to perform the corresponding decoding process. The decoded volumetric frames are then available as colour component and geometry information.

2. **Block 2 - Volumetric Data Storage and Transformation**

The MCU receives volumetric data from all the holoportation participants, each of them with different local coordinate references. After the decoding of the streams, a transformation is needed. The ***Volumetric Data Transformer***, in charge of performing the transformations of the volumes in world coordinates, includes a Compute Unified Device Architecture[11] (CUDA) based implementation that performs the coordinates conversion to the global coordinates system and, at the same time, evaluates the corresponding bounding box. The output will be stored in a RAM-based component, called ***Volumetric Data Container***, which is in charge of holding the transformed geometry data. Then, the remaining MCU components will be able to request the information stored here for the optimization of the streams.

3. **Block 3 - Field of View (FoV) Manager & Volumetric Data Collector**

To optimize the volumetric video streams delivered to each participant, the MCU needs to be aware of which participants are seen by each of the other participants, and in which positions they are. For this reason, the ***FoV Manager*** is in charge of processing frustum and position of the participant users, previously stored in the ***Player Containers (Block 1)*** and creating a list of participants that each client is seeing. The list is updated every time a scene change is detected in *Block 1* and the final output provided to the rest of the pipeline (***List of seen Participants 1, 2, …, N***). For each participant, the system creates a component called ***Volumetric Data Collector*** that receives the recently created *Lists of Participants*. The *Volumetric Data Collector* knows: (i) the participants seen by each client needed to avoid streaming redundant data, and (ii) the relative positions of the users, coming from ***Block 1***.

[11] https://developer.nvidia.com/cuda-toolkit

The participants not seen are then removed from the data to be provided, while the resolution optimization, based on the relative distances, is performed in ***Block 4***.

4. Block 4 - Level of Detail Optimization

Once the MCU is aware of which participants are seen by each participant, the resulting volumetric video has to be transmitted with a resolution that depends on the relative distance and positions between them. In 3D video, the usual nomenclature for the resolution is **LoD**. ***Block 4*** is in charge of requesting the volumetric video, stored in the *Volumetric Data Container* (***Block 2***) with the highest LoD available, and subsampling the number of voxels depending on the distances between users. The goal is to reduce the amount of data to be delivered if the distance does not allow the user to appreciate the maximum resolution available. To know if the LoD needs to be reduced, ***Block 4*** receives the LoD requested by the *Volumetric Data Collectors* (***Block 3***). If the LoD requested is the maximum allowed, the MCU will avoid performing redundant operations and will deliver the Volumetric Video as it was previously received. On the other hand, if the LoD requested is lower than the maximum, the process called ***LoD manager*** will be activated and will perform a CUDA based operation for the reduction of the LoD. The output will be the downsampled Volumetric Data.

5. Block 5 - Fusion, Encoding and Transmission

Block 5 is in charge of performing the Fusion previously described and of creating the scene to be delivered to each participant, thanks to the inputs created by the previous blocks. The first component involved is the ***Fusion Cache*****.** The *Fusion Cache* receives the data related to the fused scene composition from the *Volumetric Data Collectors* (***Block 3***) and, when a set of volumetric videos has to be delivered, a process is activated to check if such scene has been previously created and stored. If the requested scene is not available in the *Fusion Cache*, the system performs the following steps:

1. Request the needed Volumetric Data, after the LoD optimization, from ***Block 4.***
2. Perform a fusion of the Volumetric Data, from different participants.
3. Provide the fused Volumetric Data to the *Bank of Encoders*, which will assign an encoding thread for each user.
4. Transmit the corresponding encoded data to the clients.

In addition, the MCU includes a smart system to avoid performing redundant operations. After the steps described above, the compressed fusion is indeed stored in the Fusion Cache for future use. Within Block 5 the system is indeed capable of analysing if a newly requested fusion was previously performed for another user. This may often happen given that several users, placed close to each other, may be observing the same area. In this case, the set of steps related to fusion and encoding are skipped and the previously created fusion is directly delivered.

The next key innovations are being applied to that MCU component in 6G-XR. First, it has migrated to Linux to allow for its virtualization (VNF) and dynamic orchestration (WP2). Second, the mixing/fusion and transcoding features of the MCU have been decoupled, so they can be used independently or jointly for each instance of the MCU. Third, new interfaces between the MCU and Orchestrator and SFU components have been developed to enable the coexistence of multiple parallel MCUs per session, and even of the MCU with the newly developed SFU. These key innovations, and other planned ones, will be described with further details in the next deliverable D3.2 – “Final versions of XR enablers”.

4.2.2 Requirements

The MCU is a heavy in-cloud media function in the sense that it must handle multiple resource intensive volumetric video streams and process them in real-time. It is typically deployed on cloud computing platforms in/for traditional videoconferencing services, but the implications / benefits of deploying it on Edge servers are assessed in the framework of 6G-XR for holographic communication services

A series of associated requirements can be highlighted:

- It shall be deployed on a server and network with enough bandwidth capacity (>=1 Gbps), as it becomes a traffic hub.
- It shall exploit parallelization, GPU (e.g., NVIDIA RTX 3060 onwards) and CUDA processing.
- It shall be dynamically orchestrated and instantiated, so development in Linux-based OSs becomes necessary.
- It shall interface the evolved version of the Holo Orchestrator and SFU modules.
- Multiple MCUs shall be deployed for single sessions.

Likewise, a series of associated KPIs can be highlighted:

- It shall support at least 12 concurrent users per session.
- It shall be able to keep E2E delays within acceptable limits.

4.2.3 Initial hardware

The MCU has been successfully deployed in a server machine with Intel Xeon Gold 6140 @2.30GHz, with 2 NVIDIA GPUs (Tesla T4 16GB) and 2 Ethernet adaptors X550 TX 10 Gig LOM. Its deployment on other – more lightweight – machines will be explored later in the project.

4.2.4 Initial software

The MCU has been developed in C++. The initial version was developed and run on Windows and later migrated to Linux in 6G-XR project, mainly due to the better support for software virtualization / containerization.

4.3 REMOTE RENDERER

4.3.1 Overview of the component

The operation of delegating computationally expensive processing to a remote server is called "computation offloading". In the case of VR experience, the rendering operation is one of the most expensive tasks, requiring hardware acceleration (e.g., using GPUs) to process volumetric video stream. Moreover, it might also affect the battery life when using mobile devices. Offloading the computation to a remote server means performing the rendering task on a remote server equipped with hardware acceleration and sending back to the mobile device a plain video stream generated by rendering the volumetric video stream. As a result, the processing on the user's device is lighter, and can run without requiring hardware acceleration.

The use of remote rendering is perfectly aligned with the vision of the future 6G network, where cloud/edge services are expected to play an important role in enabling the processing of VR applications, and where the wireless access network should provide the improvements in data rates and latency, so it is not perceived that the rendering task is performed at the remote location instead of locally. Moreover, edge locations are meant to be closer to the user compared to cloud ones, which results in further improvements in terms of latency when the rendering tasks are performed at the edge instead of the cloud.

This section describes the Remote Renderer developed in 6G-XR to enable access to VR experiences to devices that cannot support local rendering due to their low computational capabilities. The Remote Renderer consists in an UP AF that combines VR content rendering technologies and network virtualization to take advantage of the characteristics of the edge infrastructure to generate personalized media session for the participants of a VR experience. Thus, it is designed to be deployed as a VNF within the edge infrastructure of the 6G-XR project.

The Remote Renderer is developed to render volumetric video content generated with a capture system into a VR scene. Then, it generates a plain 2D or 360º video stream by selecting a viewpoint inside the VR scene and delivers it to the user's device. The output video stream can be personalized by means of the 6DoF information collected at the user's device and shared with the Remote Renderer. The information is employed to adapt the viewpoint within the VR scene from which the plain 2D or 360º video stream is generated. This allows user devices such as laptops or VR headsets to be freed from the processing load, relying on network communications to receive the already processed content.

Figure 8 shows the logical modules of the Remote Renderer and the relation with the video players described in this report (Section 5.2 and Section 5.3). The Surrogate Player module acts similarly to a native video player, enabling the receiving and decoding of inputs from Holo Orchestrator, SFU and/or MCU, but avoiding displaying them. These inputs could be configuration messages and events provided by the Holo Orchestrator of the VR experience, or data sources from SFU/MCU, such as Point Cloud, 2D/360º video and audio streams. While the configuration is immediately applied to any of the modules of the Remote Renderer, the data sources are rendered in the Rendering Engine module. The result of the rendering operation is a 2D/360º video stream with its audio taken by inserting a virtual camera and a virtual microphone within the VR scene. The stream is later used by the Streaming Server module, which encodes and sends it to the video player through the most appropriate streaming protocol. The Streaming Server also acts as a receiving endpoint of the 6DoF information generated in the video player and sent to the Remote Renderer. The 6DoF information is employed by the Rendering Engine module to adjust the virtual camera within the VR scene such that the resulting 2D/360º video is personalized considering the position and the viewpoint of the participant.

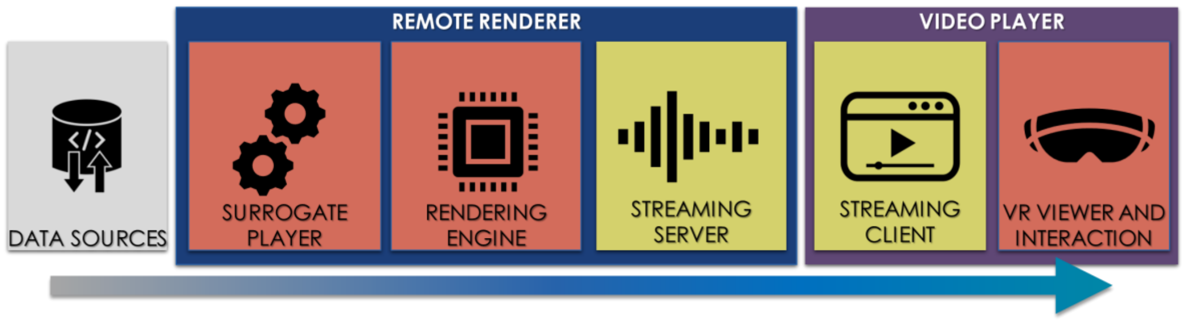


Figure 8. Logical modules of the Remote Renderer and Video Player.

The Streaming Server modules can generate both WebRTC and Dynamic Adaptive Streaming over HTTP (DASH) streams, even though the interaction of the user with the VR scene is available when using WebRTC only. WebRTC is the most appropriate protocol for interactive VR experiences as it has a data channel for sharing user's 6DoF information and, most importantly, for its real-time latency. Due to its intrinsic latency, DASH is not suitable for interaction, but it is the best alternative when considering the scalability in terms of connected users to the VR experience. Consequently, the video player could be either a WebRTC or DASH player depending on the generated stream at the Remote Renderer. Both WebRTC and DASH players are presented in Section 5, when describing the adaptive low-latency XR delivery solutions of 6G-XR.

Figure 9 shows the communication diagram of the Remote Renderer and its interaction with other components, such as the Monitoring System (Section 9.1), the Holo Orchestrator (Section 7.1) and the video player (Section 5.2 for WebRTC and Section 5.3 for DASH). To begin, the video player requests to join a VR session to the Holo Orchestrator, informing that it needs a remote endpoint to perform the content rendering. The Holo Orchestrator receives the request, assigns a Remote Renderer, and provides information to both the server and the player to configure the VR session. Then, the video player connects to the Remote Renderer to complete the session joining process.

After the session is joined, three loops run in parallel: rendering, interaction, and configuration. In the rendering loop, the media processing and transmission take place. Volumetric video and audio are provided by external sources, such as SFU or MCU, to the Remote Renderer. Internally, the server decodes it (Surrogate Player), renders and applies the personalized viewpoint (Rendering Engine), and generates the encoded 2D or 360º video and audio stream (Streaming Server) that is delivered to the video player. Finally, the player decodes and visualizes it on the device screen. This loop is valid for both WebRTC and DASH communication between the server and the player.

In the interaction loop, the video player collects the 6DoF information from connected sensors or input devices. This information, describing user's movement (translation and/or rotation) within the rendered VR scene, is sent to the server which uses it to update the viewpoint to provide the personalized video stream. This loop is only valid for WebRTC streaming, as DASH does not provide sufficient low latency to allow a smooth interaction with the VR scene. It means that the personalized stream is provided only with WebRTC by employing its data channel to send the 6DoF information. The DASH stream is instead common to all the users, as they cannot interact with the VR scene.

Finally, in the configuration loop, the video player captures metrics characterizing the streaming session and sends them to the Monitoring System. The Holo Orchestrator retrieves them (e.g., based on configured alarms if specific thresholds are surpassed) to take decisions on the encoding profiles that the Remote Renderer should use. The new encoding configuration is applied dynamically, although the process works differently for WebRTC and DASH. In WebRTC, the unique generated stream is adapted to match the provided configuration. In DASH, different representations are generated all the time, but the Media Presentation Description (MPD), consisting of a manifest that describes the streaming content, is periodically updated to show only the ones selected by the Holo Orchestrator at any moment.

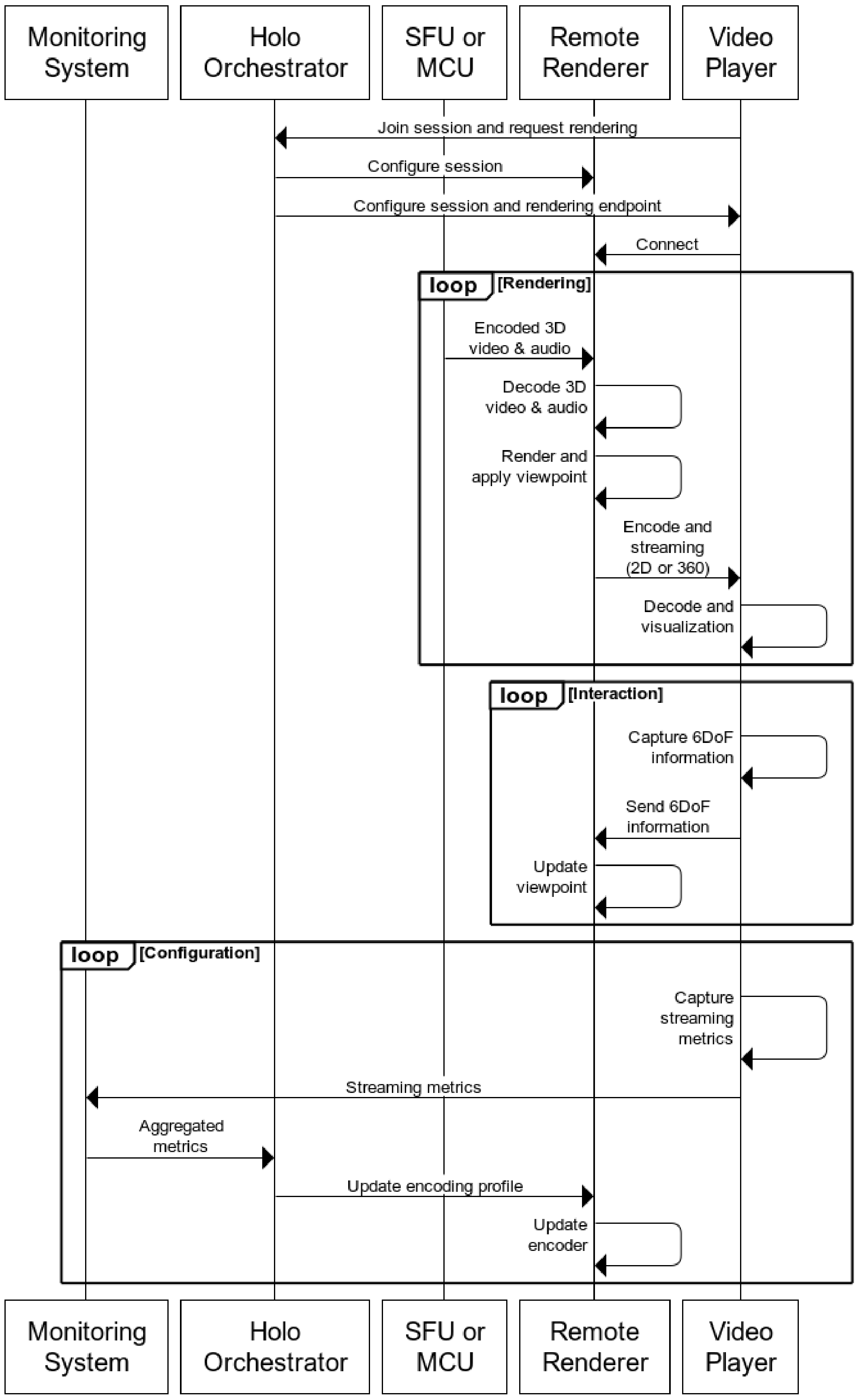


Figure 9. Communications diagram of the Remote Renderer.

4.3.2 Requirements

To define the requirements, it is necessary to consider different factors that influence both the computational load of the Remote Renderer and the QoS and the Quality of experience (QoE) when using the Remote Renderer to access VR experience from the user's device:

- Management of heterogeneous sources: the server must be able to manage different input sources, including virtual scenes, volumetric video streams and audio.
- Interaction capture: the player is not limited to display the already rendered content, but also shares user's interaction information with the server. To do this, the player must be able to access sensors or input devices that generate this information. The server receives the interaction information and adapts the rendered video stream considering user's viewpoint.
- User profiles: two different user profiles are envisioned to participate in a VR experience, namely interactive user and passive user. The first one has sensors to capture interaction information such that its video stream is personalized. In this case, the WebRTC protocol is employed to prioritize latency. The second one does not interact with the VR scene and its viewpoint is common to all the other passive users. In this case, a DASH stream is employed to prioritize scalability in terms of number of connected users.
- Processing capacity: a higher number of users requires higher processing capacity. The use of GPU can cope with this by enabling hardware acceleration for both media rendering and media encoding processes.
- Heterogeneous user equipment: the capabilities of the devices used to access the VR experience can vary between users. Both users connected with a laptop and users with VR headsets must be supported to enable a wider access to the VR experience.

Considering the aforementioned factors, the requirements to be met by the Remote Renderer are identified and presented in Table 1.

Table 1. Requirements for the edge rendering component.

Category	Requirement	Details
Inputs (data sources and configuration)	Prerecorded point cloud	Locally stored Point Cloud files or using socket.io communication
	Live point cloud	Point Cloud using socket.io communication
	VR scene	Local VR scene or using socket.io communication
	Audio	Local prerecorded audio or live via socket.io communication
Rendered data	Output generation	Virtual camera in VR scene to record video Virtual microphone in VR scene to record audio
	Format for interactive user	Video: RAW 2D or 360º Audio: RAW Mono
	Format for passive user	Video: RAW 2D or 360º Audio: RAW Mono
Interaction	Interactive user	6DoF information generated in the video player and transmitted to the server to generate the personalized stream.

		The Renderer can serve one (single player mode) or multiple video players (multi-player mode).
	Passive user	The user has no interaction.
	Transmission of interaction	- Interactive user: WebRTC data channel - Passive user: not necessary
	Application of viewpoint	Virtual camera movement in the scene
Output codecs	Interactive user	Video: VP8 or H.264 Audio: OPUS Mux: not employed
	Passive user	Video: H.264 or HEVC/H.265 Audio: AAC Mux: Fragmented MP4
Output streaming protocols	Interactive user	WebRTC with bidirectional communications: - Video and audio over SRTP/UDP - 6DoF over data channel
	Passive user	DASH with different video representations
	Streaming adaptation	- WebRTC: dynamic adaptation of video stream (resolution, frame rate, bitrate) - DASH: update of MPD manifest to show different representations
Processing	OS	Ubuntu 22.04 LTS
	Hardware	- Intel i7 or Xeon CPU with 12 cores* - RAM 32GB DDR4* - NVIDIA with NVENC encoding support[12] *Recommended for 4 users, the actual requirements may vary depending on the number of connected users
	Software	Unity and GStreamer
	Containerization	Docker and Helm
Communication with Holo Orchestrator and SFU/MCU	Protocol	Socket.io
	Messages and Events	- Session management - Input source (audio and video) - Output configuration (codecs and protocols)

The use of Unity[13] as the framework for the development of the remote rendering system is a fundamental requirement to guarantee better compatibility and integration with the Holo Orchestrator and native players, also developed with Unity and based on i2CAT HoloMIT SDK[14].

[12] https://docs.unity3d.com/Packages/com.unity.renderstreaming@3.1/manual/faq.html#issues-with-the-environment

[13] https://unity.com/

[14] https://i2cat.net/holoportation-technology/

GStreamer[15] is one of the most widely employed multimedia framework and is used by VICOM to add further multimedia encoding and streaming capabilities to the Unity framework.

All the requirements described in this section have been used to design and develop the initial version of the Remote Renderer. The hardware and software employed for this initial version are described in the following subsections.

4.3.3 Initial hardware

Regarding the hardware, a virtual machine with the following capabilities has been used to deploy and test the initial version of the Remote Renderer. The GPU is employed in passthrough mode and not shared with any other virtual machine.

Table 2. Hardware employed to deploy the edge rendering component.

Hardware	Specifications
CPU	Intel Xeon 12 Cores (2095.076 MHz)
RAM	32GB
Storage	125GB
GPU	NVIDIA Quadro RTX 4000

The above configuration allows the connection of up to 4 users with personalized audio and video streams (2 video streams having resolution 972x1080 each one of them) simultaneously. It is important to note that this initial release does not include all the required capabilities, and more processing is envisioned in the final release that could make the number of connected users to vary. For example, live point cloud stream ingestion is not yet supported. Processing such streams would be demanding, potentially reducing the number of simultaneous users.

4.3.4 Initial software

The current implementation of the Remote Renderer includes only two of the three logical modules: Rendering Engine and Streaming Server. It means that the current software can only render media content available locally and generate the video streams as output to be delivered to the connected users. The Surrogate Player will be added later to allow access to live media content from other components, such as SFU (Section 4.1) and/or MCU (Section 4.2).

The features already available in the current release are:

- **Inputs**: only pre-recorded and locally stored audio and video files (2D video or volumetric video/Point Cloud).
- **Rendering**: audio and video already integrated into a predefined VR scene (still not possible to configure). Virtual camera and microphone also integrated into the VR scene to generate the outputs in 2D or 360º format.
- **Interaction**: available through WebRTC data channel. Both information coming from laptop input devices (mouse and keyboard) and VR Headset (sensors and joysticks) are available.

[15] https://gstreamer.freedesktop.org/

- **Output codecs and protocols**: all the required codecs and protocols are available, except HEV/H.265 video codec for DASH that will be added later. Since the codecs that will be employed are hardcoded, it is necessary to enable their selection through APIs.
- **Streaming adaptation**: not available yet.
- **Containerization**: both Dockerfile and Helm Chart are generated, although not deployed yet in the computing infrastructure of the 6G-XR project.
- **Communications with Holo Orchestrator and SFU/MCU**: planned for a future release.

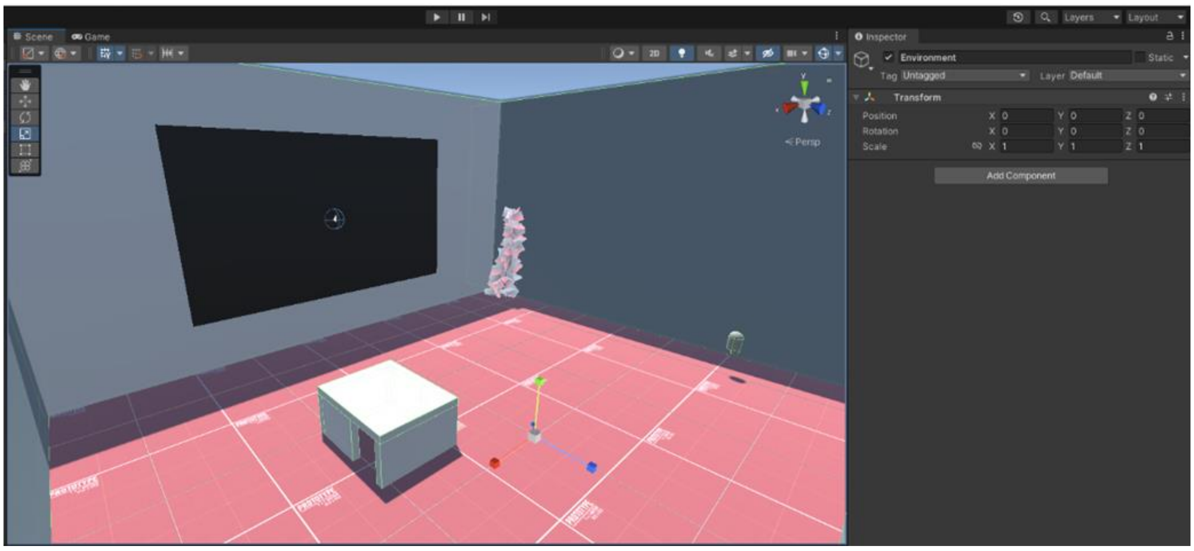
The developed solution works in Ubuntu 22.04 OS LTS environment and includes Unity and GStreamer frameworks, where this last one is integrated as a Unity plugin. The programming languages of the solution are mostly C++ and C#.

Table 3 lists the dependencies of the Remote Renderer, considering only the Rendering Engine and Streaming Server modules available in this first release. Further dependencies might be included when adding the missing Surrogate Player module.

Table 3. Software dependencies of the Remote Renderer and their versions.

Software and frameworks	Version
Unity	2021.3.31f1
Unity Render Streaming	3.1.0-exp.7
Vulkan	1.3.204
GStreamer	1.22
Nvidia driver	535.161.07
Nvidia Cuda Toolkit	12.2

Figure 10 shows the Remote Renderer running in the Unity Integrated Development Environment (IDE). A preconfigured virtual environment is presented where 3D objects can be added. A virtual camera and a virtual microphone are also elements that can be added in the VR scene to generate the sources for the output streams in WebRTC and DASH.



Environment
Tag Untagged Layer Default
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Figure 10. Remote Renderer running in Unity IDE.

The Remote Renderer can work without Graphical User Interface (GUI), i.e., by enabling the headless mode of the Unity-based application and containerized. The containerized version runs with the software environment presented in Table 4.

Table 4. Environment for the deployment of the containerization Remote Renderer.

Software	Version
Ubuntu OS	22.04
Docker	25.0.3
Docker Compose	2.23
Kubernetes	1.29
Nvidia Container Toolkit	1.15.0
NVIDIA GPU Operator	23.9.0

In the following release, i.e., the final version of the Remote Renderer, the work will focus mostly on developing and integrating the missing Surrogate Player module. This will enable communication with the other components, such as the Holo Orchestrator, the SFU and/or MCU, that should provide the input data sources to be rendered. Moreover, to enable streaming adaptation strategies, metrics exporters will be created to collect data on WebRTC and DASH performance. Other improvements will include adding missing codecs and provide the possibility to be remotely configured by the Holo Orchestrator.

5 ADAPTIVE LOW-LATENCY XR DELIVERY

This section presents the first versions of the XR Enablers developed to provide access to VR experience from heterogeneous devices, leveraging adaptive and low-latency multimedia delivery protocols. The different components are described in the following subsections, together with their respective requirements, as well as the corresponding initial software and hardware employed for the development.

5.1 NATIVE PLAYER

5.1.1 Overview of the component

The endpoint for the clients of XR services consists of a native Unity-based player (Windows build), which implements: (i) the necessary session management features interfacing the Orchestrator; (ii) the processing and exchange of audio and volumetric video streams for the real-time communications; and (iii) a set of interaction and presentation features. The implementation details for the above-listed components and modules can be found in [3].

6G-XR has departed from a fully functional version of this Unity-based player (outcome of EU H2020 VR-Together project[16]), whose features are provided as Unity package and associated SDK. The SDK is composed of the following key components:

- **HoloCore**: it integrates core functionalities, such as multi-threading, thread safe queues, and the interfaces between core components the package.
- **Voice**: It is used for audio communications, in charge of grabbing and encoding/decoding the audio of the users using state of the art audio codecs.
- **PointCloud**: This component includes the E2E pipeline to be able to provide holograms as Point Clouds, including capturing (using Intel RealSense or Azure Kinect sensors, and currently also integrating Raytrix sensors), encoding/decoding (using anchor codecs), and rendering (with custom shaders that are able to read the information and render it in a 3D world).
- **SocketIO**: This component implements the supported communications protocols to enable real-time multi-party data delivery, making use of a proper multithreading provided by HoloCore, and of WebSocket and Socket.IO libraries.

The native players, integrating the SDK, communicate and exchange data between themselves via SFUs or via MCUs, as detailed in the respective sections of this report, and as sketched in Figure 11.

[16] https://vrtogether.eu/

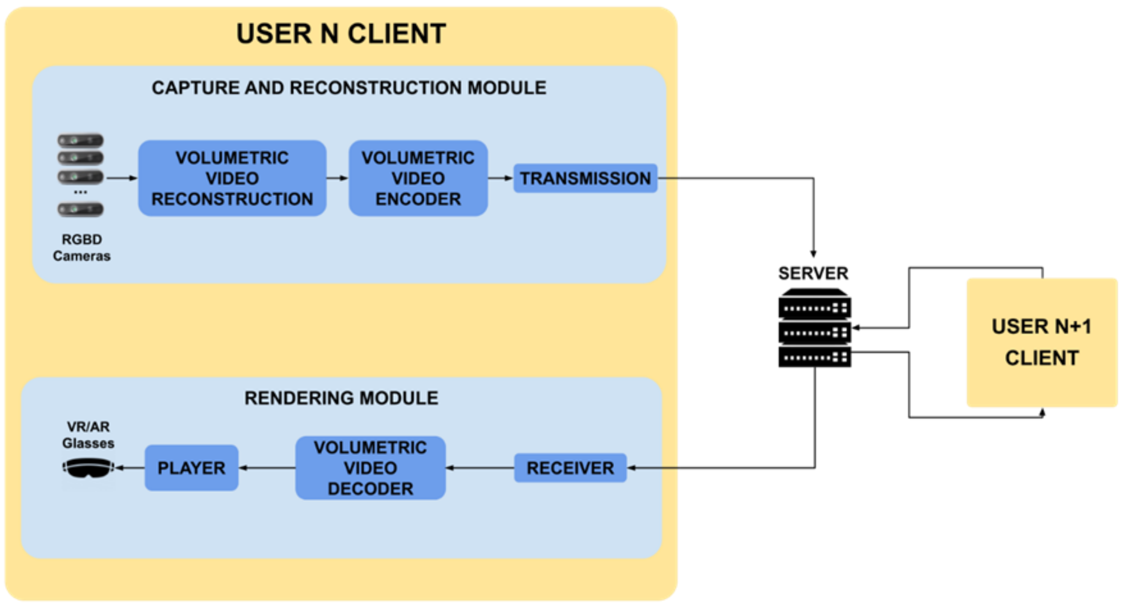


Figure 11. High-level system architecture to connect Unity-based players including the SDK to enable holographic communications. Note that the server box can represent either an SFU or MCU

The next key innovations are being applied to that Unity-based player in 6G-XR. First, it is being adapted to integrate the volumetric capture setups from Raytrix (Section 3). Second, it is being adapted to allow dynamic rate control from the clients, based on the network-assisted Rate Control APIs provided by the network (WP4). Third, the SDK for holoportation is being migrated to Linux, so it can be integrated with the Remote Renderer component, then virtualized and dynamically orchestrated (provided by WP2). These key innovations, and other planned ones, will be described with further details in the D3.2 – "Final versions of XR enablers".

5.1.2 Requirements

The Unity-based player, integrating the SDK for holographic communications, needs to run on a desktop computer or laptop equipped Windows 10 VR-ready device and GPU capabilities (e.g., NVIDIA RTX onwards). That PC then connects via cable to a VR headset for media presentation.

The player has been tested in Unity versions since v2020.3 one, with at least one of the Scriptable Render Pipelines and the new Input System active and enabled.

In SFU mode, the player has successfully run in sessions with up to 12 concurrent users. In MCU mode, the player has successfully run in sessions with up to 16 concurrent users.

5.1.3 Initial software

The Unity-based player, integrating the SDK for holographic communications, needs to run on a Windows 10 VR ready device, and it has been tested with a variety of Unity[17] versions, starting from 2020.3 and including more recent ones, with a key requirement of having at least one of the Scriptable Render Pipelines and the new Input System active and enabled.

[17] https://unity.com/

5.1.4 Initial hardware

The Unity-based player, integrating the SDK for holographic comms, needs to run on a Windows 10 VR ready device, like a desktop or laptop, with GPU capabilities (from NVIDIA RTX onwards). That PC then connects via cable to a VR headset for media presentation. Different VR headsets have been successfully tested so far, including Oculus Rift, Oculus Quest 1/2/3, Oculus PRO, and HTC VIVE.

5.2 WEBRTC STREAMING TO WEB PLAYER

5.2.1 Overview of the components

This section describes the web video player for WebRTC streaming and interaction that works with the Remote Renderer described in Section 4.3. It also includes the description of the necessary WebRTC signalling server to negotiate multimedia and network parameters before starting the communications between the Remote Renderer and the player.

The WebRTC protocol has been chosen because it is a widely employed solution for multimedia bidirectional communications in real-time scenarios and supported by most web browsers. It provides sub-second latency and enables to generate a personalized flow for each connected user (point-to-point communication). Therefore, it is ideal to satisfy a scenario with interactive users.

Figure 12 shows the fundamental elements of the WebRTC streaming solution composed of a server (Remote Renderer described in Section 4.3), which encodes and packages the content through standard codecs and protocols, and the Video Player (WebRTC Player). The WebRTC player has the role of receiving and decoding the content for display on the screen. The Signalling server is necessary during the initialization of the communication to negotiate media and network parameters.

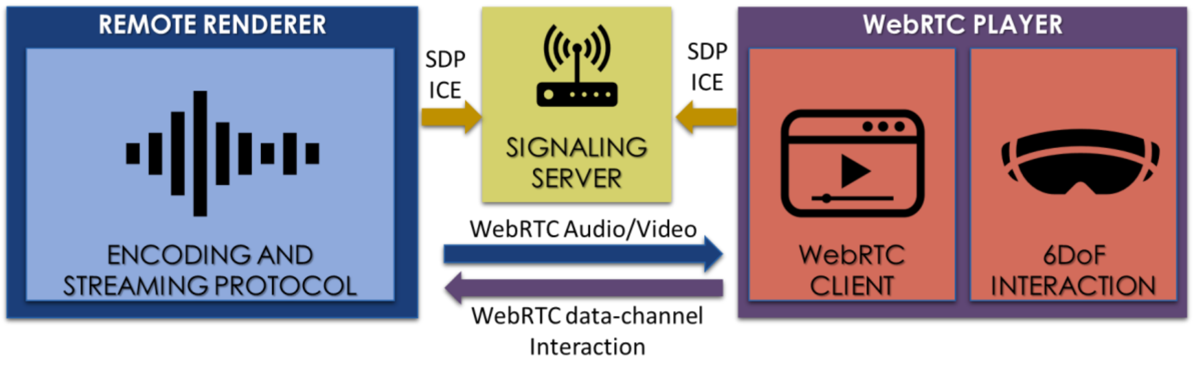


Figure 12. Components of WebRTC streaming.

More in detail, the different components are:

- Remote Renderer: this server, described in Section 4.3, includes a streaming server module capable of generating a WebRTC output. Moreover, it can receive the interaction information from the video player to personalize the video stream delivered though the WebRTC communication.
- WebRTC player: this video player is responsible for receiving the video and audio streams sent by the Remote Renderer through the WebRTC protocol and playing them. It is equipped with interaction capabilities, meaning that it collects and sends user's 6DoF information to the Remote Renderer. This player is based on web technologies that allow it to run within a web browser. As a result, it can run in any laptop where the interaction is provided through mouse

and keyboard, or in a VR Headset, where the interaction is generated through the sensors and the joysticks.

- Signalling server: the signalling server is an essential component in any WebRTC communication. It acts as an intermediary between the two peers of the communication, e.g., the Remote Renderer and the WebRTC player. Its main function is to facilitate the exchange of signalling information, such as negotiation of media session and connection parameters between the peers.

Figure 13 shows the communications diagram of the three components of the WebRTC streaming solution, which is based on Figure 9 and adds more details when WebRTC is specifically employed for the communication between Renderer and player. After the session configuration and to initialize the WebRTC communication, the WebRTC player makes a negotiation request using an Offer SDP with its multimedia parameters, through the signalling server. The signalling server sends the offer to the Remote Renderer, which generates a response, i.e., Answer SDP, with its multimedia parameters. The generated response is sent to the signalling server which subsequently sends it to the player. After this, the Interactive Connectivity Establishment (ICE) candidates are defined for the negotiation of network parameters. Like SDP negotiation, the player sends its candidates to the Remote Renderer via the signalling server, and then the Remote Renderer responds by sending its ICE candidates.

Once the negotiation is finished, the rendering loop starts. The Renderer renders the volumetric data and audio provided by external sources, such as SFU or MCU, applies the viewpoint and generates the 2D or 360^{o} video and audio output. Subsequently, the rendered data is sent to the WebRTC player though the communication channel established during the negotiation.

In the interaction loop, the WebRTC player collects 6DoF information and sends it to the server through the WebRTC data channel. This information is employed by the Renderer to update the viewpoint and generate the personalized video stream.

Finally, in the configuration loop, the video player uses *getStats()*[18] function in order to retrieve WebRTC metrics from the web browser where it is running. Since *getStats()* is a function defined in the WebRTC standard, it should be available in any browser implementation. These metrics are transferred to the monitoring system, where the Holo Orchestrator can access them and make decisions on the encoding profile. In the end, the Renderer applies the chosen profile by modifying its encoder configuration.

[18] https://www.w3.org/TR/webrtc-stats/

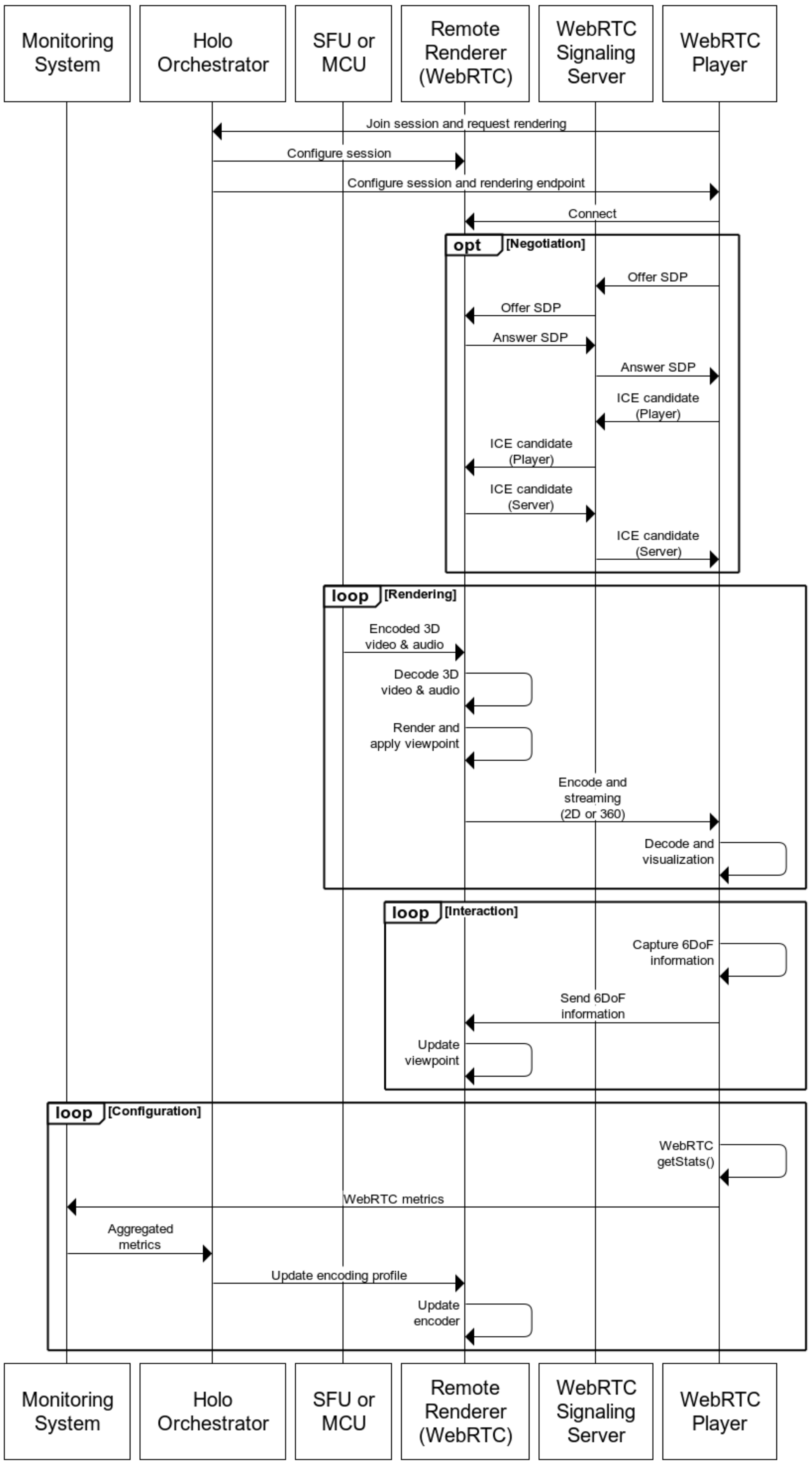


Figure 13. Specific communications diagram for WebRTC protocol.

5.2.2 Requirements

To define the requirements, the following factors for the WebRTC streaming solution are to be considered:

- **Delay in video distribution**: the delay between generation of content on the Renderer and its display on the player can affect the quality of the user experience. Real time latency (sub-second) is an essential factor in future XR services to provide a solid and natural experience between different participants.
- **Interaction with the VR scene**: interactive users have tight requirements in terms of latency since it affects their ability to interact with the VR environment. It means that the data channel, employed to transmit the 6DoF information and the video/audio channels, and the viewpoint application must also work in real time.
- **Variability of access network**: when a wireless access network (Wi-Fi or mobile network) is used to connect the Renderer and the player, it is necessary to consider that the coverage can vary from time to time. This can affect video playback, causing artifacts or video freezes, when the delivery cannot be performed due to insufficient transmission resources (jitter, bandwidth, etc.). The WebRTC stream must be flexible to allow dynamic video encoding configuration in real time. This helps to compensate for the variability of the access network and not compromise the stability, fluidity and latency of the transmitted video.
- **Heterogeneous user equipment**: the capabilities of the devices used to access the experience can vary between users. Both users connected with a laptop and users with a VR headset must be supported. It results that the WebRTC player must be lightweight and run in web browsers included in laptops and VR headsets.

Considering the aforementioned factors, the requirements of the WebRTC streaming components, such as WebRTC player and signalling server (Remote Renderer already presented in Section 4.3), have been identified and presented in Table 5.

Table 5. Requirements of the WebRTC streaming components.

Category	Requirement	Details
Audio and video formats to stream	2D video	Format: RAW video Maximum resolution: 4K Maximum frame rate: 60fps (maximum frame rate depends on configured resolution and processing hardware)
	360º video	Format: RAW video Maximum resolution: 4K Maximum frame rate: 60fps (maximum frame rate depends on configured resolution and processing hardware)
	Audio	Mono
Encoding	Interactive user	Video: VP8 or H.264 Audio: OPUS Mux: not employed
Protocols	Audio and video streaming	SRTP/UDP audio and video channels included in WebRTC
	Interaction through 6DoF information	Bidirectional data channel included in WebRTC

	Signalling / negotiation	WebSocket communications for SDP and ICE negotiations
Video adaptation	Interactive user	Performed in the Remote Renderer by means of WebRTC metrics provided by the WebRTC player
Supported devices	Interactive user device	Web browser on laptop or VR headset
Processing	OS	Signalling server: Ubuntu 22.04 LTS WebRTC player: any OS that includes a web browser
	Hardware	Signalling server: any HW capable to run a Node.js application WebRTC player: any HW with support for CPU or GPU video decoding
	Software	Signalling server: Node.js WebRTC player: - Laptop: Chrome web browser - VR headset: web browser embedded in the VR headset
	Containerization	Docker and Helm

5.2.3 Initial software

Signalling Server:

The Signalling Server is a Node.js[19]-based application. It is implemented with JavaScript and TypeScript programming languages. Since its only functionality is to provide connection between the two WebRTC peers (Remote Renderer and WebRTC Player), it embeds a library for enabling WebSocket communications.

The software dependencies and their versions are shown in Table 6.

Table 6. Software versions used in the signalling server.

Software and frameworks	Version
Node.js	21.2.0
@types/ws (Node.js WebSocket library)	8.5.3

WebRTC Player:

The WebRTC Player is implemented with native WebRTC technologies. To improve the visualization and allow both 2D and 360º videos, A-Frame[20] has been integrated. A-Frame is an open-source web

[19] https://nodejs.org/en

[20] https://aframe.io/

framework developed by Mozilla that is specifically designed for building VR experiences on the web. It simplifies the creation of VR content by providing an HTML-based declarative syntax that allows developers to define 3D scenes and interactions directly on their web pages. It can be used from any device that has a web browser with WebXR.

The WebRTC Player is also responsible for capturing 6DoF information from input devices (keyboard and mouse if using a laptop, or sensors and joysticks if using a VR headset). This information is used to enable the interaction of users with each other and with the 3D virtual environment. The resulting WebRTC Player is shown in Figure 14.

The development used mainly JavaScript programming language. For the deployment, Node.js[21] and Vite[22] are employed.

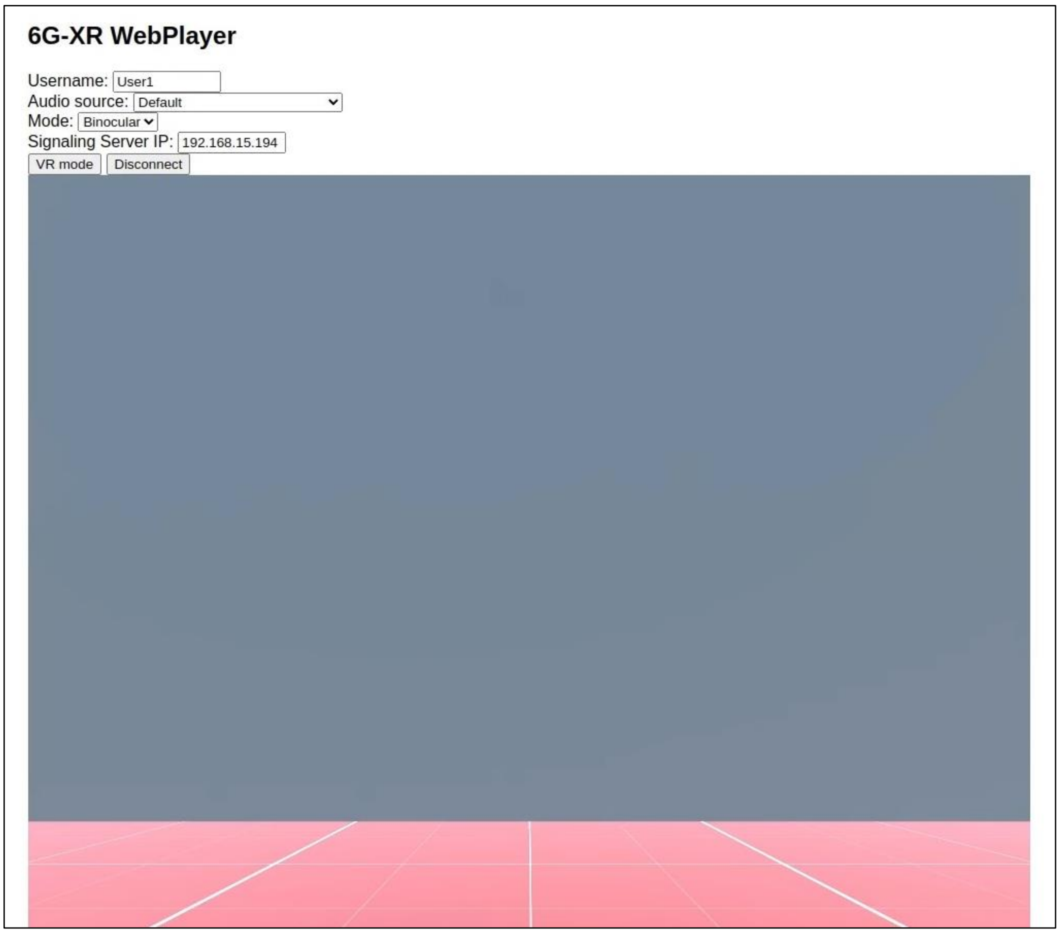
The software dependencies and their versions are shown in Table 7.

Table 7. Software versions used in the WebRTC Player.

Frameworks	Version
Node.js	21.2.0
Vite	4.5.0
A-Frame	1.4.2

[21] https://nodejs.org/en

[22] https://vitejs.dev/


6G-XR WebPlayer

Username: User1
Audio source: Default
Mode: Binocular
Signaling Server IP: 192.168.15.194

Figure 14. WebRTC Video Player.

5.2.4 Initial hardware

Regarding the hardware on which the Signalling Server and WebRTC Player are implemented, it is important to note that both are hardware-agnostic. It means that they do not require any specific hardware, as they can be deployed as Node.js containerized applications.

The WebRTC Player is tested by using the Chrome web browser when accessing with a laptop, and the Browser application included in the Meta Quest 2[23] VR headset.

5.3 DASH STREAMING TO WEB PLAYER

5.3.1 Overview of the components

This section describes the web video player for DASH streaming that works with the Remote Renderer described in Section 4.3. It also includes the description of the necessary HTTP Server where the DASH MPD and its audio and video segments are stored to be served to the video player. Although DASH has higher latency compared to WebRTC (Section 5.2), it is still a valid option when the interaction with the VR scene is not required.

[23] https://www.meta.com/es/en/quest/products/quest-2/

DASH is the widely employed solution for multimedia communications when scalability rather than real time is sought. DASH is based on the HTTP protocol which ensures extensive scalability compared to WebRTC. However, its disadvantage is its intrinsic delay of the multimedia communication due to media segments creation. Thus, it is not suitable for interactive communications, but it is considered the ideal fit for passive media consumption, where the interaction is not necessary, and the higher scalability allows a wider number of participants. It means that all the participants have the same viewpoint from which they participate in the VR experience without interacting with it.

Figure 15 shows the fundamental elements of the DASH streaming solution composed of the DASH stream producer, i.e., the Remote Renderer described in Section 4.3, which encodes and packages the content through standard audio and video codecs, and the DASH stream consumer, i.e., the DASH Player, which receives and decodes the content for displaying it on the screen. Between them, a common HTTP Server is necessary to store the MPD and segments generated by the Remote Renderer and serve them to the DASH player.

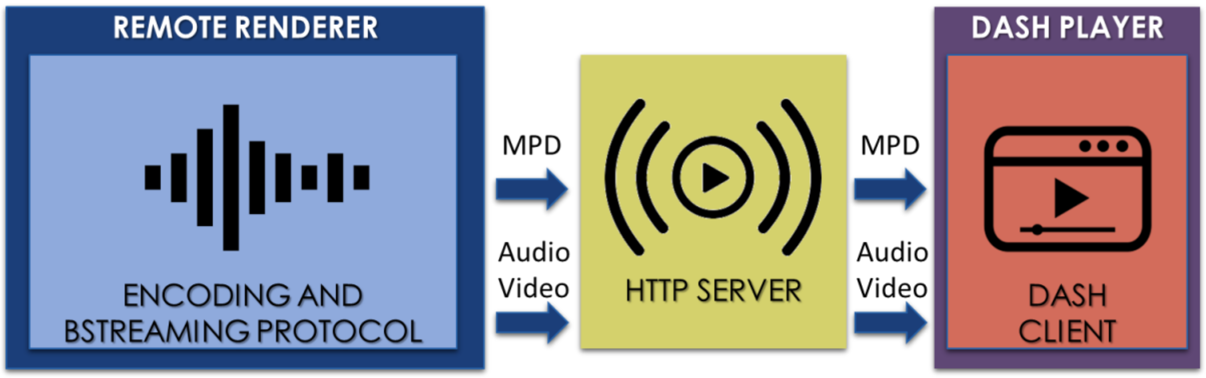


Figure 15. Components of DASH streaming.

More in detail, the different components are:

- Remote Renderer: as described in Section 4.3, this server includes a streaming server module capable of generating a DASH output. A DASH stream consists of a MPD manifest describing the content, and media segments where the audio and video data are encoded. All these files are written locally in the server. In this case, no interaction information from the video player is received as the DASH stream is unique for all connected participants.
- HTTP Server: when generating the DASH stream, the Remote Renderer creates the MPD manifest and the segments and then write them on the disk. It does not provide any network capabilities to deliver them to the video player. For such a reason, this HTTP server is necessary to serve the generated files (MPD and segments) to the video player through HTTP.
- DASH player: this video player is responsible for performing HTTP request to the HTTP server to download the DASH stream, decode its content and display it. It does not provide any interaction capabilities. This player is based on web technologies that run within any web browsers.

Figure 16 shows the communications diagram of the streaming solution using DASH, which is based on Figure 9 and adds more details when DASH is specifically employed for the communication between the Remote Renderer and the player. After the session establishment and configuration, the rendering loop starts to render volumetric video content provided by an external source, such as SFU or MCU. The Remote Renderer generates the rendered representation in 2D or 360° format and encodes it with H.264 or HEVC/H.265 codecs for video and AAC codec for audio. The encoding process of the video (not the audio) could be performed up to three times to generate three different representations of the same video and exploit the adaptive streaming capabilities of the DASH protocol. These three parallel encoding processes do not affect the latency if all of them are performed in real time (hardware acceleration is required to achieve it). The encoded files are subsequently multiplexed into MP4 and segmented. The MP4 segments are stored on the HTTP server and the MPD manifest is generated with the segment information so that the DASH player can perform HTTP requests to receive the MPD manifest and MP4 segments. Finally, the DASH player decodes the segments and views them on the user's device screen.

When using DASH for passive consumption, there is no interaction loop as in WebRTC. The configuration loop is the only one which runs in parallel to the rendering loop. The DASH player performs measurements while downloading the DASH segments and sends them to the monitoring system. The Holo Orchestrator can access these metrics to make decisions on the video representations included into the MPD manifest. It means that, even the remote renderer generates all the video representations configured when starting the stream, not all of them are shown to the DASH player. For example, the Holo Orchestrator might detect that the current state of the network does not provide enough transmission capacity to allow timely reception of the highest representation and then, it decides to remove it from the MPD manifest.

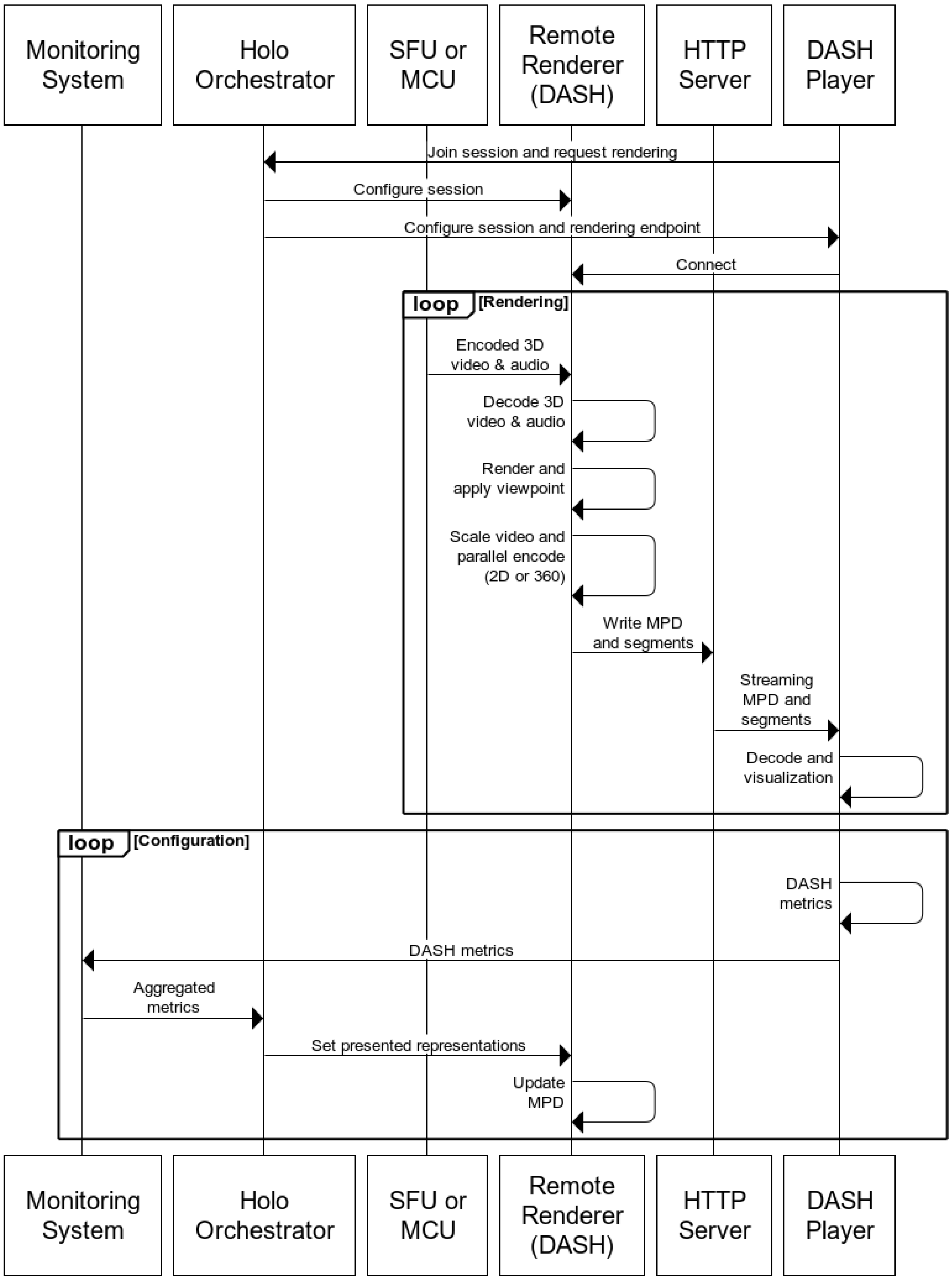


Figure 16. Specific communications diagram for DASH protocol.

5.3.2 Requirements

To define the requirements, the following factors for the DASH streaming solution have been considered:

1. Latency and synchrony in video distribution: when using DASH, the user already accepts to have higher latency compared to WebRTC, but it does not accept asynchronous visualization. All the participants should be visualizing the same video instant of the overall video stream, otherwise they could experience a different quality (QoE).
2. Video delivery scalability: the variable number of users affects the processing capabilities needed to generate the output video streams. Video encoding tasks are compute-intensive and can rely on GPU hardware acceleration solutions to increase scalability.
3. Access network variability: when a wireless access network (Wi-Fi or 5G/6G cellular network) is used to connect Renderer and player, it is necessary to consider that the coverage can vary from time to time. This can affect video playback, causing artifacts or video freezes, when the delivery cannot be performed due to insufficient transmission resources (jitter, bandwidth, etc.). The DASH stream must allow the creation of different video representations of the same video content such that each player can decide the one that fits with the network status. The server can also influence the decision of the player by adding or removing the representations available in the MPD. Each video representation is characterized by a specific encoding configuration. This strategy is based on periodical update of the MPD and helps to compensate for the variability of the access network and not compromise the stability, fluidity and latency of the transmitted video.
4. Heterogeneous user equipment: the capabilities of the devices used to access the experience can vary between users. Both users connected with a laptop and users with a VR headset must be supported. It results that the DASH player must be lightweight and run in web browsers included in laptops and VR headsets.

Considering the aforementioned factors, the requirements of the DASH streaming components, such as HTTP server and DASH player (Remote Renderer already presented in Section 4.3), have been identified and presented in Table 8.

Table 8. Requirements of the DASH streaming components.

Category	Requirement	Details
Audio and video formats to stream	2D video	Format: RAW video Maximum resolution: 4K Maximum frame rate: 60fps (maximum frame rate depends on configured resolution and processing hardware)
	360º video	Format: RAW video Maximum resolution: 4K Maximum frame rate: 60fps (maximum frame rate depends on configured resolution and processing hardware)
	Audio	Mono
Encoding	Passive user	Video: H.264 or H.265 Audio: AAC Mux: Fragmented MP4
Protocols	Type of MPD	Live MPD with periodical updates
	MPD delivery	HTTP request form DASH player to HTTP server

	Media segment delivery	HTTP request form DASH player to HTTP server
Video adaptation	Passive user	Performed in the Remote Renderer by means of DASH metrics provided by the DASH player
Supported devices	Passive user device	Web browser on laptop or VR headset
Processing	OS	HTTP server: Ubuntu 22.04 LTS DASH player: any OS that includes a web browser
	Hardware	HTTP server: any HW capable to run a Node.js application DASH player: any HW with support for CPU or GPU video decoding
	Software	HTTP server: Node.js DASH player: - Laptop: Shaka-player running in Chrome web browser - VR headset: Shaka-player running in web browser embedded in the VR headset
	Containerization	Docker and Helm

5.3.3 Initial software

HTTP Server:

The HTTP Server is a simple Node.js application using the default HTTP-Server library to serve files. The software dependencies and their versions are shown in Table 9.

Table 9. Software versions used in HTTP Server.

Software and frameworks	Version
Node.JS	12.22.9
HTTP-Server	14.1.1

DASH Player:

The DASH Player is based on the Shaka Player and deployed with Node.js. The software dependencies and their versions are shown in Table 10.

Table 10. Software versions used in DASH Player.

Software and frameworks	Version
NodeJS	12.22.9
Shaka Player	4.3.0

Figure 17 shows a demonstration of the Dash Player where a comparison with the source is made. Source (Remote Renderer on the left) and destination (DASH Player on the right) communicate locally (localhost), having a latency of approximately 3 seconds. Further tests will be performed later when the Remote Renderer and HTTP Server will be deployed in the Edge infrastructure.

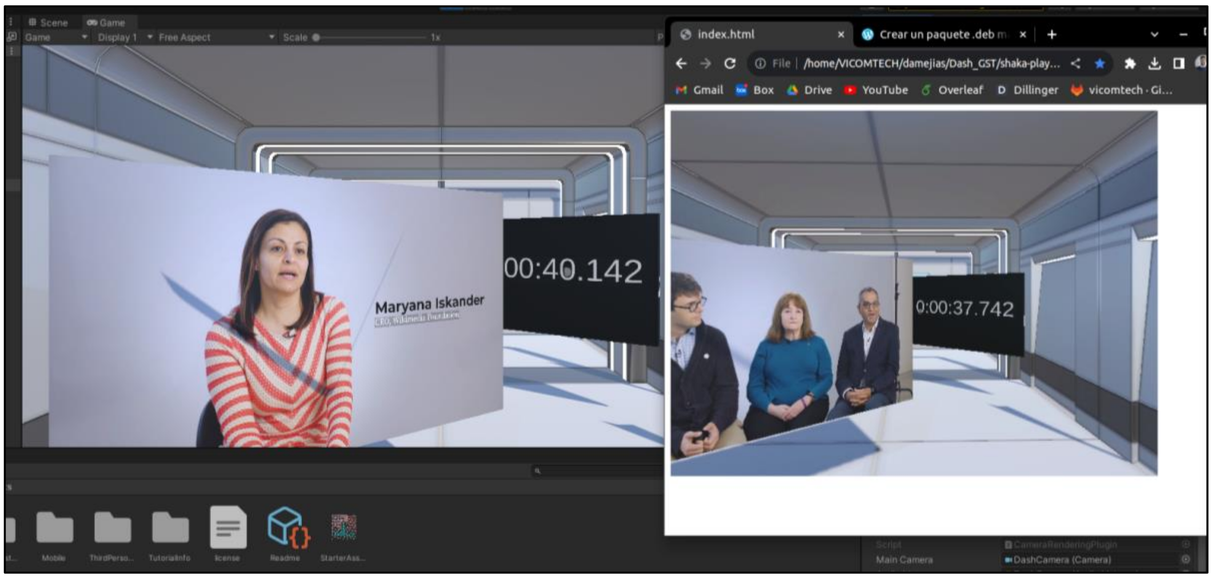


Figure 17. DASH Player based on Shaka Player.

5.3.4 Initial hardware

Regarding the hardware on which the HTTP Server and DASH Player are implemented, it is important to note that both are hardware-agnostic. It means that they do not require any specific hardware, as they can be deployed as Node.js containerized applications.

It is important to note that the HTTP Server needs to serve the content generated by the Remote Renderer described in Section 4.3. As a result, in this first release it has been containerized together with the Remote Renderer.

The DASH Player is tested by using Chrome web browser when accessing it with a laptop.

6 MULTI-MODAL SYNCHRONIZATION

In the 6G-XR project, device and media synchronization play a key role, as the project deals with real-time streaming and communications involving various sensors, streams, media modalities, and distributed users.

This section analyses the alternatives for media synchronization according to 3GPP and presents the first versions of clock and media synchronization mechanisms used in 6G-XR to support the XR Enablers that have time constraints or perform time-dependent processing.

6.1 OVERVIEW ON SYNCHRONIZATION IN 3GPP

In this section, we present the proposed synchronization procedures for XR and multi-modal applications. For exploiting these results, we have chosen to peruse potential standardization opportunities, and therefore, implementation of these procedures will be out of scope for 6G-XR.

6.1.1 Network-assisted media synchronization in 3GPP

Table 11 includes information that may be used for synchronization. The synchronization status parameters may include those that may be used to associate UE(s), Protocol Data Unit (PDU) Session(s), and Data Flow(s) that are part of the same multi-modal data set. The identifying information may be an XR and Media services (XRM) session identifier. Data Flow(s) may be identified with an IP 5-Tuple. PDU Session(s) may be identified with a PDU Session ID. UE(s) may be identified by an International Mobile Subscriber Identity (IMSI), Subscription Concealed Identifier (SUCI), or External Identifier.

Table 11. Synchronization Parameters.

Information Element	Description / Examples
XRM Session Identifier	The XRM Session Identifier is identifying Information that can be used to associate the rest of the information in this table with UE(s), PDU Session(s), and Data Flow(s) that are part of the same multi-modal data set.
Presentation Time	Presentation time may be used to synchronize an application's separate (single-modal) data streams (e.g., audio, video, subtitles) when the media are presented to the user. Presenting to user includes (but not limited to) video frame being displayed on a display, sound played out through speakers, haptic information being conveyed to user through actuation.
PDU Set arrival/received time	The PDU Set arrival/received time is the timestamp specifies the time a specific PDU Set has been received by the UE. The PDU Set may be identified with a combination of an XR Session ID, a PDU Set identifier, and a Synchronization Group Identifier.

Payload/Content type	The Payload/Content type indicates the type of data that is associated with the synchronization information (e.g., audio, video, haptic). This information is helpful because synchronization threshold requirements differ per each type of mode. Therefore, along with timing information, the Payload/Content type can be sent.
Synchronization Error Calculations	Synchronization Error Calculations indicate the time difference between any two data flows (e.g., audio-video, video-haptic). Per each flow of data being used by the application, the synchronization error value is calculated. Either PDU Set arrival/received time, or the presentation time could be used for calculating this value. Alternatively, synchronization error values of both PDU Set arrival/received time and presentation times may be used. In case some flows are distributed among other UEs, a single UE receives/collects PDU Set arrival/received time or the presentation time information from other UEs for calculating this value(s). The flows may be identified with a PDU Set Flow Identifier, a Synchronization Group Identifier, or an IP 5-Tuple.
Synchronization Information that was collected from other UEs	This information may be collected via device-to-device communications and may indicate the degree to which two or more flows are synchronized. The information may be expressed in units of time. The Flows may be identified with a combination of UE identifier, XR Session ID, a PDU Set identifier, and a Synchronization Group Identifier. The UE identifier may be a GPSI, IP Address, or a 5G-S-TMSI.
Information about buffer overflow or underflow events	This information may indicate the number, or frequency, of buffer overflow or underflow events. The information may be associated with a flow identifier (e.g., a PDU Set Flow or an IP 5-tuple).

6.1.2 Detecting synchronization status

A UE may detect that data from a flow that the UE is transmitting or receiving is not sufficiently synchronized with a second flow. The second flow may be associated with the UE or a second UE.

For instance, as shown in Figure 18, a UE may send the synchronization information and the information about the associated PDU Session, IP 5-tuple, or QoS Flow to the network in a NAS-SM message. The message may be received by the Session Management Function (SMF) that is associated with the PDU Session. The SMF may use the information to take actions such as sending updated QoS Rules to the UE or QoS Enforcement Rules (QER) to the User Plane Function (UPF). The SMF may also forward the synchronization information to the Policy Control Function (PCF) so that the PCF may update Policy and Charging Control (PCC) Rules or UE Route Selection Policy (URSP) rules. The SMF may also forward the synchronization information to the Network Data Analytics Function (NWDAF) so that the NWDAF may consider the synchronization information when generating network data analytics.

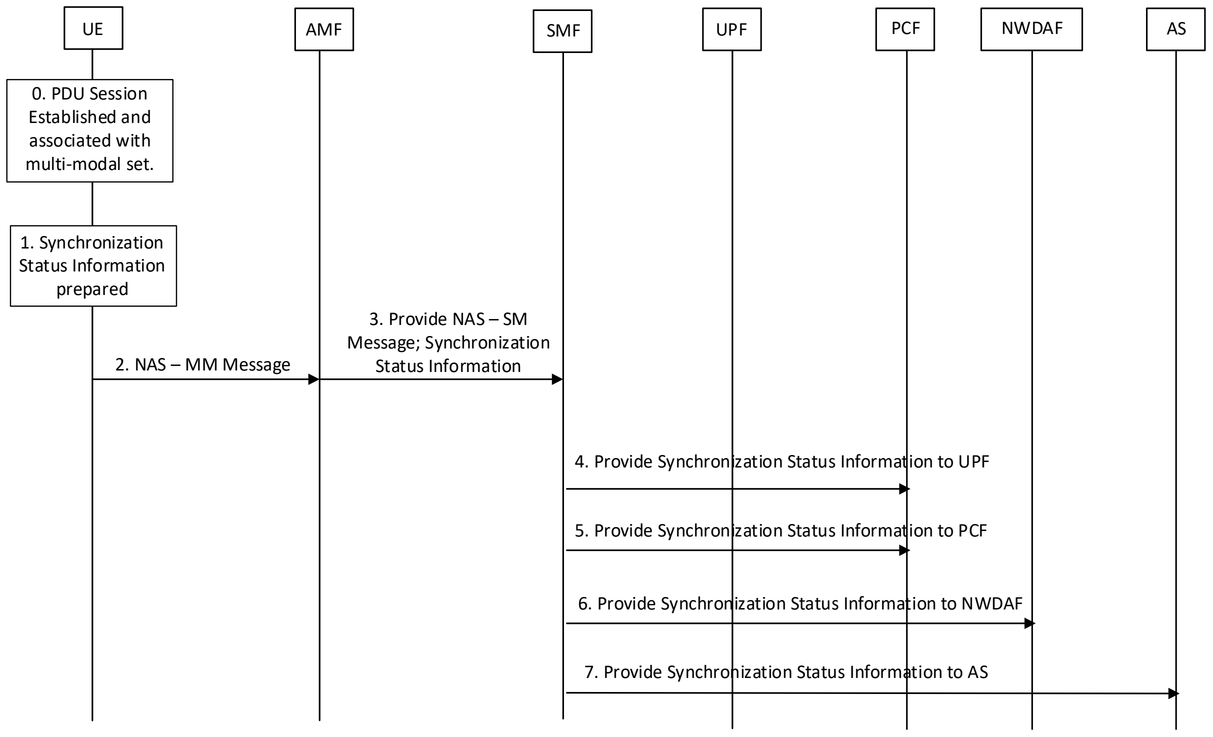

UE
AMF
SMF
UPF
PCF
NWDAF
AS
0. PDU Session Established and associated with multi-modal set.
1. Synchronization Status Information prepared
2. NAS – MM Message
3. Provide NAS – SM Message; Synchronization Status Information
4. Provide Synchronization Status Information to UPF
5. Provide Synchronization Status Information to PCF
6. Provide Synchronization Status Information to NWDAF
7. Provide Synchronization Status Information to AS

Figure 18. Exemplary Procedures for Synchronization Status Reporting via the Control Plane.

Alternatively, Figure 19 illustrates an example where the synchronization status information is sent to the network via UP signalling. The UP signalling may terminate at an Application Server (AS) and the AS may forward the synchronization status information to network functions and/or trigger network functions to take action to improve the overall QoE. The X5 interface is an example of an interface that may be enhanced to allow the UE hosted application to send synchronization status information over the UP.

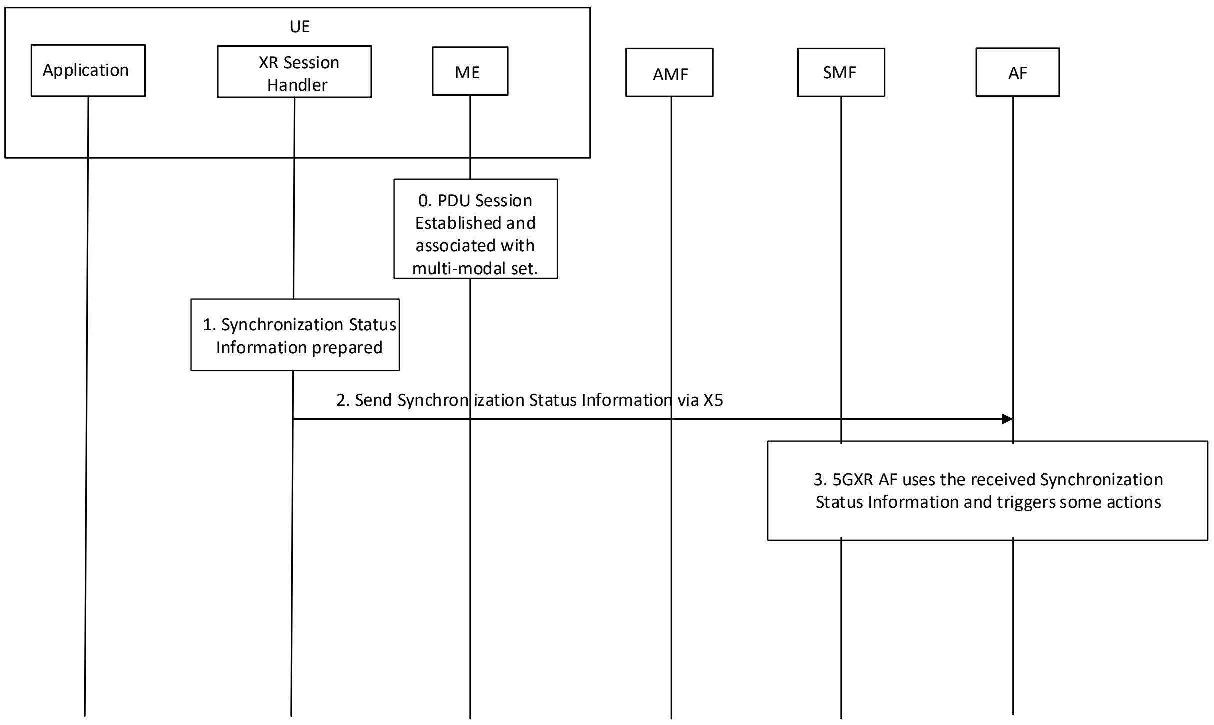

UE
Application
XR Session Handler
ME
AMF
SMF
AF
0. PDU Session Established and associated with multi-modal set.
1. Synchronization Status Information prepared
2. Send Synchronization Status Information via X5
3. 5GXR AF uses the received Synchronization Status Information and triggers some actions

Figure 19. Exemplary Procedures for Synchronization Status Reporting via the User Plane.

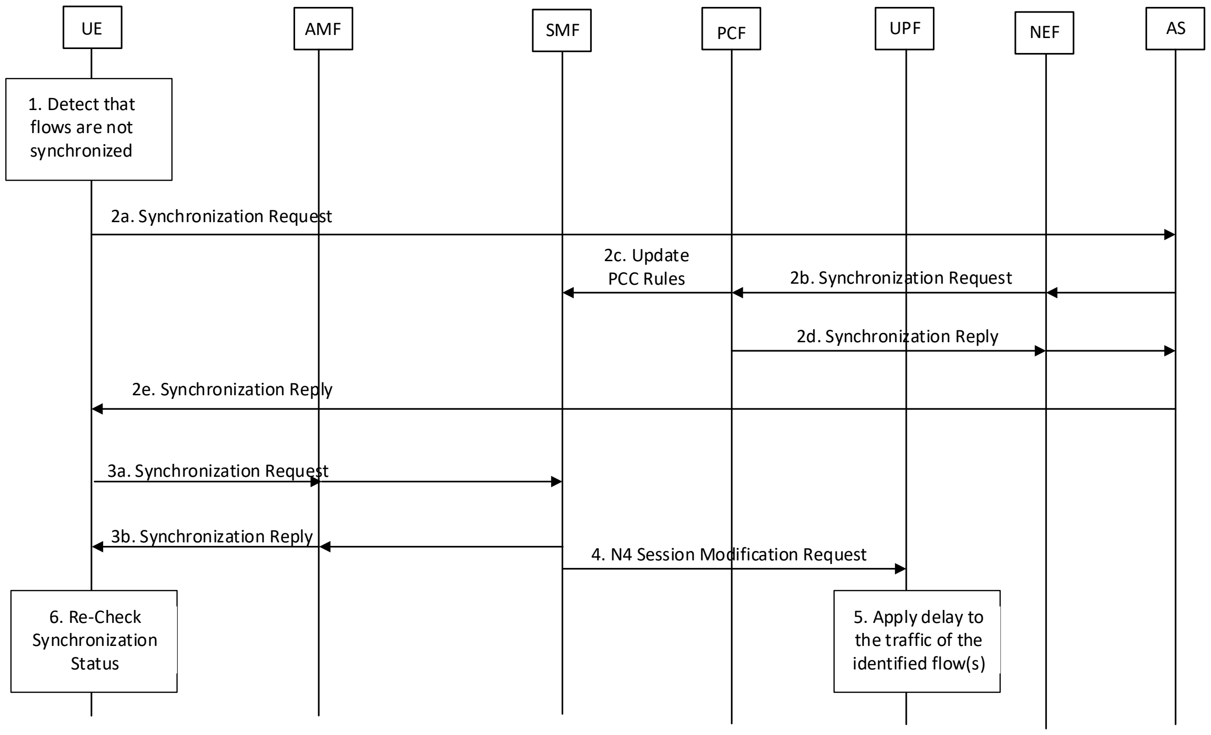

UE
AMF
SMF
PCF
UPF
NEF
AS
1. Detect that flows are not synchronized
2a. Synchronization Request
2c. Update PCC Rules
2b. Synchronization Request
2d. Synchronization Reply
2e. Synchronization Reply
3a. Synchronization Request
3b. Synchronization Reply
4. N4 Session Modification Request
6. Re-Check Synchronization Status
5. Apply delay to the traffic of the identified flow(s)

Figure 20. Exemplary Procedures for Network-Assisted Media Synchronization.

Figure 20 demonstrates procedures for network-assisted media synchronization. The UE may detect that a first flow is not synchronized with one or more other flows (PDU sets, PDUs).

The UE may send an application layer message to an AS to indicate that the first flow is not synchronized with one or more other flows. The request from the AS/Network Exposure Function (NEF) triggers the PCF to send updated PCC rules to the SMF.

Alternatively, the UE may send a Synchronization Request to the SMF directly. This message may indicate that the first flow is not synchronized with one or more other flows.

Accounting for the synchronization error, if any, SMF may send an N4 message to the UPF to configure the UPF to introduce a delay to one or more flows to ensure synchronous delivery of inter-related data.

6.2 CLOCK SYNCHRONIZATION

In the 6G-XR project, synchronization across VNFs (XR Enablers) deployed in computing infrastructure and end user applications running on UE, are major requirements to enable the accurate execution of time-dependent or delay-sensitive operations. As a result, the 6G-XR project employs two different synchronization methods to improve the overall synchronization capabilities when deploying the developed XR Enablers: Clock synchronization, described in this section, and Media synchronization, described in Section 6.3.

6.2.1 Overview on clock synchronization protocols

Clock synchronization is necessary in order to provide the correct time to any device or host that execute any XR Enablers. The, in this section the alternatives for clock synchronization are compared to select the appropriate one for the 6G-XR project.

Network Time Protocol (NTP) and Precision Time Protocol (PTP) represent the most employed protocols to provide system clock synchronization. Table 12 describes the main differences between them.

Table 12. Comparison between NTP and PTP.

Feature	NTP	PTP
Accuracy	Millisecond	Sub-microsecond
Synchronization mechanism	NTP Client requests to NTP Server	PTP Grandmaster sends information to slaves
Scope	Wired and Wireless networks	Only Ethernet networks. Research activities within SNS projects to enable wireless networks.
Hardware requirements	Hardware-independent	Specific Ethernet adapters

Both protocols enable the exchange of 64 bits timing information, but PTP provides better accuracy as relies on hardware timestamping instead of software implementation. Moreover, concerning synchronization mechanism, NTP relies on a client-server communication, while PTP relies on a master-slave one.

Finally, NTP is more flexible in terms of network that can be employed as it works with any networks, while PTP works only with Ethernet as enabler for deterministic networking. Research activities to enable deterministic networking on wireless networks, including synchronization based on PTP, is still ongoing in SNS projects (e.g., DETERMINISTIC6G[24] and PREDICT-6G[25] projects).

6.2.2 Requirements and implementation

Table 13 presents the requirements concerning clock synchronization. It is remarkable that all the requirements are satisfied by employing a standard protocol such as NTP to synchronize the clock of the hosts at the edge where Media Functions run or of the UEs with the client application.

Table 13. Clock synchronization requirements.

Category	Requirement	Description
Telemetry	Host OS to provide correct timestamp to metric exporters	Metric exporters employ timestamps to relate the measurement to the exact time it was generated. This is necessary for real-time operations or post-processing.
Media Streaming	Host OS to provide correct date/time to web browsers	Default implementation of web video players employ date/time provided by Host OS to the web browser in order to synchronize.
Media Streaming	Host OS to provide correct date/time to Unity environment	Default implementation of Unity-based applications employ date/time provided by Host OS to the Unity environment in order to synchronize.

The necessary clock synchronization for Media Functions and Monitoring System of 6G-XR will use NTP protocol due to several reasons that makes it more suitable than PTP:

- NTP works independently on the network type (wired or wireless) and network adapter, while PTP only works with Ethernet and with compatible network adapters. It means that NTP provides more flexibility as it can be employed both to synchronize Media Functions at the edge with Ethernet and UEs with wireless adapters (Wi-Fi or 5G). PTP would limit the synchronization only to Ethernet connected devices.

- Even if the accuracy of PTP (microseconds) is higher than NTP (milliseconds), such accuracy is not needed by Media Functions where the minimum appreciated error delay depends on the frame rate of the streamed video. Employing a high frame rate such as 60 fps means a minimum delay of 16 ms between frames, which is 10x higher than the error that can be caused by NTP.

- Monitoring System and Media Functions developed in 6G-XR work independently from the protocol employed for clock synchronization, as they rely also on media synchronization mechanisms to improve the synchronization. Moreover, both NTP and PTP expose 64 bits timing

[24] https://deterministic6g.eu/

[25] https://predict-6g.eu/

information to the system that can be accessible by any functions running on the system itself. It means that NTP could be substituted in the future by PTP, when wireless adapter will be enabled, and the application will have in any case 64 bits of timing information.

Media Functions will be deployed as Helm Charts[26] within Linux-based environments (e.g., Ubuntu). Thus, NTP implementations of NTP Server and NTP Client are widely available in Linux environment and can easily be deployed to provide timestamp information to the applications.

The NTP Server could be either public or private, deployed in one of the Edge infrastructures of the 6G-XR project. In both cases, an NTP client is deployed and configured in any Host OS to access the NTP Server and synchronize itself.

6.3 MEDIA SYNCHRONIZATION

Media synchronization represents a further step to improve synchronization among XR Enablers, as it moves from providing exact time to devices and hosts (clock synchronization, described in Section 6.2) to also providing exact time to the application. Thus, the media synchronization employed to improve the application layer synchronization of any XR Enablers is described in this section.

In the context of 6G-XR and its use cases, media synchronization is needed for the following operations:

- Estimate E2E delays: the delay across different modules of the E2E chain can be measured only by having an accurate synchronization of each module. This allows to correctly generate timestamps to be applied to every Media Unit (MU) (i.e., video frame or audio sample).
- Performance monitoring: to collect performance metrics from the planned UCs, the synchronization of the host system where the VNFs (XR Enablers) are deployed is necessary. The objective is to employ performance metric exporters whose information has the correct timestamp. Thus, it is possible to relate the collected measurements to the exact time they were generated and to perform real-time operations or post-experiment data processing.
- Launch and/or activate features of the XR Enablers whenever needed: some features provided by the XR Enablers depend on clock synchronization. For example, the DASH Player uses current timestamp to calculate media segments to download time to time from the HTTP Server.
- Media service operations: some media services might have tight timing constraints. For example, any XR interactive applications, where the user can interact with the VR scene, need to work in real-time as they are highly sensitive to time delay. A delay in user interaction directly affects its QoE and might also cause motion sickness.

In the end, enabling all the operations mentioned above guarantees consistency when processing several media streams at the same time (e.g., synchronized audio and video to avoid lip-sync issues) and enables a coherent XR experience when one or more users enter a VR scene (e.g., a movement in the physical world must be synchronized with the VR representation).

[26] https://helm.sh/docs/topics/charts/

6.3.1 Requirements and implementation

In this section, the requirements of 6G-XR Media Functions in terms of media synchronization, the specific implementations carried out, and the solutions already integrated with the Media Functions, are explained.

The required different media synchronization variants and features, together with the specific implementations or integration carried out to fulfil them, are outlined in Table 14.

Table 14. Media synchronization requirements and carried out implementations.

Category	Requirement	Implementation
Intra-media synchronization	The captured Media Unit (MU), i.e., video frame and/or audio sample, at the source needs to have an accurate timestamp that must be preserved until reaching the destination client.	Timestamps are inserted into each MU, based on the clock of the host, and preserved during media encoding and transmission.
	The SFU needs to preserve the original timing patterns of MUs for each of the incoming/outcoming streams.	The SFU does not modify timestamps, and it waits until the entire MUs are received to relay them.
Inter-media synchronization	The original temporal dependence (synchrony) between MUs of different media streams originated by the same origin needs to be preserved until reaching the destination client (i.e., avoiding lip-sync).	Media streams (video and audio) are not multiplexed when captured, and inter-media sync is provided on a best-effort basis. If necessary, the audio and video streams are re-synchronized at the Remote Renderer when transmitting them to the destination: - Video and audio streams are synchronized and multiplexed when employing DASH. - Video and audio streams are synchronized with transport level protocols (RTP/RTCP) when employing WebRTC.
	The SFU needs to preserve the original timing patterns of	The SFU does not modify timestamps, and it waits until the entire MUs are received to relay them.

	MUs for each of the incoming/outcoming streams.	
Inter-source synchronization	Synchronization across capturing sensors for 3D reconstruction of the same modality.	It is provided for multiple camera sensors when the 3D reconstruction module is run locally. It relies on the CoaXPress CXP-12 provided to connect the multi-camera setup. Wall-clock timestamps to capture frames from each sensor could also be inserted to allow for an in-sync volumetric reconstruction.
Inter-destination synchronization	Synchronization across multiple destination devices/clients, for each of the received media streams, needs to be guaranteed. - This applies to native clients and to web video players. - Synchronization between native clients and web video players would involve delaying real-time communications for native clients, which is not desired.	Audio and video streams at the destination are synchronized comparing the clock of the host and contextual information (metadata) provided by media streams: - DASH player: timestamp included in DASH MPD. - WebRTC player: RTP/RTCP packets timestamping. - Native clients: no inter-destination synchronization is handled to avoid adding delays and affecting the user's experience. This client is expected to be the best performant, having lower delay since it does not need the Remote Renderer.

6.3.2 Initial evaluation of media synchronization

This section analyses the outcomes obtained when testing the media synchronization achieved with the first versions of the XR Enablers described in this document. The tests are carried out by proposing a QoS measurement method based on image and audio processing, which enables measuring the E2E video and audio latency and inter-destination (only web video players have been considered) and intra-media synchronization [6].

Specifically, in a multiuser XR experience scenario delivered through WebRTC, the following XR Enablers are employed:

- Remote Renderer (described in Section 4.3): it is configured in multi-player mode, where each user/video player receives a personalized media stream (specific viewpoint is applied). It uses WebRTC protocol with VP8 and OPUS codecs for video and audio, respectively.

- Signalling Server for WebRTC (described in Section 5.2): the Signalling server is only necessary for the synchronization and does not affect the media synchronization.

- WebRTC Player for receiving and playing WebRTC stream (described in Section 5.2): a different WebRTC player is run for each of the connected users. All users are using a laptop with Chrome web browser where the WebRTC runs.

Testing is repeated considering different access networks such as Ethernet, Wi-Fi, and 5G Stand Alone for the connectivity of the clients/video players. Our analysis focuses on a context where five users are connected to the Remote Renderer and receive a personalized media stream simultaneously.

Table 15 shows the summary of the obtained results when measuring the E2E latency and inter-destination asynchrony, also called inter-device asynchrony. The employed method is explained in [6]. The shown values describe the average values of the 5 users. The lowest average values for both video and audio E2E latency are obtained in the case of Ethernet, followed by 5G, and ending with Wi-Fi. Regarding inter-destination asynchrony, in the case of Ethernet or 5G, we obtain values below 75 ms. In the case of Wi-Fi, we see that the inter-device asynchrony for video rises to 514.2 ms.

Table 15. Initial synchronization results.

		Ethernet	Wi-Fi	5G
Average E2E Latency (ms)	Video	227.54	362.46	282.67
	Audio	185.22	324.59	304.17
Inter-destination asynchrony (ms)	Video	54.2	514.2	71.8
	Audio	45	69	38.8

Figure 21 visually presents audio and video latencies experienced by each of the 5 connected users. The bars represent the minimum latency recorded for each device and for each stream (audio and video) during the test, while the upper whiskers illustrate the variability, or jitter, of these data, representing the highest value and the average of all values. Then, a composite of all player samples is depicted (bars and whiskers on the right of each graph labelled as inter-device), providing insights of inter-destination asynchrony for both video and audio.

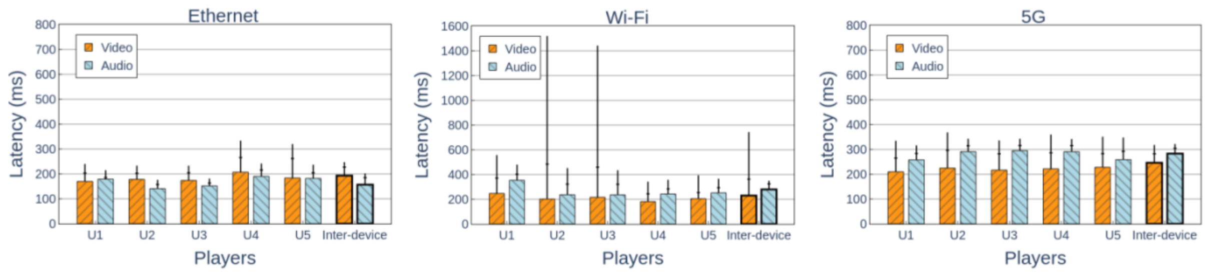


Figure 21. Inter-destination asynchrony.

Regarding intra-media synchronization, also known as lip-sync, the obtained results can be seen in Figure 22. It is important to note that the y-axis is not the same for all three graphs, as the results obtained with Wi-Fi show outliers far from the values obtained with the rest of the technologies.

Specifically, the presence of outliers in the Wi-Fi data, distant from the corresponding values observed with alternative technologies, highlights anomalous behaviour within the Wi-Fi network, as asynchrony values of up to 650 ms have been recorded. Upon closer examination, it becomes apparent that this intra-media asynchrony can be attributed to instances where video playback experienced interruptions, leading to temporal disparities between video and audio streams. Despite these interruptions, the audio component remained largely unaffected. In the case of Ethernet, it is observed that the values of intra-media asynchrony generally remain below 100 ms, and even dip below 50 ms in the case of 5G.

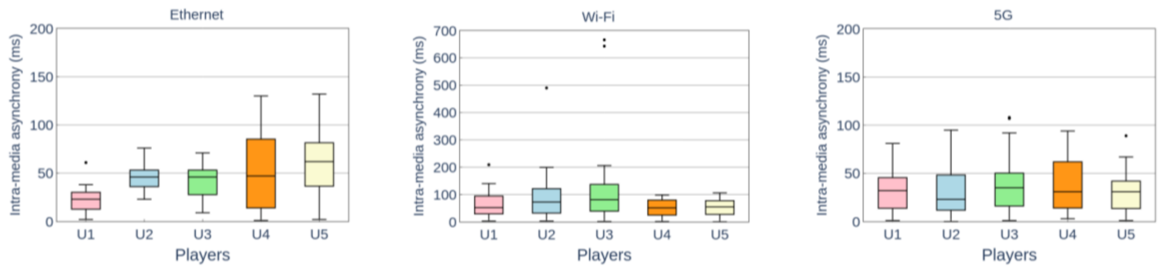


Figure 22. Intra-media synchronization.

More detailed information about the QoS measuring method and the implemented setup is described in [6].

7 SESSION MANAGEMENT AND XR MEDIA ORCHESTRATION

This section presents the XR Enablers in charge of session management and orchestration of the VR experience based on network user plane, and the AR experience based on network control plane.

7.1 HOLO-ORCHESTRATOR

7.1.1 Overview of the component

The Holo Orchestrator is an AF composed of different modules and services to allow the establishment, appropriate configuration, and lifecycle management of multi-user holographic communication sessions, as shown in Figure 23 and briefly described as follows:

- User Manager (UM): it is in charge of registering, managing and offering information / data from registered clients, scenarios and other in-cloud components, by using a MongoDB. By using the adopted holographic communication platform by i2CAT (HoloMIT[27]), users need to be logged in the platform before creating / joining a session, and then must select a virtual scenario on which the session will be established (e.g., a virtual meeting room, a museum).
- Session Manager (SM): it is in charge of managing the lifecycle of multi-user sessions (i.e., creating, joining, leaving and eliminating sessions) for each involved user/client and for each selected virtual scenario, by storing the associated information on a MongoDB. It is also in charge of interfacing the other services of the Orchestrator, like the Clock Manager and the Index/Connection Manager (both introduced next), to be able to select the most appropriate in-cloud media function(s) - VNFs - to handle the communications for each session (i.e., SFU, MCU or Remote Renderer), the in-cloud servers where to instantiate them, and then communicate this information to the involved clients.
- Clock Manager (CM): it is in charge of ensuring a coherent notion of time to all involved entities in the media session. It can act as a clock source against which to synchronize to, or it can just provide a reference to a NTP server.
- Index/Connection Manager (ConM): it is in charge of interfacing the edge orchestration platform (being developed in WP2) for selecting the most appropriate location where to deploy in-cloud VNFs for media processing and communication (i.e., SFUs, MCUs, Remote Renderers) and managing their lifecycle. This module will also be an endpoint for the envisioned Network-as-a-Service (NaaS) APIs for enhanced XR services, like: (i) Edge-Cloud APIs (details in WP2 deliverables); and (ii) Network-assisted Rate Control (details in WP4 deliverables).

[27] https://i2cat.net/holoportation-technology/

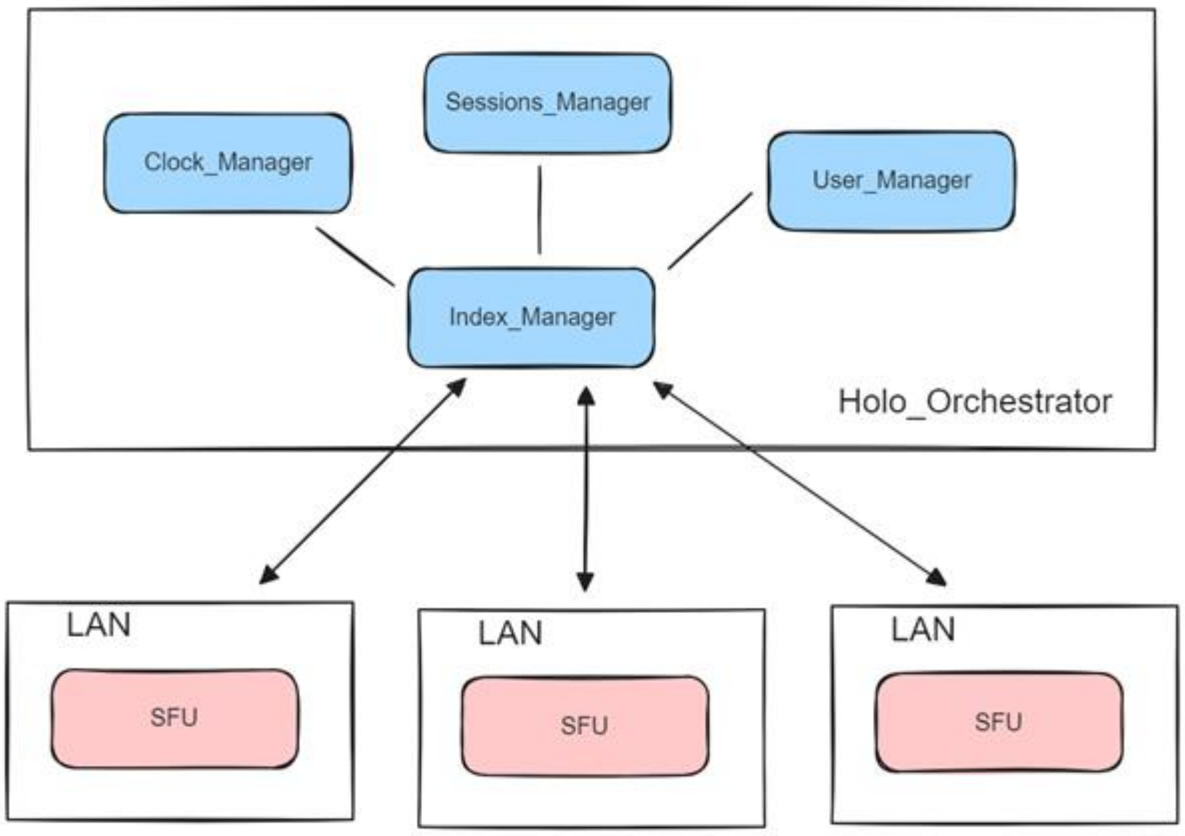


Figure 23. High-level Overview of Holo Orchestrator modules and services.

7.1.2 Requirements

The Holo Orchestrator does not have any limit regarding the number of registered users (UM) and/or sessions (SM) beyond the limits provided by MongoDB and the server on which it has been installed.

Regarding the CM, a key requirement and an associated KPI is to provide synchronization levels in the order of a few milliseconds, which is feasible when using well-known clock synchronization protocols, like NTP.

Regarding the Index Manager, no strict scalability challenges apply, as such a component does not handle the orchestration of computing resources over the cloud continuum, but just keeps track of the ones to be used and being used via a well-defined metadata model.

The UM and the SM need to be always active, ready and reachable, if holographic communication services need to be provided anytime, and from anywhere.

7.1.3 Initial hardware

The Holo Orchestrator has been tested and run on a variety of physical PCs and servers (both running Windows and Ubuntu OS), with no specific hardware requirements. It has also been successfully deployed on different Azure VMs, with the following specs: Standard DS1 v2 (1 vCPU, 3.5 GiB memory RAM); Standard DS2 v2 (2 vCPU, 7 GiB memory RAM); and Standard D4s v3 (4 vCPU, 16 GiB memory RAM.

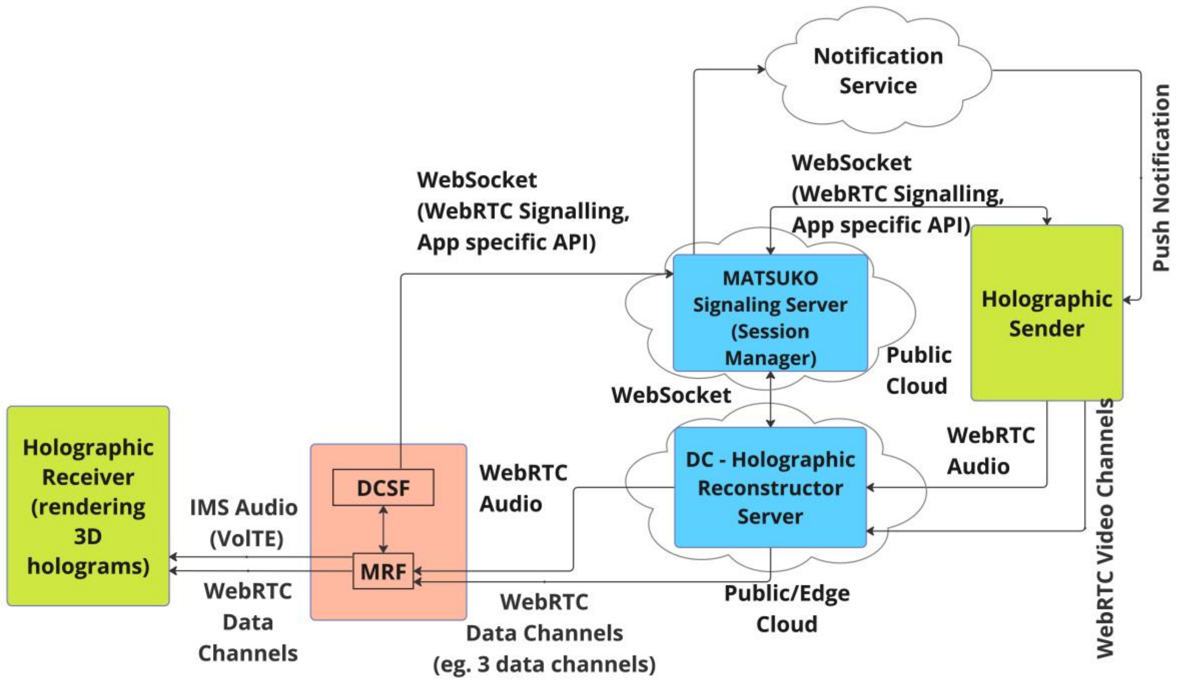
7.1.4 Initial software

The Holo Orchestrator requires the installation of Node.js[28] and socket.io[29] and it has been successfully installed and run on Windows 10 and Linux (Ubuntu 22.04) machines (including Virtual Machines on Azure). It has also been virtualized as a docker container and helm chart, which eases its deployment on any machines, including those provided by hyper-scalers.

7.2 IMS SESSION MANAGER

7.2.1 Overview of the component

Figure 24 shows the IMS session management E2E components adaptation as a mechanism to establish connectivity between MATSUKO AR application, the IMS system, and the UE. This signalling mechanism is required to establish WebRTC and IMS connections. The session management service exposes a WebSocket API for communication between the services. The MATSUKO AR application, specifically 3D reconstruction backend component, uses this API for registration. After registration, the media server becomes ready to be used for processing input data.



Notification Service
WebSocket (WebRTC Signalling, App specific API)
WebSocket (WebRTC Signalling, App specific API)
Push Notification
MATSUKO Signaling Server (Session Manager)
Public Cloud
Holographic Sender
WebSocket
Holographic Receiver (rendering 3D holograms)
IMS Audio (VolTE)
DCSF
MRF
WebRTC Audio
DC - Holographic Reconstructor Server
WebRTC Audio
WebRTC Video Channels
WebRTC Data Channels
WebRTC Data Channels (eg. 3 data channels)
Public/Edge Cloud

Figure 24. E2E MATSUKO Components Adaptation to IMS System – Architecture.

When the connection establishment between the DCSF and the MATSUKO signalling server is ready, the IMS session management provides the necessary information of the media server, i.e., the holographic reconstructor server, as depicted on Figure 24.

[28] https://nodejs.org/en

[29] https://socket.io/

The service has been extended with additional APIs to adapt to the IMS signalling. Future extension of the API is possible to handle several calls. Initial support for such multiple calls has been implemented.

7.2.2 Requirements

The IMS session management server is a lightweight component with minimum system requirements. As its function is to establish the signalling and manage the session using small WebSocket messages during the initiation of the call, its network requirements are minimal (10kbps).

7.2.3 Initial hardware

The MATSUKO application with the IMS session management is deployed on Azure VMs using a distributed containerised approach. The VM which hosts the IMS session management runs on Ubuntu 18.04.6 LTS with 2 vCPUs and 4 GB RAM. The VM which hosts the 3D reconstruction application runs on Ubuntu 18.04.6 LTS using NVIDIA T4 GPU with 4 vCPUs and 28 GB RAM.

For E2E testing, an iPhone device that supports 3D sensors and FaceID is used for capturing (e.g., iPhone 12 Pro, iPhone 13, iPhone 14 Pro). For receiving and viewing the content, any device supporting and implementing IMSDC client can be used (in this case, Samsung Galaxy S22/S23 families).

7.2.4 Initial software

On the server side, the MATSUKO application server components are containerised microservice applications running as Docker containers in a Kubernetes cluster on Azure. On the UE side, the MATSUKO iOS application is installed on the iPhone, which captures the data. On the rendering side, the WebGL application is used and is downloaded and executed in the dialler of the browser during an IMS call session.

8 INFRASTRUCTURE CONFIGURATION

This section presents the infrastructure enablers to allow the configuration of the computing and network infrastructure such to host the XR Enablers and provide them with the necessary resources. These enables also allow dynamic modifications of the allocated resources such as to adapt the infrastructure to the XR Enablers requirements at any given time.

8.1 XR APPLICATION TRAFFIC REQUIREMENTS EXTRACTION

8.1.1 Overview of the component

XR applications can have dynamic requirements and it is important to understand them in order to provide an appropriate network configuration and guarantee QoS and determinism, e.g., when employing Time-Sensitive Networking (TSN) to protect time-critical XR workloads. At the same time, it is important to understand these requirements without the need to modify the applications.

A software solution, referred to as “traffic profiler”, is developed to perform two main tasks transparently to the XR applications: a) extract traffic requirements and b) use this requirement to configure the packet scheduling in a TSN setup.

8.1.2 Requirements

This component helps to achieve network QoS KPIs on dynamic workloads such as latency upper-bound constraints for XR applications in the range of a few milliseconds.

The main requirement for this component is the deployment of a TSN-enabled network, that implements the 802.1AS and 802.1Qbv standards. A Linux machine is necessary for the component to run. Most importantly, applications do not need to be modified.

8.1.3 Initial software

This traffic profiler consists of three main parts (Figure 25):

1) A profiler process that analyses the application traffic such as cycle periods, jitter, or packet sizes. The profiled data is then the basis to configure the network for optimized QoS.

2) A scheduling component which determines a time-aware schedule to configure TSN devices.

3) A monitoring component, to close the feedback loop and trigger the further profiling or re-scheduling in case the achieved KPIs are insufficient.

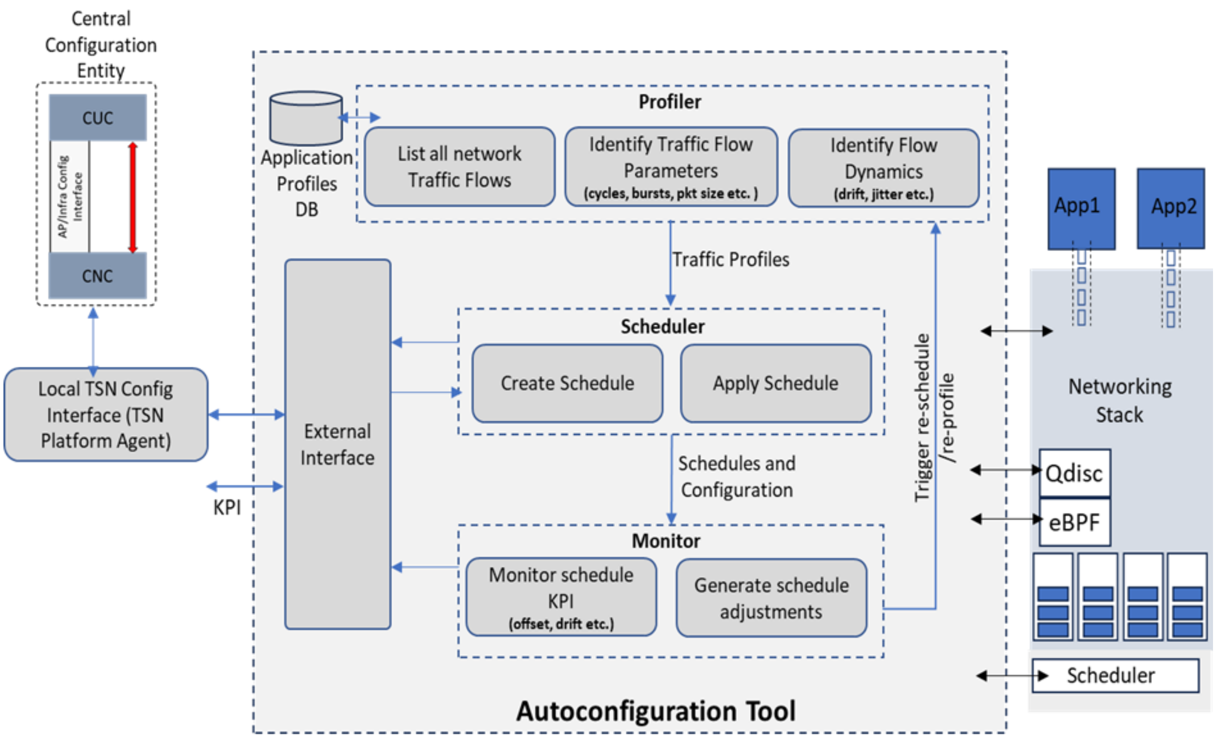

Central Configuration Entity
CUC
AP/Infra Config Interface
CNC
Local TSN Config Interface (TSN Platform Agent)
KPI
Application Profiles DB
Profiler
List all network Traffic Flows
Identify Traffic Flow Parameters (cycles, bursts, pkt size etc.)
Identify Flow Dynamics (drift, jitter etc.)
Traffic Profiles
External Interface
Scheduler
Create Schedule
Apply Schedule
Schedules and Configuration
Monitor
Monitor schedule KPI (offset, drift etc.)
Generate schedule adjustments
Trigger re-schedule /re-profile
Autoconfiguration Tool
App1
App2
Networking Stack
Qdisc
eBPF
Scheduler

Figure 25. Overview of the traffic profiler: 1) Profiling of application, 2) Configuration of time-aware schedule, 3) monitoring of KPIs to trigger further profiling.

A study on the effectiveness of this component on a productivity application in XR is being conducted. The results and gathered data can be made available to understand the potential benefits of the enabler under development.

8.2 SCALABILITY ENABLER

8.2.1 Overview of the component

This component will implement scaling in and scaling out actions from the point of view of computational resources for each XR application, driven by the agreed QoE level for each end-user.

This involves the implementation of a set of policies and controlling mechanisms that will trigger scaling in/out actions of efficient media functions over the compute infrastructure, that will feed the NorthBound Interface (NBI) of the edge orchestrators.

This means that, based on the monitored resource utilization (CPU, RAM) by each application, this component will be able to trigger, when needed, the order to allocate more resources (CPU, RAM), this is scaling out mechanisms, or, on the contrary, to release resources in case the application is behaving well from the QoE point of view.

Besides, it is planned that this component will support some prediction capabilities by means of the implementation of Machine Learning (ML) models, which will anticipate when those CPU scaling in/out section will be needed based on previously “learned” experience. Neural Networks will be trained with actual CPU usage data for this purpose.

The required hardware to implement such component is yet to be defined.

8.2.2 Requirements

The scalability enabler needs to be aware of the following monitored KPIs in order to perform its operations:

- Compute resources utilization by VNF/app (in terms of number of cores, RAM, GPU or storage)
- Edge node locations and their availability (in terms of number of cores, RAM, GPU or storage)
- Monitoring events related to app management
- QoE degradation
- Regarding the predictive operations, at least a data set with previous CPU usage by similar VNF/app and user will be needed.

8.2.3 Initial software

The following software tools are envisioned for the implementation at the different steps:

1. Model development, using open-source tools such as TensorFlow or Keras.
2. Model deployment and serving, with open-source tool such as TensorFlow Serving.

8.3 EDGE CONTINUUM ENABLER

8.3.1 Overview of the component

This component enables the migration or re-allocation of the XR applications considering a distributed deployment across the edge continuum, including multiple edge domains. This is done considering the specific XR application requirements in order to guarantee the required QoE, with special focus on latency reduction.

The edge continuum enabler would be consuming the edge infrastructure NBI API that provides the app developer with the possibility of deploying the app in the optimal cloudlet in the edge continuum, to help achieve the minimum possible E2E latency. This will be done at the network level by activating and configuring the proper routing in the mobile network and in this way influencing the traffic routing from the user device towards the edge instance of the application (see D2.1 [4] for further details at network level).

Therefore, the enabler will interact with the mechanisms below, this is with the North Bound Interface of the Edge Orchestrator (see details on 6G-XR architecture in D1.2 [7]), in order to be able to re-allocate the application to a new edge cloudlet to find the minimum routing path that reduces the delay.

Besides, it is planned that this component will be incorporating to some extent prediction capabilities by means of the implementation of ML models that will anticipate when those migration actions need to be triggered based on previously "learned" experience.

The required hardware to implement such component is yet to be defined.

8.3.2 Requirements

This component needs to be aware of the following monitored KPIs to perform its operations:

- VNF/app instance edge localization
- Compute resources utilization by VNF/app (in terms of number of cores, RAM, GPU or storage)
- Edge node locations and their resource availability (in terms of number of cores, RAM, GPU or storage)
- Monitoring events related to app management
- QoE degradation, especially related to latency
- Regarding the predictive operations, at least a data set with historical latency values and traffic congestion detection

8.3.3 Initial software

The following software tools are envisioned for the implementation at the different steps:

1. Model development, with open-source tools such as TensorFlow or Keras.
2. Model deployment and serving, with open-source tool such as TensorFlow Serving.

9 KPI AND TELEMETRY

This section presents the first version of the monitoring system being developed to store, visualize and exploit KPIs collected from the XR Enablers.

9.1 MONITORING SYSTEM

9.1.1 Overview of the component

This section reports on the progress made for collecting and displaying metrics from key components of the developed XR Enablers, including the Unity players (both for the production and consumption interfaces), SFU, MCU, and Orchestrator.

The system has been designed and implemented by using widely adopted components for such purposes, like Prometheus[30] and Grafana[31]. Its high-level architecture is sketched in Figure 26, and its main components are briefly introduced next:

- HoloMIT SDK for Unity: It is the component providing relevant metrics for the native XR player, both at the production and consumption sides, from different sub-components of the E2E pipeline.
- Prometheus Push Gateway: It enables the connection between the HoloMIT SDK for Unity and Prometheus for the reception and storage of metrics.
- Prometheus: It is a toolset for alerting and monitoring, gathering metrics and storing them in a time-series database from several sources, including the Push Gateway. Prometheus further offers querying tools for retrieving and examining metrics data.
- Grafana: It is a platform for analytics and visualization. It establishes a data source connection with Prometheus and pulls metrics data for display. Grafana gives the ability to design dashboards for data analysis, tracking system performance, and visualizing metrics trends. Grafana's connection to Prometheus as a data source must be appropriately configured, potentially requiring the intervention of extra components. It may entail creating panels, alerts, and dashboards based on metrics data obtained from Prometheus.

[30] https://prometheus.io/

[31] https://grafana.com/

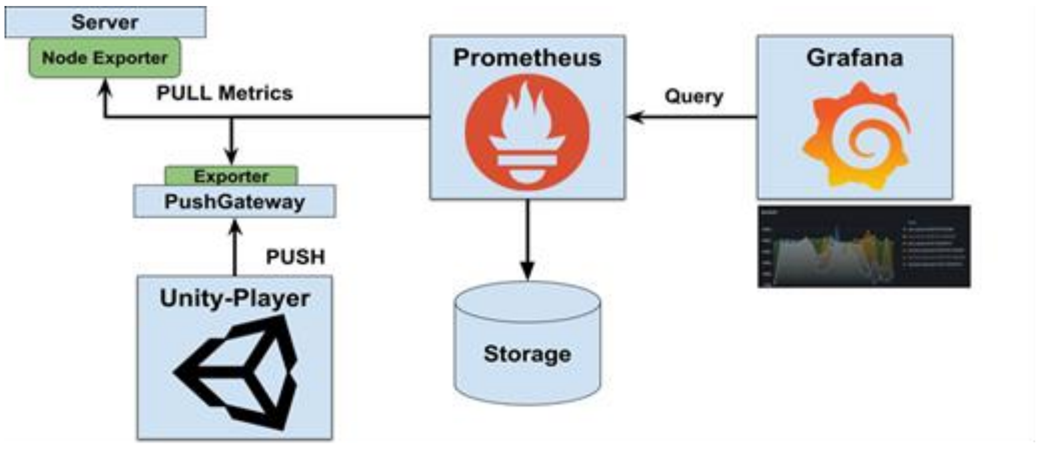


Figure 26. High-level architecture of the metrics measurement and registration system.

The overall workflow for metrics registration and visualization is summarized as follows:

- Metrics from the Unity application/player, SFU and/or MCU are collected.
- The data for the configured metrics are pushed to the Prometheus Push Gateway.
- Prometheus periodically pulls metrics data from the Push Gateway and stores them in its time-series database.
- Grafana connects to Prometheus as a data source and retrieves metrics data.
- Users access Grafana to visualize metrics data through custom dashboards and panels.

On the server side, other components may retrieve metrics data from Prometheus for further analysis or integration with other (sub-)systems.

9.1.2 Key Performance Indicators

This section reports on a selection of metrics measured/reported from/for each XR Enabler.

<u>Native Player</u>:

Table 16. KPIs of the Native Player component.

Metrics	Description
Resources Usage Metrics	
Sent / Received Bandwidth (Mbps)	Total traffic sent / received by the native player (Mbps)
CPU Usage (%)	Percentage of CPU resources used by the Remote Renderer
RAM Usage (MB)	RAM resources used by the Remote Renderer
GPU Usage (%)	Percentage of GPU resources used by the Remote Renderer
Point Cloud Encoding / Transmission	
Frames per second (fps)	Number of frames per second that are encoded and transmitted
Points per Cloud (#)	Number of points per each Point Cloud frame

Average Point Size (#)	Average size of each Point within a Point Cloud frame
Encoding latency (ms)	Latency of the encoding process (ms)
Point Cloud Decoding / Reception	
Frames per second (fps)	Number of frames per second that are received and decoded
Points per Cloud (#)	Number of points per each Point Cloud frame
Average Point Size (#)	Average size of each Point within a Point Cloud frame
Decoding latency (ms)	Latency of the decoding process (ms)
E2E latency (ms)	Latency of the end-to-end pipeline

Selective Forwarding Unit:

Table 17. KPIs of the SFU component.

Metrics	Description
CPU Usage (%)	Percentage of CPU resources used by the SFU
Input bandwidth (Mbps)	Amount of traffic received by the SFU
Output bandwidth (Mbps)	Amount of traffic forwarded by the SFU
Uplink delay / frame (ms)	Delay from the originating clients to the SFU for each incoming frame

Multipoint Control Unit:

Table 18. KPIs of the MCU component.

Metrics	Description
Incoming frame latency (ms)	Latency for each incoming frame to the MCU
Input fps (#)	Number of frames / second received by the MCU
Frame decoding time (ms)	Latency to decode each incoming frame
Frame decoding rate (#)	Frames that are decoded per time interval
Frame fusion size (MB)	Size of each fused frame
Frame fusion latency (ms)	Latency to fuse frames from each player
Frame fusion fps (#)	Number of frames / second received by the MCU per time interval (s)
Frame encoding time (ms)	Latency to encode each fused frame
Fusion MCU latency (ms)	Latency of the MCU fusion process
Output fps (#)	Number of frames / second delivered by the MCU to each player

Remote Renderer:

Table 19. KPIs of the Remote Renderer component.

Metrics	Description
CPU Usage (%)	Percentage of CPU resources used by the Remote Renderer
RAM Usage (MB)	RAM resources used by the Remote Renderer
GPU Usage (%)	Percentage of GPU resources used by the Remote Renderer

WebRTC Player:

Table 20. KPIs for WebRTC player.

Flow	Metric	Description
Audio	timestamp (ms)	Current timestamp
	jitter (ms)	RTP packet jitter for this media flow
	packetsLost / second (packets/s)	Total number of RTP packets lost per second for this media flow
	packetsReceived / second (packets/s)	Total number of RTP packets received per second for this media flow, it includes retransmissions
	bytesReceived / second (bytes/s)	Total number of bytes received per second for this media flow, it includes retransmissions
	estimatedPlayoutTimestamp (ms)	This is the estimated playout time of this track at the receiver device
Video	timestamp (ms)	Current timestamp
	jitter (ms)	RTP packet jitter for this media flow
	packetLost / second (packets/s)	Total number of RTP packets lost per second for this media flow
	packetsReceived / second (packets/s)	Total number of RTP packets received per second for this media flow, it includes retransmissions
	bytesReceived / second (bytes/s)	Total number of bytes received per second for this media flow, it includes retransmissions
	estimatedPlayoutTimestamp (ms)	This is the estimated playout time of this track at the receiver device
	framesReceived / second (fps)	Represents the number of full frames received per second
	frameHeight (pixels)	Represents the height of the last received frame
	frameWidth (pixels)	Represents the width of the last received frame
	framesDecoded / second (fps)	Represents the total number of frames successfully decoded

DASH Player:

Table 21. KPIs for DASH player.

Metric	Description
width (pixels)	The width of the current video track
height (pixels)	The height of the current video track
streamBandwidth (bps)	The bandwidth required for the current video stream
estimatedBandwidth (bps)	The current estimated network bandwidth
playTime (ms)	The total playback time
bufferingTime (ms)	The total time spent in a buffering state
liveLatency (ms)	The time between capturing a frame and displaying it on the screen of the end device

Holo Orchestrator:

Table 22. KPIs for the Holo Orchestrator component.

Metric	Description
Active Sessions (#)	Number of active holoconferencing sessions managed by the Orchestrator
Number of users / session (#)	Number of active users for each running session
Types of users / session	Arrays with the types of representation formats of each user for each session
Active SFUs / session (#)	Number of active SFUs for each running session
Active SFUs / session (#)	Number of active MCUs for each running session

Scalability enabler:

Table 23. KPIs for Scalability enabler

Metrics	Description
CPU Usage (%)	Percentage of CPU resources used by each application
RAM Usage (MB)	RAM resources used by the application
GPU Usage (%)	Percentage of GPU resources used by the application
Available CPU	Percentage of CPU resources available in the edge infrastructure
Available RAM (MB)	Amount of RAM available in the edge infrastructure
Available GPU Usage (%)	Percentage of GPU resources available in the edge infrastructure

Edge Continuum enabler:

Table 24. KPIs for the Edge Continuum enabler

Metrics	Description
CPU Usage (%)	Percentage of CPU resources used by each application
RAM Usage (MB)	RAM resources used by the application

GPU Usage (%)	Percentage of GPU resources used by the application
Available CPU	Percentage of CPU resources available in the edge infrastructure
Available RAM (MB)	Amount of RAM available in the edge infrastructure
Available GPU Usage (%)	Percentage of GPU resources available in the edge infrastructure
Latency (ms)	Average latency in the network

As a proof of concept, Figure 27 shows some screenshots captured by the Grafana dashboard reporting on key metrics from the native Unity player and the SFU (co-located in the same machine of the Holo Orchestrator in this case).

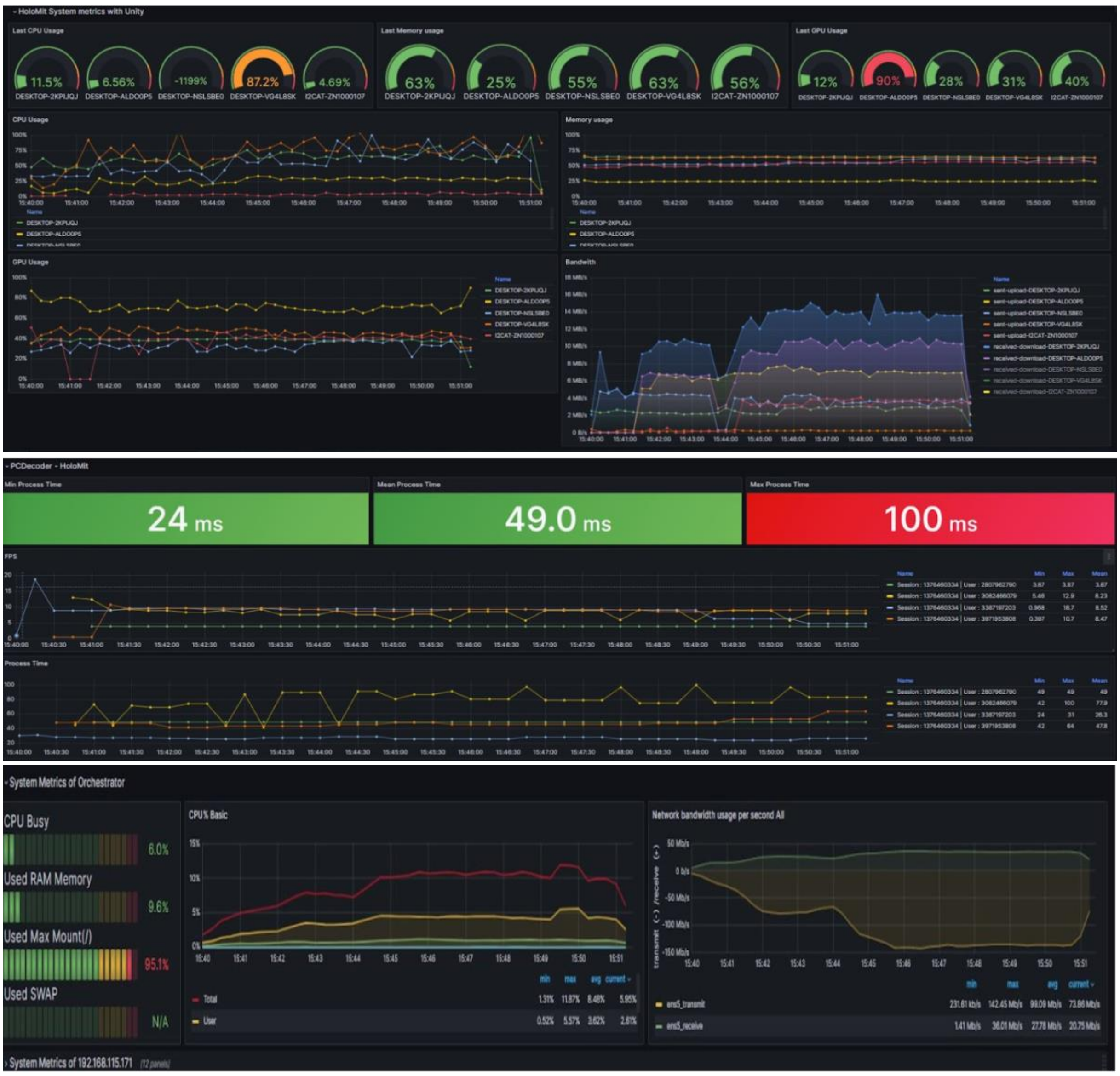


Figure 27. Grafana dashboards showing collected metrics from XR Enablers.

9.1.3 Initial hardware

This monitoring system does not impose strict computing (CPU or GPU) or memory (RAM) requirements for installation and execution. However, the monitored metrics require storage space to allocate the Prometheus database with the metrics. Storage capacity of at least a few tens of GBs is required.

9.1.4 Initial software

This monitoring system has been installed and can run on different Windows and Ubuntu machines, including physical ones and deployments on Azure. For the main software requirements and dependencies, please refer to the documentation from Prometheus[32] and Grafana[33].

9.1.5 Dataset exporters

At the time of writing this report, the monitoring system can measure, report, register and visualize the metrics in near real-time. The next deliverable D3.2 – "Final versions of XR enablers" will report on new modules to be able to export the metrics to appropriate databases or structured textual files to be later used for the creation of datasets.

[32] https://prometheus.io/

[33] https://grafana.com/

10 SUMMARY

This report presents the XR Enablers that are currently under development in WP3 of 6G-XR. The objective of these XR Enablers is to enable the necessary E2E media pipeline for the deployment of three WP6 UCs:

- UC1 - Resolution Adaptation or Quality on Demand
- UC2 - Routing to the Best Edge
- UC3 - Control Plane Optimization

The UC1 and UC2 use the XR enablers integrated with the network user plane for VR experiences. The UC3 evolves the IMS system as a component of the network control plane for AR experiences. To achieve these objectives, the following capabilities are covered by the presented XR Enablers:

- Volumetric capture sensors and reconstruction.
- Cloud/edge-enabled multimedia processing for scalability and wider user's access.
- Multiprotocol multimedia delivery across heterogeneous end devices.
- Clock and media synchronization capabilities.
- Multimedia session management and orchestration.
- Infrastructure configuration enablers to optimize resources allocation for the multimedia processing.
- Monitoring system for KPIs related with multimedia processing.

An initial release (first version) of the XR Enablers functions has been generated and described in this document, including the hardware and software employed for their development.

These XR Enablers will be developed further during the remaining months of the project and, finally, integrated within the computing infrastructure of the 6G-XR South Node test facilities, namely 5Tonic (Madrid, Spain) and 5GBarcelona (Barcelona, Spain), where the three WP6 UCs will be tested.

11 REFERENCES

[1] 6G-XR, "Requirements and use case specifications," Deliverable D1.1, September 2023.

[2] Guo, Kaiwen, et al. "The relightables: Volumetric performance capture of humans with realistic relighting." ACM Transactions on Graphics (ToG) 38.6 (2019): 1-19.

[3] Langa, Sergi Fernández, et al. "Multiparty holomeetings: Toward a new era of low-cost volumetric holographic meetings in virtual reality." IEEE Access 10 (2022): 81856-81876.

[4] 6G-XR, "Orchestration, AI techniques, End-to-end slicing and Signalling for the core enablers – design," Deliverable D2.1, February 2024.

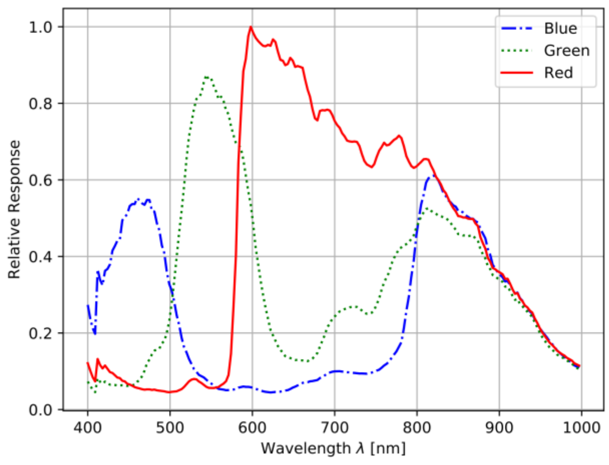
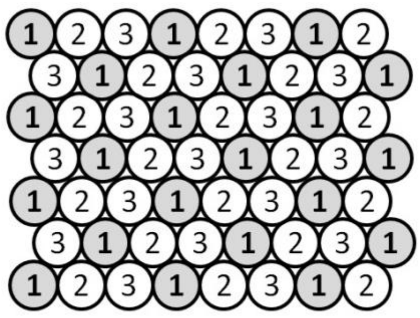
[5] Fernandez, Sergi, et al. "Addressing Scalability for Real-time Multiuser Holo-portation: Introducing and Assessing a Multipoint Control Unit (MCU) for Volumetric Video." Proceedings of the 31st ACM International Conference on Multimedia. 2023.

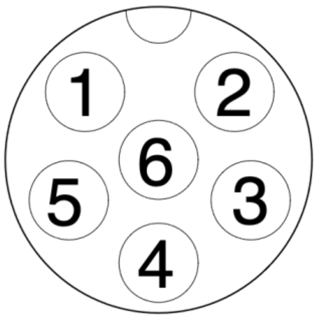
[6] Yeregui, Inhar, et al. "Edge Rendering Architecture for multiuser XR Experiences and E2E Performance Assessment." 2024 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB). IEEE, 2024.

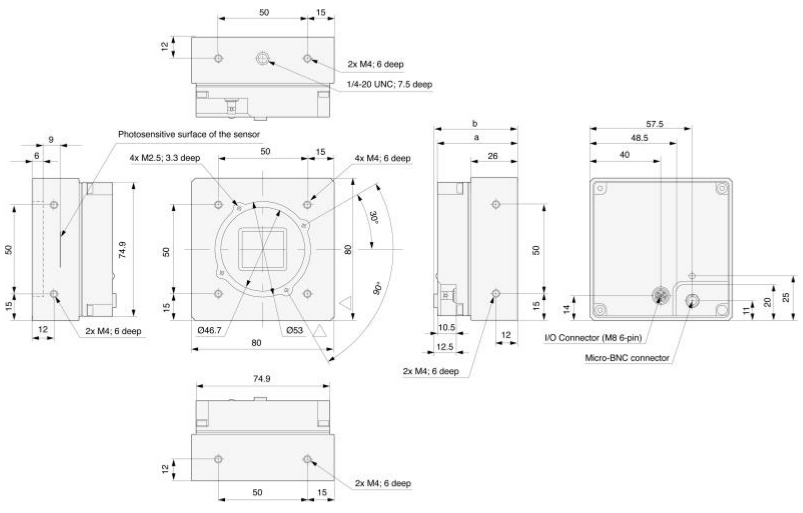
[7] 6G-XR, "Reference architecture description," Deliverable D1.2, June 2024.

APPENDIX A - R32 LIGHT-FIELD CAMERA FACTSHEET

R32 factsheet	
Sensor	
Image sensor	Onsemi XGS 32000
Lateral resolution (H x V)	6560 x 4948 pixel^2
Lateral resolution (MegaPixel)	32.4 MP
Effective lat. resolution (H x V)	3280 x 2474 pixel^2
Effective lat. resolution (MegaPixel)	8.1 MP
Active area	21.0 x 15.8 mm^2
Pixel length	3.2 µm
Shutter type	Global shutter
Frame rate	36 fps
ADC resolution	12 bits
Spectrum	Colour

Spectral response	
Cover glass removed?	Yes
Sensor interface	HiSPi
Micro lens array	
MLA type	L3-D125-A018-VRE-VI
Light-field mode	Galilean multi focused plenoptic 2.0
Micro lens types	3
Geometry	hexagonal
Layout	
Aperture	f/1.8
Lens pitch	125 µm
Camera interface	CoaXPress (2.5–12.5 Gbps); Micro-BNC (HD-BNC) connector

Camera interface bandwidth	CXP Speed	Bandwidth	Cable length	fps @ full res.
	CXP-2	2.5 Gbps	180m	7
	CXP-3	3.125 Gbps	100m	9
	CXP-5	5 Gbps	60m	14
	CXP-6	6.25 Gbps	40m	18
	CXP-10	10 Gbps	40m	29
	CXP-12	12.5 Gbps	30m	36
Power	**Recommended**: Power over CoaXPress (PoCXP): 24 VDC supplied via the camera's Micro-BNC (HD-BNC) connector. 11 W (typical) **Not Recommended**: Power supply via I/O connector: operating voltage 24 VDC. Minimum 18.6 VDC. Maximum 26 VDC.			
I/O	M8 6-pin female connector (IEC 61076-2-104) Recommended mating connector: M8 6-pin male			
Pinout	Pin	Line	Function	
	1	-	24 VDC power	
	2	Line 1	Opto-coupled I/O input	
	3	-	Ground for opto-coupled I/O	
	4	Line 2	General purpose I/O (GPIO)	
	5	Line 3	General purpose I/O (GPIO)	
	6	-	Ground for camera power and General Purpose I/O (GPIO)	
Size (L x W x H)	50 x 80 x 80 mm^3			

	 2x M4; 6 deep 1/4-20 UNC; 7.5 deep Photosensitive surface of the sensor 4x M2.5; 3.3 deep 4x M4; 6 deep Ø46.7 Ø53 2x M4; 6 deep I/O Connector (M8 6-pin) Micro-BNC connector 2x M4; 6 deep 2x M4; 6 deep
Weight	550 g
Mount	Custom, thermal decoupling of lens and camera body
OEM	Basler Lens
Type	Based on F-S35-2528-45M-S-SD
Focal length	24.5 mm
Aperture*	f/1.8
Focus range*	0.2 - 3.5m
Angle of View (on R32)	Horizontal: 55.7° Vertical: 39.1°