Distributed Implementation of eXtended Reality Technologies over 5G Networks

---

Author:

Diego González Morín

A dissertation submitted in partial fulfillment of the requirements for the interuniversity degree of Doctor of Philosophy in

Multimedia and Communications

Universidad Carlos III de Madrid

Advisors:

Pablo Pérez

Ana García Armada

Tutor:

Ana García Armada

February, 2023

A Cris y Fiko.
A mis padres.
A Guille.
A la familia que se elige.

"La suerte solo ayuda a aquellos a los que pilla preparados"
Papá

This work has received funding from the European Union (EU) Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie ETN TeamUp5G, grant agreement No. 813391.

Acknowledgements

En primer lugar, el mismo puesto que ocupáis en mi vida, quiero dar las gracias a Cris y a Fiko. Habéis sido un pilar fundamental, no solo en la consecución de estos 4 años de investigación, si no tambien en otras tantas partes de mi vida igual de importantes. No todos los pasos han sido fáciles, pero siempre habéis conseguido sacarme no solo una sonrisa, si no también lo mejor de mi.

Papá y mamá, siempre habéis sido y seréis mi ejemplo. Lo mejor de mi me lo habeis enseñado vosotros. A trabajar duro, a ser responsable, a ser buena persona. De manera literal y figurada, no estaría aquí, ni habría conseguido esto, sin vosotros, sin vuestro apoyo incondicional, y sin vuestra confianza ciega en mis capacidades. Gracias Guille, eres el reflejo perfecto de todo lo bueno que tienen papá y mamá. De ti aprendo a cultivar mi inteligencia emocional, mi empatía, a ver el mundo de una manera mucho menos egoísta. A frenar y ver las cosas con calma, con una perspectiva nueva.

Moran, Ceci, Andrés, Ele, Burri, Pepelu, Garmen y muchos otros. La familia que se elige. Tenéis un hueco enorme en mi vida y, al igual que de la familia que elijo, he aprendido y aprendo mucho de vosotros. Aportáis alegría, siempre me habéis puesto todo facil. Habéis conseguido que no hayan habido momentos malos.

Gracias Pablo, estos cuatro años has sido un supervisor y compañero excelente. En ningún momento has dejado de confiar en mi y siempre me has dado libertad absoluta para ir, venir, probar, rehacer. Es una increible ventaja y placer tenerte como referencia. También quiero dar las gracias a Ana, en todo momento has sacado los momentos que hiciesen falta para ayudarnos. Hemos aprendido muchísimo de ti, y te has convertido en una referencia personal y profesional por la pasión y el compromiso que pones en todo lo que haces. No olvidarme de Manu, este trabajo tiene una parte de ti, de tu ayuda con los entregables, con las publicaciones, con la burocracia. Siempre con una sonrisa y a la primera.

A todo el equipo de Nokia XR Labs. Es decir, a ti, y a ti, y a ti... He aprendido algo de cada uno de vosotros. Os he visto más que a mi familia, y conseguís que nuestro lugar de trabajo se convierta en una segunda casa. Especial agradecimiento a Ester y Alvaro. Ester, siempre trabajadora, responsable, y sin perder la sonrisa y la buena actitud. Alvaro, el lider de nuestro pequeño manicomio que consigue encajar cual maestro de relojería todas las piezas tan dispares de nuestro equipo.

Last, but not least, thank you Periklis, Athanasios and Daniele. It has been a real pleasure to get to know Greece along your company. I was warmly welcome from the very first moment, and the months spent there are an absolute highlight from the past four years.

Published and submitted content

All the material from all the sources included in this thesis are singled out with typographic means and explicit references.

Journal Articles

Publications

- **D. Gonzalez Morin**, E. Gonzalez-Sosa, P. Perez, and A. Villegas "Full Body Video-Based Self-Avatars for Mixed Reality: from E2E System to User Study", accepted in Springer Virtual Reality.
  Contribution: Main co-author, contributing equally as E. Gonzalez-Sosa.
  URL[Preprint]: https://arxiv.org/abs/2208.12639
  This is fully included in Chapter 4.

Submissions

- **D. Gonzalez Morin**, D. Medda, A. Iossifides, P. Chatzimisios, A. Garcia Armada, A. Villegas, P. Perez "An eXtended Reality Offloading IP Traffic Dataset and Models", submitted to IEEE Transactions on Mobile Computing.
  Contribution: Main author.
  URL[Preprint]: https://arxiv.org/abs/2301.11217
  This is fully included in Chapter 6.

- E. Gonzalez-Sosa, A. Gajic, **D. Gonzalez Morin**, G. Robledo, P. Perez, A. Villegas "Real Time Egocentric Segmentation for Video-self Avatar in Mixed Reality", submitted to IEEE Transactions on Visualization and Computer Graphics.
  Contribution: Supporting co-author. Design decissions and data pre-processing. Co-supervisor of Andrija Gajic's master thesis.
  URL[Preprint]: https://arxiv.org/abs/2207.01296
  This is mentioned in Chapter 4.

Magazine Articles

Publications

- **D. Gonzalez Morin**, P. Pérez and A. Garcia Armada, "Toward the Distributed Implementation of Immersive Augmented Reality Architectures on 5G Networks," in IEEE Communications Magazine, vol. 60, no. 2, pp. 46-52, February 2022.
  DOI: 10.1109/MCOM.001.2100225
  Contribution: Main contributor.
  URL: https://ieeexplore.ieee.org/document/9722829
  This is fully included in Chapter 2.

- **D. Gonzalez Morin**, M.J. García Morales, P. Pérez and A. Garcia Armada and A. Villegas, "FikoRE: 5G and Beyond RAN Emulator for Application Level Experimentation and Prototyping," accepted in IEEE Network Magazine.
  Contribution: Main contributor.
  URL[Preprint]: https://arxiv.org/abs/2204.04290
  This is fully included in Chapter 5.

Conference Articles

Publications

- **D. Gonzalez. Morín**, A. Garcia Armada and P. Pérez, "Cutting the Cord: Key Performance Indicators for the Future of Wireless Virtual Reality Applications" 12th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP), 2020, pp. 1-6.
  Contribution: Main author and contributor.
  DOI: 10.1109/CSNDSP49049.2020.9249445 **URL:** https://ieeexplore.ieee.org/document/9249445
  This is mentioned in Chapter 2.

- E. Gonzalez-Sosa, A. Gajic, **D. Gonzalez Morin**, M. Escudero-Viñolo, A. Villegas "Egocentric Human Segmentation for Mixed Reality" EPIC Workshop at IEEE Computer Vision and Pattern Recognition Conference (CVPR), 2020.
  Contribution: Supporting co-author. Design decissions and data preprocessing. Co-supervisor of Andrija Gajic's master thesis.
  URL: https://arxiv.org/abs/2005.12074
  This is mentioned in Chapter 4.

- E. González-Sosa, P. Perez, **D. Gonzalez Morin** and A. Villegas, "Subjective Evaluation of Egocentric Human Segmentation for Mixed Reality" IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR), 2021, pp. 232-236.
  Contribution: Supporting co-author. Supporting co-author. Design decissions and data pre-processing. Co-supervisor of Andrija Gajic's master thesis.
  DOI: 10.1109/AIVR52153.2021.00053
  URL: https://ieeexplore.ieee.org/abstract/document/9644385
  This is mentioned in Chapter 4.

- E. Gonzalez-Sosa, G. Robledo, **D. Gonzalez Morin**, P. Perez and A. Villegas, "Real Time Egocentric Object Segmentation for Mixed Reality: THU-READ Labeling and Benchmarking Results" IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), 2022, pp. 195-202.
  Contribution: Supporting co-author. Design decissions and data pre-processing.
  DOI: 10.1109/VRW55335.2022.00048
  URL: https://ieeexplore.ieee.org/abstract/document/9757563
  This is mentioned in Chapter 4.

- **D. Gonzalez Morin**, E. Gonzalez–Sosa, P. Perez–Garcia and A. Villegas, "Bringing Real Body as Self-Avatar into Mixed Reality: A Gamified Volcano Experience," 2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), 2022, pp. 794-795.
  Contribution: Main co-author, contributing equally as E. Gonzalez-Sosa.
  DOI: 10.1109/VRW55335.2022.00248
  URL: https://ieeexplore.ieee.org/abstract/document/9757562
  This is mentioned in Chapter 4.

- **D. Gonzalez Morin**, F. Pereira, E. González, P. Pérez and A. Villegas, "Democratic Video Pass-Through for Commercial Virtual Reality Devices", IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), 2022, pp. 790-791.
  Contribution: Main author and contributor.
  DOI: 10.1109/VRW55335.2022.00246
  URL: https://ieeexplore.ieee.org/abstract/document/9757394
  This is mentioned in Chapter 4.

- **D. Gonzalez Morin**, M.J. Lopez Morales, P. Pérez, and A. Villegas. "TCP-Based Distributed Offloading Architecture for the Future of Untethered Immersive Experiences in Wireless Networks", in ACM International Conference on Interactive Media Experiences, 2022.
  Contribution: Main author and contributor.
  DOI: 10.1145/3505284.3529963

URL: https://dl.acm.org/doi/abs/10.1145/3505284.3529963
This is mentioned in Chapter 3.

Patents

- Accepted patent as a co-author related to immersive 360 video technologies.
- Two accepted patents as co-author related to network orchestration for EXtended Reality (XR) Quality of Experience estimation and reporting.

Other research merits

Conference Articles

Publications

- R. Kachach, S. Morcuende, **D. Gonzalez Morin**, P. Perez, E. Gonzalez-Sosa, F. Pereira and A. Villegas, "The Owl: Immersive Telepresence Communication for Hybrid Conferences" 2021 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct), 2021, pp. 451-452.
  Contribution: Supporting co-author. Implemented several functionalities of the described prototype.
  DOI:10.1109/ISMAR-Adjunct54149.2021.00104
  URL: https://ieeexplore.ieee.org/abstract/document/9585789

- R. Kachach, **D. Gonzalez Morin**, F. Pereira, P. Perez, I. Benito, J. Ruiz, E. Gonzalez and A. Villegas, "A Multi-Peer, Low Cost Immersive Communication System for Pandemic Times", 2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), 2021, pp. 691-692.
  Contribution: Supporting co-author. Wrote the article. Implemented several functionalities of the described prototype.
  DOI:10.1109/VRW52623.2021.00228
  URL: https://ieeexplore.ieee.org/abstract/document/9419314/

- M. J. Lopez Morales, D. A. Urquiza, **D. Gonzalez Morin** et al., "MOOC on "Ultra-dense Networks for 5G and its Evolution": Challenges and Lessons Learned" 31st Annual Conference of the European Association for Education in Electrical and Information Engineering (EAEEIE), 2022, pp. 1-6.
  Contribution: Supporting co-author. Wrote part of the article. Designed and created 4 chapters of the MOOC.
  DOI:10.1109/EAEEIE54893.2022.9819989
  URL: https://ieeexplore.ieee.org/abstract/document/9819989

Contribution to Courses

- Massive Open Online Course (MOOC) entitled "Ultra-dense networks for 5G and its evolution". This MOOC focused on 5G networks, with a special focus on ultra dense networks. I contributed with some lectures describing what

are the most relevant use cases and how can 5G and beyond 5G expand their current boundaries. Link to the course: https://www.edx.org/course/ultra-dense-networks-for-5g-and-its-evolution

Abstract

The revolution of Extended Reality (XR) has already started and is rapidly expanding as technology advances. Announcements such as Meta's Metaverse have boosted the general interest in XR technologies, producing novel use cases. With the advent of the fifth generation of cellular networks (5G), XR technologies are expected to improve significantly by offloading heavy computational processes from the XR Head Mounted Display (HMD) to an edge server. XR offloading can rapidly boost XR technologies by considerably reducing the burden on the XR hardware, while improving the overall user experience by enabling smoother graphics and more realistic interactions. Overall, the combination of XR and 5G has the potential to revolutionize the way we interact with technology and experience the world around us.

However, XR offloading is a complex task that requires state-of-the-art tools and solutions, as well as an advanced wireless network that can meet the demanding throughput, latency, and reliability requirements of XR. The definition of these requirements strongly depends on the use case and particular XR offloading implementations. Therefore, it is crucial to perform a thorough Key Performance Indicators (KPIs) analysis to ensure a successful design of any XR offloading solution.

Additionally, distributed XR implementations can be intrincated systems with multiple processes running on different devices or virtual instances. All these agents must be well-handled and synchronized to achieve XR real-time requirements and ensure the expected user experience, guaranteeing a low processing overhead. XR offloading requires a carefully designed architecture which complies with the required KPIs while efficiently synchronizing and handling multiple heterogeneous devices.

Offloading XR has become an essential use case for 5G and beyond 5G technologies. However, testing distributed XR implementations requires access to advanced 5G deployments that are often unavailable to most XR application developers. Conversely, the development of 5G technologies requires constant feedback from potential applications and use cases. Unfortunately, most 5G providers, engineers, or researchers lack access to cutting-edge XR hardware or applications, which can hinder the fast implementation and improvement of 5G's most advanced features. Both technology fields require ongoing input and continuous development from each other to fully realize their potential. As a result, XR and 5G researchers and developers must have access to the necessary tools and knowledge to ensure the rapid and satisfactory development of both technology fields.

In this thesis, we focus on these challenges providing knowledge, tools and solu-

tiond towards the implementation of advanced offloading technologies, opening the door to more immersive, comfortable and accessible XR technologies. Our contributions to the field of XR offloading include a detailed study and description of the necessary network throughput and latency KPIs for XR offloading, an architecture for low latency XR offloading and our full end to end XR offloading implementation ready for a commercial XR HMD. Besides, we also present a set of tools which can facilitate the joint development of 5G networks and XR offloading technologies: our 5G RAN real-time emulator and a multi-scenario XR IP traffic dataset.

Firstly, in this thesis, we thoroughly examine and explain the KPIs that are required to achieve the expected Quality of Experience (QoE) and enhanced immersiveness in XR offloading solutions. Our analysis focuses on individual XR algorithms, rather than potential use cases. Additionally, we provide an initial description of feasible 5G deployments that could fulfill some of the proposed KPIs for different offloading scenarios.

We also present our low latency muti-modal XR offloading architecture, which has already been tested on a commercial XR device and advanced 5G deployments, such as millimeter-wave (mmW) technologies. Besides, we describe our full end-to-end complex XR offloading system which relies on our offloading architecture to provide low latency communication between a commercial XR device and a server running a Machine Learning (ML) algorithm. To the best of our knowledge, this is one of the first successful XR offloading implementations for complex ML algorithms in a commercial device.

With the goal of providing XR developers and researchers access to complex 5G deployments and accelerating the development of future XR technologies, we present FikoRE, our 5G RAN real-time emulator. FikoRE has been specifically designed not only to model the network with sufficient accuracy but also to support the emulation of a massive number of users and actual IP throughput. As FikoRE can handle actual IP traffic above 1 Gbps, it can directly be used to test distributed XR solutions. As we describe in the thesis, its emulation capabilities make FikoRE a potential candidate to become a reference testbed for distributed XR developers and researchers.

Finally, we used our XR offloading tools to generate an XR IP traffic dataset which can accelerate the development of 5G technologies by providing a straightforward manner for testing novel 5G solutions using realistic XR data. This dataset is generated for two relevant XR offloading scenarios: split rendering, in which the rendering step is moved to an edge server, and heavy ML algorithm offloading. Besides, we derive the corresponding IP traffic models from the captured data, which can be used to generate realistic XR IP traffic. We also present the validation experiments performed on the derived models and their results.

Contents

List of Figures

List of Tables

List of Acronyms

3GPP Third Generation Partnership Project

5QI 5G QoS Identifiers

API Application Programming Interface

AR Augmented Reality

BET Blind Equal Throughput

BLER Block Error Rate

CDF Cumulative Distribution Function

CNN Convolutional Neural Network

CPU Central Processing Unit

CQI Channel Quality Indicator

CSI Channel State Information

DL Downlink

DoF Degrees of Freedom

DR Distributed Reality

E2E End-to-End

eMBB Enahnced Mobile Broadband

FDD Frequency Division Duplex

FIFO First-In-First-Out

FVV Free-Viewport Video

gNB g-Node B

GPU Graphical Processing Unit

HARQ Hybrid Automatic Repeat Request

HMD Head Mounted Display

IMU Inertial Measurement Unit

IoT Internet of Things

IP Internet Protocol

KPI Key Performance Indicator

LOS Line of Sight

LTE Long Term Evolution

M2P Motion to Photon

MCS Modulation and Coding Scheme

MEC Multi-Access Edge Computing

MIMO Multiple Input Multiple Output

ML Machine Learning

mMTC Massive Machine-Type Communications

mmW Millimeter Wave

MR Mixed Reality

MT Maximum Throughput

NPN Non-Public Networks

NQ Netfilter Queues

OFDM Orthogonal Frequency-Division Multiplexing

PDCP Packet Data Convergence Protocol

PDF Probability Distribution Function

PF Proportional Fair

PPD Points per Degree

QAM Quadrature Amplitude Modulation

QoE Quality of Experience

QoS Quality of Service

RAN Radio Access Network

RB Resource Block

RBG Resource Block Group

RLC Radio Link Control

RSP Received Signal Power

RSRP Reference Signal Received Power

RTP Real-time Transport Protocol

RTT Round Trip Time

SINR Signal to Noise and Interference Ratio

SLAM Simultaneous Localization and Mapping

SoE Sense of Embodiment

SoP Sense of Presence

SR Scheduling Request

TBS Transport Block Size

TCP Transmission Control Protocol

TDD Time Division Duplex

TTI Transmission Time Interval

UDP User Datagram Protocol

UE User Equipment

UL Uplink

URLLC Ultra-Reliable Low-Latency Communications

VR Virtual Reality

XR eXtended Reality

Chapter 1

Introduction

EXtended Reality (XR) refers to a group of technologies that seek to enhance and expand human perception of the physical world. This includes Virtual Reality (VR), which creates immersive digital environments, and Augmented Reality (AR), which overlays digital information on the physical world. In between both edges of immersive media technologies we have other solutions such as Mixed Reality (MR) [1] or Distributed Reality (DR) [2]. XR technologies have seen significant developments in recent years, and have potential applications in a wide range of fields, from gaming and entertainment to healthcare and education.

The recent interest in XR technologies, driven in part by the announcement of Meta's plans for a new type of social interaction called the Metaverse [1], has led to significant investment in the technology. This has resulted in lighter, more affordable XR Head Mounted Displays (HMDs), hardware, and algorithms, improving the overall user experience.

XR technologies typically involve the use of an HMD to render virtual content. In VR, the HMD completely occludes the real world, completely cutting the user off from it. AR HMDs, on the other hand, use transparent displays or video pass-through technologies to overlay virtual content on the real world.

Current XR devices have reached eye-level resolution, with more than 70 Points per Degree (PPD), such as the Varjo XR-3 [2]. However, this level of resolution is only possible with tethered devices connected to high-end computing platforms with state-of-the-art Graphical Processing Units (GPUs). XR technologies have also advanced to the point where fully wireless VR and AR devices with extensive capabilities are now available. For example, the Meta Quest 2 [3] is a wireless VR HMD with a resolution of 1832x1920 per eye, hand tracking capabilities and a robust 6 Degrees of Freedom (DoF) head tracking.

This fast development of XR technologies is enabling novel use cases that involve a new way of interacting with the real world and with each other. However, there are still several technological constraints, such as limited computing power or battery life, that hinder XR applications from reaching their full potential. In addition, novel applications such as the Metaverse require real-time connections between a large number of users.

[1] https://about.facebook.com/
[2] https://varjo.com/products/xr-3/
[3] https://www.meta.com/es/quest/products/quest-2/

Expanding the current boundaries of XR technologies and applications requires the use of high-end hardware with powerful GPUs for real-time rendering and Machine Learning (ML) processing. As a result, the most technologically advanced immersive devices are still tethered, bulky, uncomfortable, and expensive. The release of advanced networks, such as the fifth generation of cellular networks (5G), has led to the exploration of distributed solutions for immersive technologies as a means of enabling wireless, lighter, and more affordable devices while increasing the overall user experience. The goal is to offload some or all of the heavy processing tasks to a nearby server, improving the available computing power while reducing the XR HMD processing requirements. XR offloading can become a key enabler that can provide significant benefits, including improved user experience, reduced demand on local computing resources, and the ability to support a larger number of users simultaneously.

XR offloading is not a trivial task. Both AR and VR devices handle intense data rates in a limited amount of time: the update rate must be kept above 60 Hz to ensure a successful experience and avoid any discomfort, such as motion sickness [3], to the user. Consequently, successful task offloading for VR and AR applications requires a robust network that can satisfy the extremely tight latency and throughput requirements.

According to Third Generation Partnership Project (3GPP) specifications, 5G includes a set of three services: Enahnced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communications (URLLC) and Massive Machine-Type Communications (mMTC). Both eMBB and URLLC 5G services match XR applications' extremely high communication data rate requirements under demanding low latency constraints, which can enable offloaded immersive solutions. Therefore, 5G has been considered a natural key technology enabler for the future development of XR technologies.

1.1. Motivation

Successful XR offloading requires a tightly optimized pipeline, including but not limited to the architecture, network or the XR HMD itself. It involves a complex ecosystem in which many tools have to be set and configured to seamlessly interact with each other. As shown in Fig. 1.1, any complex XR offloading application involves multiple algorithms which need to run in different devices along the entire architecture: edge, cloud, device, etc. Where these algorithms should run and what the network and hardware requirements are for them to run in real-time must be understood in detail before any successful XR offloading attempt.

Consequently, the first natural step is to understand what are the requirements that XR offloading imposes to the network and offloading architecture. There are

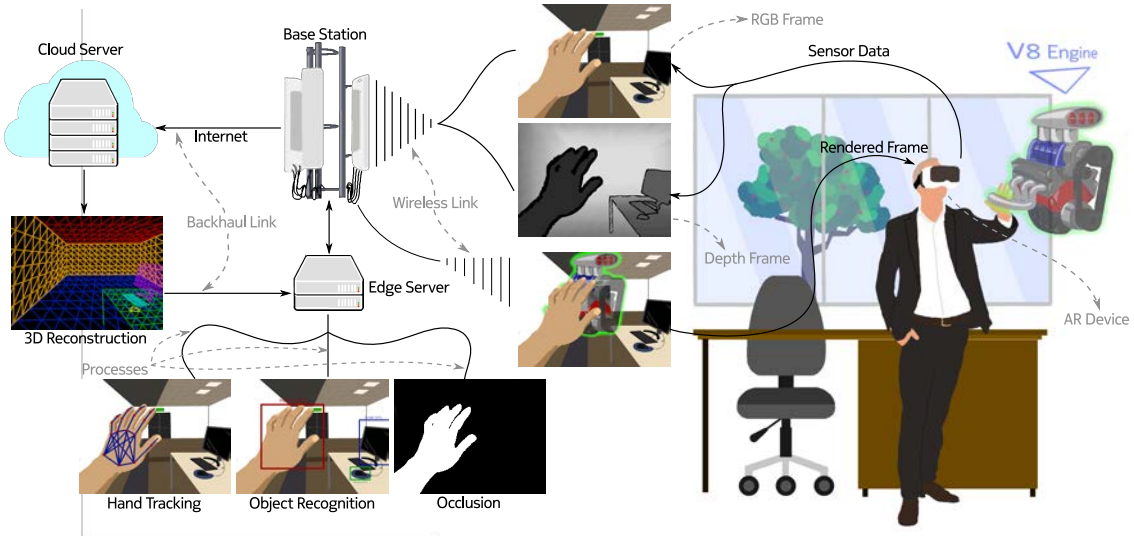


Figure 1.1: Simplified offloading architecture for a complex XR application.

multiple studies that propose a set of requirements and implementations for achievable XR offloading. Multiple VR offloading research examples focus on the requirements and resource handling techniques for Multi-access Edge Computing (MEC) based offloading, as in [4][5][6][7]. Other studies, on the other hand, focus on novel resource allocation techniques specifically designed for VR offloading, as in [8]. Finally, there are relevant works which thoroughly analyze the network requirements for VR offloading, as in [9][10]. All the described VR research examples are focused on intense downlink traffic as they consider the main VR offloading use case to be focused on split rendering approaches [11]. In this case, the rendering process is moved to a nearby server which continuously receives the updated poses of the head and hands, and renderS the realistic immersive frames accordingly. This approach allowS to display very realistic scenes in light and comfortable VR devices with limited computing power. However, VR immersiveness is not only affected by the visual resolution: the user needs to seamlessly interact and move within the virtual scenario. Besides, any VR experience should aim to achieve a high level of Sense of Embodiment (SoE)[12] as it is directly connected to how the users perceive themselves within the VR experience. State of the art VR research aims to produce novel methods and algorithms to improve the user experience, interaction and self awareness. These methods include, for instance, hand tracking [13], object detection [14], or egocentric human segmentation [15]. Most of these methods involve high uplink data rates to transmit the sensor data captured in each frame while the end to end latencies should lay below the sampling period of the device, which is smaller than 16 ms in current VR devices, such as the Meta Quest 2 with an update rate between 72 and 90 Hz.

Similarly, AR applications should not only properly place the virtual content but also allow both the user and the real scenario to interact with it. This requires the real environment to be analyzed as accurately as possible and in real-

time, requiring multiple heavy duty algorithms, such as semantic segmentation and 3D reconstruction, to concurrently run in real-time. Even though there are some real-time state-of-the-art implementations of these processes [16][17], they require modern hardware which is usually not portable and energy consuming. Besides, the AR content should be rendered with a motion to photon delay below 15 ms [3], deadline to which most of the previously mentioned processes need to comply. Even more constraint is the case in which the user needs to interact with a virtual object: the virtual object's reaction and haptic feedback latencies must be smaller than 10 ms [3]. There are some studies [18] which thoroughly analyze the distributed implementation of AR, proposing several offloading architectures and some use case-dependent requirements.

In general, XR offloading studies are focused on specific use cases' requirements. However, they lack a thorough understanding of how and where the individual algorithms should run, and what are the individual offloading requirements. Each individual XR application requires different combinations of XR algorithms. Knowing the specific offloading requirements for each algorithm is a key factor for an optimal architecture design: different offloading combinations can be deployed for the same use case or application depending on the available network resources. With this thesis, we aim to provide a detailed description of the most fundamental algorithms of XR. Besides, we propose a set of throughput and latency requirements for offloading different combinations of these fundamental algorithms, filling the current gap in XR offloading requirements research studies.

With a precise and relevant set of XR offloading requirements, researchers can design and develop optimized offloading architectures accordingly. There have been several relevant research examples that describe and test actual XR offloading architecture implementations. For example, in [19], the authors propose an efficient offloading architecture specifically optimized for mobile AR. Additionally, the authors in [20] propose an offloading architecture carefully designed to reduce energy consumption. Finally, in [21], a reinforcement-learning self-optimizing distributed offloading architecture is proposed. While some of these architectures have been benchmarked, they often only focus on resource allocation or energy consumption. However, complex XR offloading imposes tight throughput and latency requirements on both the uplink and downlink, which can only be fulfilled with an extremely optimized architecture and a strong network. In this thesis, we propose our own offloading architecture, specifically optimized for XR offloading using the relevant Key Performance Indicators (KPIs) obtained in the theoretical analysis. Our goal is to provide a novel architecture that is capable of sustaining the intense throughput and latency requirements expected for a wide range of offloading combinations.

In general, most XR offloading studies follow a theoretical approach or validate proposed architectures using synthetic traffic and network simulators. However, it is rare to find research venues that propose end-to-end XR offloading implementations

tested on commercial hardware using actual wireless access points. In this thesis, we implement a full end-to-end system to demonstrate the potential of XR offloading and the performance of our offloading architecture in a realistic commercial deployment. Our goal is to offload our egocentric human segmentation algorithm, which allows users to see themselves in real-time within a VR scene [15]. This algorithm is at the cutting edge of video-based avatars and requires a high-end GPU to run in real-time, justifying the need to offload the process to a nearby server. Along with the detailed explanation of the implemented end-to-end solution, we describe the user study we conducted to validate both the offloading architecture and our egocentric human segmentation solution.

During the experiments with both the isolated offloading architecture and the end-to-end system, we realized that low frequency bands (sub-6) 5G cannot sustain the throughput and latency requirements of the most demanding scenarios. The use cases we are testing require an advanced wireless network, such as Millimeter Wave (mmW) technologies, to sustain the throughput and latency requirements. However, most advanced 5G technologies are still under development and have not reached their full potential. We identified this important technology gap which can slow down the development of distributed XR implementations. One popular alternative is the use of simulation tools that realistically model the behavior of not only available 5G technologies, but also other implementations that are foreseen in the standard but not yet available. In particular, XR researchers could potentially use real-time emulators to test their solutions on a realistic yet simulated 5G deployment. There are multiple 5G emulators in the state of the art such as NS-3 [22], OpenAirInterface (OAI) 5G-RAN [23], or Simu5G [24]. However, these emulation tools are too limited in terms of number of simulated User Equipments (UEs) or throughput capabilities (less than 10 Mbps) to be considered a reference testbed for XR offloading testing and prototyping. Therefore, we decided to build our own emulator, FikoRE, with the goal of realistically modeling the network while achieving high throughput, above 1 Gbps, and a massive number of UE [25][26]. In this thesis, we present, describe, and validate FikoRE as a real-time open-source 5G emulator. We also present a realistic setup for Free-Viewport Video (FVV) rendering and distribution, in which FikoRE has been used as a prototyping testbed.

After years of researching distributed XR implementation, we can conclude that XR offloading is a complex task with tight requirements. The network must be carefully designed and configured, which requires a long iteration and research process. Both XR technologies and telecommunications research must collaborate to achieve a complex goal that cannot be fulfilled without each other's knowledge. However, telecommunications researchers and developers often do not have access to fully developed distributed XR implementations, which generate the necessary traffic to test their solutions. A popular trend is to rely on recorded or modeled traffic data. While there are already some complete XR traffic models available in

the standards [27], they only consider a subset of relevant use cases. Additionally, their video transmission models, while precise, lack information about inter-packet arrival time. This inter-packet arrival time, defined as the time between individual Internet Protocol (IP) packets within a single video frame, can be a relevant parameter to model accurately, especially for resource allocation algorithms. During the development of this thesis, we created our own XR offloading traffic dataset which complementary covers other use cases not included in [27]. We also proposed a set of XR traffic models, derived from our dataset, which include inter-packet arrival times, filling what we believe is an important research gap.

1.2. Contributions

In this thesis, we focus on the requirements, technologies and tools which are necessary to fulfil and implement in order to achieve successful distributed XR implementations. In particular, our contributions are:

- A thorough breakdown of XR technologies in their most fundamental algorithms to extract their individual requirements and how they interact with each other. The conclusions from this breakdown are reflected in [28] and [29], where we also derive, by solving a simple optimization problem, a set of required donwlink and uplink peak throughput and end to end latency values for a selected group of VR and AR offloading scenarios. Besides, we also propose different 5G configurations and high-level architectures which, according to the standards, can fulfil the proposed requirements.

- A novel implementation of a Transmission Control Protocol (TCP) based architecture specifically designed for XR offloading, presented in [30]. Besides, we present field results of our offloading architecture for different scenarios and offloading algorithms. In the same study, we carry out some throughput and latency becnhmarking experiments in different wireless access technologies, including 5G mmW frequency band. In this thesis we also introduce our work in progress extension of our offloading architecture, which incorporates H.264 [31] encoding transmitted over Real-time Transport Protocol (RTP) [32] for video streaming, which relaxes the overall throughput and latency requirements.

- Full end to end implementation of an offloading solution on a commercial VR HMD, thoroughly described in [33]. In our end to end implementation, we offload our real-time deep-learning egocentric segmentation algorithm which identifies which pixels from an egocentric point of view corresponds to the VR user. The end goal is to allow the users to see themselves within the VR scene. Our end to end implementation allows to use our novel state of the art

segmentation algorithm, in real-time, on a Meta Quest 2 [4]. Besides, we perform a fairly thorough user study to understand how the algorithm performs and if the offloading architecture is affecting the Quality of Experience (QoE) at all.

- A full stack 5G Radio Access Network (RAN) emulator which outperforms, in terms of scalability and throughput handling capabilities, any other emulator in the state of the art. The emulator, named FikoRE and described in [25], has been tested on a real deployment of Free-Viewport Video streaming. Besides, it has been validated using measurements obtained from field tests using an actual mmW deployment. FikoRE source code has been published openly in [26].

- An XR offloading traffic dataset captured using our offloading architecture, detailed in [34]. From the dataset, we derive the correspondent traffic models, with higher level of detail than the models proposed in the specifications [27]. Besides, the captured offloading use cases are complementary to the ones proposed in [27]. The models have been validated using FikoRE. Finally, the dataset has been made public as part of FikoRE repository [26].

1.3. Thesis Outline

The present thesis is organized as follows:

- In Chapter 2, we describe the current state of the art of XR technologies including a breakdown of AR and VR technologies. We also present the optimization problem that we designed to estimate a set of throughput and latency KPIs for different relevant offloading scenarios. Finally, we include a summary of the possible 5G configurations that can sustain the estimated requirements for the selected offloading scenarios.

- In Chapter 3, we detail our offloading architecture. We described the implementation details of both our TCP version and first approach to the RTP with H.264 encoding version. We also present some validation experiments carried out on both WiFi and mmW technologies.

- In Chapter 4, we present our end to end offloading implementation which has been tested on a commercial VR HMD. Besides, we describe the user study experiments carried out to validate our end to end offloading implementation.

- In Chapter 5, we provide a detailed description of FikoRE, our real-time full stack 5G RAN emulator. We provide fundamental implementation details,

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

followed by the results obtained in the carried out validation experiments. Finally, we show the performance of FikoRE on a Free-Viewport Video streaming application.

- In Chapter 6, we describe the steps followed to capture a realistic XR offloading traffic dataset. Besides, we provide model distributions estimated using the measured traffic, which are validated using FikoRE.
- In Chapter 7, some conclusions and future work are pointed out.

Chapter 2

Key Performance Indicators and Potential Network Architectures for eXtended Reality Offloading

XR offloading imposes tight and demanding requirements to the network, which have been thoroughly studied from a theoretical perspective in the state of the art. However, successful real-time offloading for XR applications imposes extreme throughput, latency and even reliability requirements to the network which strongly depend on the specific process to be offloaded and target application. Besides, these process-specific requirements are also different depending on the targeted XR technology. The visual requirements for AR are considerably lower than VR, while the spatial and semantic awareness of the real scenario is crucial in AR.

As described in Chapter 1, most of AR and VR offloading research examples are focused on generalist approaches based on specific high-level use cases, which impose very high throughput and latency requirements on the downlink side. In both AR and VR, the perceived QoE does not only depend on the visual quality, but other relevant factors such as pose estimation precision, interactivity, spatial awareness, and a large etcetera. The sense of immersiveness, for instance, is low if the user has very poor interaction possibilities with the virtual content, regardless of its visual quality. The individual algorithms which directly affect the immersiveness can be considerably enhanced by providing access to remote computational resources. As different XR implementations and applications might use different combinations of algorithms, we decided to study XR offloading requirements from a lower level: we study the offloading latency and throughput KPIs of each individual XR algorithm that can be potentially offloaded. These individual theoretical KPIs can facilitate the design and deployment of complex offloading architectures regardless of the targeted use case.

In this chapter, we present the followed procedure to analyze XR relevant algorithms and extract a set of theoretical throughput and latency KPIs from this analysis. We achieve this by thoroughly analyzing the most relevant XR algorithms along with their data flow. The goal is to understand what are the processing requirements in terms of processing times, input and output data flows and interaction between them. This information is then used as input to solve an optimization problem resulting in the targeted network KPIs. We use the obtained KPIs to pinpoint which of the proposed algorithms are firm candidates to be offloaded. Finally, we propose high-level architecture based on the standards which can satisfy the obtained requirements for a set of arbitrary offloading scenarios.

2.1. A Breakdown of eXtended Reality Algorithms

XR technologies and applications gather multiple algorithms that interact with each other. To estimate the throughput and latency KPIs of a potential distributed XR implementation we need to understand each algorithm individually analyzing its input and output flows, update rates and hardware requirements. We analyze the state of the art of the most relevant XR algorithms to obtain these characteristics and specifications. The analysis is later used to derive the target throughput and latency KPIs for a set of arbitrary potential distributed implementations.

2.1.1. Sensor Capture and Preprocessing

State of the art XR devices include multiple sensors: RGB and depth cameras, microphones, inertial measurement units (IMUs), controllers, etc. These sensors provide the necessary input data to the most relevant XR algorithms. Therefore, we need to provide an accurate estimate of the data rates and frequency updates for a successful throughput and latency KPIs analysis.

As described in Chapter 1, AR and VR technologies are on opposite edges of XR technologies. Therefore, we decided to use different devices as reference for AR and VR. On the AR side, we take the Hololens 2 [35] as a reference, being the most recent and advanced AR device available. Running at 60 Hz, the Hololens 2 is producing around 500 Mbps of raw sensor data. The RGB camera captures frames at a 1080p (1920x1080) resolution while the depth camera has a square resolution of 1024x1024.

We use the Varjo XR-3 [36] as the VR reference HMDs to estimate the sensor data specifications. The Varjo XR-3 is the most advanced commercial VR HMD in terms of resolution. The XR-3 also incorporates video pass-through capabilities, via a set of high-resolution stereo cameras, enabling to use them as an AR device or for DR [2] applications. The RGB cameras are also 1080p. The depth sensor captures 1920x1080 resolution frames.

In general, this sort of data is transmitted using video encoding algorithms such as H.264 which considerably reduces the necessary bitrate, being the combination of H.264 encoding and RTP transmission is the most efficient solution in terms of latency and throughput. However, it can introduce artifacts in the received video stream due to packet loss, encoding quality or delays. In this study we try to obtain a set of network requirements for advanced XR offloading which can improve the current state of the art of XR technologies. Many of the described algorithms are sensitive to this artifacts which can decrease the overall quality of experience. Therefore, for uplink sensor data transmission in this study we consider per frame encoding using JPEG which requires higher bitrates imposing more constraint network

Table 2.1: Reference RGB and Depth camera feeds resolution, size per frame and expected data rates.

Sensor	Resolution	Frame Size	Data Rate
Varjo XR-3 RGB	1920x1080@90	2.50 Mbits	225 Mbps
Varjo XR-3 Depth	1920x1080@90	2.50 Mbits	225 Mbps
Hololens 2 RGB	1920x1080@60	2.50 Mbits	150 Mbps
Hololens 2 Depth	1024x1024@60	1.33 Mbits	80 Mbps

requirements, but avoiding the introduction of potential artifacts in the stream.

According to [37], a typical JPEG compression ratio assumes a weight of 1.2 bits per pixel. We use this reference to estimate the weights values per frame summarized in Table 2.1. On the other hand, the audio capture and Inertial Measurement Unit (IMU)'s feeds are considerably smaller than the camera feeds and, therefore, we decided to neglect them from the subsequent analysis for simplification.

2.1.2. Position and Orientation Estimation

Distributed implementations of XR applications might lead to a new generation of wireless and light high-end VR devices. Consequently, the user will have the chance to freely move within the virtual scene rather than just acting as a static observer. In this new scenario it is even more crucial to accurately track the device's pose in real-time as pose inaccuracies can provoke motion sickness. On mobile scenarios, the device translations are not bounded and the tracking errors can critically increase, which might completely ruin the experience. Therefore, XR devices must be able to accurately estimate its own pose in real-time using the available sensor data. This estimation is done using Simultaneous Localization and Mapping (SLAM) algorithms [38]. While there are multiple SLAM implemantations which rely on only a single RGB stream [39], in general, a robust XR-targeted SLAMs algorithm requires the combination of at least an RGB camera and an IMUs. In some examples, the SLAMs algorithm is also fed with the depth information or multiple RGB feeds, increasing the algorithm's accuracy and robustness. On the other hand, the output bitrate could be neglected as it is only composed by some few dozens of floats for every processed frame. In general, SLAM algorithms which combine an IMU and one or several RGB cameras can succesfuly track the device using low camera frame rates (below 10 Hz). However, due to the fast and dynamic nature of XR head movements, SLAM algorithms for XR require camera update frequency of 30 to 60Hz. High update rates usually improve the accuracy, while demanding more computational power [40].

In some cases, especially for AR, the SLAMs algorithm output is used for 3D reconstruction, requiring the generated point cloud and RGB/depth key frames to

be exported, considerably increasing the output bit rate. Offloading the SLAMs process would require both the 5G eMBB and URLLCs capabilities to fulfil the constrained update times. See Table 2.2 for an itemized summary of the data flow and timing requirements.

3D Reconstruction

The goal of 3D reconstruction is to model, with different grades of accuracy depending on the final goal, the 3D morphology of the real scenario surrounding the user. It is a key algorithm in many AR applications for generating realistic interactions between the real and virtual objects.

3D reconstruction can be considered an extension of SLAMs algorithms to an extent in which some SLAMs algorithms simultaneously handle the pose tracking, and the 3D reconstruction such as ORB-SLAM [39]. In general, the input of the state-of-the-art algorithms consists of the continuously updated SLAMs pointcloud, the device's estimated pose, and the depth and RGB cameras' feeds [16]. The output data consist of the estimated 3D mesh updates along with the corresponding color textures. It is not efficient to continuously export the entire mesh, but only the sections which are modified. The offloading requirements are less restrictive as the generated mesh updates can be transmitted at a lower rate, 0.5-1 Hz, and higher latencies (below a second), allowing this process to be offloaded to a further server. Finally, the final 3D reconstruction could be stored on the device or a server after each session and be reloaded in a posterior session. See Table 2.2 for a summary of 3D reconstruction data flow and timing requirements.

2.1.3. Semantic Understanding Algorithms

Semantic understanding algorithms obtain semantic information from the real scene, such as which real objects are in the scene along with their shape, pose, and size. In our analysis, we focus in three of the most important semantic understanding algorithms:

- **Object detection** is a very populated research topic for its importance on uncountable fields such as autonomous driving or AR. Consequently, its state of the art has improved rapidly aiming to decrease the size of object detection deep learning networks while preserving their accuracy. Therefore, current algorithms reach real time performance on standalone devices with limited computing power such as smartphones. Two key examples of these shallow networks are the YOLO-Tiny and YOLO-Lite which run at 244Hz on a GPU and 20Hz on a Central Processing Unit (CPU) respectively [14]. In the latter, the CPUs usage is very high while achieving lower detection accuracies than

in GPUs-enabled computing systems. The input is generally the RGB feed from a single camera while the output is simply the position, size and label of the bounding box. The output is therefore light and can be neglected in comparison with other heavy data flows. In this case, the update rate requirements are not as tight as some filters can be used to estimate the position of the detected objects in between updates.

- **Hand tracking** it is a recent feature included in most XR devices which is becoming crucial as it allows the users to interact with the virtual content with their own hands instead of using a remote control device [41] [13] [42]. However, these algorithms still consume an important share of the available in-built computing resource. There are multiple hand tracking approaches. In [41], the authors propose a generative adversarial network (GAN) based solution to solve the hand tracking problem from a single RGB sensor. On the contrary, in [42], the authors just use a single depth sensor as the only input to their proposed method. Finally, in [13], both the RGB and depth feeds are used for real-time hand tracking estimation. We consider this last setup as our reference scenario to numerically analyze hand tracking algorithms as it is the most demanding one in terms of end to end latency as it is a critical algorithm which considerably affects the user experience. Regarding the update rate, in [43] the authors achieve an update rate of 50Hz on a GPUs-less device. We assume update frequency rates of 60 to 90Hz for hand tracking state of the art examples running on high-end hardware, with processing times between 8 and 12 ms (see Table 2.2)

- **Semantic segmentation** can be understood as a natural extension of object detection algorithms: once the object is detected, its accurate shape and 2D position is extracted. This step can be used, for instance, to handle the occlusion of dynamic objects like the user's hands. The way the users perceive their own hands can drastically affect the sense of embodiment and the immersiveness [44]. There is a new research path which explores methods to allow the users to see themselvesf within the virtual scenario without the use of avatars. The so called egocentric segmentation techniques rely on computer vision algorithms to segment the hands or other human body parts from a real-time egocentric video feed. The segmented feed is included and seamlessly blended with the virtual content. This solution is useful in both AR and VR. In AR, this segmentation mask can be used to realistically occlude the virtual content, as in [45]. Dynamic occlusion handling is a tough task which is still considered an open AR problem. If the virtual content is not properly occluded behind a moving object, for instance the user's hands, it would produce a wrong perception of its position within the real world, ruining the experience. It requires a perfect segmentation of the moving object, with very constrained processing times. Some state of the art examples can run in real

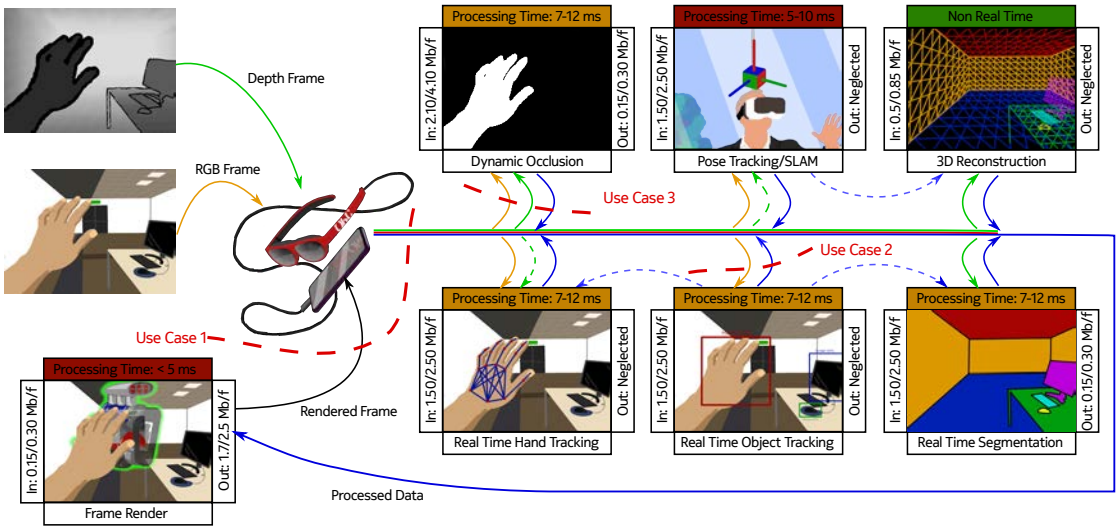


Figure 2.1: Simplified AR architecture including estimated processing times and required input and output frame sizes.

time, as in [46], reaching 60 Hz on a high-end GPUs. In VR, this technique allows to substitute virtual avatars by the segmented real-time feed from the HMDs's frontal stereo cameras, as in [47] [33]. The main goal of the egocentric semantic segmentation algorithms is to improve the sense of embodiment, the sense of presence (SoP) and the interaction with the virtual or even real objects as in [48]. Egocentric segmentation should run at the same data rate as the device's update rate to avoid any flicker or miss alignment which can ruin the experience. In the case, the input corresponds to the RGB feeds which can be complemented with the depth camera feeds. The output consists of a pixel classification mask per frame considerably increasing the output bit rate in comparison to the other two algorithms.

The summary of the data flow and timing requirements for offloading the described algorithms is summarized in Table 2.2. Besides, Fig. 2.1 depicts a schematized representation of the main described algorithms for the particular case of a distributed AR implementation.

2.1.4. XR Engine and Frame Rendering

The XR engine simulates the physics of the virtual content and renders a new frame accordingly. While in VR this is an extremely demanding step which requires high end GPUs for advanced virtual scenes, in AR this step is not extremely demanding, allowing the devices to be completely wireless. In both cases, targeting to decrease the size and weight of XR HMDs, it is natural to consider the possibility of offloading the rendering step.

High resolution VR scene rendering targets to achieve human-eye resolution with

Table 2.2: Summary of input and output size per frame, update frequencies and estimated processing times for the analyzed XR processes. In green the input data, in red the output data.

Algorithm	Data@Resolution	Freq.	Frame Size	Bitrate	Times
SLAM	RGB@1920x1080	60 Hz	2.50 Mbits	150 Mbps	7-12ms
	Depth@1024x1024	60 Hz	1.33 Mbits	80 Mbps	
	Pose	60 Hz	<<1 Mbits	<<1 Mbps	
	Keyframes	20 Hz	3.83 Mbits	77 Mbps	
3D Reconstruction	Pose	60 Hz	<<1 Mbits	<<1 Mbps	+100ms
	Keyframes	20 Hz	3.83 Mbits	77 Mbps	
	3D Mesh + Texture	1 Hz	<<1 Mbits	<<1 Mbps	
Object Detection	RGB@1920x1080	60 Hz	2.50 Mbits	150 Mbps	7-12ms
	Bounding Box	60 Hz	<<1 Mbits	<<1 Mbps	
Hand Tracking	RGB@1920x1080	60 Hz	2.50 Mbits	150 Mbps	7-12ms
	Depth@1024x1024	60 Hz	1.33 Mbits	80 Mbps	
	Hand pose	60 Hz	<<1 Mbits	<<1 Mbps	
Ego. Segmentation	RGB@1920x1080	60 Hz	2.50 Mbits	150 Mbps	7-12ms
	Depth@1024x1024	60 Hz	1.33 Mbits	80 Mbps	
	Mask@1920x1080	60 Hz	0.32 Mbits	20 Mbps	
AR Rendering	Masks and Pose	60 Hz	0.32 Mbits	20 Mbps	5-12 ms
	Frame@1920x1080	60 Hz	2,5 Mbits	150 mbps	
VR Rendering	Masks and Pose	90 Hz	0.32 Mbits	29 Mbps	5-12 ms
	Frame@7680x1080	90 Hz	3.33 Mbits	300 Mbps	

at least 4K per eye and a density of 60 pixels per degree per eye demanding intense computational resources: the Varjo XR-3 specifications require the use of at least an NVIDIA GeForce® RTX 2080 Ti GPUs. In this case, for both AR and VR rendering, the transmission of the rendered frames must be done following a very optimized pipeline including H.264/H.265 encoding over RTP. With this setup, and as described in [34], we have maximum frame sizes of 3.33 Mbit.

The input to the render process is the updated position of the virtual objects and the device, given by the VR engine, such as Unity or Unreal. The engine is also in charge of using the data produced by the aforementioned algorithms. State of the art devices are optimized to deliver rendering times below 5 ms. Refer to Table 2.2 for a detailed summary of the processing and data flow specifications of the rendering process.

2.2. Numerical Analysis

The goal of this section is to offer an initial estimation of the required network KPIs, in terms of required peak throughput in each transmission direction, Uplink (UL) and Downlink (DL), and two-way transmission latency, necessary for a successful distributed XR implementation.

For our analysis, we simplify the ideal transmission time of a single frame, considering that it is only affected by the amount of data to be sent divided by the effective throughput. We express the transmission time of a single frame as:

$$t_f = \frac{S_f}{T}, \tag{2.1}$$

in which t_f is the transmission time of frame f in seconds, S_f the size in bits of f and T the effective throughput in bits per second. This throughput can be understood as the necessary peak throughput that needs to be sustained during t_f. After the frame is transmitted, the link is considered to be idle until the next frame is sent. As a consequence, our analysis is focused on the peak throughput which the network should be able to provide rather than an estimation of the necessary mean throughput. For simplification, we consider that T already includes all the possible sources of throughput loss such as packet loss, re-transmission or packet duplication, among others. We also define the total transmission latency L as the aggregation of the ideal propagation time and other network-specific latencies such as the ones associated with the Hybrid Automatic Repeat Request (HARQ) process, among others. The total network transmission time t_N can be written as:

$$t_N = \frac{S_u}{T_u} + \frac{S_d}{T_d} + L \tag{2.2}$$

in which S_u and S_d are the frame size on the uplink and downlink sides respectively, and T_u and T_d the effective peak throughput. If we consider the processing time t_p the aggregated time of all the non-networked processes involved in the VR application, we can then express the total update time t, which has to be smaller than the device's update period $1/f_d$, as:

$$t = \frac{S_u}{T_u} + \frac{S_d}{T_d} + L + t_p < \frac{1}{f_d}, \tag{2.3}$$

The processing times for the different possible VR offloading scenarios are hard to be accurately estimated in a general case: they depend not only on the characteristics of the specific set of algorithms used, but also on the hardware and software implementations. Consequently, our goal is to estimate the less restrictive sets of uplink and downlink peak throughput and latencies which satisfy Eq. 2.3, given different processing times. To achieve this, we propose a simple optimization problem which aims to maximize a reward function:

$$T_u, T_d, L = \arg\max_{T_u, T_d, L} R(T_u, T_d, L, t_p, S_u, S_d). \tag{2.4}$$

As we want to find the less restrictive set of peak throughput and latency KPIs, the reward function is designed to try to maximize the latency and reduce the peak throughput while increasing the total update time. The reward function is expressed

as:

$$R(T_u, T_d, L, t_p, S_u, S_d) = w_L(1 - \frac{L^{\max} - L}{L^{\max} - L^{\min}}) + w_{T_u}\frac{T_u^{\max} - T_u}{T_u^{\max} - T_u^{\min}} + w_{T_d}\frac{T_d^{\max} - T_d}{T_d^{\max} - T_d^{\min}} + w_t\frac{t_N + t_p}{\frac{1}{f_p}}, \tag{2.5}$$

in which $L^{\max}$, $L^{\min}$, $T_u^{\max}$, $T_u^{\min}$, $T_d^{\max}$ and $T_d^{\min}$ are the maximum latency and peak throughput values and w_L, w_{T_u}, w_{T_d}, w_t weights to modify the reward function. These weights are used to adjust the importance during the optimization of each of the target parameters: T_u, T_d, L or t. We defined a set of constraints to solve the proposed optimization problem which can be summarized as:

$$\frac{S_u}{T_u} + \frac{S_d}{T_d} + L + t_p < \frac{1}{f_d} \tag{2.6}$$

$$kT_u^{\max} \leq T_d^{\max} \tag{2.7}$$

$$T_u \leq T_u^{\max} \tag{2.8}$$

$$T_d \leq T_d^{\max} \tag{2.9}$$

$$T_u^{\max} + T_d^{\max} <= T^{\max} \tag{2.10}$$

$$L \geq L^{\min} \tag{2.11}$$

For our analysis, we set the minimum latency $L^{\min}$ to 1 ms and $T^{\max}$ to 10 Gbps according to the initial capabilities of ultra-dense 5G networks. The initial deployments of 5G networks implement Time Division Duplex (TDD) method to separate the inward and outward signals. According to 3GPP TS 38.213 V.16.1.0 [49] standard, we can assume 4 times more slots allocated for the downlink signaling. As a consequence, we assign 2 Gbps for $T_u^{\max}$ and 8 Gbps for $T_d^{\max}$. Therefore, the value of k in Eq. 2.7 is set to 4. We solve the optimization problem for a set of t_p ranging from 1 ms to $(1/f_p - 2)$ ms, obtaining groups of peak throughput and latencies for each considered processing time. For our analysis, we consider that no adaptive offloading technique [50] is used as we want to study the most restrictive case in which the environment is highly dynamic. In this case, all the captured frames are sent to the server. The optimization problem has been solved using Python's SciPy library with the Nelder-Mean algorithm.

2.3. Reference Scenarios and Results

As AR and VR technologies have different requirements for the described algorithm, we are proposing separated use cases for AR and VR.

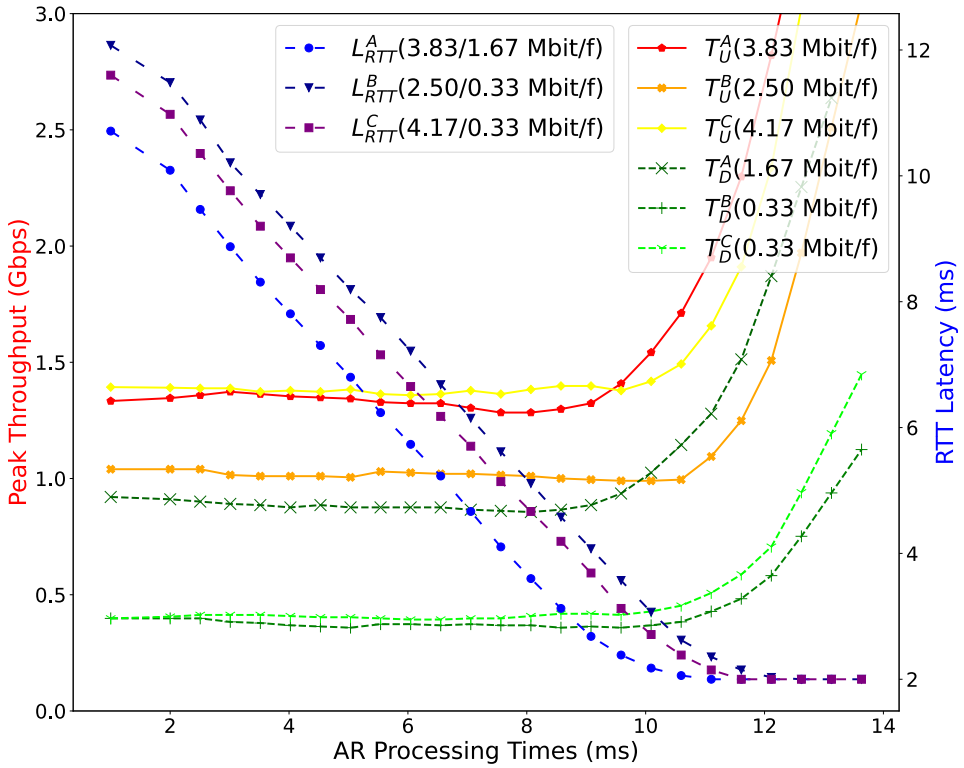


Figure 2.2: Set of optimal Round Trip Times and Uplink and Downlink peak throughputs for a set of given AR processing times for the three AR use cases running at 60 Hz.

2.3.1. AR Use Cases

For the sake of simplicity, we choose the 3 most representative AR offloading scenarios. In use case A, we analyze the full AR offloading scenario, which is necessary to enable ultra-light mobile AR devices with almost no computation capabilities. In the case in which the AR device includes a computationally powerful companion or can achieve heavier computations, only some of the mentioned algorithms need to be offloaded. We analyze the offloading of two different demanding algorithms separately: object detection and semantic segmentation (use case B) and occlusion handling (use case C), as they are crucial AR computationally intensive algorithms. The red dash lines in Fig. 2.1 represent which algorithms are offloaded in each use case. Notice that the throughput values are not the mean values but the peak throughput values that the network needs to supply during the frame transmission time.

AR Use Case A. Full Offloading

In this first use case, the full AR processing stack is offloaded from the device. In this setup, the uplink stream is expected to include all the sensor data. On the downlink side, the system transmits the rendered scene back to the device. The

estimated mean uplink rate for this first use case is between 150 and 230 Mbit/s for a device running at 60 FPS. On the downlink side, we estimate the transmission to require 1.67 Mbit/frame, as the transmitted data corresponds to the high-definition rendered frame. In Fig. 2.2, we can observe the estimated pairs of round-trip latencies and uplink and down-link peak throughput optimal values that satisfy the 16.6 ms of total update time for different AR processing times. The depicted latency corresponds to the total round-trip latency, including any latency added by the entire network stack. We can observe that for AR processing times between 6 and 8 ms, the estimated round-trip latencies are 4 ms and above. The two way latency requirements rapidly increases with the processing times. Within these bounds, the required peak throughput values lie around 0.85 Gbps on the uplink.

AR Use Case B: Object Detection and Segmentation

In this case, the only offloaded process is the object detection and segmentation algorithm. The uplink stream corresponds to the RGB and depth feeds, while the downlink only includes the single-channel segmentation mask. This setup requires uplink and downlink rates of around 2.5 Mbit and 0.33 Mbit per frame respectively. To analyze the current use case, we choose the same AR processing times span as in the previous example. As expected, we can observe in Fig. 2.2 how the downlink peak throughput requirements considerably drop below 0.4 Gbps as the downlink data steam is slimmer than in the previous use case. On the uplink side, the required peak throughput remains high, with estimated values below 1.2 Gbps. The estimated roundtrip latency is higher than 5 ms for AR processing times below 8 ms.

AR Use Case C: Occlusion Handling Offloading

Occlusion handling algorithms require at least the depth and RGB frames to be constantly transmitted. Besides, some state-of-the-art algorithms use the output from the 3D reconstruction and hand tracking algorithms, which in this case are sent from the device. Consequently, we estimate the required input to weight 4.17 Mbit per frame while the output is lighter: 0.33 Mbit per frame from the high-resolution occluding mask. We can observe in Fig. 2.2 how the estimated peak throughput and round-trip latency requirements slightly increase compared to the previous use case. However, real-time occlusion is a hard and unsolved problem which we can expect to require higher AR processing times. In that case, the required round-trip latency might drop below 4 ms with uplink peak throughput values above 1.4 Gbps.

2.3.2. VR Use Cases

We propose three main distributed VR implementations as reference use cases for our throughput and latencies KPIs analysis.

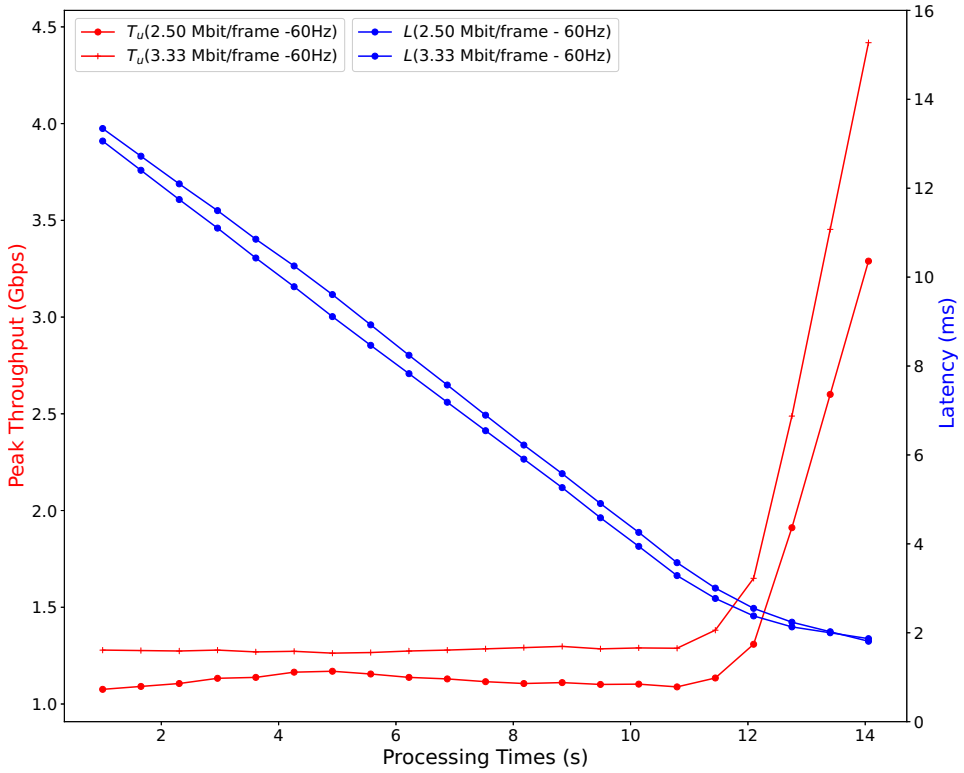


Figure 2.3: Peak throughput and latency results for different values of processing times t_p for the VR rendering offloading scenario (use case A).

VR Use Case A: VR Rendering Offloading

In this scenario we consider that the only offloaded process is the rendering step. In high resolution VR applications this is the most demanding step both in terms of data transmission and computational requirements, and can be considered our KPIs baseline. In this scenario, the uplink data rate is neglected as no media is transmitted, only virtual object updates and other metadata. On the donwlink side, the transmitted data is the rendered frame, which consists of 2 4K frames, one for each eye. According to Table 2.2, the downlink frame size lays between 2.5 and 3.33 Mbit. The update rate used for this first experiment is set to 60 Hz. For this first experiment we used $w_{T_u} = 0.25$, $w_{T_d} = 0.25$, $w_L = 0.25$ and $w_t = 0.25$ so no target parameter is prioritized during the optimization.

In Fig. 2.3 we can observe the output from the proposed experiment. As we described in Section 2.1, the rendering times for high-end devices usually require a minimum of 5 ms to render a new frame depending on the hardware and setup used. For processing times above this value, we estimate round-trip transmission latencies decreasing from 8 ms. On the other hand, the downlink peak throughput estimated requirements are already quite severe, oscillating between 1.2 and 1.3 Gbps.

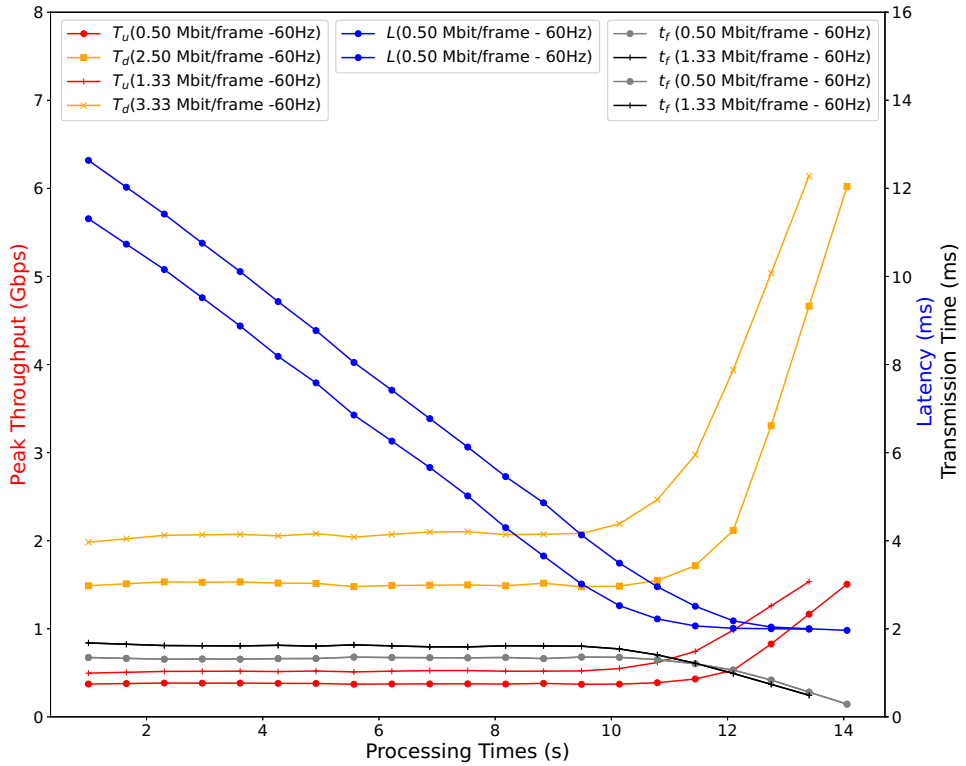


Figure 2.4: Peak throughput and latency results for different values of processing times t_p for the VR rendering offloading scenario with uplink transmission of depth sensor data for real-time processing (use case B).

VR Use Case B: VR Rendering Offloading and Depth Sensor Data Uplink Transmission

In this experiment, we consider the case in which the rendering is offloaded from the device along with another depth-based algorithm, such as hand tracking. We chose this scenario as it requires high uplink data rates without being as demanding as the full offloading scenario. Consequently, the downlink frame size is identical to the one described in the first scenario. On the contrary, in this case the uplink frame size is estimated to be between 0.5 and 1.33 Mbits. The update data rate is again considered to be 60 Hz. In this case, we used $w_{T_u} = 0.25$, $w_{T_d} = 0.25$, $w_L = 1$ and $w_t = 0.25$ to try to prioritize the latency optimization and target values some milliseconds above the low threshold.

Fig. 2.4 depicts the obtained values for this second experiment. From the analysis in previous sections we can expect processing values below 10 ms. We can observe in Fig. 2.4 how the required round-trip latency decreases below 3 ms, with minimum values above 9 ms. Besides, the peak downlink throughput remains stable around 2 Gbps for processing times below 10 ms. On the uplink side, we estimated peak throughput values around 0.4 Gbps. In both the downlink and uplink, the peak throughput starts exponentially increasing for processing values above 9 ms. In this experiment, we can also observe the black and gray curves which determine the time

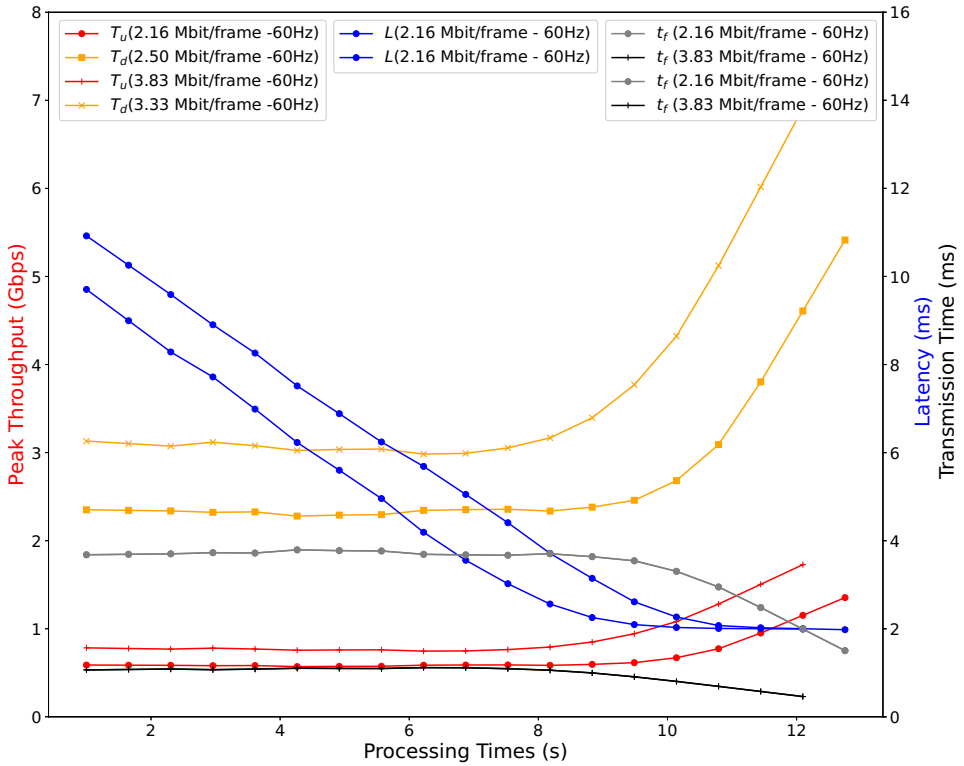


Figure 2.5: Peak throughput and latency results for different values of processing times t_p for the full VR offloading scenario (use case C). Update rate: 60 Hz.

in ms for which the network must sustain the uplink and downlink peak throughputs respectively.

VR Use Case C: Full VR Offloading

In this last experiment, we propose the full VR offloading scenario in which the entire processing pipeline is offloaded from the device. In this scenario, the downlink is the same as in the previous experiments: the high-definition rendered frame. However, the uplink transmits now the full sensor data. We only consider the stereo RGB and depth frames: other data, such as audio or IMUs measurements can be neglected. In this scenario, we have an uplink frame size that lays between 2.16 and 3.83 Mbit. This is the most interesting scenario as it demands the maximum from the network at both the downlink and uplink sides. The used update rate is kept at 60 Hz. We also used the same weight values as in the previous experiment.

Fig. 2.5 shows the results from this last experiment. We can observe how the uplink peak data rate has similar values to the ones obtained in the previous cases. However, the uplink transmission time is more than 3 times higher. On the other hand, the downlink peak throughput requirements have considerably increased with values around 3 Gbps. On the contrary, there are no considerable changes on the latency as the optimization problem is configured to keep the latency values high.

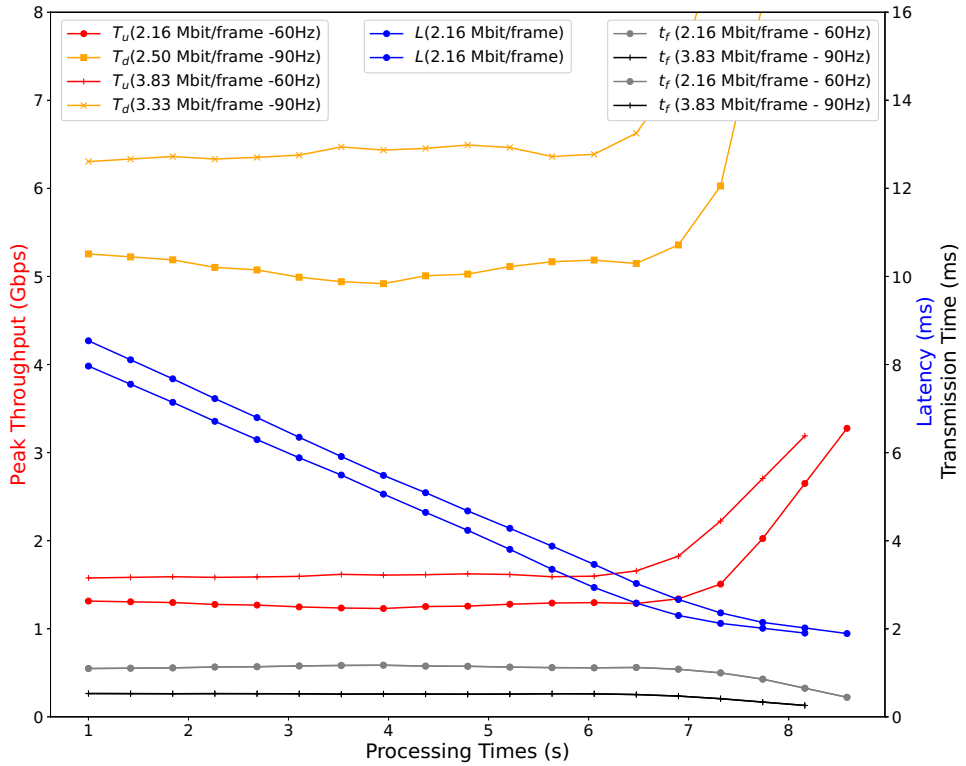


Figure 2.6: Peak throughput and latency results for different values of processing times t_p for the full VR offloading scenario (use case C). Update rate: 90 Hz.

Finally, we repeated the same experiment but setting the update rate at 90 Hz. We can observe in Fig. 2.6 that in this situation, the peak throughput requirements are extremely high with peak throughput above 5 Gbps on the downlink with two-way latencies below 4 ms.

2.3.3. 5G RAN Configuration

To fulfil the above detailed requirements, we propose an initial 5G RAN configuration which can lead to a successful XR offloading schemes. In general, XR offloading demands tight peak throughput requirements on the uplink side, with values above 1 Gbps. On the downlink side, the required peak throughput can reach similar values. Finally, the required roundtrip network latency might decrease to values below 4 ms if we are constrained to hardware providing XR processing times above 8 ms. These requirements demand a very specific and well-designed network configuration. The required roundtrip latencies are extremely hard to achieve for several reasons. Current RAN are very complex systems with very different processes which add extra latency to the transmission. First, low latency demanding processes must run as close as possible to the gNB, avoiding any backhauling connectivity if possible. Consequently, the use of well-equipped Multi-Access Edge Computing (MEC) systems is crucial. There are other potential sources of latency such as the HARQ process. This acknowledgement-based error control tool is a key element of current

wireless networks. However, it can induce high packet transmission latencies if many retransmissions are required. To avoid this scenario, the Modulation and Coding Scheme (MCS) selection must be well aligned with the current network conditions. Incorrectly selected MCS might produce high error rates and, consequently, higher latencies and peak throughput requirements.

Consequently, the scheduling algorithm must be designed to prioritize the XR user. This prioritization is crucial both to achieve high peak throughput and low latency communication. Packets corresponding to the prioritized user should be allocated as soon as they are ready, along with any necessary packet retransmission. Besides, the TDDs scheme should be carefully selected. In the AR use cases, there is a higher demand on the uplink side, TDD slot configurations must prioritize the uplink stream. According to the 3GPP TS 38.213 [49] specification, TDDs configurations 34 to 42 and 50 to 55 prioritize higher the uplink stream. On the other hand, VR use cases impose much higher throughput requirements on the downlink side. Therefore, an ideal TDDs configuration for VR use cases might be 11DL:1GP:2UL TDDs. Ideally, in both AR and VR the slot configuration should be dynamically selected depending on the network status, connected users and scheduling decisions, as described in the 3GPP TS 38.213 specification [49].

From the resource allocation side, it is key to select the proper subcarrier spacing (or numerology) to ensure that the network can provide the required data rates. According to the 3GPP TS 38.306 specification [51], the estimated data rate requirements can be sufficed using a subcarrier spacing of 120 KHz (numerology 3) and a bandwidth of 400 MHz. High numerology also has positive impact on the latency, as they correspond to shorter Orthogonal Frequency-Division Multiplexings (OFDMs) symbols. In all scenarios, 256-Quadrature Amplitude Modulations (QAMs) modulation must be used to ensure high bit efficiency and sufficient available throughput. However, VR use case C corresponding to the full offloading scenario, imposes requirements which can only be fulfilled with advanced 5G network deployments, such as mmWs frequency, multiple MIMO layers and carrier aggregation solutions enabling bandwidths above 2 GHz according to the 3GPP TS 38.101 specification [52], becoming a key enabler of portable immersive XR. Besides, we would need to make use of other crucial capacities of 5G, such as Non-Public Networkss (NPNs) which, as described in 3GPP TS 28.807 [53], allow controlled user prioritization and very specific scheduling schemes.

XR traffic should travel through a specific network slice where RAN resources are properly reserved and Quality of Service (QoS) policies are established throughout the network. This slice could coexist with other slices in the same NPN with different requirements (e.g., mMTC for Internet of Things (IoT) devices in a connected factory, and eMBBs, mostly with downlink capabilities, for general intranet/internet access).

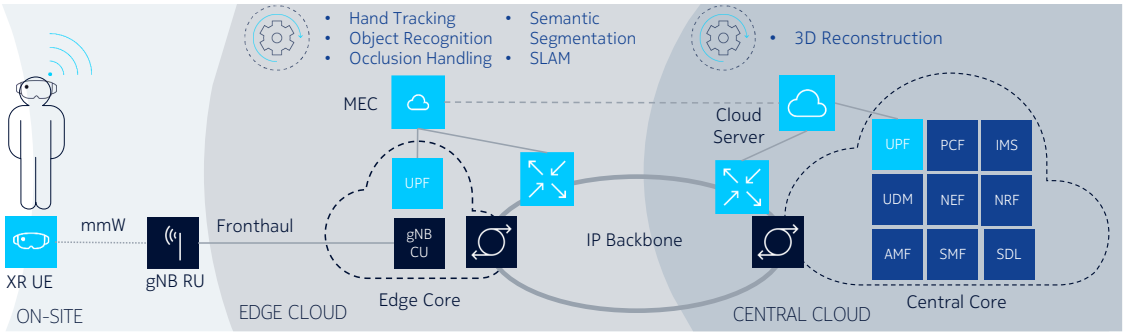


Figure 2.7: Proposed 5G-RAN architecture for a succesful distributed XR deployment.

Finally, as the offloading sets demanding hardware requirements for the MEC, particularly in terms of GPUs resources, it is important to use dynamic resource allocation and virtualization. The idle MEC capacity available in low-usage hours (e.g. during the nights) may be dynamically released to other tasks that can be performed offline and scheduled for such hours (e.g. heavy data analytics). Therefore, the XR slice should also be able to acquire and release the computing resources whenever they are needed.

The proposed 5G network architecture for XR offloading, is shown in Fig. 2.7. We can observe how the latency-constrained algorithms, such as dynamic occlusion, semantic segmentation, and object tracking, run on a MEC with no required backhaul connection. Other non-real-time processes, such as 3D reconstruction, can run on a further server, which involves the backhaul link and the 5G core.

2.3.4. Conclusions

In this chapter we have analyzed the main technology and network requirements for an immersive XR distributed architecture and how 5G can become a key enabler of fully immersive XR. We have described the current state of the art of XR algorithms, along with their individual required inputs and outputs, and hardware requirements. We have studied the latency and peak throughput requirements necessary to successfully offload some of the described algorithms. Furthermore, we have numerically exemplified how the latency, throughput and XR processing times are related with each other, proposing possible network configurations that may guarantee them. The estimated required peak throughput and latency values call for the use of 5G milimeter wave spectrum along with massive MIMO, smart resource allocation slicing, and NPNs. Besides, we have described the importance of novel and optimized scheduling approaches to fullfil the tight requirements for XR offloading. Finally, we characterized the parameters and algorithm distribution of a plausible 5G network architecture to enable the presented distributed XR offloading implementation. As a main conclusion we can foresee that the time has come for

the implementation of XR on feasible devices thanks to the efficient architecture, throughput and latency enabled by 5G.

Chapter 3

EXtended Reality Offloading Architecture

The KPIs described in Chapter 2 are a crucial starting point to design an optimized offloading architecture which can fulfil the demanding throughput and latency KPIs necessary to offload the most relevant and impacting XR algorithms. In this chapter we present our offloading architecture. The first and most stable version of our architecture relies on TCP protocol and individual JPEG encoding and has been carefully designed to reduce any processing overhead which can degrade the network performance. We are currently implementing RTP and H.264 functionalities which are also presented in this chapter. However, as the new functionalities are still a work in progress, we only include consolidated results from the TCP version of the architecture. The TCP version of the architecture has been tested for different offloading scenarios and conditions on two different wireless networks: WiFi and 5G millimeter wave technologies. Besides, to test the network on alternative millimeter wave configurations, currently not available on the actual 5G millimeter rollouts, we used FikoRE, our open source 5G RAN real-time emulator, fully described in Chapter 5. The results achieved with the TCP version of the architecture show great performance for the tested immersive media scenarios, highlighting the relevance of millimeter wave technology for the future of immersive media applications.

The goal of this chapter is to give a practical overview of our functional offloading implementation and present some preliminary results while highlighting what other improvements are still required both from the network and the architecture sides to reach XR offloading full potential. Our contributions are:

- Design and implementation details of our novel offloading architecture, including both the TCP and RTP-based functionalities. The architecture handles multiple streams from different users, allowing concurrent offloading services and traffic routing while aiming for a low latency and reliable data exchange.
- Description of the experimental approach for testing our architecture in different offloading scenarios and wireless networks: we describe the set of experiments we carried out to show the performance of both our architecture and the network itself. The main goal of these experiments is to show what can currently be achieved and what is still lacking from both the network and architecture.
- Brief presentation of the obtained results: we show and analyze the obtained results, paving the ground for future researchers.

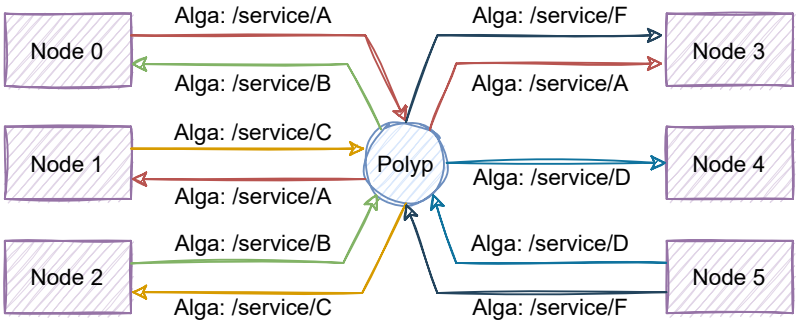


Figure 3.1: Simple example of a distributed system implemented with the architecture main components: Alga and Polyp.

3.1. Architecture Description

The main goal of the implemented architecture is to ensure a reliable, scalable and flexible offloading tool optimized for low latency communication. We understand the proposed architecture as a service provider for immersive media applications: the architecture back-end provides a set of services to which the immersive device subscribes to. Consequently, the data flow follows a publisher-subscriber approach, which facilitates the dissemination of the information in distributed systems [54] as the one we are proposing. We refer as nodes to each individual component of the distributed architecture which subscribes or publishes to an available data channel (see Fig. 3.1). The available data channels advertised by other nodes are referred as topics. Our architecture is composed of two main components which are in charge of efficiently distributing the information between the different nodes:

- Alga: is the core communication custom library which allows the direct data exchange between nodes. Alga allows a node to publish or subscribe to one or more topics, handling the data reception and transmission.
- Polyp: is the traffic routing agent which maps the publishers with their correspondent subscribers. Polyp receives publishing or subscribing petitions from the connected nodes and routes the traffic accordingly: it receives packets from the nodes, check their destination and route them to all the targeted nodes. Consequently, polyp has to be designed and implemented to ensure high data managing efficiency to avoid any unnecessary delays.

The idea of the proposed architecture is that the potential offloaded algorithms are offered by a MEC or a distributed system of processing servers as services. Each of the offered services' inputs and outputs are associated to topics. Each immersive application can publish their sensor data to arbitrary services, and can subscribe to the service's outputs using their unique topic (see Fig. 3.2). With this approach not only the scalability of the proposed pipeline is ensured but it also simplifies the implementation efforts for the immersive applications providers.

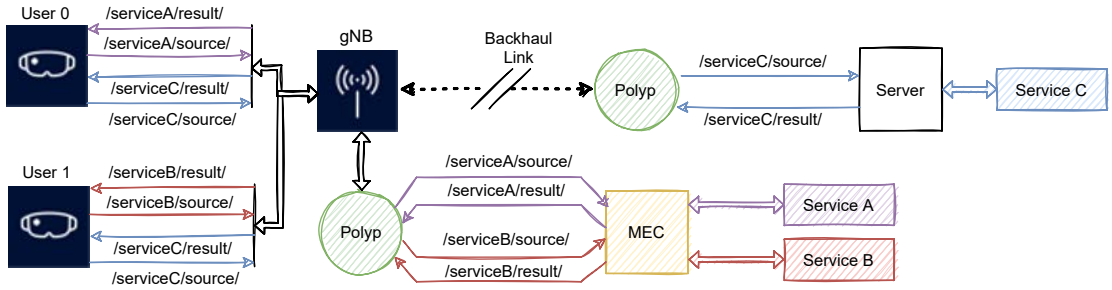


Figure 3.2: A distributed offloading implementation for immersive applications using the proposed architecture and 5G.

To keep the communication latencies low, there should be at least one Polyp instance running as close to the MEC and the serving g-Node B (gNB) as possible. Polyp is in charge of routing the traffic from and to the services running in different instances within the nearby MEC. Besides, our architecture is designed to handle other services which does not require real-time processing: for these services, Polyp can communicate with other Polyp instances through the internet. In this scenario, the immersive application can stream their sensor data to further servers to do non real-time heavy processing, such as photogrammetry realistic reconstruction [55]. Consequently, the proposed architecture can simultaneously handle real-time and non-critical services using the same protocol and data flow.

3.1.1. Alga: Implementation Details

Alga is implemented as a reliable publisher-subscriber communication library which relies on two different protocols: TCP and RTP. In both options, we have built both the publishers and subscribers to allow both synchronous and asynchronous communication. In the synchronous mode, both the receiving and sending steps block the main process. In the asynchronous mode, we create an independent thread for each opened socket so that that receiving and sending steps do not block the main process. Besides, we have implemented and efficient callback system for the subscribers. Any custom method can be attached to such callback to handle the incoming messages at will. In general, the RTP version of our architecture is designed for video streaming. On the other hand, metadata and other sensor data, such as IMUs, are transmitted using our TCP-based implementation.

TCP

We implemented TCP-based transmission in our architecture for use cases in which we need to prioritize reliability over maximizing the throughput. In both AR and VR lost frames suppose a great degradation of the user experience, specially for the most latency-critical algorithms such as VR rendering or hand segmentation.

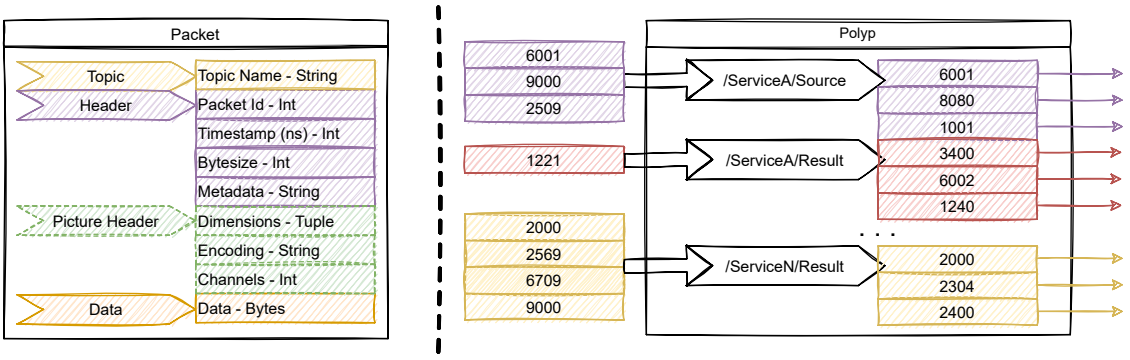


Figure 3.3: Left: Detailed structure of our custom packets. Right: Schematic simplified representation of Polyp's internal design.

The TCP functionality of Alga is implemented using a well-known and well documented TCP-based communication library: ZeroMQ[5]. ZeroMQ has been widely tested and bechmarked, showing outstanding performance both in terms of latency and throughput [56]. ZeroMQ allows to simply connect to an arbitrary endpoint or bind to a socket and start receiving and sending data through that port. Binding is a key functionality as it allows to listen to all the packets coming through an arbitrary port regardless the endpoint source. Besides, ZeroMQ already allows to configure ports as publishers or subscribers, discarding the received packets which do not correspond to the subscribed topic. This feature, combined with the binding capabilities, allows a fast implementation and an optimal performance as it discriminates the packets coming to a bind port by topic, avoiding any extra processing on the binding side.

We have implemented three types of publishers/subscribers, which use different kinds of custom packets. All of the packets are divided into three sub-packets: topic, header, and data. The topic is a simple string with the topic to which the socket is subscribed or publishing to. The header includes relevant metadata: packet id, timestamp, byte size, and a free field for other metadata, see Fig. 3.3 for a detailed structure of our packets. The header and data changes depending on the publisher/subscriber type:

- Picture: it allows to publish or subscribe to three or four channels color images. It implements JPEG encoding and decoding capabilities, and is able of transforming the image data to bytes, and reconstructing the image when received. The header in this case also adds the image metadata: height, width, channels, and encoding format. The data sub-packet includes the image (encoded or not) bytes.

- Unsigned 8 Bits Picture: is specifically designed for semantic segmentation algorithms offloading, in which the result is a single channel 8 bits frame. The

[5]https://zeromq.org/

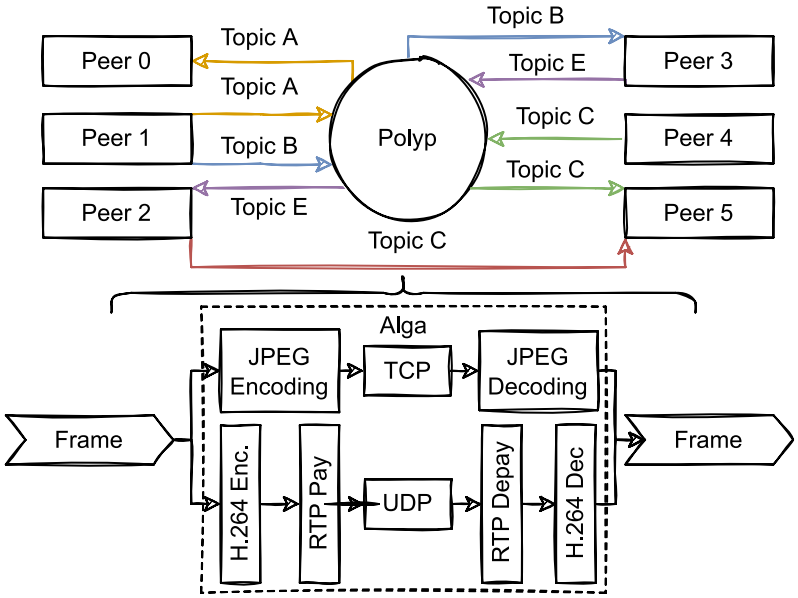


Figure 3.4: The proposed offloading architecture strategy and simplified data flow for a general multi-peer scenario (top). Alga data flow for both TCP-JPEG and RTP-H-264 implemented transmission pipelines (Bottom).

implementation also allows single channel JPEG encoding and decoding. The header in this case removes the channel information as it is no longer required.

- Metadata: this type is designed to transmit metadata such as configuration information, position and orientation updates or the result from algorithms such as object tracking which outputs just positions, orientations and sizes. The header in this case is the originally described one. The data in this case is sent as regular strings.

RTP

While JPEG encoding allows to have full control on the individual frames, it is not as efficient in terms of compression as video-targeted encoders such as H.264. We decided to implement H.264 encoding capabilities to our offloading pipeline. However, transmitting encoded video over TCP is not efficient: video traffic is typically split in packets of around 1.5 kbytes, considerably increasing the TCP's ACK-NACK overhead. Therefore, we decided to implement RTP over User Datagram Protocol (UDP) as an extra functionality of Alga.

The camera or rendering streams can now be transmitted as H.264 encoded video. For this purpose, we incorporate GStreamer[6] in charge of encoding and decoding the video stream and transmit or receive the RTP frames. For traffic control reasons, and to make it compatible with Polyp's and TCP-Alga's functionalities, we need

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

to have control over the individual video frames and attach them the associated metadata, such as destination topic, timestamps, etc. This metadata can be useful also for performance analysis or bottleneck detection. To achieve this goal, we use RTP extended headers. The metadata is then added to each video frame as an RTP extended header, which can be decoded and read on the receiving end. This is achieved using GStreamer in-built functionality. Alga's data flow for RTP and H.264 modes is depicted in Fig. 3.4.

From the sending peer, the frames are fed to Alga in RAW RGB format. Alga injects the RAW frame along with its associated metadata to GStreamer encoding and transmitting pipeline. If the traffic goes through Polyp, it is in charge of transmiting it to the subscribed peers. In both this case or the case in which the traffic is transmitted between final peers, GStreamer pipeline receives and decodes the RTP frame to the specified format. Once decoded, the frame can be accessed by the application layer.

3.1.2. Polyp: Implementation Details

Polyp has been designed to efficiently route traffic coming from a socket to another arbitrary socket. The key component of Polyp is the mapping between the topics and destination ports. When a new service or node is added to the system, Polyp receives the relevant information: subscriber/publisher(s) topic(s) and the correspondent endpoint(s). Polyp then maps these topics with such ports so it only requires the topic information from the incoming packets to correctly route them, see Fig. 3.3 for a schematic representation of Polyp's inner processes.

Polyp is implemented to support TCP traffic via ZeroMQ and UDP/RTP traffic. The rules to determine which type of traffic to expect coming in and out of each topic are selected by the user and can be dynamically modified. Polyp binds to a set of arbitrary ports to which the nodes and services connect to receive or send data through an arbitrary topic. When handling TCP traffic, Polyp relies on ZeroMQ inbuilt topic discrimination and binding capabilities. We implemented our own logic to handle the topic-based routing for RTP/UDP traffic.

3.2. Experiments and Results

All the experiments and results described in the the following sections are focused on the TCP-based version of the proposed architecture. While the architecture has been already tested in field VR offloading scenarios in wireless networks [47], we wanted to benchmark it in different scenarios and wireless networks. Consequently, we designed our experimental setup to test a wide range of combination of scenarios and offloaded algorithms. With these experiments, we aim to understand the current

limitations of our implementation and study how we and other researchers can move toward an even more optimal offloading solution.

We focused in three main offloading scenarios. In all the cases, the uplink feed is the sensor data, which we consider to be just a single camera feed. The main difference between the scenarios is the resulting data, which is sent back through the downlink stream to the device:

- Scenario A - Full offloading: in this scenario the donwlink side is composed by the rendered frame, which is sent back to the device. This is the most latency-critical scenario, as lost or late frames can produce nausea or discomfort to the user.
- Scenario B - Real-time segmentation offloading: in this case, the downlink stream includes the individual frames resulting from the segmentation algorithm.
- Scenario C - Light downlink algorithms: some heavy algorithms, such as object tracking [14] or simultaneous localization and mapping (SLAM) [38], only output some metadata as the only results. This metadata is sent back to the device.

The last two scenarios are less restrictive in terms of latency if their implementations include any latency correction algorithm, allowing to have end to end latencies above the sampling rate period. Even though current AR and VR devices already include time warping capabilities to reduce the effects or rendering delays or jitter, it is still crucial to decrease latencies to the minimum, especially in scenario A. The greater these latencies are, the harder it is for the time warping algorithm to overcome its effect. If the offloading architecture adds extra latency, the correction effect of the time warping algorithm would decrease, so we aim for the architecture to not add extra frames of latency which could degrade the experience even if time warping is available.

We decided to test the selected scenarios in three different networks to understand how our architecture adapts to their particularities. Our goal is to understand the limitations of both our architecture and the network in each offloading scenario:

1. WiFi: we decided to test our architecture on a WiFi network as this technology is well stablished as the most used wireless network for indoor tasks. The outstanding performance of the newest releases, such as the release 802.11ax which allow throughputs way higher than 1 Gbps [57], adequate for immersive media offloading. In our experiments we use a Netgear R6400 router.
2. mmW 5G Network: mmW technology is considered to become one of the key enablers of novel technologies, including VR and AR offloading, over the

coming years. Consequently, we decided to test our architecture on a mmW prototype we have access to. The setup we used incorporates 8 Component Carriers (CC) with 100 MHz of bandwidth each. It is configured with numerology 3 which corresponds to a carriers sub-spacing of 120 KHz. Only two of the available subcarriers are configured to allow uplink traffic, using a 1UL:4DL (1 uplink slot granted for every 4 downlink slots assigned) TDD configuration. We used the Askey RTL6305 mmW modem which, by the time we carried out the experiments, was not capable of aggregating the uplink subcarriers. Consequently, on the uplink side only 100 MHz of bandiwdth with TDD 1UL:DL were available. The experimental setup is using 256-QAM modulation. Millimiter wave communication is a recently introduced technology which is still in a very early stage, both from the modems and gNB sides. However, we still considered relevant for the research ecosystem to test the architecture on our mmW setup as we could gain insights on how to optimize our architecture for the future of this groundbreaking telecommunication technology.

3. 5G-RAN Emulator: we decided to use our FikoRE, our open-source 5G-RAN emulator[25], described in Chapter 5, to test the network with different and currently not commercially available configurations. More specifically, mmW possible configurations are still limited, constraining the uplink scheduling grants. Besides, mmW commercial modems have not reached their full potential yet. For this reason, we have decided to emulate in real-time a mmW gNB with optimal configuration parameters. The main goal is to emulate, with a high level of accuracy, how our architecture performs on a mmW setup better configured and optimized for both uplink and downlink communication. Besides, FikoRE allows to model other latencies such as the link latency between gNB and a nearby MEC. In the proposed experiments, FikoRE runs on a Aorus Laptop with 16 GB of RAM and an Intel® Core™ i7-10870H CPU @ $2.20GHz \times 16$.

3.2.1. Architecture Setup

We decided to test an architecture setup in which a device is offloading an algorithm to a MEC. The MEC has several offloading services available, and an instance of Polyp runs on it, handling and routing the income data to and from the target service. Fig. 3.5 depicts the architecture setup and data flow we have used for the experiments. Even though we are aware of the importance of the selected packetization schemes or video encoding/decoding techniques to the overall behavior of the architecture, our main goal for the proposed experiments is to benchmark the throughput and latency capabilities of our implementation. Consequently, we decided to send the frames individually and remove all the processing overhead non related to the architecture itself, such as the encoding and decoding steps.

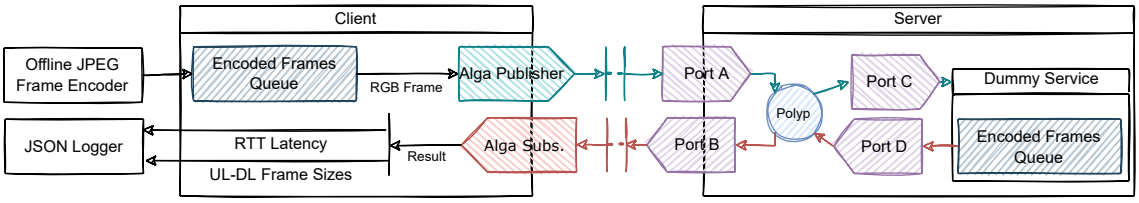


Figure 3.5: Detailed data flow of the used experimental setup.

Therefore, we used the same 4K (3840 × 2160) ten seconds video for all the experiments. We resized each frame of the video to resolutions 2, 4 and 8 times smaller than the original, for a total of 4 videos. Then, we encoded each frame of each video to JPEG and store them in memory. These files are used both in the server and client to load the frames that are sent each time. The same process is done for the frames corresponding to the resultant segmentation mask in the offloading scenario B: each frame is offline masked, according to a random color mask, and encoded to single channel JPEG. The process is repeated for the 4 chosen resolutions.

While the architecture is also prepared to receive and send packets asynchronously, to isolate and fairly evaluate each individual frames transmission, the data flow is synchronous (see Fig. 3.5) and follows the next scheme:

- When the result from the previously transmitted frame is received, the client takes an encoded frame from the stored files and sends it through its publisher to an arbitrary port.
- Polyp, running in the same machine as the server, receives the packet and retransmits it to the receiving port to which the target service is attached.
- The target service receives the packet on the subscriber. In this case, we use a dummy service which just discards the packet and immediately loads an encoded frame from the previously created files and sends it back to the client. If we are on the offloading scenario C, only random metadata of fixed size (360 bytes) is sent back.
- Polyp re-routes the received reply from the server to the correspondent receiving port on the client side.
- The client receives the reply, discards it, logs the round trip time and uplink and downlink frame sizes, and re-initializes the loop.

3.2.2. Polyp Evaluation

The first step was to evaluate the overhead introduced by Polyp's packet routing. As we consider Polyp to be running in the MEC which is offering different offloading

Table 3.1: Decomposition of the source and results frame size used in each experiment.

	A - Rendering Offloading				B - Segmentation Offloading				C - Metadata Offloading		
	UL		DL		UL		DL		UL		DL
	Pixels	Mbits	Pixels	Mbits	Pixels	Mbits	Pixels	Mbits	Pixels	Mbits	Mbits
R1	3840x2160	8.14	3840x2160	8.14	1920x1080	2.81	1920x1080	2.09	1920x1080	2.81	<0.001
R2	960x540	1.00	3840x2160	8.14	960x540	1.00	960x540	0.83	960x540	1.00	<0.001
R3	540x270	0.35	1920x1080	2.81	540x270	0.35	540x270	0.32	540x270	0.35	<0.001

services, we can assume Polyp's routing is done on the local host network. Consequently, we decided to run the entire proposed experimental pipeline in the same computer for different source and result frames and trace the overhead latencies added by Polyp. We repeated the experiments 4 times, one for each resolution. In all the 4 rounds of experiments, the time overhead mean was in all cases smaller than 1 ms.

After this simple experiment we can conclude that in this type of setup, in which Polyp is running in the same machine as the offloading services, there is no relevant time overhead added by Polyp. Therefore, in the following experiments we do not use Polyp to route the packets to the dummy server, and we directly connect or bind the server's and client's publishers and subscribers.

Alga Results

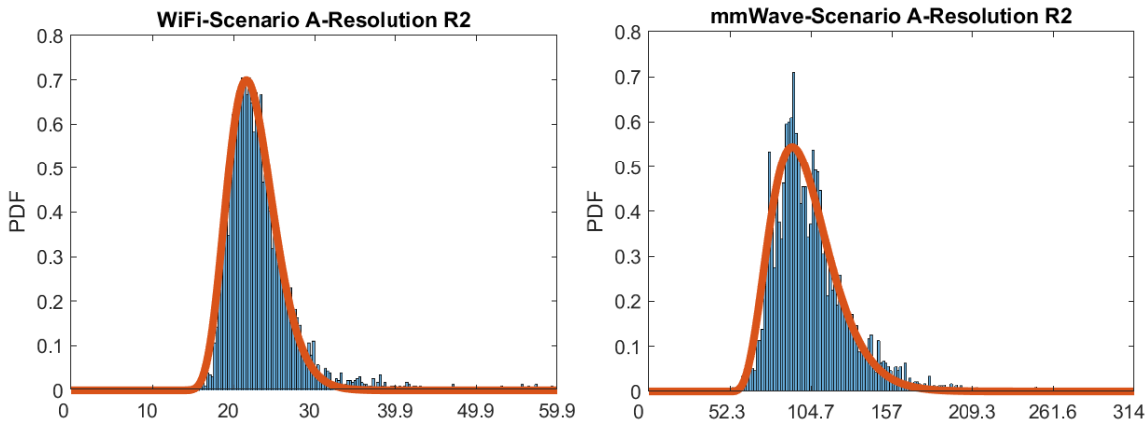
We did not evaluate all the possible combinations of source and results frames. On the contrary, we chose only the most representative combinations, shown in Table 3.1. The sizes shown in Table 3.1 are the mean sizes of all the JPEG-encoded frames with the given resolution. As in scenario B the downlink frames are single channel JPEG-encoded masks, we can observe their sizes to be smaller than the ones with same resolutions in the other scenarios. We have three different resolution setups for each scenario. Table 3.1 defines the IDs for each experiment, being R1, R2 and R3 the 3 resolution combinations for each offloading scenario, A, B or C.

We also analyzed the basic performance of both the WiFi and mmW setups. We used iperf[7] to estimate the TCP throughput capabilities. We let the test run for $100s$ in each setup, with no other users connected. In FikoRE case, we assumed a one way latency between the core and the MEC of 3ms. FikoRE is configured exactly as the actual mmW setup, but assuming the 1UL:4DL TDD configuration is extended along the 8 subcarriers, and the modem is capable of performing carrier aggregation. This configuration multiplies by 8 the actual uplink throughut capabilities of the actual mmW experimental setup. The simulated modem is placed 50 meters away from the gNB. This configuration is used in all the emulated experiments. The

[7]https://iperf.fr/

Table 3.2: Summary of each network setup's Iperf performance.

1 - WiFi			2 - mmW			3 - Fiko - 5G-RAN Emulator		
DL (Mbps)	UL (Mbps)	Ping (ms)	DL (Mbps)	UL (Mbps)	Ping (ms)	DL (Mbps)	UL (Mbps)	Ping (ms)
∼200	∼200	<4	∼3000	>80	<12	>3500	>1500	6



(a) (b)

Figure 3.6: Example of the estimated PDF (in orange) for the obtained round trip times histograms in milliseconds (in blue) for two arbitrarily chosen examples: Scenario A and resolution R2 for both WiFi (a) and the mmW base station setup (b).

summary of the achieved TCP performance on Iperf in the three cases is shown in Table 3.2. The ping values measurements were taken with empty buffers.

We are using the same set frames along the duration of each experiment. As a consequence, we only need to focus on the measured Round Trip Time (RTT) latency on the application layer. For each experiment, we transmitted both ways a total of 10000 frames through the architecture. In each iteration, the round trip time was measured and saved for later analysis. We decided to obtained the RTT latencies' probability density functions for each experiment, giving a statistical view of the RTT performance of our architecture.

By a quick inspection, our intuition indicated that the Probability Distribution Function (PDF) followed a Gamma distribution [58] with a certain time offset. This intuition matches the fact that any delay in a communication network is defined by a minimum value (the offset), which is caused by physical and computing limitation factors, with a greater occurrence in the delays close to the offset and some values with a much lower occurrence in higher delays. Besides this, our intuition was supported by a study on the prediction of RTT for wireless networks [59]. The definition of the offset Gamma distribution can be expressed as:

$$f_X(x, x_i, \alpha, \theta) = \frac{(x - x_i)^{\alpha-1} e^{-(x-x_i)/\theta}}{\theta^\alpha \Gamma(\alpha)}, \text{with } \Gamma(\alpha) = \int_0^\infty \frac{t^{\alpha-1}}{e^t} dt, \tag{3.1}$$

Table 3.3: Summary of the estimated PDF values for each experiment: rendering (A), segmentation (B) and metadata (C) offloading; and high (R1), medium (R2), and low (R3) frame resolution. See Table 3.1 for details.

	WiFi								
	A-1	A-2	A-3	B-1	B-2	B-3	C-1	C-2	C-3
Mean (ms)	36,43	23,54	10,58	25,36	11,61	3,98	8,43	4,23	2,52
95th Percentile	43,35	31,21	13,52	40,92	19,84	5,50	10,47	5,56	3,68
Std (ms)	4,52	5,33	1,86	8,97	4,61	0,97	1,32	1,02	0,86
Offset (ms)	23,75	12,43	7,17	9,63	5,32	2,61	5,88	2,86	1,78
Shape	9,8	11,46	5,96	3,93	2,49	2,56	4,81	3,69	1,09
Scale	0,24	0,17	0,34	0,52	0,74	0,5	0,32	0,18	0,39
	mmWave								
	A-1	A-2	A-3	B-1	B-2	B-3	C-1	C-2	C-3
Mean (ms)	190,53	105,83	51.05	58,22	27,11	19,07	42,80	22,39	17,95
95th Percentile	259,82	153,00	70,62	82,05	36,47	27,46	61,80	27,81	25,36
Std (ms)	41,8	31,04	18,24	35,87	8,12	6,7	11,91	6,35	4,55
Offset (ms)	102,62	51,45	32,72	32,98	21,42	16,85	22,74	8,41	14,98
Shape	4,96	5,79	6,9	2,55	1,1	0,24	3,88	6,66	0,86
Scale	0,5	0,33	0,09	0,23	0,25	1,51	0,38	0,16	0,41
	FikoRE 5G-RAN Emulator								
	A-R1	A-R2	A-R3	B-R1	B-R2	B-R3	C-R1	C-R2	C-R3
Mean (ms)	16,14	10,76	8,67	10,99	8,57	8,51	9,88	8,54	8,5
95th Percentile	17.92	11,69	9,48	12,72	9,44	8,65	10,62	9,41	8,65
Std (ms)	3,14	1,34	0,96	1,49	0,96	0,43	0,82	0,38	0,34
Offset (ms)	14,67	10,38	8,37	10,15	8,34	8,37	9,21	8,32	8,35
Shape	0,96	0,71	1,59	0,59	0,51	1,56	0,82	1,99	1,98
Scale	0,13	0,07	0,02	0,34	0,04	0,06	0,21	0,07	0,09

with α a shape parameter, θ the scale parameter, β the inverse of the scale parameter, and x_i the offset. These are the parameters that need to be adjusted given the input data. We found that 10000 iterations was enough to accurately adjust an offset Gamma PDF, for each experiment, from the RTTs histogram. To adjust the PDF, we defined an optimization function of the following type:

$$\min_{x_i,\alpha,\theta} \sum_{x} |f_X(x, x_i, \alpha, \theta) - h_X(x, s)|, \tag{3.2}$$

where $h_X(x, s)$ is the histogram adjusted to the x axis of the PDF and the experiment s. The optimization problem was solved using Matlab. Fig. 3.6 shows the obtained RTT histograms and their estimated PDFs from two of the presented experiments. We can observe a great level of agreement between the theoretical PDF and the histogram after solving the optimization problem defined in Eq. 3.2.

Apart from this, the mean and variance of the RTT have been calculated, since these statistic parameters are the simplest that characterize any distribution, and can be applied with a good agreement for Gamma distributions. The obtained RTT metrics for each resolution and scenario combination and wireless technology

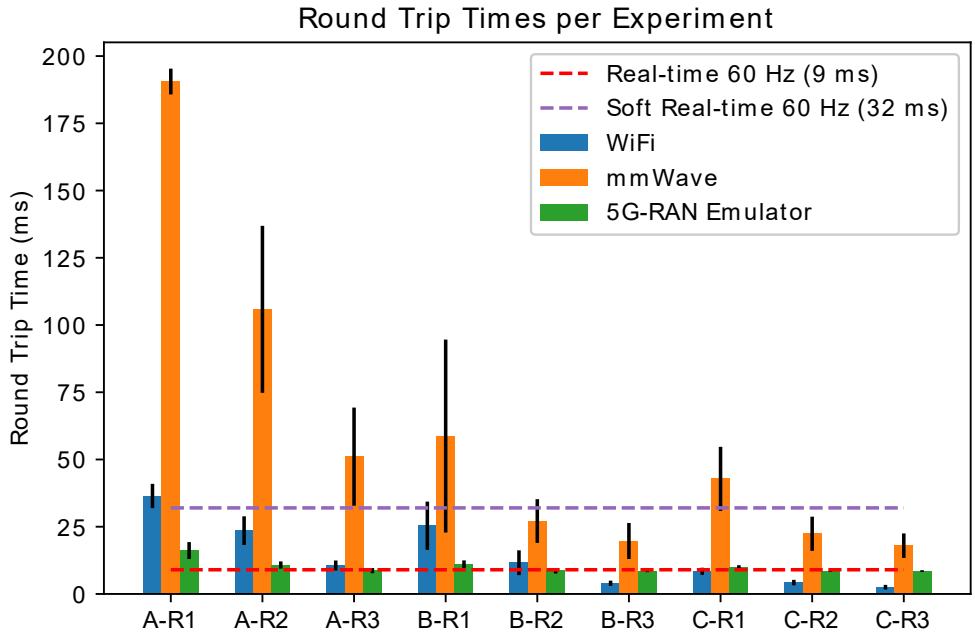


Figure 3.7: Graphical summary of the round trip time results from all the experiments.

experiment is shown in Table 3.3. Besides, and to show when most of the frames arrives on time in each scenario and wireless technology, we estimated the 95th percentiles shown also in Table 3.3. Aiming to give a visual insight of the results we built Fig. 3.7 which depicts the bar plot of the mean and variance RTT from each experiment. Notice that we have added two thresholds in Fig. 3.7: one is delimiting the maximum RTTs which support hard real-time and the other referring to the soft real-time deadline. We understand hard real-time as the process in which the total end to end latency, including the processing overhead, is smaller than the frame update period (16.6 ms). We chose this deadline to be 8 ms according to the processing overhead times assumed in [28]. In specific VR offloading applications, as the one described in [47], higher RTTs (<50 ms) provide a sufficient quality of experience, as buffering techniques can overcome the delay effects. Consequently, we consider 32 ms as our reference soft real-time deadline for this particular less constrained offloading applications.

We can directly observe that the performance on the actual mmWave setup is the poorest one. We were expecting this behaviour as the mmW technology, and especially the experimental setup we have access to, is still improving, with almost no commercial roll-outs worldwide. The available mmW modems are still very limited on the uplink side: the Askey RTL6305 is not performing any succesful carrier agreggation on the uplink. Consequently, the uplink effective bandwidth is limited to less than 30 MHz, considerably reducing the effective throughput. Besides, the fact that our architecture is based on TCP reduces the network exploitation

on poorly performing networks: delay and throughput are competing resources in TCP. However we can observe that in some scenarios, the soft real-time deadlines are achieved. Consequently, for some of the proposed scenarios, the current sub-optimal development stage of mmW technologies is already useful for particular immersive offloading tasks.

Our WiFi experimental setup considerably favors our TCP-based architecture as the measured baseline ping end to end latencies were smaller than 4 ms. Consequently, all the experiments but the most demanding in terms of throughput (A-R1) were meeting the soft real-time deadline. Besides, only three experiments showed deadlines above the hard real-time requirements (A-R1, A-R2, B-R1). These limitations could be overcome using a router which implements MIMO or multilink transmission to provide even higher throughputs.

Finally, as we were expecting a priori, the best performance is given by the emulated mmW setup. First, it is key to acknowledge that in this setup, the entire architecture pipeline was running on the same machine, removing any possible network degradation that could be produced by any of the network components involved. Besides, the mmW configuration used, which assumed the use of modems capable of performing carrier aggregation, enables almost 200 MHz just for uplink transmission. This allows more than 8 times uplink throughput than in the actual mmW setup. We can observe that only the first scenario and resolution combination (A-R1) is not fulfilling the hard real-time requirements. However, we can observe that the mean RTTs are not going below 8-7 ms. This is due to two facts: we modeled a fixed simulated core to MEC one-way latency of 3 ms, and TDD scheduling adds extra non-avoidable latencies, specially on the uplink side. This last limitation could be overcome using network slicing which allows greater scheduling flexibility and use case specific optimizations.

3.2.3. Emulated mmW-based Offloading on a Realistic Scenario

The goal of this last set of experiments is to test our architecture in a more realistic scenario using the 5G-RAN emulator: we simulated multiple users attached to the same base station which are sharing network resources with the virtual immersive user. The goal is to emulate an actual mmW deployment and understand how the performance degrades as other users are connected to the same base station. We included 120 virtual users in the simulator which are consuming 5 Mbps and 1 Mbps of downlink and uplink traffic respectively. We chose these values as they represent the case in which these virtual users are on a video-call, receiving and sending HD 1080 and SD 480 video streams respectively. This number of users was selected as this combination of users and uplink and downlink throughputs sufficiently degrades the overall performance of the offloading architecture. The users are placed randomly within a 1000 m radius from the base station. Besides,

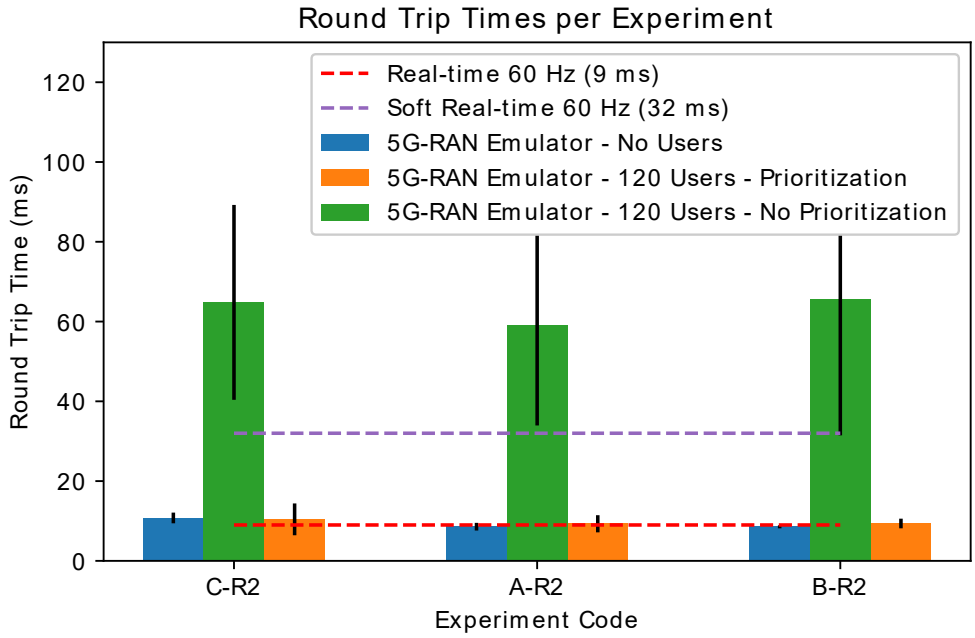


Figure 3.8: Graphical summary of the round trip time results from all the experiments.

the emulator has user prioritization capabilities, handled by the emulator's scheduler implementation. Therefore, we tested the scenario with no user prioritization against the one with it. We used proportional fair [60], a well studied solution, as the ruling scheduling algorithm. As we already have the previous results as a baseline, we decided to focus only on the most relevant resolutions for each offloading scenario: R2 in all the offloading Scenarios A, B, C from Table 3.1. We choose this uplink frames resolution (960x540) as it is, along with 1080x720, a typical resolution for VR offloading applications as the one proposed in [47].

The obtained results are depicted in Fig. 3.8. We can observe the high performance degradation when other users are consuming resources from the base station and no user prioritization is used: measured RTTs grow 6 times bigger than in the no users setups. However we can observe that there is almost no visual RTT values differences between the user-less and prioritization setups. We can, on the contrary, observe that the standard deviation is slightly higher for the scenario with simulated users. This is justified by the fact that, even though the user prioritization is forcing the scheduler to grant transmission slots to the prioritized user, proportional fair scheduling algorithm guarantees periodic serving to the other users. This produces small peaks of latency on the prioritized user, which, as we can observe in Fig. 3.8, are small enough to be neglected.

3.3. Conclusions

In this chapter, we have presented our communication architecture specially optimized for immersive media offloading. First, we have described the main characteristics and implementation details of the proposed architecture, including both main protocols implemented, RTP and TCP, and encoding techniques, JPEG or H.264. As RTP transmission and H.264 encoding functionalities are still a work in progress in our architecture, our experimental evaluation has considered only the TCP-based version of our architecture.

We have described the main experiments that were performed to test our implementation. The architecture was tested on both WiFi and a 5G mmW wave setup. As the current development of mmW wave has still not reached its full potential, we have also tested our network on a 5G emulator which is able of modeling a configurable mmW base station. The goal of this last experiment is to test the potential of our offloading architecture on a fully functional mmW setup.

The architecture was tested on different offloading scenarios, frame resolutions, and the three wireless networks setups: WiFi, mmW, and emulated mmW. The results where focused on measuring the round trip time at the application level for each scenario and wireless technology. The obtained results showed a high performance of our architecture in WiFi, meeting at least the soft real time deadline for every test case but one. On the contrary, mmW results were not as successful: the test cases with highest input or output resolutions did not satisfy the soft real time requirements. This was expected as the current development of mmW technologies is currently limited, considerably constraining the uplink throughput capabilities. However, even with the current limitations, mmW can already be used in several offloading use cases according to the results. Finally, using the emulated mmW setup, configured for exploiting the actual throughput capabilities, we managed to obtain outstanding results with our offloading architecture. The only test case in which the emulated mmW setup could not perform in hard real-time is the scenario in which both the uplink and downlink frames' resolution was 4K. From these results, we conclude that the Scenario A, in which the rendered immersive scene is sent back to the device, which is unavoidably a hard real-time offloading use case, is not well suited for TCP communication protocol. In this case, our work in progress RTP transmission and H.264 video encoding functionalities can provide a more optimized performance for this kind of traffic.

Furthermore, we tested our architecture in a realistic mmW rollout, using the 5G mmW emulator. We added multiple virtual users to the emulated scenario, consuming throughput resources from the base station. When no user prioritization was used, the network degradation was such that the measured RTT increased up to 8 times compared to the no user scenario. We performed the same experiments using user prioritization, showing almost an identical performance as the scenario with no

users. The experiments with the mmW emulated setup has shown the potential of both mmW technologies and the proposed offloading architecture. The combination of these technologies, and the newest releases of WiFi, can become a key technology enabler for the future of immersive media technologies.

These results already give a detailed overview of the throughput and latency capabilities of the proposed architecture for the particular application of immersive media offloading. However, we plan to extend the presented experiments and results in the short term by testing how the selected packetization schemes and encoding/decoding approach affect the overall behaviour of the architecture, both from a quantitative and quality of the immersive experience point of views.

Chapter 4

End-to-End Offloading Solution: Real-time Egocentric Human Segmentation

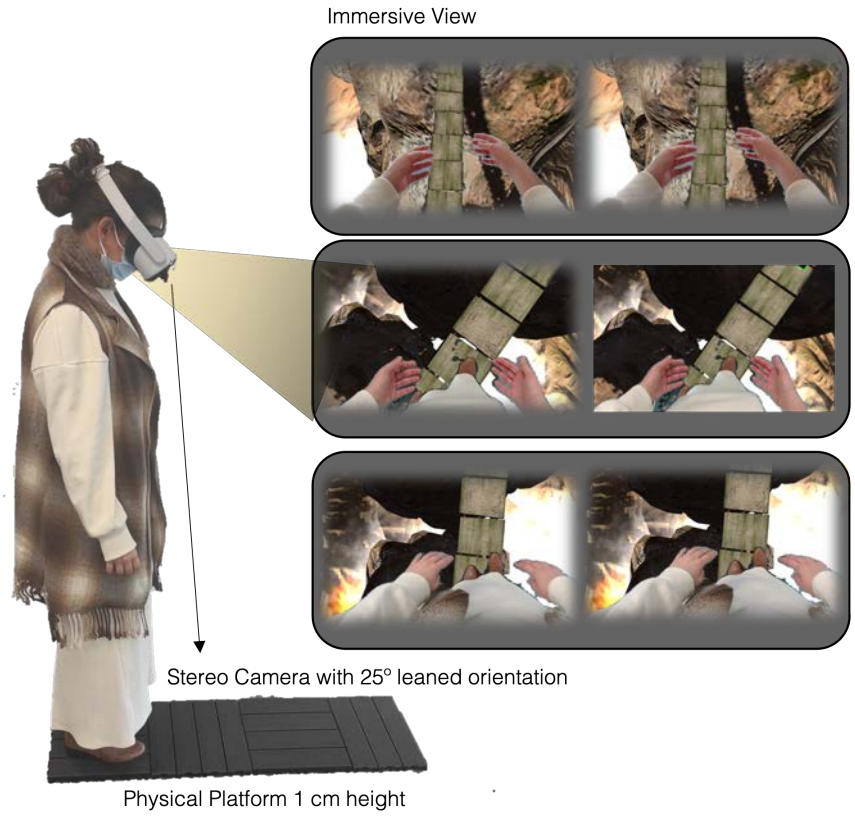


Figure 4.1: Example of video-based self-avatar in an immersive experience.

In Chapter 3, we describe our optimized offloading architecture and its performance evaluation experiments' results. The architecture has only been evaluated in terms of throughput and latency capabilities. The next natural step is to test it on a realistic offloading deployment to understand its impact on the immersivenes and other subjective metrics. To the best of our knowledge, there is no example of a full end to end XR offloading deployment running on a commercial XR HMD in the state of the art. In this Chapter, we present our end to end offloading solution which we deployed using a Meta Quest 2.

Over the last three years, we have worked on the development of a real-time egocentric body segmentation algorithm [15], [61], which, by the means of a stereo camera placed on the front of the VR HMD, allows the user to see themselves within the VR experience, as in Fig. 4.1. This solution aims to override the traditional virtual avatars used to represent the users in VR experiences. Our algorithm runs

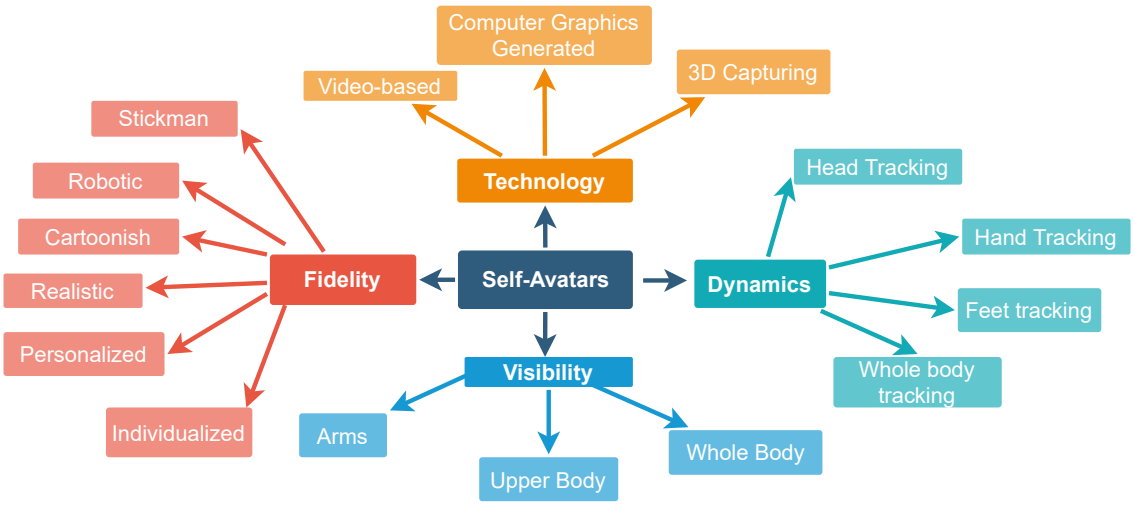


Figure 4.2: Self-avatars can be described based on their visibility, fidelity and dynamics. There are different related technologies

in real time at a frame rate above 50 Hz. However, this is achieved on a Intel Core i7-11800H and a Nvidia Geforce RTX 3070 GPU. Therefore, to include this algorithm in a commercial wireless device such as the Meta Quest 2 [8], we need to provide a reliable and optimized connection between the device and a server running the algorithm. In this chapter, we present our end to end offloading solution which allows the users to see themselves in VR using our egocentric body segmentation algorithm which runs in a nearby server.

4.1. Background of User Self-Perception in XR

The use of self-avatars, the virtual representation of the user's body from its own perspective, is starting to become ubiquitous in XR. The possibility of seeing your own body while immersed brings many benefits. First, it increases the users' Sense of Presence (SoP), as it shifts the user from being an observer to really experiencing the virtual environment [62], [63]. Besides, it enhances spatial perception and distance estimation [12], [64] while bringing a positive impact on cognitive load [65], trust and collaboration [66]. Apart from increasing the SoP, self-avatars also increase the Sense of Embodiment (SoE) [44], [67]. As defined by Kilteni *et al.* [68], Sense of Embodiment refers to the feeling of being inside, controlling and having a virtual body. Sense of Embodiment includes the sense of *ownership* as one's self attribution of a body, sense of *location* defined as one's spatial experience of being inside a body, and sense of *agency* as the sense of having global motor control over a body.

In a nutshell, an avatar is a user representation that can be described based

[8]https://store.facebook.com/en/quest/products/quest-2/

on its visibility, fidelity and dynamics (see Fig. 4.2). In what concerns to avatars visibility, the minimal configuration of an avatar just relies on modelling the users' hands [69]. On the other hand, the current trend is to model the partial body (e.g., upper body limbs [70]) or even the entire human body [71].

Once decided the level of visibility, it is necessary to decide which avatar fidelity or realism level (shape and texture for skin and clothes), should be used for modelling the human body. Existing works for hand representations range from abstract forms [44], minimal iconic representation [44], robot hands, to more realistic ones with gender-matched and skin-matched limbs [72]. Fidelity approaches for full body avatars includes minimal representations, such as the stickman appearance [67], cartoonish ones (e.g., the ones used in Mozilla Hubs), realistic avatars usually found in human model libraries such as Rocketbox library [73], or personalized ones found in VR platforms (e.g. Virbella). The latest tendency relies on models that preserve the user's body shape such as using a skinned multi-person linear body (SMPL) model [74], or 3D scanning systems such as point-cloud representation for fully personalized avatars [75]. There is still room for improvement with modelling identity-preserving avatars, specially concerning real-time performance issues. It remains an interesting challenge, since it has been proven the benefits of avatar fidelity and realism level in object size perception [71], [76], body ownership [77], presence [78], co-presence [79], among others.

One remaining challenge is how to capture all movements from the user's body so that the avatar can accurately mimic them. It is an important feature as it facilitates interaction tasks in MR [80]. Modelling head and hands movements have been widely studied. Head tracking is provided by almost all standard HMDs. Besides, there are recent commercial solutions for hand tracking, such as Leap Motion sensor [81], or camera-based hand tracking as in the Meta Quest 2 or HTC VIVE. On the other hand, less work has been done on full body tracking, including lower limbs and torso. Some works proposed using 2 VIVE trackers attached to the feet for allowing foot tracking and visibility [80], [82], [83]. Although traditional approaches for full body capture are based on IMUs sensors placed on a suit that the user wears [78], [84], [85], there are emerging solutions based on computer vision that can track the user's body movements using one single depth sensor such as Azure Kinect v2 [86]. Apart from the already mentioned challenges of full body tracking and avatar appearance resembling the user's own identity, there is still the subjective factor of whether users accept the avatar as a virtual representation of themselves or not.

An alternative approach to computer-generated avatars is the use of video-based self-avatars. This can be done through the combination of VR video pass-through and computer vision algorithms: user's real body is incorporated into the virtual scene by segmenting the egocentric vision of a stereo camera placed in front of or integrated with the HMD. The VR community has explored this idea over the last decade. For instance, Bruder *et al.* proposed skin segmentation algorithm to incorpo-

rate users' hands handling different skin colors [87]. Conversely, a floor subtraction approach was developed to incorporate users' legs and feet in the virtual environment. Assuming that the floor appearance was uniform, the body was retained by simply filtering out all pixels not belonging to the floor [87].

Pigny et Dominjon [88] were the first to propose a deep learning algorithm to segment egocentric bodies. Their architecture was based on U-NET and it was trained using a hybrid dataset composed of images from the COCO dataset belonging to persons and a 1500-image custom dataset created following the automatic labelling procedure reported in [61]. The segmentation network has two decoders: one for segmenting egocentric users, the other for exocentric users. However their segmentation execution speed could not currently totally keep up with the cameras update rate while maintaining a decent quality. They reported 16 ms of inference time for 256×256 images, and observed problems of false positives that downgrade the AV experience.

Later, in [70], the authors developed a system able to segment the user's hand and arms using a dataset composed of $43K$ images from TeGO, EDSH, and a custom EgoCam dataset, using semantic segmentation deep learning network based on DeepLabv3+. For the experiment, they used an HTC Vive HMD, a monocular RGB camera, and a Leap Motion controller to provide free-hands interaction camera. Inference time takes around 20ms for $360x360$ RGB video in a workstation provided with Nvidia Titan Xp GPU (12GB memory). They manage to run VR rendering and Convolutional Neural Network (CNN) segmentation in the same workstation.

There are other color-based approaches [48], [89], [90] which can be deployed in real time but they tend to work well just for controlled conditions. Further, they failed at dealing with different skin colors or long-sleeve clothes [89]. Alternatively, depth cameras solutions have been proposed [91]–[93]. Despite being effective for some situations, they still lack precision to provide a generic, realistic, and real time immersive experience.

4.2. Contribution

Our real-time video-based self-avatar algorithm is inspired in Thundernet's architecture [94], a deep CNN specifically designed for real-time semantic segmentation. Our improvements and optimizations from the original architecture are described in [15], [61]. Our solution, achieves a frame rate of 66 fps for an input resolution of $640x480$, thanks to its shallow design. To train it, we created a $10,000$ images dataset with high variability in terms of users, scenarios, illumination, gender, among others. This chapter investigates how to integrate our segmentation algorithm with a commercial device, in particular the Meta Quest 2, using our offloading architecture described in Chapter 3. Our main contributions can be summarized as:

- a detailed description of our End-to-End (E2E) system which manages to integrate our real time egocentric segmentation algorithm in a realistic MR experience using our own offloading architecture and custom hardware and software implementation. The E2E system is composed of three main modules: video pass-through capable VR device, real-time egocentric body segmentation algorithm, and optimized offloading architecture.
- a subjective evaluation of the video-based self-avatar technology integrated in a gamified immersive experience, conducted by 58 users. To the best of our knowledge, it is the first work that includes a user study using full body video-based self-avatars.

4.3. System Design and Implementation

The generation of real-time, accurate and effective video-based self-avatars for MR requires the following solutions to be integrated as a single MR end-to-end system:

- **Video pass-through capable VR device**: captures the real scenario using a frontal stereo camera, accurately aligns it to the user's view and head movement, and renders it, in real-time, within the immersive scene.
- **Real-time egocentric body segmentation algorithm**: identifies the pixels corresponding to the user's body from the frames captured by the frontal camera.
- **Optimized offloading architecture**: egocentric body segmentation algorithms require high-end hardware not available in current state of the art HMDs. We integrate our offloading architecture described in Chapter 3 in the proposed E2E system to provide an optimized communication link between the VR HMD and the server running the segmentation algorithm.

4.3.1. Custom Video pass-through Implementation

There are several VR devices with video pass-through capabilities which are already commercially available, such as the Varjo XR-3[9] or the HTC VIVE Pro[10]. These devices are still tethered, constraining the range of possible use cases of the proposed system. Consequently, we decided to build our own video pass-through solution [95] for the Meta Quest 2[11], as it is a well-known commercially successful stand alone VR device. For this purpose, we had to build our own hardware along with the

[9]https://varjo.com/products/xr-3/
[10]https://www.vive.com/eu/product/vive-pro/
[11]https://store.facebook.com/en/quest/products/quest-2/

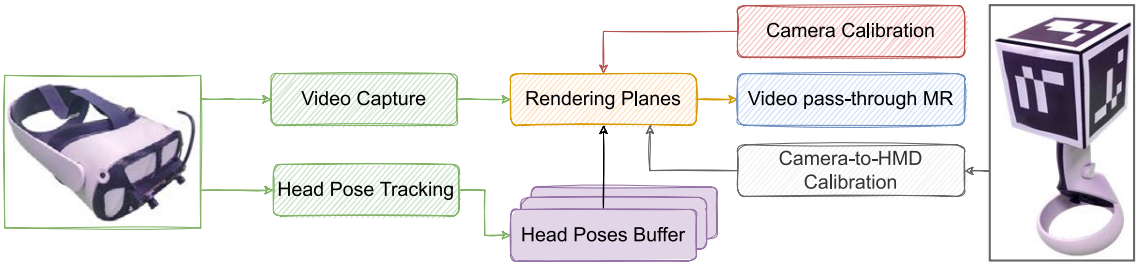


Figure 4.3: Simplified representation of our custom video pass-through implementations.

necessary software to effectively integrate the captured video into the VR scene, ensuring a perfect alignment between the captured feed and head movement.

Stereo Camera and 3D-printed Attachment

Our video pass-through system must capture high definition stereoscopic video at a rate of at least 60 Hz to comply with many MR video pass-through use cases[12]. We chose the ELP 960P HD OV9750 stereo camera[13] which provides a maximum resolution of 2560x960 at a 60Hz update rate and a field of view of 90°. Most commercially available video pass-through MR devices incorporate stereo cameras which are aligned with the user's eyes. On the other hand, we realized after some initial tests that for our particular use case tilting the stereo cameras 25° towards the ground provided a better experience and ergonomics for the user. We designed a custom 3D printed attachment to fix the stereo camera to the VR device with the described offset as in Fig. 4.3.

Video Capture

Aligned with most of AR and VR applications, our solution is built using Unity 19.3. Consequently, we must be able to access the stereo camera feed from Unity to be able to process and render it within the immersive scenario. Our system has been designed to work both in tethered and wireless modes. Due to the device's particularities, both modes differ in the way that the video stream from the stereo camera is captured:

- **Wireless Mode:** as the Quest 2 relies on Android as the base operative system, we built a native Android plugin to capture the video frames from the stereo camera. The plugin is based in the cross-platform UVC library [96] for webcam video capturing. We implemented a custom interfacing plugin to allow Unity to access the captured frames with as low latency as possible.

[12]https://developer.oculus.com/resources/oculus-device-specs/
[13]https://bit.ly/3qGHY8h

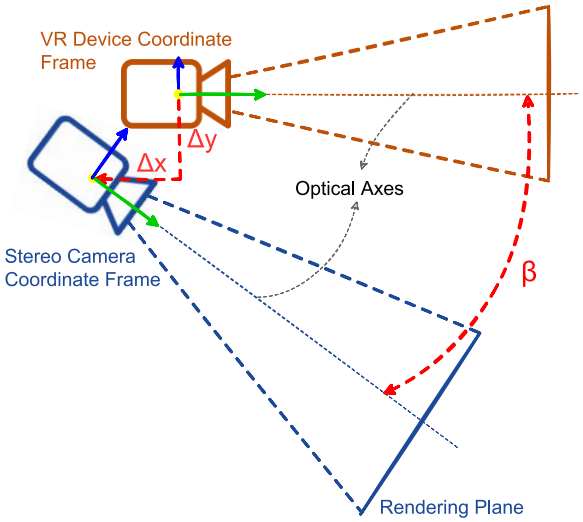


Figure 4.4: Schematized representation of the VR device's and stereo camera's rendering frame's coordinate frames. In orange, the VR device rendering camera coordinate frame. In blue, the rendering planes coordinate frame. In red, the parameters to be estimated using our custom calibration method.

- **Tethered Mode:** in this case, we capture the video frames as a simple webcam using Unity's C# Application Programming Interface (API).

Rendering Planes Placement

In any video pass-through MR application, the rendered stereo camera's feed and user's head movement must be perfectly aligned to avoid any user discomfort or VR sickness. This misalignment can come from three non-exclusive sources: lens distortion effects, inaccurate placement of the planes where the camera feed is rendered, and motion-to-photon delays. We need to apply the necessary calibration steps to reduce the effect of the aforementioned misalignment sources.

The captured video frames from the stereo camera are rendered into two virtual planes (in blue in Fig. 4.4), which we refer to as rendering planes, one for each eye. The first step is to obtain the intrinsic parameters $[f_x, f_y, c_x, c_y]$ and distortion coefficients $[k_1, k_2, k_3, p_1, p_2]$ from the stereo camera so we can accurately correct the camera's lens distortion and properly scale the rendering planes. We used a well-known camera calibration technique described by Z. Zhang [97].

We used a custom shader for Unity to correct the camera's lens distortion using the estimated parameters, applied to the rendering planes' material. The shader implements the following distortion correction equation [98] for each pixel p_{xy}:

$$x' = x + \bar{x}(k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2\bar{x}^2) + 2p_2\bar{x}\bar{y} \tag{4.1}$$

$$y' = y + \bar{y}(k_1 r^2 + k_2 r^4 + k_3 r^6) + p_2(r^2 + 2\bar{y}^2) + 2p_1\bar{x}\bar{y}, \tag{4.2}$$

where $\bar{x} = x - c_x$, $\bar{y} = y - c_y$, $r = \sqrt{\bar{x}^2 + \bar{y}^2}$. Finally, we need to properly scale the rendering planes according to the intrinsic camera parameters, so that the real objects captured by the stereo camera are perceived with the appropriate size. Using an arbitrary distance(d_r) between the rendering frames and the origin of the virtual cameras assigned to each eye (eye cameras) we can estimate the scale S for a given resolution $[R_x, R_y]$:

$$S = [s_x, s_y] = [d_r \frac{R_y}{f_y}, s_y \frac{R_x}{R_y}]. \tag{4.3}$$

Once the camera has been correctly calibrated, we can solve the inaccurate placement of the rendering planes. The goal is to estimate the pose of the stereo camera with respect to the VR device's coordinate frame, as in Fig. 4.4. Specifically, we need to estimate the accurate position and orientation of the rendering frames with respect to the virtual cameras coordinate frames. The goal is to calculate the geometrical relationship $[\Delta x, \Delta y, \beta]$, depicted in Fig. 4.4 between both coordinate frames.

While this calibration is traditionally achieved using a classic camera-IMU (inertial measurement unit) calibration method [99], commercial devices do not provide access to raw sensor data with accurate time-stamps. Consequently, we decided to design our own workaround calibration method. Our method relies on the complementary hand controllers commonly available with current state of the art VR devices. These controllers are accurately tracked by the VR device relative to its own coordinate frame. Consequently, we designed and 3D printed an attachment for the Meta Quest 2 controllers which includes an Aruco [100] cube as the one shown in Fig. 4.3.

Using a standard Aruco tracking library [100] written in C++ along with the estimated camera intrinsic parameters and distortion coefficients, we can accurately track the Aruco cube and, consequently, the correspondent controller's pose using the incorporated stereo camera. This estimation can be used to directly infer the geometrical relationship between the stereo camera's and VR device's coordinate frames. Consequently, we can directly estimate $[\Delta x, \Delta y, \beta]$, by estimating the required translation and rotation of the stereo camera's coordinate frame within the VR scene to perfectly align the VR controller pose as measured by the VR device and by the stereo camera. This relationship is constant, as the stereo camera is rigidly attached to the VR device.

Delay Alignment Buffers

Even though the stereo camera lens distortion has been properly corrected and the video feed rendered on well-aligned and scaled rendering planes within the virtual scene, we need to introduce a method to remove the misalignment produced by the

motion to photon latency. This latency produces an unnatural decoupling between the stereo feed, the head movement, and, consequently, the virtual scene. This artifact not only degrades the experience but can produce VR sickness.

In a delay free system, we could just place the stereo camera coordinate frame in a fixed relative position and orientation with respect to the VR device coordinate frame. However, in a realistic scenario, this setup would produce the effect that the visual feed is delayed with respect to the user's head movement. To overcome this issue we add a delay alignment buffer, which stores the position and orientation corresponding to the stereo camera's coordinate frame. The size of the buffer corresponds to an arbitrary time t_c which depends on the empirically estimated motion to photon delay. We assume t_c to be constant for the same stereo camera and VR device models. Consequently, by placing and rotating the stereo camera feed always to the first pose stored in the buffer, we achieve an accurate head movement to stereo feed alignment.

4.3.2. Egocentric Segmentation

This is a crucial step in our setup: we need an accurate real-time egocentric body segmentation algorithm which allows to identify which pixels, from the stereo camera feed, correspond to the user's body. Our algorithm is based on deep learning techniques. Particularly, it is based on the use of semantic segmentation networks [101]. In this case, the segmentation is performed from first point of view (egocentric vision).

The designed algorithm must meet two requirements: *i*) real-time performance achieving an update rate above 60Hz, and *ii*) high quality segmentation in uncontrolled conditions. Concerning the first requirement, we designed our architecture inspired in Thundernet [94], a very shallow semantic segmentation algorithm composed of: *i*) an encoding subnetwork; *ii*) a pyramid pooling module (PPM) that enables to extract features at different scales; and *iii*) a decoding subnetwork. Several modifications were applied to the baseline architecture: *i*) larger pooling factors of 6, 12, 18, 24 were used in the PPM module to better extract features of larger input images, *ii*) three additional long skip connections between encoding and decoding subnetworks for refining object boundaries were added, and *iii*) weighted cross entropy loss function was used to handle class imbalance between human body (foreground) and background. For more detailed information please refer to our work [15].

To achieve high quality segmentation, a data-centric approach was followed, putting a strong emphasis on the variability of the training data. We created a dataset of almost 10000 images composed of three datasets: *i*) Ego Human dataset, a semi synthetic dataset composed of egocentric body parts (arms, lower limbs, torsos) merged with realistic backgrounds that facilitates the process of groundtruth gener-

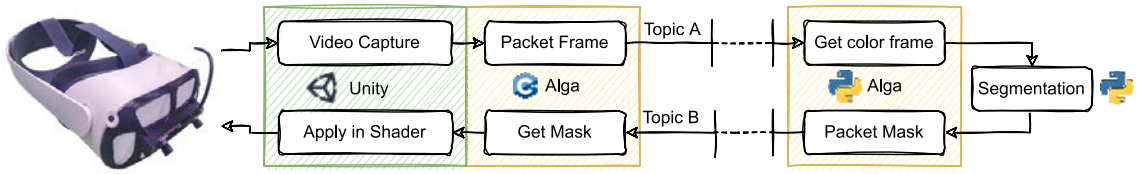


Figure 4.5: Final E2E simplified architecture, including the languages in which each agent was built.

ation; *ii*) a subset of THU-READ dataset, originally created for action recognition, whose segmentation groundtruth was created manually using Amazon Mechanical Turk, and *iii*) EgoOffices: an egocentric dataset that was manually captured using a portable custom. The dataset was captured by more than 25 users and multiple realistic scenarios, such as different apartments or office rooms. As the hardware setup involves the use of a stereo camera for providing stereo vision, to the user, we re-trained Thundernet architecture with images composed of two copies of the same images resulting in a 640×480 image, to replicate stereo vision. For more details, please refer to [15].

4.3.3. End to End System

The final E2E setup is depicted in Fig. 4.5. The principal data flow of the E2E architecture is the following:

1. **Stereo camera frame capture:** As we have already described, the VR client runs Unity and is in charge of obtaining individual frames from the stereo camera video feed. The individual frames are moved to the underlying C++ plugin running our offloading architecture.

2. **Transmit stereo color frame to server:** The offloading architecture, via Alga, is in charge of packing each frame and transmit them via TCP through an arbitrary topic. The segmentation server, runs another instance of Alga and receives the frames through the same topic.

3. **Egocentric body segmentation:** The server then infers the segmentation mask for each received frames.

4. **Transmit result back to VR client:** The mask is transmitted back to the VR client via Alga, through another arbitrary topic.

5. **Shader application on rendering frames:** Finally, the client applies the received masks to the custom shader attached to the rendering frames. The shader is in charge of rendering exclusively the stereo camera pixels corresponding to the user's body according to the received mask.

4.3.4. Hardware Setup

The egocentric segmentation server was running on a PC with an Intel Xeon ES-2620 V4 @ 2.1Ghz with 32 GB of RAM and powered with 2 GPU GTX-1080 Ti with 12GB RAM each. On the VR client side, we considered two modalities:

- Standalone Mode: The immersive scene is running directly on the Meta Quest 2 HMD, to which the stereo camera is directly connected.
- Tethered Mode: The immersive scene runs on a workstation to which a Meta Quest 2 is connected via an USB-C cable using the "Link" mode. The stereo camera is connected to the workstation. In this case the client workstation includes an Intel Core i7-11800H and a Nvidia Geforce RTX 3070 GPU.

We used Unity version 2019.3 on the client side, where the stereo frames are captured at a resolution of 1280x480. For the validation experiments we used a Netgear R6400 router, providing symmetric wireless 200 Mbps when a single user is connected [30].

4.4. Performance Evaluation

The proposed offloading architecture has been thoroughly benchmarked in terms of throughput and latency in multiple relevant scenarios and wireless technologies[30]. We decided to extend the benchmark by testing the architecture on the final E2E system. To obtain the following results we used the exact hardware setup described in Section 4.3.4.

4.4.1. Offloading Architecture and Video Pass Through

We first aim to evaluate the performance of our offloading architecture alone in the following setups:

- Wireless Scenario: Meta Quest 2 in Standalone mode using WiFi. We want to understand how the implemented system behaves on a device with limited computing resources.
- Tethered Scenario: Meta Quest 2 connected to a workstation. In this case the system has considerably higher computing resources. The workstation is connected to the server via an ethernet cable, with a 1 Gbps interface.

In this first round of experiments, we removed the segmentation algorithm from the server, directly transmitting back to the client a single channel 8-bit mask of the incoming frame. This mask is obtained using a chroma-key algorithm which

Table 4.1: Performance evaluation results for the tethered and wireless scenarios along with the server processing times.

	Tethered	Wireless	Server
Mean	7.37 ms	10.39 ms	16.7 ms
Variance	14.37 ms	19.38 ms	0.006 ms
95-th Perc.	14.0 ms	18.0 ms	31.2 ms

adds a negligible overhead on the server side. For each scenario, we repeated a set of 4 experiments each transmitting 1000 frames. No other users are connected to the router. We store the times that each frame takes from the moment it is loaded in memory on the client side to when the final mask is received from the server on the client side. The initial results are shown in the first two columns of Table. 4.1. Our E2E setup provides mean round trip times of 7.37 and 10.30 milliseconds on tethered and wireless scenarios respectively, with confidence intervals below 20 ms in both cases.

4.4.2. Segmentation Server

In this case, the goal is to measure the total time consumed by the server in performing all the necessary steps for a frame: reception, necessary transformations such as scaling, and inference. Similarly to the previous set of experiments, we carried out 4 experiments each processing 1000 frames. We obtained a mean time of 16.7 ms, as shown in Table 4.1, which guarantees an update rate of 60 Hz, complying with MR applications real-time requirements.

4.4.3. End to End System

Finally, by aggregating the results obtained in the previous two experiments we obtained E2E mean RTTs of 29.67 ms and 32.29 ms. This latency is low enough to allow the use of delay buffers to ensure a proper alignment of the color frame and masks, as described above.

Delay buffers

As the obtained RTTs values are higher than the device's update period of 16 ms, the hand pixels appear to be misaligned with respect to the segmentation mask. The misalignment comes from the fact that the segmentation mask arrives to the VR client an average of 32.29 ms later than its corresponding stereo pixels are loaded in memory and rendered. The feeling is as if the mask follows the actual hand. This artifact, shown in Fig. 4.6, affects the overall experience and must me removed.

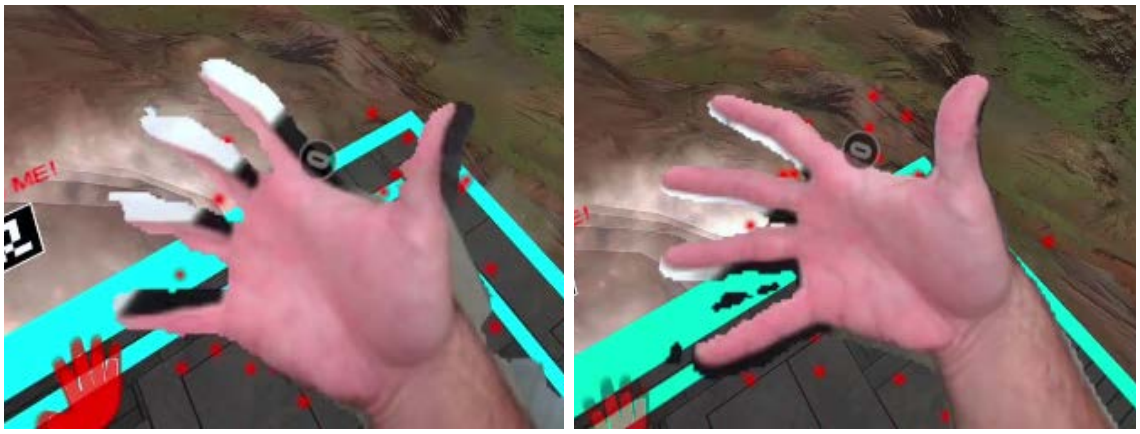


Figure 4.6: Left: Mask-color pixels alignment observed when no delay buffer is used. Right: alignment achieved using the delay buffer.

To overcome this issue, we decided to add a frame buffer of an arbitrary size according to the update rate of the camera and the measured mean RTT (32.29 ms). By doing this we overcome the artifacts generated by the system's added delays, ensuring the mask and its corresponding stereo pixels to be rendered simultaneously. Notice that the size in time of this delay buffer must be added to the alignment buffer described in Section 4.3.1.

4.5. Conclusions

In this chapter, we have thoroughly described the state of the art of user representation for MR applications, including a wide range of solutions such as realistic full-body avatars, virtual hands only, or video-based full-body representation. We have highlighted their advantages and disadvantages, focusing specially on the users perception of presence and embodiment. Besides, we reviewed specific state of the art implementations of video-based egocentric segmentation solutions for MR applications, highlighting the need of a more accurate and fast segmentation algorithm. To the best of our knowledge, we could not find any E2E implementation which allows the usage of deep learning based segmentation solutions on commercial wireless VR devices.

Furthermore, we have presented our E2E system which provides full body video-based self-avatar user representation for MR applications. The E2E can be divided in three different modules: video pass-through solution for the Meta Quest 2, our real-time egocentric body segmentation algorithm, and our optimized offloading architecture. All the different components and implementation details have been described in detail.

Our implementation allows to run the proposed segmentation algorithm in real-time on a commercial VR device, the Meta Quest 2. We achieved round trip times

below 11 ms over WiFi if no algorithm is running on the server. Consequently, the system allows to use other computationally demanding solutions on standalone VR devices such as object tracking or camera-based full body tracking, extending the current boundaries of MR solutions and applications.

The full E2E results showed the implementation's capability to fulfil the real-time requirements, providing update rates above 60 Hz using WiFi and running the egocentric human segmentation algorithm. Finally we described our delay-correction buffers to overcome the round trip latencies above the update rate of 60 Hz ensuring an accurate alignment between the color and received mask frames.

Chapter 5

FikoRE: 5G and Beyond RAN Emulator for Application Level Experimentation and Prototyping

A fast development of XR technologies and other 5G use cases demands accessible and simple-to-use 5G testbeds. The network requirements that these applications impose, described in Chapter 2, demand a thorough design of the resource allocation techniques, including network slicing and user prioritization. Besides, the network needs to dynamically adapt to the particular applications, based on QoE metrics, by communicating required QoS indicators using 5G QoS Identifiers (5QI). However, some specific 5G deployments and configurations are still not available for many application developers and researchers, which can potentially hinder the fast development not only of distributed XR implementations, but other ground-breaking technologies.

One feasible solution to accelerate the development of advanced technologies, such as distributed XR, is to use RAN simulators, allowing application developers to test the performance of their solutions under different network setups and configurations. However, and to the best of our knowledge, there is no 5G emulator available that is capable of handling actual IP traffic above 1 Gbps from multiple IP sources, while simulating hundreds of virtual UEs, and modelling advanced 5G deployments' behaviour with sufficient accuracy, as described in Section 5.1. These requirements are crucial for an emulated 5G testbed to be broadly used by application developers of demanding use cases, such as XR offloading or the Metaverse.

In this chapter we present FikoRE (Framework for Immersive communication networK Optimization RAN Emulator), an open-source [26] RAN emulator targeting to lower the entry barrier for application layer researchers to test their real-time solutions in a realistic RAN deployment. FikoRE, implemented in C++, is capable of handling high throughput (above 1 Gbps) real IP traffic allowing researchers to directly test advanced and demanding use cases such as XR offloading. Besides, due to the thoughtful modelling assumptions that have been made, FikoRE achieves a high scalability level supporting hundreds of UEs in real-time, regardless of their target throughputs, while realistically emulating an advanced 5G setup as we prove in a set of validation experiments. Furthermore, FikoRE has already been tested on a realistic Free-Viewport Video streaming application, as described in Section 5.4. These characteristics and validation experiments allows the potential consideration of FikoRE as a reference testbed for wireless XR applications and other demanding novel use cases.

5.1. State of the Art of 5G RAN Emulation

There are many RAN simulators in the state-of-the-art widely used in research and industry. Vienna Simulator [22] and NS-3[102] are two of the most well-known examples. Another interesting example is Matlab 5G Toolbox [103], a very useful tool for telecommunications researchers as the flexibility provided by Matlab allows them to rapidly test their own models or algorithms. Finally, 5G-LENA [104] is a popular 5G simulator with end-to-end simulation capabilities. It is designed as a plugged module for NS-3.

More interesting for our purpose are the system level RAN emulators, capable of handling actual IP traffic while simulating, in real-time, a specific RAN technology. There are multiple RAN emulators in the state-of-the-art, some of them included as an available functionality in the aforementioned simulators, as in NS-3[22]. However, the scalability in terms of throughput and simultaneous UEs is too limited [105] to be used as an emulator for novel 5G use cases. A relevant effort is the OpenAirInterface (OAI) 5G-RAN [23] simulation module, which has real-time emulation capabilities. Similarly, its scalability limitation, requiring more than 20 GB of memory to emulate 100 UEs in real time, discards it as a potential 5G testbed for advanced XR use cases.

Finally, we can mention Simu5G [24], which has proven to achieve real-time performance while handling actual IP traffic and emulating a full end-to-end, multi-user and multi-gNB 5G setup. However, its real-time performance cannot support high throughput. It is limited to less than 2 Mbps of actual IP traffic, hindering Simu5G to become a relevant testbed for wireless XR testing and prototyping.

There are relevant 5G emulators commercially available, such as the ones from Polaris [106] and Keysight [107], but they are specially focused on testing 5G equipment rather than potential applications.

From the brief analysis of 5G RAN emulation and simulation tools available in the state of the art, we can conclude that FikoRE can potentially become the reference platform for advanced 5G use cases prototyping and development.

5.2. FikoRE Structure

We have organized FikoRE in 3 main modules:

- **Traffic Capture/Generator (TC/TG)**: it (a) generates virtual traffic, according to the user-selected TG model, it (b) parses actual IP traces to generate realistic virtual traffic or it (c) captures actual IP traffic coming from an arbitrary set of ports.
- **MAC Layer**: it handles the resource allocation step, implemented as an

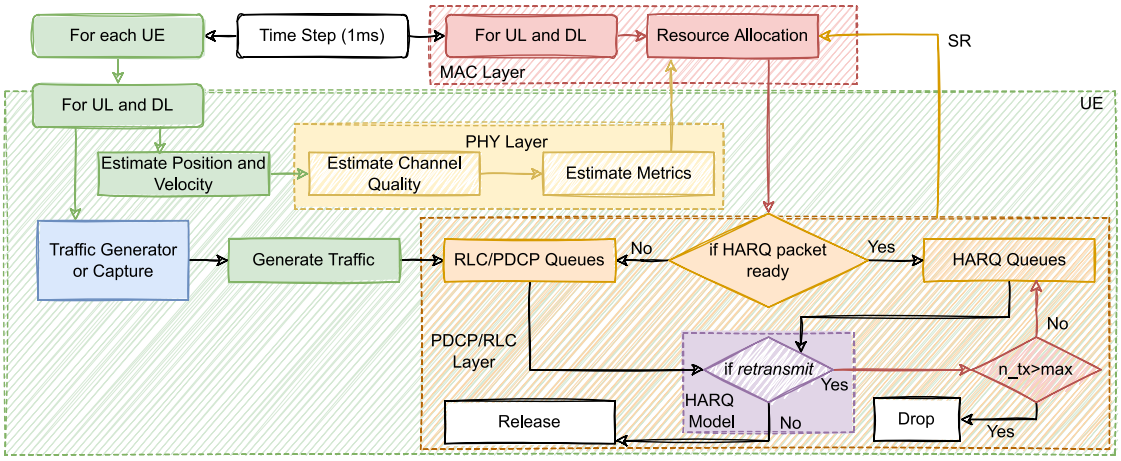


Figure 5.1: Simplified information flow between FikoRE's main modules.

abstraction of the details described in 3GPP 38.214 [108], 3GPP 38.211 [109] and 3GPP 38.306 [51] specifications.

- **User Equipment (UE)**: it models a single UE, including its position estimation, according to a user-selected mobility model, channel quality estimation, and packet handling. The main modules included in each emulated UE are:
  - **Radio Link Control and Packet Data Convergence Protocol Layer**: it models the behavior of the Radio Link Control (RLC) and Packet Data Convergence Protocol (PDCP) layers with a high abstraction level.
  - **PHY Layer**: it models the main channel quality measurements, such as the Signal to Noise and Interference Ratio (SINR), Reference Signal Received Power (RSRP), HARQ retransmissions probabilities, etc. The models implemented in this layer are described in 3GPP 38.901 [110].

These modules are instantiated for each transmission direction: UL and DL. The overall data flow, depicted in Fig. 5.1, can be summarized as:

1. Each emulated UE's IP traffic is generated or captured by its own TG module. The packets are sent to the RLC/PDCP layer where they are queued in the IP buffer.
2. The PDCP/RLC layer indicates to the MAC layer how many bits are available for transmission for each UE via Scheduling Request (SR).
3. With this information along with each UE's emulated Channel State Information (CSI), estimated by the PHY layer module, the MAC layer allocates frequency and time resources to each UE.

4. The PDCP/RLC layer, according to the allocated resources by the MAC layer, segments the IP packets into smaller blocks.
5. Using the emulated CSIs, the PHY layer module determines the transmission latency and HARQ retransmission probabilities of each block.
6. If the block must be retransmitted, it is moved to the correspondent HARQ queue to be allocated again.
7. Finally, when the block is successfully transmitted it is moved back to the RLC/PDCP layer module, where the integrity and ordering of the full IP packets are checked.
8. Successful IP packets are then released. Corrupted or lost IP packets are dropped.

The described steps run once every 1 millisecond. This deadline is strict and must be met to allow a proper synchronization between actual IP traffic and the emulator's behaviour.

5.2.1. Traffic Capture and Generation

The Traffic Generator (TG) module is used for simulated UEs which are not linked with any actual IP traffic. It periodically creates virtual IP packets for both uplink and downlink transmission directions. There is a TG instance in each UE, handling both UL and DL traffic. FikoRE implements two TG options:

- **Simple Traffic Model**: generates, in each time step, as many packets of an arbitrary size as necessary to fulfil a given arbitrary mean throughput goal. The user can freely select the target UL and DL throughputs, and the size of the simulated IP packets.
- **IP Traffic Traces**: parses a specially formatted text file which stores a simplified version of IP traces captured by specific software such as Wireshark. Each line corresponds to an IP packet's timestamp and payload in bytes. This functionality allows to add virtual UEs to the emulation session which are demanding realistic traffic (such as XR offloading) to the network.

For UEs handling actual IP traffic, the TG is replaced by a Traffic Capture (TC) module, which incorporates Netfilter Queues (NQ) [111] library. NQ is a Linux user space API which provides access to packets that have been queued by the firewall. To filter packets coming from or to specific ports or addresses, the adequate firewall rules must be set by the user, as described in [26]. Each emulated UE linked to actual IP traffic have two queues assigned: for UL and DL directions. These queues

IDs are provided to the TC module as a configuration parameter. FikoRE interfaces with NQ assigning a callback to each of the UE's queues, which receives each IP packet's ID and size. This information, along with the callback timestamp is used to create virtual IP packets identical to the ones generated by the TG module. The received packet's ID can be used via NQ to drop or release the associated actual IP packet.

5.2.2. MAC Layer

Tuning, improving or even re-designing the resource allocation step handled by the MAC layer can enhance the network performance enabling novel use cases or applications. Our goal is to provide a representative yet simple to configure and modify MAC layer module.

This module implements a sufficiently detailed abstraction of the resource allocation procedures described in 3GPP 38.214 [108] and 3GPP 38.211 [109] specifications. More specifically, a Resource Allocation Grid is built by the emulator for each transmission direction (UL and DL), according to the user-selected parameters, such as the bandwidth and numerology. The smallest isolated resource unit handled by our implementation is a Resource Block (RB), containing 12 subcarriers in frequency and 14 OFDM symbols, in Resource Block Groups (RBGs), as described in the specifications.

The goal of the resource allocation step is to optimally assign these RBs/RBGs to the available UEs according to a selectable metric, estimated in each timestep for each UE and RB/RBG pair. The current version of the emulator implements well-known metrics, as described in [112], such as First-In-First-Out (FIFO), Proportional Fair (PF), Blind Equal Throughput (BET) and Maximum Throughput (MT).

We have implemented a simple user prioritization scheme in which each UE can be configured with a different level of prioritization (5QIs). This feature is key in scenarios in which extremely demanding applications, such as XR offloading [28][30], share network resources with other UEs.

The MAC layer module also determines which modulation and coding rate is used for each UE and RB/RBG according to the Channel Quality Indicator (CQI) index reported by the PHY layer module which is used to map a corresponding MCS index. The MCS index, which represents the current channel quality of a specific UE, is mapped to the modulation and coding rate according to the tables in 3GPP 38.214 [108]. The MCS index can be estimated for the entire bandwidth (wideband mode) or for each sub-band, RBG/RB (sub-band mode), both supported by FikoRE. The estimated modulation order and coding rates are then used to determine how many bits can be transmitted in each RB/RBG to the assigned UE. We used the

following formula slightly modified from 3GPP 38.306 [51]:

$$\text{bits} = \nu \cdot Q_m \cdot f \cdot R \cdot N^{sc} \cdot B \cdot (1 - OH) \tag{5.1}$$

where ν is the number of used Multiple Input Multiple Output (MIMO) layers, Q_m is the modulation order, f is the scaling factor (configuration parameter), N^{sc} is the number of subcarriers in an RB/RBG, and OH is the estimated overhead whose possible values are given in the 3GPP 38.306 [51] specification. B is a parameter we used to handle TDD. It represents how many OFDM symbols, estimated by the TDD module, are granted for DL or UL transmission in the evaluated RB/RBG. If Frequency Division Duplex (FDD) is selected, then B is always the maximum number of OFDM symbols within the RB/RBG. Then, a Transport Block Size (TBS) is estimated for each UE-RB/RBG pair, according to 3GPP 38.306 [51]. This number of bits are requested to the RLC/PDCP layer, which segments the IP packets in blocks accordingly.

Contrarily to other emulators such as Simu5G [24] the resource allocation steps are done every 1 millisecond instead of every Transmission Time Interval (TTI) or slot. We assume that the estimated channel quality is invariant along the 1 millisecond sub-frame and perform the resource allocation step along all the slots within the sub-frame accordingly. While our simplification implies a decreased modelling accuracy in the cases in which the UEs move sufficiently fast (coherence time is smaller than 1 millisecond), it considerably reduces the processing overhead increasing the overall scalability.

5.2.3. User Equipment

The UE module models the individual UEs connected to the emulated gNB. In each timestep, the UE module is designed to:

1. **Generate Traffic**: using the TG model or from the incoming real IP traffic.
2. **Update its Position**: according to the selected mobility model: random walk, waypoint and Manhattan.
3. **Estimate its CSIs**: using the PHY Layer module which implements the models from the 3GPP 38.901 [110].
4. **Handle Packets**: the packets are handled by the UE's RLC/PDCP module and can be queued, depending on several emulated metrics, in 3 different buffers: IP, HARQ and Packet Release buffers.

The main procedures of the UE module are handled by the following main modules:

User Equipment: RLC/PCDP Layer

The RLC/PDCP layer represents a simple abstraction of all the higher layer levels within the full stack of 5G and beyond networks. There is an instance of the RLC/PDCP layer in each UE instance. This module queues all the IP packets (real and simulated) from each UE. Via a SR, the RLC/PDCP layer communicates to the MAC layer how many bits are ready for transmission for each UE. When a slot is allocated to an UE by the MAC layer, the RLC/PDCP layer splits the IP packets into smaller blocks (if necessary) of the estimated TBS. These blocks are retransmitted according to a simple HARQ retransmission probability expression as:

$$p(\text{NACK}) = (1 - (1 - f_{\text{BLER}}(\gamma))^{n^{\text{tx}}}) \cdot r^{n_{\text{tx}}} \tag{5.2}$$

where $f_{\text{BLER}}(\gamma)$ estimates the Block Error Rate (BLER) given the SINR γ in the time step when the packet was originally transmitted, n^{tx} is the number of times the evaluated packet was already retransmitted, and r is the HARQ error reduction factor left as a configuration parameter. If there is retransmission, the packet is moved to the HARQ buffer. Otherwise, it is moved to the Packet Release buffer. These time deadline for these packets to be released from both buffers are estimated according to the PHY layer air transmission delays, HARQ ACK/NACK period, and other modelled or selectable processing delays. There is one instance of the IP, HARQ and Packet Release buffers for each transmission direction (UL/DL) within each UE.

The Packet Release buffer also implements packet re-ordering capabilities as an actual RLC layer. Besides, it also checks the integrity of full IP packets: if one block from an IP packet is dropped (maximum number of retransmissions is reached) then the entire IP packet is dropped.

User Equipment: PHY Layer

The PHY layer models the most relevant channel quality metrics estimation: SINR, RSRP, MCS, among others. These metrics are used by the MAC layer for performing the resource allocation step. The key metric, from which all the others are derived is the SINR. We calculate the SINR from the Received Signal Power (RSP), and the measured noise and interference. To estimate the RSP, we model the macroscopic fading which includes the Line of Sight (LOS) probability, Pathloss and Shadowing attenuation using the models described in 3GPP 38.901 [110] specification, given for 6 different outdoor and indoor scenarios. Besides, we also model the Rayleigh Fading as the module of uncorrelated zero mean gaussian random variables. Both the microscopic (Rayleigh) and macroscopic fading are modelled according to the UE's velocity and doppler law, coherence bandwidth and correlation distance, the latter estimated according to 3GPP 38.901 [110]. Other constant values are required

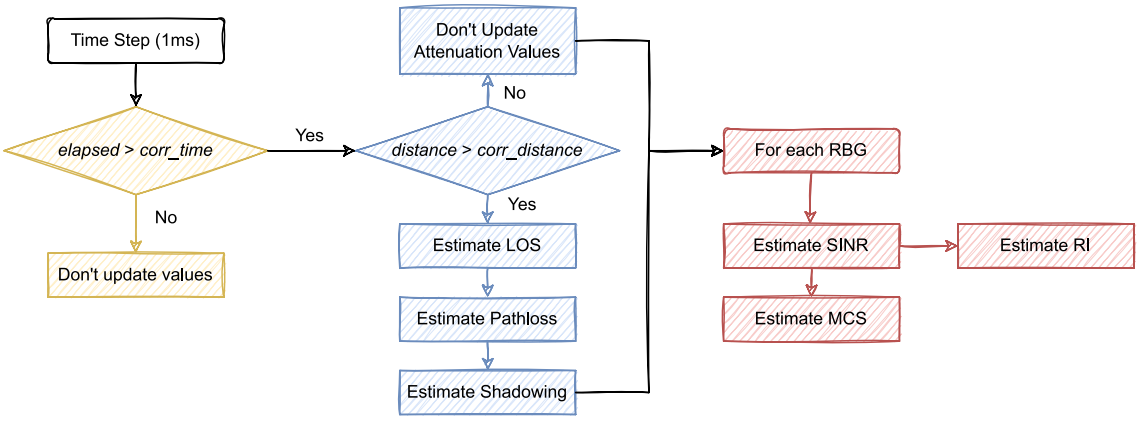


Figure 5.2: PHY Layer module's simplified data flow.

but are left as configuration parameters for the user: transmission and reception antenna gains, and the transmission power.

To allow the concurrent emulation of multiple UEs, we need to reduce FikoRE's computation overhead as much as possible. Therefore, the noise and interference are modelled at the beginning of the emulation session and left constant. While the noise is given as a configuration parameter, the interference is modelled using the same RSP models as described before but applied to other background gNBs and UEs sufficiently close to the simulated gNB and UEs. These background gNBs and UEs are only considered for the precomputation of the interference. The number and distance of the interfering gNBs and UEs, and their base frequencies and other basic configuration are left as configuration parameters.

Once the noise, interference and RSP are modelled, the SINR can be directly calculated. The SINR is reported as a linear average of each RB SINR values. The bandwidth size of each SINR report depends on the arbitrary CSI report configuration parameters. The SINR is used to estimate the MCS index for each UE and RB/RBG. To obtain the SINR-MCS relations, we performed link level simulations using Matlab, obtaining a set of SINR-BLER curves for each MCS. These curves are estimated for different configurations according to the maximum number of MIMO layers and number of RB per RBG.

Finally, we have the Rank Indicator Model which determines if the signals received in each antenna of the UE or gNB are sufficiently uncorrelated, so that several MIMO layers can be used. As we are not estimating any correlation matrices, we proposed a simple model where the total transmission power is constant for a given UE or gNB and is uniformly divided among each antenna used for MIMO transmission. Consequently, we can estimate for each number of MIMO layers the threshold SINR values for each extra added MIMO layer from which it is more efficient to use an extra layer in terms of data rate. We built a simple look-up table with this information. The overall data flow of the PHY layer module is depicted in Fig. 5.2.

5.2.4. Multithreading and Real-time Discrete Emulation

FikoRE has been designed as a multi-threaded emulator to expand its scalability as much as possible. As a real-time discrete emulator handling actual IP traffic, all the threads must be well synchronized. FikoRE has two main processes which have to run one after the other: MAC Layer resource allocation and UE handling. The first process consists of two threads, one for each transmission direction (UL and DL). In the second case, each UE is running in a separated thread. We implemented a custom threading agent which launches the MAC layer threads, waits for them to join, and then launches the UE's threads. When all the UE's threads have joined, the main timing module idles until the 1 millisecond deadline is met to run the next time step, launching the MAC layer threads. If, for the selected configuration or hardware limitations, each time step cannot be processed in less than a millisecond, the emulator user is notified, and the emulation aborted.

5.3. Validation Experiments

We performed a set of experiments to validate and show FikoRE's models and performance. To ensure repeatability, the different stochastic distributions' seed is fixed along the experiments. We also assumed that all the UEs were in LOS with the gNB.

5.3.1. MCS-Throughput Validation

The first experiment targets to validate all the layers but the PHY layer: we want to study if the mapping between the reported MCS and measured application throughput is comparable to the one measured in a real deployment. For this, we took several measurements on an actual mmW 5G outdoors deployment. We used iperf3 to continuously transmit IP packets to an iperf3 server running on a nearby MEC server. We used an Askey RTL6305 modem to wirelessly connect our laptop to the mmW gNB. We continuously logged the MAC layer level DL throughput and the associated MCS index. These pairs of MAC layer throughput and indexes were compared to the ones obtained by FikoRE. Both the emulator and actual mmW setup were running on the 26.5 GHz frequency band, for a total of 800 MHz of bandwidth separated in 8 carriers. All the carriers were configured with 4DL:1UL TDD configuration. In both cases, only one UE is connected to the gNB. We emulate a scenario in which the UE slowly moves away from the gNB to obtain all the possible MCS values. The results comparison is shown in Fig. 5.3. We can observe how in most of the cases the mean throughput difference between the actual and emulated cases is low, showing a final average error smaller than a 5.3%. We can observe how lower MCS indexes show proportionally larger throughput differences than the mid and

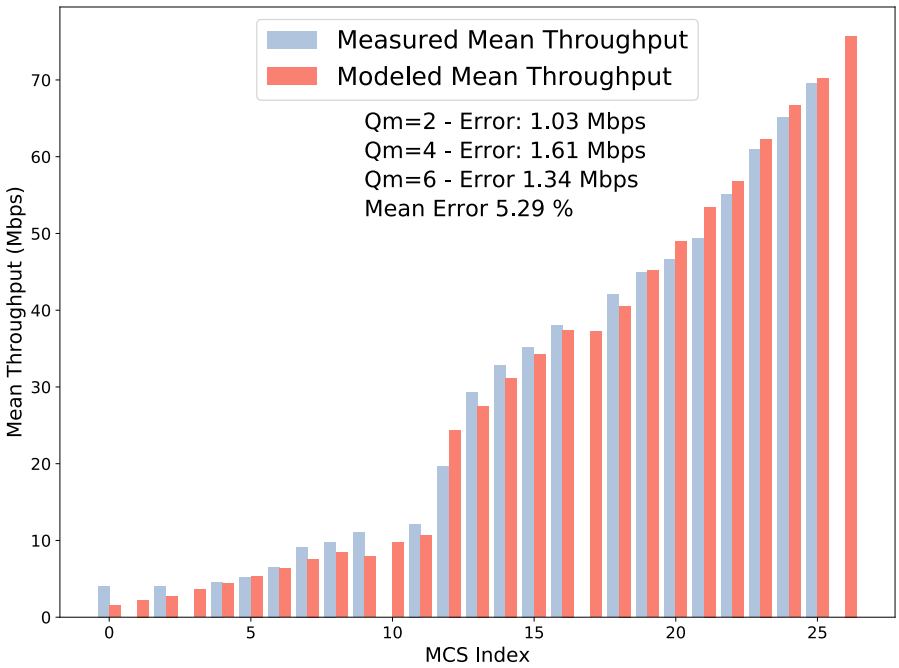


Figure 5.3: MCS index versus measured mean throughput per RB for actual measurements on a mmW setup (blue) and simulated values(red).

high indexes.

5.3.2. Allocation Metrics

The resource allocation metrics estimation of the MAC layer is also a key step as the selected or implemented metric directly affects the overall performance for particular applications. In [113] the authors test, on the Vienna Simulator [22], how the application layer throughput is affected by the selected metric for a fixed number of connected UE. To validate our metrics and resource allocation step implementation we decided to repeat the experiment using our emulator. We tested two different setups: 3.5 GHz frequency band with 10 MHz of DL bandwidth as in [28] (Configuration A) and 26.5 GHz frequency band with 400 MHz of DL bandwidth (Configuration B). We tested our implementation of the FIFO, PF, BET and MT metrics. To be able to compare our results with the ones obtained in [113], we configured FikoRE to simulate 20 full-buffer UEs. The results obtained are depicted in Fig. 5.4. We can directly compare configuration A results to the ones obtained in [113], showing to be almost identical. Being the Vienna Simulator, a widely recognized RAN simulator, this comparison itself can serve as a validation of our metric estimation and resource allocation implementation.

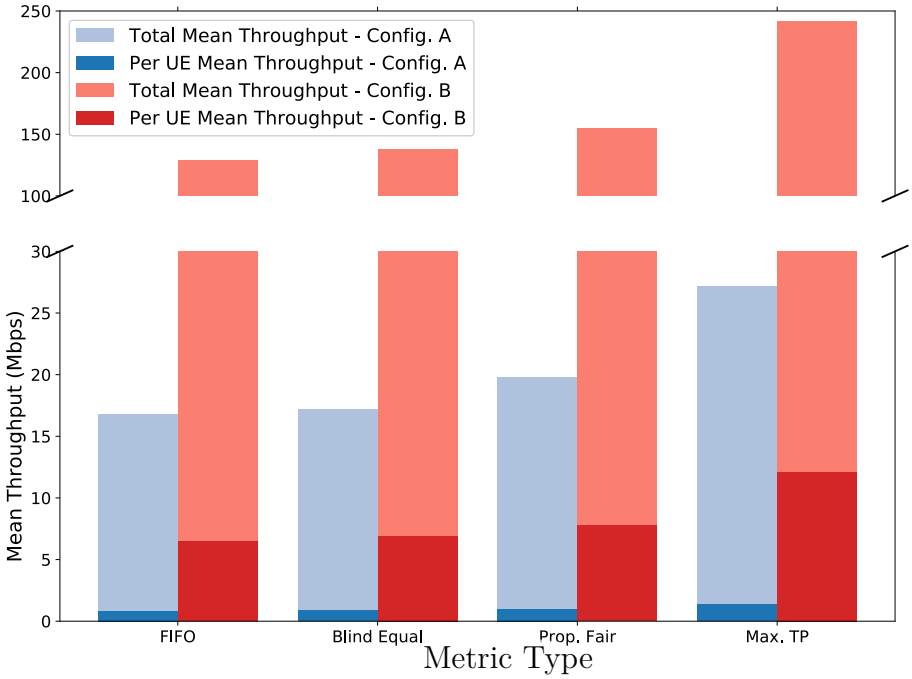


Figure 5.4: Throughput values obtained using different metric types and 20 simulated users. In red: mmW frequency band. In blue: 3.5 GHz frequency band.

5.3.3. User Prioritization

Our next target is to test how FikoRE handles actual IP traffic when multiple concurrent simulated UEs share resources from the same emulated network. Besides, we also aim to test how the selected user prioritization level affects the traffic of the prioritized UE (actual IP traffic in this case). In this experiment, we run the emulator configured on the 3.5 GHz frequency band and 40 MHz of DL bandwidth, using the PF metric. For injecting actual IP traffic, we used iperf3 with a target throughput higher than the maximum that can be provided by the available bandwidth to make sure the associated UE is receiving the maximum possible throughput given the emulated network setup. We repeat the experiments with a different prioritization level each time: no prioritization, medium, high and maximum prioritization. The other simulated UEs are in all cases set to low priority.

We repeated the same experiments for both TCP and UDP traffic. The number of simulated UEs ranged from 10 to 250. The simulated UEs are demanding 5 Mbps from the network, which corresponds to a user watching a high-definition online video. The results of the measured mean DL throughput for the actual IP traffic UE are shown in Fig. 5.5. As expected, the measured transmitted DL throughput rapidly decays as the number of simulated UEs grows in the medium and no prioritization cases. We can observe the importance of user prioritization when

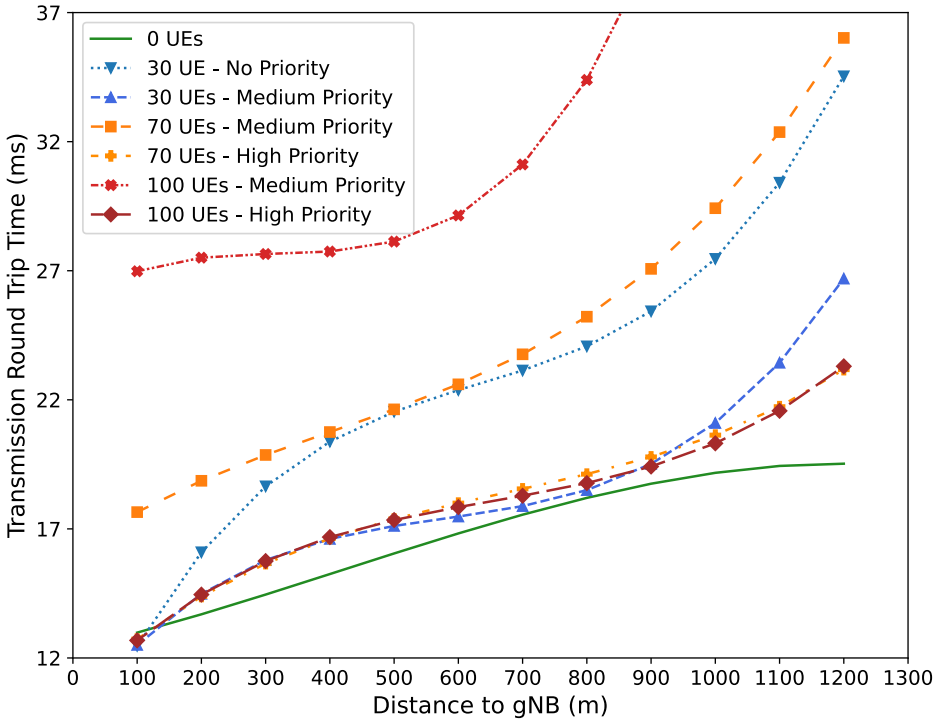


Figure 5.5: Emulated throughput for a user handling real IP data and sharing resources with other simulated UEs. Different UE prioritization levels are used for the real IP traffic UEs.

particular UEs must sustain a minimum network quality, such as in XR offloading [30]. When the UE is sufficiently prioritized, the measured throughput decreases at a much lower rate, sustaining high throughputs even when many UEs are demanding network resources. With this experiment we showcase not only FikoRE's capacity to simulate multiple UEs while handling actual IP traffic in real time, but also highlight the importance of user prioritization allocation techniques to ensure sufficient quality of service for advanced use cases.

5.3.4. XR Traffic Use Case

In the following experiment, we evaluate FikoRE's performance when actual XR IP traffic is handled while simulating multiple concurrent UEs. In this setup, one UE is handling actual XR IP traffic captured by the emulator via Netfilter. The experiment is identical as the previous one, but using actual XR traffic. FikoRE simulates multiple concurrent UEs transmitting 5 Mbps UL and DL of synthetic simulated video traffic. The emulated XR UE is placed at different distances to the gNB in each experiment, while the other simulated UEs are placed randomly at a distance between 100 and 1500 meters from the gNB. To ensure repeatability, the

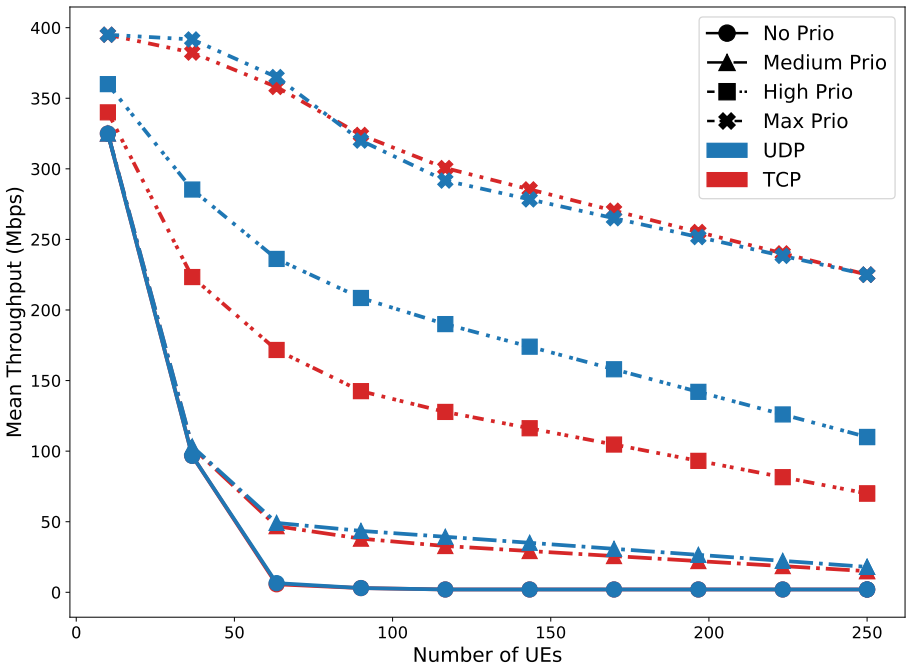


Figure 5.6: Emulated throughput for a user handling real IP data and sharing resources with other simulated UEs. Different UEs prioritization levels are used for the real IP traffic UE.

same random generator seed is used across experiments.

The XR IP traffic is generated using the XR setup used to generate the traffic models described in [34], for the full offloading scenario with medium resolution on the UL and 72 Hz 4K resolution on the DL. The mean throughput corresponds to around 41 Mbps and 119 Mbps for the UL and DL streams respectively. More details on how to test XR solutions with the emulator are described in [26].

In this experiment, we run the emulator configured on the 26.5 GHz frequency band and 200 MHz to ensure sufficient throughput for multiple UEs. The TDD configuration used ensures a balanced number of UL and DL slots. We used PF as the allocation metric. We repeat the experiments with a different number of UEs, between 0 and 100, and prioritization levels: no prioritization, medium, and high. The other simulated non-immersive UEs are in all cases set to low priority. The results of the measured round-trip times (RTT) for the actual IP traffic UE, with a server just transmitting the DL rendered frames without processing the UL frames, are shown in Fig. 5.6. As expected, the measured RTTs increase with the distance and the increment is proportional to the number of UEs.

We can also observe, as in the previous experiment, the strong effect of the prioritization values, decreasing the total RTT in more than a half. We can observe

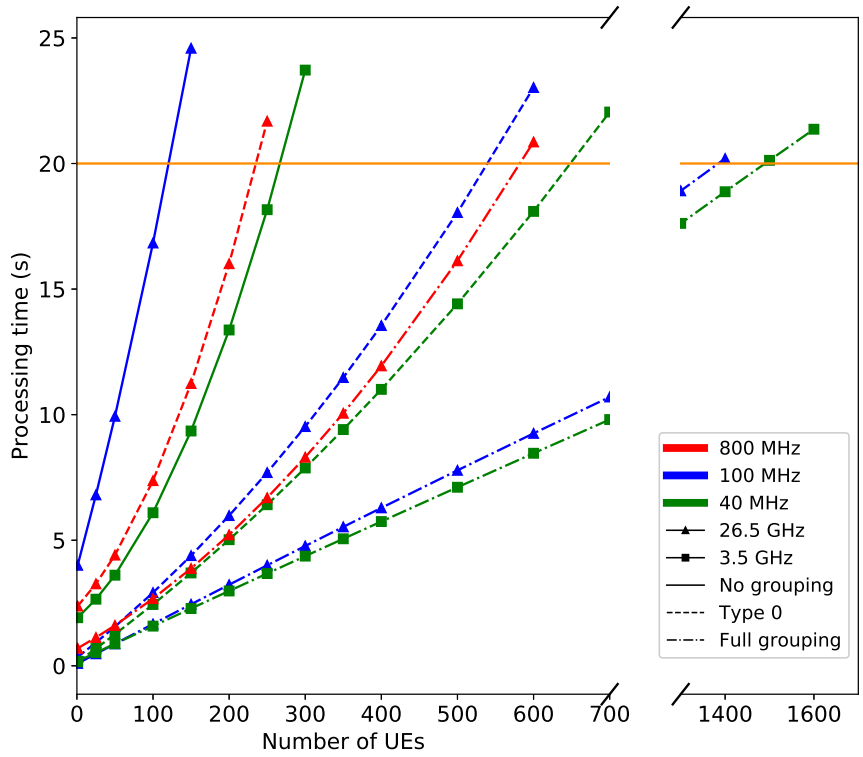


Figure 5.7: Number of simultaneous UEs that FikoRE can handle in real-time under different network configurations. In orange, the real-time total processing time deadline.

the importance of user prioritization when particular UEs must sustain a minimum network quality as in XR offloading. When the UE is sufficiently prioritized, the measured RTT decreases at a lower rate even when many UEs are demanding network resources. As in the previous experiment, we can observe how crucial these user prioritization techniques become for very demanding use cases such as XR and the Metaverse.

5.3.5. Performance Evaluation

We tested the number of simultaneous UEs that FikoRE can handle while meeting the 1 millisecond deadline in different network configurations, using laptop including an Intel Core i7-11800H with 16 GB of RAM memory. We tested 3 different bandwidth sizes (40, 100, 800 MHz), frequency bands (3.5 and 26.5 GHz) and RB grouping (type 2 or no grouping, type 0 or frequency grouping, or both frequency and time grouping). Each experiment run for 20 seconds of simulation time. We depict the actual runtime in Fig. 5.7 for the different combinations. We can observe that our thoughtful design and simplified assumptions allow FikoRE to handle between 100 and 1600 UEs depending on the configuration. Note that, contrarily to

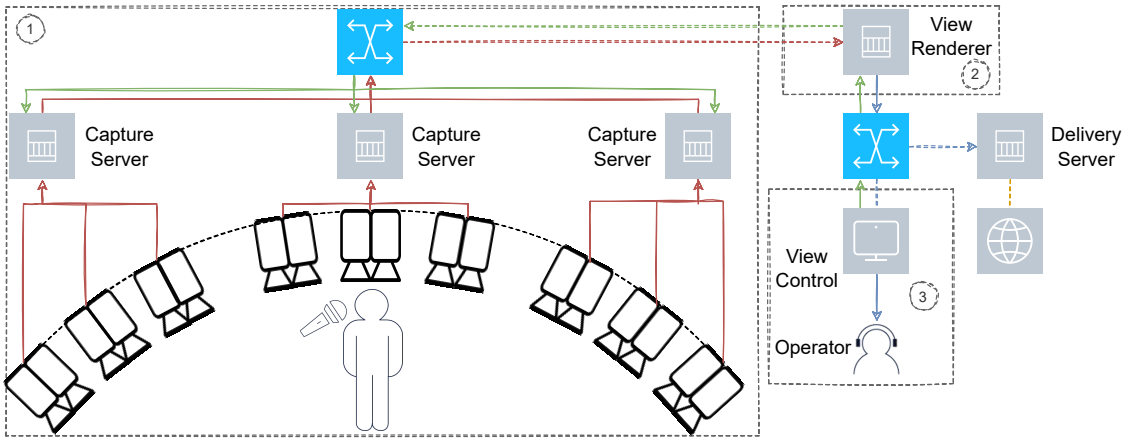


Figure 5.8: Schematic representation of the used FVV Live solution. In green, the control stream. In red, the raw depth and RGB streams. In blue, the rendered view. The main modules of the FVV are numbered as: i) Acquisition, ii) Virtual view computation, and iii) Virtual camera control module.

[24], we do not distinguish between background and foreground UEs, and all the UEs behave identically, which imposes higher computation requirements but more useful emulations.

We also tested the maximum actual IP traffic our emulator is able to handle in real-time. With a sufficiently high bandwidth, we achieved a maximum throughput of 6.5 Gbps of Iperf traffic, much higher that well-stablished state-of-the-art RAN emulators ($<$ 10 Mbps). We also tested the logging overhead in multiple scenarios, for a logging frequency of 100 Hz. We measured a mean overhead of less than 5% of the total timestep runtime.

5.4. Advanced Use Case: Free Viewpoint Video Distribution over Emulated 5G

Free Viewpoint Video (FVV) is a technology that allows the user to freely navigate a scene selecting an arbitrary path at any moment, all around the scene. FVV's tight requirements in terms of throughput and latency demand the use of advanced wireless network such as 5G.

In this test case scenario, we focus on the production and streaming of live FVV services through FikoRE's emulated 5G networks. The proposed FVV system, FVV Live[114], is a real-time, end-to-end FVV system based on consumer electronics Stereolabs ZED cameras. The general architecture of FVV Live, depicted in Fig. 5.8, can be divided in three main modules:

- **Acquisition Module** consisting of three capture servers, each one of them managing a set of three ZED cameras. All cameras perform the capture of

the scene on a synchronized way and deliver several RGB+D(epth) streams that are post-processed, encoded and transmitted over RTP. The RGB data is processed in a two stage chain that involves lossy encoding and RTP transmission. On the other hand, depth data post-processing implies the following stages: i) depth correction, ii) 12-bit quantization, iii) bit re-arrangement to fit the 12-bit depth map on a 8-bit YUV 4:2:0 frame (8 bits for Y + 4 bits for U and V) and iv) lossless encoding + RTP transmission.

- **Virtual View Rendering Module** comprising a view rendering server that performs the view synthesis computation according to the virtual camera position chosen by the operator through the control console. The input is the RGB+D video streaming and the control messages which indicate the desired position of the synthesized view. The synthesized virtual view is encoded and RTP transmitted to a control module, in which the user can monitor it and dynamically select the target view.

- **Control Module** used by an individual user or by an operator in a production setup to visualize the synthesized stream and dynamically change its position.

FVV Live has already been integrated with a real 5G Non Stand Alone (NSA) 3x RAN, as described in [115]. On these first field tests, the authors realized that the current development level of 5G mmW technologies cannot reach its full potential yet due to limitations of the configuration parameters, considerably hindering the distributed implementation of an advanced FVV system.

Our target is to analyze the performance of the proposed FVV Live implementation on a fully developed version of 5G mmW technologies emulated by FikoRE. The end goal is not only to validate FikoRE, but to learn how to improve and adapt our FVV approaches to advanced 5G deployments.

5.4.1. Experimental Setup

In the current experimental setup, cameras are connected to capture servers over a USB 3.0 link and capture servers are wired to the rendering server by an Ethernet link. To simplify and accelerate the experimentation process, a camera simulation module is used so the system is able to transmit prerecorded RGB+D streams. This module makes it convenient to run all kind of tests without having to physically deploy cameras.

The setup chosen to perform the experiments is shown in Fig. 5.9. The camera simulator module runs on a separated machine streaming 9 synchronized prerecorded RGB+D feeds over 9 different ports. The camera simulator machine is connected to the rendering server using an Ethernet connection. The rendering server is running FikoRE, which centralizes all communications between camera

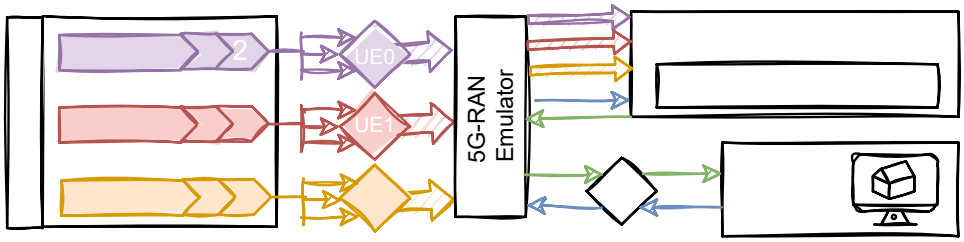


Figure 5.9: Architecture of the system used to perform the simulations. Main modules are the camera simulator, the 5G emulator, the view render and the control console.

simulator, view render and control console, so every packet is processed according the configured 5G emulation rules. The rendering server computes the synthetic view according to the virtual camera position chosen by the operator using the control console. The resulting view is encoded and transmitted back to the control console. The control console is connected to the processing server through a wired Ethernet connection.

Regarding the simulated video sequences, different resolutions and complexity levels have been considered which result in different bit-rate demands mainly because of the lossless encoding process of depth data. Two resolutions and frame-rates have been considered (1080@30 fps and 720@30 fps) and three different scene complexities have been defined: simple, medium and complex, where less scene complexity implies less bit-rate requirements. Finally, FVV Live allows to choose different transmission modes using all the 9 cameras or a subset of them according to the selected view, reducing the bitrate.

Packet losses have been evaluated by transmitting several video sequences from the camera simulator to the view renderer through FikoRE on different 5G configurations and measuring the packet error rate on the view renderer.

We also measured the Motion to Photon (M2P) latency, defined as the time span between the instant the camera operator requests a view change (by sending a control message to the virtual renderer) and the time when the new rendered view is visualized on the control console. More details on the M2P measurement methodology can be found in [114].

5.4.2. Scenarios and Results

Several tests have been carried out to evaluate the simulation capabilities of the FikoRE and the performance of FVV Live over the 5G simulated network.

Table 5.1: Average bit-rate and packet losses for both field test and emulation scenarios

	3 RGB+D				5 RGB+D			
	Simulation		Field Test		Simulation		Field Test	
	Mbps	% loss	Mbps	% loss	Mbps	% loss	Mbps	% loss
720 Simple	36.8	0	36.8	0	54.2	0	54.2	0
720 Medium	46.9	0	46.9	0	79.4	0	79.4	0
720 Complex	135.7	0	135.7	0.001	218.02	0.016	212.6	2.5
1080 Simple	165.7	0	165.7	0	-	-	-	-
1080 Complex	259.0	30	>220	32	-	-	-	-

Basic scenario

The first set of tests (basic scenario) recreates the real field deployment of FVV Live running over Nokia's 5G infrastructure, described in [115]. The aim of these tests is mainly to assess the accuracy of FikoRE with measurements of a real deployment. In [115], the network was configured with a 4DL:1UL for a total of 100 MHz. In this scenario, the system can only make use of a total of 20 MHz for UL transmission. Hence, FikoRE is configured with identical bandwidth, TDD configuration, frequency band (26.5 GHz), numerology and modulation order as in such deployment. No virtual users are created apart from the ones associated with the actual cameras' traffic the only ones being simulated by FikoRE. The scheduler used is the well-known Proportional Fair.

The sequences transmitted vary on resolution (720p and 1080p) and complexity (simple, medium and complex for 720p and simple and complex for 1080p). Also, transmission modes with a subset of 3 and 5 RGB+D streams have been used. For each configuration, a test has been carried out measuring the total output bit-rate, the packet error rate and the M2P latency.

Table 5.1 shows the results for the simulations and the actual field tests [115] respectively. Measurements obtained in the simulation match those of the field tests. For UL rates over 200 Mbps, this basic configuration starts reaching its limits [115], with the simulator setup being slightly more tolerant than the field results.

As we can see, for 720p, the simulated 5G network is able to manage all combinations of complexities and number of RGB+D streams with zero or minimum packet losses on the same range as the actual field tests results, except for the 720 complex at 5 RGB+D streams that performs better on the simulation, in terms of packet loss percentage. For 1080 simple at 3 RGB+D streams case, for both simulated and actual field tests performs with minimum losses. On the other hand, the current 5G configuration is not able to manage the 1080 complex sequence generating massive packet losses.

Table 5.2: M2P basic scenario results comparison

Test	Average M2P (ms)	Std Dev (ms)
Simulated test	221	62
Field test	279	67

Table 5.3: Packet losses for advanced scenario simulations

#added users	%loss, no priority	%loss, FVV priority
0	0.077	-
30	0.078	0.076
80	0.047	0.075
130	28.25	0.118
160	-	0.054
180	-	+30

Additionally, M2P latency has been measured for this specific configuration, using a 720 medium complexity sequence. Table 5.2 shows a comparison of both scenarios, simulated and field tests. The difference between both measurements could be due to the fact that in the field tests, the control console was connected to the network using a LTE link while the simulated measurement considered a 5G connection of the control console.

5.4.3. Advanced scenarios

These set of simulations have been carried out using a video sequence whose bit-rate is not supported on the basic scenario (Section 5.4.2) so we can evaluate the differences of the results on both cases and also the possible impact of the expected future 5G networks over FVV Live. A scenario with 9 RGB+D streams, medium complexity, 1080@30 fps sequence with an average bit-rate of 525 Mbps has been selected. Although this sequence overloads the bit-rate capacity of the basic scenario 5G configuration, it is expected to be supported by the simulated advanced 5G configurations.

In this scenario, we assume a fully developed mmW deployment, both from the

Table 5.4: M2P results for advanced scenario simulations

Test	Average M2P (ms)	Std (ms)
100 users, FVV priority	123	19
100 users, no priority	117	25
160 users, FVV priority	145	34
160 users, no priority	188	43

UE and base station sides. Consequently, we have a similar emulator configuration as the previous scenario, but with a total of 800 MHz available with 4DL:1UL TDD, increasing the effective UL bandwidth up to 160 MHz. Moreover, to test our implementation on a more realistic scenario, we decided to incorporate multiple virtual users to the simulation, which consume resources from the RAN. Several scenarios have been defined in terms of additional users added to the simulated 5G radio link (0, 30, 80, 130, 160 and 180 additional users). The users are consuming 5 Mbps of UL traffic. Thus, we can evaluate the performance of FVV Live running on a next generation 5G network, where the radio channel bandwidth is a shared resource between FVV Live and other users. As the emulator incorporates user prioritization capabilities, we decided to test our FVV Live implementation in two different scenarios: i) users' and FVV Live traffic have the same scheduling priority assigned on the emulation run, and ii) FVV Live has the highest priority. Each scenario has been tested using the 1080 medium complexity sequence (525 Mbps). For M2P latency evaluation several tests have been carried out using a 1080 simple sequence for different number of additional users (100 and 160 users) with and without high priority for FVV Live.

Table 5.3 shows the results for packet error rate measurements. As it can be seen, if all users have the same priority as FVV Live, the network manages the transmission maintaining FVV Live under a very low percentage of packet losses until the number of additional users reaches 130. At this point the high packet losses makes it impossible for the view renderer to synthesize any virtual view. On the other hand, if FVV Live priority is higher than other users' priority, the system is able to work under a very low packet losses percentage until the number of users reaches 180.

Table 5.4 shows the results for M2P latency tests for different number of users and different priority configurations. As it can be seen, for 100 additional users, the M2P latency measurements are similar between the equal priority scenario and the FVV Live prioritized one, because 100 users are not enough to saturate the network resources so the priority policies are not necessary. On the contrary, for a higher number of additional users it seems that the priority policy has a significant impact on FVV Live M2P performance because more users imply less resources for each user and thus, prioritized users get advantage on the bandwidth assignment over non-prioritized users.

5.5. Conclusions

In this chapter, we have presented FikoRE, our RAN open-source real-time emulator specifically designed for application-level experimentation and prototyping. The main goal of FikoRE is to allow researchers and developers to test novel advanced use cases with a special focus on high throughput solutions such as XR offloading.

After a thorough analysis of the state-of-the-art of Long Term Evolution (LTE) or 5G emulation tools, we highlighted the gap of real-time RAN emulators capable of handling enough throughput to test these demanding applications, being their maximum throughput limited to below 10 Mbps in most of the cases.

We described in detail the different design decisions, models and architecture implemented in our emulator, which makes it a great candidate to become an enabler of novel use cases and technologies which cannot yet be tested on cutting edge technologies such as 5G's mmW frequency bands.

Besides, we have tested FikoRE in several scenarios to validate its different modules. We have first validated how well the MCS index to application layer throughput mapping is achieved by FikoRE. We achieve this validation by comparing actual mmW measurements with our emulated measurements. The scheduler implementation and resource allocation are validated by comparing our emulated results with the results described in [28]. Furthermore, we have shown FikoRE's actual IP traffic handling capabilities while handling hundreds of simulated UEs. In this last set of experiments, we have shown the importance of a well-designed user prioritization approach for particularly demanding applications such as XR offloading.

We have also evaluated the overall performance of the emulator, showing how FikoRE is capable of emulating between 100 and 1,600 UEs in real-time, depending on the configuration. We have also tested its current actual IP traffic throughput limits, reaching a maximum of 6.5 Gbps. These results indicate that FikoRE can become a strong testing and emulation tool for the future of XR offloading solutions and the Metaverse, among other demanding use cases.

Finally, we have tested FikoRE's performance on a realistic Free Viewport Video (FVV) production deployment. The goal of this validation experiment is to highlight FikoRE's capability to emulate advanced 5G configurations that are still not available. Besides, we also aim to compare FikoRE's modelling capability by comparing the proposed FVV system (FVV Live) performance on an actual and emulated mmW deployment.

The proposed experiments included, first, the emulation of a real field end-to-end deployment of FVV Live running over a real 5G network. The results were compared to the measured metrics obtained on an actual 5G mmW deployment [115]. The results obtained from the comparison of both experiments were almost identical, which served as a validation of FikoRE's modelling accuracy. Second, simulations of advanced 5G configurations, that are not currently available for real deployments, have been performed to evaluate its impact on FVV Live system in terms of packet losses and M2P latency. This second experiment shows the advantages of using FikoRE to rapidly evaluate the impact of future 5G technology on complex media production systems, such as FVV Live, without the need for very complex deployments and for hardware configurations that currently are not available.

Chapter 6

IP Traffic Models for Advanced eXtended Reality Offloading Scenarios

XR offloading is a complex task with extreme requirements in terms of latency and throughput which requires a well designed and configured network, as described in Chapter 2. Successful XR offloading deployments requires a continuous development and improvement of the network, with a special focus on carefully designed resource allocation algorithms. However, it is not trivial for developers or researchers to have access to fully developed XR offloading implementations that can be used for testing and evaluating novel algorithms or network configurations. The current trend is to rely on pre-recorded or modelled traffic data, which is then fed to various simulation environments or actual wireless access network deployments. Pre-recorded traffic data allows to use extremely realistic data with simple use case-agnostic tools, such as tcpreplay [14]. On the other hand, traffic models allow to generate longer traffic traces while providing a greater flexibility than with pre-recorded traffic data. Even though it is true that the traffic characteristics for each XR use case can be very diverse, thus making it difficult to define a general purpose model, the access to modelled or pre-recorded XR traffic data can considerably accelerate and simplify the testing and prototyping steps.

A number of previous works deal with immersive multimedia applications' traffic capture and modeling or present ready-to-use models. Authors of [116] provide details on specific use cases employing augmented and virtual reality and how one can approximately model their behaviors using the models from 5G-PPP [117], [118]. In [119], the authors modeled augmented reality downlink traffic using a classical two-state Markovian process. In [120], a complete framework aimed to model XR application is presented, alongside an accurate statistical analysis and an ad-hoc traffic generator algorithm. Furthermore, the work carried out in [120] has been exploited to create a VR traffic generator framework for the ns-3 simulator [121]. Authors of [122] model 3GPP-compliant traffic cases for next generation mobile networks applications, which include advanced gaming, but no explicit AR/VR/XR case is considered. Generally speaking, the previous works mostly focus on providing models for the downlink traffic. However, as described in [28][29] and Chapter 2, advanced XR technologies require multiple complex algorithms to run simultaneously to provide the user with sufficiently high level of interaction, immersiveness and user experience. These algorithms, in many cases, require high-end hardware

[14] https://tcpreplay.appneta.com/

and therefore, can be considered potential offloading candidates. They require to be continuously fed with the sensor streams captured by the XR HMD, which can be as heavy or more as ultra-realistic rendered frames, imposing strict requirements on the uplink transmission. Aligned with this idea, 3GPP has recently included very detailed traffic models for AR and XR in Release 17 [27], differentiated according to the type of data streamed. While the considered VR use cases are still centered only on distributed rendering solutions with a special focus on downlink traffic, the AR traffic models also consider complex and heavy uplink traffic.

In this chapter, we provide realistic traffic traces and the associated models for two separate state-of-the-art extended reality offloading scenarios, both for downlink and uplink. Our goal is to complement and improve the models proposed in [27]. First, the proposed scenarios, described in Section 6.1, complement the ones described in [27]. Besides, we provide the raw data, which can be useful for many researchers not only to use it as it is for simulation or prototyping purposes, but for generating other models more suitable for their use. We also provide a set of XR traffic models obtained for the traces. Differently from other models in the state of the art, we also model the inter-packet arrival time, which, as we show in Section 6.6, can be extremely relevant for XR offloading resource allocation algorithms design.

The proposed scenarios are based on the research achieved over the last years and presented in previous chapters. The first scenario corresponds to ultra realistic VR rendering offloading, presented in Chapter 2. The second one corresponds to the egocentric body segmentation algorithm offloading scenario which end to end implementation is described in Chapter 4. These offloading scenarios sit on the very edge of the current state of the art. For this reason, we truly believe that our contribution will provide valuable tools to design, test, improve or extend wireless network solutions both in academia and industry. To our knowledge, this is the first work that provides both an accurate traffic dataset and the validated models for the mentioned use cases and applications. Our main contributions can be summarized as:

- An XR offloading traffic dataset for two different relevant offloading scenarios, captured for multiple streaming resolutions.
- XR traffic models obtained from the captured traces, including the inter-packet arrival time models, not available in most of the models provided in the state of the art.
- A thorough validation of the proposed models using a realistic 5G RAN emulator, showing how an accurate inter-packet arrival time can considerably improve the quality of the models for specific applications.

6.1. XR Offloading Scenarios

Our goal is to capture a relevant IP traffic dataset for two demanding XR offloading scenarios, that is, full XR offloading (Scenario A) and egocentric human segmentation algorithm offloading (Scenario B). In Scenario A, described in chapter 2 all the processing but the sensor capture is moved from the XR device to a nearby server. Differently from [27], we consider the VR HMD to be a very light device in charge of only capturing the sensor data. The sensor data are streamed to the server, where they get processed. The sensor info is used to render a new high-definition VR frame which is sent back to the device. This is a very relevant use case for advanced and future networks, which can enable ultra-light and wearable XR devices. In our case, we consider the sensor data to be generated by a stereo camera feed and inertial sensors. The inertial sensors traffic can be neglected as its associated throughput is much lower than the stereo camera feed throughput [28], [29]. This is an extremely demanding use case as the round trip times should lay below the frame update period, i.e., around 11 ms for a device running at 90 Hz. While there are some techniques to slightly expand this time budget, such as XR time warp [123], the latency requirements are still tight, especially for ultra-high definition XR scenes rendering, encoding, and transmission.

Scenario B, described in Chapter 4, focuses on the particular case of egocentric body segmentation [33], since this is a promising state-of-the-art solution for XR applications. The upstream traffic includes the stereo camera traffic while the server is sending back simple binary masks to the device in which the white pixels correspond to the user's body. The received masks are used by the XR device to render only the pixels corresponding to the user's body within the VR scene. While still a demanding offloading use case, the overall requirements are much lower than in Scenario A, since the downlink stream is just composed of single-channel binary masks.

6.2. Offloading Architecture

Our offloading architecture, described in [30] and Section 6.2, relies on two main agents to share data between different peers. On one hand, we have Alga, which connects individual peers. On the other hand, we have Polyp, a data re-router and replicator, in charge of transmitting the data from one source to one or multiple listening peers. We implemented a publisher-subscriber approach based on topics. When a client subscribes to a topic, Polyp is in charge of re-routing and replicating all the data of the topic toward this client. Similarly, when a client publishes data to a topic, Polyp ensures that these data are transmitted to all the peers subscribed to this topic. Our architecture allows direct communication between end clients without having to use Polyp. Polyp itself is a peer that can subscribe or publish to

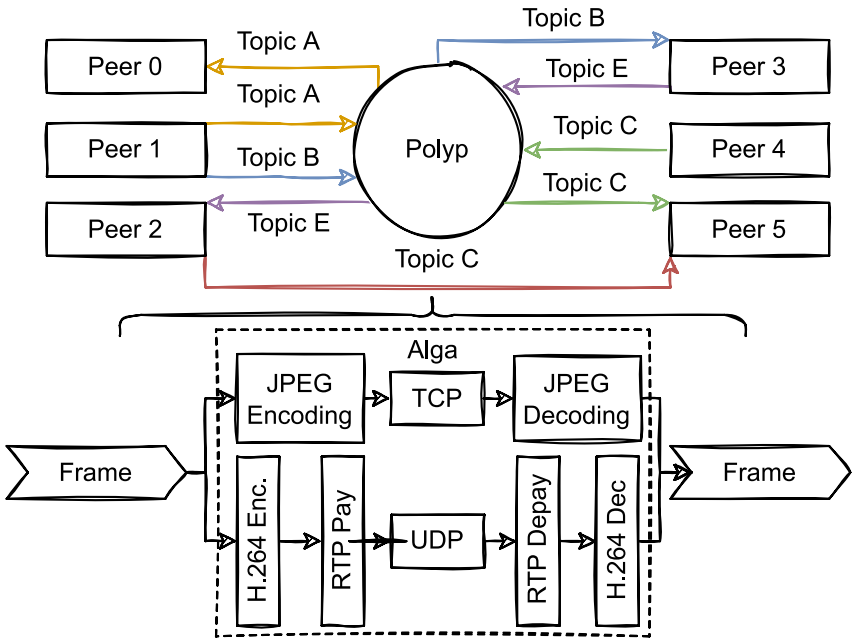


Figure 6.1: The proposed offloading architecture strategy and simplified data flow for a general multi-peer scenario (top). Alga data flow for both TCP-JPEG and RTP-H-264 implemented transmission pipelines (bottom).

a topic. Alga is in charge of creating all the necessary connections and transmitting the data. The general representation of our architecture is depicted in Fig. 6.1.

The first version of this offloading architecture, implemented Alga using TCP for IP traffic transmission. Besides, to efficiently avoid TCP disadvantages, we sent each frame separately encoded in JPEG. This architecture served us to use our ML egocentric body segmentation algorithm, running on a nearby server, with a commercial XR HMD, the Meta Quest 2 [15]. However, joint JPEG encoding and TCP transmission, while useful in many scenarios due to their associated reliability, as described in [30], were not originally designed to support high throughput and low latency. Therefore, we extended Alga's functionality incorporating H.264 video encoding [31] and RTP (real-time transport protocol) over UDP transmission [32]. To encode the sensor streams in H.264 and pack the data in RTP frames the architecture uses GStreamer [16].

For traffic control reasons and to preserve compatibility with Polyp's in-built functionalities, we need to have control over the individual video frames and attach the metadata associated with them, such as the destination topic, timestamps, etc. This metadata can also be useful for performance analysis or bottleneck detection. To achieve this goal, we use RTP extended headers. Thus, the metadata is added to each video frame as an RTP extended header, which can be decoded and read on

[15]https://www.meta.com/en/quest/products/quest-2/
[16]https://gstreamer.freedesktop.org/

the receiving end. This is achieved using GStreamer in-built functionality. Alga's data flow for both TCP and RTP/UDP modes is depicted in Fig. 6.1.

From the sending peer, the frames are fed to Alga in raw RGB format. Alga injects the raw frame along with its associated metadata into the GStreamer encoding and transmitting pipeline. If there are multiple peers subscribed to the same topic, the traffic is replicated and routed by Polyp, leaving this traffic untouched and just accessing the headers to read the target destination. In both this case and the case of direct traffic transmission between end peers using just Alga, the GStreamer pipeline receives and decodes the RTP frame. Once decoded, the frame can be accessed by the application layer.

6.3. Traffic Capture Methodology

As described in Section 6.2, our offloading architecture implementation has already been tested on a full end-to-end offloading solution using a commercial XR device, the Meta Quest 2. However, we decided to use a high-end laptop to emulate the XR offloading IP traffic for two main reasons:

- **Uncontrollable overhead** – our architecture is optimized for wireless offloading via WiFi or advanced RAN networks such as 5G. We need to capture the data on the transmitting peer to avoid any overhead introduced by the wireless transmission, traffic routing, congestion, etc. These potential sources of overhead can lead to latencies, jitter, or packet loss which strongly depend on the used configuration, wireless technology, and other external factors. It is out of the scope of this work to model the network behavior and its associated configuration. However, we could not find an efficient manner to capture the IP traffic being transmitted from the XR device.
- **Cover demanding XR offloading use cases** – the Meta Quest 2 is not capable of handling demanding XR offloading use cases due to its limited computation capabilities. Our target is to cover XR offloading use cases which are still not possible with current XR or wireless access points technologies.

Following these considerations, all the data were captured using a high-end laptop, with an Intel Core i7-10870H CPU @ 2.20 $GHz \times 16$, and 16 GB of RAM incorporating a Nvidia GeForce RTX 3070 GPU, running Ubuntu 18.04 LTS. The offloading architecture was set up and configured identically to an actual XR offloading deployment. For simplification and as we were not using an actual XR HMD, the IP traffic from each stream (Scenario A and Scenario B uplink and downlink streams) was captured via separate capture runs. Therefore, we used prerecorded data to capture the IP traffic. According to the scenarios described in Section 6.1 we recorded data for the following streams:

Table 6.1: Uplink and downlink resolutions and frame rates used to generate the proposed XR IP traffic dataset.

	Resolution @ FPS		
	Stream 1	Stream 2	Stream 3
High	2560 × 960 @ 60	3840 × 1920 @ 90	2560 × 960 @ 60
Medium	1920 × 720 @ 60	3840 × 1920 @ 72	1920 × 720 @ 60
Low	1280 × 480 @ 60		1280 × 480 @ 60

- **Stream 1 – Uplink stereo camera stream**: This corresponds to the frontal stereo camera data, which are transmitted in both offloading Scenarios A and B. The recorded data were obtained from the same stereo camera used in the end-to-end offloading solution [33], described in Chapter 4. We recorded a continuous stereo video stream of 2560×960 resolution at 60 Hz, the maximum supported by the camera. While XR devices are expected to run at rates above 90 Hz, the sensor data are not required to be updated so fast [28], [29]. The prerecorded data had a length of 15 minutes.

- **Stream 2 – Downlink rendered frames**: This corresponds to the immersive frames rendered on the server in Scenario A. In this case, we used a high-definition stereo video from a first person video game. The recorded video has a resolution of 3840 × 1920 and an update rate of 90 Hz.

- **Stream 3 – Downlink segmentation masks**: This corresponds to the binary pixel classification output by the egocentric body segmentation ML algorithm in Scenario B. From the stream 1, we estimated the black and white binary single channel masks for each frame using the segmentation algorithm described in [15]. Therefore, the resolution and update rate is the same as in stream 1 (2560 × 960 @ 60 Hz).

To expand and add extra value to the presented dataset, we downscaled the three streams to different sets of resolutions/update rates that can be useful for potential researchers and applications. A summary of all the resolutions and update rates we used to generate the traffic data is shown in Table 6.1.

Each of the resolution/frames per second (FPS) and transmission direction (uplink or downlink) streams was captured separately. To capture the IP traffic, the client reads the individual raw frames, one by one, from the selected stream and sends them using the described architecture. As both the client and server run on the same machine to accelerate and simplify the capture process, we set a streaming client connected to a server, which just discards the incoming packets using GStreamer's *Fakesink* module to avoid any additional overhead. There is no in-

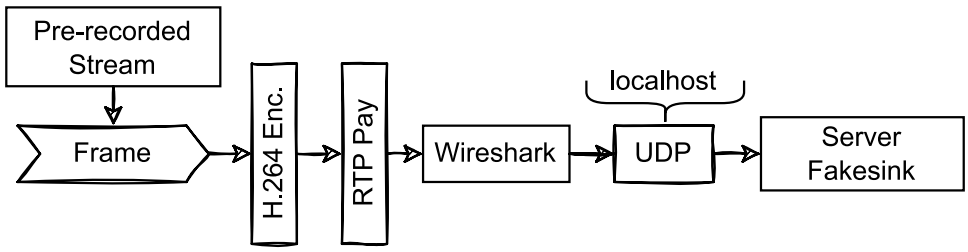


Figure 6.2: Simplified representation of the final setup used to capture the XR offloading IP traffic dataset.

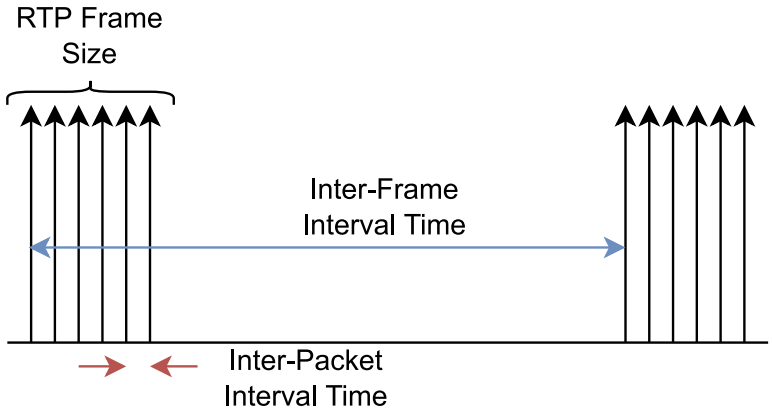


Figure 6.3: Simplified IP packets (black arrows) representation packed in several RTP frames. The RTP frame size, inter-frame, and inter-packet interval times are illustrated.

stance of Polyp and both client and server are directly connected using Alga in H.264-RTP mode: the raw frames are encoded using H.264 and packetized as RTP frames to be transmitted via UDP, in localhost. The IP traffic was captured using Wireshark, generating an individual packet capture (PCAP) file for each capture run. The final simplified capturing setup is depicted in Fig. 6.2. Each capture run had a duration of 10 minutes, for a total of 110 minutes of data.

6.4. Traffic Modeling

In addition to releasing the PCAP files publicly, we made a systematic effort to statistically model the most relevant video streaming IP traffic parameters: i) RTP frame size, defined as the size of each individual RTP frame, ii) inter-frame interval, that is, the time between individual RTP frames and iii) inter-packet interval, i.e., the time between successive packets within an individual RTP frame. These parameters are depicted schematically in Fig. 6.3. The main goal is to allow potential researchers, in the context of wireless communication systems analysis and evaluation, to generate realistic XR IP traffic, online or offline, based on the models derived.

Table 6.2: RTP frame size, inter-frame interval, inter-packet interval and individual IP packet sizes basic statistics from the captured data

Stream 1 - Uplink Stereo Camera

	Frame size (bytes)			Inter-Frame Interval (ms)		
	Mean	Std. Dev.	95th perc.	Mean	Std. Dev	95th perc.
Low	34602.44	9529.36	55735	16.76	0.26	17.12
Mid	86149.87	19936.04	132384	16.76	0.50	17.53
High	232084.33	28141.99	269008	16.80	2.57	21.29
	Inter-Packet Interval (ms)			IP Packet Size (bytes)		
	Mean	Std. Dev.	95th perc.	Mean	Std. Dev	95th perc.
Low	3.94	6.08	17.10	1280.79	356.58	1428
Mid	3.53	5.47	17.27	1364.81	244.83	1428
High	4.55	11.02	6.43	1403.88	154.31	1428

Stream 2 - Downlink Rendered Frames

	Frame size (bytes)			Inter-Frame Interval (ms)		
	Mean	Std. Dev.	95th perc.	Mean	Std. Dev.	95th perc.
72 Hz	207968.42	122929.70	396402	13.88	0.05	13.94
90 Hz	163548.89	116837.86	339396	11.11	0.04	11.17
	Inter-Packet Interval (ms)			IP Packet Size (bytes)		
	Mean	Std. Dev.	95th perc.	Mean	Std. Dev.	95th perc.
72 Hz	3.41	9.18	4.85	1400.04	171.38	1428
90 Hz	3.66	9.08	6.91	1392.66	191.91	1428

Stream 3 - Segmentation Masks

	Frame size (bytes)			Inter-Frame Interval (ms)		
	Mean	Std. Dev.	95th perc.	Mean	Std. Dev.	95th perc.
Low	4968.50	2175.03	7708	16.76	0.20	17.05
Mid	8273.98	3921.00	13970	16.75	0.61	17.77
High	24378.90	11440.59	43458	17.10	3.30	22.44
	Inter-Packet Interval (ms)			IP Packet Size (bytes)		
	Mean	Std. Dev.	95th perc.	Mean	Std. Dev.	95th perc.
Low	7.01	6.17	15.04	749.83	517.39	1428
Mid	5.87	9.61	15.34	933.53	527.48	1428
High	7.54	24.80	24.63	1216.27	419.88	1428

6.4.1. Data Pre-processing

The PCAP files are large and contain a lot of information that can be useful in future works, such as the transmitted bytes themselves or other relevant metadata. To derive the traffic models, we store the payload, timestamps and the new RTP frame marker bit of the individual IP packets coming into the arbitrary port used for transmission. This bit information is necessary to identify a new frame. Data pre-processing takes place in two steps, as follows.

In the first step, we obtain a list of all the captured IP packets, ordered according to their timestamp. For each packet, we keep the payload in bytes, the timestamp, and a custom boolean indicator, i.e., a combination of the bit marker and the timestamp separation, which determines if the IP packet initiates a new RTP frame or

not. This first pre-processing step is implemented using Python and Scapy[17] library to parse the PCAP file.

In the second step, we go through all the IP packets and group them in individual RTP frames according to the custom boolean indicator. Then we estimate, for each RTP frame, the total size in bytes (RTP frame size), the time in between consecutive frames (inter-frame intervals), and the time in between consecutive IP packets (inter-packet intervals). These parameters are stored in three separate arrays and saved as an NPY (Python NumPy format) file. These NPY files are the ones used to model the IP traffic. This second step is implemented in Python as well.

These two steps are applied to all the captured PCAP files. The final outputs are stored as individual files to easily identify each capture run. In Table 6.2 we show the basic statistics, i.e., mean value, standard deviation, and 95^{th} percentile of all the captured data cases, for the frame size, inter-frame interval, inter-packet interval, and IP packet size. The packet size information is useful to generate synthetic data from the fitted models.

6.4.2. Prior Data Analysis

Before taking any modeling decisions, we studied the histograms of the pre-processed data. In particular, we plotted the histograms for all the parameters to be modeled for all the captured data. In Fig. 6.4 we present examples of the RTP frame size, inter-frame interval, and inter-packet interval histograms, for the high-resolution Stream 1 and 3 (at 60 Hz), as well as Stream 2 at 90 Hz.

We observe, in all streams, that the inter-frame intervals are evenly distributed around a mean value that coincides with the frame update period according to the selected FPS value. Due to variable rate encoding, which guarantees low latency, the coding rate and the frame size may include peaks and variations. For the 60 Hz captured data, this is not an issue since the encoder is faster than the frame update period for all cases and resolutions. However, for very high resolutions and frame update rates, the coding rate needs to dynamically adapt, resulting in frame sizes with more than one peak, as shown for Stream 2 in Fig. 6.4. This also affects the standard deviation of the inter-frame interval, which is reduced in Stream 2 cases due to the stricter encoding time requirements.

Regarding the potential distributions to model the target parameters, we observe, that in both Stream 1 and 3, these parameters can be modeled as unimodal continuous distributions. On the other hand, we observe that the distribution of the RTP frame sizes of Stream 2 presents two local maxima. These local maxima are smaller for the higher frame update rate (90 Hz) depicted in Fig. 6.4. Nevertheless, we decided to model Stream 3 RTP frame sizes as continuous unimodal distributions

[17]https://scapy.net/

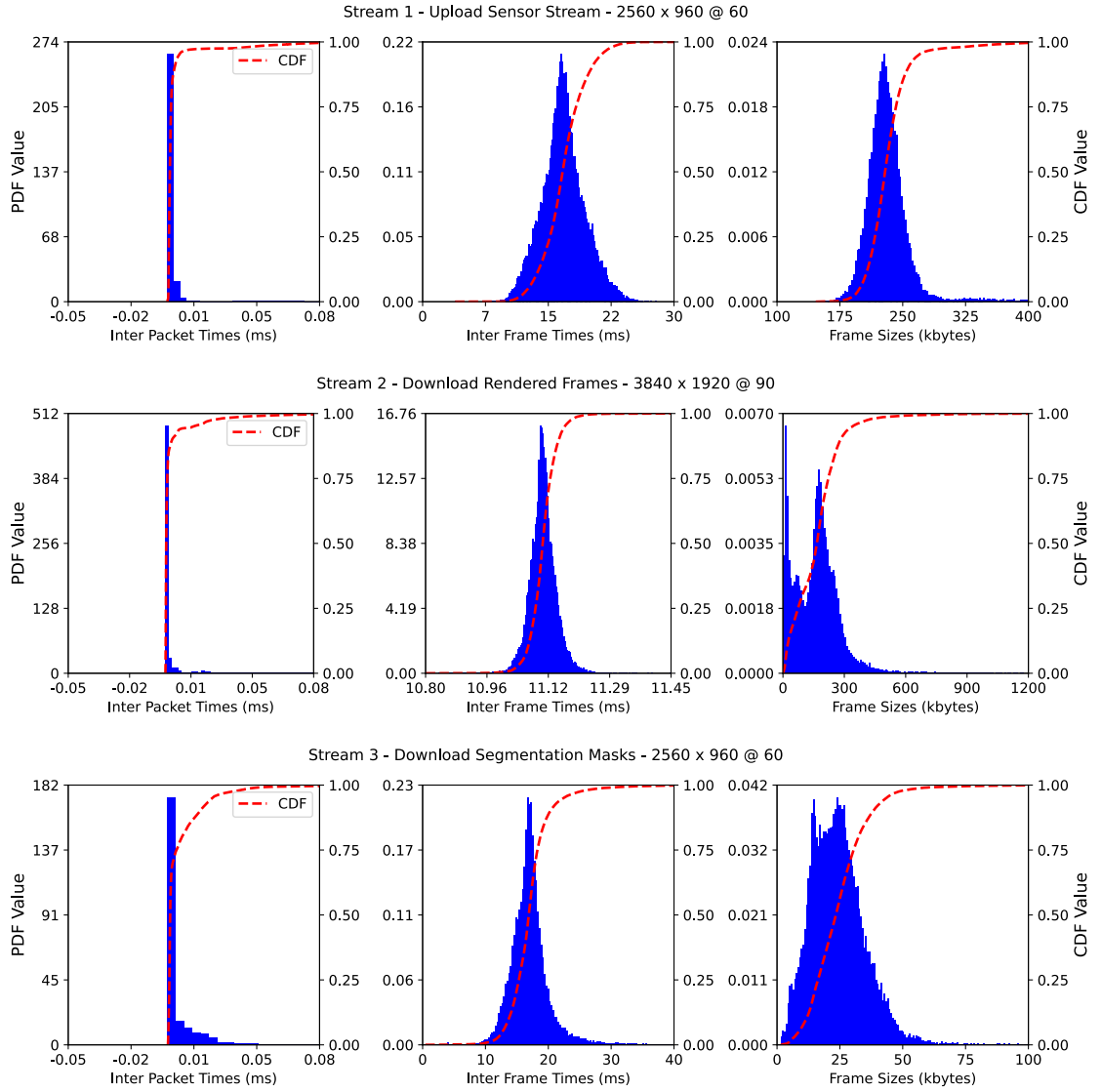


Figure 6.4: Histograms and CDFs from the captured data for Stream 1, 2 and 3 for the target parameters: inter-packet interval times, inter-frame interval times and frame sizes.

as well and check if they provide a sufficiently good fitting before testing multimodal distributions.

In Fig. 6.4 we can observe that the inter-packet interval distribution seems not to be unimodal, since slight changes in convexity appear. However, the inter-packet intervals lay in the order of the microsecond, as shown in Table 6.2. On that scale, many external sources can affect the measured value, such as the operating system particular operations, Wireshark processing, etc. Again, modeling these possible external factors that can affect the inter-packet intervals is out of the scope of this work. Therefore, we choose to move forward with the simple approach of modeling the inter-packet interval time as a unimodal continuous distribution.

6.4.3. IP Traffic Models

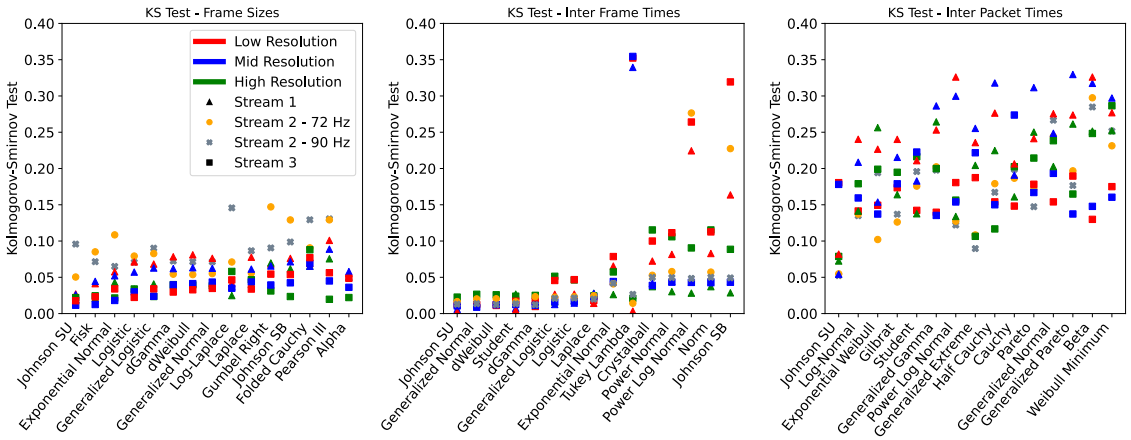


Figure 6.5: Best KS test scoring models for the target parameters to be modelled: RTP frame sizes, inter-frame interval times, and inter-packet interval times.

There is a wide range of well-established and commonly used continuous distributions for the parameters under consideration. To find the best candidate distributions that fit our data, we used Python's Scipy library[124]. Scipy is capable of modeling more than 90 different continuous distributions. We decided to fit all the distributions available and evaluate their goodness of fit using the Kolmogorov-Smirnov (KS) test [125]. The KS test quantifies the distance between the empirical Cumulative Distribution Function (CDF) $F_n(x)$ of a sample and the fitted CDF of an arbitrary distribution $F(x)$ as:

$$\mathrm{KS} = \sup_x |F_n(x) - F(x)|, \tag{6.1}$$

where $\sup_x$ is the supremum of all the set of distances across x values. The lower the KS test value, the better the fitting of the candidate distribution with the captured data. The KS test results of the 15 best-fitted distributions for each parameter and stream type are depicted in Fig. 6.5, sorted from best (left) to worst (right). We observe that Johnson's S_U distribution [126] obtains the best mean KS value across all the captured data. This distribution was proposed by N. L. Johnson in 1949 and has been historically used in finances. The key characteristic of Johnson's S_U distribution is its flexibility which originates from its four parameters that allow the distribution to be either symmetric or asymmetric. The PDF is expressed as

$$\begin{aligned} f(x,\gamma,\delta,\lambda,\varepsilon) = \frac{\varepsilon}{\delta \cdot m(x,\gamma,\delta)} \phi\Big\{\gamma + \delta \log(x) \\ \cdot [k(x,\gamma,\delta) + m(x,\gamma,\delta)]\Big\}, \end{aligned} \tag{6.2}$$

where

$$k(x,\gamma,\delta) = \frac{x-\gamma}{\delta}, \quad m(x,\gamma,\delta) = \sqrt{k(x,\gamma,\delta)^2 + 1}, \tag{6.3}$$

with γ and δ being the location and scale parameters, respectively, λ and ε the Johnson's S_U specific shape parameters, and $\phi(\cdot)$ the PDF of the normal distribution.

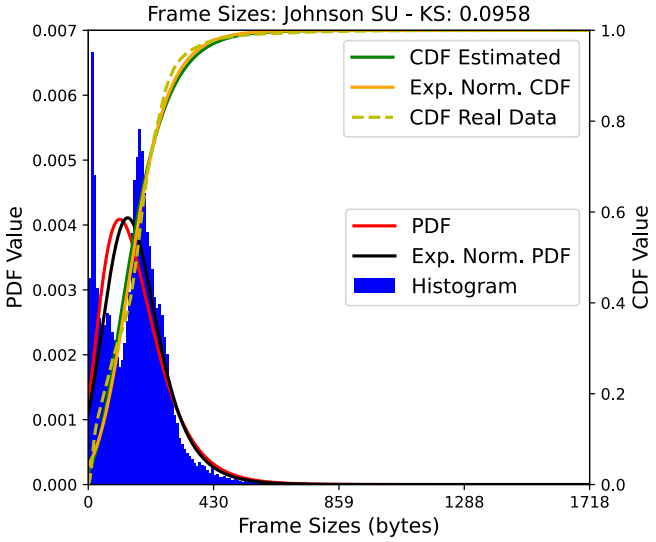


Figure 6.6: Johnson SU and exponential normal fitted distributions' PDFs and CDFs for the Stream 2 and 90 Hz case.

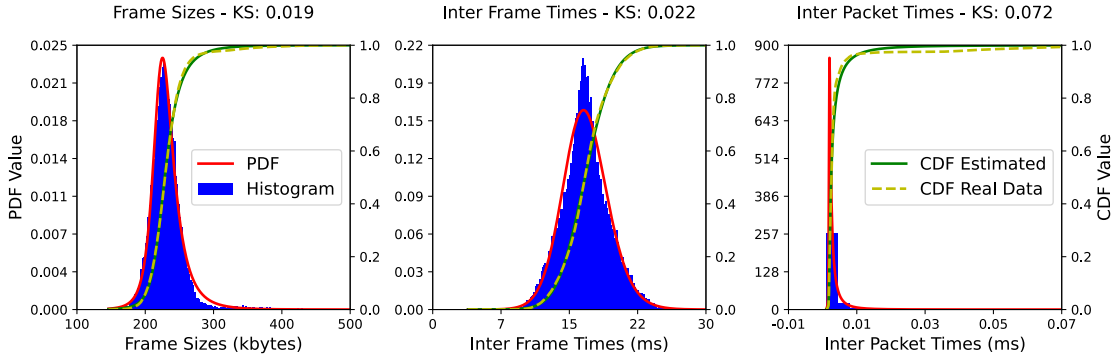


Figure 6.7: Histograms and CDF from the captured data for Stream 1 for the parameters: inter-packet interval times inter-frame interval times and frame sizes. On top, the Johnson SU fitted distribution's PDF and CDF.

By further inspecting Fig. 6.5, we notice that for the RTP frame sizes, the only case that Johnson's S_U does not provide the best fit is for the Stream 2 @ 90 Hz, for which the Exponential Normal distribution is the best. However, as we can see in Fig. 6.6 the practical differences between the two distributions for Stream 2 @ 90 Hz are small enough. Besides, even if Johnson's S_U fit is not as accurate as in the other RTP frame size distributions (see Fig. 6.7), the measured KS values obtained are low enough, with a good fit for the larger packet sizes. Therefore, we decided to model and evaluate the RTP frame sizes using the Johnson's S_U distribution for all the captured data. Similarly, we decided to use Johnson's S_U distribution to model also the inter-frame and inter-packet intervals. The parameters of Johnson's S_U distribution, as given by Python Scipy Library, for all the traffic parameters under consideration and captured data are summarized in Table 6.3.

Table 6.3: Fitted Johnson's S_U distribution parameters for the RTP frame size, inter-frame interval, and IP inter-packet interval, along with the KS test values.

Stream 1 – UL stereo camera

	Frame size			Inter-frame interval			Inter-packet interval		
	Low	Medium	High	Low	Medium	High	Low	Medium	High
Location (γ)	28204.99	73230.79	220497.49	0.01676	0.01682	0.01507	1.61 E-6	1.71 E-6	1.86 E-6
Scale (δ)	4990.40	12559.32	20161.10	0.000204	0.00062	0.007549	4.11 E-8	4.08 E-8	16.5 E-8
Shape-A (a)	-0.869	-0.781	-0.478	0.0281	0.114	-0.679	-1.200	-1.169	-1.451
Shape-B (b)	1.123	1.180	1.239	1.222	1.551	3.158	0.465	0.517	0.705
KS-Test	0.027	0.025	0.0192	0.0025	0.0059	0.0221	0.082	0.054	0.072

Stream 2 – Downlink rendered frames

	Frame size		Inter-frame interval		Inter-packet interval	
	70 Hz	90 Hz	70 Hz	90 Hz	70 Hz	90 Hz
Location (γ)	216500.38	-136187.44	0.01388	0.0111	1.59 E-6	1.62 E-6
Scale (δ)	52883.59	14897.09	4.373 E-5	2.30 E-5	5.40 E-8	5.31 E-8
Shape-A (a)	0.120	-9.761	-0.109	-0.132	-1.209	-1.258
Shape-B (b)	0.844	2.691	1.491	1.517	0.542	0.521
KS-Test	0.095	0.050	0.016	0.012	0.054	0.052

Stream 3 – Downlink segmentation mask

	Frame size			Inter-frame interval			Inter-packet interval		
	Low	Medium	High	Low	Medium	High	Low	Medium	High
Location (γ)	4144.51	5547.71	-3633.71	0.01676	0.01675	0.016346	1.17 E-6	1.53 E-6	2.14 E-6
Scale (δ)	1962.34	3879.20	16105.91	0.000118	0.000795	0.00231	1.91 E-8	6.07 E-8	20.70 E-8
Shape-A (a)	-0.442	-0.857	-3.800	-0.0092	-0.0096	-0.228	-5.705	-2.425	-0.995
Shape-B (b)	1.441	1.586	2.987	1.063	1.635	1.130	0.957	0.606	0.493
KS-Test	0.017	0.011	0.022	0.012	0.010	0.0227	0.180	0.178	0.078

6.5. Realistic Traffic Generation

Our next goal is to build a tool that allows the generation of realistic XR offloading IP traffic. Such a tool is useful for researchers and application developers to generate and use synthetic data for analysis or incorporate it into complex link-level or system-level simulations. While other video XR traffic [27] state-of-the-art models only consider the frame size and inter-frame interval for generating synthetic data, we believe that including the inter-packet interval data extends the applicability of our models to a wider range of research efforts. For instance, when designing novel or advanced resource allocation techniques, an accurate inter-packet interval model might be extremely useful and lead to better and more appropriate solutions.

To create synthetic data we have to generate random values from the fitted distributions. Towards this end, we used Scipy's *rvs* in-built function, which uses the inverse of the CDF to generate random samples from a specific distribution. In addition, we need the size of the individual RTP packets. In the real captured data this is not constant, as shown in Table 6.2, in terms of the IP packet sizes, especially for Stream 3, since the way the segmentation mask is coded and organized in RTP packets varies from the regular color video stream (1 and 2). In general, the packets of each RTP frame have a fixed size chosen in the encoding/RTP framing pipeline

Algorithm 1: Synthetic IP packets generation algorithm.

input : N_{RTP}, s_{RTP}
output: A sequence of IP packets
while $N_{RTP}^{generated} < N_{RTP}$ **do**
1. $s_{IP} \leftarrow \mathrm{FS}_{\mathrm{random}}$
2. $N_{IP} \leftarrow s_{RTP}/s_{IP}$
3. **while** $N_{IP}^{generated} < N_{IP}$ **do**
   3.1 $\Delta t_{IP} \leftarrow \mathrm{IPI}_{\mathrm{random}}$
   3.2 $t_s \leftarrow t_s + \Delta t_{IP}$
   3.3 Store new IP packet P(t_s, s_{RTP})
   3.4 $N_{IP}^{generated} \leftarrow N_{IP}^{generated} + 1$
4. $\Delta t_{IF} \leftarrow \mathrm{IFI}_{\mathrm{random}}$
5. $t_s \leftarrow t_s + \Delta t_{IF}$
6. $N_{RTP}^{generated} \leftarrow N_{RTP}^{generated} + 1$

(1442 bytes in our case). The first (including the RTP header) and the last are usually different. Depending on the chosen pipeline and configuration there may be smaller packets also in between, as in our case. However, these phenomena happen rarely as we can observe in the packet size histograms. The significant difference between the mean and the maximum packet sizes in low throughput streams, such as Stream 3, is expected because the number of packets between the first and last within an RTP frame is small (smaller than 5 in the low resolution Stream 3 case). Therefore, we consider two IP packet size options: i) the mean size value, as in Table 6.2 or ii) the 95$^{\mathrm{th}}$ percentile value. We refer to case i) as *Mean Packet* and case ii) as *Max Packet*.

Once we have the generators and packet sizes, we can easily define a procedure for synthetic realistic IP traffic generation, as described in Alg. 1. For each RTP frame among the N_{RTP} to be generated, we begin by getting its size s_{RTP} from the selected RTP generator, and by choosing the IP packet size s_{IP} equal to Max Packet or Mean Packet. Then we compute the total number of packets N_{IP} simply by dividing s_{RTP} by s_{IP}. We continue by generating N_{IP} packets of size s_{IP}, each with a specific timestamp t_s. The timestamp is computed by adding a random inter-packet interval Δt_{IP} to the previous packet timestamp. Once all N_{IP} packets are generated, a new randomly picked inter-frame interval Δt_{IF} is added to the current timestamp. The random values are generated from the modelled distributions. The above procedure is repeated for each RTP frame.

The described algorithm can be easily implemented in any programming language and therefore used in any simulation environment. Additionally, it can be used to create synthetic traffic traces by storing the generated packets in a separate PCAP file and utilize them at a later time.

Table 6.4: Mean throughput of the generated synthetic data in comparison with the captured traffic's throughput.

Stream 1 — Uplink stereo camera

	Captured (Mbps)	Max. size (Mbps)	Error (%)	Mean size (Mbps)	Error (%)
Low	16.51	16.49	0.12	16.49	0.12
Med.	41.11	41.15	0.09	40.96	0.36
High	110.55	110.50	0.05	110.27	0.25

Stream 2 — Downlink rendered frames

	Captured (Mbps)	Max. size (Mbps)	Error (%)	Mean size (Mbps)	Error (%)
70 Hz	119.79	121.36	1.29	120.83	0.86
90 Hz	117.76	117.74	0.02	118.60	0.71

Stream 3 — Downlink segmentation mask

	Captured (Mbps)	Max. size (Mbps)	Error (%)	Mean size (Mbps)	Error (%)
Low	2.37	2.35	0.89	2.36	0.51
Med.	3.95	3.92	0.66	3.94	0.35
High	11.4	11.44	0.38	11.44	0.32

6.6. Validation Experiments

In this section, we test the traffic generated with the methodology described in the previous section, over a realistic RAN scenario, to determine its ability to accurately mimic the behavior of the captured XR offloading data traffic. To do this, we first compare the average throughput obtained from the captured data with the corresponding generated synthetic data obtained. Then, we study the behavior of the different synthetic data models in terms of application layer throughput and latency in the most relevant offloading scenarios using a real-time 5G RAN emulator. Finally, we thoroughly examine the impact of the type of traffic model used on resource allocation by comparing synthetic data from different models with actual XR traffic.

The first step is to compare the generated mean throughput of the synthetic data using the modeled Johnson's S_U distribution with the captured data. The mean throughput results of the captured and synthetic data, for both Max Packet and Mean Packet cases (IP packet sizes), are shown in Table 6.4. The differences between the synthetic and captured data throughput are also included as percentage differences. We can observe that the throughput differences are low in all cases, with a peak of 1.29% for Stream 2 and 72 Hz case. All other cases present differences below 1%, for both IP packet sizes, with an average error of 0.435%, and 0.438%, for Max Packet, and Mean Packet case, respectively.

The next evaluation step is to compare the behavior of both synthetic and cap-

tured data on a realistic advanced 5G RAN deployment. Towards this end, FikoRE, our open-source 5G RAN real-time emulator [25], [26] described in Section 5. For our validation experiments, FikoRE runs as a simulator since we are not injecting actual IP traffic, but the traces from the captured or synthetic data. We tested the two scenarios described in Section 6.1, with the following setup:

- **Scenario A – Full Offloading**: On the downlink side, we chose to evaluate the 72 Hz rendered frames stream since it represents the current rendering offloading possibilities of commercial XR devices such as the Meta Quest 2. The Meta Quest 2 devices are capable of performing offloaded rendering, via a WLAN network, to a laptop in charge of rendering the immersive scene. The recommended setup is 72 Hz, being the rendering resolution of 1832×1920 per eye, which is slightly smaller than our captured data for the rendering frames stream. The uplink corresponds to the sensor stream (Stream 1) with a stereo resolution of 1920×720.
- **Scenario B – Egocentric Human Body Segmentation**: Successful deployment of this scenario is described in Chapter 4. While our deployment uses smaller resolutions, we evaluated the scenario in which both Stream 1 and 3 use a resolution of 1920×720.

Both offloading scenarios were evaluated in three different network configurations:

- **Configuration A – Multiple background UEs and a single immersive UE with PF**: In this scenario we simulated multiple UEs which are transmitting 5 Mbps of traffic in each direction. The throughput is not continuous, but is synthetically generated using the video streaming models from [127] applicable for streaming applications such as Netflix. The emulator is set up with a single carrier of 100 MHz bandwidth on the 26.5 GHz mmW frequency band. Resource allocation takes place based on the PF metric [128], using 1:1 (downlink:uplink) time division duplexing (TDD). We tested this network with a single immersive UE and 0, 20, 40, 60, 80 and 100 background UEs with 5 Mbps traffic in each direction. The network starts saturating around 80 simultaneous UEs.
- **Configuration B – Mutiple immersive UEs with PF**: In this scenario we have multiple immersive UEs, all using the same synthetic data. The throughput per UE is much higher than in Configuration A, so we increased the total bandwidth to 200 MHz in order to be able to simulate more UEs before reaching network UE saturation.
- **Configuration C – Mutiple immersive UEs with MT**: this setup is identical to Configuration B only changing the resource allocation metric used from PF to MT [128].

Table 6.5: Common simulation parameters used in all the experiment runs

Simulation Parameters	
TDD Configuration	1(UL):1(DL)
Modulation	256-QAM
Frequency Band	26.5 GHz
UE MIMO Layers	2
Allocation Type	0
Allocation Configuration	1
Scenario	Rural Macrocell

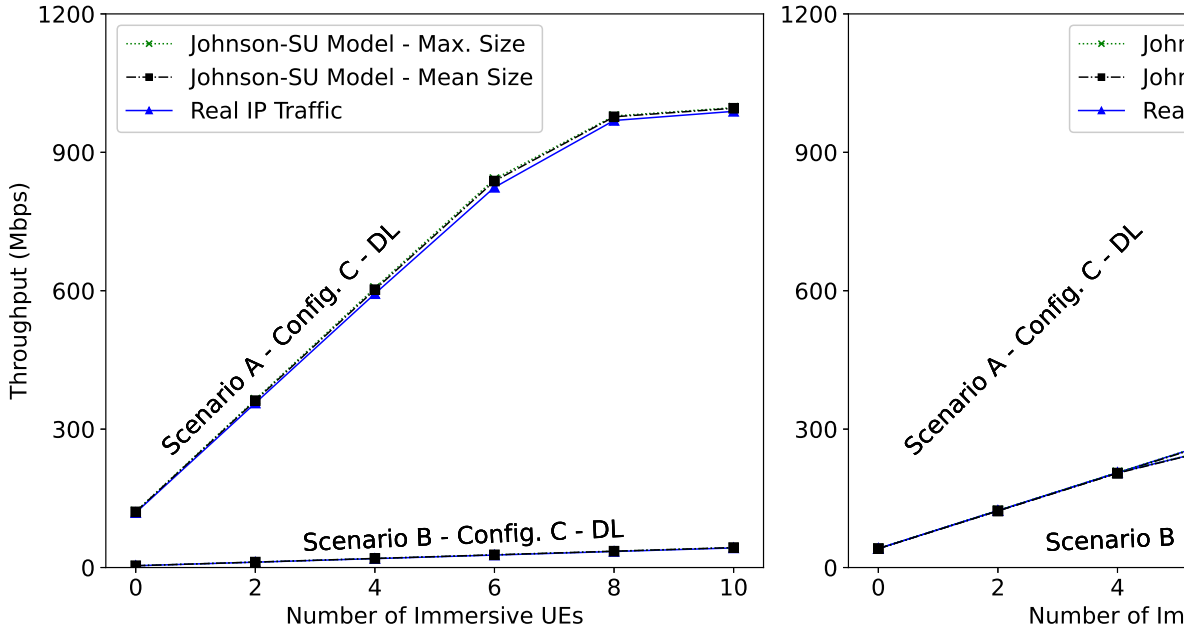


Figure 6.8: Mean downlink throughput measured for Scenario A Configuration C for the captured and synthetic data.

All three configurations have in common the simulation parameters included in Table 6.5. Each individual simulation run has a duration of 500 seconds and is repeated for each combination of configuration, number of UEs, offloading Scenarios (A and B), and type of data (synthetic with both packet types and captured data). In all cases, there is a "principal" immersive UE closer to the emulated gNB than the other simulated UEs, from which we obtained the measurements used in this analysis. The goal is to study and compare the behavior of each type of IP traffic data at the application level, so we evaluated the application layer throughput and latency. The throughput is measured as the total mean throughput transmitted by all UEs. The latency, is measured only for the principal immersive UE. All the stochastic models, including the initial position of the non-principal UEs, have the same random seed across the experiments. The principal UE is placed 100 m away from the gNB to ensure it has priority regardless of the metric used for allocation,

Table 6.6: Emulated application level throughput and latency comparison between captured and synthetic traffic. The synthetic experiments are repeated using the maximum and mean packet sizes.

Configuration A – Multiple background UEs and a single immersive UE with PF

	Scenario A: Full offloading				Scenario B: Deep learning offloading			
	Throughput error		Latency error		Throughput error		Latency error	
	DL	UL	DL	UL	DL	UL	DL	UL
Max. pkt size (%)	0.85	0.06	0.07	0.27	0.28	0.06	1.48	0.23
Mean pkt size (%)	0.59	0.13	0.58	0.38	0.35	0.13	1.55	0.51

Configuration B – Multiple immersive UEs with PF

	Scenario A: Full offloading				Scenario B: Deep learning offloading			
	Throughput error		Latency error		Throughput error		Latency error	
	DL	UL	DL	UL	DL	UL	DL	UL
Max. pkt size (%)	0.57	0.23	0.71	0.15	1.55	0.16	0.04	0.48
Mean pkt size (%)	0.65	0.46	0.29	0.32	1.50	0.28	0.03	0.36

Configuration C – Multiple immersive UEs with MT

	Scenario A: Full offloading				Scenario B: Deep learning offloading			
	Throughput error		Latency error		Throughput error		Latency error	
	DL	UL	DL	UL	DL	UL	DL	UL
Max. pkt size (%)	1.72	0.23	0.83	0.34	1.55	0.16	0.06	0.20
Mean pkt size (%)	1.28	0.46	0.42	0.38	1.92	0.28	0.03	0.12

while the rest are placed randomly, at a longer distance.

The application layer mean throughput results obtained for the downlink transmission of Scenario A for Configuration C, are depicted in Fig. 6.8. It is evident that the difference between the real, and the modeled data, for the total of UEs, is very low. We observe that from 8 UEs onward, the network starts saturating and the throughput does not increase linearly. This is because UEs with worse channel quality get fewer allocation grants. The measured latency behaves similarly showing low differences. Furthermore, similar results were obtained for all other configurations and scenarios. Overall, the throughput and latency differences between the captured and synthetic data obtained from FikoRE simulations are gathered in Table 6.6. These differences are expressed by the relative mean error across emulation runs with different numbers of UEs. We observe that they are very low, below 2%, in all cases. Besides, the differences between the Max Packet and Mean Packet cases of the IP packet sizes are negligible, with a mean difference of less than 0.04%. These results validate the goodness of the fitting of the proposed models for

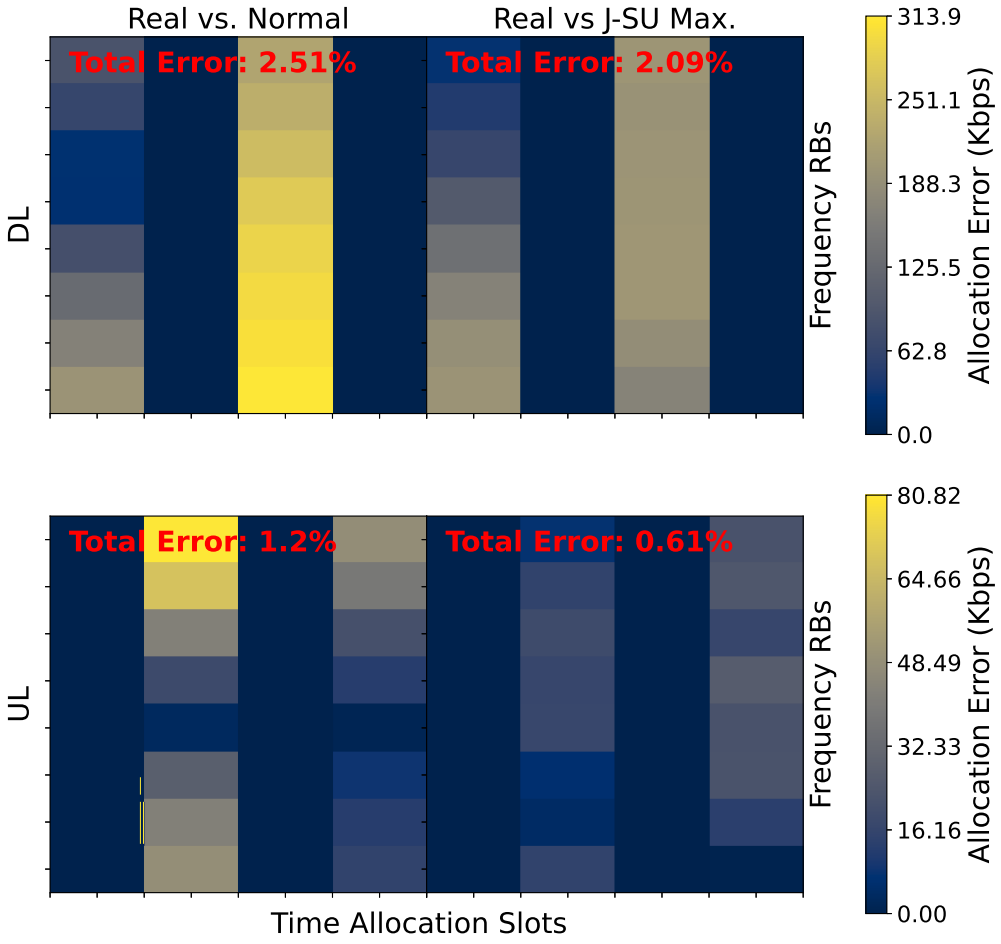


Figure 6.9: Example of allocation error matrices for both transmission directions (UL and DL) between the captured and synthetic data. The error is estimated for the entire grid. This case corresponds to Scenario A and Configuration B.

application-level simulations.

As a further step, we assess how well the synthetic traffic data generated with our model behave on the lower layers of the stack compared to the captured data. In particular, we study the resource allocation differences when using the captured or synthetic data as input for the simulator. Besides, we highlight the necessity of an accurate model which includes also the inter-packet intervals, contrary to the models proposed in [27]. In this context, we generated synthetic data using a simple Normal model using the statistical metrics from the captured data included in Table 6.2. However, instead of generating multiple IP packets within an RTP frame, we generated all the bits within the RTP frame in the same timestamp. By doing so we do not only highlight the necessity of an accurate model in terms of RTP frame size and inter-frame interval, but also the relevance of the inter-packet interval models. We refer to this simpler model as “Norm” model.

For this validation step, we also used FikoRE which is capable of logging every single allocation step, that is, how the resources are allocated at each subframe. We use this information to compare the differences in terms of the allocated throughput, for each RB within the allocation grid. More specifically, we measure the number of bits allocated for each RB and each UE. The number of RBs along the time and frequency axes depend on the bandwidth and selected numerology. The allocation error is estimated by comparing the bits allocated to each RB and UE when using the synthetic data from different models and using the actual XR traffic. We can build, for each UE, the allocation matrices illustrated in Fig. 6.9. These matrices express the resource allocation differences, or allocation errors, between a selected model and the actual XR traffic in bits per second, so the metric does not depend on the total duration of the simulation run. To estimate the allocation error of the entire grid as a percentage of the total transmitted error, we use the formula

$$e(\%) = 100\frac{\sum_{i=1}^{K} |t_c(i) - t_m(i))|}{\sum_{i=1}^{K} t_c(i)}, \tag{6.4}$$

where $t_m(i)$ and $t_c(i)$ denote the allocated throughput of the model being evaluated, and the captured data, respectively, for the ith RB ($1 \leq i \leq K$) along the total simulation time, with K the total number of RBs. To really understand how the different sources of traffic data are being allocated, we decided to simulate a single UE, the principal one, in each run. By doing this, we avoid the effects of the selected configuration (such as the allocation metric, UEs channel quality, etc.) that directly affect the resource allocation procedure and can lead to inaccurate conclusions.

Using the same configuration parameters described in Table 6.5, we tested multiple combinations of total bandwidth and numerology μ, which directly affect how the resource allocation grid is built, for a single immersive UE. In specific, we tested bandwidths of 40 MHz with $\mu = 1$, 100 MHz with $\mu = 2$, 200 MHz with $\mu = 2$, and 200, 400, 800 MHz with $\mu = 3$. Each simulation run had a duration of 500 seconds. The simulations were repeated for each configuration, Scenario (A and B), and source of data (captured, Jonhson's S_U with Max Packet size, and Norm). The synthetic data generated using the Mean Packet size presented no evident differences with the Max Packet size option.

Observing the measured allocation errors depicted in Fig. 6.10 we can extract several conclusions. First, we notice that the allocation error is considerably higher for the Norm simpler model compared to the proposed Johnson's S_U model. Besides, the error difference increases rapidly in favor of Johnson's S_U model as we configure the emulator with more total bandwidth. Increasing the numerology also negatively impacts the performance of the Norm model. For low bandwidths, the error difference is low, as an RTP frame does not fit in a single subframe and has to be transmitted along several subframes. Therefore, the entire resource allocation grid gets saturated and the allocation differences, being estimated in comparison with the total allocated throughput in each RB, become hard to measure. On the

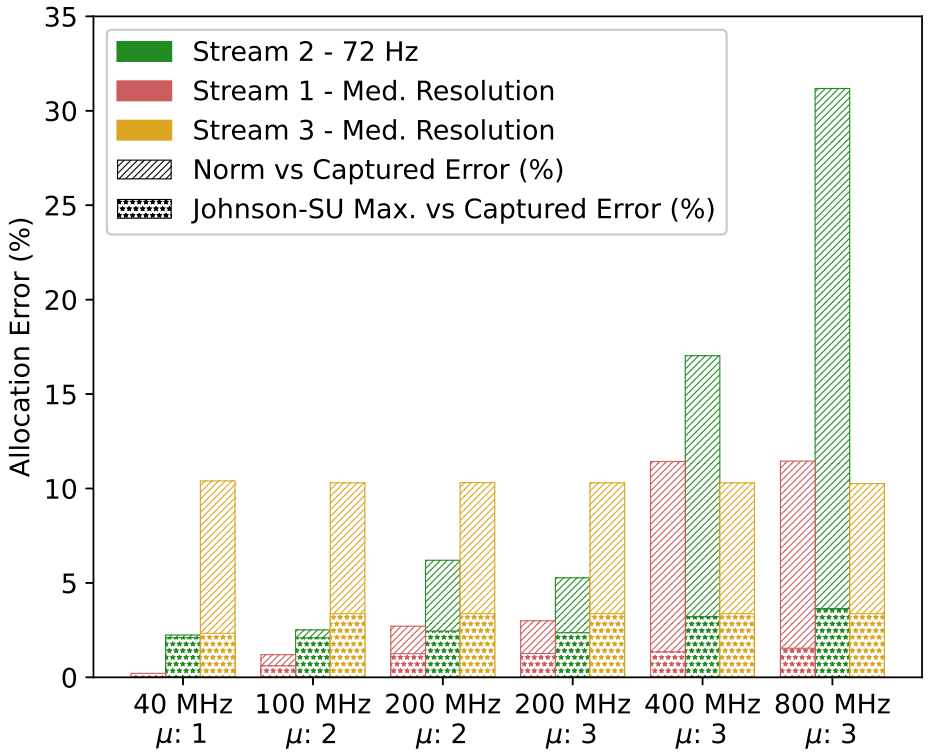


Figure 6.10: Measured allocation errors between the captured and synthetic data using our emulation tool for each transmission direction (UL and DL) and validation Scenario (A and B) for different numerology and bandwidth configurations. All simulations were done for a single UE.

contrary, for higher bandwidths, not all the RBs are allocated for each RTP frame and the differences become more noticeable. Our intuition is that the difference that we observe for higher bandwidths could also be observed if we could discard the saturated subframes. In addition, the allocation error of the proposed Johnson's S_U models remains almost constant along the test configurations, which clearly is not the case for the Norm model. Thus, we get a strong hint of the importance of obtaining accurate models which include the inter-packet intervals, especially for high numerologies and bandwidths, in designing successful resource allocation techniques.

6.7. Conclusions

This chapter aims to provide realistic traffic traces and associated models for XR offloading scenarios to complement and improve upon the models proposed in previous works, such as [27]. We proposed two XR offloading scenarios that are at the cutting edge of the current state of the art. The first scenario represents a full offloading solution in which the XR HMD captures and transmits sensor data to a nearby

server or MEC facility for processing and rendering ultra high definition immersive frames, which are then transmitted back to the device. The second scenario focuses on offloading heavy ML algorithms, specifically our novel real-time egocentric human body segmentation algorithm, which allows users to see themselves within the virtual scene.

The traffic data was captured using our own offloading architecture, described in [30], with additional functionality presented in this chapter. To avoid any uncontrollably overhead that we did not aim to model, the data has been captured on the sender side, using a local host network. The IP traffic was captured for multiple resolutions for both the uplink and downlink streams and both offloading scenarios.

The collected data was cleaned and post-processed, and we conducted a thorough analysis to determine the most appropriate modeling approach. We chose to model the three main components of video traffic – frame size, inter-frame interval, and inter-packet interval – using continuous single modal distributions. While many video or XR traffic models, such as [27], do not include inter-packet interval information, we consider it a crucial feature to include in XR traffic models, especially for resource allocation techniques design and optimization, as demonstrated in our validation experiments. We fit multiple continuous distributions to the data for all resolutions and found that the Johnson-SU distribution provided the best fit, as determined by the measured fitting goodness estimated using the KS test.

The Johnson-SU distribution was fitted for all target parameters, scenarios, and resolutions, and the fitted parameters are summarized in Table 6.3. With these models, we generated synthetic data and used it in validation experiments with our open-source 5G RAN emulator [26]. These experiments compared the performance of the captured data and synthetic data from both the application and resource allocation layers. At the application layer, we found that our models can generate realistic XR traffic data for the proposed scenarios. In the resource allocation layer, we demonstrated the importance of including inter-packet interval time for designing advanced resource allocation techniques specifically optimized for XR offloading.

In conclusion, the data and models presented in this chapter are valuable resources for the design, testing, improvement, and expansion of wireless network solutions in both academia and industry. They offer a comprehensive approach for studying XR offloading scenarios and provide insight into the importance of considering inter-packet interval times for resource allocation techniques. Overall, this chapter provides a valuable contribution to the field of wireless networks and XR technology.

Chapter 7

Conclusions and Future Research

7.1. Summary and Conclusions

In this thesis, we have studied the requirements and implemented some of the necessary tools and algorithms to achieve successful distributed XR applications. All our contributions, both in terms of theoretical analysis and implementation efforts, sit on the very edge of XR offloading technologies.

First, we have thoroughly analyzed the state of the art of XR technologies. Particularly, we have broken down both AR and VR technologies identifying their main individual algorithms. Each of these algorithms' state of the art has been studied, extracting the hardware and real-time requirements, and identifying their input and output expected data format and size. The results from the theoretical analysis have been used as input to extract a set of realistic throughput and latency requirements for a selected group of use cases. We proposed a simple optimization problem to extract these requirements. Differently from other state of the art requirements studies, ours is focused on the algorithms rather than high level use cases. This allows to extrapolate our results to other custom use cases not covered in our work. Besides, our analysis is not only considering the DL traffic, but also highlights the tight requirements imposed by the heavy UL traffic foreseen in most of AR and VR distributed implementations. Finally, we have used the estimated requirements to propose potential 5G architectures which can fulfil them.

The proposed requirements have been used to design our own XR offloading architecture. In general, there are few research examples that describe a full offloading architecture implementation. Besides, the authors usually do not take into account the heavy UL traffic expected to be sustained when designing the architecture. In this thesis, we have presented our own offloading architecture. We have provided exhaustive implementation details of the initial version of our architecture. This version, based on JPEG encoding via TCP, has been successfully tested using an actual VR HMD on WiFi and 5G mmW technologies. Besides, we have provided an initial description of our newest functionality which allows H.264 encoding transmitted over RTP.

The TCP version of our architecture has been used to implement, to the best of our knowledge, one of the first full end to end distributed XR applications. In particular, our egocentric semantic segmentation algorithm runs, in real-time, in a server wirelessly connected to a commercial VR HMD, the Meta Quest 2, using

our architecture. The offloaded algorithm takes as input the RGB feed captured by a stereo camera placed on the front of the HMD, and estimates which pixels corresponds to the user's body creating a binary mask. The mask is transmitted back to the HMD, rendering only the pixels from the frontal stereo camera that are identified as the user's body. With our implementation, the users are able to see themselves within the VR scene, in real time, substituting traditional virtual avatars. This solution is a breakthrough in XR technologies. In this thesis we have described all the implementation details of our end to end solutions, along with the results from the carried out user study experiments. The goal of these experiments was to validate both the segmentation algorithm and the end to end implementation.

Another research gap that we have identified is the difficulty to access fully developed 5G deployments which can be used to test novel distributed XR implementations. While a typical alternative would be the use of real-time network emulators, the ones available in the state of the art are not scalable enough (in terms of number of simulated UEs) or can not support high throughput in real time. In this thesis we present FikoRE, our real-time 5G RAN emulator capable of simulating hundreds of 5UEs while supporting high throughput of actual IP traffic (above 1 Gbps) in real time. The emulator is fully described in this thesis, and is complemented by other details described in FikoRE's open source repository [26]. We have also performed a set of validation experiments to check FikoRE's emulation accuracy by comparing the emulated behavior with measurements taken from an actual mmW deployment. Besides, we have showcased its potential as a reference testbed for advanced 5G use cases by testing it along a fully developed Free-Viewport Video streaming implementation.

Finally, to support telecommunication engineers and researchers in the development of optimized solutions which aim to facilitate and fulfil the requirements of distributed XR offloading implementations, we have created a realistic XR traffic dataset using the RTP version of our architecture and our VR hardware. We have captured the data for XR offloading use cases which are complementary to the ones presented in the specifications [27]. Besides, we have presented XR traffic models derived from the captured data so that other researchers or engineers can generate their own synthetic realistic XR traffic. The models have been validated in several experiments using FikoRE. The proposed dataset and models can become relevant tools for those researchers that have no access to fully developed distributed XR implementations.

Given all these contributions, we can conclude that we have provided relevant advances towards the implementation of successful distributed XR solutions, paving the way for more immersive, wearable and affordable XR. In this thesis, we have already pushed the limits of current XR implementations by means of 5G and other advanced wireless networks. And more essential, we have highlighted how important is for the development of XR technologies, the improvement of wireless technologies,

and vice-versa.

7.2. Future Research

Based on the contributions and achievements presented in this thesis, we propose the following research directions for future work:

- Further development of our offloading architecture to fully integrate H.264 video streaming over RTP in a complete end-to-end XR offloading application. This will involve finalizing the tools needed to include this functionality in a commercial device deployment.
- Perform subjective experiments using our fully developed end-to-end architecture, similar to the ones proposed in Chapter 4, to understand the impact of different encoding schemes and protocols on the overall XR experience in relation to network quality of service. The aim is to allow the application to adapt dynamically based on measured network KPIs.
- Test our architecture in a more complex end-to-end offloading scenario with multiple peers and offloading services. While the proposed offloading architecture is capable of handling simultaneous peers subscribing or publishing to different services, we have not yet explored techniques for optimizing and managing shared resources on the server or MECs side. Therefore, our goal is to investigate methods for allocating computing resources and managing multiple offloading requests on the MECs.
- Study potential techniques for optimizing XR offloading performance using 5QIs to specify the desired level of service quality. We also aim to dynamically adapt these 5QIs based on the received network KPIs and the measured QoE on the application side. This is a complex task requiring the synchronized orchestration of multiple agents across the end to end solution. The goal is to enable the network to dynamically allocate resources in order to ensure a minimum overall QoE.
- Continuously develop FikoRE, our 5G RAN emulator, with a focus on scalability and the inclusion of advanced features such as handover and network slicing. We are interested in studying, implementing, and testing novel resource allocation algorithms specifically designed for XR offloading and other XR-related applications, such as the metaverse.

Bibliography

[1] M. Speicher, B. D. Hall, and M. Nebeling, "What is mixed reality?", in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19, Glasgow, Scotland Uk: Association for Computing Machinery, 2019, 1–15. DOI: 10.1145/3290605.3300767. [Online]. Available: https://doi.org/10.1145/3290605.3300767.

[2] A. Villegas, P. Pérez, and E. González-Sosa, "Towards a distributed reality: A multi-video approach to xr", in *Proceedings of the 11th ACM Workshop on Immersive Mixed and Virtual Environment Systems*, ser. MMVE '19, Amherst, Massachusetts: Association for Computing Machinery, 2019, 34–36. DOI: 10.1145/3304113.3326111. [Online]. Available: https://doi.org/10.1145/3304113.3326111.

[3] F. Zheng *et al.*, "Minimizing latency for augmented reality displays: Frames considered harmful", in *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2014, pp. 195–200. DOI: 10.1109/ISMAR.2014.6948427.

[4] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network", *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[5] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for mec", in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, 2018, pp. 1–6.

[6] A. A. Al-Habob, A. Ibrahim, O. A. Dobre, and A. G. Armada, "Collision-free sequential task offloading for mobile edge computing", *IEEE Communications Letters*, vol. 24, no. 1, pp. 71–75, 2020.

[7] C. Liu, M. Bennis, and H. V. Poor, "Latency and reliability-aware task offloading and resource allocation for mobile edge computing", in *2017 IEEE Globecom Workshops (GC Wkshps)*, 2017, pp. 1–7.

[8] J. Park and M. Bennis, "Urllc-embb slicing to support vr multimodal perceptions over wireless cellular systems", in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–7.

[9] E. Bastug, M. Bennis, M. Medard, and M. Debbah, "Toward interconnected virtual reality: Opportunities, challenges, and enablers", *IEEE Communications Magazine*, vol. 55, no. 6, pp. 110–117, 2017.

[10] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, "Toward low-latency and ultra-reliable virtual reality", *IEEE Network*, vol. 32, no. 2, pp. 78–84, 2018. DOI: 10.1109/MNET.2018.1700268.

[11] H. Kato, T. Kobayashi, M. Sugano, and S. Naito, "Split rendering of the transparent channel for cloud ar", in *2021 IEEE 23rd International Workshop on Multimedia Signal Processing (MMSP)*, 2021, pp. 1–6. DOI: 10.1109/MMSP53017.2021.9733514.

[12] E. A. McManus, B. Bodenheimer, S. Streuber, S. de la Rosa, H. H. Bülthoff, and B. J. Mohler, "The influence of avatar (self and character) animations on distance estimation, object interaction and locomotion in immersive virtual environments", in *Proceedings of the ACM SIGGRAPH Symposium on Applied Perception in Graphics and Visualization*, ser. APGV '11, Toulouse, France: Association for Computing Machinery, 2011, 37–44.

[13] F. Mueller, D. Mehta, O. Sotnychenko, S. Sridhar, D. Casas, and C. Theobalt, "Real-time hand tracking under occlusion from an egocentric rgb-d sensor", in *The IEEE International Conference on Computer Vision (ICCV) Workshops*, 2017.

[14] J. Pedoeem and R. Huang, *Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers*, 2018. DOI: 10.48550/ARXIV.1811.05588. [Online]. Available: https://arxiv.org/abs/1811.05588.

[15] E. Gonzalez-Sosa, A. Gajic, D. Gonzalez-Morin, G. Robledo, P. Perez, and A. Villegas, "Real time egocentric segmentation for video-self avatar in mixed reality", 2022. [Online]. Available: https://arxiv.org/abs/2207.01296.

[16] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration", *ACM Trans. Graph.*, vol. 36, no. 4, 2017. DOI: 10.1145/3072959.3054739. [Online]. Available: https://doi.org/10.1145/3072959.3054739.

[17] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "Icnet for real-time semantic segmentation on high-resolution images", in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., Cham: Springer International Publishing, 2018, pp. 418–434.

[18] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "A survey on mobile augmented reality with 5g mobile edge computing: Architectures, applications, and technical aspects", *IEEE Communications Surveys and Tutorials*, vol. 23, no. 2, pp. 1160–1192, 2021. DOI: 10.1109/COMST.2021.3061981.

[19] J. Jung, J. Ha, S.-W. Lee, F. A. Rojas, and H. S. Yang, "Novel applications of vr: Efficient mobile ar technology using scalable recognition and tracking based on server-client model", *Comput. Graph.*, vol. 36, no. 3, 131–139, 2012. DOI: 10.1016/j.cag.2012.01.004. [Online]. Available: https://doi.org/10.1016/j.cag.2012.01.004.

[20] Z. Li, C. Wang, and R. Xu, "Computation offloading to save energy on handheld devices: A partition scheme", in *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, ser. CASES '01, Atlanta, Georgia, USA: Association for Computing Machinery, 2001, 238–246. DOI: 10.1145/502217.502257. [Online]. Available: https://doi.org/10.1145/502217.502257.

[21] Z. Ning *et al.*, "When deep reinforcement learning meets 5g-enabled vehicular networks: A distributed offloading framework for traffic big data", *IEEE Transactions on Industrial Informatics*, vol. 16, no. 2, pp. 1352–1361, 2020. DOI: 10.1109/TII.2019.2937079.

[22] M. Müller *et al.*, "Flexible multi-node simulation of cellular mobile communications: The vienna 5g system level simulator", *EURASIP Journal on Wireless Communications and Networking*, vol. 2018, Sep. 2018. DOI: 10.1186/s13638-018-1238-7.

[23] A. Virdis, N. Iardella, G. Stea, and D. Sabella, "Performance analysis of openairinterface system emulation", in *2015 3rd International Conference on Future Internet of Things and Cloud*, 2015, pp. 662–669. DOI: 10.1109/FiCloud.2015.77.

[24] G. Nardini, D. Sabella, G. Stea, P. Thakkar, and A. Virdis, "Simu5g–an omnet++ library for end-to-end performance evaluation of 5g networks", *IEEE Access*, vol. 8, pp. 181 176–181 191, 2020. DOI: 10.1109/ACCESS.2020.3028550.

[25] D. Gonzalez Morin, M. J. Lopez Morales, A. Perez Pablo Garcia Armada, and A. Villegas, *Fikore: 5g and beyond ran emulator for application level experimentation and prototyping*, 2022. [Online]. Available: http://arxiv.org/abs/2204.04290.

[26] Nokia, *Fikore: 5g-network-emulator*, [online]: https://github.com/nokia/5g-network-emulator, accessed Sep. 2022.

[27] 3GPP, "TR 38.838, Study on XR (Extended Reality) Evaluations for NR, V17.0.0", Tech. Rep., 2022.

[28] D. González Morín, P. Pérez, and A. García Armada, "Toward the distributed implementation of immersive augmented reality architectures on 5g networks", *IEEE Communications Magazine*, vol. 60, no. 2, pp. 46–52, 2022. DOI: 10.1109/MCOM.001.2100225.

[29] D. González Morín, A. García Armada, and P. Pérez, "Cutting the cord: Key performance indicators for the future of wireless virtual reality applications", in *2020 12th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, Online: IEEE, 2020, pp. 1–6. DOI: 10.1109/CSNDSP49049.2020.9249445.

[30] D. González Morín, M. J. López Morales, P. Pérez, and A. Villegas, "Tcp-based distributed offloading architecture for the future of untethered immersive experiences in wireless networks", in *ACM International Conference on Interactive Media Experiences*, ser. IMX '22, Aveiro, JB, Portugal: Association for Computing Machinery, 2022, 121–132. DOI: 10.1145/3505284.3529963. [Online]. Available: https://doi.org/10.1145/3505284.3529963.

[31] ITU-T, "H.264 : Advanced video coding for generic audiovisual services", Tech. Rep., 2021.

[32] RFC, "RFC 3550 - RTP: A Transport Protocol for Real-Time Applications", Tech. Rep., 2003.

[33] D. Gonzalez Morin, E. Gonzalez-Sosa, P. Perez, and A. Villegas, "Full body video-based self-avatars for mixed reality: From e2e system to user study", 2022. [Online]. Available: https://arxiv.org/abs/2208.12639.

[34] D. G. Morin *et al.*, "An extended reality offloading ip traffic dataset and models", 2023. [Online]. Available: https://arxiv.org/abs/2301.11217.

[35] Microsoft, *Hololens 2 - overview, features and specifications*, [online]: https://www.microsoft.com/en-us/hololens/hardware, accessed Sep. 2022.

[36] Varjo. (). Varjo-XR3 Pro product specifications, [Online]. Available: https://varjo.com/products/xr-3/ (visited on 12/07/2022).

[37] G. Wallace, "The jpeg still picture compression standard", *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992. DOI: 10.1109/30.125072.

[38] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots", in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2502–2509. DOI: 10.1109/ICRA.2018.8460664.

[39] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras", *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017. DOI: 10.1109/TRO.2017.2705103.

[40] M. Z. Zia *et al.*, "Comparative design space exploration of dense and semi-dense slam", in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1292–1299.

[41] F. Mueller *et al.*, "Ganerated hands for real-time 3d hand tracking from monocular rgb", in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[42] J. Taylor *et al.*, "Efficient and precise interactive hand tracking through joint, continuous optimization of pose and correspondences", *ACM Trans. Graph.*, vol. 35, no. 4, Jul. 2016.

[43] S. Sridhar, F. Mueller, A. Oulasvirta, and C. Theobalt, "Fast and robust hand tracking using detection-guided optimization", in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3213–3221.

[44] F. Argelaguet, L. Hoyet, M. Trico, and A. Lecuyer, "The role of interaction in virtual embodiment: Effects of the virtual hand representation", in *2016 IEEE Virtual Reality (VR)*, 2016, pp. 3–10.

[45] M. Roxas, T. Hori, T. Fukiage, Y. Okamoto, and T. Oishi, "Occlusion handling using semantic segmentation and visibility-based rendering for mixed reality", in *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*, 2018, pp. 1–8.

[46] A. K. Hebborn, N. Höhner, and S. Müller, "Occlusion matting: Realistic occlusion handling for augmented reality applications", in *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2017, pp. 62–71. DOI: 10.1109/ISMAR.2017.23.

[47] D. González Morín, E. Gonzalez-Sosa, P. Pérez, and A. Villegas, "Bringing real body as self-avatar into mixed reality: A gamified volcano experience", in *2022 IEEE Virtual Reality (VR)*, Online: IEEE, 2022, pp. 3–10.

[48] P. Perez *et al.*, "Immersive gastronomic experience with distributed reality", in *2019 IEEE 5th Workshop on Everyday Virtual Reality (WEVR)*, 2019, pp. 1–6.

[49] 3GPP, "NR; Physical layer procedures for control", 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.213, Apr. 2020, Version 16.1.0.

[50] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality", in *The 25th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '19, Los Cabos, Mexico: Association for Computing Machinery, 2019. DOI: 10.1145/3300061.3300116. [Online]. Available: https://doi.org/10.1145/3300061.3300116.

[51] 3GPP, "NR; User Equipment (UE) radio access capabilities", 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.306, Jan. 2020, Version 15.8.0.

[52] ——, "NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone", 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.101, Jul. 2018, Version 15.2.0.

[53] ——, "Study on management of Non-Public Networks (NPN)", 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 28.807, Jun. 2020, Version 16.1.2. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3616.

[54] R. Baldoni, M. Contenti, and A. Virgillito, "The evolution of publish/subscribe communication systems", in *Future Directions in Distributed Computing: Research and Position Papers*. Berlin, Heidelberg: Springer-Verlag, 2003, 137–141. DOI: 10.5555/1809315.1809344.

[55] J. McCarthy, "Multi-image photogrammetry as a practical tool for cultural heritage survey and community engagement", *Journal of Archaeological Science*, vol. 43, Mar. 2014. DOI: 10.1016/j.jas.2014.01.010.

[56] P. Sommer, F. Schellroth, M. Fischer, and J. Schlechtendahl, "Message-oriented middleware for industrial production systems", in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, Munich, Germany: IEEE, 2018, pp. 1217–1223. DOI: 10.1109/COASE.2018.8560493.

[57] M. S. Afaqui, E. G. Villegas, and E. López-Aguilera, "Ieee 802.11ax: Challenges and requirements for future high efficiency wifi", *IEEE Wireless Communications*, vol. 24, pp. 130–137, 2017. DOI: 10.1109/MWC.2016.1600089WC.

[58] H. C. S. Thom, "A note on the gamma distribution", *Monthly Weather Review*, vol. 86, no. 4, pp. 117 –122, 1958. DOI: 10.1175/1520-0493(1958)086<0117:ANOTGD>2.0.CO;2. [Online]. Available: https://journals.ametsoc.org/view/journals/mwre/86/4/1520-0493_1958_086_0117_anotgd_2_0_co_2.xml.

[59] S. Yasuda and H. Yoshida, "Prediction of round trip delay for wireless networks by a two-state model", in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, Barcelona, Spain: IEEE, 2018, pp. 1–6. DOI: 10.1109/WCNC.2018.8377039.

[60] R. Kwan, C. Leung, and J. Zhang, "Proportional fair multiuser scheduling in lte", *IEEE Signal Processing Letters*, vol. 16, no. 6, pp. 461–464, 2009. DOI: 10.1109/LSP.2009.2016449.

[61] E. Gonzalez-Sosa, P. Perez, R. Tolosana, R. Kachach, and A. Villegas, "Enhanced self-perception in mixed reality: Egocentric arm segmentation and database with automatic labeling", *IEEE Access*, vol. 8, pp. 146 887–146 900, 2020.

[62] B. Lok, S. Naik, M. Whitton, and F. P. Brooks, "Effects of handling real objects and self-avatar fidelity on cognitive task performance and sense of presence in virtual environments", *Presence*, vol. 12, no. 6, pp. 615–628, 2003.

[63] M. Slater and M. Usoh, "The influence of a virtual body on presence in immersive virtual environments", in *Proc. of VR*, 1993, pp. 34–42.

[64] E. Ebrahimi, L. S. Hartman, A. Robb, C. C. Pagano, and S. V. Babu, "Investigating the effects of anthropomorphic fidelity of self-avatars on near field depth perception in immersive virtual environments", in *2018 IEEE conference on virtual reality and 3D user interfaces (VR)*, IEEE, 2018, pp. 1–8.

[65] A. Steed, Y. Pan, F. Zisch, and W. Steptoe, "The impact of a self-avatar on cognitive load in immersive virtual reality", in *2016 IEEE Virtual Reality (VR)*, 2016, pp. 67–76.

[66] Y. Pan and A. Steed, "The impact of self-avatars on trust and collaboration in shared virtual environments", *PloS one*, vol. 12, no. 12, e0189078, 2017.

[67] R. Fribourg, F. Argelaguet, A. Lécuyer, and L. Hoyet, "Avatar and sense of embodiment: Studying the relative preference between appearance, control and point of view", *IEEE transactions on visualization and computer graphics*, vol. 26, no. 5, pp. 2062–2072, 2020.

[68] K. Kilteni, R. Groten, and M. Slater, "The sense of embodiment in virtual reality", *Presence: Teleoperators and Virtual Environments*, vol. 21, no. 4, pp. 373–387, 2012.

[69] T. J. Dodds, B. J. Mohler, and H. H. Bülthoff, "Talk to the virtual hands: Self-animated avatars improve communication in head-mounted display virtual environments", *PloS one*, vol. 6, no. 10, e25759, 2011.

[70] M. Gruosso, N. Capece, and U. Erra, "Exploring upper limb segmentation with deep learning for augmented virtuality", 2021.

[71] N. Ogawa, T. Narumi, H. Kuzuoka, and M. Hirose, "Do you feel like passing through walls?: Effect of self-avatar appearance on facilitating realistic behavior in virtual environments", in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–14.

[72] D. Dewez *et al.*, "Influence of personality traits and body awareness on the sense of embodiment in virtual reality", in *2019 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2019, pp. 123–134.

[73] M. Gonzalez-Franco *et al.*, "The rocketbox library and the utility of freely available rigged avatars", *Frontiers in virtual reality*, p. 20, 2020.

[74] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "Smpl: A skinned multi-person linear model", *ACM transactions on graphics (TOG)*, vol. 34, no. 6, pp. 1–16, 2015.

[75] A. Thaler *et al.*, "Visual perception and evaluation of photo-realistic self-avatars from 3d body scans in males and females", *Frontiers in ICT*, p. 18, 2018.

[76] N. Ogawa, T. Narumi, and M. Hirose, "Virtual hand realism affects object size perception in body-based scaling", in *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, IEEE, 2019, pp. 519–528.

[77] G. Gorisse, O. Christmann, S. Houzangbe, and S. Richir, "From robot to virtual doppelganger: Impact of visual fidelity of avatars controlled in third-person perspective on embodiment and behavior in immersive virtual environments", *Frontiers in Robotics and AI*, vol. 6, p. 8, 2019.

[78] T. Waltemate, D. Gall, D. Roth, M. Botsch, and M. E. Latoschik, "The impact of avatar personalization and immersion on virtual body ownership, presence, and emotional response", *IEEE transactions on visualization and computer graphics*, vol. 24, no. 4, pp. 1643–1652, 2018.

[79] K. Yu, G. Gorbachev, U. Eck, F. Pankratz, N. Navab, and D. Roth, "Avatars for teleconsultation: Effects of avatar embodiment techniques on user perception in 3d asymmetric telepresence", *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 11, pp. 4129–4139, 2021.

[80] Y. Pan and A. Steed, "How foot tracking matters: The impact of an animated self-avatar on interaction, embodiment and presence in shared virtual environments", *Frontiers in Robotics and AI*, p. 104, 2019.

[81] S. Serra, R. Kachach, E. Gonzalez-Sosa, and A. Villegas, "Natural user interfaces for mixed reality: Controlling virtual objects with your real hands", in *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, IEEE, 2020, pp. 712–713.

[82] M. Bonfert, S. Lemke, R. Porzel, and R. Malaka, "Kicking in virtual reality: The influence of foot visibility on the shooting experience and accuracy", in *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, IEEE, 2022, pp. 711–718.

[83] L. L. Bozgeyikli and E. Bozgeyikli, "Tangiball: Foot-enabled embodied tangible interaction with a ball in virtual reality", in *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, IEEE, 2022, pp. 812–820.

[84] C. Xu, J. He, X. Zhang, X. Zhou, and S. Duan, "Towards human motion tracking: Multi-sensory imu/toa fusion method and fundamental limits", *Electronics*, vol. 8, no. 2, p. 142, 2019.

[85] L. Jayaraj, J. Wood, and M. Gibson, "Improving the immersion in virtual reality with real-time avatar and haptic feedback in a cricket simulation", in *2017 IEEE international symposium on mixed and augmented reality (ISMAR-adjunct)*, IEEE, 2017, pp. 310–314.

[86] M. Gonzalez-Franco *et al.*, "Movebox: Democratizing mocap for the microsoft rocketbox avatar library", in *2020 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, IEEE, 2020, pp. 91–98.

[87] G. Bruder, F. Steinicke, K. Rothaus, and K. Hinrichs, "Enhancing presence in head-mounted display environments by visual body feedback using head-mounted cameras", in *2009 International Conference on CyberWorlds*, 2009, pp. 43–50.

[88] P. Pigny and L. Dominjon, "Using cnns for users segmentation in video see-through augmented virtuality", in *2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, 2019, pp. 229–2295.

[89] L. P. Fiore and V. Interrante, "Towards achieving robust video selfavatars under flexible environment conditions", *International Journal of VR*, vol. 11, no. 3, pp. 33–41, 2012.

[90] T. Günther, I. S Franke, and R. Groh, "Aughanded virtuality-the hands in the virtual environment", in *Proc. of IEEE 3DUI*, 2015, pp. 157–158.

[91] M. Rauter, C. Abseher, and M. Safar, "Augmenting virtual reality with near real world objects", in *Proc. IEEE VR*, 2019, pp. 1134–1135.

[92] G. A. Lee, J. Chen, M. Billinghurst, and R. Lindeman, "Enhancing immersive cinematic experience with augmented virtuality", in *IEEE International Symposium on Mixed and Augmented Reality*, 2016, pp. 115–116.

[93] G. Alaee, A. P. Deasi, L. Pena-Castillo, E. Brown, and O. Meruvia-Pastor, "A user study on augmented virtuality using depth sensing cameras for near-range awareness in immersive vr", in *IEEE VR's 4th Workshop on Everyday Virtual Reality (WEVR 2018)*, vol. 10, 2018, p. 3.

[94] W. Xiang, H. Mao, and V. Athitsos, "Thundernet: A turbo unified network for real-time semantic segmentation", in *2019 IEEE winter conference on applications of computer vision (WACV)*, IEEE, 2019, pp. 1789–1796.

[95] D. González Morín, F. Pereira, E. González, P. Pérez, and A. Villegas, "Democratic video pass-through for commercial virtual reality devices", in *2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, IEEE, 2022, pp. 790–791.

[96] K. Tossel. (). libuvc: a Cross-Platform Library for USB Video Devices. (2021, Dec), [Online]. Available: https://ken.tossell.net/libuvc/doc/.

[97] Z. Zhang, "A flexible new technique for camera calibration", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[98] D. C. Brown, "Close-range camera calibration", *Photogramm. Eng.*, vol. 37, Dec. 2002.

[99] F. M. Mirzaei and S. I. Roumeliotis, "A kalman filter-based algorithm for imu-camera calibration: Observability analysis and performance evaluation", *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1143–1156, 2008.

[100] S. Garrido-Jurado *et al.*, "Automatic generation and detection of highly reliable fiducial markers under occlusion", *Pattern Recognition*, vol. 47, no. 6, pp. 2280 –2292, 2014.

[101] Y. Guo, Y. Liu, T. Georgiou, and M. S. Lew, "A review of semantic segmentation using deep neural networks", *International journal of multimedia information retrieval*, vol. 7, no. 2, pp. 87–93, 2018.

[102] G. F. Riley and T. R. Henderson, "The ns-3 network simulator", in *Modeling and Tools for Network Simulation*, K. Wehrle, M. Güneş, and J. Gross, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34. DOI: 10.1007/978-3-642-12331-3_2. [Online]. Available: https://doi.org/10.1007/978-3-642-12331-3_2.

[103] Matlab, *5g toolbox*, [online]: https://www.mathworks.com/products/5g.html, accessed Sep. 2022.

[104] N. Patriciello, S. Lagen, B. Bojovic, and L. Giupponi, "An E2E simulator for 5G NR networks", *Simulation Modelling Practice and Theory*, vol. 96, p. 101 933, 2019. DOI: https://doi.org/10.1016/j.simpat.2019.101933. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X19300589.

[105] T. Molloy, Z. Yuan, and G.-M. Muntean, "Real time emulation of an lte network using ns-3", in *25th IET Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, 2014, pp. 251–257. DOI: 10.1049/cp.2014.0694.

[106] P. Networks, *Nettest 5g network emulator*, [online]: https://www.polarisnetworks.net/5g-network-emulators.html, accessed Sep. 2022.

[107] Keysight, *5g network emulation solutions*, [online]: https://www.keysight.com/zz/en/solutions/5g/5g-network-emulation-solutions.html, accessed Sep. 2022.

[108] 3GPP, "NR; Physical layer procedures for data", 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.214, Jan. 2022, Version 17.0.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3216.

[109] ——, "NR; Physical channels and modulation", 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.211, Jan. 2022, Version 17.0.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3213.

[110] ——, "Study on channel model for frequencies from 0.5 to 100 GHz", 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.901, Jan. 2022, Version 16.1.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3173.

[111] Netfilter, *Netfilter queues*, 2022. [Online]. Available: https://netfilter.org/projects/libnetfilter_queue/.

[112] F. Capozzi, G. Piro, L. Grieco, G. Boggia, and P. Camarda, "Downlink packet scheduling in lte cellular networks: Key design issues and a survey", *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 678–700, 2013. DOI: 10.1109/SURV.2012.060912.00100.

[113] C. Mehlfuehrer *et al.*, "The vienna lte simulators-enabling reproducibility in wireless communications research", *EURASIP Journal on Advances in Signal Processing*, vol. 2011, 1–14, Dec. 2011. DOI: 10.1186/1687-6180-2011-29.

[114] P. Carballeira *et al.*, "Fvv live: A real-time free-viewpoint video system with consumer electronics hardware", *IEEE Transactions on Multimedia*, vol. 24, pp. 2378–2391, 2022. DOI: 10.1109/TMM.2021.3079711.

[115] P. Pérez *et al.*, "Live free-viewpoint video in immersive media production over 5g networks", *IEEE Transactions on Broadcasting*, vol. 68, no. 2, pp. 439–450, 2022. DOI: 10.1109/TBC.2022.3154612.

[116] J. Navarro-Ortiz, P. Romero-Diaz, S. Sendra, P. Ameigeiras, J. J. Ramos-Munoz, and J. M. Lopez-Soler, "A survey on 5G usage scenarios and traffic models", *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 905–929, 2020.

[117] A. Osseiran *et al.*, "Scenarios for 5G mobile and wireless communications: the vision of the METIS project", *IEEE communications magazine*, vol. 52, no. 5, pp. 26–35, 2014.

[118] (2016). Air interface framework and specification of system level simulations. FANTASTIC-5G, Deliverable D2.1, [Online]. Available: http://fantastic5g.com/wp-content/uploads/2016/06/FANTASTIC-5G_D21_public.pdf.

[119] P. Schulz, A. Traßl, N. Schwarzenberg, and G. Fettweis, "Analysis and Modeling of Downlink Traffic in Cloud-Rendering Architectures for Augmented Reality", in *2021 IEEE 4th 5G World Forum (5GWF)*, IEEE, 2021, pp. 188–193.

[120] M. Lecci, M. Drago, A. Zanella, and M. Zorzi, "An Open Framework for Analyzing and Modeling XR Network Traffic", *IEEE Access*, vol. 9, pp. 129 782–129 795, 2021.

[121] M. Lecci, A. Zanella, and M. Zorzi, "An ns-3 implementation of a bursty traffic framework for virtual reality sources", in *Proceedings of the Workshop on ns-3*, 2021, pp. 73–80.

[122] B. Bojovic and S. Lagen, "Enabling ngmn mixed traffic models for ns-3", in *Proceedings of the 2022 Workshop on ns-3*, 2022, pp. 127–134.

[123] J. M. P. van Waveren, "The Asynchronous Time Warp for Virtual Reality on Consumer Hardware", in *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, ser. VRST '16, Munich, Germany: Association for Computing Machinery, 2016, 37–46. DOI: 10.1145/2993369.2993375. [Online]. Available: https://doi.org/10.1145/2993369.2993375.

[124] P. Virtanen *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python", *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

[125] F. J. M. Jr., "The kolmogorov-smirnov test for goodness of fit", *Journal of the American Statistical Association*, vol. 46, no. 253, pp. 68–78, 1951. DOI: 10.1080/01621459.1951.10500769.

[126] N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika*, vol. 36, no. 1/2, pp. 149–176, 1949. [Online]. Available: http://www.jstor.org/stable/2332539 (visited on 12/19/2022).

[127] R. Porat, M. Fischer, S. Merlin, *et al.*, *11ax Evaluation Methodology*, https://mentor.ieee.org/802.11/dcn/14/11-14-0571-12-00ax-evaluation-methodology.docx, 2016.

[128] F. Capozzi, G. Piro, L. Grieco, G. Boggia, and P. Camarda, "Downlink Packet Scheduling in LTE Cellular Networks: Key Design Issues and a Survey", *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 678–700, 2013. DOI: 10.1109/SURV.2012.060912.00100.