D3.2: Final 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	31/07/2025
Submission date	25/07/2025
Deliverable lead	VICOM
Version	1.0
Authors	Roberto Viola (VICOM), Inhar Yeregui (VICOM), Daniel Mejías (VICOM), Arne Erdmann (Raytrix), Andreas Pinnow (Raytrix), Stefano Spyropoulos (Raytrix), Hossameldien Abdalaleem (Raytrix), Tim Holtorf (Raytrix), Mario Montagud (i2CAT), Isaac Fraile (i2CAT), Antonio Calvo (i2CAT), Marc Martos (i2CAT), Genís Castillo (i2CAT), Chathura Sarathchandra (IDE), Matus Kirchmayer (MATSUKO), Aurora Ramos (CGE), Enrique Lluesma (CGE), Javier Godas (CGE), Ernesto Correa (CGE), Rafael Rosales (INTEL), Hamza Chahed (INTEL), Fernando Pargas (TID)
Reviewers	Valerio Frascolla (INT), Hemalatha Vulchi (IT), Mohammed Al-Rawi (IT)
Abstract	The 6G-XR project is focused on developing an infrastructure to support eXtended Reality (XR) services, addressing key use cases in Augmented Reality (AR) through the exploitation of an advanced network control plane and in Virtual Reality (VR) through the utilisation of an advanced network user plane. This document presents the final versions of the XR Enablers, i.e., a set of functions designed to enable a complete end-to-end multimedia pipeline for AR/VR experiences. These XR Enablers incorporate capabilities such as multi-sensor volumetric capture and reconstruction, XR processing across the cloud-edge continuum, adaptive and low-latency content delivery, multi-modal synchronisation, session management, media orchestration, and performance monitoring through a Key Performance Indicator (KPI)-driven system. These enablers will be integrated in the use cases selected by 6G-XR, supporting experimental trials and enabling the evaluation of relevant system KPIs.
Keywords	5G/6G, Augmented Reality (AR), Virtual Reality (VR), Holographic Communications, User Plane, Control Plane, Multimedia Functions, Media Synchronisation.

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	27/02/2025	Table of contents	Roberto Viola (VICOM)
V0.2	20/06/2025	First draft	All partners (see authors)
V0.3	26/06/2025	Version with comments from external reviewers	Valerio Frascolla (INTEL), Hemalatha Vulchi (IT)
V0.4	03/07/2025	Consolidated version after addressing comments from external reviewers	All partners (see authors)
V0.5	14/07/2025	Version with comments from technical manager	Mohammed Al-Rawi (IT)
V1.0	24/07/2025	Final version	All partners (see authors)

DISCLAIMER

Co-funded by the European Union

6GSNS

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)**.

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

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 document constitutes the final deliverable (D3.2) of Work Package 3 (WP3) – "XR Enablers" within the 6G-XR project. The objective of D3.2 is to explain the latest iterations in the development and deployment of the Virtual Network Functions (VNFs) for eXtended Reality (XR), envisioned within the overall WP3 activities.

These VNFs are intended to be fundamental components of both the network user plane and the control plane of a communication system, facilitating real-time holographic communications, encompassing Virtual Reality (VR) and Augmented Reality (AR). Consequently, they are essential for the implementation of three of the five use cases (UCs) planned within the 6G-XR project (refer to D1.1 [1] for additional information):

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

The integration of these VNFs into the aforementioned three UCs is executed at the 6G-XR South Node test facilities, specifically 5TONIC (Madrid, Spain) and i2CAT (Barcelona, Spain).

D3.2 provides a detailed description of the final versions of the XR Enablers, including the hardware and software resources utilised for their development, as well as a comparison with their first versions described in D3.1 [2].

The XR Enablers encompass functionalities such as multi-sensor volumetric capture and reconstruction, cloud/edge XR processing, adaptive and low-latency XR delivery, multi-modal synchronisation, session management, and media orchestration. Enablers pertaining to infrastructure configuration and a KPI monitoring system are also included in this final report to furnish a complete overview of the integration of the XR Enablers and their interoperability with the communication infrastructure.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

2D	2-dimensional
3D	3-dimensional
5G	Fifth Generation
6DoF	6 Degrees of Freedom
AF	Application Function
API	Application Programming Interface
AR	Augmented Reality
ATSSS	Access Traffic Steering-Switching-Splitting
AS	Application Server
BE	Best Effort
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
DASH IF	DASH Industry Forum
DCSF	Data Channel Signalling Function
E2E	End-to-End
EC	European Commission
EU	European Union
FoV	Field of View
fps	Frames Per Second
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ICE	Interactive Connectivity Establishment
IEAP	Intelligent Edge Application Platform
IMS	IP Multimedia Subsystem
IMS-AS	IMS Application Server
IMSDC	IMS Data Channel
IP	Internet Protocol
IPU	Image Processing Unit
KPI	Key Performance Indicator
LF-SDK	Light-Field SDK
LoD	Level of Detail
LoS	Level of Service
LSTM	Long-Short Term Memory
LTS	Long Term Support
MCU	Multipoint Control Unit
ML	Machine Learning
MLA	Microlens Array
MLOPs	ML Operations
MNO	Mobile Network Operator
MoQ	Media over QUIC
MPD	Media Presentation Description
MU	Media Unit

NaaS	Network-as-a-Service
NIC	Network Interface Controller
NTP	Network Time Protocol
OS	Operating System
PC	Personal Computer
QoD	Quality on Demand
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
REST	Representational State Transfer
RGBD	Red Green Blue Depth
RoI	Region of Interest
RoQ	RTP over QUIC
RTCP	Real-time Transport Control Protocol
RTP	Real-time Transport Protocol
SDK	Software Development Kit
SDP	Session Description Protocol
SFU	Selective Forward Unit
SM	Session Manager
SNS	Smart Networks and Services
SRTP	Secure RTP
TCP	Transmission Control Protocol
TSN	Time-Sensitive Networking
UC	Use Case
UDP	User Datagram Protocol
UE	User Equipment
UM	User Manager
UP	User Plane
UPF	User Plane Function
UTC	Universal Time Coordinated
vCPU	virtual CPU
VM	Virtual Machine
VNF	Virtual Network Function
VR	Virtual Reality
WebRTC	Web Real-Time Communication
WP	Work Package
WTSN	Wireless TSN
XR	eXtended Reality

1 INTRODUCTION

The main purpose of D3.2 is to summarise the outcomes of Work Package 3 (WP3) "XR Enablers" of the "6G eXperimental Research infrastructure to enable next-generation XR services" (6G-XR) project. The overarching objective of WP3 is the creation of enablers for XR multimedia processing, encompassing both AR and VR technologies. Consequently, each task within WP3 is dedicated to the implementation of specific enablers or multimedia functionalities, addressing various dimensions of the End-to-End (E2E) media pipeline as follows:

- Task 3.1 (T3.1) concentrates on volumetric capture sensors and the associated reconstruction processes.
- Task 3.2 (T3.2) capitalises on edge computing resources to facilitate the scalability of multimedia processing capabilities.
- Task 3.3 (T3.3) employs heterogeneous communication protocols to deliver multimedia content to users employing devices with diverse processing and interaction capabilities.
- Task 3.4 (T3.4) provides the necessary clock and media synchronisation capabilities to ensure a consistent and coherent experience for users participating in the media communications.
- Task 3.5 (T3.5) focuses on the orchestration of media sessions.
- Task 3.6 (T3.6) enables the collection of KPIs relevant to multimedia processing, intended for subsequent evaluation or real-time system adjustments.

This document outlines the final implementation of the XR Enablers, representing the multimedia functions within the E2E media pipeline. The XR Enablers, described in a preliminary implementation in D3.1 published in project month 18 (M18), have been further refined and developed throughout the duration of WP3 (M19-M31) and, finally, have been deployed and integrated with the computing infrastructures provided by WP2 "Networking and Computing Enablers".

Furthermore, the solutions developed within WP3 are instrumental in demonstrating three UCs pertaining to the VR user plane (UC1 and UC2, detailed in D1.1 [1]) and the AR control plane (UC3, detailed in D1.1 [1]). These UCs are validated within WP6 "Validation of Holographic and 3D Digital Twin Use cases".

1.1 OBJECTIVES OF THE DELIVERABLE

The objectives of D3.2 are to:

- Describe the user plane and data plane communications that have been developed and integrated, to be employed for demonstrating three WP6 UCs (UC1 and UC2 focus on the VR user plane, while UC3 focuses on the AR control plane).
- Detail the progress and describe the final versions of the developed multimedia functions, i.e., the XR Enablers, essential for the E2E media pipeline. For each XR Enabler, a general description of the component is provided, along with relevant information on the hardware and software utilised for its development, where applicable. A comparison with the previous release described in D3.1 is also provided.

- Present the components responsible for delivering synchronisation capabilities across the participants of a media session, as well as the components designed for orchestrating the various functions within the E2E media pipeline.

- Describe the infrastructure configuration enablers and the monitoring system employed by the XR Enablers. This includes how network resources are configured and how KPIs are collected from the different multimedia functions of the E2E media pipeline.

1.2 STRUCTURE OF THE DELIVERABLE

D3.2 is structured as follows:

- Chapter 2 presents the implemented E2E communications to support the demonstration of three WP6 UCs.

- Chapter 3 details the multi-camera capture system for generating volumetric video streams. Additionally, it describes the Edge-supported video reconstruction solution, which aids in enhancing the quality of the volumetric video through the fusion of multiple input streams.

- Chapter 4 outlines the components intended for deployment on the cloud/edge infrastructure to achieve scalability and broader accessibility for XR services. These components encompass various media processing functionalities.

- Chapter 5 describes the streaming protocols and video players utilised for delivering media to the end user's device. The device may employ a native player with capabilities for volumetric capture, 6 Degrees of Freedom (6DoF) interaction, and content consumption, or a streamlined web-based player where volumetric capture functionality is not available.

- Chapter 6 presents the activities on clock and media synchronisation mechanisms, meant for ensuring a synchronised XR experience among participants and to facilitate the monitoring of multimedia performance metrics.

- Chapter 7 details the Holo Orchestrator, designed for the VR user plane and the IP Multimedia Subsystem (IMS) session manager intended for the AR control plane.

- Chapter 8 describes the infrastructure configuration mechanisms employed for allocating infrastructure resources and/or selecting the appropriate ones to execute the multimedia processing components.

- Chapter 9 reports on the monitoring system utilised to evaluate performance metrics across the E2E media pipeline.

- Finally, Chapter 10 concludes the document with a comprehensive summary of this report.

1.3 TARGET AUDIENCE OF THE DELIVERABLE

This deliverable is a public report that targets the project consortium and its stakeholders, i.e., academic and research organisations, industries active in the field of focus of 6G-XR, the EU commission services, and the general public.

2 END-TO-END DIAGRAM OF COMMUNICATIONS

XR services are typically developed by strategically chaining together building blocks or Application Functions (AFs) along an E2E multimedia pipeline, spanning from content capture to presentation on the end device. Within Mobile Network Operator (MNO) networks, these XR AFs can be integrated into either the Control Plane (CP), where they manage session control and coordination, or the User Plane (UP), where they handle media processing, communication, and content delivery.

The 6G-XR project, through its WP3, provides a set of XR Enablers that operate across both the UP and CP domains. These enablers are deployed over a distributed cloud-to-edge continuum infrastructure, as outlined in WP2, and leverage resources from the Radio Access Network (RAN) and core network, as described in WP4. The architecture and role of the VR user plane and the AR control plane are detailed in the following subsections.

2.1 VR USER PLANE

Figure 1 provides a high-level overview of an E2E multimedia pipeline designed to support real-time multiuser holographic communication services in shared VR scenarios. This pipeline integrates a sequence of XR Enablers responsible for media processing in the UP. The key XR Enablers are briefly introduced as follows:

- **Volumetric Capture and Reconstruction**: UP AFs deployed at the User Equipment (UE) with optional support of edge/cloud processing. Their role is to integrate volumetric capture sensors and reconstruction capabilities. XR systems commonly utilise real-time capture sensors—either single or multiple—which often support dynamic adjustment of resolution and granularity, even during an active session.
- **Selective Forwarding Unit (SFU)**: an UP AF deployed as a cloud or edge component. It is responsible for forwarding media streams from source clients to their intended destination clients.
- **Multipoint Control Unit (MCU)**: an UP AF also deployed as a cloud or edge component. In addition to forwarding media streams between clients, it offers advanced capabilities such as mixing and transcoding, enabling the delivery of personalised, unified streams to each target client.
- **Remote Renderer**: an UP AF deployed as a cloud or edge component. Acting as a surrogate player in the network, it renders 2-dimensional (2D) or 360° video streams and delivers them to lightweight client players.
- **Native (full-fledged) or web (lightweight) players**: UP AFs deployed at the UE. They are software components used by clients for media content creation and/or consumption, as well as for handling required user interactions and communicating with other remote users.
- **Holo Orchestrator**: an UP AF dedicated to orchestration and session management, deployed as a cloud component. It manages session control and coordinates with edge orchestration entities to ensure efficient resource use and service delivery.
- **Monitoring System**: a telemetry AF intended for monitoring, deployed as a cloud component. It collects and hosts key Quality of Service (QoS) metrics, resource usage data, and activity logs

(i.e., telemetry), which are essential for evaluating session performance and enabling dynamic adaptations.

The streaming and communication protocols, along with the control protocols used to exchange metadata related to session management with the Holo Orchestrator, are integral components of the E2E VR platform, as shown in Figure 1.

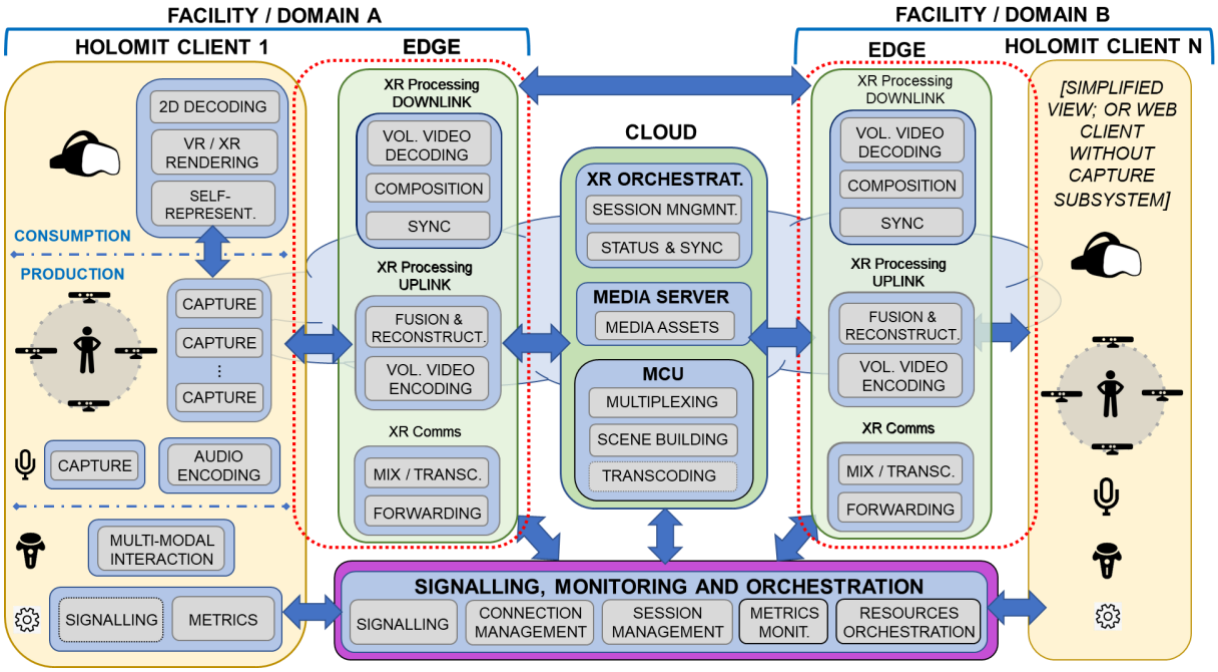


Figure 1. Components for VR User Plane.

2.2 AR CONTROL PLANE

Figure 2 presents a high-level overview of the holographic communication service enabled by the integration of an AR application with the IMS, which forms part of the network CP. The figure illustrates the architecture of the IMS data channel, highlighting its key components and communication flows. The data channel elements are distributed across IMS and public cloud infrastructures. The IMS cloud components are responsible for managing the signalling between the consumer UE and the AR media servers. A signalling server, deployed in the public cloud, facilitates the establishment of connectivity between the Data Channel Signalling Function (DCSF), the media server, and the Agent UE through the exchange of WebSocket messages.

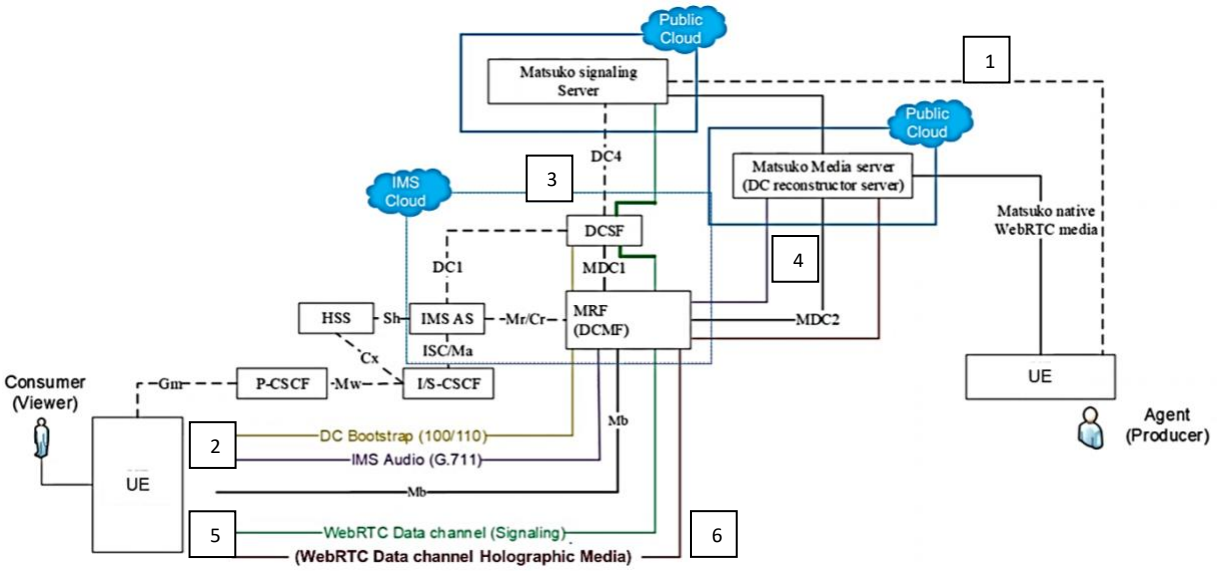


Figure 2. IMS data channel architecture design.

To enable the exchange of holographic data through the UP, a set of preliminary actions must be executed by the CP to ensure proper communication between the involved network components.

The information flow depicted in Figure 2 unfolds across six key steps:

1. The process begins with the agent (producer) registering with the signalling server session manager to initiate the session and await the consumer (viewer) user device to initiate a standard IMS call to the IMS Data Channel (IMSDC) service.
2. Following the call initiation, the consumer device downloads the AR-enabled application from the IMS Application Server (IMS-AS), which is represented through the native phone dialer on the viewer's device.
3. The signalling server then receives a registration request for the holographic call.
4. In response, it automatically retrieves information from the available data channel reconstruction servers to identify and connect to the most optimal one based on current conditions.
5. Once the connection is determined, the signalling server notifies the agent user of the established link, prompting the exchange of Session Description Protocol (SDP) messages and initial session parameters.
6. Once the session has been established, the agent gains access to the selected reconstruction server and transitions to the UP for the media transmission. The 2D/ 3-dimensional (3D) video data uses the Web Real-Time Communication (WebRTC) data channel, while the audio is streamed through IMS. Both 2D/3D video data and audio are synchronised.

3 MULTI-SENSOR VOLUMETRIC RECONSTRUCTION

This section presents the 6G XR Volumetric Capturer enhancements since the deliverable D3.1. To maintain clarity and avoid redundancy, this section highlights the principal advancements in the capture and reconstruction components. These include enhancements to camera hardware and optics, the development of novel calibration and data fusion algorithms, and the implementation of a real-time Volumetric Reconstruction application, which was demonstrated during the 6G-XR General Assembly held in Oulu, Finland, on 28 January 2025, as shown in *Figure 3*. References to D3.1 are used only to clarify baseline configurations. Section 3.1 ("Video Capture") details the redesigned multi-camera layout, upgraded optics, and the introduction of an automated extrinsic-calibration pipeline plus aberration-correction filtering. Section 3.2 ("Video Reconstruction") covers the new real-time Volumetric Reconstruction application, capable of integrating heterogeneous light-field streams at ≤ 30 fps with ≤ 100 ms overall latency, and describes enhancements to intrinsic calibration and guided-filter smoothing. Wherever relevant, we summarise the state at D3.1, then contrast it with the finalised D3.2 implementation in a concise evolution table.

Figure 3. Demonstration of the 6G XR volumetric capture and reconstruction pipeline at the 6G-XR General Assembly. (Left) The Volumetric Capturer is set up at Raytrix offices in Kiel, Germany. (Right) Raytrix is presenting a live volumetric holoported user in Oulu, Finland.

3.1 VIDEO CAPTURE

The Volumetric Capturer acquires four synchronised light-field streams and converts them into RGB + depth (RGBD) frames for downstream volumetric reconstruction. In D3.1, four identical R32v1 light field cameras arranged symmetrically provided the capture coverage. Since then, the gained experience with the R32v1 has allowed for substantial design updates, which have been implemented in the new R32v2 cameras.

3.1.1 Description of the component

Since D3.1, we have introduced version 2 of the R32 in a new "3 + 1" camera arrangement. Three R32v2 light field cameras are spaced evenly at 120° intervals around the subject, ensuring comprehensive full-body coverage. The fourth R32v2 camera is positioned front-facing, directly above one of the body cameras, explicitly optimised for facial capture. This dedicated face camera provides notably higher lateral and depth resolution (< 5 mm XYZ), due to its smaller capture area (500 mm × 333 mm), which is critical for accurately capturing subtle facial expressions—areas naturally prioritised by human perception during interactions. In contrast, general body language occurs at significantly larger spatial scales, tolerating the slightly lower lateral resolution (< 10 mm XYZ) provided by the three body-oriented cameras.

The transition from R32v1 to R32v2 includes targeted hardware adjustments informed by practical experience with the earlier model, which are:

- The microlens array pitch was increased from 125 µm to 500 µm, and the aperture was narrowed from f/1.8 to f/3.8. These modifications intentionally reshape the redundancy-vs-depth curve of the camera, shifting the optimal focus plane from the traditional redundancy of 2 to a redundancy of 4. Contrary to standard plenoptic 2.0 design principles, which typically aim to maximise peak resolution at one depth plane, this approach sacrifices peak sharpness to maintain a consistently high resolution (within 30% of its peak) across the entire ±1 m depth range (total 2 m depth of field). Although larger microlenses inherently introduce some additional aberrations, the reduced aperture size (smaller f-number) effectively mitigates overall optical aberrations compared to the previous design.

- A guided-filter post-processing stage is integrated into the RGBD extraction pipeline to reduce residual microlens-induced depth artefacts, improving the immediate quality of RGBD frames for subsequent volumetric reconstruction.

- The focal lengths of the main lenses were increased to 35 mm for the body and 105 mm for the face cameras. The reduced aperture enabled this focal-length adjustment, which further reduces aberrations and their adverse effects on depth estimation. This simplifies calibration while maintaining the capability to capture large fields of view even in confined spaces.

These hardware adaptations significantly enhanced the performance of the Volumetric Capturer, improving its depth resolution from ≤ 50 mm to ≤ 10 mm for body cameras and ≤ 5 mm for the dedicated face camera.

3.1.2 Final hardware

- **Raytrix R32v2 cameras (×4):**
    - Onsemi XGS 32.4 MP global-shutter sensors (3.2 µm pixels, 36 fps), unchanged from D3.1 for proven reliability.
    - Thermal-stabilized, rigid housing and custom lens mount, unchanged from D3.1 for long-term stability.
    - Improved microlens array: 500 µm pitch Microlens Array (MLA) with f/3.8 aperture.
    - New main lens: focal length increased to 35 mm (body)/ 105 mm (face).
- **Connectivity to Edge:** CoaXPress 12.5 Gbps links (≤ 30 m) with PoCXP power; FPGA timing distributor provides sub-µs sync, unchanged from D3.1.

3.1.3 Final software

- GeniCam interface, unchanged from D3.1.

3.1.4 Evolution compared to the previous release

Between D3.1 and D3.2, the Volumetric Capturer was upgraded with a specialised “3+1” camera arrangement optimised for improved facial detail. The redesigned R32v2 cameras introduced larger microlenses, a narrower aperture, and longer focal-length lenses, resulting in significantly enhanced

depth resolution. A comparison of the changes from R32v1 to R32v2 is presented in Table 1 and the factsheet of R32v2 is attached as Appendix A - R32 light-field camera factsheet.

Table 1. Comparison of the first and final releases of the Video Capture.

Feature	First release (R32v1)	Final release (R32v2)
Camera arrangement	Four identical R32v1 cameras (90° intervals)	Three body cameras (120° intervals) + one dedicated face camera
MLA	125 µm pitch, f/1.8 aperture	Redesigned MLA: 500 µm pitch, narrower f/3.8 aperture
Redundancy-vs-depth curve	Redundancy increases from 2 to 12 times over the capture range	Redundancy increases from 3.5 to 4.5 times over the capture range
Lateral resolution philosophy	Maximise peak resolution	Maintain sufficient resolution until the capture range edges
Depth resolution	≤ 50 mm	≤ 10 mm (body), ≤ 5 mm (face)
Main lens focal length	24.5 mm	35 mm (body), 105 mm (face)

3.2 VIDEO RECONSTRUCTION

The video reconstruction component integrates multiple RGBD streams produced by the Volumetric Capturer into a single coherent volumetric representation in real-time.

3.2.1 Description of the component

Since D3.1, we have developed a dedicated Volumetric Reconstruction application capable of combining heterogeneous RGBD streams from multiple synchronised light-field cameras into a unified volumetric output. This application addresses practical challenges, notably efficient computational load balancing and synchronisation of RGBD streams from cameras assigned to one or more Graphic Processing Units (GPUs). Due to processing variability, RGBD frames from different streams or GPUs can arrive asynchronously. Thus, a dedicated resynchronisation mechanism was introduced, ensuring that all camera streams are temporally aligned with minimal latency overhead before fusion.

The system supports adaptive reconstruction resolutions, dynamically addressing network bandwidth limitations or computational congestion. This capability also allows prioritisation of critical regions of interest, such as faces, with higher detail, while simultaneously reconstructing body regions at lower resolutions when needed. The reconstruction pipeline is highly scalable, supporting an arbitrary number of cameras with different frame rates, resolutions, or types, and was demonstrated at the

internal General Assembly in Oulu in January 2025, showcasing stable real-time reconstruction at ≤ 30 fps with E2E latency ≤ 100 ms.

In addition to stream synchronisation and fusion, significant enhancements were introduced in intrinsic and extrinsic calibration algorithms. The extrinsic calibration pipeline was automated, dramatically reducing calibration duration (typically < 1 min). The intrinsic calibration methodology was improved based on extensive analysis of optical aberrations in wide-angle configurations, significantly enhancing accuracy.

Lastly, the guided-filter post-processing stage, mentioned in Section 3.1, mitigates depth artefacts arising from optical aberrations. This edge-preserving filter enhances the final volumetric reconstruction quality by smoothing depth inaccuracies without compromising critical geometric detail.

3.2.2 Final hardware

- **Compute configuration:** The edge server is equipped with one NVIDIA RTX 4090 GPU per camera, each hosting an Image Processing Unit (IPU) instance, ensuring real-time RGBD extraction, and an Euresys Coaxlink Quad CXP-12 frame grabber (unchanged from D3.1).

3.2.3 Final software

The Volumetric Reconstruction application is a custom software system that transforms multiple synchronised RGBD streams into a unified volumetric representation in real time. It is designed for scalability, adaptability, and integration, supporting standalone use and future deployment within the broader 6G XR stack.

Key software components include:

- **Calibration Module**

  This module ensures accurate geometric alignment across all RGBD streams. It supports both intrinsic and extrinsic calibration:

  Intrinsic calibration models the internal optics of each camera, correcting for lens distortion and aberration. This improves the accuracy of depth maps, particularly at the image periphery.

  Extrinsic calibration computes the position and orientation of each camera within the Capturer to ensure proper spatial alignment during data fusion. The new pipeline is fully automated, completes in < 1min, and achieves accuracy at or below the native depth resolution (~10 mm).

- **Image Processing Unit:**

  Each incoming raw light-field stream is processed by a dedicated IPU, which converts the plenoptic input into calibrated RGB + depth (RGBD) frames in real time. This process includes optical decoding, disparity estimation, and view synthesis. IPUs run on GPU and are optimised for high throughput with minimal latency.

  A real-time (GPU-based) guided filter smooths depth maps, reducing microlens-induced noise while preserving fine geometric detail in the fused output.

  The system supports dynamic Level-of-Detail, allowing higher fidelity for key regions (e.g., the face) while lowering detail in less critical areas to save bandwidth and computation.

- **Stream Resynchronisation**

  This layer aligns frames in time before fusion to compensate for asynchronous processing, especially when streams share GPUs. It ensures consistency with minimal added latency.

This final version of the software was successfully demonstrated live at the January 2025 6G-XR General Assembly, sustaining ≤ 30 fps and < 100 ms E2E latency under realistic conditions.

3.2.4 Evolution compared to the previous release

Between D3.1 and D3.2, the video reconstruction component transitioned from a concept-level architecture to a robust, real-time application capable of multi-stream volumetric fusion. Key advancements include an automated calibration pipeline, per-stream RGBD extraction via dedicated IPUs, GPU load balancing with stream resynchronisation, and adaptive resolution handling for enhanced fidelity where needed. Together, these changes enable scalable and latency-optimised reconstruction with significantly higher spatial and temporal accuracy than the initial version. Table 2 presents a comparison between the initial and final versions of the video reconstruction.

Table 2. Comparison of the first and final releases of the video reconstruction.

Feature	First release	Final release
System status	Conceptual prototype (no E2E integration)	Fully integrated, real-time reconstruction pipeline
Calibration	Manual, structured wizard	Fully automated extrinsic calibration, improved intrinsic modelling
Stream handling	Conceptual prototype	Demonstrated per-stream resynchronisation for multi-GPU setups
Latency / Frame rate	Not defined	≤ 100 ms latency, ≤ 30 fps real-time output demonstrated

4 CLOUD/EDGE XR PROCESSING AND SCALABILITY

This section presents the final version of the XR Enablers related to cloud/edge XR processing, whose aim is to 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 Description of the component

In traditional video conferencing services, SFUs are typically used for the exchange of multimedia information between the involved clients. 6G-XR has departed from a functional SFU (an outcome of the EU H2020 VR-Together project[1]), built on top of Node.js[2], 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[3], as illustrated in Figure 4. A more detailed architecture of the SFU is provided in Figure 5.

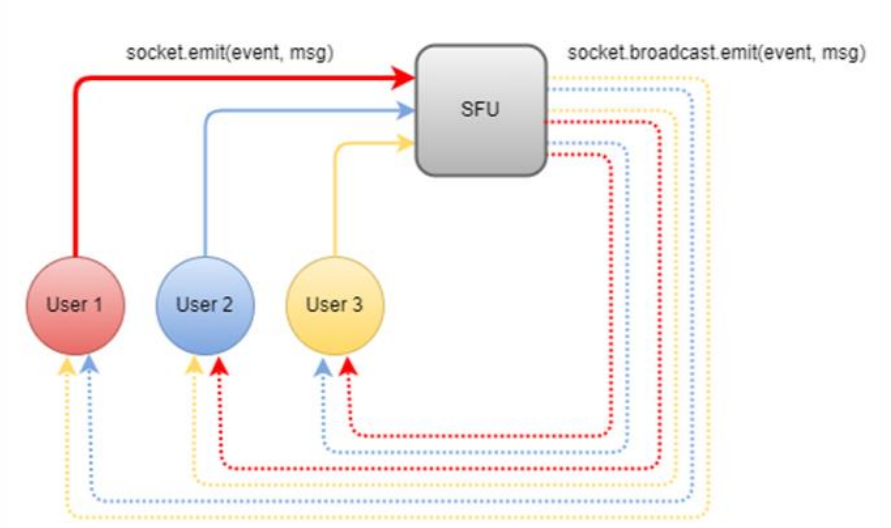


Figure 4. High-level communication architecture when adopting a Selective Forwarding Unit.

[1] https://vrtogether.eu/

[2] https://nodejs.org/

[3] https://socket.io/

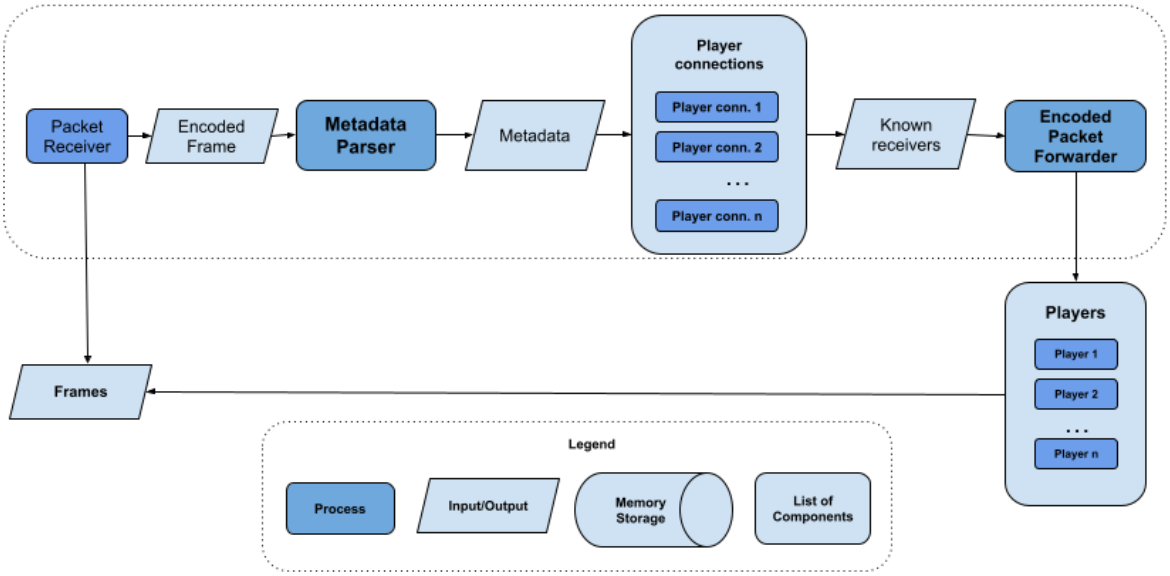


Figure 5. SFU Architecture.

Diverse key innovations have been added to the SFU component in 6G-XR during its first and second periods. The main innovation in the first 6G-XR period, reported in D3.1 [2], can be summarised as:

- The SFU has been virtualised (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).
- Novel interfaces so that: (i) the Holo Orchestrator (Section 7.1) can identify and select specific instances of SFUs for running sessions, or even instantiate them, based on specific criteria (e.g., deployment domains, edge resources); and (ii) the SFU can be interfaced to other in-cloud VNFs, like an MCU (Section 4.2) and a Remote Renderer (Section 4.3).
- The SFU can be decoupled into two independent modules/services:
    - **Media Manager**: responsible for forwarding the audio + video data between the involved clients.
    - **Events Manager**: responsible for forwarding relevant metadata between the involved clients, like their positions in the virtual space, or any other specific events that can be originated/triggered in the media session (e.g., interactions with the environment).

The SFU is an UP AF just in charge of data forwarding, not performing any media processing tasks like stream multiplexing and/or transcoding. 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 typically becomes a scalability bottleneck.

To overcome such bottleneck, two main innovations have been added to the SFU in the second 6G-XR period:

- A modular distribution and deployment architecture has been devised so that more than 1 SFU can be concurrently enabled and used for a given media session, thus enhancing scalability (Figure 6). 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.

Conducted tests have proven that the number of concurrent users per session when using >1 SFU can be doubled compared to the usage of a single SFU.

- New modules of the SFU have been developed, so that the relative positions and viewports, or Field of View (FoV), of each participant in the session are considered; that allows to decide whether their associated streams are to be forwarded to each destination or not. To achieve this, a visibility (v) matrix between the participants is built and dynamically updated, based on binary values (i.e., visible or not) based on specific and configurable viewport and distance thresholds. This is sketched in Figure 7, where it is shown that Client_5 and Client_6 are considered not visible (i.e. v=0) for Client_1, and thus their streams are not delivered to that target client. By applying this strategy for each target client, the scalability of the session largely increases, being limited to the number of clients that are supported within the configured FoV at any moment.

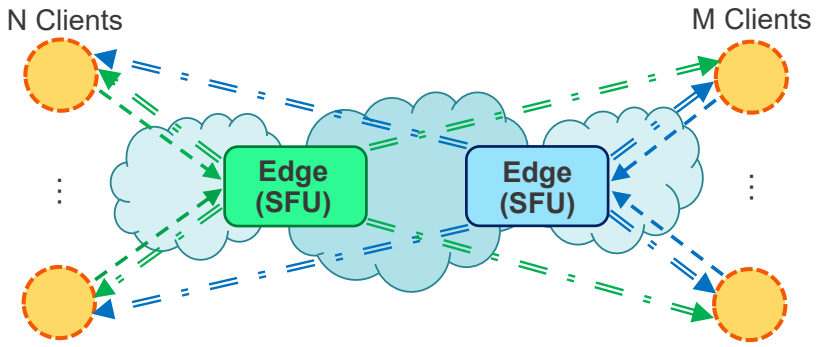


Figure 6. High-level schemes of sessions with clients connecting to two different SFUs.

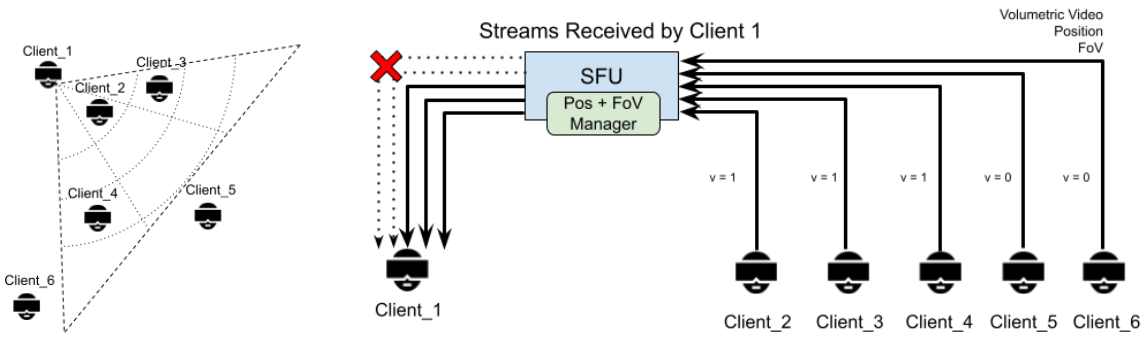


Figure 7. Position- and FoV-aware delivery module of the SFU.

4.1.2 Final 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. Conservative specifications for a server hosting both the SFU and Holo Orchestrator could be: 4 virtual Central Processing Units (vCPU), 8 GB RAM, 20 GB storage. It has also been successfully deployed on a server on the Microsoft Azure cloud computing platform with Standard DS1 v2 specs (1 vCPU, 3.5 GB RAM).

4.1.3 Final 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 (VMs) on Azure. It has also been virtualised as a Docker container and a Helm chart, which eases its deployment on any machine, including those provided by hyper-scalers.

4.1.4 Evolution compared to the previous release

Table 3 lists and summarizes the features and capabilities of the SFU, comparing the final released version to the first released one, reported in D3.1 [2].

Table 3. Comparison of the first and final releases of the SFU.

Feature	First release	Final release
Virtualisation	Docker	Helm Chart
Architecture / Modularisation	SFU Selection Different managers for media and events data	Support for multi-SFU architectures
Scalability	Limited to the scalability limits of a single SFU (bandwidth is the most limiting factor) (~8 users/session)	SFU selection in the most appropriate Edge / Network (latency reduction >100ms) Support for multi-SFU architectures, with load balancing (~15 users/session) Position and viewport-aware delivery (>20 users/session)

4.2 MULTIPOINT CONTROL UNIT

4.2.1 Description 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 the EU H2020 VR-Together project[4]) 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 to provide a single personalised output Point Cloud stream to each involved user in the

[4] https://vrtogether.eu/

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 in Figure 8, 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.

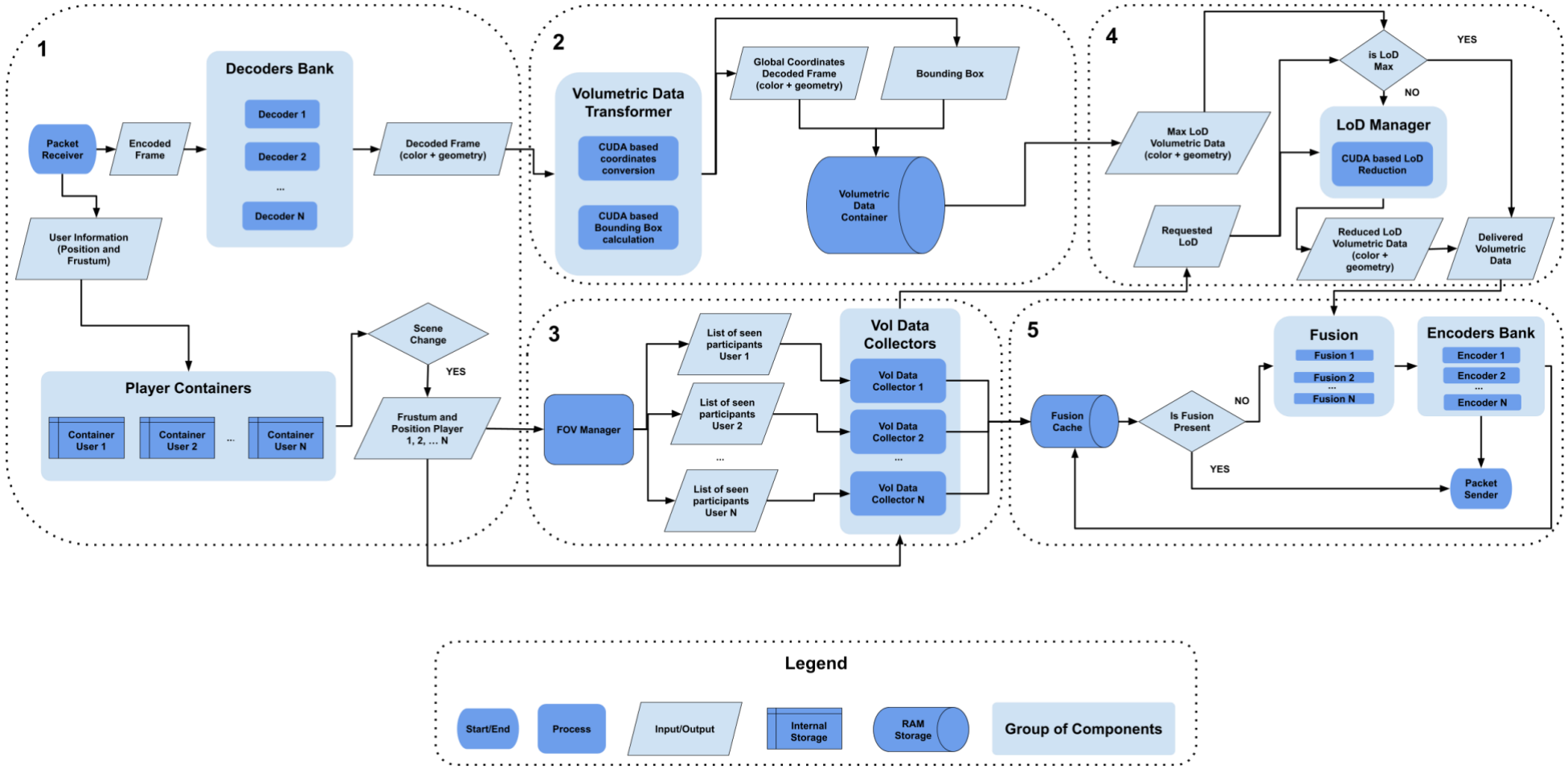


Figure 8. MCU architecture.

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 instantiates 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 optimisations, such as user position and frustum (visible area coordinates). The *Player Containers* provides 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 is allocated to perform the corresponding decoding process. The decoded volumetric frames are then available as colour components 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[5] (CUDA) based implementation that performs the coordinates conversion to the global coordinate system and, at the same time, evaluates the corresponding bounding box. The output is then stored in a RAM-based component, called ***Volumetric Data Container***, which is in charge of holding the transformed geometry data. FInally, the remaining MCU components are able to request the information stored here for the optimisation of the streams.

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

To optimise 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 the 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 is 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***.

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

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

4. **Block 4 - Level of Detail (LoD) Optimisation**

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 avoids performing redundant operations and delivers the Volumetric Video as previously received. On the other hand, if the LoD requested is lower than the maximum, the process called ***LoD manager*** is activated and performs a CUDA-based operation for the reduction of the LoD. The resulting output is then 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 a 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 optimisation, 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 assigns 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 is skipped, and the previously created fusion is directly delivered.

The next key innovations have been applied to that MCU component in 6G-XR, mostly in its first period (and thus already reported in D3.1 [2])

- It has migrated to Linux to allow for its virtualisation (VNF) and dynamic orchestration (WP2).
- 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.

- New interfaces between the MCU, the Holo 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.

4.2.2 Final hardware

The MCU has been successfully deployed and tested in two server machines with the following specifications:

- Processor: 2 x Intel (R) Xeon (R) Gold 5218R (2.1 GHz), 40C/80T; Memory: 115 GB RAM; GPU NVIDIA Tesla T4 16GB.
- Processor: AMD Ryzen Threadripper 3970X (3.70Ghz), 32C; Memory: 16 GB; GPU: NVIDIA GeForce RTX 3060Ti.

4.2.3 Final software

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

4.2.4 Evolution compared to the previous release

The evolution of the MCU compared to its departing version was mostly performed in the first period of 6G-XR and thus reported in D3.1 [2]. In the second period, the efforts were focused on improving the scalability of the SFU component (see Section 4.1), due to the promising performance obtained in the first explorative tests.

4.3 REMOTE RENDERER

4.3.1 Description of the component

The Remote Renderer developed by the 6G-XR project is meant to enable the consumption of VR experiences for devices with limited local processing capabilities. Deployed as a VNF within the edge infrastructure, the Remote Renderer leverages VR content rendering technologies and network virtualisation to generate two different types of media streams:

- Personalised media stream: employed for interactive VR users, featuring a media session based on the WebRTC protocol. The remote renderer renders volumetric video content into a VR scene and then produces a 360° mono or stereo video stream based on a selected viewpoint, which can be personalised using the 6DoF information obtained from the user's device. As a consequence of the above, each media stream is transferred in real-time and consumed by only one user.

- Non-customised media stream: employed for passive VR consumption, featuring a media session based on the DASH protocol. The remote renderer renders volumetric video content into a VR scene and then produces a 2D, 360° mono or stereo video stream. In this case, the 6DoF information is not received from the user's device, as the viewpoint is fixed during the media session. As a consequence of the above, the media stream is unique for all users and is transferred through the standard HTTP protocol.

The internal components of the Remote Renderer, shown in Figure 9, are the following:

- Surrogate Player: it functions similarly to a native video player by receiving and decoding various inputs from orchestration, switching, and mixing units. These inputs include configuration messages, events, and data sources such as Point Cloud data, as well as 2D and 360° video and audio streams. However, unlike a typical player, it avoids displaying these inputs directly.

- Rendering Engine: responsible for rendering the data sources received by the Surrogate Player into a 2D or 360° video stream with accompanying audio. It achieves this by inserting a virtual camera and a virtual microphone within the VR scene. Critically, this module utilises the 6DoF information received from the user's device to adjust the position and orientation of the virtual camera, thereby personalising the generated video stream according to the user's viewpoint within the VR scene.

- Streaming Server: it performs two key functions. First, it encodes the 2D or 360° video stream generated by the Rendering Engine and transmits it to the user's video player using the most appropriate streaming protocol. Second, it acts as a receiving endpoint for the 6DoF information generated by the user's video player and sends this data to the Rendering Engine for viewpoint adjustments.

Two different video players are employed depending on the type of media stream (personalised stream with WebRTC or non-customised with DASH). The connection of the video players to the Remote Renderer and the entire workflows for both WebRTC and DASH are explained in Sections 5.2 and 5.3.

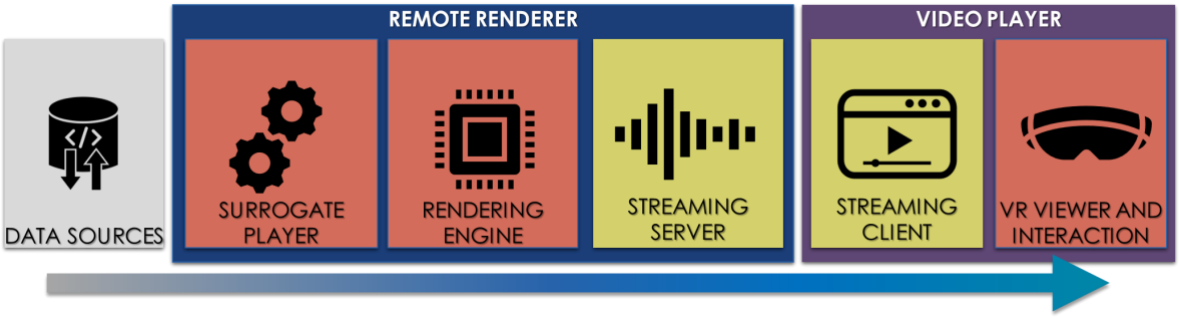


Figure 9. Logical modules of the Remote Renderer and Video Player.

4.3.2 Final hardware

For the final version of the Remote Renderer, two different setups are employed to deploy and test its functionalities. The first one consists of a VM with a Kubernetes installation, the second one of a physical machine with bare-metal Kubernetes. Table 4 shows the characteristics of both. In both cases, the GPU is employed in passthrough mode and not shared with any other VM or container. These configurations have allowed the generation of 360° Stereo video stream up to 4K at 60 FPS.

Table 4. Hardware employed to deploy and test the Remote Renderer.

Hardware	VM specifications with Kubernetes	Bare-metal Kubernetes specifications
CPU	Intel Xeon 12 Cores (2.1 GHz)	AMD Ryzen Threadripper 3970X 32 Cores (3.70GHz)
RAM	32GB	16GB

GPU	NVIDIA Quadro RTX 4000	NVIDIA RTX 3060 Ti

4.3.3 Final software

The final implementation of the Remote Renderer includes all three logical modules, while the Surrogate Player was still missing in the previous release. The implementation of the Surrogate Player is based on HoloMIT SDK v3.2.0 and allows to revive real-time volumetric video and audio from other components, such as SFU (Section 4.1) and/or MCU (Section 4.2).

The final solution is mostly based on C++ and C# programming languages and works in the Ubuntu 22.04 Operating System (OS) LTS environment. It has Unity and GStreamer frameworks as main development dependencies. Some GStreamer libraries have also been modified to make the DASH implementation compliant with DASH Industry Forum (DASH IF) recommendations[6]. These modifications have been submitted as three separate merge requests to the official repository[7,8,9]. The complete list of the Remote Renderer dependencies is shown in Table 5.

Table 5. Software dependencies of the Remote Renderer and their versions.

Software and frameworks	Version
Unity	2022.3.50f1
Unity Render Streaming	3.1.0-exp.7
HoloMIT SDK	V3.2.0
Vulkan	1.3.204
GStreamer (modified, merge requests submitted to official repository)	1.24
Nvidia driver	535.161.07
Nvidia Cuda Toolkit	12.2

Figure 10 and Figure 11 show the final Remote Renderer running as a standalone application. In the virtual room, 3D objects in point cloud format can be rendered (Figure 10), while virtual cameras and microphones are instantiated to capture the audio and video to be sent to the video player. Moreover, interactive users through WebRTC are shown through an avatar in the scene (Figure 11), while passive users through DASH are not visualised.

[6] https://dashif.org/

[7] https://gitlab.freedesktop.org/gstreamer/gstreamer/-/merge_requests/7886

[8] https://gitlab.freedesktop.org/gstreamer/gstreamer/-/merge_requests/8168

[9] https://gitlab.freedesktop.org/gstreamer/gstreamer/-/merge_requests/8608

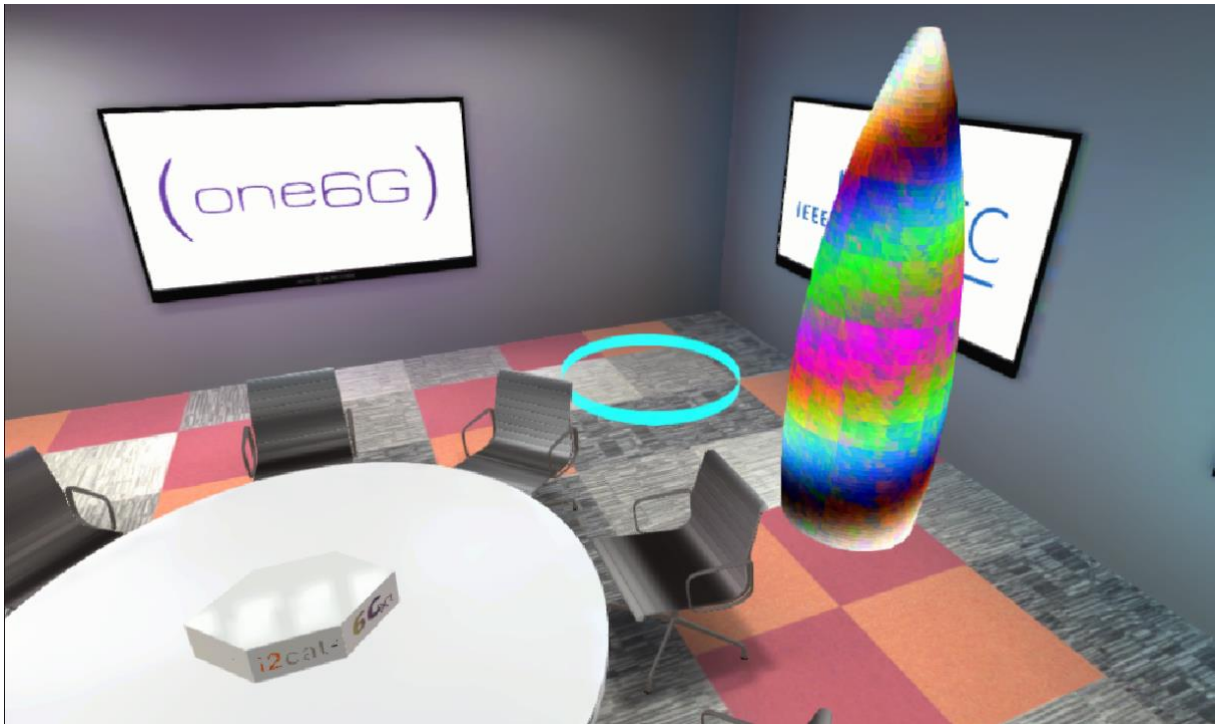


Figure 10. Remote Renderer running as a standalone application with a synthetic point cloud in the VR scene.

Figure 11. Remote Rendering running as a standalone application with a 3D Avatar in the VR scene.

Finally, the Remote Renderer can also work without a Graphical User Interface (GUI), i.e., by enabling the headless mode of the Unity-based application. This allows to have a containerised version running with the software environment presented in Table 6.

Table 6. Environment for the deployment of the containerisation Remote Renderer.

Software	Version
Ubuntu OS	22.04
Docker	27.5.1
Docker Compose	2.32.4
Kubernetes	1.32.1
Nvidia Container Toolkit	1.15.0
NVIDIA GPU Operator	24.9.2

Thanks to the containerisation, the Remote Renderer has its Dockerfile for building purposes and Helm Chart for deployment on a Kubernetes cluster.

4.3.4 Evolution compared to the previous release

The major change compared to the previous release consists of adding the Surrogate Player module. This module is crucial for the interconnection with other XR Enablers, such as the SFU (Section 4.1) and the Holo Orchestrator (Section 7.1). Furthermore, several features of the modules Rendering Engine and Streaming Server were improved. All the improvements and changes over the previous version of the Remote Renderer are described in Table 7.

Table 7. Comparison of the first and final releases of the Remote Renderer.

Feature	First release	Final release
Inputs	Pre-recorded and locally stored audio and video files (2D video or volumetric video/Point Cloud)	Real-time volumetric video and audio received through the Surrogate Player
Rendering	2D Mono or 360° Mono	2D Mono, 360° Mono or 360° Stereo
Interaction	WebRTC: translation and rotation through data-channel DASH: no interaction	WebRTC: translation through data-channel and rotation performed locally in the UE using 360° video DASH: rotation performed locally in the UE using 360° video
Output codecs and protocols	WebRTC: VP8, H.264 and OPUS DASH: H.264 and AAC	WebRTC: VP8, H.264 and OPUS DASH: H.264 and AAC RTP over QUIC (RoQ): H.264 Media over QUIC (MoQ): H.264
Streaming adaptation	Not available	Rate Control REST API for WebRTC and DASH

DASH IF compliance	Not compliant	Compliant. Three merge requests submitted to the official GStreamer repository
Containerisation	Dockerfile and Helm Chart with no configurable parameters	Dockerfile configurable through JSON and Helm Chart through ConfigMaps
Communications with Holo Orchestrator and SFU	No	Session management with Holo Orchestrator and multimedia communications with SFU (audio and video volumetric video)

5 ADAPTIVE LOW-LATENCY XR DELIVERY

This section presents the final versions of the XR Enablers developed to provide access to VR experiences 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 Description of the component

The endpoint for the clients of XR services consists of a native Unity-based player (Windows build), which implements:

- The necessary session and resource management features interfacing the Holo Orchestrator.
- The processing and exchange of audio and volumetric video streams for real-time communications.
- A set of multimodal content presentation features.
- A set of interaction features, with the VR environment, and with other remote users.

6G-XR has departed from a fully functional version of this Unity-based player (outcome of the EU H2020 VR-Together project[10]), whose features are provided as a Unity package and associated Software Development Kit (SDK).

The implementation details for the departing native player components and modules can be found in [3], while the components deployed in the first 6G-XR period are detailed in D3.1 [2].

In the second 6G-XR period, the native player has been extended and evolved by:

- Supporting sessions in which a native player can receive data from multiple SFUs (more details in Sections 4 and 7).
- Integrating the latest version of the volumetric video capture sub-system (more details in Section 3), supporting both the sensors by Raytrix and modern affordable RGB-D sensors, like Orbbec Femto Bolt[11].
- Enabling dynamic adjustment of the data rate by modifying the volumetric video resolution or the encoding parameters.
- Integrating new interfaces with the Holo Orchestrator with the goal of:
    - Receiving dynamic endpoints of the SFU to connect to.

[10] https://vrtogether.eu/

[11] https://www.orbbec.com/products/tof-camera/femto-bolt/

- Receiving alerts from the Holo Orchestrator, indicating the need for adjusting the data rate.

- Finalising the migration to Linux, for integration with the Remote Renderer (Section 4.3).

In addition, a new web-based headless client, developed using Node.js and Socket.IO, has been developed to ease testing. This headless client handles pre-recorded holograms, streams them to remote clients, and receives the holograms from the other involved native/web clients in a shared session, but without the need for media capture/rendering or encoding/decoding.

5.1.2 Final hardware

The Unity-based player, integrating the SDK for holographic communications, needs to run on a desktop PC or laptop with GPU capabilities (NVIDIA RTX onwards). That PC then connects via cable to a VR headset for media presentation. Different VR headsets have been successfully tested, including Oculus Rift, Oculus Quest 1/2/3, Oculus PRO, and HTC VIVE. It also supports the connection with volumetric video capture sub-systems with diverse RGB-D sensors (Azure Kinect, Intel Real Sense, Orbbec Femto Bolt) and light field sensors (Raytrix), including single-sensor and multi-sensor setups.

5.1.3 Final software

The Unity-based player, integrating the SDK for holographic communications, needs to run on a Windows 10 or 11 VR-ready device, and has been tested with a variety of Unity[12] versions, starting from 2020.3 and including more recent 2022.3.X releases, with a key requirement of having at least one of the Scriptable Render Pipelines and the new Input System active and enabled.

The Linux version has been tested with Ubuntu 22.04 and 24.04.

5.1.4 Evolution compared to the previous release

Table 8 lists and summarizes the features and capabilities of the Native Player, comparing the final released version to the first released one, reported in D3.1 [2].

Table 8. Comparison of the first and final releases of the Native Player.

Feature	First release	Final release
Volumetric Capture Subsystem Integrated	Azure Kinect sensors	Azure Kinect sensors Raytrix sensors (Section 3) Orbbec Femto Bolt
Holographic Transmission	Support only holograms captured in real-time	Support both holograms captured in real-time and pre-recorded holograms
Data Rate	Fixed	Adaptive, based on instructions received from the Holo Orchestrator

[12] https://unity.com/

SFU connection	Pre-established at the start of the session, and with a single SFU	Dynamic configuration during the session lifetime, supporting connection to multiple SFUs.
Operating System support	Windows and partial Linux implementation	Windows and Linux implementation

5.2 WEBRTC STREAMING TO WEB PLAYER

5.2.1 Description of the component

This section details the web-based Video Player and Signalling Server designed for WebRTC streaming, which operates in conjunction with the Remote Renderer described in Section 4.3 (more specifically, the "Streaming Server" module). Figure 12 illustrates how these two elements are integrated with the Remote Renderer:

- The WebRTC Video Player is responsible for receiving and displaying on the screen the content that has been previously encoded and streamed by the Remote Renderer. Additionally, it provides interactive functionality through the WebRTC data-channel.
- The Signalling Server facilitates the negotiation of multimedia and network parameters before initiating the multimedia communication between the Remote Renderer and the player.

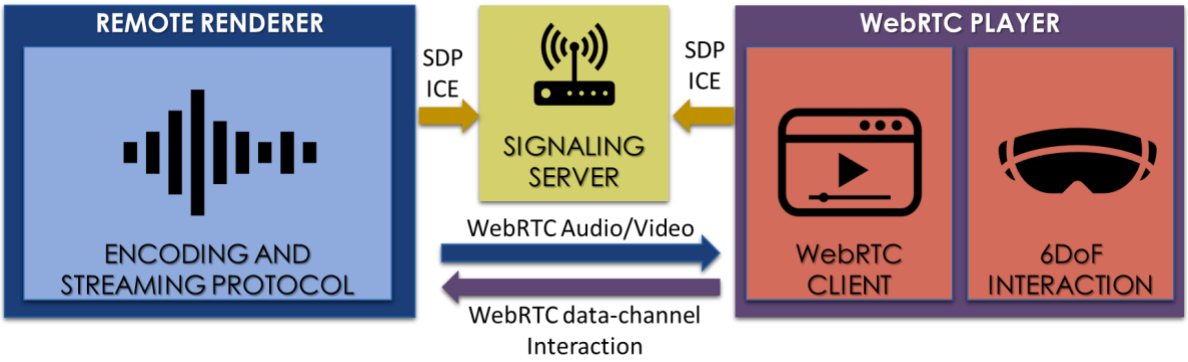


Figure 12. Components of the WebRTC streaming enabler.

The diagram presented in Figure 13 illustrates the operational workflow of the WebRTC-based multimedia streaming designed for real-time remote rendering. The process begins when a user aims to join a remote rendering session via the WebRTC Video Player. He sends a request to join a session and activate the Remote Renderer. This request is handled by the Holo Orchestrator (Section 7.1), which responds by deploying the Remote Renderer and configuring the session parameters. Once the renderer is initialised, it publishes the stream through the Signalling Server. The WebRTC Player subscribes to this stream using the provided Signalling Server endpoint, thereby initiating the WebRTC negotiation phase. This phase comprises an exchange of SDP offers and answers through the Signalling Server, enabling both peers to agree on the media transmission parameters. The WebRTC Player sends an SDP offer, which the Remote Renderer answers with a corresponding SDP response. In addition, Interactive Connectivity Establishment (ICE) candidates are exchanged in both directions to determine the most effective communication path between the peers.

Once the peer-to-peer connection is established, the system enters a continuous rendering and streaming loop. During this loop, the Remote Renderer decodes incoming volumetric video and audio data from the SFU (Section 4.1) or MCU (Section 4.2), renders the scene based on the current viewpoint, and encodes the output stream for transmission. The rendered content, formatted as either 360° Mono or Stereo video with associated audio, is streamed to the WebRTC Player. The player decodes this stream and visualises it for the user.

Simultaneously, a dedicated interaction loop captures the 6DoF input from the user, encompassing both positional and rotational data. This information is transmitted back to the Remote Renderer, which utilises it to update the viewing perspective in real time. This interaction mechanism ensures that the rendered scene dynamically adapts to the user's movement and orientation.

In parallel with the rendering and interaction loops, the system maintains a monitoring and rate control feedback loop. The WebRTC Player collects performance metrics via the WebRTC standard *getStats()*[13] API, which are periodically transmitted to the Monitoring System. The Monitoring System aggregates this data and sends it to the Holo Orchestrator. Based on this feedback, the orchestrator instructs the Remote Renderer to dynamically adjust the video encoder settings through its Rate Control REST API.

Overall, this workflow supports scalable, low-latency media delivery with adaptive viewpoint rendering and robust network adaptation. It enables remote rendering applications that demand both high-quality rendering and seamless user interaction over constrained and variable networks.

[13] https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/getStats

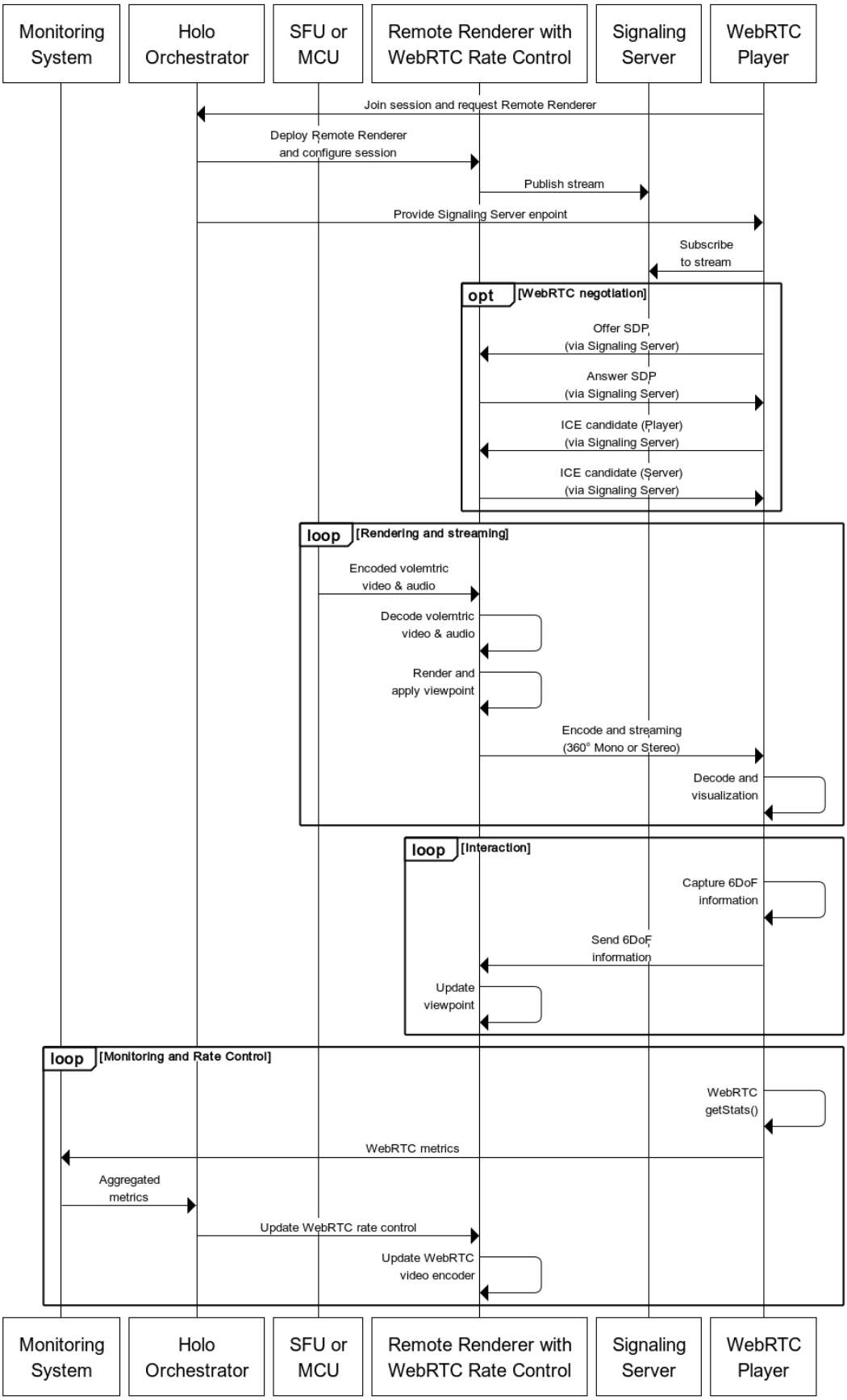


Figure 13. Communications for WebRTC streaming between the Remote Renderer and WebRTC Video player.

5.2.2 Final software

Signalling Server:

As it is a simple server only employed for negotiation among the two WebRTC peers (the Remote Renderer and the WebRTC Video Player), the final version of the Signalling Server did not receive any relevant updates compared to the previous one. It consists of a Node.js application developed using both JavaScript and TypeScript and incorporates a WebSocket library as a fundamental dependency to provide communications with the two WebRTC peers and enable the negotiation.

The software dependencies of the Signalling Server and their versions are shown in Table 9.

Table 9. 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 Video Player:

The WebRTC Video Player is built using native WebRTC technologies. To enhance visualisation capabilities and support 360° Mono and Stereo video playback, the player integrates Three.js, an open-source web framework for creating VR experiences on the web. It is compatible with any device that supports WebXR via a web browser.

In addition to media playback, the WebRTC Video Player captures 6DoF inputs from various devices and sends them to the Remote Renderer to provide real-time interaction. Among input devices, the keyboard and mouse are employed when using a laptop, while motion sensors and joysticks are employed when wearing a VR headset. These inputs facilitate user interaction within the virtual environment and with other users. An illustration of the WebRTC Player is provided in Figure 14 (remote renderer VR scene and live volumetric holoportation) and Figure 15 (VR simulator mode).

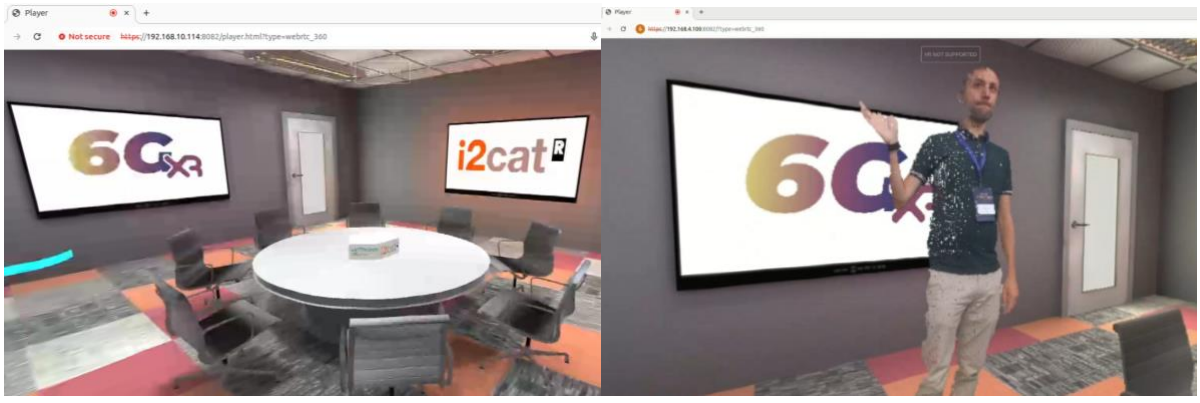


Figure 14. WebRTC Video player in laptop browser. (Left) VR Scene. (Right) Live volumetric holoportation.

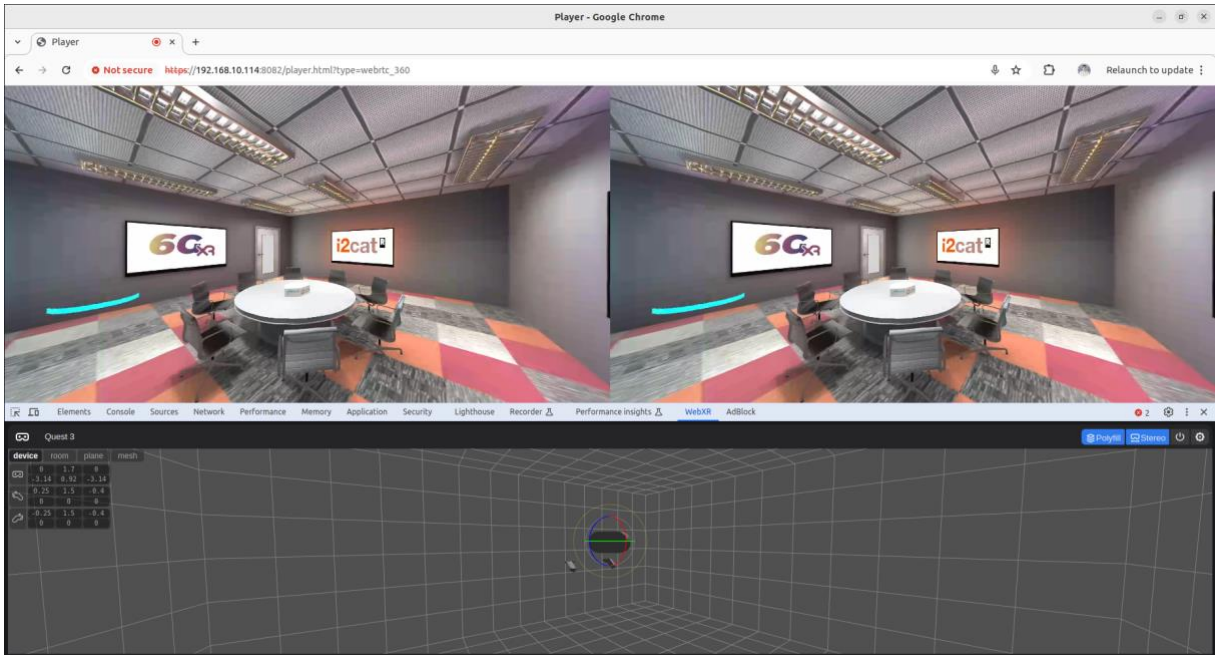


Figure 15. WebRTC Video player in laptop browser with VR simulator.

The development used mainly the JavaScript programming language, while the deployment is performed through Node.js[14] Three.js[15]. The software dependencies and their versions are shown in Table 10.

Table 10. Software versions used in the WebRTC Player.

Frameworks	Version
Node.js	21.2.0
Three.js	R127

Finally, both the Signalling Server and WebRTC Video Player have been containerised through Docker and described through a Helm Chart.

5.2.3 Final hardware

In terms of hardware for running the Signalling Server and the WebRTC Video Player, it is important to mention that they are hardware-agnostic. Since they are based on Web technologies, they do not rely on any specific hardware components and can be deployed as containerised Node.js applications across a variety of platforms.

For the final deployment, the Signalling Server and the WebRTC Video Player are deployed as containerised applications on Kubernetes through their Helm Charts. To test the WebRTC Video Player,

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

[15] https://threejs.org/

the Chrome web browser on standard laptops and the built-in Browser on the Meta Quest 2[16] and Meta Quest 3[17] VR headsets are employed.

5.2.4 Evolution compared to the previous release

While the Signalling Server remains almost the same as the one developed in the first release, some further developments have been executed on the WebRTC Video Player. These changes are presented in Table 11.

Table 11. Comparison of the first and final releases of the WebRTC streaming enabler.

Feature	First release	Final release
Video Player - Video format	2D Stereo, where each video stream is sent over a separate WebRTC media channel	360° Mono and Stereo, where each video stream is sent over a unique WebRTC media channel
Video Player - Interaction	Translation and rotation information sent over WebRTC data-channel	Translation information is sent over the WebRTC data-channel, while rotation is performed locally

5.3 DASH STREAMING TO WEB PLAYER

5.3.1 Description of the component

This section presents the web-based DASH Video Player and HTTP server designed for DASH streaming, which operates in conjunction with the Remote Renderer described in Section 4.3. The purpose of the HTTP Server is to host the DASH Media Presentation Description (MPD) file and its associated audio and video segments, which are generated by the Remote Renderer and server to the Video Player through the HTTP protocol.

While DASH introduces higher latency compared to WebRTC (see Section 5.2), it remains a suitable alternative in scenarios where real-time interaction with the VR environment is not necessary and higher latency can be tolerated. Instead, DASH is ideally suited for passive media consumption, where users experience the VR scene from a fixed viewpoint and do not engage in interactive behaviours. As a widely adopted solution for scalable multimedia delivery, DASH leverages the HTTP protocol to enable robust scalability in terms of connected users.

Figure 16 illustrates how the DASH Video Player and HTTP Server are integrated with the Remote Renderer:

- The HTTP Server facilitates the hosting of the MPD and the multimedia segments, including audio and video, that are generated by the Remote Renderer.

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

[17] https://www.meta.com/es/en/quest/quest-3/

- The DASH Video Player is responsible for requesting the MPD from the HTTP Server and parsing it. Then, it starts downloading the multimedia segments, decoding them and displaying them on the device screen.

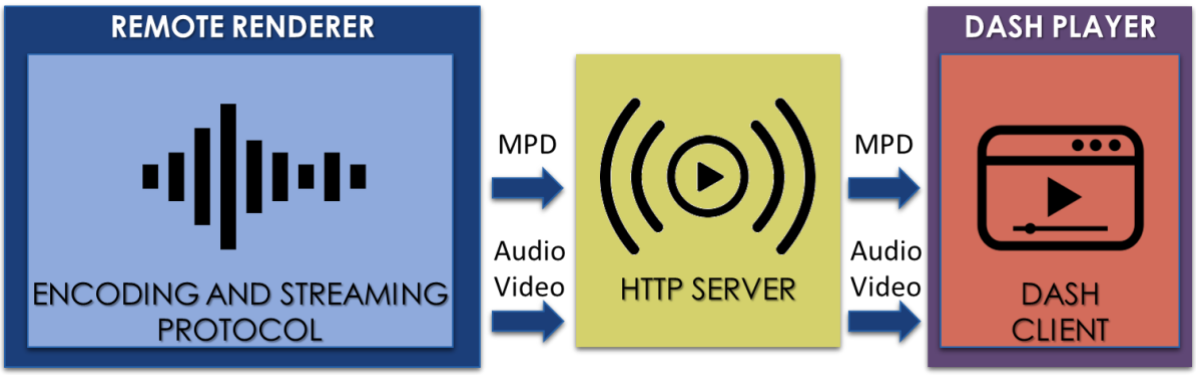


Figure 16. Components of the DASH streaming enabler.

This diagram in Figure 17 illustrates the operational workflow of a DASH–based communication for remote rendering volumetric content, emphasising a passive user's session with fixed viewpoint and adaptive rate control. In the starting phase, the Holo Orchestrator has pre-deployed the Remote Renderer, providing it with the necessary configuration to connect to the SFU (Section 4.1) or MCU (Section 4.2) to retrieve the volumetric video and audio.

When the DASH Video Player requests to initiate a session for passive media consumption, the Holo Orchestrator responds by providing the HTTP Server endpoint. Then, the core rendering and streaming process operates in a continuous loop. Initially, the Remote Renderer decodes incoming volumetric video and audio streams. The rendering of these streams within the VR scene is performed from a fixed viewpoint to generate a video output in 2D Mono, 360° Mono or 360° Stereo format with associated audio. Then, the video is scaled at different resolutions (representations), being encoded in parallel and generating DASH media segments. The resulting segments, along with the associated DASH MPD, are written to disk and made available via the HTTP server. The DASH Video Player retrieves the MPD and segments from the HTTP server and begins decoding and visualising the content.

Parallel to this media delivery workflow, the system includes a monitoring loop. The DASH Player periodically collects playback and streaming performance metrics, which are transmitted to the Monitoring System. The Monitoring System aggregates this data and feeds it back to the Holo Orchestrator. Based on this feedback, the Holo Orchestrator acts on the Rate Control REST API of the Remote Render to adjust the MPD to show only the desired representations. This closed feedback loop allows the system to respond dynamically to varying network conditions, ensuring smooth video delivery and maintaining a balance between quality and reliability.

In summary, this workflow offers a scalable and robust framework for passive volumetric content streaming using the DASH protocol. It is particularly well-suited for UCs where interaction is not required but where high-quality and scalable video transmission is essential.

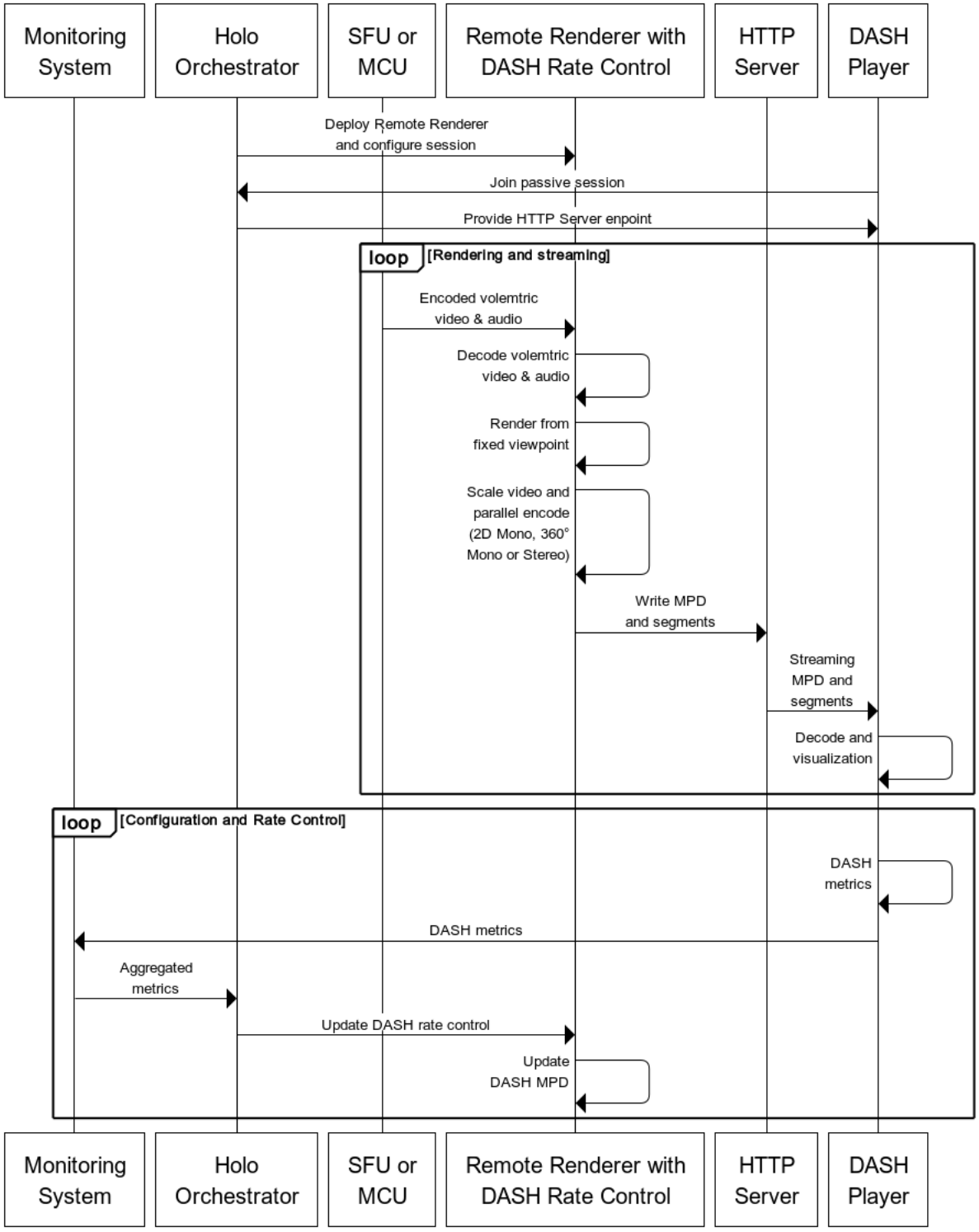


Figure 17. Communications for DASH streaming between the Remote Renderer and WebRTC Video player.

5.3.2 Final software

HTTP Server:

The HTTP Server employs the same technologies as the previous release. It consists of a Node.js application using the default HTTP Server library to serve MPD and segment files. As an improvement,

a UTC timing information compliant with the DASH IF recommendation[18] is also served to both the Remote Renderer and the DASH Video Player. This information is necessary to provide a better synchronisation at the multimedia application level between the multimedia provider and consumer. All the final software dependencies and their versions are shown in Table 12.

Table 12. Software versions used in the HTTP Server.

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

DASH Video Player:

As mentioned in Section 4.3.3, the improvements on the GStreamer framework make it compliant with DASH IF recommendations. For such a reason, the final player is no longer based on the Shaka Player, but it employs the Dash.js player as it is the reference player provided by DASH IF. Like the previous release, this new player is also deployed through Node.js. All the software dependencies of the player and their versions are shown in Table 13.

Table 13. Software versions used in DASH Player.

Software and frameworks	Version
NodeJS	16
Dash.JS	5.0.0

Figure 18 shows a demonstration of the final implementation of the DASH Video Player.

[18] https://dashif.org/dash.js/pages/usage/clock-sync.html

Figure 18. DASH Player based on Dash.js.

Finally, both the HTTP Server and DASH Video Player have been containerised through Docker and described through a Helm Chart.

5.3.3 Final hardware

Both the HTTP Server and the DASH Video Player are hardware-agnostic, meaning they do not depend on any specific hardware configuration. They can be deployed as containerised Node.js applications across a wide range of platforms.

It is worth noting that the HTTP Server is responsible for delivering the media content generated by the Remote Renderer, as described in Section 4.3. To streamline deployment in this final release, the HTTP Server has been containerised with the Remote Renderer in a unique Docker container and deployed through a Helm Chart on Kubernetes.

The DASH Video Player is instead provided with a separate Dockerfile for building and its own Helm Chart for deployment. It has finally been tested using the Chrome web browser on a standard laptop and the Browser on the Meta Quest 2 and PICO 4[19] VR headsets.

5.3.4 Evolution compared to the previous release

Table 14 presents the improvements implemented in the final release of the HTTP Server and DASH Video Player, comparing them with their first release.

Table 14. Comparison of the first and final releases of the DASH streaming enabler.

Feature	First release	Final release

[19] https://www.picoxr.com/es/products/pico4e

HTTP Server – UTC synchronisation	Not available	It serves UTC timing information to the Remote Renderer and DASH Video Player for synchronisation purposes
DASH Video Player - Framework	Shaka-player	Dash.js player for better compliance with DASH IF recommendations
DASH Video Player – UTC synchronisation	Not available	It receives the UTC timing information from the HTTP Server

6 MULTI-MODAL SYNCHRONISATION

In the 6G-XR project, device and media synchronisation 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 examines the achievements in clock and media synchronisation mechanisms implemented in 6G-XR to support XR Enablers that operate under time constraints or require time-aware processing.

6.1 CLOCK SYNCHRONISATION

In the 6G-XR project, synchronisation between VNFs, serving as XR Enablers within the computing infrastructure, and end-user applications, operating on UE, constitutes a fundamental requirement for the precise execution of time-sensitive and delay-critical multimedia transmission and interactive XR operations. To meet this requirement, the 6G-XR project adopts two distinct synchronisation mechanisms aimed at enhancing the overall synchronisation performance of the deployed XR Enablers: **Clock synchronisation**, described in this section, and **Media synchronisation**, addressed in Section 6.2.

Clock synchronisation is required for synchronising the device or host where the XR Enablers are deployed and running. To achieve it, the 6G-XR project employed Network Time Protocol (NTP), as it has an ease to use client-server communication architecture that simplifies the synchronisation of both the VM at the Edge locations, where XR Enablers such as SFU, MCU and Remote Renderer are deployed, and the UEs, where the Native Web Players are instead executed. Furthermore, the clock synchronisation enables the accurate tracking of measurements that are performed thanks to the monitoring system described in Section 9.1. To ease access to an absolute clock source, like NTP or an alternative one, the Holo Orchestrator also includes the Clock Manager (Section 7.1), which can serve as a clock reference distributor module. Table 15 resumes the XR Enablers requiring clock synchronisation information and the operations performed with it.

Table 15. Clock synchronisation within XR Enablers.

XR Enabler	Host or device	Clock synchronisation
Remote Renderer (Section 4.3)	Unity-based containerised application running on the WP2 Edge nodes	The remote renderer uses the date/time provided by the Host OS to the Unity environment to apply timestamps to audio and video buffers
Native Player (Section 5.1)	Unity-based standalone application running on the UE	The native player uses the date/time provided by the Host OS to the Unity environment to apply timestamps to audio and video buffers. Moreover, a metric exporter embedded in the native players employs the timestamp to relate the measurement to the exact time it is generated

Web Players (WebRTC in Section 5.2 and DASH in Section 5.3)	Web applications running on the UE	The clock information is necessary to measure the E2E delay of the media communications on the web player. Moreover, metric exporters embedded in the web players employ the timestamp to relate the measurement to the generated exact time
Monitoring system (Section 9.1)	Prometheus time series database running on a WP2 Edge node.	It receives the information from metrics exporters and uses the clock information to store it in real time. Then, it supports both real-time monitoring and post-experiment analysis

For all the XR Enablers operations, the usage of NTP is the best solution due to its lightweight design and requirements. Moreover, it is worth noticing that it is not necessary to have a more accurate synchronisation than NTP, as the degree of accuracy mostly depends on the video frames per second that are processed by the XR Enablers. For instance, even with a high frame rate of 60 frames per second, the minimum delay between video frames is approximately 16 ms (synchronisation in the order of tens of ms), which is an order of magnitude greater than the synchronisation error typically associated with NTP (synchronisation in the order of some ms). Thus, NTP sufficiently meets the timing requirements without introducing any practical limitations.

Finally, the NTP Server employed for the XR Enablers synchronisation is provided by Edge/Cloud infrastructures of the 6G-XR project, like the Clock Manager integrated as part of the Holo-Orchestrator. Thus, all the host and devices are connected to an absolute clock Server, like NTP, to retrieve the synchronisation information and update their internal clock accordingly.

6.2 MEDIA SYNCHRONISATION

Complementary to Clock synchronisation, Media synchronisation is employed to deal with time-dependent operations at the application level, where the multimedia information is processed, as it allows the XR Enablers to be synchronised with a common time source. Thus, Media synchronisation represents an advanced step in achieving temporal alignment across XR Enablers.

In the context of 6G-XR and its UCs, Media synchronisation is essential for the following operations:

- Triggering or activating multimedia features of XR Enabler when needed: Some multimedia functionalities are time-dependent and rely on both clock and media synchronisation. For instance, a DASH player uses the current timestamp (clock timestamp) to determine which media segment should be downloaded at a given moment from the HTTP server (application timestamp).

- Holographic communication operations: XR Enablers employed within holographic communications UCs include advanced operations with are highly sensitive to timing constraints. 6DoF capture and interactions, for example, require real-time responsiveness, as any delay in user interaction can degrade the Quality of Experience (QoE) and may even lead to motion sickness.

- Estimating E2E delays: Accurate measurement of delays across different components in the E2E media pipeline requires precise synchronisation. This enables the correct generation of timestamps for every Media Unit (MU), such as video frames or audio samples.

Ultimately, enabling the operations above ensures consistency in processing multiple media streams simultaneously, such as maintaining audio-video sync to avoid lip-sync issues, and supports a coherent XR experience. This is especially important in shared virtual environments, where actions in the physical world must be accurately reflected in the VR scene for one or more users entering or interacting in real time. Table 16 resumes the XR Enablers and their implemented mechanisms for media synchronisation.

Table 16. Implementation of media synchronisation mechanisms within XR Enablers.

XR Enablers	Media synchronisation	Implemented synchronisation mechanisms
Video capture (Section 3.1) and Video reconstruction (Section 3.2)	Inter-source synchronisation	CoaXPress CXP-12 cables are employed to connect the multiple camera sensors setup and allow the 3D reconstruction module to run locally Wall-clock timestamps to capture frames from each sensor is inserted to allow for an in-sync volumetric reconstruction
SFU (Section 4.1)	Intra-media and inter-media synchronisation	The SFU does not modify timestamps, and it waits until all the packets containing an MU are received to relay them. Thus, it preserves the original timing patterns of MUs for each of the incoming/outgoing streams An Event manager has been added to exchange time-dependent information among XR enablers, like Native players
Native Player (Section 5.1)	Inter-destination synchronisation	Timing information is retrieved from the Clock Manager of the Holo Orchestrator No inter-destination synchronisation is handled to avoid adding delays and affecting the user's experience
Remote Renderer and WebRTC Web Player (Section 5.2)	Intra-media and inter-media synchronisation, inter-destination synchronisation	Timestamps are inserted into each MU, based on the clock of the host, and preserved during media encoding and transmission Video and audio streams are synchronised with transport-level protocols (RTP/RTCP)

		when employing WebRTC such that they can be correctly played at the destination No inter-destination synchronisation is handled to avoid adding delays and affecting the user's experience
Remote Renderer (Section 4.3) and DASH Web Player (Section 5.3)	Intra-media and inter-media synchronisation, inter-destination synchronisation	Timestamps are inserted into each MU, based on the clock of the host, and preserved during media encoding and transmission Video and audio streams are synchronised and multiplexed when employing DASH. Moreover, DASH MPD includes metadata related to timing information such as publish time and current time in UTC format A UTC Timing server has been implemented to provide a correct timestamp to the player such to timely request the media segments to be played at the destination. Additionally, this server also allows inter-destination synchronisation

6.2.1 Final evaluation of media synchronisation

In D3.1, the initial results of media synchronisation testing with the first versions of the XR Enablers were presented. These tests focus on measuring E2E video and audio latency, inter-destination asynchrony, and intra-media synchronisation using a QoS measurement method based on image and audio processing. The tests were conducted in a multi-user XR scenario delivered via WebRTC, employing a Remote Renderer, a WebRTC signalling server, and WebRTC players across different access networks, such as Ethernet, Wi-Fi, and the fifth Generation of cellular technology (5G) Stand Alone. The results indicated that Ethernet provided the lowest E2E latency and inter-destination asynchrony, while Wi-Fi 6 showed higher latencies and significant issues with synchronisation. For further details, refer to the deliverable D3.1 [2] or to the published papers with the complete analysis of the results [6].

In D3.2, further experiments are conducted to measure the delay from the time the image is generated until it is received by the user (E2E). These measurements are performed by means of WebRTC and DASH streaming solutions. Moreover, new streaming protocols based on QUIC are emerging as potential replacements for WebRTC and DASH and offer benefits like connection migration, stream multiplexing and multipath delivery. With the contribution of the REQUIEM project[20] (awarded under Open Call 1 of the 6G-XR project), protocols such as QUIC and Media over QUIC (MoQ) are integrated and tested to provide a wider comparison of streaming protocols for holographic communications.

[20] https://netsoft.gsuite.tmit.bme.hu/projects/requiem

In the experiment, multiple captures are performed to calculate the average latency and its deviation for each protocol. The tests employ the Remote Renderer described in Section 4.3, where the WebRTC solution integrates the Unity Render Streaming, while DASH, QUIC, and MoQ rely on GStreamer integration. All tests were performed over both 5G and Wi-Fi 6 networks to evaluate performance under different network conditions. The obtained results are shown in Table 17.

Table 17. Latency with video stream at 1080P/60FPS/10MBPS

WiFi				
Protocol	**Sender**	**Receiver**	**$Latency_{avg}$ (ms)**	**$Latency_{dev}$ (ms)**
WebRTC	Remote Renderer (Section 4.3)	WebRTC web player (Section 5.2)	82	12
DASH	Remote Renderer (Section 4.3)	DASH web player (Section 5.3)	9399	13
QUIC (RTP)	Remote Renderer (Section 4.3) + OC1 Requiem	GStreamer pipeline from OC1 Requiem	248	2
MoQ	Remote Renderer (Section 4.3) + OC1 Requiem	MoQ player from OC1 Requiem	159	12
5G				
Protocol	**Sender**	**Receiver**	**$Latency_{avg}$ (ms)**	**$Latency_{dev}$ (ms)**
WebRTC	Remote Renderer (Section 4.3)	WebRTC web player (Section 5.2)	129	11
DASH	Remote Renderer (Section 4.3)	DASH web player (Section 5.3)	10227	14
QUIC	Remote Renderer (Section 4.3) + OC1 Requiem	GStreamer pipeline from OC1 Requiem	293	31
MoQ	Remote Renderer (Section 4.3) + OC1 Requiem	MoQ player from OC1 Requiem	194	20

All protocols performed better over Wi-Fi than 5G, with improvements of up to 36% in the case of WebRTC. This protocol, natively implemented in Unity, shows the lowest latency overall, thanks to its optimised rendering and encoding pipeline. QUIC delivers promising results, with latency close to real-

time performance, showing only a 77 ms difference over Wi-Fi and 65 ms over 5G when compared to URS WebRTC. MoQ also performed well, outperforming most protocols and ranking just below URS WebRTC in terms of latency. As expected, DASH shows the highest latency among all protocols, as it relies on HTTP/TCP and is not designed for real-time streaming applications, but rather for on-demand video consumption.

A complete analysis of the obtained results is presented in two conference papers [7],[8].

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 the network user plane, and the AR experience based on the network control plane.

7.1 HOLO-ORCHESTRATOR

7.1.1 Description 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. Figure 19 provides an overview of the main modules and services of the Holo Orchestrator, which are briefly described next:

- User Manager (UM): 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[21]), users need to be logged in the platform before creating/joining a session and then must select a virtual scenario on which the session can be established (e.g., a virtual meeting room, a museum). In the first period, the UM handled semantic information about connected clients, like their nickname (as well as their user representation type and session name, in conjunction with the Session Manager (SM)). In the second period, API REST interfaces have been added to retrieve IP address (and port) for each connected client, so to provide this info to network entities for: (i) best Edge selection (e.g., based on available zones; (ii) enabling Access Traffic Steering-Switching-Splitting (ATSSS) features – from WP4 –for the XR service at the stream level.
- Session Manager: 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 (CM) and the Index/Connection Manager (ConM), to be able to select the most appropriate media function(s) - VNFs - to handle the communications for each session (i.e., SFU, MCU or Remote Renderer), the cloud or edge servers where to instantiate them, and then communicate this information to the involved clients. In the second period, the SM has been extended to support multi-SFU architecture and sessions from the Remote Rendering components.
- Clock Manager: 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 provide a reference to an NTP server. The development of the CM was completed in the first period of the project and thus reported in D3.1 [2].
- Index/Connection Manager: in charge of interfacing the edge orchestration platform (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 has been extended in the second period of the project to implement the end-points

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

for two envisioned Network-as-a-Service (NaaS) APIs: (i) Network-assisted Rate Control (details in D4.3 [9]) to trigger rate adaptations or request Quality on Demand (QoD) to the network (details in D2.3 [10]); (ii) Edge-Cloud APIs (details in D2.3 [10]). It has also been extended to handle session and lifecycle management in sessions where multiple SFUs and Remote Renderers can intervene, and it also interfaces with the Metric Monitoring system (details in Section 9.1) to register alerts and receive notifications from it.

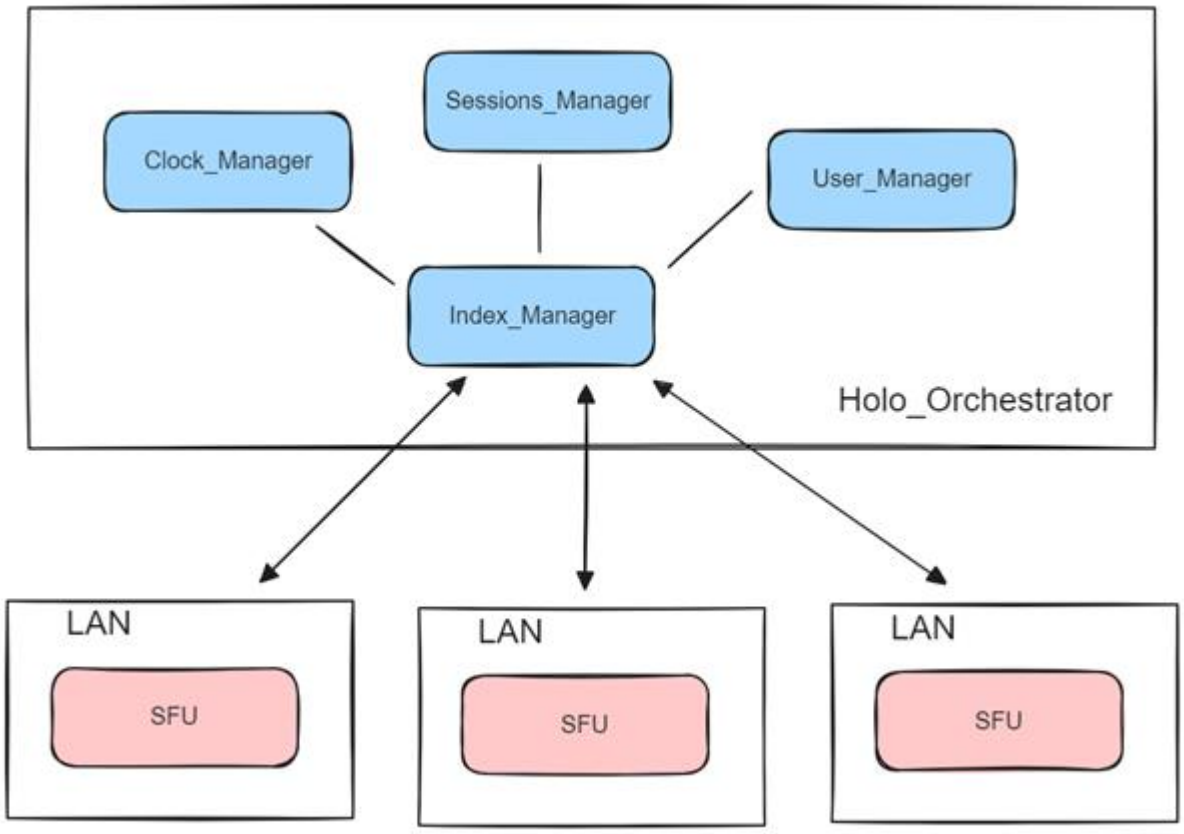


Figure 19. High-level Overview of Holo Orchestrator modules and services.

7.1.2 Final 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.

7.1.3 Final software

The Holo Orchestrator requires the installation of Node.js[22] and socket.io[23] and it has been successfully installed and run on Windows 10 and Linux (Ubuntu 22.04) machines (including VMs on Azure). It has also been virtualised as a Docker container and a Helm chart, which eases its deployment on any

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

[23] https://socket.io/

machine with reasonable resources (e.g., tested on laptops and a PC with AMD Ryzen Threadripper 3970X CPU (3.70 GHz) and 16 GB RAM), including those provided by hyper-scalers.

7.1.4 Evolution compared to the previous release

Table 18 lists and summarizes the features and capabilities of the Holo-Orchestrator, comparing the final released version to the first released one, reported in D3.1 [2].

Table 18. Comparison of the first and final releases of the Holo-Orchestrator.

Feature	First release	Final release
Virtualisation	Docker	Docker and Helm Chart
Information about clients	Just semantic information, like nickname, user representation type, and session name	API REST to retrieve and inform about IP address (and port) for each connected client, so this information can be provided to network entities.
Session Management	Session Management with sessions served by a single SFU	Session and Lifecycle Management with sessions served by more than one SFU, and with intervention of Remote Renderers. New endpoints for two NaaS APIs: (ii) Network-assisted Rate Control to trigger rate adaptations or request QoD to the network; (ii) Edge-Cloud APIs
Metrics	-	It includes features to register Level of Service rules against the Metrics Monitoring Subsystem (e.g., upper delay thresholds for specific clients) and get alerts when these thresholds are reached/surpassed

7.2 IMS SESSION MANAGER

7.2.1 Description of the component

The IMS SM serves as the foundational orchestrator for secure, low-latency, and real-time signalling across distributed components involved in immersive holographic communication experiences.
The IMS SM is a core component responsible for orchestrating session signalling and coordination within an XR streaming distributed infrastructure in immersive holographic communication, particularly between the consumer UE "viewer", and the AR media servers. It facilitates the real-time signalling required to establish, maintain the WebRTC-based holographic communication sessions over the IMS data channel.

Cloud-Based Signalling Architecture
At the core of the signalling infrastructure is a dedicated Signalling Server deployed within a public cloud environment. This server acts as an intermediary between the DCSF, the hologram media processing servers, and the Agent UE "producer". Communication between these entities is performed via bi-directional WebSocket connections, enabling the low-latency exchange of control-plane messages necessary for session negotiation.

WebSocket Endpoints and SDP Exchange
The IMS SM exposes a set of specialised WebSocket endpoints via its integrated HTTPS server:

/sender – for AR media producer devices (Agent UE)

/receiver – for consumer devices (Viewer UE)

/hologram – for holographic rendering and streaming backends

Clients connecting to these endpoints utilise WebRTC SDP offers and answers to negotiate media transmission parameters, including supported codecs, transport configurations (ICE candidates), and encryption details. This signalling exchange establishes peer-to-peer paths for holographic content delivery.

Session Lifecycle and Participant Management
Beyond signalling, the SM also functions as a stateful session controller, handling the lifecycle of user interactions and room-based collaboration spaces. Its responsibilities include:

- Session Creation: Enables producers to instantiate a new session (holographic call or AR interaction).
- Session Join/Leave: Allows viewers to join or disconnect from existing sessions.
- Participant Management: Tracks active participants and synchronises their state within each session.
- Room Cleanup: Detects and automatically frees unused or expired sessions to optimise resource utilisation.
- Media Server Registry: Maintains a dynamic registry of available hologram media processing nodes and assigns them to sessions as needed.

Implementation Details
The SM is implemented in the Go programming language, leveraging its concurrency for efficient WebSocket handling and real-time signalling workloads. It follows the official Standard Go Project Layout (https://github.com/golang-standards/project-layout) to ensure maintainability, modularity, and adherence to industry best practices.

Key architectural considerations include:
- Scalability: Designed to handle thousands of concurrent signalling sessions through lightweight, event-driven handling.
- Extensibility: Built to support future extensions such as identity management, advanced analytics, or QoS.
- Interoperability: Ensures compatibility with IMS-based networks and standard WebRTC signalling conventions.

7.2.2 Final hardware

The IMS SM is deployed as a lightweight Docker container, ensuring portability and easy integration into cloud environments. It runs on Microsoft Azure VM with Ubuntu 22.04 LTS, using standard CPU-based infrastructure without the need for GPUs or specialised hardware. It operates efficiently on general-purpose VMs (2–4 vCPUs, 4–8 GB RAM). All media processing is offloaded to dedicated servers, keeping the SM hardware-agnostic and cost-efficient.

7.2.3 Final software

The IMS SM Docker container encapsulates all necessary runtime components required for session signalling and control. At its core, the container includes an embedded HTTPS server that exposes dedicated WebSocket endpoints used to manage signalling flows between clients and servers.

A key component of the software stack is its integrated WebRTC module, which provides support for Secure Real-time Transport Protocol (SRTP), enabling encrypted media streams for both sending and receiving. The implementation includes a multiplexing decoder/encoder capable of handling audio, video, and data channels over a single transport session, adhering to WebRTC multiplexing standards.

Internally, the WebRTC module has a versatile architecture, with separate components responsible for connection handling, SDP negotiation, session state tracking, and error recovery. The Go-based implementation takes advantage of concurrent WebSocket sessions, ensuring high responsiveness and non-blocking operations under heavy load. This design provides the scalability and resilience required for real-time AR/XR communication environments.

7.2.4 Evolution compared to the previous release

Table 19 summarizes the improvements implemented in the IMS SM, comparing the final released version to the first released one.

Table 19. Comparison of the first and final releases of the IMS Session Manager.

Feature	First release	Final release
Hologram resolution	Basic low resolution used for first tests with limited bandwidth usage	Optimisations of bandwidth usage by hologram 2D/3D data in order to control and adapt the hologram resolution
UX/UI in WebGL	Basic rendering of the hologram in WebGL, with no background and no control	Improved rendering of the hologram in WebGL, with added background, and added control for the user to rotate the angle of view of the hologram
Diagnostics and Troubleshooting	Simple logs about connectivity among participants	Verbose logs about session creation, joining, leaving and freeing have been added in order to troubleshoot potential connectivity issues

8 INFRASTRUCTURE CONFIGURATION

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

8.1 XR APPLICATION TRAFFIC REQUIREMENTS EXTRACTION

8.1.1 Description of the component

This enabler is designed to enhance resource allocation within a Wi-Fi connection, ensuring deterministic QoS for XR traffic by reserving access time to the communication channel. Figure 20 depicts a demo setup of the developed enabler. The enabler implements a TSN gating mechanism at the Wi-Fi interfaces of communicating nodes (i.e., stations and access points), allowing it to coordinate traffic transmission, prevent collision and minimise delays for real-time XR traffic. A key concept of this mechanism is to generate a network-wide schedule that specifies time slots for each node to transmit its real-time traffic (XR stream) and best-effort traffic (BE stream), via Wireless TSN (WTSN) servers and ethernet Network Interface Controllers (NICs). This scheduling process is vital in wireless networks where the communication medium is shared among multiple nodes.

To create these schedules, the enabler must possess details about the real-time traffic, including payload size, cycle time, burstiness, and other relevant parameters. However, traditional XR applications lack awareness of the gating mechanism, which hinders their ability to convey scheduling requirements or adapt to network manager-provided configurations. This enabler includes a second capability for monitoring and profiling of traffic flows. By estimating traffic patterns, the system generates a traffic profile utilised for schedule calculation and optimisation. This comprehensive approach ensures efficient resource utilisation and reliable QoS management within Wi-Fi networks, particularly for XR applications that demand timely data transmission.

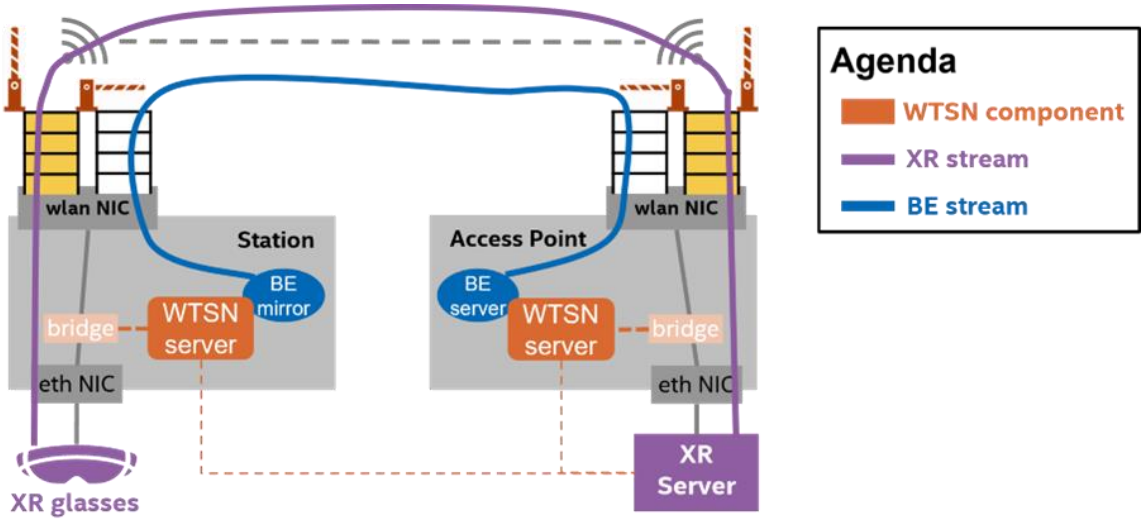


Figure 20. Demo setup enabling an XR-Application together with background traffic.

8.1.2 Final hardware

As depicted in Figure 20, the demo uses two servers equipped with AX210 NICs, supporting Wi-Fi 6, which act as a software access point and a Wi-Fi station, one GPU-equipped server that runs an XR application, and a Meta Quest 3 Headset that allows human interaction with the setup.

8.1.3 Final software

The software used includes:

- On the XR server node: Windows 11 OS, Unity editor to host and run the XR application, Hololight stream SDK[24] to allow XR streaming.
- On the AP and STA servers: Ubuntu 22.04, WTSN server running as a daemon to profile traffic and configure the Wi-Fi link.
- On the glasses: Hololight client application to read data from the sensors, send it to the server side and render the XR experience for the user.

8.1.4 Evolution compared to the previous release

Table 20 summarizes the improvements implemented in this enabler, comparing the final released version to the first released one.

Table 20. Comparison of the first and final releases of the XR application traffic requirements extraction.

Feature	First release	Final release
Hardware	Linux server on the access point side only. The XR Headset is the Wi-Fi client	Linux server (e.g. Ubuntu 22.04) on both sides of the Wi-Fi link (i.e., access point and Wi-Fi station)
Traffic Profiling	On the access point side	On the access point side
Wi-Fi resource configuration	No control over the Wi-Fi link access and resources on the XR client side (unidirectional resource control)	full control of the Wi-Fi link access and resources on both sides (bidirectional resource control)

8.2 MACHINE LEARNING-BASED EDGE CONTINUUM ENABLER

8.2.1 Description of the component

This enabler is designed to optimise the management of application instances deployed at edge nodes by forecasting future resource utilisation scenarios. It employs a time-series AI model that processes input data reflecting the current and historical resource workload status of all edge application instances. Subsequently, it predicts the resource usage for each application instance deployed across

[24] Holo-Light-GmbH/Hololight-Stream-SDK-Trial: Trial of Hololight Stream Software Development Kit.

the nodes. Specifically, the model predicts CPU and Random Access Memory (RAM) resource consumption per application instance.

This enabler comprises various components, as illustrated in Figure 21. The primary component is the Intelligent Edge Application Platform (IEAP)[25], responsible for managing and orchestrating the edge nodes and applications. It incorporates APIs that facilitate real-time data retrieval to construct the dataset. The Data Collector periodically utilises these APIs to acquire new data, formats it for optimal use, and stores it in a TimescaleDB[26] instance. This database is part of NetAnticipate[27], a cloud-native, highly scalable, self-learning AI and machine learning (ML) framework. Within the context of this enabler, NetAnticipate assists data scientists by offering predefined templates for specific scenarios and provides an ML Operations (MLOps) environment to manage the lifecycle of ML models.

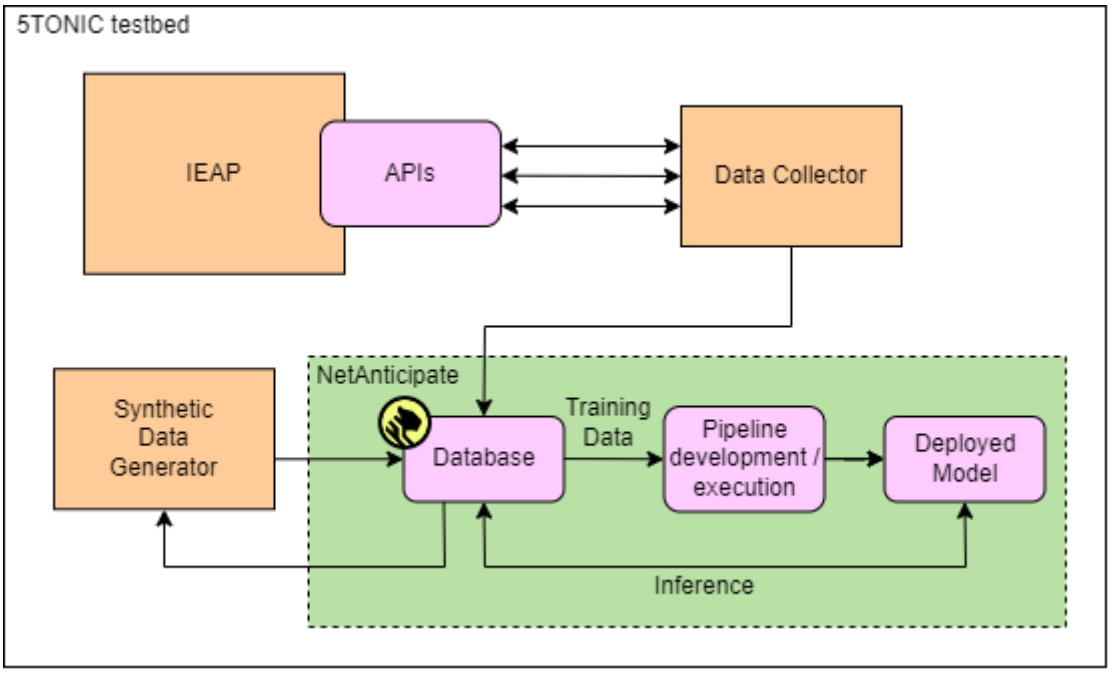


Figure 21. Edge Continuum Enabler high-level diagram.

To forecast resource usage values, a Long-Short Term Memory (LSTM) neural network has been employed, as it is a widely recognised model for time-series prediction due to its capacity to learn and retain long-term temporal dependencies. Although the LSTM model demonstrates sufficient generalisation capabilities across various applications, to validate the functionality of the enabler, three distinct applications are deployed on edge nodes. These applications represent diverse operational characteristics and are integrated with a dedicated API to enable dynamic control over resource utilisation, specifically, the ability to programmatically increase or decrease their CPU and RAM consumption on demand. Each application operates in one of two states, *idle* or *demanding*, with the demanding state induced through application-specific mechanisms:

[25] https://www.researchgate.net/figure/Capgeminis-Intelligent-Edge-Application-Platform_fig1_372867069

[26] https://docs.tigerdata.com/#TimescaleDB

[27] https://www.capgemini.com/wp-content/uploads/2023/03/NetAnticipate_-brochure_Mar-23.pdf

- FFmpeg[28]: An open-source multimedia framework. The *demanding* state is triggered by activating a multimedia pipeline that encodes a video stream.
- DeepStack[29]: An AI server hosting pre-trained models. The *demanding* state is induced by continuously performing inference tasks on two distinct images in parallel.
- Stress-ng[30]: A Unix-based stress testing utility. The *demanding* state is activated via specific input parameters during execution.

The *idle* state for each application is restored by terminating the associated processes, effectively halting resource-intensive operations.

To orchestrate the transitions between the two states, an auxiliary custom application named Workloads Generator was deployed on a separate VM within the 5TONIC testbed. This application autonomously manages the two states of the aforementioned applications to produce a structured time series pattern suitable for model training. Specifically, it generates two random timestamps daily—one between 08:00 and 09:30, and another between 17:00 and 18:30—for each application. At the first timestamp, the application is placed under the *demanding* state; at the second one, the state is changed to *idle*. This routine is executed only on weekdays, with weekends reserved for operations in the *idle* state, thereby simulating realistic workday usage patterns.

Nevertheless, to train and evaluate the LSTM model effectively, a large and representative dataset is required, as relying solely on real-time data collection over extended periods is deemed impractical. Consequently, a Synthetic Data Generator is introduced. This tool extrapolates one year of historical data based on a single day of real measurements collected by the Data Collector component. The differences between real and synthetic data are shown in Figure 22 and Figure 23. Although divergences are observed at finer levels of granularity, the overall trend remains consistent when analysed at a broader scale, as depicted in Figure 24, making it feasible for the AI/ML model to learn the overall pattern.

These components—Workloads Generator, Synthetic Data Generator, and Data Collector—are all custom-developed Python applications.

[28] https://ffmpeg.org/

[29] https://deepstack.readthedocs.io/en/latest/

[30] https://github.com/ColinIanKing/stress-ng

Figure 22. Real pattern for a weekday from 09:05 to 22:00.

Figure 23. Synthetic pattern for one weekday from 09:05 to 22:00.

Figure 24. Synthetic pattern for an entire week from 23-05-2025 09:05 to 30-05-2025 22:00.

Following the training phase, the LSTM model was evaluated using real data collected from applications deployed on edge nodes. As illustrated in Figure 25, the values forecasted by the model are plotted alongside actual measurements for the edge node located in Madrid. The results demonstrate that the model successfully captures the underlying temporal patterns in the data, accurately predicting transitions between demanding and idle states for each application.

Figure 25. Forecasted values against real data at Madrid node from 23-05-2025 09:05 to 30-05-2025 22:00.

In Figure 26, predictions are compared against synthetic data. While the dataset was also used during training—making it less suitable for rigorous performance validation—it nonetheless confirms that the model has learned key behavioural patterns, such as the absence of demanding events during weekends.

Figure 26. Forecasted values against synthetic data at Madrid node from 23-05-2025 09:05 to 30-05-2025 22:00.

It is important to note that some forecasted values exhibit slight temporal shifts relative to the actual events, visible in both Figure 25 and Figure 26. This discrepancy arises from the inherent randomness in the start and end times of demanding periods in both real and synthetic datasets. The model demonstrates sufficient accuracy in identifying *demanding* and *idle* intervals. This predictive capability can be leveraged to classify application states using predefined thresholds, enabling proactive resource management and scheduling.

To address the validity of the model under real-world conditions, Figure 25 illustrated a direct comparison between predictions and empirically observed data was conducted at the edge node located in Madrid. The real measurements were generated through the controlled execution of three applications—FFmpeg, DeepStack, and Stress-ng—whose operating states were dynamically orchestrated by the Workloads Generator. This auxiliary tool automatically switches each application between idle and demanding states at randomized times within predefined weekday intervals, effectively simulating realistic workday usage patterns. This setup enabled the creation of a representative validation scenario in which the ability of the model to anticipate transitions in resource consumption could be rigorously tested. The results demonstrate that the LSTM model accurately forecasts these state changes, capturing, for example, increases in CPU usage during active phases and decreases during idle periods. This predictive capability confirms that the model has effectively learned the relevant temporal patterns and that its performance is sufficiently reliable to support proactive orchestration strategies, such as dynamic scaling or application migration at the network edge.

8.2.2 Final hardware

The final hardware infrastructure comprises four distinct bare-metal servers, each dedicated to hosting specific software components. These machines, illustrated at the hardware layer in Figure 27, have the following configurations:

- LEG15COMP1: A high-performance server featuring 128 GB of RAM, 1.2 TB of storage, and 40 CPU cores.
- LEG15COMP2: Identical in configuration to LEG15COMP1, with the addition of an NVIDIA Tesla P4 GPU, enabling hardware acceleration for AI and ML workloads.
- LEG15COMP4: While offering the same CPU capabilities (40 logical cores), this server is more resource-constrained in terms of memory and storage, equipped with 32 GB of RAM and 100 GB of disk space.
- LEG15UTILS1: A utility server provisioned with 64 GB of RAM, 1 TB of storage, and 4 logical CPU cores, suitable for lightweight or auxiliary services.

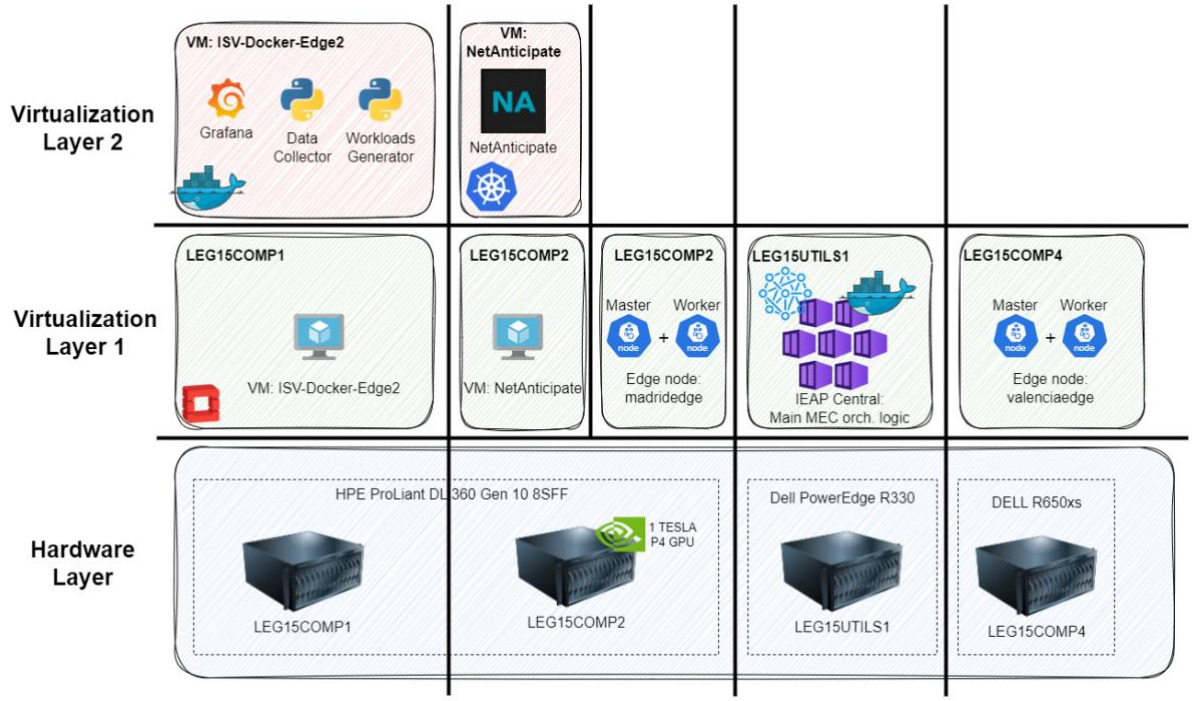


Figure 27. Final hardware and software.

8.2.3 Final software

The complete software stack is illustrated in Figure 27, specifically across Virtualisation Layers 1 and 2. The Python-based applications—Data Collector and Workloads Generator—which are integral to the enabler functionality, have been previously described in Section 8.2.1.

The deployment details across the infrastructure are as follows:

- On LEG15COMP1, within the VM ISV-Docker-Edge2, a Grafana instance is deployed. This Docker-based application is responsible for visualising the various metrics and graphs referenced in Section 8.2.1.
- The NetAnticipate framework is deployed on a Kubernetes cluster hosted within a VM of the same name, running on LEG15COMP2. Additionally, this bare-metal server also hosts a second Kubernetes cluster that functions as the Madrid edge node.
- The Valencia edge node is deployed on LEG15COMP4, also using Kubernetes for orchestration.
- The centralised IEAP logic, which manages both edge nodes and their associated applications, is deployed as a set of Docker containers on LEG15UTILS1.
- Although not shown in Figure 27 due to its one-time execution, the Synthetic Data Generator has been developed and executed locally.

8.2.4 Evolution compared to the previous release

The ML-based edge continuum enabler encompasses both the Scalability Enabler and the Edge Continuum Enabler, as previously introduced in Deliverable D3.1, the implementation of which had not yet begun at that time. Thus, this enabler has been developed during the second period of WP3 to analyse application-level consumption data to generate predictive insights into future system states. These insights can subsequently inform proactive orchestration strategies, such as migration or scaling decisions, that are executed transparently without impacting the end-user experience.

9 KPI AND TELEMETRY

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

9.1 MONITORING SYSTEM

9.1.1 Description of the component

This section describes the modules developed for collecting and displaying metrics from key components of the developed XR Enablers, including the Native player, SFU, MCU, Holo Orchestrator and Web Players.

In the first project period, as reported in D3.1 [2], the system was designed and implemented by using widely adopted components for such purposes, like Prometheus[31] and Grafana[32]. Its high-level architecture is sketched in Figure 28, and its main components are:

- HoloMIT SDK for Unity: it provides relevant metrics of the Native Player, both as producer and consumer of XR content.
- Prometheus Push Gateway: it enables the connection between the HoloMIT SDK for Unity and Prometheus for the reception and storage of metrics.
- Prometheus: toolset for alerting and monitoring, gathering metrics and storing them in a time-series database from several sources, including the Push Gateway. It further offers querying tools for retrieving and examining metrics data.
- Grafana: platform for analytics and visualisation establishing a data source connection with Prometheus and pulls metrics data for display. It gives the ability to design dashboards for data analysis, tracking system performance, and visualising metrics trends.

[31] https://prometheus.io/

[32] https://grafana.com/

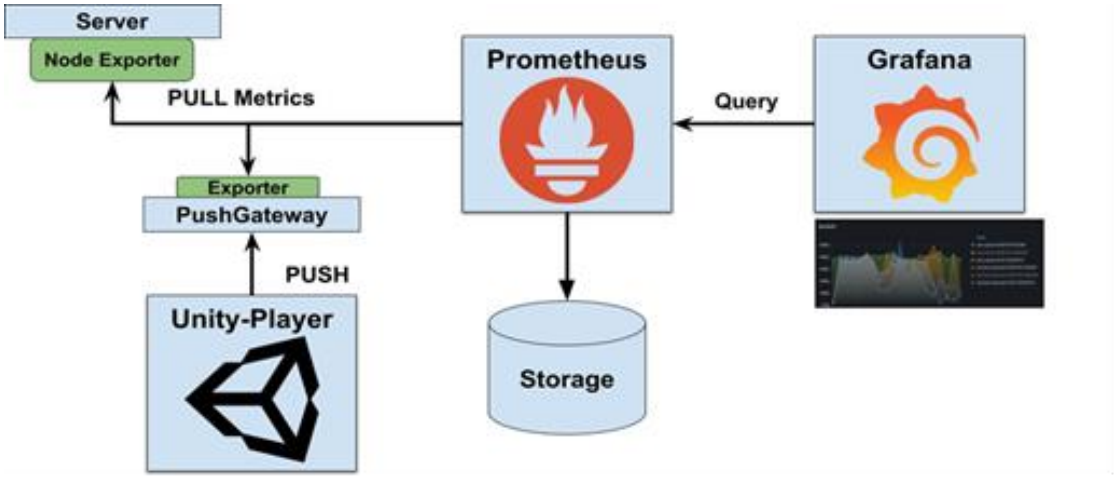


Figure 28. High-level architecture of the metrics measurement and registration system.

The overall workflow for metrics registration and visualisation is summarised as follows:

- The Native player generates its internal metrics (other XR enablers such as SFU, MCU, Holo Orchestrator and Web Players work similarly).
- The configured metrics are pushed to the Prometheus Push Gateway.
- Prometheus periodically pulls metrics data from the Push Gateway and stores it in its time series database.
- Grafana connects to Prometheus as a data source and retrieves metrics data.
- Users access Grafana to visualise metrics data through custom dashboards and panels.

In the second period of the project, the metrics monitoring system has been evolved and extended to be able to register Level of Service (LoS) rules from the Holo Orchestrator and to trigger alerts to the Holo Orchestrator when such LoS rules are violated. Figure 29 provides a high-level overview of the developed workflow, while Figure 30 provides more details about the three newly added components:

- AlertManager: it retrieves the alerts sent by the Prometheus instance to be able to start the pipeline of action against that alert.
- AlertManager2Kafka: it interfaces with the AlertManager to connect to a Kafka bus.
- Kafka bus: it retains messages on a publication/subscription basis; another module can connect to the Kafka bus to check the alerts sent by the AlertManager and perform a corrective action.

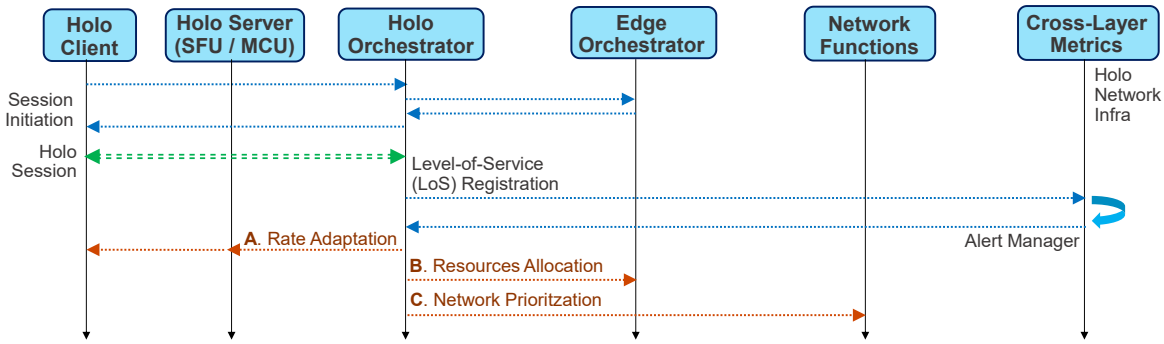


Figure 29. High-level workflow between the Holo Orchestrator and metrics monitoring sub-system.

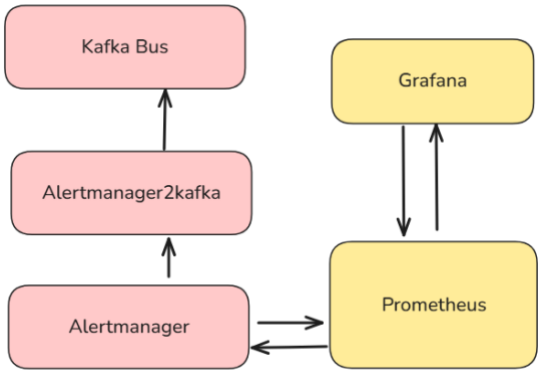


Figure 30. New modules of the metrics monitoring sub-system to notify about alerts.

9.1.2 Key Performance Indicators

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

Native Player:

Table 21. 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 22. 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 23. KPIs of the MCU component.

Metrics	Description
Incoming frame latency (ms)	Latency for each incoming frame to the MCU
Input fps (#)	Number of frames/sec 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/sec 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/sec delivered by the MCU to each player

WebRTC streaming (Remote Renderer to WebRTC Web Player):

Table 24. KPIs for WebRTC streaming from the Remote Renderer to the WebRTC Web player.

Flow	Metric	Description
Audio	timestamp (ms)	Current timestamp
	jitter (ms)	RTP packet jitter for this media flow
	packetsLost/sec (packets/s)	Total number of RTP packets lost per second for this media flow
	packetsReceived/sec (packets/s)	Total number of RTP packets received per second for this media flow, it includes retransmissions
	bytesReceived/sec (bytes/s)	Total number of bytes received per second for this media flow, it includes retransmissions
	totalSamplesDuration	Represents the total duration in seconds of all samples that have been received.
	Audio Level	Represents the audio level of the media source.
Video	timestamp (ms)	Current timestamp
	jitter (ms)	RTP packet jitter for this media flow
	jitterBufferDelay (s)	Time taken for each frame of video from the time the first packet is received by the buffer of the player to the time it is output
	packetLost/sec (packets/s)	Total number of RTP packets lost per second for this media flow
	packetsReceived/sec (packets/s)	Total number of RTP packets received per second for this media flow, it includes retransmissions
	bytesReceived/sec (bytes/s)	Total number of bytes received per second for this media flow, it includes retransmissions
	totalFreezesDuration (s)	Total time that the video has been stopped in the same frame without being able to continue playback
	framesReceived/sec (fps)	Represents the number of full frames received per sec
	frameHeight (pixels)	Represents the height of the last received frame
	frameWidth (pixels)	Represents the width of the last received frame
	framesDecoded/sec (fps)	Represents the total number of frames successfully decoded

DASH streaming (Remote Renderer to DASH Web Player):

Table 25. KPIs for DASH streaming from the Remote Renderer to the DASH Web player.

Metric	Description
audio repIndex	The current audio representation index
audio bitrate (kbit/s)	The encoding bitrate of the current audio representation
audio bufferLevel (s)	Current audio buffer level
video repIndex	The current video representation index
video bitrate (kbit/s)	The encoding bitrate of the current video representation
video bufferLevel (s)	Current video buffer level
video framerate (fps)	Frame rate of the current video representation
video frameHeight (pixels)	Height of the current video representation
video frameWidth (pixels)	Width of the current video representation
liveLatency (s)	Live latency during playback

averageThroughput (kbit/s)	Average estimated network throughput

Holo Orchestrator:

Table 26. 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 MCUs / session (#)	Number of active MCUs for each running session

ML-Based Edge Continuum enabler:

Table 27. KPIs for the ML-Based Edge Continuum enabler

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

As a proof of concept, Figure 31 shows some screenshots captured by the Grafana dashboard reporting on key metrics captured at XR Enablers such as the native Unity player and the SFU for a real running holographic communication session.

Figure 31. Grafana dashboards showing collected metrics from XR Enablers.

9.1.3 Final hardware

This monitoring system does not impose strict computing (CPU or GPU) or memory requirements for installation and execution. However, the monitored metrics require storage space to allocate the Prometheus database for the metrics that will be captured during the testing of 6G-XR use cases. Storage capacity should be in the order of a few tens of gigabytes (GBs) for such a purpose.

9.1.4 Final 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[33] and Grafana[34].

9.1.5 Evolution compared to the previous release

Table 28 lists and summarizes the features and capabilities of the Monitoring System, comparing the final released version to the first released one, reported in D3.1 [2].

Table 28. Comparison of the first and final releases of the Monitoring System.

Feature	First release	Final release
Deployment/Virtualisation	Deployment on machines on bare-metal	Deployment using Docker Compose or Helm Charts

[33] https://prometheus.io/

[34] https://grafana.com/

Aggregation of metrics	Not available	Prometheus from different XR enablers can be aggregated within the same metrics monitoring sub-system
Alerting Sub-system	Not available	Development of an alerting sub-system and associated workflow to register LoS rules and get notifications when such rules are violated

9.1.6 Dataset exporters

The final release of the monitoring system is capable of measuring, reporting, and visualising key performance and operational metrics in near real-time. Furthermore, it incorporates advanced functionalities such as alert management to enhance system supervision.

By leveraging Prometheus and the Prometheus Push Gateway, the system also facilitates the seamless generation and export of datasets to databases or structured textual formats. However, the actual dataset collection will be conducted during the validation and testing phases of the use cases defined in WP6. Then, a comprehensive description of the collected datasets will be included in the corresponding WP6 deliverable (D6.1 “Holographic Use Case integration and validation and KPVI assessment”).

10 SUMMARY

This deliverable details the XR Enablers developed within WP3. The objective of these XR Enablers is to establish the necessary E2E multimedia pipeline to support the deployment of three UCs:

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

UC1 and UC2 leverage the XR Enablers integrated with the network user plane to deliver VR experiences, whereas UC3 focuses on the evolution of the IMS system as a key component of the network control plane, enabling AR services.

To support these use cases, the XR Enablers cover a broad set of capabilities:

- Volumetric capture through multi-sensor setups and 3D reconstruction.
- Cloud/edge-enabled multimedia processing to ensure scalability and broaden access for end users.
- Multiprotocol media delivery compatible with heterogeneous devices.
- Clock and media synchronisation mechanisms.
- Session management and media orchestration functionalities.
- Infrastructure configuration tools to optimise resource allocation for media processing tasks.
- A KPI monitoring system focused on multimedia performance metrics.

The final versions of the developed XR Enablers are described in this document, including the hardware and software components used for their implementation.

The XR Enablers have been integrated into the computing infrastructure of the 6G-XR South Node test facilities, namely 5TONIC (Madrid, Spain) and i2CAT (Barcelona, Spain), where the three UCs will be tested.

11 REFERENCES

[1] 6G-XR, "Requirements and use case specifications," Deliverable D1.1, September 2023. Available at: https://6g-xr.eu/deliverables/

[2] 6G-XR, "Initial versions of XR enablers," Deliverable D3.1, June 2024. Available at: https://6g-xr.eu/deliverables/

[3] Fernández, Sergi, et al. "Multiparty holomeetings: Toward a new era of low-cost volumetric holographic meetings in virtual reality." IEEE Access, volume 10. 2022. Doi: https://doi.org/10.1109/ACCESS.2022.3196285

[4] 6G-XR, "Orchestration, AI techniques, End-to-end slicing and Signalling for the core enablers – design," Deliverable D2.1, February 2024. Available at: https://6g-xr.eu/deliverables/

[5] Fernández, 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. Doi: https://doi.org/10.1145/3581783.361377

[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. Doi: https://doi.org/10.1109/BMSB62888.2024.10608249

[7] Mejías, Daniel, et al. "Streaming Remote rendering services: Comparison of QUIC-based and WebRTC Protocols." 2025 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB). IEEE. To appear online in 2025.

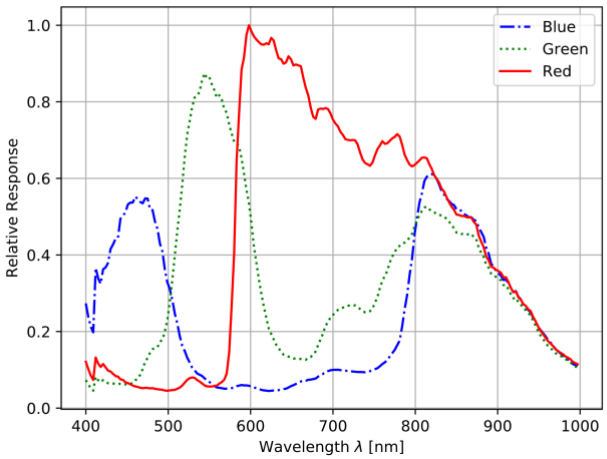
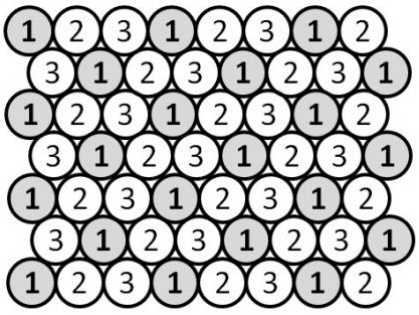
[8] Mejías, Daniel, et al. "Remote Rendering for Virtual Reality: performance comparison of multimedia frameworks and protocols." 2025 IEEE International Mediterranean Conference on Communications and Networking (MeditCom). IEEE. To appear online in 2025.

[9] 6G-XR, "Final deployment of beyond 5G RAN, core, and open-source networks, disruptive RAN technologies and trial controller," Deliverable D4.3, June 2025. Available at: https://6g-xr.eu/deliverables/

[10] 6G-XR, "Core and Edge enablers delivery result", Deliverable D2.3, May 2025. Available at: https://6g-xr.eu/deliverables/

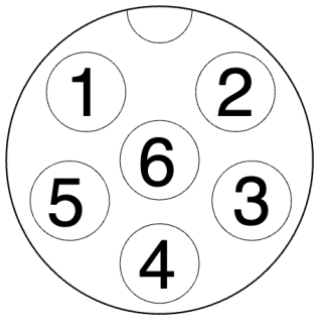
APPENDIX A - R32 LIGHT-FIELD CAMERA FACTSHEET

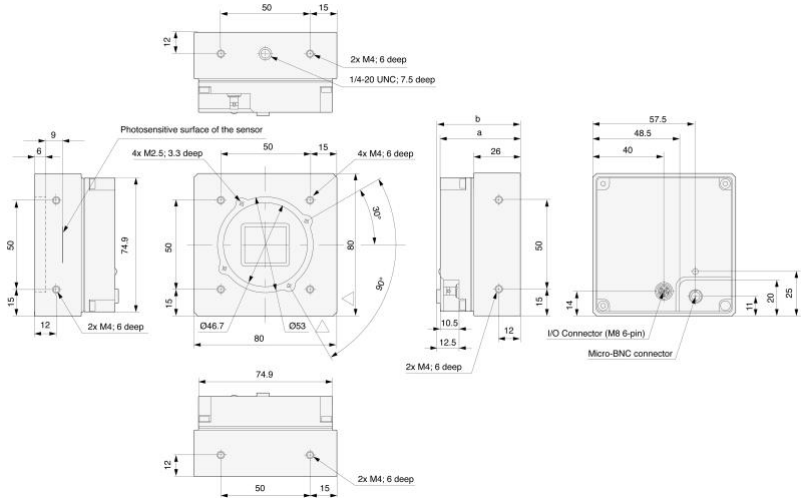
R32v2 factsheet	
	
Sensor	
Image sensor	Onsemi XGS 32000
Lateral resolution (H x V)	6560 x 4948 pixel^2
Lateral resolution (MegaPixel)	32.4 MP
Effective lat. resolution (H x V)	3280 x 2474 pixel^2
Effective lat. resolution (MegaPixel)	8.1 MP
Active area	21.0 x 15.8 mm^2
Pixel length	3.2 µm
Shutter type	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	L1-D500-A010-Vae-VI
Light-field mode	Galilean multi focused plenoptic 2.0
Micro lens types	3
Geometry	hexagonal
Layout	
Aperture	f/3.8
Lens pitch	500 µm
Camera interface	CoaXPress (2.5–12.5 Gbps); Micro-BNC (HD-BNC) connector

<table>
<tr><td>Camera interface bandwidth</td><td colspan="4">CXP Speed</td><td>Bandwidth</td><td>Cable length</td><td>fps @ full res.</td></tr>
<tr><td rowspan="6"></td><td colspan="4">CXP-2</td><td>2.5 Gbps</td><td>180m</td><td>7</td></tr>
<tr><td colspan="4">CXP-3</td><td>3.125 Gbps</td><td>100m</td><td>9</td></tr>
<tr><td colspan="4">CXP-5</td><td>5 Gbps</td><td>60m</td><td>14</td></tr>
<tr><td colspan="4">CXP-6</td><td>6.25 Gbps</td><td>40m</td><td>18</td></tr>
<tr><td colspan="4">CXP-10</td><td>10 Gbps</td><td>40m</td><td>29</td></tr>
<tr><td colspan="4">CXP-12</td><td>12.5 Gbps</td><td>30m</td><td>36</td></tr>
<tr><td>Power</td><td colspan="7">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.</td></tr>
<tr><td>I/O</td><td colspan="7">M8 6-pin female connector (IEC 61076-2-104)

Recommended mating connector: M8 6-pin male</td></tr>
<tr><td>Pinout</td><td colspan="2">Pin</td><td colspan="2">Line</td><td colspan="2">Function</td></tr>
<tr><td rowspan="6"> 
</td><td colspan="2">1</td><td colspan="2">-</td><td colspan="2">24 VDC power</td></tr>
<tr><td colspan="2">2</td><td colspan="2">Line 1</td><td colspan="2">Opto-coupled I/O input</td></tr>
<tr><td colspan="2">3</td><td colspan="2">-</td><td colspan="2">Ground for opto-coupled I/O</td></tr>
<tr><td colspan="2">4</td><td colspan="2">Line 2</td><td colspan="2">General purpose I/O (GPIO)</td></tr>
<tr><td colspan="2">5</td><td colspan="2">Line 3</td><td colspan="2">General purpose I/O (GPIO)</td></tr>
<tr><td colspan="2">6</td><td colspan="2">-</td><td colspan="2">Ground for camera power and General Purpose I/O (GPIO)</td></tr>
<tr><td>Size (L x W x H)</td><td colspan="7">50 x 80 x 80 mm^3</td></tr>
</table>

	
Weight	550 g
Mount	Custom, thermal decoupling of lens and camera body
OEM	Basler Lens
Type	Based on F-S35-3528-45M-S-SD
Focal length	35 mm
Aperture*	f/1.8
Focus range*	0.2 - 3.5m
Angle of View (on R32)	Horizontal: 33° Vertical: 25°