Seamless Immersion: The Crucial Role of E2E Latency in 6DoF VR Content Delivery

Inaugural Dissertation
of the Faculty of Science,
University of Bern

presented by

Alisson Patrick Medeiros de Lima

from João Pessoa, Brazil

Supervisor

Prof. Dr. Torsten Braun
Institute of Computer Science
Faculty of Science of the University of Bern, Switzerland

Seamless Immersion: The Crucial Role of E2E Latency in 6DoF VR Content Delivery

Inaugural Dissertation
of the Faculty of Science,
University of Bern

presented by

Alisson Patrick Medeiros de Lima

from João Pessoa, Brazil

Supervisor

Prof. Dr. Torsten Braun
Institute of Computer Science
Faculty of Science of the University of Bern, Switzerland

Accepted by the Faculty of Science.

Bern, February 2024

The Dean:
Dr. Marco Herwegh

Abstract

The fifth generation (5G) and sixth generation (6G) mobile communication systems are envisioned to support several low latency applications, such as Virtual Reality (VR), through Ultra-Reliable Low-Latency Communications (URLLC). This thesis provides significant contributions to address the challenges of low latency communications and processing for VR applications. The contributions are categorized into three research areas. First, we investigate a resource provisioning mechanism that guarantees resource availability and prioritization for real-time VR services. Second, we develop an edge framework to orchestrate VR services through offloading and migration strategies considering the requirements of Six Degrees of Freedom (6DoF) VR applications. Lastly, we propose a novel network routing strategy, which optimizes the latency performance for 6DoF VR applications by calculating paths based on their latency requirements.

In the first contribution, we show that our solution provides several improvements compared to state-of-the-art solutions, in which we enhance the resource provisioning requests for *high-priority* VR services, reduce the amount of over-provisioning resources in edge servers, and reduce the overall service outages whenever edge resources become unavailable. In the second contribution, we demonstrate that our solution outperforms widely adopted mechanisms for service migration to reduce End-to-end (E2E) latency in exchange for a moderate increment in power consumption. We also show significant gains in selecting higher video resolutions for 6DoF VR applications based on E2E latency while providing more accepted context migrations. Finally, in the third contribution, we implement and compare state-of-the-art routing algorithms against our proposal. We consider the Key Performance Indicators (KPIs) flow network latency, path latency, over-provisioned latency, E2E latency, flow network throughput, frame rate, video resolutions, and execution time.

The proposed approaches in this thesis hold significant importance for the future of VR applications, particularly in reducing latency and enhancing user experiences. With the growing demand for immersive and interactive VR content, achieving ultra-low latency becomes critical. Therefore, the dynamic resource provisioning mechanism, the VR service orchestration, and the novel network routing strategy developed in this thesis contribute to overcoming the technical limitations of VR systems and maximizing the potential of edge computing. The findings presented in this thesis offer insights for network operators, service providers, and researchers seeking to advance the state-of-the-art in low latency VR applications.

List of Publications

Paper [1]

A. Medeiros, T. Braun, A. Di Maio, and A. Neto, "REACT: A Solidarity-based Elastic Service Resource Reallocation Strategy for Multi-access Edge Computing," *Physical Communication*, p. 101 380, 2021.

Paper [2]

A. Medeiros, A. Di Maio, T. Braun, and A. Neto, "Service Chaining Graph: Latency-and Energy-aware Mobile VR Deployment over MEC Infrastructures," in *Global Communications Conference*, IEEE, 2022, pp. 6133–6138.

Paper [3]

A. Medeiros, A. Di Maio, T. Braun, and A. Neto, "TENET: Adaptive Service Chain Orchestrator for MEC-enabled Low-latency 6DoF Virtual Reality," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, 2023.

Paper [4]

A. Medeiros, A. Di Maio, and T. Braun, "FLATWISE: Flow Latency and Throughput Aware Sensitive Routing for 6DoF VR over SDN," *IEEE Transactions on Network and Service Management*, 2023. ***Submitted***

List of Figures

List of Tables

List of Acronyms

3DoF Three Degrees of Freedom.

6DoF Six Degrees of Freedom.

AR Augmented Reality.

ATW Asynchronous TimeWarp.

AVS Automotive Video Streaming.

CDN Content Delivery Network.

CPU Central Processing Unit.

DO DSCP-Optimal.

DSCP Distributed Service Chain Problem.

E2E end-to-end.

FLATWISE Flow Latency and Throughput Aware Sensitive Routing.

FoV Field of View.

FPS Frames per Second.

GPU Graphics Processing Unit.

HMD Head-Mounted Display.

JFA Joint Flow Allocation.

KPI Key Performance Indicator.

MEC Multi-access Edge Computing.

MTP Motion to Photon.

MVR Mobile Virtual Reality.

NFV Network Function Virtualization.

OOM Out of Memory.

PoP Point of Presence.

QoE Quality of Experience.

QoS Quality of Service.

RAN Radio Access Network.

REACT MEC-suppoRted sElf-adaptive elAstiCiTy.

RTT Round-Trip Time.

SCG Service Chaining Graph.

SDN Software-Defined Networking.

SFC Service Function Chaining.

SLA Service-Level Agreement.

SP Shortest Path.

SWP Shortest-Widest Path.

TENET disTributed sErvice chaiN orchEstraTor.

URLLC Ultra-Reliable Low-Latency Communications.

VM Virtual Machine.

VNF Virtual Network Function.

VPA Vertical Pod Autoscaler.

VR Virtual Reality.

WP Widest Path.

WSP Widest-Shortest Path.

Contents

Chapter 1

Introduction

1.1 Overview

The next generation of Virtual Reality (VR) systems is set to undergo a significant transformation with the emergence of new video technology, such as Six Degrees of Freedom (6DoF), which allows users movement in 3-dimensional space, considering head position, head movement, and overall orientation to enable more immersive and interactive experiences [5]. 6DoF videos will feature higher resolutions and higher frame rates, requiring End-to-end (E2E) latency less than 5 ms for VR applications with strong interactions [6]. This advancement pushes the boundaries of visual quality and realism, requiring bitrates over 1 Gbit/s, where the network infrastructure must deliver ultra-low latency of less than 1 ms [7]. Beyond network optimization, the underlying edge infrastructure and VR Head-Mounted Displays (HMDs) are crucial in maintaining real-time responsiveness and preserving the sense of presence within the virtual environment [8]. Consequently, the widespread adoption of 6DoF VR applications is likely to impose a considerable network infrastructure and edge computing effort to meet future video technology demands, which can potentially introduce bottlenecks and congestion to the network infrastructure [9].

Today's Head-Mounted Displays (HMDs) are typically resource-constrained in terms of Central Processing Unit (CPU), Graphics Processing Unit (GPU), and energy. Deploying 6DoF VR applications requires multiple parallel decoders to speed up the decoding process. This underscores the need for efficient and optimized delivery mechanisms for 6DoF VR applications to ensure smooth and immersive experiences. Consequently, the computing power demands of 6DoF VR applications are significant, and it is limited to edge streaming scenarios where high computational resources are available. Therefore, this thesis investigates how reducing the E2E latency enhances the content delivery to VR systems. We consider latency optimizations in network infrastructure, edge computing, and VR HMDs to achieve this goal. We develop new strategies to reduce the E2E latency for 6DoF VR applications, in which we consider the deployment prioritization of compute-intensive VR tasks in the network edge, optimizations on the placement of VR tasks during user mobility, and a new routing approach based on network and throughput requirements of 6DoF VR applications.

1.2 Motivation

The realization of the 5G and 6G architectures is guided by novel technologies and new trends in user demands for modern applications, such as tactile Internet, autonomous vehicles, immersive media services, eHealth, etc [10]. However, the constant evolution of multimedia services has imposed several challenges to support new video technologies, more advanced resolutions, and higher frame rates, where the user experience goes from single sense to five senses: vision, hearing, touch, taste, and smell. As a result, such applications require superior network capabilities in terms of scalability, bandwidth, and latency. The main challenges are related to the network infrastructure, which must support much higher bandwidth while offering ultra-low network latency [11]. Figure 1.1 shows the evolution of multimedia services in which applications have rapidly increased their bandwidth requirements from 1 Mbit/s to over 1 Gbit/s, with expectations of reaching over 1 Tbit/s in the future.

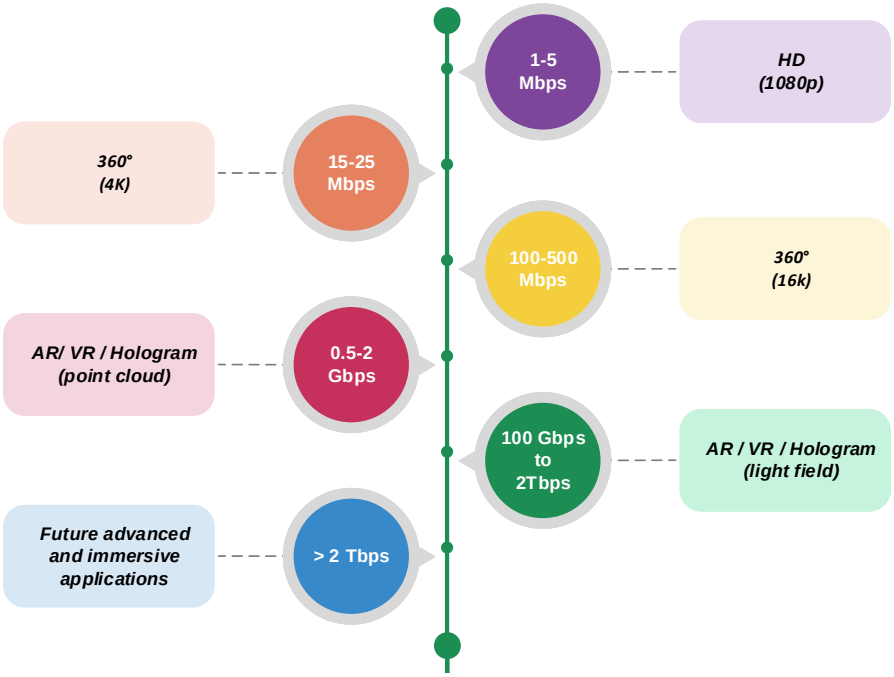


FIGURE 1.1: Multimedia Services Evolution.

To support those new applications, especially latency-sensitive, applications must offload their computing-intensive tasks (services) to edge computing [12]. *Edge computing* refers to the enabling technologies allowing computation to be performed at the edge network on downstream data on behalf of cloud services [13]. Besides, edge computing can potentially address the concerns of battery life constraints, bandwidth cost saving, and latency reduction for multimedia services. A *service* is defined as a virtualized instance of a physical function that is *cloudified* and offloaded to cloud hosts or network edges, e.g., video encoding, video transcoding, load balancing, content caching, network address translation, etc [14]. One of the main benefits is the reduction of the computational latency that can be reduced by using specialized edge infrastructures to process high-intensive computing tasks.

Resource Provisioning in Edge Computing Infrastructures

Modern immersive communication systems like VR demand extreme network and computing performance. Their quality depends significantly on the mobile network infrastructure's elasticity. *Resource elasticity* is defined as a system's ability to adapt to service workload fluctuations by adjusting resource configurations and provisioning close to the demand [15]. Therefore, elasticity strategies to support stringent and heterogeneous requirements imposed by current and upcoming 5G applications become essential to accelerate their adoption. Following this trend, telecom operators have adopted the *telco-cloud* paradigm to support on-demand edge computing resource elasticity [16]. Thus, they are broadly redefining their cloud infrastructures following the Multi-access Edge Computing (MEC) concept to achieve the requirements of 5G applications [17], [18].

MEC provides computing resources at the network edges, allowing telecom operators to fulfill latency requirements for future applications and offer service delivery at the edge of the mobile network [19]. One primary problem with MEC is its limited computing and communication resources [20], [21]. This may negatively affect the Quality of Service (QoS) for immersive systems in high-service demand situations, as network or MEC resources may become insufficient to support them [22]. To maintain satisfactory QoS in these circumstances, services typically migrate from overloaded to less-loaded MEC servers [23]. However, this approach requires service check-pointing and restarting for *stateful* services, which may lead to long service downtime if the migration process has to transfer a large amount of data [24].

When resources become scarce in MEC servers, the state-of-the-art elasticity mechanism will not meet the ideal resource allocation of the new service load. Hence, the elasticity mechanism triggers, in turn, the time-costly migration procedure, leading to the search for another cloud or edge server to deploy the target service. Although the migration meets the needed performance at another server, the resulting migration costs are too high, e.g., downtime and migration time, as the whole migration time is extremely time-consuming [14]. Optimal resource provisioning for MEC is an ongoing challenge [25]. On the other hand, many works in cloud computing propose new resource-elasticity strategies [26]. However, it is essential to develop elasticity strategies adapted to MEC since edge servers may run out of resources as service providers offer more computing resources for applications as consumer demand increases [27]. Developing resource management strategies tailored to MEC is essential to guarantee immersive communication systems' deployment.

State-of-the-art resource elasticity algorithms are reactive, where auto-scaling is started only after the service's resource usage crosses a predefined threshold. Some of the most popular reactive elasticity solutions, such as Amazon EC2, Microsoft Azure, and Google Cloud Platform, deploy heuristic auto-scaling schemes as reactive-based solutions meet cloud demands [28]. For the schemes mentioned above, after an elasticity request, the elasticity mechanism will fail to provide auto-scaling procedures when the requested resources are no longer available in a particular MEC server. As a result, the reactive model is likely to produce multiple attempts until it matches the resource configurations that suit the new

service load. We define the time needed for the auto-scaling procedure to converge and find a suitable resource allocation as *elasticity attempt window*. During the *elasticity attempt window*, the service will suffer from quality degradation due to resource saturation until matching optimal new resource patterns.

Due to the limited resource characteristics of MEC, its resources must be enhanced to support immersive communication system deployments through resource elasticity strategies that consider both MEC resource limitations and application requirements. Thus, we assume that over-provisioned resources must exist in virtualized MEC servers that support multi-tenancy, preventing virtual entities, e.g., containers and virtual machines, from being provisioned whenever their load changes. However, this will result in the over-provisioning of MEC resources and increase implementation costs. Based on this, new elasticity solutions tailored to MEC systems capable of overcoming resource scarcity and resource over-provisioning are needed to support the deployment of immersive systems. Among the various immersive systems, in this thesis, we consider VR as the primary use case, which serves as the basis for investigating the problems and solutions proposed in this thesis.

Virtual Reality, Head-Mounted Displays, and Six Degrees of Freedom Videos

VR systems artificially render a virtual environment with cognitive and sensorimotor characteristics, providing an advanced immersive reality through 6DoF videos to support both body and head motion, where the viewing direction and position can change [5]. Although VR systems have attracted considerable attention in recent years, it is infeasible to meet the requirements to support 6DoF videos by processing 6DoF videos on HMDs [9]. Implementing 6DoF VR is challenging because it requires multiple decoders operating under low latency and high bandwidth, leading to extreme computing power and high energy consumption on VR HMDs. Beyond those requirements, providing 6DoF VR becomes more challenging due to the VR interaction latency under the limited computation capability of HMDs. Thus, the massive adoption of 6DoF VR depends on the processing capability of HMDs to support unprecedented low latency and ultra-high throughput requirements. To overcome technical limitations, VR systems rely on URLLC [29].

A primary computing latency bottleneck arises because VR systems comprise multiple compute-intensive components (services), e.g., motion prediction, Field of View (FoV) prediction, hand tracking, encoding, and decoding, where some service inputs depend on the output of other services. In general, the required E2E latency is in the order of milliseconds. It has been pointed out that an E2E latency of more than 5 ms for advanced VR applications would lead to cybersickness [6], [30]. To put this challenge in perspective, a display running at 60 Hz, 90 Hz, and 120 Hz is updated every 16.67 ms, 11.11 ms, and 8.33 ms, respectively [31]. To overcome the technical limitations of VR systems, e.g., computing processing, specialized hardware platforms have been widely adopted in the field of VR to support the offloading of VR-intensive computing services from VR HMDs, aiming to achieve low latency and reduce energy consumption. This strategy significantly restricts VR applications by limiting the user's mobility range, particularly for *tethered* HMDs. Introducing wireless communications

in VR systems dramatically extends VR applications for mobile users, as it unleashes VR's true potential by enabling Mobile Virtual Reality (MVR), i.e., wireless HMDs, to provide user experience from anywhere at any time, where HMDs do not have wires to constantly recharge their batteries or transfer data for tasks processed on specialized hardware [32], [33].

However, wireless VR also raises several technical challenges to support MVR applications [34]. For example, wireless (*standalone*) HMDs must rely on a constrained onboard computing capability and limited energy supply for their operation merely by HMD processing [35]. Consequently, 6DoF VR applications are most likely restricted to *edge streaming* scenarios due to their high computing power demands [8], [20], [21]. Since it is impractical to use specialized hardware platforms to support VR use cases with high mobility features, e.g., VR-Automotive Video Streaming (AVS), MEC arises to support VR technical limitations by deploying computing and service delivery at the network edge to process VR-intensive computing services [20], [21]. However, coordinating such a plethora of VR services, especially during user mobility, yields several challenges.

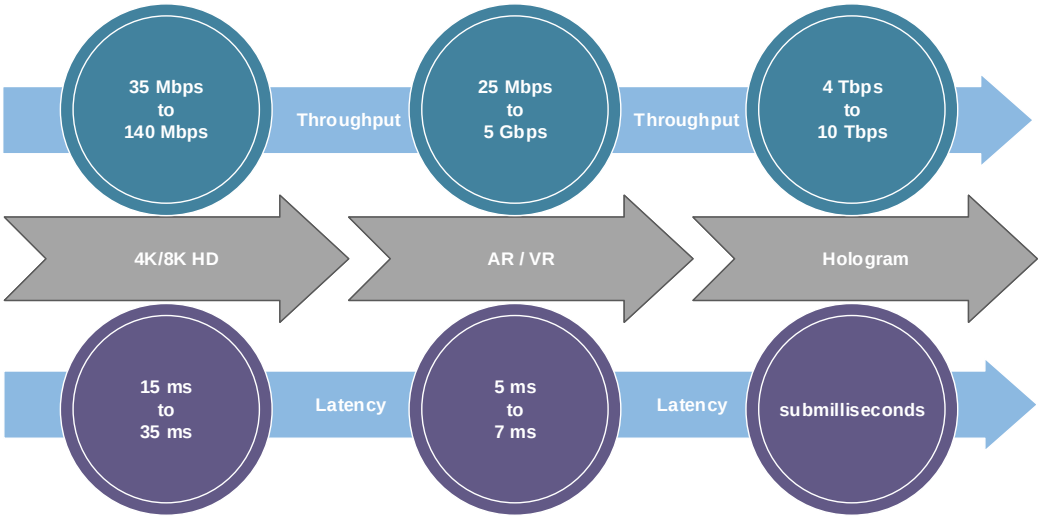


FIGURE 1.2: Throughput and Latency Requirements for new Multimedia Applications.

Network Throughput and Latency Challenges to Support Multimedia Applications

Although edge network helps to reduce the computational complexity of running VR applications, using edge networks is insufficient to meet the stringent requirements of 6DoF VR applications. Therefore, the network infrastructure must be optimized to handle the new demands of 6DoF VR applications. Even considering that extreme communication requirements, e.g., latency and throughput, will be achieved by 6G networks, the constrained computation, network latency, and energy consumption impose restrictions on processing 6DoF videos on VR HMDs [36]. Moreover, strong interaction for VR services, i.e., Cloud VR games, should have Round-Trip Time (RTT) at a maximum of 5 ms. These VR applications demand bandwidth from 1 Gbit/s to 5 Gbit/s [7]. For advanced VR immersion experiences, i.e., full-view 24 K 3D video, a RTT below 10 ms is required. Figure 1.2 shows the relationship between throughput and latency for new multimedia applications, such as VR, Augmented Reality (AR), and holograms, where throughput goes higher, whereas E2E latency falls lower.

To address the challenges by evolving VR video content, network optimization becomes a critical aspect of ensuring seamless transmission and user experience. To improve the current network infrastructure management, Software-Defined Networking (SDN) emerges as a new architecture that can provide highly flexible traffic routing and network resource management for immersive communications [37], [38]. Traditionally, Internet routing protocols prioritize network throughput over network latency since most video applications do not have stringent latency requirements [39].

Conventional routing protocols like IGRP and EIGRP consider factors such as throughput, latency, and network load when selecting paths within a network. Typically, they select paths with the highest throughput capacity while minimizing latency [40]. However, this strategy fails to effectively optimize overall application throughput usage while providing low latency for all VR users on the network. It overlooks how assigning a path to a specific flow affects the network latency for other flows during periods of network congestion, in which alternative paths prioritizing network throughput are often chosen, leading to increased network latency [41].

Hence, allocating paths from VR flows to a cloud server can significantly influence network congestion. Consequently, alternative paths are required to accommodate the unique demands of different VR applications. The coordination of edge processing, network latency reduction, and throughput insurance are crucial to achieve the lowest possible E2E latency for VR applications while satisfying VR application QoS requirements. However, managing a multitude of VR application flows, each with different deployment approaches, policies, and requirements, while guaranteeing E2E latency and throughput in a large-scale network scenario with congestion raises several challenges.

In this thesis, we primarily address the latency reduction for 6DoF VR and the challenge of supporting their deployment over edge networks while meeting their stringent latency requirements in the network infrastructure. Therefore, we design and evaluate solutions to minimize the E2E latency for 6DoF VR applications, where we consider optimizations in VR HMDs, the network edge infrastructure, and network infrastructure. To achieve this goal, we investigate different research areas such as dynamic resource provisioning and deployment prioritization of VR services over edge networks, the efficient VR service orchestration, offloading, and migration over edge networks, and the design and investigation of a new routing strategy to support ultra-low latency requirements for 6DoF VR in constrained SDN environments. First, we explore a novel edge computing resource provisioning approach to ensure the availability and prioritization of resources for real-time VR services deployed at the network edge. Second, we develop an edge framework to orchestrate VR services through offloading and migration strategies considering the requirements of 6DoF VR applications to minimize the overall E2E latency. Lastly, we propose a novel network routing strategy that optimizes the latency performance of 6DoF VR applications by approximating the latency of the calculated paths to the latency required by each VR application. The research areas addressed in this thesis, the research questions, and the contributions are described in the following sections.

1.3 Problem Statement

This thesis explores innovative strategies to achieve low-latency communications and processing for VR applications. It addresses the challenges posed by 6DoF VR systems, which require exceptionally low latency to ensure an immersive and responsive experience. The research encompasses three primary areas of investigation, each targeting specific aspects of achieving low latency for 6DoF VR systems. These areas are introduced in the following sections with concrete research questions that guide the study.

1.3.1 Resource Provisioning and Prioritization for VR over Edge Networks

With the growing demand for immersive and interactive VR experiences, edge computing has emerged as a promising solution to support VR-intensive computing tasks, which are offloaded from VR HMDs to the network edge to support the latency requirements of such applications while reduces the energy consumption on VR HMDs. However, supporting VR service deployment on edge networks while providing resource provisioning guarantees in a scenario with limited resources is challenging. Therefore, the problem of resource scarcity at the network edge can harm VR-intensive computing services that are sensitive to latency. This is because VR services require different resource provisioning patterns, and the edge infrastructure may only accommodate some of these services. To address this issue, the following research questions have been developed to provide a resource provisioning mechanism to support the deployment of VR tasks.

Research Question 1.1: How to design a resource provisioning mechanism for edge computing infrastructures to support the deployment of multiple heterogeneous VR services?

Research Question 1.2: How can the resource scarcity problem be effectively addressed in the coexistence of several VR services deployed on a shared edge infrastructure?

Research Question 1.3: How to prioritize the resource provisioning of VR intensive-computing tasks in a distributed edge infrastructure with limited computing resources?

1.3.2 Service Orchestration, Offloading, and Migration to Support 6DoF VR

To overcome the technical limitations of VR systems, e.g., computing processing, specialized hardware platforms have been widely adopted in the field of VR to support the offloading of VR-intensive computing services from VR HMDs, aiming to achieve low latency and reduce energy consumption. However, this strategy significantly restricts VR technology's application domain by limiting user's mobility range. Consequently, 6DoF VR is most likely restricted to *edge streaming* scenarios due to its high computing power demands, where VR services, e.g., encoding, decoding, are deployed on the network edge. However, coordinating such a plethora of VR services, especially during user mobility, yields several challenges, which are described through the following research questions.

Research Question 2.1: How can VR services be distributed across the MEC infrastructure to reduce the E2E latency of VR applications?

Research Question 2.2: What is the trade-off between the VR application's E2E latency and the mobile HMD's energy consumption by adopting different strategies for offloading VR-intensive computing services from mobile HMDs to MEC infrastructure?

Research Question 2.3: How does the decision on where VR services are deployed impact the E2E latency, and how does it affect the selection of video resolutions for VR systems?

1.3.3 Network Routing Strategy to Support Ultra-low Latency for 6DoF VR

To address the challenges posed by evolving VR video content, network optimization becomes a critical aspect of ensuring seamless transmission and user experience. Traditionally, Internet routing protocols were designed to prioritize network throughput over network latency because delivering the content was the primary goal. Thus, paths between video clients and servers are selected, considering throughput as the primary metric. However, this strategy fails to optimize overall application throughput while providing low latency for 6DoF VR applications. It overlooks how path assignments affect network latency for other flows during congestion, leading to increased network latency, and does not provide paths with throughput and latency guarantees. To address this issue, the following research questions have been considered to design and investigate a novel network routing strategy to support ultra-low latency requirements for 6DoF VR applications.

Research Question 3.1: How does the decision on the path of a 6DoF VR flow impact network latency and throughput for subsequent application flows in a large-scale network?

Research Question 3.2: How can overall network latency and throughput be optimized for applications deployed on the network?

Research Question 3.3: How does processing latency impact the decision on which path must be selected to fulfill the requirements of 6DoF VR applications?

1.4 Thesis Contributions

The contributions of this thesis are categorized into three parts associated with the three research areas mentioned above. The first part includes the work in **Paper [1]**, which addresses the research questions in research area 1.3.1. This part provides a dynamic resource provisioning mechanism to prioritize VR services with heterogeneous requirements deployed in edge infrastructures. The second part includes the works in **Paper [2]** and **Paper [3]**, which addresses the research questions in research area 1.3.2. In this part, we develop an edge framework to orchestrate VR services by proposing offloading and migration strategies that consider the requirements of 6DoF VR applications to minimize E2E latency. Finally, the third part includes the work in **Paper [4]**, which addresses the research questions in research area 1.3.3. This part proposes a new network routing strategy to support E2E latency requirements for 6DoF VR applications. The main works are summarized as follows.

1.4.1 Resource Provisioning and Prioritization for VR over Edge Networks

To answer the research questions in Section 1.3.1, in **Paper [1]**, a resource provisioning mechanism is presented to provide resource guarantees and prioritization to real-time VR services, which are offloaded and deployed in MEC infrastructures. However, to provide latency guarantees for high-priority VR services, their performance must be ensured when the edge infrastructure resources become scarce, where edge resources must be allocated for such services whenever needed, regardless of the resources available in each VR server. To address this problem, we present MEC-suppoRted sElf-adaptive elAstiCiTy (REACT), a mechanism that leverages resource provisioning among different VR services running on a shared MEC server. The main goal of REACT is to minimize the harmful effects of service migration while keeping more services running over the same MEC server.

REACT adopts an adaptive and solidarity-based strategy to redistribute resources from over-provisioned services to under-provisioned VR services over MEC infrastructures. REACT is an alternative strategy to avoid service migration due to resource scarcity. The key idea of the REACT is to prioritize resource provisioning for real-time VR applications, especially those sensitive to latency. With such prioritization, REACT enhances the performance of high-priority VR services, especially when the edge infrastructure resources become scarce. We design our resource provisioning approach considering that VR computing-intensive services, such as 6DoF VR services, take priority (*high-priority services*) over other services classified as *low-priority services*.

We consider a resource-constrained edge infrastructure to evaluate our approach against Kubernetes, a reactive algorithm baseline approach to provide resource provisioning in MEC servers. The considered edge infrastructure consists of interconnected MEC servers, each offering different computing and memory resources to a set of running services, each with distinct and specific resource requirements. Our evaluation assesses both REACT and Kubernetes' performance on a real testbed, in which the following KPIs are considered: elasticity events accomplishment (auto-scaling events), acceptance ratio of elasticity events, service outages, elasticity attempts (auto-scaling requests), residual resources, and response time. To assess their impact in handling elasticity events, both Kubernetes and REACT adopt the same elasticity approach to scale-up/down resources of MEC services.

Real testbed results show that REACT outperforms Kubernetes' elasticity approach by accomplishing up to 18% more elasticity events, reducing service outages by up to 95%, reducing elasticity attempts by up to 95%, reducing over-provisioned resources by up to 33%, 38%, and 73% for CPU cycles, RAM and bandwidth resources, respectively. Finally, REACT reduces response time by up to 15%. The results suggest that, in edge resource scarcity situations, REACT improves resource provisioning requests for *high-priority* VR services compared to Kubernetes, optimizes resource provisioning in edge computing infrastructures by reducing the amount of over-provisioning resources, and reduces the overall service outages whenever MEC resources become unavailable. A detailed description of this solution is presented in Chapter 3.

1.4.2 Service Orchestration, Offloading, and Migration to Support 6DoF VR

To answer the research questions in Section 1.3.2, in **Paper [2]** and **Paper [3]**, we propose disTributed sErvice chaiN orchEstraTor (TENET), a new edge orchestrator to refactor 6DoF VR applications into atomic services to increase the computing capacity of VR systems aiming to reduce the E2E latency of 6DoF VR applications. Those services are chained and deployed across HMDs and MEC servers in high mobility scenarios over real-edge network topologies. We provide algorithms for latency and energy trade-off, path calculation based on E2E latency, and management of VR applications to ensure acceptable E2E latency along with TENET architecture.

We also investigate the Distributed Service Chain Problem (DSCP) to find the optimal service placement of services from a service chain such that its E2E latency does not exceed 5 ms. DSCP problem is $\mathcal{NP}$-hard. We provide an integer linear program to model the system, along with a heuristic, TENET, which is one order of magnitude faster than optimally solving the DSCP problem. We show that DSCP implementation is unfeasible whenever there are too many VR users and network nodes. TENET supports offloading, migration, and orchestration of VR services deployed across HMDs and MECs to ensure acceptable E2E latency for MVR applications. Besides, TENET is developed according to an optimization problem that jointly minimizes latency and power consumption. TENET also optimizes the selection of better video resolutions for VR systems.

We evaluate the performance of Meta applications in terms of frame rate, computing latency, and energy consumption to model service workloads [42]. We provide an extensive evaluation in Meta HMD to model VR services with real workloads in a simulated environment to demonstrate the benefits of orchestrating VR services over MEC servers during user mobility. We use those application metrics to model 6DoF VR service workloads in a simulated environment to evaluate system scalability, E2E latency, power consumption, video resolution selection, context migrations, and execution time. We use a physical 5G network infrastructure map of Bern, Geneva, and Zurich. Based on those topologies, we model both network and computing latencies used in TENET simulation environment.

We compare TENET to DSCP implementation and well-known service migration algorithms in terms of E2E latency, power consumption, video resolution selection based on E2E latency, context migrations, and execution time. We discuss the importance of analyzing the latency and power consumption trade-off for VR systems. Besides, we show that TENET reduces E2E latency in exchange for power consumption over all cities compared to state-of-the-art migration algorithms. We also show that TENET deploys VR services in edge servers with more computing capacity, which provides more advanced video resolution, e.g., 8K, 4K, 1440p, and 1080p. We also demonstrate TENET accomplishes more accepted context service migrations using its deployment strategy on VR services. Finally, we show that TENET can process service placement as quickly as state-of-the-art migration algorithms. A detailed description of these solutions is presented in Chapter 4.

1.4.3 Network Routing Strategy to Support Ultra-low Latency for 6DoF VR

To answer the research questions in Section 1.3.3, in **Paper [4]**, we present a novel intra-domain routing algorithm with throughput guarantees for minimizing the overall E2E latency performance for all 6DoF VR applications deployed in SDN infrastructures. Flow Latency and Throughput Aware Sensitive Routing (FLATWISE) provides an adaptive routing approach that can squeeze or relax the path calculation based on the E2E latency requirement of 6DoF VR applications. FLATWISE approximates the E2E latency of the calculated path with the E2E latency required by each 6DoF VR application by analyzing the impact of path assignment on other 6DoF VR applications. It aims to overcome the limitations of current routing protocols by providing reliable E2E latency and throughput guarantees.

FLATWISE analyzes the impact of 6DoF VR E2E latency requirements on path selection decisions and the influence of path assignments on E2E latency for all VR applications deployed on the network. Besides, FLATWISE provides an adaptive routing approach, where it squeezes or relaxes the node selection in each iteration based on the achieved latency of the path and the target latency required by each 6DoF VR flow. Therefore, the primary feature of FLATWISE is to approximate the E2E latency of each calculated path to the E2E latency required by each 6DoF VR application. As a result, FLATWISE can establish a dynamic on-demand path calculation based on the E2E latency for each VR flow, where it can provide different paths between any source and destination according to the latency requirement, even with the same network conditions. FLATWISE also incorporates VR user location awareness, network load balancing, and congestion-aware routing to ensure E2E latency guarantees for 6DoF VR applications. By considering these features, FLATWISE prioritizes the search of paths based on E2E latency, ensuring seamless and immersive experiences for 6DoF VR applications in scenarios with network congestion.

We investigate the Joint Flow Allocation (JFA) problem to find paths for all flows in a network such that it determines the optimal path for each flow in terms of throughput and latency. The JFA problem is $\mathcal{NP}$-hard. We use Mixed Integer Linear programming to model the system, along with a heuristic, FLATWISE, which is one order of magnitude faster than optimally solving the JFA problem. We also provide algorithms for path allocation based on latency awareness and show the importance of considering E2E latency awareness in the path calculation process. We also show the practical implementation of FLATWISE compared to the Widest-Shortest Path (WSP) and Shortest-Widest Path (SWP) approaches.

We assess our proposed method's performance on a realistic simulated 5G network infrastructure map of the cities of Bern, Geneva, and Zurich. Based on those topologies, we model both network and computing latencies used in the FLATWISE simulation environment. Extensive simulations demonstrate that FLATWISE significantly reduces flow latency, over-provisioned latency, E2E latency, and algorithm execution time. Besides, FLATWISE improves flow throughput and frame rate compared to related work approaches. A detailed description of this solution is presented in Chapter 5.

1.5 Thesis Outline

This section provides a brief overview of the thesis structure. The rest of the thesis is structured as follows.

Chapter 2 provides a comprehensive literature review, discussing existing research on low-latency communications for VR applications. The theoretical background on 6DoF VR systems is presented, elaborating on their characteristics and challenges. Moreover, related research in network routing strategies, VR service orchestration, and resource provisioning for VR services deployed on edge networks are explored to establish a foundation for the subsequent chapters.

Chapter 3 introduces the resource provisioning mechanism REACT. This chapter explores how to provide resource guarantees and prioritization for real-time VR services deployed on MEC servers. The challenge of resource scarcity at the network edge is addressed. REACT's adaptive and solidarity-based strategy is discussed to show how to redistribute resources among different VR services efficiently. This chapter also presents real testbed results to demonstrate the superior performance of REACT compared to Kubernetes' elasticity approach, showcasing its benefits in reducing service outages, elasticity attempts, and over-provisioned resources while improving response time.

Chapter 4 focuses on developing an edge framework for VR service orchestration. The trade-off between the E2E latency of VR applications and the power consumption of mobile HMDs is analyzed. The DSCP is introduced to optimize service placement for reduced E2E latency. The formulation of the integer linear program for DSCP and the heuristic algorithm TENET is detailed, along with their effectiveness in reducing latency and optimizing VR service orchestration. The evaluation of TENET through simulations and comparison with other service migration algorithms is also presented.

Chapter 5 introduces the proposed network routing strategy to support ultra-low latency requirements for 6DoF VR applications. The JFA problem, which forms the basis of our new routing approach, is explained in detail. The formulation of the integer linear program for the JFA and the heuristic algorithm FLATWISE are presented, emphasizing their role in optimizing E2E latency performance for VR applications. The evaluation of FLATWISE through extensive simulations is discussed, and the results are compared against state-of-the-art routing algorithms to showcase its efficacy in reducing E2E latency for 6DoF VR applications.

Finally, Chapter 6 summarizes the thesis contributions, highlighting the key findings and achievements of each research area. The implications of the results are discussed, and the limitations of the proposed strategies are acknowledged. Areas for future research will be identified to enhance further low-latency communications for 6DoF VR applications, aiming to ensure the continuous development of innovative strategies in this field.

Chapter 2

Background and Related Works

2.1 Background

2.1.1 Edge Computing

Edge computing is a new paradigm that supports migrating computing tasks from remote cloud servers to local edge servers, performing data preprocessing and analysis near the data sources [43]. In such a scenario, MEC is an evolution of edge computing for mobile use. MEC is a new concept that provides computation, storage, and processing functions to the wireless network side [44]. MEC enables more mobile devices to quickly and efficiently process their tasks. The main goal of the MEC is to export cloud functions to the mobile network edge, increasing available bandwidth and reducing latency. Unlike the general architectural model, mobile hardware architecture is used more in communications, using multiple SDN controllers and virtualization to support data processing. Edge computing has the potential to be seamlessly integrated into multiple applications, products, and services, offering enhanced capabilities and improved efficiency. A few noteworthy examples of where edge computing can be leveraged include:

- ***Internet of Things*** devices, such as smart appliances and sensors, can benefit from edge computing. By executing code on the device itself rather than relying on cloud-based processing, these devices can offer more responsive and efficient user interactions.
- ***Self-Driving Cars***: Autonomous vehicles demand split-second decision-making capabilities. Edge computing allows them to process data and react in real-time without needing instructions from a distant cloud server.
- ***Video Conferencing***: High-bandwidth applications like interactive video conferencing can suffer from latency. By offloading decoders to the edge, the source of the video can be brought closer to the processing, reducing latency and enhancing the quality of the video.
- ***VR and AR Task Offloading***: AR and VR applications are data-intensive and demand low latency. Edge computing can offload processing tasks to the edge, minimizing latency and ensuring a more immersive and responsive experience for users of these technologies.

The shift toward edge computing advances how VR videos are processed and delivered, providing video decoding processing closer to VR users [45]. This proximity to the end-user brings advantages, further expanding the range of possible applications. In the context of VR and its associated use cases, achieving ultra-low latency and high bandwidth becomes crucial. These two factors are critical for delivering a seamless and immersive VR experience. Therefore, the stringent latency requirement of VR underscores the necessity of deploying private 5G networks and adopting edge computing solutions, as they are indispensable in guaranteeing the success of VR applications, where near-instantaneous data processing and transfer are essential for optimal user experiences [46].

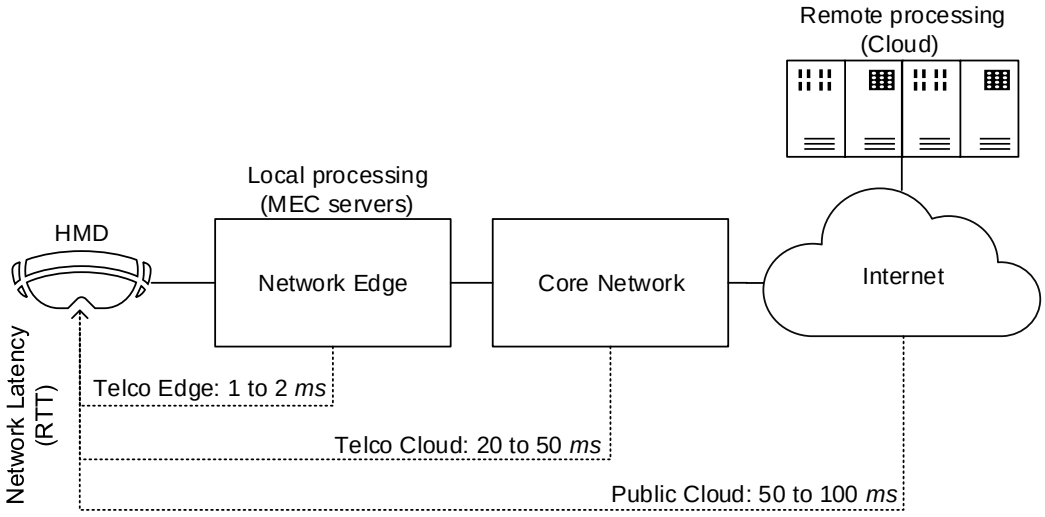


FIGURE 2.1: Cloud, Telco Cloud, and Telco Edge Latencies.

The latency plays a critical role in VR systems. In general, the required E2E latency is in the order of milliseconds [47]. The current latency scenario is based on the cloud computing environment and can vary between 50 ms and 100 ms. In a potential scenario, network operators through telco-cloud deployments (supported by MEC) will enable latency between 20 ms and 50 ms. *Telco cloud* is a highly resilient cloud infrastructure that allows telecom providers to deploy services more quickly and respond faster to changes in network demand. Currently, the main telco cloud approaches are Open RAN, O-RAN, and vRAN.

While the telco cloud offers reduced latency, it still needs to meet the latency demands of advanced VR applications. To achieve such ultra-low latency, telecom operators consider the telco edge, which will offer latency from 1 ms to 2 ms or even down to 1 ms. The *telco edge* is an initiative that promotes a federated solution in which applications and services are executed and processed at a location close to the end user, allowing service providers to provide high bandwidth at the edge and low latency services. Some authors argue that network latency of more than 5 ms would lead to cybersickness, an uncomfortable and nauseating customer experience for VR systems [6]. Therefore, most VR applications require latencies between what telco clouds and telco edge can deliver. However, advanced VR applications require ultra-low latency, which only telco edge can deliver. Figure 2.1 shows the network latency requirements of cloud, telco cloud, and telco edge architectures.

Virtualization

Virtualization has emerged as a pivotal technology in modern computing, revolutionizing how IT resources are utilized and managed. Virtualization is the main characteristic of cloud computing and edge computing approaches. In the dynamic landscape of cloud computing and edge computing, virtualization is crucial in optimizing infrastructure, enhancing scalability, and enabling efficient resource allocation. Therefore, virtualization involves creating virtual instances of computing resources such as servers, storage, and networks [48]. Its core principles include abstraction, isolation, and encapsulation, allowing multiple virtualized instances to run on a single physical server.

The main benefits of virtualization in cloud and edge computing are:

- ***Resource Optimization***: Efficient use of hardware resources, reducing costs.
- ***Scalability***: Easily scale resources up or down based on demand.
- ***Flexibility***: Support for diverse operating systems and applications.
- ***Disaster Recovery***: Quick recovery through snapshots and backup capabilities.

Service Function Chaining

Service Function Chaining (SFC) is a concept in networking that involves the definition and instantiation of an ordered set of network services to be applied to packets as they traverse a network [49]. It is a method of specifying a sequence of network services that should be applied to traffic flows in a network. Besides, SFC represents a network capability that facilitates application-driven networking by orchestrating the orderly connection of various service functions. The management of the lifecycle of these service function chains is made possible through the integration of two relatively recent technologies: SDN and Network Function Virtualization (NFV). These technologies offer the prospect of significant enhancements in terms of efficiency, effectiveness, and flexibility within network operations. The key components of a SFC architecture are described in the following.

- ***Service Function*** is a specific network capability or feature that can be applied to packets as they pass through a network. Examples of service functions include firewalls, intrusion detection systems, load balancers, and content filters.
- ***SFC*** is the ordered set or sequence of service functions that are to be applied to packets as they traverse the network. The service functions are linked together to form a chain, and traffic flows through these functions in the specified order.
- ***Classifier*** determines which service function chain a particular packet should follow based on defined policies. It examines packet headers or other attributes to make this decision.
- ***Service Function Forwarder*** is responsible for encapsulating and forwarding packets to the appropriate service functions based on the information provided by the classifier.
- ***Service Function Path*** represents the path through the network that a packet takes as it traverses the specified service functions.

SFC, coupled with the capabilities of virtualization, stands as a transformative technology in modern network architectures. The dynamic orchestration of network services through virtualized instances not only enhances efficiency and scalability but also paves the way for innovative applications in emerging immersive services, such as AR, VR, holograms, etc.

Serverless Computing

Serverless computing is a cloud and edge computing execution model where service providers automatically manage the infrastructure to execute and scale applications [50]. In a serverless architecture, developers focus on writing code for their application's functionality, and the service provider takes care of the underlying infrastructure, including server provisioning, maintenance, and scaling. The Key characteristics of serverless computing include:

- ***No Server Management***: Developers do not need to worry about server provisioning, scaling, or maintenance. The service provider handles these tasks automatically.
- ***Event-Driven***: Serverless applications are typically event-driven, meaning they respond to events and triggers, such as changes in data or HTTP requests.
- ***Automatic Scaling***: Serverless platforms scale applications automatically in response to the number of incoming requests or events.
- ***Pay-per-Use Pricing***: users are billed based on actual usage rather than pre-allocated resources. This can result in cost savings.

Besides, the relationship between serverless computing and SFC has the following characteristics:

- ***Decoupling of Services***: Both serverless computing and SFC aim to decouple specific functionalities or services from the underlying infrastructure. In serverless, this decoupling is achieved by abstracting away servers entirely, while SFC decouples and orders network services in a prescribed sequence.
- ***Event-Driven Architecture***: Serverless applications often follow an event-driven architecture, responding to triggers or events. Similarly, SFC is designed to facilitate the ordered application of network services based on specific events or policies, aligning with the dynamic nature of modern network requirements.
- ***Abstraction of Infrastructure***: Both serverless and SFC abstract away the complexity of managing underlying infrastructure. In serverless, developers are abstracted from servers, and in SFC, the network service chain abstracts the network functions from the underlying network infrastructure.

While serverless computing primarily addresses application deployment and execution in the cloud or edge, SFC focuses on orchestrating and managing network services within a network infrastructure. While they serve different domains, the shared context is the abstraction and automation of underlying resources to enhance flexibility and efficiency.

2.1.2 Virtual Reality

Virtual reality is a simulated 3D environment that enables users to explore and interact with virtual surroundings in a way that approximates reality [51]. The environment is created with computer hardware and software, where users might also need to wear helmets or goggles to interact with the environment, such as HMDs. As a result, VR systems have revolutionized how humans interact with machines, transitioning from traditional two-dimensional interfaces to more immersive and interactive three-dimensional paradigms.

VR systems are expected to completely change how we interact with the world through a new virtualized immersive environment. They play a critical role in cities' future, including education, transportation, health, entertainment, and media [11]. The leading VR goal is to bring unprecedented experiences, providing an immersive level where users will feel part of it [52]. Thus, advanced VR applications will offer interactions, visuals, and sounds so real that users will achieve high-quality immersion experiences.

Challenges for Enabling High-quality Experiences in VR Systems

Several research fields are being investigated to achieve such an immersion level, including sound quality, visual quality, and intuitive interactions. However, immersive media services, especially VR, are emerging technologies relying on URLLC [34]. In practice, VR technology provides a certain level of immersion because three main features are used simultaneously: visual, sound, and interactions. However, these three main features must be improved simultaneously to enable advanced immersive experiences for the next generation of VR systems. In particular, VR demands stringent requirements in terms of low latency.

VR will offer unprecedented experiences and possibilities, such as immersive movies and shows, live concerts, sports, and social interactions. Therefore, a uniform experience characterizes most VR applications, in which VR applications can be used anywhere and anytime, even in high mobility environments like cars, trains, or even head movement must be considered [53]. One critical aspect of these applications is that problems like lags, stutters, and stalls are unacceptable for user experience and comfort, especially during user mobility.

VR has posed several challenges to HMDs and the current network infrastructure in supporting ultra-high throughput and ultra-low latency [54]. These challenges will become dramatically challenging once VR applications become massively consumed. Thus, the massive adoption of VR relies on how the next generation of HMDs and mobile networks will achieve VR requirements. To overcome the requirements demanded by VR, the future generation of HMDs and mobile networks have to support unprecedented requirements for ultra-low latency and ultra-high throughput. In this scenario, VR E2E latency can be classified as computing and network latency. There are several sources of latency in VR systems. In particular, low latency responses are required for advanced VR videos that use 6DoF technology. Some latencies include sensor, cloud, or edge processing, network, requests overhead, buffering, rendering, or feedback latency.

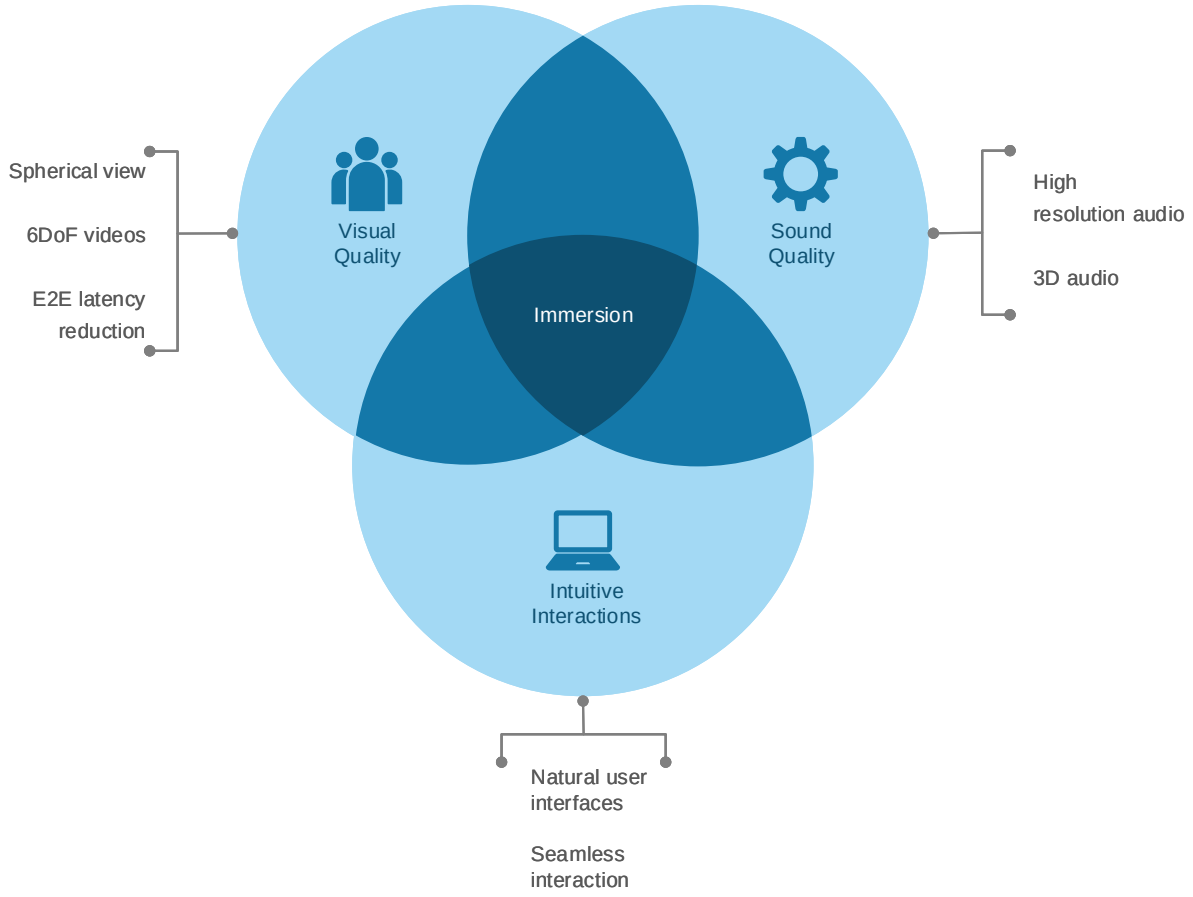


FIGURE 2.2: Extreme Requirements to Support VR Applications.

One of the main challenges is computing latency due to the hardware limitations VR HMDs [6]. The computing latency is critical because the entire VR application is affected if there is a lag in any VR task [31]. Another challenge is network latency because the network must ensure ultra-low latency while supporting high bandwidth guarantees for VR applications [55]. Therefore, supporting advanced VR applications through network infrastructure presents several challenges, as these applications require low latency, high bandwidth, and consistent connectivity to provide a seamless and immersive experience. Figure 2.2 shows the extreme requirements to support advanced VR applications.

360^o Video Content

These strict latency requirements relate to new video technologies, e.g., Three Degrees of Freedom (3DoF) or 6DoF, that VR systems must support to provide more advanced experiences [56]. Therefore, the user's perspective within a virtual environment can be conceptualized as a spatial sphere in a VR system. This conceptual sphere encompasses the entire visual experience, where the horizontal FoV spans 360 degrees while the vertical FoV covers 180 degrees. When a user engages with an HMD, each of their eyes captures a portion of the comprehensive spherical data, the extent of which is contingent on the specific FoV offered by the HMD being used. Unlike regular videos, spherical videos capture the entire panoramic view, enabling users to choose their viewing direction freely.

The potential for enhancing 360^o content lies in incorporating interactions, a concept called *interactive 360^o content*. Nevertheless, the range of possible interactions remains restricted, given that users cannot physically move their heads or hands within the virtual space. Instead, they are confined to a fixed viewpoint and can only utilize a specific pointer for interaction. This limitation draws a parallel to the diminished engagement experienced in interactive videos.

The immersive nature of 360^o video content provides a more engaging and interactive experience for viewers. It's often used in several applications, including virtual tours, marketing campaigns, documentaries, and storytelling experiences. Viewers can feel present in the environment or event captured in the video, enhancing the sense of immersion and allowing for a more personalized and dynamic viewing experience.

Advanced Video Transmission for VR Systems

The process of transmitting 360^o videos from cloud servers to HMDs involves the implementation of two approaches, namely *full-view transmission* and *FoV transmission*. These methods delineate the strategies for transmitting visual data across the network infrastructure to facilitate immersive VR experiences [57]. The choice between those methods depends on the specific use case, available technology, and the desired user experience. Both methods have advantages and limitations, and the importance of this distinction lies in tailoring the VR experience to meet the specific requirements and constraints of a given application or scenario.

The full-view transmission scheme involves the transmission of 360-degree images from cloud servers to user HMDs. In this approach, switching and rendering these images occurs locally on the user's device when users turn their heads within the virtual environment [58]. However, it is essential to note that VR imagery, despite maintaining the resolution of a single eye, carries significantly higher data rates than conventional 2D videos. This substantial increase in data requirements, typically five to ten times greater, is primarily attributed to several factors, including the need for a higher frame rate, greater bit depth, and the inherently panoramic nature of 360-degree content.

In the full-view transmission scheme, each data frame received by the user's HMD contains a comprehensive representation of the entire spatial sphere, encompassing the available visual content. When users interact with the virtual environment by altering their viewing angles, the corresponding interaction signals are processed locally on their HMDs [59]. Thus, HMDs decodes the specific FoV information necessary for the user's current perspective from the frames they have already loaded, restoring the relevant visual data to provide users with a seamless and natural viewing experience.

The core principle underlying the FoV transmission scheme revolves around prioritizing the high-quality transmission of the portion of the VR environment that lies within the user's immediate FoV [60]. This approach entails sending only the visual data relevant to the user's perspective, conserving bandwidth and promoting efficiency. It is worth noting that, at present, the specific technologies and standards for implementing the FoV transmission

scheme have not yet been defined or standardized, representing a realm where ongoing innovation and development continue to shape the landscape.

On the other hand, in the full-view transmission scheme, users often observe only a fraction of the complete spatial sphere, with the remainder effectively consuming network bandwidth without serving any purpose. This practice results in significant inefficiency, leading to the substantial squandering of valuable network resources. To address this particular concern, the industry has put forward a solution in the form of the FoV transmission scheme, which seeks to transmit VR images in a manner that aligns with the user's current FoV.

In content preparation, a full-view coding stream is generated with a deliberately non-uniform quality distribution across different sections of the FoV. The FoV is essentially divided into two distinct segments: (i) the region within the FoV itself, and (ii) the region lying outside of it. High-quality coding techniques are employed within the FoV area to ensure exceptional image quality, while lower-quality coding is applied outside the FoV. This approach optimizes the utilization of network resources and encoding efficiency.

Full-view transmission and FoV transmission come with their challenges, such as high bandwidth requirements, storage, processing, and ultra-low latency [61]. As a result, transmitting these media types requires considerable effort from both the network infrastructure and the HMD hardware [62]. Typically, The transmission of both full-view and FoV content from cloud servers to VR HMDs involves different phases, which are:

- ***Point of Presence (PoP) Servers***: PoP servers are strategically distributed across various geographical locations to reduce latency and improve content delivery. These PoP servers host the complete 360-degree VR content for full-view transmission. However, for FoV transmission, PoP servers store only the portion of the VR environment corresponding to the user's field of view.
- ***Segmentation***: VR content is divided into smaller segments or chunks to facilitate efficient streaming. These segments are often based on time intervals or spatial regions, e.g., in degrees. Given the nature of full-view transmission, each segment covers the entire 360-degree sphere. However, in FoV transmission, the segments focus only on what the user is looking at. This can be determined through user head-tracking data.
- ***Transcoding***: Segments are transcoded into different quality levels to adapt to network conditions and HMD capabilities. This step ensures that users with different HMD hardware and network conditions can access the content optimally.
- ***Packaging***: Transcoded segments are then packaged into a suitable format, such as HTTP Adaptive Streaming or MPEG-DASH, allowing dynamic quality adjustment and seamless playback.
- ***Content Delivery Network (CDN)***: The segmented, transcoded, and packaged content is distributed across a CDN network. CDNs store these segments on edge servers close to the VR users, reducing latency and improving delivery speed. Besides, considering the user's

location and head orientation, the CDN network ensures quick access to the FoV segments that can be fetched directly from edge servers.

- ***Decoding***: The HMD decodes the received segments from edge servers and combines them to recreate the 360-degree view. VR users can freely navigate and explore the VR environment and interact with other VR users. In FoV transmission, the HMD decodes and displays only the relevant FoV segments corresponding to the user's line of sight. As the user moves and adjusts their gaze, the content within the FoV dynamically updates to maintain a seamless experience.

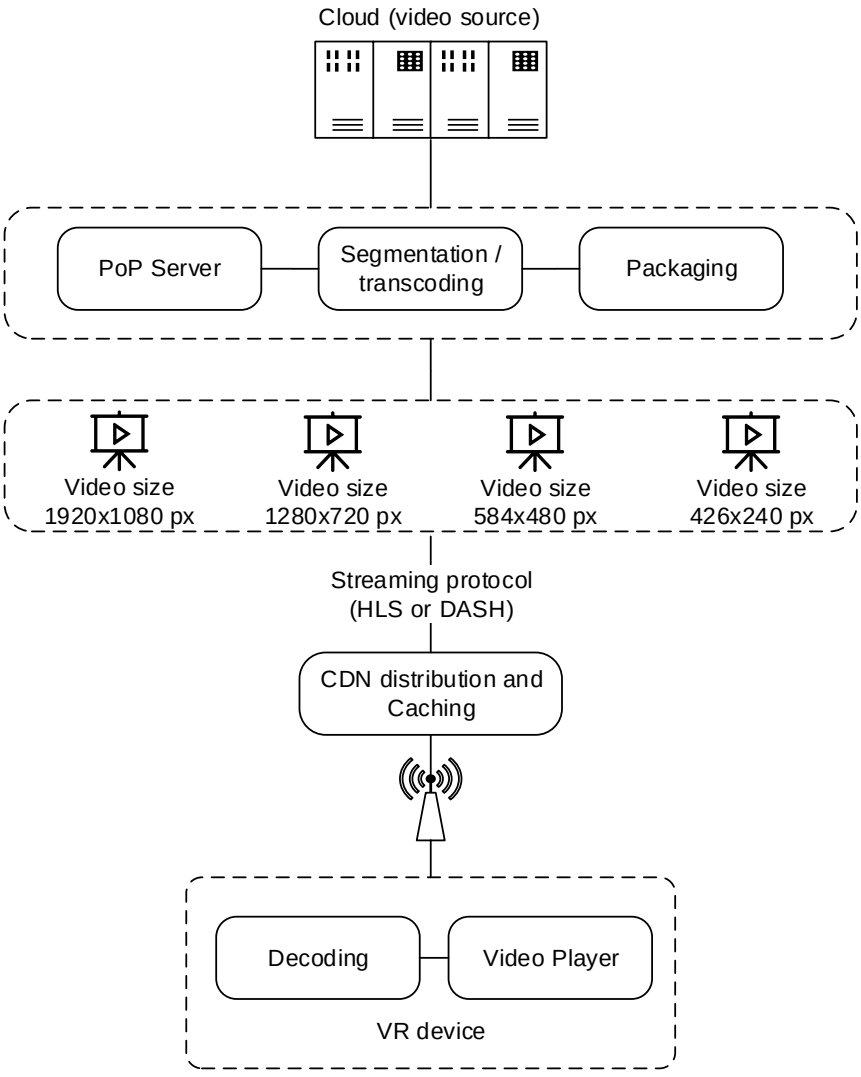


FIGURE 2.3: Full-view and FoV E2E Transmission Processing Pipeline for VR Content

Figure 2.3 shows the entire transmission pipeline for both full-view and FoV schemes. The key difference between full-view and FoV transmission lies in the segmentation and storage at PoP servers. Full-view transmission encompasses the entire VR environment, while FoV transmission selectively transmits and stores content relevant to the user's field of view, making it more bandwidth-efficient and responsive to user interactions.

2.1.3 Three and Six Degrees of Freedom

The immersive quality of VR can be characterized by the extent of degree of freedom it offers to users. The development of new video technology, e.g., 3DoF and 6DoF, provides a more immersive and interactive experience for VR systems. This transformative technology harnesses the power of omnidirectional content to transport VR users into fully immersive virtual environments [63]. Omnidirectional content is captured to cover the entire 360-degree FoV around the capturing device, effectively encapsulating the user's visual perspective [62]. This approach to content creation allows for the generation of 3DoF or 6DoF experience within the VR environment.

The immersive experience provided by VR can be characterized by the degrees of freedom that VR systems grant to their users. Essentially, the extent and variety of degrees of freedom available directly contribute to the richness and depth of the immersive encounter within the VR environment. Consequently, depending on the specific degrees of freedom used in the virtualized environment, the immersive experience can be segmented into distinct phases, each representing a unique facet of the user's interaction and engagement with the virtual environment. These phases are structured around the multifaceted movements and interactions users can perform, fundamentally influencing the nature and quality of the overall immersive experience in VR.

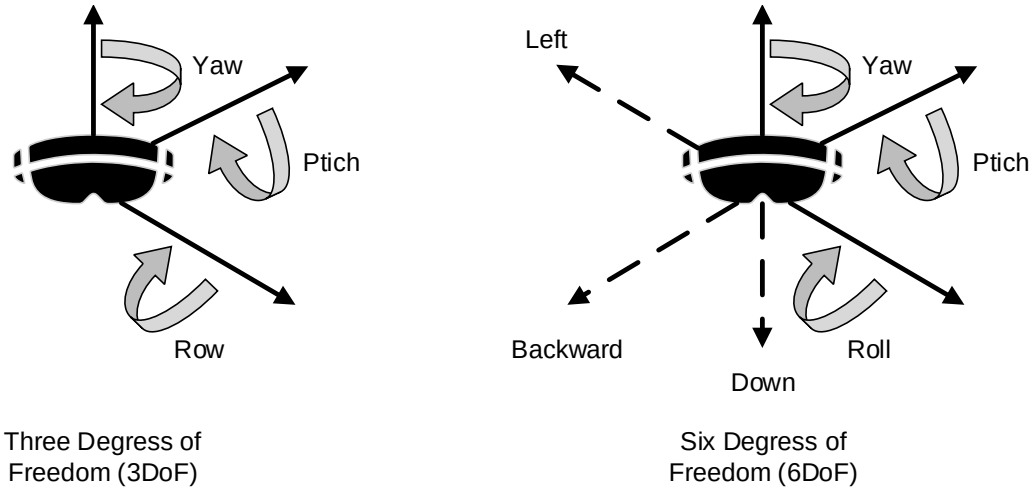


FIGURE 2.4: Viewing angle for 3DoF and 6DoF VR.

Figure 2.4 describes the standardization roadmap established by the *Moving Picture Experts Group* [64]. This roadmap subdivides the standardization process of VR into two phases, e.g., 3DoF and 6DoF, with each phase offering specific degrees of freedom within the VR domain. On the one hand, 3DoF refers to the three rotational axis, which allows turning left/right (yaw), looking up/down (pitch), and tilting the view (row). On the other hand, 6DoF refers to a fully immersive experience, providing six degrees of freedom. This includes three rotational degrees of freedom and three translational degrees of freedom, allowing users to not only rotate their viewpoint but also move within and engage with the virtual environment. The rotational movements supported by 6DoF are the following:

- ***Elevation*** provides vertical movement of a VR user, encompassing actions such as bending down or standing up. It enables users to change their viewing angle in the up and down dimension within the virtual environment.
- ***Strafe*** entails lateral movement, allowing users to shift to the left or right, effectively sidestepping within the virtual world. It enables users to modify their horizontal position within the environment.
- ***Surge*** corresponds to the forward and backward movement of a user, like walking within the virtual space. This feature facilitates dynamic transitions along the front-to-back axis.
- ***Rolling*** facilitates the swiveling movement of the head from side to side, enabling users to peer around a corner or inspect objects from different angles. It introduces a rotational aspect within the virtual world.
- ***Pitch*** provides the tilting motion of the head along a vertical axis, granting users the capacity to gaze upwards or downwards. This movement allows for changes in the vertical orientation of the viewpoint.
- ***Yaw*** supports the swiveling movement of the head along a horizontal axis, empowering users to look to the left or right and alter their horizontal perspective. It introduces a rotational aspect around the vertical axis.

Hence, through 3DoF or 6DoF, users can navigate and interact with this virtual environment, making head movements and exploring different perspectives within a specific range. However, achieving true immersion requires integrating high-resolution, high-quality, and high frame-rate stereoscopic omnidirectional video content within VR technology. As a result, HMDs have to withstand processing demands that were previously unimaginable for mobile devices. This results in a high demand for high-speed data processing and high energy consumption. Therefore, research evidence shows that it will only be possible to achieve advanced immersion with 6DoF through the use of pre-processing of VR content at the edge of the network [8], [9], [31].

The quality and resolution of the content are paramount, ensuring that the visual and audio cues effectively fool the senses and immerse users within the virtual space. Furthermore, maintaining a high frame rate is essential to deliver a smooth and responsive VR experience, avoiding motion sickness and providing a convincing and comfortable environment [6]. The demand for such exceptional quality underscores the importance of technological advancements in capturing, transmitting, and rendering omnidirectional content to realize immersive VR experiences fully.

However, the major barrier that prevents VR 6DoF from being transmitted over the network lies in the extremely high computational complexity, of which multi-view depth estimation and depth image-based rendering are challenging [62]. Furthermore, network bandwidth and latency are also challenging in a highly scalable scenario with several VR users consuming 6DoF content transmitted from cloud servers to VR HMDs [11].

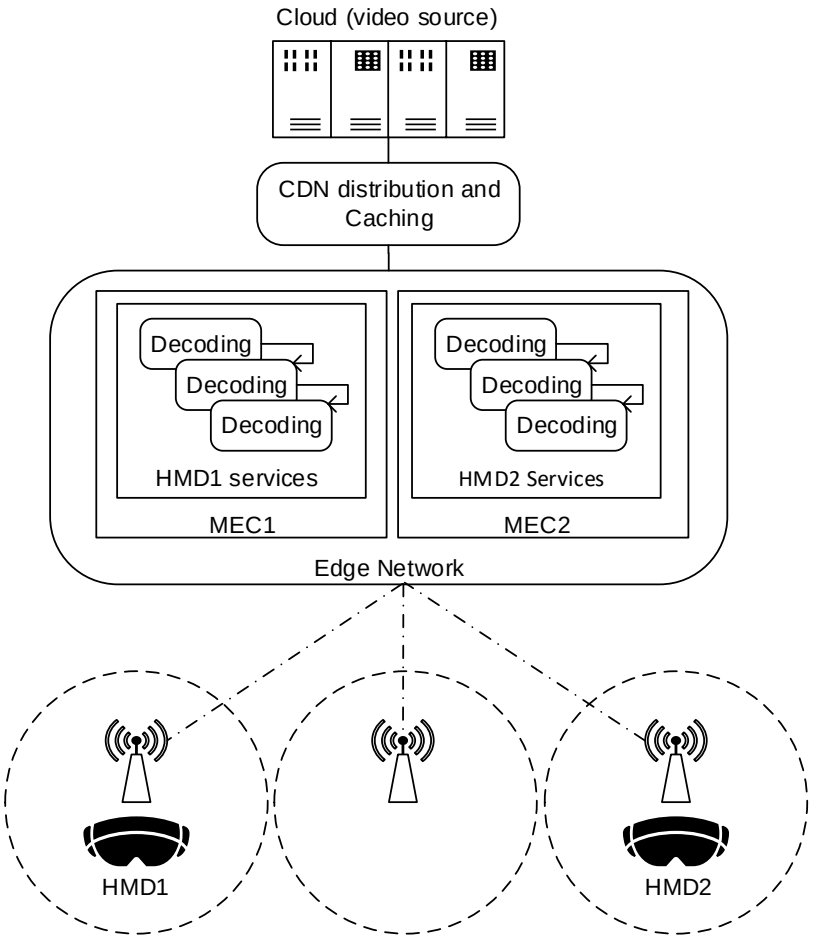


FIGURE 2.5: Edge Computing Infrastructure to Support 6DoF Processing.

As a result, VR 6DoF scenarios are limited to using specialized hardware platforms or edge servers to process 6DoF videos before delivering them to the VR HMDs. Implementing 6DoF VR streaming is challenging because it requires *multiple decoders* operating under low latency and high bandwidth, leading to extreme computing power and high energy consumption on VR HMDs. Beyond those requirements, providing 6DoF VR becomes more challenging due to the VR interaction latency under the limited computation capability of HMDs. Thus, the decoders are offloaded from VR HMDs to the network edge to process the 6DoF content received from cloud servers. Afterwards, the content is aggregated, transmitted to the HMDs, and displayed to VR users. Once VR users change their location, these decoders can be migrated to servers closer to the user. This reduces the latency for transferring the 6DoF video once processed. Figure 2.5 shows the edge computing infrastructure to support 6DoF video processing. Since network edge processing can address computational latency requirements, network latency, and bandwidth requirements are challenging due to the tight coupling of today's network infrastructure. Therefore, *Internet routing* plays a crucial role in supporting ultra-low latency requirements for 6DoF videos in highly scalable scenarios, in which optimizations to reduce the latency must be performed considering the current network infrastructure limitations.

2.1.4 Internet Routing

To address the challenges posed by the evolving VR video content, network optimization becomes a critical aspect of ensuring seamless transmission and user experience over the Internet. The *Internet* operates as a packet-switching network, facilitating the exchange of information among connected computers. This information is formatted into extended sequences of bits, known as *packets*. Critical routing decisions determine how a packet follows a specific route as these packets traverse the Internet to their destination computers. These determinations are carried out by specialized computer devices referred to as *routers*. The complex algorithms these routers employ, collectively called *routing protocols*, enable them to collaborate in making accurate routing decisions, ensuring that data reaches its intended destination efficiently and reliably [40]. Therefore, routing involves the selection of optimal paths within a network. In a computer network, several nodes (routers) are interconnected by paths. In this scenario, routing corresponds to choosing the most suitable path based on predefined criteria and rules. Figure 2.6 shows the Internet infrastructure composed of several routers, which are interconnected to provide packet routing between computers, and each connection contains a latency and throughput value associated with it.

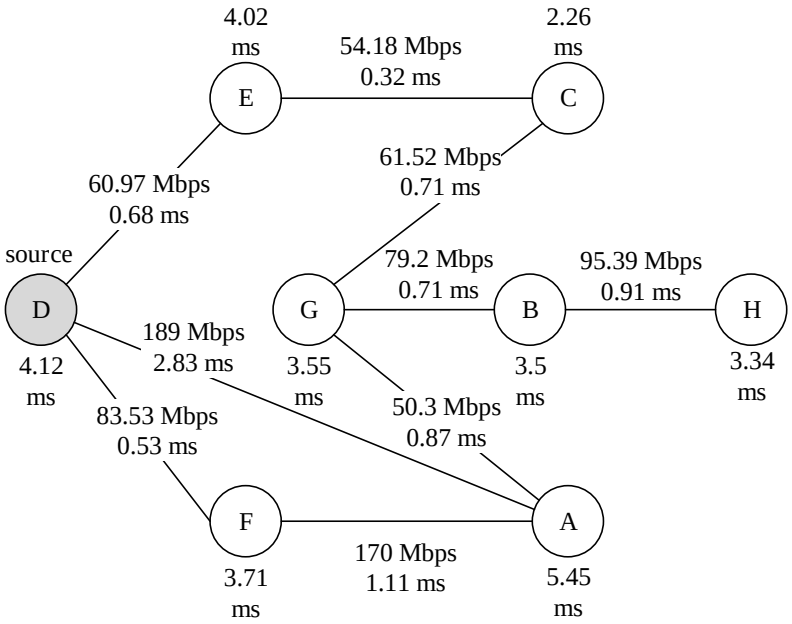


FIGURE 2.6: Internet Infrastructure Architecture.

The Internet has many dynamic routing protocols, such as OSPF, BGP, RIP, IGRP, which can work as *intra-domain* or *inter-domain* routing. In intra-domain, routing algorithms work only within domains, e.g., OSPF and RIP, while in inter-domain routing algorithms work within and between domains, e.g., BGP. Internet routing protocols prioritize throughput over latency since most applications do not have stringent latency requirements. Although this routing model has supported applications since the early days of the Internet, it will become outdated as applications require lower latency and higher throughput [39]. Thus, increasing the Internet's capacity will not keep up with the future demands of immersive applications, especially those with video content supported by 6DoF.

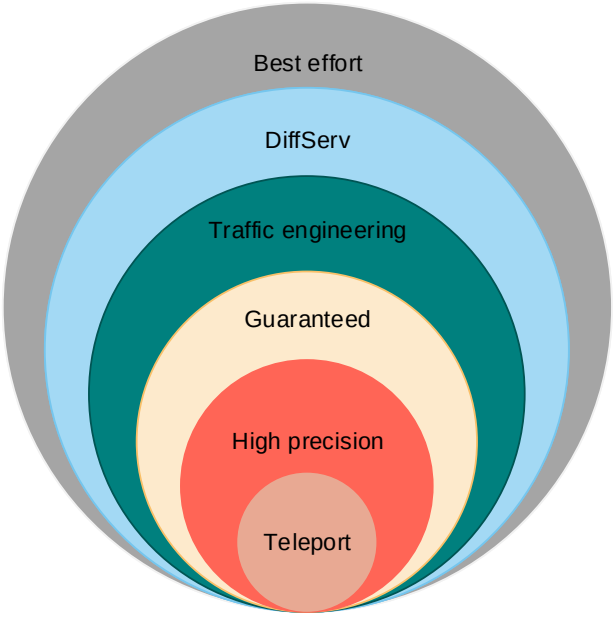


FIGURE 2.7: Approaches to managing and providing network services.

Beyond the design problems of current routing protocols, which do not take latency into account as a primary factor, the Internet architecture cannot guarantee throughput and latency because it works on top of the best-effort concept [65]. For instance, there are several inefficiencies in the protocols used in the Internet architecture, such as tunnel over tunnels, some head fields repeating each other, and transport network header tax, sometimes as high as 90%. As a result, current Internet architecture cannot guarantee delivery constraints, e.g., latency and throughput, for new applications such as VR, AR, and holograms.

While in the *best-effort* approach, there is no prioritization or guarantee of QoS, e.g., traffic is treated equally, Differentiated Services (*DiffServ*) is a more sophisticated and scalable QoS mechanism, where traffic is classified into different classes or service levels based on specific attributes or rules [66]. On the other hand, *traffic engineering* is a network management approach focused on optimizing network resource utilization and ensuring efficient traffic flow. It involves techniques for controlling and optimizing the distribution of traffic across the network, typically within a single network domain [67]. However, none of these approaches can meet the high throughput and strict latency requirements to support new services envisioned for the next generation of applications. Thus, new approaches are emerging to *guarantee* the allocation of resources to specific applications, such as VR, VR, and autonomous driving. Furthermore, a more advanced approach is needed to support *high precision* services like remote surgery and industrial Internet. Finally, the last approach is the *teleport*, to support high-definition hologram applications. Developing these new approaches aims to overcome the inefficiencies of previous approaches, e.g., best-effort, DiffServ, and traffic engineering [68]. Figure 2.7 shows the current (*best-effort*, *DiffServ*, and *traffic engineering*) and future approaches (*guaranteed*, *high precision*, and *teleport*) network service approaches, which aim to support the latency and throughput requirement for specific applications.

Traditional routing protocols such as IGRP and EIGRP consider several parameters like throughput, latency, and network load in their path (route) selection process within a network. Typically, the primary focus is selecting paths with the highest throughput capacity while minimizing latency [40]. Nonetheless, this approach fails to efficiently optimize the overall flow throughput utilization while ensuring low latency for all flows, especially those engaged in VR flows, due to their high throughput and stringent latency requirements. The current approach of prioritizing paths with the highest throughput available overlooks the implications of assigning a particular path to a specific data flow, particularly regarding its impact on network latency for other flows, especially during periods of network congestion. In such situations, alternative paths prioritizing network throughput are often chosen, resulting in higher network latency [41].

The network infrastructure must be optimized to meet the latency and throughput requirements of future applications. Therefore, some standardization entities related to Internet routing advocate developing new network optimization techniques, some related to packet routing [68]. For instance, ITU-T has established the Focus Group on Network 2030, FG-NET2030, which examines various networking use cases anticipated to arise over the next ten years [69]. This group is also responsible for defining novel networking services and capabilities designed to meet the requirements associated with these use cases.

Therefore, ITU-T defined the use cases and their network requirements as follows: (i) ***Bandwidth***, including bandwidth capacity, quality of service QoS, Quality of Experience (QoE), flexibility, and adaptable transport; (ii) ***Time***, including latency, synchronization, jitter, scheduling, coordination, and geolocation accuracy; (iii) ***Security***, including privacy, reliability, trustworthiness, resilience, traceability, and lawful intercept; (iv) ***Artificial Intelligence***, including data computation, storage, modeling, collection and analytics, autonomy, and programmability; and, (v) ***Manynets*** (i.e., seamless coexistence of heterogeneous network infrastructures), including addressing, mobility, network interface, and heterogeneous network convergence.

Following the guidelines of the FG-NET2030 focus group, telecom operators, and research groups participating in EU-funded H2020 B5G-OPEN (Beyond 5G-Optical Network Continuum) have identified several key research directions and innovations [70]. With the participation of three major operators within the European Union, namely, Telefonica (TID), Telecom Italia (TIM), and British Telecom (BT), B5G-OPEN has proposed to build an open and domain-less yet high-capacity and smart optical network [71]. In this scenario, routing approaches are being remodeled according to the resources required by each application. Therefore, shortest, widest, and segment routing approaches must be replaced by a new approach called *Preferred Path Routing* [72]. Preferred Path Routing provides path-based dynamic routing for many packet types, including IPv4, IPv6, and MPLS. This seamlessly works with a controller plane, which holds a complete network view of operator policies, while the distributed control plane provides self-healing benefits in a near-real-time fashion [73]. As a result, Preferred Path Routing is tailored to optimize network resources, reducing overall latency while increasing the efficiency of bandwidth resources.

2.1.5 The Role of E2E Latency in 6DoF VR Content Delivery

The main aim of VR is to provide a virtual environment where users feel part of it, just as they do in the real world. To achieve this, VR systems must deliver a large capacity of visual elements that are as realistic as in the real world. As a comparison, humans process nearly 5.2 Gbit/s of sound and light [74]. For VR systems to deliver an advanced experience, they have to deliver at least this amount of data (5.2 Gbit/s), or even more, which needs to be transmitted over the Internet and processed by edge servers or HMDs. In addition to the challenges posed by network and edge infrastructure, the device form factor of HMDs and their restricted computing and power capabilities further restrict VR systems from achieving this level of immersion. In this scenario, one of the most critical metrics in VR systems is the Motion to Photon (MTP) latency, referring to the time gap between an input movement and the corresponding screen update [75]. MTP latency is also known as E2E latency.

The user's ability to experience immersion can be significantly affected by the slightest delay in the user interface, whether it is related to visual or audio elements. Therefore, reducing latency is so vital for VR systems that even health problems can be caused during continuous use of VR systems operating at a higher latency than the human brain requires to process information in the real world. The total MTP latency must be less than 20 ms for a good and secure user experience in VR, which corresponds to a maximum E2E of 20 ms [6]. This is the maximum E2E latency allowed to avoid cybersickness, nauseating, and uncomfortable customer experiences. To put this challenge in perspective, a display running at 60 Hz is updated every 17 ms, and a display running at 90 Hz is updated every 11 ms. However, at 120 Hz, this is reduced to 8.3 ms.

The head movement feature in HMDs equipped with head tracking is a task that demands a more detailed exploration due to its unique set of latency requirements. When users shift their heads, they anticipate an immediate response in visuals and sound. Even the slightest delay in providing this feedback can pose issues for the overall experience. Currently, a target of achieving a MTP latency below 20 ms is needed after many VR user interactions. However, research has indicated that when the MTP latency dips below 15 ms, it essentially becomes imperceptible to the vast majority of users, underlining the significance of minimizing this latency in HMD-based interactions [76].

Minimizing latency plays a crucial role in maintaining the stability of the virtual environment as the user transitions through it in VR systems. Achieving the necessary computing latency standards for VR services solely through the processing power of HMDs is challenging, primarily due to the limited capabilities of VR devices. Typically, VR systems are composed of multiple computational-intensive tasks, e.g., motion prediction, FoV predictions, object detection, and object recognition. Therefore, several processing stages are involved before refreshing the display. For example, the E2E VR pipeline includes sampling the sensors, sensor fusion, view generation, rendering, decoding, image correction, and updating the display. As a result, more advanced VR applications, especially those using 6DoF content, are limited to edge scenarios or specialized hardware platforms.

Latency plays an essential role in VR experiences, but it is important to understand when low latency is crucial and what type of latency is being considered. Typically, there are two types of immersive experiences in VR systems, which are interactive and non-interactive [77]. Latency is critical for *interactive* VR User experiences, in which VR users interact with other users in real-time. Therefore, such interaction must be performed as fast as possible, such as real-world interactions with other people or objects. For *non-interactive* interactions, the content can use a buffer, such as a streamed 360^o video, where network latency is generally not an issue if the buffer remains, making this content more tolerant to higher and inconsistent latency.

Interactive content is particularly very sensitive to network latency. Achieving low latency becomes crucial to guarantee prompt responses in various use cases, including virtually engaging in online multiplayer games, participating in virtual ping-pong matches, overseeing remote machinery, or collaboratively designing vehicles interactively [78]. In these situations, it is imperative that the visual elements accurately depict the ongoing interactions. When network communication transmits these interactions, minimizing latency becomes a pivotal element in the user experience. For instance, it can be the decisive factor between successfully targeting an opponent or falling victim to an opponent's attack.

The lack of precise standards for stringent latency requirements beyond air interface latency (1 ms) is attributed to the nature of network transport components, which vary according to specific scenarios and deployment requirements. This integration may involve the implementation of specialized functionality in the base station, the incorporation of specific elements of the core network, and the use of proprietary software in the field of network and business system control. In contrast, the advent of 5G technology promises to provide this functionality in a consistent and standardized way, simplifying the provision of low-latency services across a broader spectrum of VR applications.

In conclusion, the role of E2E latency in 6DoF VR content delivery cannot be overstated. The immersive quality of VR experiences depends on minimizing latency, especially in interactive scenarios where real-time user interactions are paramount. Achieving low latency, below 20 ms, is crucial for ensuring user comfort and preventing issues like cybersickness. The unique demands of head movement tracking in HMDs further emphasize the need for reduced latency. However, it is essential to recognize that non-interactive content, such as streamed 360^o videos, can tolerate higher latency due to buffering. The absence of precise standards for latency requirements beyond air interface latency is a challenge, but the emergence of 5G technology holds the promise of more consistent and standardized low-latency services across a wide range of VR applications. Minimizing latency is more than just a technical challenge It is a critical factor in shaping the future of immersive virtual reality experiences. In the next section, we investigate the main and most recent work related to the different areas that can be optimized to minimize the overall latency of VR systems, where we focus on ongoing efforts to push the boundaries of immersive virtual reality experiences and enable immersive experiences for VR systems.

2.2 Related Works

2.2.1 Resource Provisioning for VR in Edge Networks

Several studies [15], [26], [28] have investigated alternative approaches for resource elasticity in cloud computing, and they conclude that the scarcity of resources cannot negatively impact services running on large cloud providers, e.g., Amazon EC2, Azure, GCP. Compared to large-scale cloud systems, a MEC server can provide lower communications latency between user and server, but it also comes with fewer resources than cloud infrastructures. The scarcity of MEC resources may affect VR service performance because some under-provisioned services might need to be migrated to another MEC server, introducing service-restart latency in some cases.

One of the most popular container orchestration tools network operators use to support edge computing is Kubernetes [79]. The massive infrastructure investments by network operators drive the move to Kubernetes, enabling containerization in the cloud and at the edge network to afford 5G MEC services based on lightweight virtualization deployments [80]. To allocate system resources to the running services, Kubernetes follows the *auto-scaling* principle, which proposes to reactively increase or decrease the resources allocated to the service according to its current demand. This process is called reactive elasticity [81]. One way in which Kubernetes can adjust the resources allocated to a service is by increasing or decreasing the resources associated with each service through a module named Vertical Pod Autoscaler (VPA). VPA is the elasticity mechanism provided by Kubernetes [82]. The VPA estimates every service's resource utilization. If their current workload goes beyond a threshold, it restarts the resource-intensive services, granting them a more suitable amount of resources. If resources are unavailable on the current server, where the service is already deployed, the VPA redeploys the service to another server. One drawback of restarting or migrating the pod is that stateful context information must be copied between two replicas (in case of a make-before-break approach) or at least stored and reloaded (in case the server does not allow the creation of another pod before tearing down the old one). Hence, Kubernetes' auto-scaling policy reduces the efficiency of resource allocation under resource scarcity conditions because it triggers several resource-reallocation rounds. Nevertheless, the auto-scaling approach adopted by Kubernetes is the approach implemented by most cloud computing providers [83].

Due to resource limitations imposed by MEC servers compared to large-scale cloud providers, a few works have investigated resource elasticity in edge networks [84]. Yuan et al. [85] propose a scheme to serve the time-varying demand for resource capacity from mobile services. The proposed solution deploys online virtual network function scaling, which realizes an on-demand resource allocation in MEC infrastructures. The proposed algorithm's evaluation was verified through numerical simulation and experiments in a real cloud environment. Tseng et al. [86] have presented a virtual resource orchestrator that implements a lightweight auto-scaling mechanism for fog computing in industrial applications. The fuzzy-based real-time auto-scaling (FRAS) mechanism provides a dynamic and low-cost

solution to the service auto-scaling problem in fog environments. The authors also compared the FRAS mechanism with Amazon AWS auto-scaling algorithm and others. Wang et al. [87] propose a framework to manage edge nodes and an auto-scaling mechanism for resource provisioning in edge nodes, which is based on three stages, i.e., handshaking, deployment, and termination. The evaluation considered the application latency and the amount of data exchanged between edge nodes.

Righi et al. [88] present the Elastic-RAN model, which proposes multi-level and adaptable resource elasticity for Cloud Radio Access Networks. Adaptivity refers to the elasticity level at which physical machines and their resources are provisioned close to the current processing needs. Li et al. [89] have proposed an auto-scaling algorithm to minimize costs and deal with unbalanced cluster load caused by resource expansion, i.e., scale-up and the data reliability caused by resource scale-down. The evaluation considers Service-Level Agreement (SLA) parameters and residual resources, e.g., CPU and memory. Antonescu et al. [90] proposes a VM-scaling algorithm for distributed enterprise information systems, which optimally detects the most appropriate scaling conditions using performance models of distributed applications based on SLA-specified performance constraints. Naha et al. [91] developed resource allocation and provisioning algorithms by using resource ranking and provisioning of resources in a hybrid and hierarchical fashion to address the problem of satisfying deadline-based dynamic user requirements in fog computing. These works focus on QoS maintenance at MEC infrastructures. However, they always consider available resources to support the required elasticity demand. Kumar et al. [26] claim that SLA violations need to be detected in the resource provisioning process when resource elasticity issues on cloud and edge servers happen. This can occur under resource scarcity conditions, jeopardizing QoS and QoE.

Li et al. [92] propose a scheduling optimization mechanism for improving consistency maintenance in edge environments. The mechanism is based on a two-level scheduling optimization scheme. If the edge data center does not have enough resources to complete, it will migrate the service to a centralized cloud data center. Castellano et al. [93] proposed DRAGON, a distributed resource assignment and orchestration algorithm that seeks optimal partitioning of shared resources between different applications running over a standard edge infrastructure. The evaluation allowed testing of the algorithm behavior after the hosting resources had been saturated, even running a low number of applications. Tasiopoulos et al. [94] proposed an auction-based resource allocation and provisioning mechanism, which produces a map of application instances in edge computing, namely Edge-MAP. Edge-MAP considers users' mobility and the limited computing resources available in edge micro-clouds to allocate resources to bidding applications. Edge-MAP can reallocate resources to adapt to the dynamic network conditions. Guo et al. [95] recommend an on-demand resource provisioning mechanism based on load estimation and service expenditure (over-provisioned resources) for edge cloud. The mechanism uses a neural network model to estimate the resource demand. However, before releasing the node resources, the user data on the node needs to be migrated to other working nodes to ensure service continuity.

Sarrigiannis et al. [96] proposed a Virtual Network Function (VNF) lifecycle management through an online scheduling algorithm, where the VNFs are orchestrated, e.g., instantiated, scaled, migrated, and destroyed, based on the actual VNF traffic. The authors also proposed an experimental evaluation based on the implementation of a MEC-enabled 5G platform. The assessment aimed to maximize the number of served users in MEC by taking advantage of the online allocation of edge resources without violating the application SLAs. Akhtar et al. [97] proposed the management of chains of application functions over multi-technology edge networks. This work provides resource orchestration and management solutions for applications over a virtualized edge computing infrastructure. Son et al. [98] propose a dynamic resource provisioning algorithm for VNFs deployed at the network edge. The algorithm automatically allocates resources in the edge and the cloud for VNFs to adapt to dynamically changing network volumes. The algorithm considers the latency requirement of different applications in the SFC, allowing latency-sensitive applications to reduce the E2E network latency by utilizing edge resources over the cloud.

The aforementioned works trigger service migration in resource scarcity situations, which can affect VR service QoS [14]. Migrating a VR service has several drawbacks, such as increased latency, traffic congestion, and network usage costs due to the data transferred between remote hosts. In the real world, where multiple network operators manage the infrastructure, migrating a VR service may take longer than expected because mobile network operators must agree to exchange the service across heterogeneous platforms. We observe that only a few studies in the literature have investigated resource elasticity in MEC, and those who do are characterized by a set of common limitations, detailed hereafter. First, resource elasticity models do not consider the resource scarcity of MEC in their design. Second, most related works frequently trigger service migration procedures. Finally, most related works do not optimize MEC resource utilization, resulting in a long *elasticity attempt window*. In **Paper [1]**, we tackle these limitations arising from previous works by proposing REACT, a self-adaptive elasticity mechanism, a heuristic solution tailored to MEC resource scarcity conditions.

Based on the literature review, we identify that new approaches need to evolve to tackle resource elasticity among MEC systems while meeting the stringent requirements of VR applications. This imposes a set of challenges when carrying out elasticity strategies in large-scale MEC scenarios since it cannot accommodate a high density of resource elasticity requests for VR systems, especially 6DoF VR applications. Thus, it becomes even more problematic by directly affecting VR application performance. Although MEC servers have computing power, with the increase of users, their limited computing power is gradually overloaded, which cannot guarantee the QoS of VR applications. The challenge involves designing an optimal resource elasticity mechanism to support VR application requirements.

We claim that MEC characteristics, e.g., resource limitation, lead to the adoption of optimal self-scaling solutions, affording QoS and resource-constrained awareness to keep VR applications always better served by the underlying MEC facilities [99]. The list of requirements we claim for an optimal solution of MEC-tailored elasticity mechanism includes the following requirements to be met:

Solutions (References)	Requirements			
	Constrained capacity	Successful auto-scaling	Elasticity attempts	Self-adaption
Kubernetes VPA	✓			
Yuan et al.[85]	✓			
Wang et al. [87]	✓			
Righi et al. [88]	✓			
Chunlin et al. [89]	✓			
Antonescu et al. [90]		✓		
Naha et al. [91]	✓		✓	
Li et al. [92]	✓		✓	
Castellano et al. [93]	✓		✓	✓
Tasiopoulos et al. [94]	✓		✓	
Guo et al. [95]	✓			
Sarrigiannis et al. [96]	✓			
Akhtar et al. [97]	✓			
REACT	✓	✓	✓	✓

TABLE 2.1: Comparison of REACT with related works towards optimal MEC-tailored elasticity.

1. Provisioning capacity in MEC environments, where MEC servers must meet the auto-scaling requests for any service;
2. Capacity to provide auto-scaling whenever the service needs more resources, employing an enhanced *elasticity attempt window* to respond to new loads;
3. Successful auto-scaling under resource scarcity conditions and decreasing the number of unsuccessful elasticity attempts;
4. Deploying a self-adaptive approach to tackle the issues that widely-used reactive auto-scaling solutions raise.

Table 2.1 compares the main characteristics of the related works concerning the requirements mentioned above and shows that none of the considered solutions can support all our claimed requirements towards optimal auto-scaling. In particular, we compared REACT against the state-of-the-art solutions considering the constrained capacity of MEC servers, the capacity to provide successful auto-scaling, the elasticity attempts, and the ability to self-adaption in scenarios with limited resources. Meeting all the above requirements ensures that VR applications have priority when deploying their services at the MEC servers. Without this type of guarantee, the latency of VR applications may be impaired due to the scarcity of MEC servers to accommodate VR services. Thus, latency may be higher when fewer MEC servers are available. As a main consequence, the QoS and QoE of such applications are directly impacted, even if the latency is increased by a few milliseconds. To address such a problem, applications that demand strict latency requirements must have priority over other applications to use network edge resources. Motivated by the limitations of the reactive approaches of related works and the research questions described in Section 1.3.1, we propose the REACT solidarity-based elasticity strategy, as described in **Paper [1]** and Chapter 3.

2.2.2 Edge-enabled 6DoF VR Deployment

Previous studies [57], [61], [63], [100] have shown that edge computing enables advanced 6DoF VR experiences by supporting the deployment of compute-intensive services. Chakareski et al. [57] we evaluate two candidates emerging technologies, Free Space Optics (FSO) and millimeter-wave, which both offer unprecedented available spectrum and data rates. The authors formulate an optimization problem to maximize the delivered immersion fidelity of the envisioned dual-connectivity 6DoF VR streaming, which depends on the WiFi and millimeter-wave and FSO link rates, and the computing capabilities of the server and the user's VR HMD. The problem is mixed integer programming, and they formulate an optimization framework that captures the optimal solution at lower complexity. To evaluate the performance of the proposed systems, the authors collect actual 6DoF measurements. Results demonstrate that both FSO and millimeter-wave technologies can enable the streaming of 8K-120 Frames per Second (FPS) 6DoF content at high fidelity.

Hou et al. [63] consider motion prediction and pre-rendering services at the edge network to enable low latency 6DoF VR. The edge-based predictive pre-rendering approach can address the challenging 6DoF VR content. The proposed VR edge intelligence comprises predicting both the head and body motions of a user accurately using past head and body motion traces. Furthermore, the authors have developed a multi-task long short-term memory model for body motion prediction and a multi-layer perceptron model for head motion prediction.

Pan et al. [100] propose a novel 5G Mobile Edge Assisted Metaverse Light-field-video System, integrating light of field collection, metaverse content construction, and an extended reality viewing scheme. The work addresses major technical challenges in light-field-video delivery through mobile networks with the edge and cloud infrastructure. Aiming to reduce computation latency, a fast sparse reconstruction algorithm for metaverse content construction is established with edge-cloud collaboration. Intelligent service for users is deployed on edge to achieve viewport-driven extended reality viewing. The proposed edge-assisted metaverse algorithm aims to reduce the computational latency of 6DoF videos.

Jeong et al. [61] propose a viewport-dependent high-efficiency video coding (HEVC)-compliant tiled streaming system on a test model for immersive video, MPEG-Immersive multiview compression reference software. Besides, the authors propose a 6DoF viewport tile selector for multiple 360^o video-tiled streaming. The authors also introduce a viewport-dependent multiple-tile extractor. The proposed system detects the user's head movement, selects the tile sets corresponding to the user's viewport, extracts tile bitstreams, and generates a single bitstream. The extracted bitstream is transmitted and decoded to render the user's viewport. The proposed viewport-dependent streaming method can reduce the decoding time and bandwidth.

Although those research efforts have been devoted to designing solutions for enhancing 6DoF VR experiences at network edges, the impact of 6DoF videos on mobile HMD has so far drawn little attention. In contrast, our work considers the characteristics of 6DoF VR videos and the restrictions of mobile HMD.

Recent works [101]–[103] study the behavior of the E2E latency and other QoS metrics of VR applications when their services are deployed on the MEC infrastructure. Mangiante et al. [104] propose FoV rendering at the network edge to optimize the bandwidth and latency required by VR 360^o video streaming. The authors also analyze the trade-off between bandwidth, compute, and latency to transmit 360^o videos. However, the authors argue that ultra-low network latency must be guaranteed to achieve real-time rendering as the experience becomes interactive and upstream control is sent from end devices to the network.

Wang et al. [101] investigate the offloading decisions of Mobile Augmented Reality (MAR) applications from multiple users, each of which is comprised of a chain of dependent tasks over a generic cloud-edge system. The authors formulate the Multi-user Multi-task MAR Application Scheduling (M3AS) problem, which is $\mathcal{NP}$-hard. The authors present Mutas, an efficient scheduling algorithm that jointly optimizes server assignment and resource management. They also consider the online version of M3AS and present OnMutas. Extensive evaluations demonstrate that Mutas and OnMutas can significantly reduce the service latency of MAR applications compared to three other heuristics.

Alencar et al. [102] propose a QoE VR-based mechanism for allocating microservice dynamically in 5G architectures, called Fog4VR. Fog4VR determines the optimal fog node to allocate the VR microservice based on latency, migration time, and resource utilization rate. The authors also present the INFORMER, an integer linear programming model aiming to find the optimal global solution for microservice allocation. Results obtained with INFORMER serve as a baseline to evaluate Fog4VR in different scenarios using a simulation environment. Results demonstrate the efficiency of Fog4VR compared to existing mechanisms in terms of cost, migration time, fairness index, and QoE.

Santos et al. [103] propose a multi-criteria SFC orchestration scheme for Multi-User VR services called MuSFiCO. MuSFiCO maps edge computing resources and instantiates SFCs on distributed servers based on latency threshold, CPU and memory resources, and bandwidth. The authors developed a constrained-based heuristic to minimize latency and compare it with the baseline monolithic deployment and cloud-based SFC algorithms. Results demonstrate the efficiency of MuSFiCO compared to other approaches in terms of latency, CPU, memory, bandwidth utilization, and orchestration decision-time.

These studies show the potential of offloading VR services to MEC servers to reduce latency. However, they do not consider scenarios where deploying a subset of services directly on HMDs would lead to a better system-wide average latency. As a result, they fail to minimize the overall E2E latency for VR systems that use the edge infrastructure to deploy their services.

Likewise, some other recent works [97], [105]–[107] study how different policies for distributing services between mobile devices and MEC servers impact the VR applications' QoS metrics, such as latency. The authors in [105] have shown that VR-intensive computing services, such as scene depth estimation, image semantic understanding, 3D scene reconstruction, and high realism rendering, must be processed in real-time to ensure natural and smooth experiences.

Lai et al. [106] show that the QoE achievable for high-quality VR applications on today's mobile hardware and wireless networks via local rendering or offloading is about 10X away from the acceptable QoE, yet waiting for future mobile hardware or next-generation wireless networks (e.g. 5G) is unlikely to help, because of power limitation and the higher CPU utilization needed for processing packets under higher data rate. Furthermore, the authors present Furion, a VR framework that enables high-quality, immersive mobile VR on today's mobile devices and wireless networks. Supplemented with video compression, use of panoramic frames, and parallel decoding on multiple cores on the phone, Furion can support high-quality VR applications on today's smartphones over WiFi, with under 14 ms latency and 60 FPS.

Younis et al. [107] propose a novel Mobile Edge Computing framework for AR applications (MEC-AR). MEC-AR is designed to take advantage of 5G cellular networks and make optimized computation-offloading decisions in a multi-tiered hierarchy. In the context of MEC resource management, the authors cast a mixed integer linear program to find an efficient application placement on the MEC-AR layers to minimize the network latency. Simulation results coupled with real-time experiments on a small-scale MEC testbed show that our hierarchical computation mechanism improves the performance of mobile AR applications in terms of both energy consumption and network latency.

Akhtar et al. [97] provide resource orchestration and management solutions for applications over a virtualized client-edge-server infrastructure. The authors investigate the problem of optimal placement of pipelines of SFCs and the steering of traffic through them over a multi-technology edge network model consisting of both wired and wireless millimeter-wave links. This problem is $\mathcal{NP}$-hard. Therefore, the authors provide a comprehensive microscopic binary integer program to model the system, along with a heuristic that is one order of magnitude faster than optimally solving the problem.

These studies show that in some cases, distributing services among MEC infrastructure reduces latency and improves other QoS metrics. However, none of these works considers power consumption on HMDs in their service offloading strategies, which may lead to unpredictable HMD battery lifetime.

Other related works [108], [109], [110], [111], [112], [113] study either latency reduction or energy consumption optimization in edge networks. Liu et al. [108] a panoramic VR video (PVRV) streaming system that is designed for modern multiconnectivity-based millimeter-wave cellular networks in conjunction with MEC. With the help of the MEC server, the trade-off among link adaptation, transcoding-based chunk quality adaptation, and viewport rendering offloading is sought to improve the wireless bandwidth utilization and mobile device's energy efficiency. Simulation results show that the proposed scheme can improve the streaming performance in energy efficiency and the quality of the received viewport over the state-of-the-art schemes.

Zheng et al. [109] investigate the scenario of multi-tiles-based wireless VR video service with the aid of MEC network, where the primary objective is to minimize the system energy

consumption and the latency as well as to arrive at a trade-off between these two metrics. The authors first cast the time-varying view popularity as a model-free Markov chain and use a long short-term memory auto-encoder network to predict its dynamics. Then, a mixed strategy, which jointly considers the dynamic caching replacement and the deterministic offloading, is designed to fully utilize the caching and computing resources in the system.

Santos et al. [110] discuss that the main obstacle between current technology and future remote, multi-user AR/VR applications is the stringent E2E latency requirement, which cannot exceed 20 ms to avoid motion sickness. The authors show that several challenges still arise concerning deploying and managing VR services. Therefore, the authors present a mixed-integer linear programming formulation for orchestrating VR services in fog-cloud infrastructures. The evaluation of realistic VR container-based SFC shows that deploying VR components hosted in a fog-cloud infrastructure can satisfy the 20 ms latency boundary.

Doan et al. [111] formulate the novel Subchain-Aware NFV service Placement (SAP) optimization model that accounts for the configuration cost for stitching together reused network functions to an SFC and strives to reuse existing subchains of consecutive network functions. The authors also developed Tabu-SAP, a Tabu search approach to solve the SAP optimization problem. Furthermore, they introduced the novel Automated Provisioning framework for MEC (APMEC). APMEC supports multiple management and orchestration frameworks through a loose coupling MEC design. Tabu-SAP evaluations indicate an eightfold increase in the number of supported SFCs compared to the state-of-the-art reuse of individual network functions.

Mandal et al. [112] analyze the network service availability considering the deployment of network service using multiple host nodes, single host nodes, and mixed-mode. In the availability analysis, the authors consider the failure perspective of VNFs and the failure perspective of host node(s). Further, they analyze the network service reliability considering the placement of VNFs of network services based on different host nodes, single host nodes, and mixed-mode. The authors also compare the availability of network services considering these three placement strategies. Comparison results show that the availability and reliability are better considering single host node-based placement of VNFs of network services.

Zheng et al. [113] show how to apply network function parallelism into SFC and embedding process such that the latency, including processing and propagation latency, can be jointly minimized. Considering parallel relationship constraints, the authors propose a novel problem called parallelism-aware service function chaining and embedding (PSFCE). The authors propose a near-optimal maximum parallel block gain (MPBG) first optimization algorithm when computing resources at each physical node are enough to host the required SFCs. When computing resources are limited, the authors propose a logarithm-approximate algorithm called parallelism-aware SFC deployment (PSFD). They found out that (i) MPBG is near-optimal, (ii) the optimization of E2E service latency largely depends on the processing latency in small networks, and (iii) PSFD outperforms the schemes directly extended from existing works regarding E2E latency.

Solutions (References)	Characteristic						
	Offload.	Migration	E2E latency	Power consum.	MEC support.	HMD support.	SFC
Chakareski et al. [57]					✓	✓	
Jeong et al. [61]	✓					✓	
Akhtar et al. [97]				✓			✓
Pan et al. [100]					✓		
Wang et al. [101]	✓				✓		✓
Medeiros et al. [1]					✓		
Alencar et al. [102]		✓					✓
Santos et al. [103]	✓			✓			✓
Ruan et al. [105]	✓			✓			
Lai et al. [106]	✓			✓			
Younis et al. [107]	✓		✓		✓		
Liu et al. [108]	✓		✓	✓			
Zheng et al. [109]		✓	✓	✓			
Santos et al. [110]	✓	✓		✓			✓
Doan et al. [111]							✓
Mandal et al. [112]	✓	✓					
Zheng et al. [113]							✓
TENET	✓	✓	✓	✓	✓	✓	✓

TABLE 2.2: Comparison of TENET algorithm to related works.

We found out that the main limitation of the state-of-the-art works is that they do not consider strict latency guarantees in their service deployment solutions, which are required to ensure that no VR application experiences latency that may impair QoS. Unlike all works presented in this section, our work considers both latency and power consumption on the HMDs to compute an optimal service offloading policy between MEC infrastructure and HMDs while considering QoS constraints. Furthermore, these works do not consider strict latency guarantees in their service deployment solutions, which are required to ensure that no VR application experiences latency that may impair QoS. Based on the limitations of the state-of-the-art works, we provide optimizations on service placement to reduce the overall E2E latency for all VR applications deployed on the system, with a trade-off procedure that analyzes the impact on the power consumption against the reduction of the overall E2E latency.

Unlike all works presented in this section, our works in **Paper [2]** and **Paper [3]** consider both latency and power consumption on the HMDs to compute an optimal service offloading policy between MEC infrastructure and HMDs, while considering QoS constraints. Table 2.2 compares the main characteristics of the related works concerning service offloading, service migration, E2E latency, power consumption, MEC-supported, HMD-supported, and SFC. Table 2.2 shows that none of the considered solutions can support all our claimed requirements for E2E latency reduction. Motivated by the limitations of the approaches presented in this section and the research questions described in Section 1.3.2, we propose TENET, as described in Chapter 4.

2.2.3 Latency Sensitive Routing Algorithms

Several studies address network routing, most of which can be applied to SDN [114]. However, most existing works only provide routing with throughput guarantees and ignore the E2E latency in routing [115]. With the emergence and envisioned widespread adoption of immersive communications such as VR, AR, and holograms, routing algorithms must ensure latency and throughput simultaneously [116]. Thus, shortest path-based approaches are suitable candidates for addressing routing problems with latency and throughput guarantees. Although shortest path-based approaches can provide paths with latency and throughput guarantees, they cannot balance network load and handle real-time demands [117]. In addition, such approaches are prone to always selecting the shortest available path in terms of latency for the flow being processed, where this flow does not always require as low a latency as the path calculated by shortest path-based approaches [118]. As a result, consecutive flows might be affected if they need ultra-low latency paths and such paths are not available on the network. Therefore, current approaches based on the shortest path do not consider how network congestion can negatively impact flows that require ultra-low latency paths, such as 6DoF VR.

Typically, routing algorithms consist of finding a path from a source node to a destination node while satisfying a specific constraint or threshold on the cost. Handler et al. [119] developed a Lagrangian relaxation algorithm for the problem of finding the shortest path between two nodes in a network, subject to a knapsack-type constraint. The authors show that finding the shortest path for all flows in a network while considering more than one metric is impractical because this problem is $\mathcal{NP}$-hard. The authors proposed a k-*th* shortest path algorithm to address such complexity. Wang and Crowcroft [120] investigated throughput-latency-based routing algorithms. They propose two routing algorithms based on throughput and latency metrics, in which they addressed the implications of those routing metrics during the calculation of new paths. However, latency and throughput are not considered simultaneously when designing the algorithms.

Wang and Crowcroft [121] have also investigated QoS routing for supporting multimedia applications. They propose the Minimum Delay Algorithm (MDA), a routing mechanism with latency and throughput guarantees, where MDA prunes all links with insufficient throughput to identify the path with the lowest latency. Yang et al. [122] also proposed QoS routing algorithms for throughput-latency-constrained applications. Those are based on computing the latency-weighted capacity for each ingress-egress pair, where they identify critical links as those whose inclusion in a path will cause the latency-weighted capacity of several ingress-egress pairs to decrease. The works mentioned above are the most relevant regarding routing with latency and throughput guarantees. However, they fail to provide a realistic assessment of the behavior of the proposed routing algorithms in a scenario with overloaded network links. Furthermore, they do not address the latency difference between the allocated path and the latency required by the flow (*over-provisioning latency*) and the E2E latency, which can negatively affect other latency-sensitive flows.

Only a handful of traffic engineering studies consider latency and throughput simultaneously when establishing routing for SDN. Soorki et al. [123] proposed a label-switched protocol routing with throughput guarantees and E2E path latency. The proposed routing algorithm uses data of the ingress–egress node pairs in the network, where the authors used LR-servers theory to compute path latency. However, the main drawback is the lack of latency guarantees in large-scale network scenarios. Tomovic et al. [124] proposed a fast and efficient throughput-latency-constrained routing algorithm for SDN. The proposed algorithm aims to maximize the utilization of network resources, where it classifies traffic flows in a finite number of categories based on the latency sensitivity level. The evaluation shows that the proposed approach to QoS provisioning leads to fewer rejected QoS requests under a wide range of system parameters than other complex solutions.

Li et al. [125] propose a fuzzy-based fast routing algorithm with latency-throughput (FRLR) guarantee over SDNs. FRLR develops the dynamic routing problem in SDN using a traffic flow classification mechanism based on a fuzzy system. In addition, FRLR classifies traffic flows based on latency to reduce computational complexity. Besides, FRLR redirects traffic flows and reduces network energy consumption by identifying critical links and selecting appropriate paths. Wu et al. [126] proposed an intelligent fuzzy-based routing algorithm for video conferencing service provisioning under latency and throughput constraints guarantees in SDN, where the details of future requests are unknown. First, the network is weighted based on critical links to reduce routing interference on future requests. Second, the proposed algorithm uses a deferral module to temporarily postpone requests with high resource demands to manage connection priorities. Although the works mentioned above address latency-based short paths while meeting throughput requirements, they fail to optimize the shortest paths for all flows arriving in the network in network congestion situations. A disadvantage in this scenario is that such approaches are not tailor-made for ultra-latency-sensitive applications such as 6DoF VR.

Other works [127]–[129] study how to ensure both throughput and latency. Gong et al. [127] designed a fuzzy routing algorithm with latency and throughput guarantees for video conferencing services over SDN. The proposed fuzzy system is based on rules that can postpone requests with high resource demands, where it distributes the network workload evenly for all requests. This is performed by maintaining the capacity to accept future requests. The authors used the following KPIs in the evaluation: number of accepted requests, average path length, energy consumption, load balancing, and average latency over different network topologies. Cheng et al. [128] investigated a guaranteed latency-throughput-loss routing algorithm based on a fuzzy approach. The proposed algorithm aims to increase the number of routed requests and improve the performance of conference services in SDN. The algorithm uses a postponement mechanism to improve the conference service, prioritizing the requests with low resource demand for connection. In addition, the algorithm has a hold time mechanism to release the reserved resources after satisfying the request requirements. This mechanism can increase the processing capacity of future requests by conserving network resources.

Zhao et al. [129] proposed a fuzzy logic-based intelligent multi-attribute routing scheme for SDN. The proposed scheme is divided into two phases: the routing path calculation and the multi-attribute routing decision-making. The authors construct the topology diagram in SDN to find efficient routing paths. To solve the uncertainty problem of multiple attributes, they apply fuzzy logic to identify the weight of each attribute in the proposed algorithm. The assessment showed that the proposed algorithm improved the packet delivery ratio and reduced average E2E latency in small network scenarios over SDN. Although the above works address routing with latency and throughput guarantees, they fail to design solutions to optimize the overall latency for all flows in the network because they do not consider how to approximate the latency of the calculated path and the latency required by each flow. Consequently, many paths with ultra-low latency are allocated to flows that do not demand such strict latency requirements. Thus, other flows that demand ultra-low latency are negatively affected as the network becomes overloaded. Therefore, the more restrictions there are on the availability of resources in a network, the more likely flows requiring ultra-low latency will be affected due to the inefficiency of allocating flows according to their demands. This raises the need for a routing algorithm tailored to flows that demand ultra-low latency.

Other works study how to ensure throughput, low latency, low packet loss rate, or link load balancing, but not latency and throughput. Alidadi et al. [130] presented a throughput guarantee with a low-complexity algorithm for traffic engineering. The proposed algorithm ensures throughput routing using a compromise between path cost, path length, and load balancing. Yang et al. [131] investigated the joint virtual function deployment and flow routing strategy to maximize the completed tasks with the guaranteed E2E latency. Alidadi et al. [132] also proposed other optimized routing algorithm for QoS traffic engineering in SDN-based mobile networks. The proposed algorithm advances throughput-restricted routing as it trade-offs between network load balancing and route length with low complexity in mobile networks.

Kamboj et al. [133] proposed a QoS-aware dynamic multipath routing scheme for enhancing QoS of high-throughput applications in an SDN-enabled network. The proposed scheme consists of three phases: flow splitting, multipath routing, and flow reordering. However, flow reordering is not implemented for complex and large network scenarios. Wang et al. [134] proposed an optimal flow and capacity allocation in multiple joint quickest paths of directed networks to address the quickest path problem (QPP). QPP presents a good link for path lengths and capacities with the transmission time of the flow. To address such a problem, the authors proposed an edge-path form traffic model for flow through multiple joint paths with different lengths in one-source, one-sink directed networks. Ali et al. [135] investigate an adaptive bitrate video transmission using cross-layer routing. The proposed algorithm considers the path selection based on the QoS-aware E2E path latency. Despite the efforts of the above works to achieve routing with QoS constraints, they do not evaluate the proposed algorithms in complex network scenarios, where congestion can occur and alternative paths are selected for network flows. Besides, some implement flow reordering, which is impractical in real networks.

Solutions (References)	Characteristic						
	Flow network latency	Path latency	E2E latency	Over-prov. latency	Network throug.	Frame rate	Path latency approx.
Handler et al. [119]		✓					
Wang et al. [120]		✓			✓		
Wang et al. [121]		✓			✓		
Yang et al. [122]		✓			✓		
Soorki et al. [123]		✓			✓		
Tomovic et al. [124]	✓	✓			✓		
Li et al. [125]	✓	✓			✓		
Wu et al. [126]		✓			✓		
Gong et al. [127]	✓	✓			✓		
Cheng et al. [128]		✓			✓		
Zhao et al. [129]		✓	✓		✓		
Alidadi et al. [130]		✓			✓		
Yang et al. [131]		✓	✓				
Alidadi et al. [132]					✓		
Kamboj et al. [133]					✓		
Wang et al. [134]		✓					
Ali et al. [135]		✓	✓			✓	
FLATWISE	✓	✓	✓	✓	✓	✓	✓

TABLE 2.3: Comparison of FLATWISE algorithm to related works.

Unlike all works presented in this section, our work in **Paper [4]** addresses routing with E2E latency and throughput guarantee to reduce the overall E2E latency for all VR applications deployed on the network. Besides, our work in **Paper [4]** considers the over-provisioned latency, E2E latency, and path latency approximation during path calculations between VR HMDs and cloud servers hosting the VR application logic. Table 2.3 compares the main characteristics of the related works concerning flow network latency, path latency, E2E latency, over-provisioned latency, network throughput, frame rate, and path latency approximation.

Table 2.3 shows that none of the considered solutions can support all our claimed requirements towards overall E2E latency reduction for 6DoF VR applications. Furthermore, we observe that most related works on routing with latency and throughput guarantees do not consider the impact of latency on the video quality. Another crucial factor neglected by related works is the relationship between the latency required by the flow and the latency offered by the calculated path (*over-provisioned latency*). Another limitation is that most state-of-the-art approaches are unaware of E2E latency during route calculation, preventing them from minimizing latency as much as possible for all flows. Finally, only using shortest path-based approaches as the flow allocation criteria might lead to bottlenecks, as all flows tend to select links with lower latency, overloading them. Motivated by the limitations of the approaches presented in this section and the research questions described in Section 1.3.3, we propose FLATWISE in **Paper [4]**, as described in Chapter 5.

2.3 Chapter Conclusions

In Section 2.1, we have presented a comprehensive overview of the theoretical background for grasping the core concepts employed in the proposed approaches of this thesis, which are covered in chapters 3, 4, and 5. In Section 2.2, we have presented the state-of-the-art works needed to address the research questions of this thesis, described in Section 1.3. The study of the related work revealed a few drawbacks to be further investigated to achieve ultra-low latency for VR systems. In the following, we summarize the main drawbacks of the related work we address in this thesis.

In Section 2.2.1, we analyzed the most recent works on resource provisioning for VR in edge networks, where we found out that new resource provisioning approaches tailored to MEC servers must be developed to ensure VR service deployment and, therefore, guarantee that VR services are deployed as close as possible from VR users. We have identified that current edge resource approaches trigger service migration in resource scarcity situations, which can affect VR service QoS. This imposes challenges when carrying out elasticity strategies in large-scale MEC scenarios since it cannot accommodate a high density of resource elasticity requests for VR systems, especially 6DoF VR applications.

In Section 2.2.2, we identified the most recent works on edge-enabled 6DoF VR deployment, where we found out that research efforts have been devoted to designing solutions for enhancing 6DoF VR experiences at network edges. However, the impact of 6DoF videos on mobile HMD has so far drawn little attention. As a result, they fail to minimize the overall E2E latency for VR systems that use the edge infrastructure to deploy their services. Furthermore, the works analyzed in Section 2.2.2 do not consider strict latency guarantees in their service deployment solutions, which are required to ensure that no VR application experiences latency that may impair QoS.

In Section 2.2.3, we analyzed the essential works on latency-sensitive routing, where we have shown that current approaches based on the shortest path do not consider how network congestion can negatively impact flows that require ultra-low latency paths, such as 6DoF VR. The works reviewed in Section 2.2.3 do not support E2E latency awareness during the routing process nor consider how the over-provisioned latency can affect other flows on the network, which prevents the routing approaches from minimizing latency as much as possible for all flows in the network.

To overcome the issues described in the related work and answer the research questions, we design and evaluate three approaches in this thesis, which aim to reduce the latency for VR applications, as described hereafter in Chapters 3, 4, and 5. In particular, Chapter 3 addresses the challenges of resource provisioning for VR in edge networks, as described in Section 2.2.1. Chapter 4 addresses the issues of edge-enabled 6DoF VR deployment, as shown in Section 2.2.2. Finally, Chapter 5 addresses the problems of latency-sensitive routing described in Section 2.2.3.

Chapter 3

Enhancing VR Deployment over Edge Networks

3.1 Introduction

With the increasing demand for immersive and interactive VR experiences, edge computing has emerged as a promising solution to support VR-intensive computing tasks [9]. Edge networks, powered by MEC infrastructure, offer proximity to VR end-users, reducing the latency by processing VR services [21]. However, MEC faces limitations in terms of computing and communication resources, which can negatively impact the quality of VR services during high-demand situations [20]. Service migration from overloaded to less loaded MEC servers is a common approach to maintaining satisfactory VR service QoS [14]. Consequently, this migration process can lead to service downtime and high migration costs if large amounts of data need to be transferred.

To address these challenges, enhancing MEC resources by developing resource provisioning strategies tailored to VR deployments and considering MEC resource limitations is essential. Therefore, VR services require resource guarantees to prevent resource shortages and maintain optimal performance. However, supporting VR service deployment on MEC servers while providing resource provisioning guarantees in scenarios with limited resources is challenging. The coexistence of several VR services with varying resource requirements further complicates the problem. As a result, deploying VR services into MEC servers to ensure low-latency and responsive VR experiences becomes more complex in a scenario with limited computing resources.

In this chapter, we investigate the problem of resource scarcity on resource-constrained MEC infrastructures and how to overcome it in the context of VR services deployment. The following sections discuss the content described in **Paper [1]**. Therefore, Section 3.1 describes the research questions addressed in this chapter and discusses the contributions of the new heuristic to address the resource scarcity problem. Section 3.2 describes the system model and formulates the resource scarcity problem. Section 3.3 presents the REACT approach in detail. Section 3.4 describes the experiment setup. Section 3.5 discusses the evaluation results. Finally, section 3.6 concludes the study in this chapter.

3.1.1 Research Questions Addressed

The contributions of this chapter aim to answer the following research questions described in Section 1.3.1.

Research Question 1.1: How to design a resource provisioning mechanism for edge computing infrastructures to support the deployment of multiple heterogeneous VR services?

Research Question 1.2: How can the resource scarcity problem be effectively addressed in the coexistence of several VR services deployed on a shared edge infrastructure?

Research Question 1.3: How to prioritize the resource provisioning of VR intensive-computing tasks in a distributed edge infrastructure with limited computing resources?

3.1.2 Chapter Contributions

To address the abovementioned challenges, we propose in **Paper [1]** REACT, a mechanism that leverages resource provisioning among different services running on a shared MEC server. We address the ***Research Question 1.1*** by exploiting how REACT adopts an adaptive and solidarity-based strategy to redistribute resources from over-provisioned to under-provisioned services in edge environments. We address the ***Research Question 1.2*** by showing that REACT is an alternative strategy to avoid service migration due to resource scarcity. The key idea of the REACT is to prioritize resource provisioning for real-time VR applications. With such prioritization, REACT enhances the performance of high-priority VR services, especially when the edge infrastructure resources become scarce. Therefore, we address the ***Research Question 1.3*** by enhancing the resource provisioning requests for high-priority VR services. Our contributions are as follows.

- We develop an adaptive and solidarity-based strategy to redistribute resources from over-provisioned services (*low-priority services*) to under-provisioned VR services (*high-priority services*) in MEC environments.
- We design our resource provisioning approach considering that VR computing-intensive services take priority (*high-priority services*) over other services classified as *low-priority services*.
- We consider a resource-constrained MEC scenario to evaluate our approach against Kubernetes, a reactive algorithm to provide resource provisioning in MEC servers.
- Compared to the baseline algorithm solution, we enhance the resource provisioning requests for *high-priority* VR services.
- We optimize the resource provisioning in edge computing infrastructures by reducing the amount of over-provisioning resources.
- We reduce the overall service outages for all VR services deployed in MEC servers whenever these server resources become unavailable.

3.2 System Model and Problem Formulation

The considered MEC infrastructure consists of a set of interconnected MEC servers, each of them offering different computing and memory resources to a set of running services, each having distinct and specific resource requirements. We assume that each MEC server's workload is modeled as a quadruple representing only four types of available resources: computation, communication, main memory, and permanent memory, whose amounts do not change over time. Since REACT redistributes resources among the services running on a single MEC server, we restrict our scope to a set S of running service instances on a single MEC server. We assume that the time in the system is divided into equal intervals called time slots, and the system produces a service resource reallocation during each time slot. REACT operates within a single time slot, so we assume that all the symbols introduced hereafter are related to a certain time slot $k \in \mathbb{N}$.

We define the *server background load* $\omega \in [0,1]^4$ as a quadruple that represents the resource load on the MEC server unrelated to running user services, e.g., OS overhead, scheduling, background, and monitoring processes, which cannot be auto-scaled. We define the MEC server load ξ as the sum of the background load ω and the total amount of resources allocated to all services running on MEC server. Equivalently, $\xi = \omega + \sum_{i=1}^{|S|} a_i$. It is worth noting that $\forall k \in \mathbb{N}, 0 \leq \xi \leq 1$, as the sum of the allocated resources for the services and the background processing on the MEC server can never exceed its maximum resource capacity.

MEC servers' resource utilization can be classified into three categories: light, medium, and heavy utilization. A MEC server is under light utilization if its $\xi \leq \tau_l$, where $\tau_l \in [0,1]$. Similarly, a MEC server is under heavy utilization if its $\xi \geq \tau_h$, where $\tau_h \in [0,1]$. If $\tau_l < \xi < \tau_h$, then the MEC server is under medium utilization. τ_l and τ_h represents 30% and 95% of the MEC server's capacity, respectively. The low and high thresholds will determine when REACT will trigger its solidarity approach. We consider that a MEC server is in a *resource scarcity* condition when its $\xi > 0.95$.

Every service $s_i \in S$ running on the system is characterized by a set of parameters, detailed hereafter. The *workload* of service s_i is indicated with $w_i \in [0,1]^4$, a quadruple in which each element represents the ratio between the service's current load and the MEC server's capacity for a specific resource type. The *resource allocation* of service s_i is indicated with $a_i \in [0,1]^4$, a quadruple in which each element represents the ratio between the amount of resources allocated for service s_i and the MEC server's capacity for a specific resource type. The *resource over-provisioning* of service s_i is defined as $o_i = a_i - w_i$, a quadruple in which each element represents the ratio between the amount of over-provisioned resources for service s_i and the MEC server's capacity for a specific resource type.

REACT classifies every service as either *donor service* or *recipient service*. A donor service d is defined as an over-provisioned service willing to transfer part of its currently unused resources to other services that need them. A recipient service r is defined as a VR service that

is currently under-provisioned and close to running out of resources, but is willing to accept resources from other donors.

REACT's solidarity approach considers that a set of recipients r, under resource scarcity conditions, are eligible for receiving resources from other over-provisioned donors d that run on the same MEC server. Donors scale-down parts of their over-provisioned resources to scale-up recipients. As long as services have residual resources, REACT can auto-scale recipients and avoid SLA violations. The computation performed by REACT to decide the amount of over-provisioned resources to transfer from a set of donors d to each recipient r is called *donation*.

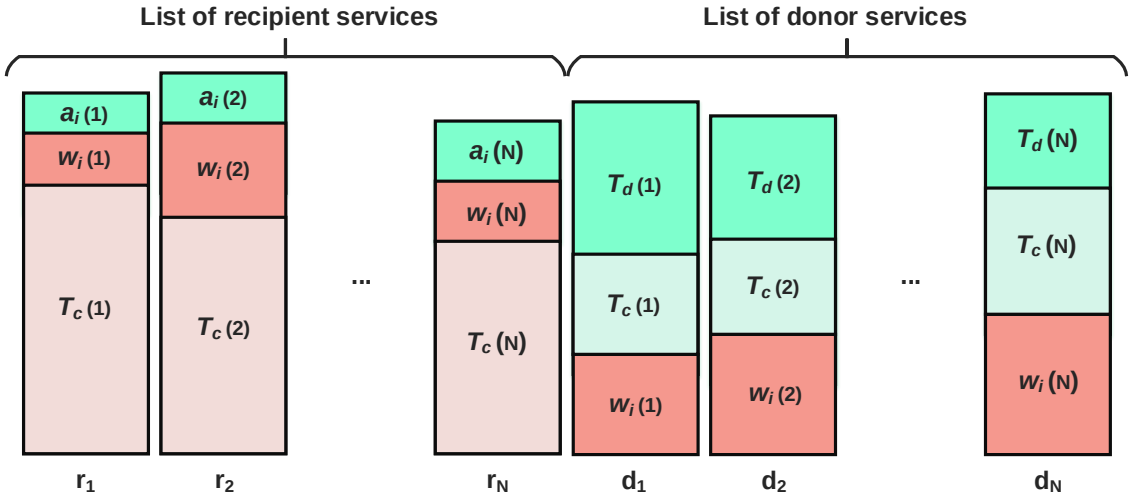


FIGURE 3.1: REACT System model.

The *committed service threshold* $T_{c(s_i)}$ is the minimum amount of resources needed by the service s_i to honor its SLAs. We define the *service donating threshold* as $T_{d(s_i)} = a_i - T_{c(s_i)}$ as the maximum amount of resources that service s_i can donate. $T_{d(d)}$ quantifies the part of the donor's over-provisioned resources o_d, aiming to scale-down donors and scale-up recipients. The expression for T_d is designed so that a donor d cannot donate more resources than what its SLA allows it when $w_d \leq T_{c(d)}$. Figure 3.1 shows the thresholds a_i, w_i, T_c, and T_d for each service in the system, where each variable is used to represent recipients r or donors d in the solidarity-based model.

Let us define q as a decision binary variable, where $q \in \{0,1\}$, assumes value 1 to perform scale-up and 0 to perform scale-down. The resource types that will be scaled up/down are denoted by $\gamma \in \{\gamma_1, \gamma_2, \gamma_3, \gamma_4\}$. The share of resources that will be scaled up/down is denoted as $z \in [0,1)$. The auto-scaling function for a service s_i represents the amount of resources that the service will either receive or donate and is denoted as $\beta(s_i, \gamma_i, q, z) = \gamma \cdot (1 + (2q-1)z)$. The total amount of resources exchanged in a *donation* from a set of donors $D' \subset D$ to a specific recipient $r \in R$ for a specific resource type γ can be defined as $\mu(r, D', \gamma) = \sum_{s \in D'} \beta(s, \gamma, 0, T_d(s))$. If the donation process involves a set of recipients $R' \subset T$ and a set of donors $D' \subset D$, then the amount of exchanged resources can be computed as:

$$\sum_{s \in R'} \left(\mu(s, D', \gamma) + \sum_{s \in D'} \beta(s, \gamma, 0, T_{d(s)}) \right) \tag{3.1}$$

The donation for a specific recipient r occurs until the sum of scale-down resources from a set of donors $d \geq T_{m(r)} \cdot 1.3$. The value of z for the i-th r in each donation procedure is set to 30%. Each donation adds 30% more resources than the current w_r in time slot k. We scale-up each $T_{m(r)}$ by 30% to avoid new donation requests in a short period. According to our analysis and the threshold practices adopted in [83], we chose 30% as the threshold. It mitigates over-provisioning and improves the time window in which the service will need another auto-scaling procedure. On the other hand, the value of z for the i-th d is set to its T_d. Hence, for any donation procedure, the property $\sum_{s \in D'} T_{d(s)} \geq T_{m(r)} \cdot 1.3$ holds. It is noted that each $T_{m(r)}$ is updated via μ. Thus, Equation 1 minimizes the over-provisioned resources in MEC servers and maximizes resource utilization. We want to maximize resource utilization as long as we can satisfy the elasticity demands and not violate SLAs.

Let us define $h_i = (w_i, a_i, o_i)$ as the *monitoring metrics* of the i-th service, i.e., the current values for its workload w_i, allocated resources a_i, and over-provisioned resources o_i. Each service monitoring metric h_i uses γ to denote the resources for service s_i, e.g., CPU, RAM, storage, and bandwidth. We can then define δ as the set of service workloads deployed in a generic MEC server, where $h_i \in \delta$. A MEC server uses δ to obtain the full-service status information, then $\delta = \sum_{i=1}^{n} h_i$, assuming that the server must check each service serially. In the considered scenario, we assume that the value of δ is updated periodically. The frequency with which δ is updated significantly influences REACT's behavior, as service monitoring is a crucial measure to determine whether the solidarity-based approach should be triggered.

3.3 Edge Resource Provisioning with REACT

This section describes the principles of REACT, its architecture, and how it operates, including the solidarity-based elasticity algorithm and its complexity analysis.

3.3.1 REACT Architecture

The efficiency behind an elasticity mechanism depends on the auto-scaling function. As edge services' requirements change over time, MEC servers will experience workload fluctuations. These workload fluctuations may result in either service over- or under-provisioning. When the load decreases, the most widely adopted reactive mechanisms will take some time to provide scale-down actions. On the other hand, auto-scaling mechanisms will scale-up and cause over-provisioning when the load increases. If resources are scarce, it will cause under-provisioning. The over-provisioning strategy reserves more resources than those needed by the service at a specific moment in time, aiming to avoid disruptions if the service requires an unexpectedly high amount of resources to support its operations in the future.

Over-provisioning demands careful deployment to prevent inefficient resource allocation. However, in situations where over-provisioned resources are low, reactive auto-scaling solutions tend to trigger several elasticity rounds until matching resource patterns to meet the new service workload, which increases the *elasticity attempt window*. We define the time needed for the auto-scaling procedure to converge and find a suitable resource allocation as *elasticity attempt window*. Even though this strategy will ensure that SLAs are not violated, it might reserve resources for services, which in turn may never use them. This would lead to inefficient MEC resource usage and unnecessary costs for the user to benefit from those MEC resources that do not positively impact the application's QoS. In under-provisioning, the allocated resources for a given service are less than the current load demand, which can cause SLA violations and service resizing penalties.

REACT provides an auto-scaling algorithm to efficiently reallocate resources among different services running on MEC servers under scarce resources. REACT solves the typical problems of reactive schemes, e.g., several auto-scaling rounds during resource scarcity situations, by re-orchestrating both networking and computational MEC resources. The main novelty of REACT, compared to other reactive resource elasticity mechanisms, is its solidarity-based resource reallocation, which defines how some resources are seized from a set of donors and transferred to a set of recipients when the system enters a resource-depletion state.

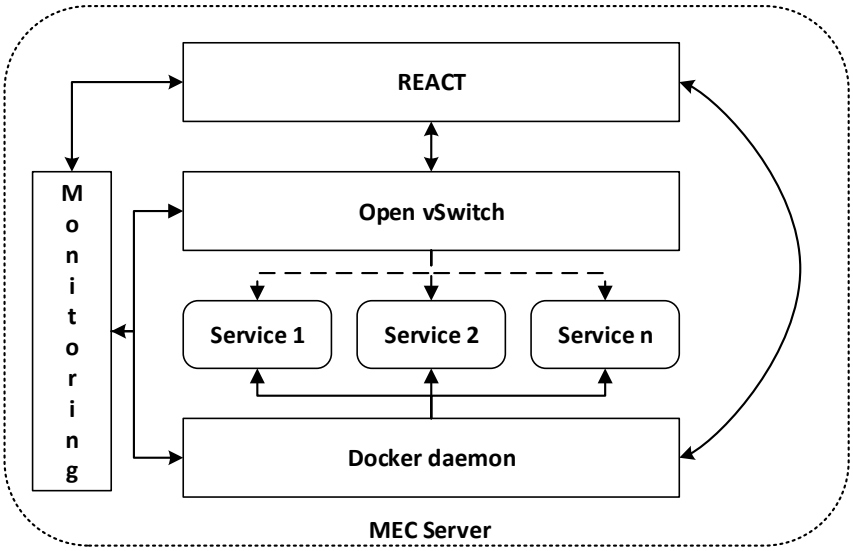


FIGURE 3.2: REACT Architecture

REACT is implemented as part of the auto-scaling component's logic without MEC architectural changes. Its solidarity-based model can be deployed in any platform that supports auto-scaling mechanisms, making REACT an agnostic solution to MEC servers. Figure 3.2 presents the REACT architecture, REACT uses the API of the monitoring system deployed on the MEC servers to obtain information about the VR services running there. Thus, REACT manages CPU and RAM resources for each service via the Docker API (*Docker daemon*). In addition, REACT also increases or decreases the bandwidth resources for each service via the *Open VSwitch*, where the flows of VR each service are managed independently.

REACT's solidarity-based elasticity takes advantage of services' resource over-provisioning to offer enhanced auto-scaling capability towards MEC efficient resource usage. In contrast, reactive solutions suffer from over-provisioning by needing successive attempts until matching the required resource amounts to the new service load when resources become scarce. It is worth noting that REACT can apply its solidarity scheme only if the MEC server is running over-provisioned services while the available resources in the system become scarce. REACT aims to mitigate service degradation due to the unavailability of resources in MEC servers and improve system efficiency by reducing over-provisioned resources. This resource reduction can also decrease the economic costs sustained by the VR users since cloud systems provide resources based on a *pay-as-you-go* pricing.

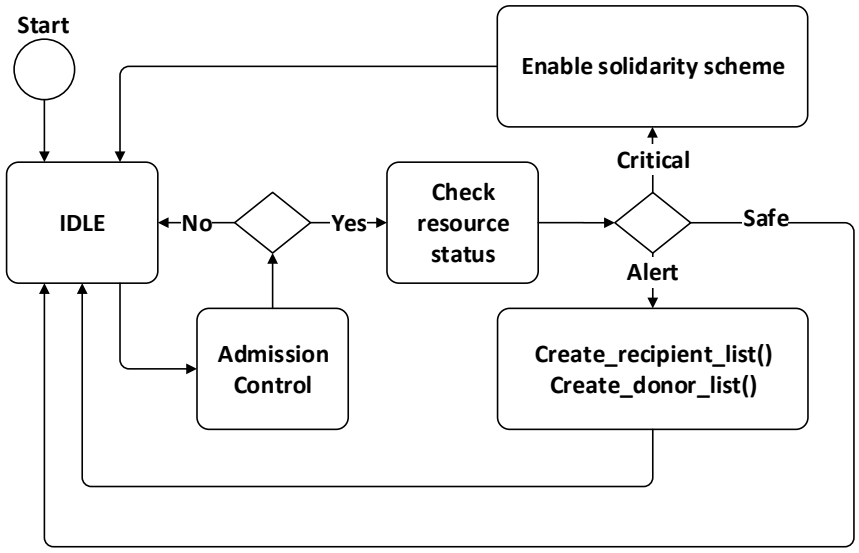


FIGURE 3.3: Conditions to enable the REACT solidarity approach.

REACT classifies a server's load into three conditions: *safe*, *alert*, and *critical*. Figure 3.3 illustrates the conditions for enabling the solidarity approach in a state diagram. In figure 3.3, *Safe* and *critical* conditions are mapped to τ_l and τ_h, respectively. The *alert* condition is enabled when the MEC server load ξ is between 80% and 95% of the MEC server's capacity. If the system is in a *safe* condition, REACT does not operate because services can be deployed immediately. When the system is in *alert* or *critical* condition, REACT takes preventive measures to reallocate resources and avoid the system entering or remaining in a *critical* condition.

REACT groups general services into a *donor list* D and VR services into a *recipient list* R, respectively. REACT adds a service s_i to the donor list if its workload $w_i \leq \mathrm{T}_c(s_i)$. The donor list and the recipient list are sorted from the smallest to the largest available residual resources and resource demands. REACT constantly maintains the recipient list and the donor list if the server reaches an alert or critical condition. Each donation involves a single recipient and one or more donors: after REACT calculates how many resources a single recipient needs, it will scale-down one or more donors and subsequently scale-up the recipient to fulfill its resource needs. REACT will start a donation process until either the R or the D is empty.

3.3.2 REACT Operation

On a generic MEC server, the REACT algorithm runs on a set of services S. First, REACT gathers the infrastructure and service monitoring data, e.g., CPU, RAM, storage, incoming and outgoing bandwidth, to create and maintain the recipient list R and the donor list S. We implement R and D as self-balancing binary search trees, i.e., *AVL tree*, aiming to optimize the solidarity auto-scaling algorithm. To access n service monitoring metrics h REACT uses δ. Then, both lists are inspected to meet the highest-priority services that experience resource bottlenecks. After this, REACT calculates the details of the service donations and updates the new a_r and a_d, respectively, in R and D. The next step is to update the service thresholds in both R and D lists deployed at the local MEC server. It can be implemented through a virtualization platform used to host the service components.

Algorithm 1 Recipient and donor service selection

Input: service_list: list
Output: R, D

```
1: function INSERTAVL(root, node)
2:     for s in service_list do
3:         if w_s ≥ t_c(s) then
4:             INSERTAVL(R, s)
5:         else
6:             INSERTAVL(D, s)
```

Algorithm 1 identifies services that are facing resource bottlenecks, i.e., R. Algorithm 1 also defines the function `InsertAVL`$(root, node)$ to insert nodes in an AVL tree (line2). Based on this algorithm, R and D lists are created and maintained by Algorithm 1. A service is classified as R if its workload $w_s \geq \mathrm{T}_c(s)$ (line 4). Algorithm 1 identifies services that can be part of the donation process provided by REACT. A potential D can be identified by inspecting service workload $w_s < \mathrm{T}_c(s)$ (line 5). In the end, R and D are sorted according to the resource needs and the number of residual resources available, respectively. Algorithm 1 is triggered before a *critical* resource condition has been reached and after the solidarity scheme is enabled.

Algorithm 2 is triggered as an infinite loop. Each iteration of Algorithm 2 requires getting the service and MEC monitoring metrics (line 2). *Critical* conditions can be identified by checking the MEC load (line 3). Every time a critical resource condition has been reached, the REACT approach is enabled. REACT builds and maintains both R and D through Algorithm 1 (line 4). In lines 5 and 8, the REACT algorithm defines functions `InOrder`$(root)$ and `ReverseOrder`$(root)$ to recursively iterate over R and D, respectively. On one hand, `InOrder`$(root)$ traverses the left *subtree*, visits the *root*, and traverses the right *subtree*. On the other hand, `ReverseOrder`$(root)$ traverses the right *subtree*, visits the *root*, and traverses the left *subtree*. Line 11 gets the required donation from a set of R. In lines 12 and 15, the algorithm gets the value of $\mathrm{T}_d(d)$. In lines 13 and 16, Equation (3.1) is used to re-orchestrate R and D. After the donation of $\mathrm{T}_d(d)$, the donor d is removed from D using function `Remove`(D) in line 17. The recursive function in line 8 is either triggered until the required donation is reached or when D is empty (line 18).

Algorithm 2 Solidarity-based auto-scaling

Input: service_list: list
Output: R, D

```
1:  function SolidarityAutoScaling
2:      MecMonitoring(δ, ξ)                                  ▷ starts δ and ξ
3:      while ξ ≥ τ_h do
4:          R, D ← ServiceSelection(service_list)
5:          function InOrder(R)
6:              if R is ∅ then return
7:              InOrder(R → left)
8:              function ReverseOrder(D)
9:                  if D is ∅ then return
10:                 ReverseOrder(D → right)
11:                 required_donation ← 1·a_R
12:                 if w_D ≤ T_c(D) then
13:                     T_d ← a_D − T_c(D)
14:                     Donation(R, D, T_d)
15:                 else
16:                     T_d ← a_D − w_D
17:                     Donation(R, D, T_d)
18:                 Remove(D)
19:                 if donations ≥ required_donation then return
20:                 ReverseOrder(D → left)
21:             ReverseOrder(D → right)
```

To prove the feasibility of implementing the REACT solidarity approach in real-time MEC servers, we provide a detailed algorithm complexity analysis. To give an accurate analysis, let us assume that: (i) n services are running on MEC server; (ii) n services are classified as donor (D) and recipient (R) services; and (iii) on average, the REACT solidarity scheme consists of 30% of R and 70% of D. Although n services are iterated/searched in line 3 with complexity $\mathcal{O}(n)$, lines 4 and 5 use *AVL tree insertion* function `InsertAVL`$(root, node)$, which has time complexity $\mathcal{O}(\log n)$. Since lines 4 and 5 of Algorithm 1 are not nested, we can derive that Algorithm 1 has time complexity $\mathcal{O}(n \log n)$.

Algorithm 2 gets MEC and service monitoring metrics in line 2 through function `MECMonitoring`(δ, ξ), which has time complexity $\mathcal{O}(n)$. Algorithm 2 uses a *while* loop in line 3 to enable the REACT solidarity model, where in each iteration, the MEC workload ξ is updated. Line 4 has time complexity $\mathcal{O}(n \log n)$ as it uses Algorithm 1. Within function `InOrder`(R), in line 8, the function `ReverseOrder`(D) has time complexity $\mathcal{O}(d)$ as it recursively iterates over D. Within function `ReverseOrder`(D), in line 17 the function `Remove`(D) performs $\mathcal{O}(1)$ as it already uses `ReverseOrder`(D) to find the node. Then, `Remove`(D) removes the donor d from D and performs the AVL rotations when needed. As R and D have a linear relationship with n and based on `ReverseOrder`(D) and `Remove`(D) algorithm analysis, in line 5 the function `InOrder`(R) has time complexity $\mathcal{O}(n^2)$ as it takes $\mathcal{O}(r)$ to recursively iterates over R, resulting int the product $\mathcal{O}(r) \cdot \mathcal{O}(d) \cdot \mathcal{O}(1)$ for searching in R, D, and removing from D.

For both `InOrder(R)` and `ReverseOrder(D)`, the comparisons during the search in each iteration, including unsuccessful search, are limited by the height of the AVL tree, which is $\mathcal{O}(\log n)$. As `InOrder(R)` and `ReverseOrder(D)` have to search all nodes, then both perform $\mathcal{O}(n)$. `InsertAVL`$(root, node)$ requires $\mathcal{O}(\log n)$ to lookup a service, plus a maximum of $\mathcal{O}(\log n)$ retracing levels on the way back to the *root*, which takes $\mathcal{O}(\log n)$. `Remove(D)` follows the same pattern of function `InsertAVL`$(root, node)$, which also has time complexity $\mathcal{O}(\log n)$ [136]. However, as it is used within `ReverseOrder(D)`, it already knows where the node is, just requiring $\mathcal{O}(1)$ to remove the node and perform the AVL rotations. As `MECMonitoring()`, `ServiceSelection()`, and `InOrder()` are not nested, the function `SolidarityAutoScaling()` has time complexity $\mathcal{O}(n^2)$. The REACT algorithm performs $\mathcal{O}(n^2)$ resource reallocation operations.

3.4 Experiment Setup

To assess their impact in handling elasticity events, both Kubernetes and REACT adopt the same elasticity approach to scale-up/down resources of MEC services. When a service reaches the resource utilization threshold of 70%, both mechanisms scale-up by 30% of the current service resource allocation. Otherwise, when the current service resource usage is $\leq$ 30%, they perform a scale-down of 20% of the allocated resources. These thresholds are commonly used in other approaches and considered as good practices for cloud computing [83]. If vertical elasticity cannot be achieved successfully, Kubernetes will ignore the elasticity event. In contrast, REACT triggers the solidarity elasticity mode.

To denote a MEC-like testbed, we design the testbed configuration as described in Figure 3.4.

The auto-scaling schemes have been implemented in an Openstack-based cloud platform, consisting of three Dell Power Edge servers, two external Dell PowerVault MD3800i that provide disk space of 20.6 TB in RAID 5, and a network backbone with 48x10 GbE-T ports and 80 Gbps backbone connection. We represent edge servers as virtual machines deployed on our MEC infrastructure. Each MEC server supports the deployment of several VR services, where these services belong to different VR users. The auto-scaling mechanisms are deployed independently in each MEC server and are unaware of other MEC server resources.

We compare the REACT algorithm against Kubernetes' auto-scaling algorithm. We provide an elasticity policy to trigger network elasticity events when resource utilization reaches 80% of reserved resources. We apply the Poisson distribution results in the OVS, allocating different bandwidth demands for each service. This feature is incorporated into Docker containers through OVS, where we set virtual tunnels for each container's virtual interface. Furthermore, we set QoS egress and ingress traffic shaping policies to ensure bandwidth limitations for each service deployed within Docker containers.

A set of 100 services is deployed in the edge server, including edge analytic services, VR services, and video services to provide dynamic behavior in a real environment. The edge server has 16 GB RAM, 8 vCPUs, and a 5 Gbit/s link. The client arrival times are modeled by

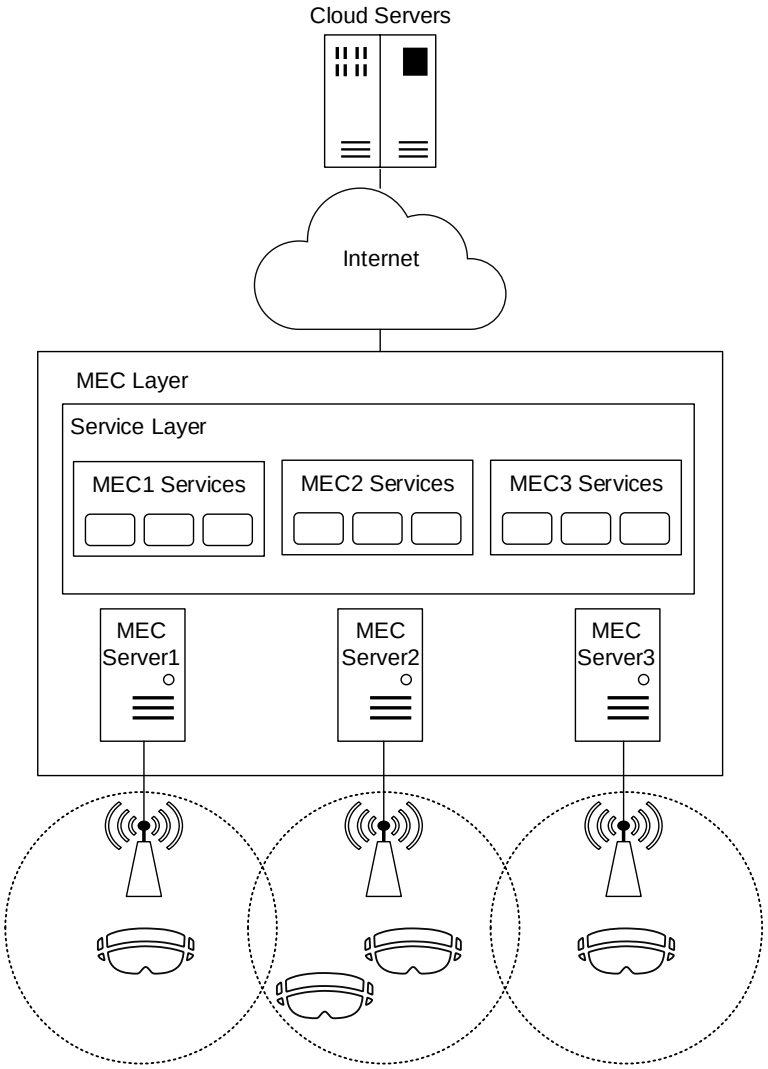


FIGURE 3.4: Testbed deployment for REACT and Kubernetes experiments.

a Poisson process for both REACT and Kubernetes. A Poisson distribution also models the elasticity time windows and service parameters such as workload, resource allocation, and over-provisioning.

We define the *elasticity time window* as the time required to trigger service elasticity events, i.e., an elasticity event is triggered at time slot k. In time slot $k+1$, another elasticity event is triggered. Then, the workload variations are triggered according to the *elasticity time window*. 1000 elasticity events are generated based on each service's Poisson distribution. Lastly, our evaluation considers that 1 vCPU represents 1024 CPU cycles per second. We use the docker flag `--cpu-shares` to control the CPU allocation priority.

To validate the approach presented in this chapter, we implemented a REACT prototype, available at [137] as open-source. The workload generated based on Poisson distribution allowed us to test REACT's and Kubernetes' algorithm performance after the MEC resources became scarce. All tests have been repeated along with 1000 elasticity events. Both REACT and Kubernetes are evaluated using the following KPIs:

1. ***Elasticity events accomplishment (KPI 1)*** measures both mechanisms' performance to accept elasticity events under resource scarcity conditions. Thus, auto-scaling requests can be denied if no resources are available.

2. ***Cumulative Distribution Function (CDF) (KPI 2)*** shows the cumulative acceptance ratio's behavior along with KPI 1 in the experiment. It shows how REACT can handle more auto-scaling requests than Kubernetes by using its service donation approach.

3. ***Service outages (KPI 3)*** measure the negative impact on services when resources become scarce. Moreover, this KPI shows how services could be either terminated or migration could be enabled due to scarcity of resources.

4. ***Elasticity attempts (KPI 4)*** are related to the algorithmic capacity to calculate new elasticity enforcement during resource scarcity conditions. A single auto-scaling request will count as one elasticity attempt if no resources are available. The mechanisms will then attempt to respond to the auto-scaling request until resources become available while elasticity attempts are counted.

5. ***Residual resource behavior (KPI 5)*** (over-provisioned) shows how over-provisioned resources are allocated during the experiments. Based on this, it is possible to understand how resource allocation could be enhanced whenever MEC resources become scarce. Besides, it identifies how service billings can be minimized while providing better MEC resource usage.

6. The ***time response (KPI 6)*** measures both mechanisms' performance to calculate and perform auto-scaling events.

3.5 Performance Evaluation

REACT and Kubernetes acceptance elasticity events rates (KPI 1) have been evaluated by measuring the number of events accepted after the hosting resources are saturated. Accepted events are related to both mechanisms' capacity to accomplish elasticity events, e.g., given an elasticity request, the mechanism can provide the auto-scaling provisioning action. In particular, Figure 3.5 shows the total accepted elasticity events by each resource type, i.e., CPU, RAM, and bandwidth. Kubernetes achieved an acceptance rate of 80'177 events. Based on this, 33.34%, i.e., 26'733, of the events were dedicated for CPU resources, 31.89%, i.e., 25'568 events, for RAM resources, and 34.77%, i.e., 27'876, for network resources. On the other hand, REACT achieved an acceptance rate of 98'848 elasticity events, where 33.56%, i.e., 33'168 events, for CPU resources, 33.02%, i.e., 32'644 events, for RAM resources, and 33.42%, i.e., 33'036 events, for network resources. REACT has accepted 18'671 more events than Kubernetes, which means a performance gain of 18.88% compared to Kubernetes. It is worth mentioning that the present evidence relies on REACT's capacity to accommodate more elasticity events through its solidarity approach.

We show the acceptance ratio of elasticity events in Figure 3.6 through a CDF (KPI 2). Also, Figure 3.6 combines all acceptance probability values, i.e., CPU, RAM, and bandwidth, and

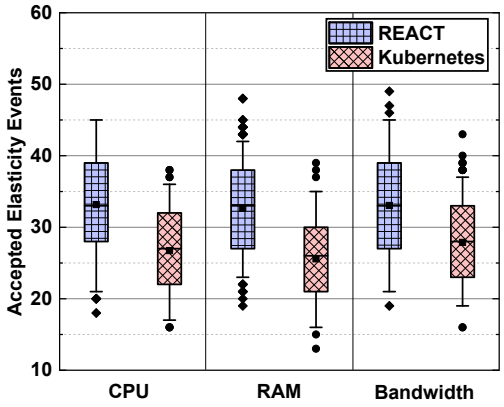


FIGURE 3.5: Impact of REACT and Kubernetes mechanisms to accomplish elasticity events throughout the testbed.

shows the cumulative probability of the elasticity events accepted by REACT and Kubernetes. REACT has a higher acceptance ratio due to its knowledge of over-provisioned resources. This feature avoids rejection events and increases the acceptance events ratio.

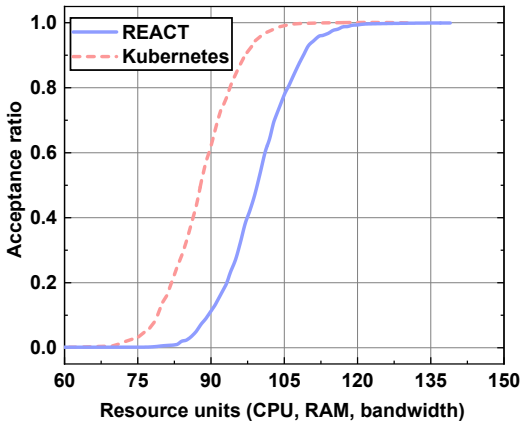


FIGURE 3.6: Acceptance ratio of elasticity events.

In containerization-based Docker, CPU is a compressible resource, i.e., the Linux kernel CPU scheduler can throttle containers if the requested amount is exceeded or the node is overloaded. Once a container reaches the limit, it will continue running. However, the operating system will throttle it and keep restricting it from using the CPU. On the other hand, it is important not to allow a running container to consume too much of the host machine's memory. By definition, RAM is a non-compressible resource. Once a container reaches the memory limit, it will be terminated because of the Out of Memory (OOM) problem, which means that the container's service will be killed. The same behavior occurs in REACT since Docker provides container virtualization for services. Kubernetes was designed to maintain the availability of the entire system. When the system goes into an over-committed state, the Kubernetes may decide to kill a set of pods to restore system stability. Generally, if a pod uses more resources than requested, that pod becomes a candidate for termination. On the other hand, REACT will try to use the residual service resources through its solidarity approach to minimize service outages and reduce service migration.

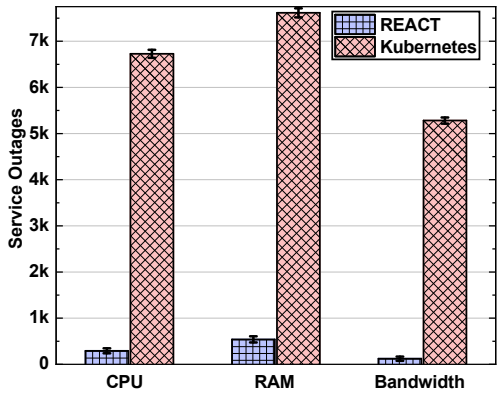


FIGURE 3.7: Influence of REACT and Kubernetes elasticity mechanisms in the testbed concerning service outages.

Figure 3.7 compares solutions in terms of *service outages* (KPI 3) during the experiments. A total of 19'626 service outage events were accomplished by Kubernetes' VPA mechanism, where 34.28%, i.e., 6'728 events, for CPU, 38.81%, i.e., 7'616 events, RAM, and 26.91%, i.e., 5'282 events, for bandwidth. Based on 1'000 elasticity events, on average, 7.616 elasticity events were affected by the OOM problem, which means that at least 8 services would have needed to be migrated to another server, totaling 8% of all services deployed. Furthermore, on average, 6.73% of CPU and 5.28% of RAM service resources were affected by the lack of resources. On the other hand, REACT accomplished 955 service outage events, equivalent to 4.85% of the total service outage events accomplished by Kubernetes. This means a reduction of approximately 95.15%, i.e., 18'671, of service outage events. For CPU, RAM, and bandwidth resources, REACT detected 293, 540, and 122 service outage events. With REACT, on average, 0.54% of services were affected by the OOM problem. At least 1 service would need to be migrated to another server, totaling 1% of all services. This fact indicates a reduction of 87.5% fewer services affected by the OOM problem than the Kubernetes. These findings support the notion that REACT is less influenced by the OOM problem and, consequently, by the enforced service migration. This implies that REACT is associated with smooth service interruption and prevents more services from becoming terminated or migrated.

Figure 3.8 shows the performance of both REACT and Kubernetes when the edge server achieves resource saturation, employing the averaging elasticity attempts analysis (KPI 4). When this state is reached, the schemes cannot serve all service elasticity requests. Then, they try to provide elasticity actions based on available resources in the edge server. REACT makes use of over-provisioned resources. During the resource scarcity situation, Kubernetes achieved 243'456 elasticity attempts, and 34.01% , i.e., 82'811 attempts, of these events were dedicated to CPU resources, 39%, i.e., 94'949 attempts, for RAM resources, and 26.9%, i.e., 65'696 attempts, for bandwidth resources. However, REACT achieved 11'280 elasticity attempts, reducing 95.36%, i.e., 232'176 attempts, compared to Kubernetes elasticity attempts. REACT distinguishes itself from Kubernetes by needing fewer resource re-orchestration rounds to assign resources for services during the scarcity of resources. It chooses a better resource configuration for all services to support more elasticity events.

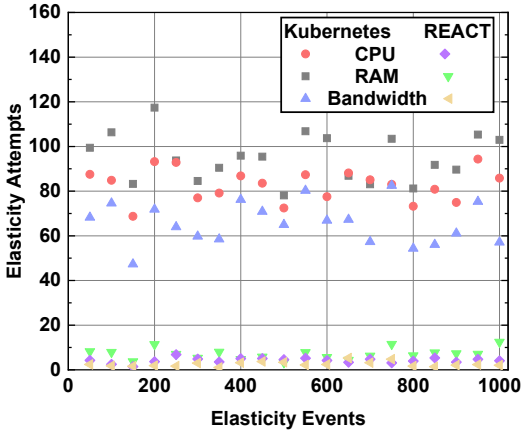


FIGURE 3.8: Elasticity attempts accomplished in the testbed due to the REACT and Kubernetes mechanisms.

We also examined the residual resources (KPI 5) for both REACT's and Kubernetes's mechanisms. Figure 3.9 shows the behavior of the residual resources of the mechanisms during the experiment events, considering a resource scarcity situation. Figure 3.9 shows the cumulative residual resources units. Kubernetes achieved an average of 2.41 vCPUs cores, residual CPU cycles, 4'985 MB of residual RAM, and 1'404 Mbps of residual bandwidth units. On the other hand, REACT achieved an average of 1.60 residual CPU cycles, 3'070 MB of residual RAM units, and 1'025 Mbps of residual bandwidth units. In this way, REACT performed an average gain of 33.88% of CPU residual resources, 38.41% of RAM residual resources, and 73% of residual bandwidth resources compared to Kubernetes.

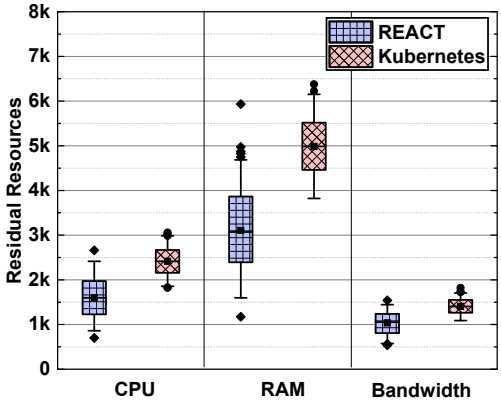


FIGURE 3.9: Effect in the residual resources led by REACT and Kubernetes elasticity mechanism on the testbed.

REACT's solidarity algorithm provides scale-down actions on residual resources of the donor list. Figure 3.10 outlines the residual resource behavior on the elasticity events in the two experiments. Therefore, REACT calculates the ratio between the currently used resources and the total resources reserved for each donor chosen. Then, REACT calculates the final amount of resources to shrink from the residual resources of the selected donor. REACT allows more efficient use of over-provisioning resources by using them more efficiently via the solidarity-based mechanism, where it takes advantage of over-provisioning.

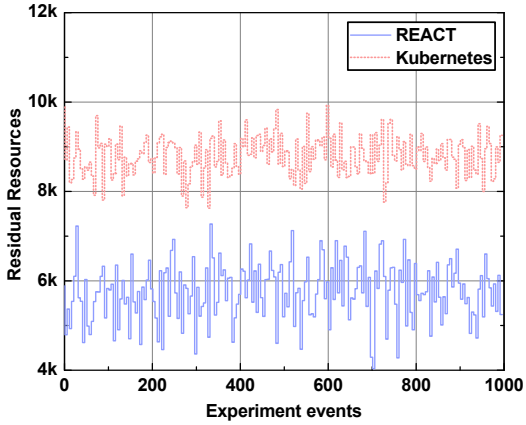


FIGURE 3.10: Cumulative residual resources behavior led by REACT and Kubernetes elasticity mechanism in the testbed.

Finally, to evaluate our REACT processing time, we compared the time needed to provide the elasticity actions and the time to provide the elasticity attempts when a resource scarcity situation is reached since both REACT and Kubernetes need to perform restricted actions to meet the current elasticity demand.

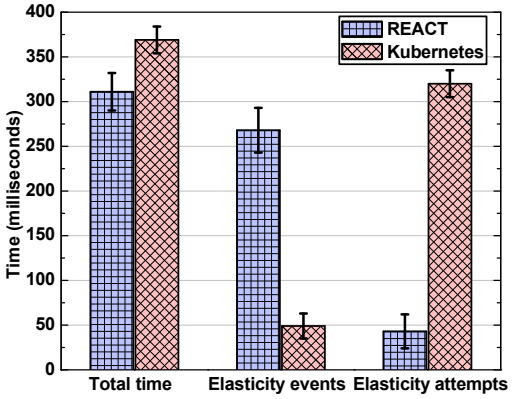


FIGURE 3.11: Processing time that REACT and Kubernetes take in the testbed to accomplish elasticity events.

Figure 3.11 shows the average time to process an elasticity request. Figure 3.11 also outlines the average processing time to accomplish elasticity events and elasticity attempts. Since Kubernetes performs fewer elasticity actions than REACT, its average processing is 49 ms, while REACT achieved 268 ms due to the solidarity actions. Regarding the *elasticity attempt window*, Kubernetes has an average of 320 ms compared to 19 ms of the REACT algorithm, considering that both mechanisms will try to perform elasticity events when resources become scarce. Lastly, the total average processing time, including elasticity events and *elasticity attempt window*, for Kubernetes is 369 ms, while REACT achieved 311 ms. REACT obtained gains in terms of processing time of 15.5% compared to Kubernetes. Indeed, the processing time is short, considering the order of magnitude of milliseconds. Hence, we demonstrate that REACT provides a response time as low as the Kubernetes algorithm.

3.6 Chapter Conclusions

This chapter proposes REACT, a self-adaptive elasticity solution for resource scarcity in MEC environments. We addressed the ***Research Question 1.1*** by exploiting how REACT uses a solidarity approach to provide resource reallocation of residual resources to prevent undesirable VR service degradation due to the scarcity of MEC resources. The new auto-scaling strategy of the REACT proposal distinguishes itself from widely-used reactive elasticity solutions in a three-manner: (i) optimal auto-scaling of both network-level (bandwidth) and compute-level (CPU, RAM, and storage) resources at network edges under resource-depletion conditions; (ii) efficient collaborative allocation of part of residuals within over-provisioned resources from a set of *donor* services to scale up a demanding *recipient*; and (iii), self-adaptive auto-scaling, which allows a high assertive resource computing scheme of *donor* services' resource residuals based on usage statistics.

REACT can minimize the harmful effects of service migration while keeping more services running over the same MEC server. We provided a detailed description of REACT, including the solidarity approach, the system model, and the REACT algorithm. We addressed the ***Research Question 1.2*** by considering a resource-constrained MEC scenario to evaluate REACT against Kubernetes. Our evaluation assesses both REACT's and Kubernetes's performance on a real testbed. Testbed results demonstrate the superior performance of REACT over Kubernetes in terms of accomplishing up to 18% more elasticity events, reducing service outages by up to 95%, reducing elasticity attempts by up to 95%, and reducing over-provisioned resources by up to 33%, 38%, and 73% for CPU cycles, RAM and bandwidth resources, respectively. Finally, REACT reduced response time by up to 15%.

In a nutshell, REACT, compared to Kubernetes, has the following improvements: (i) REACT is more agile than Kubernetes, having the ability to accommodate more elasticity events; (ii) REACT provides more resource reallocation procedures whenever the resources become scarce; (iii) REACT degrades fewer services, allowing services to remain active longer or prevent service migration; and (iv) REACT takes advantage of service over-provisioning, enhancing the residual resources. As a result, we enhanced the resource provisioning requests for *high-priority* VR services, thus addressing the ***Research Question 1.3***.

The findings and contributions in this chapter hold immense potential for advancing the capabilities of edge networks in supporting latency-sensitive VR applications. By effectively managing resources and prioritizing the deployment of VR services, the research presented in this chapter supports VR services deployed on edge networks while providing seamless, low-latency, and responsive VR experiences for VR applications. However, a distributed edge-enabled VR deployment must be considered, in which migration and offloading can change the resource availability of MEC servers during user mobility. Besides, the E2E latency must be ensured in such a scenario, where it is essential to minimize the overall E2E latency for VR systems that use the edge infrastructure to deploy their services. Therefore, in Chapter 4, we address the issues of edge-enabled 6DoF VR deployment, where we focus on reducing the overall E2E latency in a highly dynamic edge-enabled 6DoF VR environment.

Chapter 4

Orchestration of 6DoF VR Services

4.1 Introduction

VR has emerged as a transformative technology that immerses users in interactive digital environments, revolutionizing various fields such as gaming, education, training, and healthcare. The advent of 6DoF VR systems enhances the immersive experience to new levels, allowing users to move and interact within virtual spaces freely [5]. However, the seamless and surrounding 6DoF VR experience comes with demanding computational requirements, which often exceed the capabilities of VR HMDs [9].

To address the computational limitations of VR HMDs and ensure low latency and responsive experiences, the concept SFC to manage offloaded VR-intensive computing tasks to edge networks has gained significant attention [20]. Edge networks, powered by MEC infrastructure, offer proximity to end-users, reducing the latency for delivering VR services. By leveraging edge resources, VR applications can enhance their computing capacity and deliver high-quality experiences to VR users [21].

However, coordinating such a plethora of VR services, especially during user mobility, yields several challenges. In chapter 3, we have shown that merely prioritizing the deployment of VR services at the network edge does not guarantee ultra-low latency. The orchestration of VR services must consider several factors, such as user mobility, energy consumption trade-offs, and optimal distribution of VR services across edge nodes while supporting VR services with low-latency experiences.

In this chapter, we investigate the DSCP to find the optimal service placement of services from a service chain such that its E2E latency does not exceed 5 ms. The following sections discuss the content described in **Paper [2]** and **Paper [3]**. Therefore, Section 4.1 describes the research questions addressed in this chapter and discusses the contributions of the new heuristic to solve DSCP. Section 4.2 formulates the system model. Section 4.3 presents TENET, a new heuristic that solves DSCP and orchestrates VR services while providing low E2E latency. Section 4.4 describes the experiment setup. Section 4.5 discusses the evaluation results. Finally, section 4.6 concludes the study in this chapter.

4.1.1 Research Questions Addressed

The contributions of this chapter aim to answer the following research questions described in Section 1.3.2.

Research Question 2.1: How can VR services be distributed across the MEC infrastructure to reduce the E2E latency of VR applications?

Research Question 2.2: What is the trade-off between the VR application's E2E latency and the mobile HMD's energy consumption by adopting different strategies for offloading VR-intensive computing services from mobile HMDs to MEC infrastructure?

Research Question 2.3: How does the decision on where VR services are deployed impact the E2E latency, and how does it affect video resolution selection for VR systems?

4.1.2 Chapter Contributions

To address the challenges mentioned above, in **Paper [2]** and **Paper [3]**, we propose solutions to reduce the E2E latency for VR systems. We address the ***Research Question 2.1*** by showing how to distribute VR services across MEC servers to reduce the E2E latency for VR applications. We address the ***Research Question 2.2*** by proposing in **Paper [2]** an algorithm to analyze the trade-off between E2E latency and energy consumption for VR systems. We address the ***Research Question 2.3*** by proposing in **Paper [3]** the service chain orchestrator TENET, which supports offloading, migration, and orchestration of VR services deployed across HMDs and MECs to ensure acceptable E2E latency for MVR applications and optimize the selection of better video resolutions for VR systems. Our contributions are as follows.

- We provide the trade-off between E2E latency and energy consumption over three high-mobility scenarios compared to widely used service migration strategies.
- We define the DSCP to find the optimal placement of services from a service chain such that its E2E latency does not exceed 5 ms. We use integer linear programming to model DSCP objective and constraints.
- DSCP is $\mathcal{NP}$-hard, i.e., computationally expensive. Therefore, we propose a heuristic (TENET) that is one order of magnitude faster than DSCP. We also provide algorithms for path calculation based on E2E latency and management of VR applications to ensure acceptable E2E latency along with TENET architecture.
- We evaluate the performance of Meta HMD applications in terms of frame rate, computing latency, and power usage to model service workloads. We use those application metrics to model 6DoF VR service workloads in a simulated environment to evaluate system scalability, E2E latency, energy consumption, video resolution selection, context migrations, and execution time.
- We compare the TENET with traditional service migration approaches over high-mobility environments by analyzing the VR-AVS as a reference use case and show that TENET can guarantee acceptable E2E latency to a set of VR services over MEC infrastructures.

4.2 System Model and Problem Formulation

4.2.1 System Model

Our considered scenario contains a set of users, each provided with an HMD that executes a 6DoF VR application, e.g., VR games, educational tools, and navigation aids. We assume that each HMD can move around in the scenario at speeds ranging from pedestrians to vehicles and is always connected to the Internet via a 5G base station. The most challenging use case for this scenario is the VR-AVS, in which HMDs move at high speed and require low-latency video streaming.

The network infrastructure is defined as a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V} = \{v_1, \ldots, v_{|\mathbb{V}|}\}$ is a set of computing devices (i.e., MEC servers and HMDs), and $\mathbb{E} = \{e_1, \ldots, e_{|\mathbb{E}|}\}$ is the set of paths between any two elements of set $\mathbb{V}$. The set of HMDs is denoted by $\mathbb{H} \subseteq \mathbb{V}$. The maximum achievable data throughput between two elements belonging to set $\mathbb{V}$ along path e_j is indicated by B_j. The total computing resources offered by device $v_i \in \mathbb{V}$ are the maximum CPU cycles per second $C_i \in \mathbb{R}$ and the maximum GPU cycles per second $G_i \in \mathbb{R}$.

In our considered scenario, each computing device $v_i \in \mathbb{V}$ (i.e., MEC server or HMD) can execute several elementary functions, each implemented by an indivisible software module called *service*. All services operate according to the same general workflow: they take some data for input, process it, and finally output it. Examples of services that can be executed on a computing device are video encoding and decoding, FoV extraction, face tracking, body tracking, and mobility prediction. Let $\mathbb{F} = \{f_1, f_2, \ldots, f_{|\mathbb{F}|}\}$ be the set of all services. The set $\mathbb{F}_i \subseteq \mathbb{F}$ denotes the set of services deployed on the computing device $v_i \in \mathbb{V}$. The resources of the computing device v_i are shared among all services $f_m \in \mathbb{F}_i$ that are deployed on it, where the computing device grants and releases resources over time. We assume that each service $f_m \in \mathbb{F}_i$ requires *exclusive* use of a share of CPU and GPU resources provided by computing device v_i to operate correctly, meaning that the sum of all resources assigned by device v_i to its services cannot be higher than the total installed resources. The CPU and GPU cycles per second required to run a generic service f_m are denoted by

$$R_m = (R_m^c, R_m^g) \in \mathbb{R}^2 \tag{4.1}$$

The amount of CPU and GPU cycles per second allocated to a generic service f_m is denoted by

$$A_m = (A_m^c, A_m^g) \in \mathbb{R}^2 \tag{4.2}$$

To deal with service workload fluctuations, for each service f_m, it is required that

$$A_m \geq R_m \tag{4.3}$$

The output of a service can be redirected as the input of another service to perform a more complex task. Therefore, we define a *service chain* s_n as an ordered sequence of services, where the data produced by a service is the input of the following service, where some services

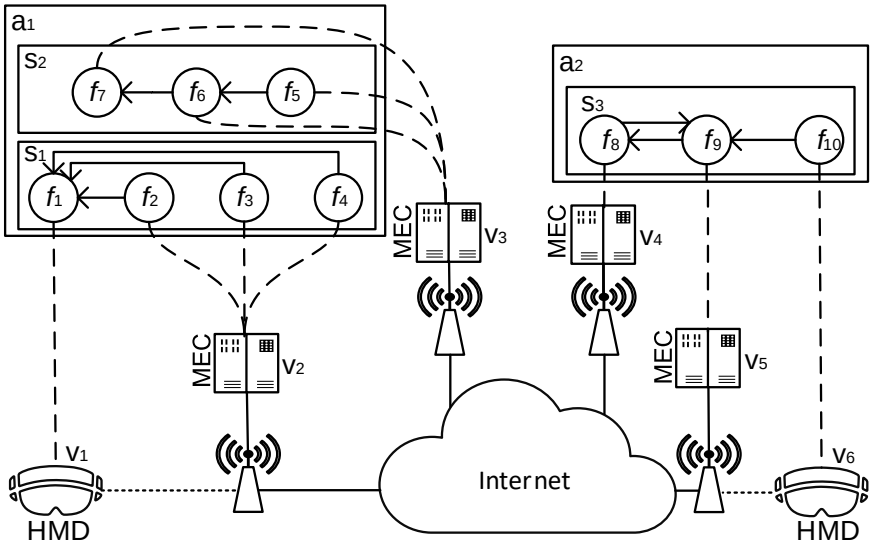


FIGURE 4.1: Service chain graph deployment on the network.

from a specific service chain may be shared among different applications, e.g., transcoding. However, we replicate the shared services if they need to be migrated. The first and last service of a chain have the task of producing and consuming the content, respectively. We define Service Chaining Graph (SCG) as the set of service chains in the whole system as $\mathbb{S} = \{s_1, \ldots, s_{|\mathbb{S}|}\}$. Each service f_m is associated with a single service chain s_n and cannot be shared by multiple service chains because a potential migration could reduce the network latency for one service chain and consequently increase the latency for another service chain. Each service f_m is associated with a service chain s_n and can be shared by multiple service chains. However, if the migration of that shared service f_m increases the E2E latency for other SFCs, we replicate that service f_m. As a result, any two service chains s_i and s_j are disjoint $\forall i, j \in \{1, \ldots, |\mathbb{S}|\}$. We define *allocation resource vector* $b_n \in \mathbb{V}^{|s_n|}$ of service chain s_n as a vector that indicates which computing device $v_i \in \mathbb{V}$ the corresponding service in the service chain s_n runs. We call $B = \{b_1, b_2, \ldots, b_{|\mathbb{S}|}\}$ the set of all allocation resource vectors (one for each service chain in the system) and B^* the set of all possible allocation resource vector sets. We define $\omega_n \in \mathbb{R}$ as the *maximum data throughput* needed between any two consecutive services of the chain s_n to communicate. We call $W = \{\omega_1, \omega_2, \ldots, \omega_{|\mathbb{S}|}\}$ the set of all maximum data throughput and W^* the set of all possible data throughput sets. Applications running in our considered scenario need to perform highly complex tasks. Therefore, we define each application a_n in the scenario as a set of one or more *service chains* whose services run in parallel on several computing devices. We denote $\mathbb{A} = \{a_1, a_2, \ldots, a_{|\mathbb{A}|}\}$ as the set of VR applications running in the system, one for each HMD. We define the set of service chains that belong to a certain application a_n as $\mathbb{S}_n \subset \mathbb{S}$, and we assume that service chains belong exclusively to one application and cannot be shared with others.

Figure 4.1 shows an example of service chain graph deployment on the network. The solid lines indicate wired connectivity, the dotted lines indicate wireless connectivity, and the dashed lines represent the set of allocation resource vectors. The example contains two VR

applications a_1 and a_2, each decomposed into service chains and highlights the allocation of each service on different computing devices in the system. A 6DoF VR application a_1 is implemented through two service chains $s_1 = (f_1, f_2, f_3, f_2, f_4)$ and $s_2 = (f_5, f_6, f_7)$, while application a_2 is implemented by a single service $s_3 = (f_8, f_9, f_{10})$. In the first service chain s_1 of application a_1, service f_1 represents a content aggregator that receives decoded video parts from services f_2, f_3 and f_4 and sends the VR video to HMD v_1. In the second service chain s_2 of application a_1, services f_5, f_6 and f_7 represent *mobility tracking*, *mobility prediction*, and *points of interest discovery*, respectively. In the service chain s_3 of application a_2, services f_{10}, f_9, and f_8 represent the VR services *decoding*, and *FoV extraction*, *FoV prediction*, respectively.

The frame rate of the video shown to the user is one of the most crucial QoS parameters of a VR application. Let us define σ_n as the number of frames per second generated by the application a_n and ϱ_n as the number of frames per second dropped by application a_n. We define the VR application QoS as the absolute number of frames per second correctly delivered to the HMD, which is represented by

$$\Theta_n = \sigma_n - \varrho_n \in \mathbb{R}, \forall a \in \{1, \ldots, |\mathbb{A}|\} \tag{4.4}$$

We assume that each HMD has limited energy resources and that their power consumption is proportional to the resources used by the services running on them. Let us define ϵ_m as the power required to run service f_m. We can then define the average system-wide power consumption Ψ per HMD as the sum of all power consumptions of services running on HMDs, divided by the total number of HMDs in the system, i.e.,

$$\Psi = \frac{1}{|\mathbb{A}|} \sum_{\{i:v_i \in \mathbb{H}\}} \sum_{\{m:f_m \in \mathbb{F}_i\}} \epsilon_m \tag{4.5}$$

It is worth noting that Ψ is a function of the allocation resource vector set B, as deploying services on either the HMD or the MEC server will change the energy expenditure of the system's mobile computing devices.

We denote the *computational latency* of service f_m as p_m, which is the computational execution time taken to run service f_m regardless of where it is deployed. We define the computational latency P_i of service chain s_i as the sum of the computational latencies of all services along the chain, i.e.,

$$P_i = \sum_{\{m:f_m \in s_i\}} p_m \tag{4.6}$$

Assuming that all service chains of application a_n run in parallel, we can now define the computational latency P_n of the application a_n as the maximum computational latency of all its service chains, i.e.,

$$P_n^* = \max_{\{i:s_i \in \mathbb{S}_n\}} P_i \tag{4.7}$$

Every service in a chain receives the information from the previous service, processes it, and forwards it to the following service in the chain. We denote the latency to transmit the data from a service f_m in the chain to the following service in the chain as k_m. In practice, the latency between consecutive services in a chain is equal to the network latency between the two computing devices that host the services or close to zero if the services are deployed on the same computing device. Therefore, we define the network latency K_i for service chain s_i as the sum of the network latencies between every two services along the chain, i.e.,

$$K_i = \sum_{\{m:f_m \in s_i\}} k_m \tag{4.8}$$

For the last service of service chain s_i, we assume $k_{|s_i|} = 0$.

Application a_n is implemented by a set of service chains $\mathbb{S}_n \subseteq \mathbb{S}$ that run in parallel. Therefore, we can now define the *network latency* K_n^* of application a_n as the maximum network latency of all its service chains, i.e.,

$$K_n^* = \max_{\{i:s_i \in \mathbb{S}_n\}} K_i \tag{4.9}$$

It is worth noting that K_n^* is a function of the allocation resource vector set B.

We define the *total E2E latency* L_n of application a_n conservatively as the sum of its network and computing latencies, i.e.,

$$L_n = K_n^* + P_n^*, \forall a \in \{1, \ldots, |\mathbb{A}|\} \tag{4.10}$$

Finally, we define the average system-wide E2E latency L as the average of the total E2E latency of all applications in the system.

$$L = \frac{1}{|\mathbb{A}|} \sum_{n=1}^{|\mathbb{A}|} L_n \tag{4.11}$$

4.2.2 Problem Formulation

Every service chain $s_n \in \mathbb{S}$ might be composed of several services f_m, where these services are distributed over different computing devices v_i. We introduce the *Distributed Service Chain Problem* (DSCP), a combinational optimization problem consisting of finding the optimal service placement of a service chain s_n composed of n services f_m such that the E2E latency of s_n does not exceed $\varphi_n = 5\,\text{ms}$.

To achieve such latency, we propose a service allocation algorithm to solve DSCP efficiently. Our proposed algorithm relies on the backtracking method, as the search space of service placement to meet the acceptable E2E latency is large and high-dimensional. With backtracking, the optimization procedure discards solutions whenever the latency exceeds the acceptable E2E latency. The defined DSCP can be solved by computing the values of

the specified utility function for all possible service allocations in the network and select the allocation that yields the highest utility as the solution. However, this approach is impractical due to the large search domain. In particular, each service $f_m \in s_n$ is independently deployable over a system that contains $|\mathbb{V}|$ devices. This means that, to find the globally optimal service allocation resource vector B for a single service chain s_n, the utility of all $|\mathbb{V}|$ possible service resource allocation combinations must be evaluated, which corresponds to a time complexity of $\mathcal{O}(n)^2$ function evaluations to optimize a single service chain deployment. This computation scales linearly with the set of all service chains $\mathbb{S}$ in the system, to make up an even larger computational load, which results in a time complexity of $\mathcal{O}(n)^3$. However, there are more combinations to be evaluated in the service placement process, for instance, the set of paths available $\mathbb{E}$, their throughput W and the network latency K_n^*, the computing latency P_n^* and resource availability of each computing device $v_i \in \mathbb{V}$. Therefore, an algorithm to solve DSCP has a time complexity of $\mathcal{O}(2^n)$.

Our objective is to compute an optimal allocation resource vector set B for all service chains in the system, which minimizes the total E2E latency and power consumption for all applications in the system while guaranteeing the acceptable QoS. Therefore, we introduce a *power sensitivity* coefficient $\alpha \in [0, 1]$ that the policy maker can set to a number closer to 1 to prefer lower latency over low power consumption and closer to 0 to prefer the opposite outcome. The coefficient α can be based on the user's and application's preference. To define the DSCP we use a cost function:

$$U = \alpha L + (1 - \alpha)\Psi \tag{4.12}$$

The cost function is minimized by exploring the set of all possible allocation resource vectors, subject to a set of network operation constraints listed in Optimization Problem 4.13.

$$\underset{B \in B^*}{\text{minimize}} \quad U = \alpha L + (1 - \alpha)\Psi \tag{4.13}$$

subject to

$$L_n \leq \varphi_n \quad \forall n \in \{1, \ldots, |\mathbb{A}|\} \tag{4.13a}$$

$$\Theta_n \geq \Delta_n \cdot \sigma_n \quad \forall n \in \{1, \ldots, |\mathbb{A}|\} \tag{4.13b}$$

$$\sum_{n=1}^{|\mathbb{S}|} \omega_n \cdot c_n(e_j) \leq B_j \quad \forall j \in \{1, \ldots, |\mathbb{E}|\} \tag{4.13c}$$

$$\sum_{\{m: f_m \in \mathbb{F}_i\}} A_m^c \leq C_i \quad \forall i \in \{1, \ldots, |\mathbb{V}|\} \tag{4.13d}$$

$$\sum_{\{m: f_m \in \mathbb{F}_i\}} A_m^g \leq G_i \quad \forall i \in \{1, \ldots, |\mathbb{V}|\} \tag{4.13e}$$

The cost function should be minimized while guaranteeing that the total E2E latency for each application a_n in the system is not higher than an upper bound φ_n defined for each

application (constraint 4.13a). To impose a sufficient QoS for immersive VR applications, every application a_n in the system must have a rate of video frames correctly delivered to the HMD of not less than a fraction Δ_n of the video frame rate σ_n generated by the application a_n (constraint 4.13b). Each path e_j between two computing devices has a maximum achievable data throughput of B_j, meaning that the total data throughput of all services communicating between the two computing devices connected by path e_j should be less than B_j. Let $c_n(e_j)$ be a function that counts how often the service chain s_n traverses path e_j. We now introduce a constraint for each path e_j in the system, formulated as follows: the sum of the throughput ω_n of all service chains s_n in the system, each multiplied by $c_n(e_j)$, should be less than the maximum achievable throughput B_j on path e_j (constraint 4.13c). For each MEC server v_i, the sum of the CPU resources A^c_m and GPU resources A^g_m allocated to all services running on it should not be larger than the total CPU resources C_i and GPU resources G_i installed on MEC server V_i (constraints 4.13d and 4.13e).

4.3 Managing Mobile VR Services with TENET

This section introduces TENET, a novel orchestrator to solve the DSCP. Furthermore, this section describes in detail the process of offloading VR services into service chains, the management of chain dependencies, the latency and energy trade-off, the path calculation to formulate the E2E latency, the orchestration of VR services, and the architecture of TENET.

4.3.1 Offloading VR Services

Typically, VR applications have inputs, processing services, and outputs. The processing services manage the inputs, e.g., cameras, gyroscopes, microphones, GPS, and compute specific services to produce the outputs. Among those services, *auxiliary* services, e.g., FoV prediction, motion prediction, scene depth estimation, image semantic understanding, and 3D scene reconstruction, enhance VR user experience. TENET identifies and offloads *auxiliary* services to alleviate the computation burden on VR HMDs. Nevertheless, services that may demand enormous processing power or high energy consumption can also be offloaded to the network edge, e.g., decoder or transcoding. By offloading VR-intensive computing services, VR HMDs only execute *mandatory* services and display the virtualized environment received from MEC servers. Hence, TENET provides high-quality immersive experiences by ensuring the acceptable QoS for VR applications. This deployment strategy enhances the QoS of VR users by increasing the battery life of HMDs and reducing the HMDs' heat while ensuring acceptable E2E latency for mobile VR users and preventing HMDs from running out of computing resources. On the one hand, VR-intensive computing services are offloaded to MEC servers, which prevents these tasks from being deployed at the cloud infrastructure. On the other hand, SCG reduces the amount of data transferred over the network and reduces the network latency for VR applications.

4.3.2 Managing Chains Dependencies

A VR application may have different service chains. Each service chain follows specific criteria to maintain the acceptable E2E latency. However, these VR services are not fully chained. Each service chain is isolated from other chains that belong to the same VR application to prevent latency bottlenecks in most priority services. This strategy allows TENET to deploy the most priority services with fewer dependencies, preventing failure in one service and decreasing the latency. One possible issue when offloading VR services is the dependency on the offloadable services of the VR application. Each VR application might be decomposed into *independent* VR services, i.e., without input from other offloaded services, such as *decoding* and *encoding* services. However, VR services with *mutual dependency* or even service chains with *mutual dependency* may coexist in the same VR application. A service with *mutual dependency* indicates that it needs input from other services. To mitigate the *service dependency problem*, we only consider VR services classified as a low priority to have a *mutual dependency*. Otherwise, the offloaded VR services should be *independent*.

4.3.3 Latency and Energy Trade-off Procedure

Algorithm 3 Latency and Energy Tradeoff

Input: s_n, h_i
Output: *E2E latency minimized*

1: **for** f_m **in** s_n **do**
2: $v_i \leftarrow$ DISCOVERMECS(h_i)
3: $L_n \leftarrow$ GETE2ELATENCY(v_i)
4: Initialize α ▷ $0 \leq \alpha \leq 1$
5: **if not** α **then return** MIGRATION(v_i, f_m)
6: **if** $f_m \in h_i$ **then**
7: **if** $v_i \neq \varnothing$ **and** $L_n < h_p$ **then**
8: **return** OFFLOADSERVICE(h_i, v_i, f_m)
9: $L_n \leftarrow$ GETE2ELATENCY(f_m) ▷ $f_m \in v_i$
10: **if** $v_i < L_n$ **then return** MIGRATION(v_i, f_m)
11: **return** REVERSEOFFLOADING(h_i, v_i, f_m)

Algorithm 3 shows TENET's latency and energy trade-off procedure. First, TENET discovers the MEC server $v_i \in \mathbb{V}$ to host a service f_m based on HMD location $h_i \in \mathbb{H}$ (line 2). v_i is discovered considering the E2E latency L_n (line 3). Algorithm 3 uses α to define the priority of latency over energy (line 4). The value of α can be derived according to each application's QoS requirement. The lower the latency is, the higher is the value of α. When α is configured, the service is deployed on HMD h_i to ensure acceptable latency at the cost of the battery. If $\alpha = 0$, the service is migrated to v_i. For each service f_m deployed on h_i, if v_i provides lower latency than h_i, then f_m is offloaded from h_i to v_i (lines 5-8). Otherwise, f_m is already deployed in the MEC infrastructure. If v_i has lower latency than the current MEC server hosting f_m, then f_m is migrated to v_i (line 10). Lastly, if there is no MEC server v_i to host f_m with the desired E2E latency, reverse offloading is performed to bring the f_m back to h_i (line 11).

4.3.4 Path Calculation based on E2E Latency

Algorithm 4 returns a path from the source node s to a destination node d based on the network and computing latency of each node available in graph $\mathbb{G}$. Algorithm 4 extends the original Dijkstra's algorithm by considering not only the weight of each path $e_j \in \mathbb{E}$ but also the cost to run service f_m on MEC server v_i. In each search, Algorithm 4 only considers network cost of path e_j to reach d and the computing latency p_m of d. We also optimize Dijkstra's searching by splitting the graph $\mathbb{G}$ into zones to search for a set of edge servers v_i to host a particular service f_m (line 4). Therefore, the two differences between this modified version of the Dijkstra Algorithm and its original version are the inclusion of computational latency p_m at each transversal node d and the partitioning of the graph into zones. We consider that the zones are uniformly created with the same size based on the geographical location of each city, which can consider neighborhoods or points of interest. The search starts in zone Z (line 5), which contains the base station where the HMD h_i is connected. If Algorithm 4 finds a server, the search stops. Otherwise, the next searching zone Z is provided considering h_i proximity, the direction of h_i's mobility, and the resource availability of MEC servers.

Algorithm 4 Extended Dijkstra's Algorithm

Input: $\mathbb{G}$: graph, s: vertex, L_n
Output: *dist*, *prev*

1: **for** vertex $v \in G$ **do**
2: $dist[v] \leftarrow \infty$, $prev[v] \leftarrow \varnothing$
3: $dist\ [s] \leftarrow s_p$ ▷ init $dist[s]$ with s' computing latency
4: $Z \leftarrow \mathbb{G}$ ▷ split $\mathbb{G}$ into zones Z
5: **while** $Z \neq \varnothing$ **do** ▷ search in $\mathbb{Z}$ where a_n is connected
6: $u \leftarrow$ EXTRACT-MIN(Z)
7: **for** each edge $e = (u, v)$ **do**
8: **if** $dist[v] > (dist[u] - w[u_p]) + w[e_k] + w[e_p]$ **then**
9: $dist[v] \leftarrow (dist[u] - w[u_p]) + w[e_k] + w[e_p]$
10: $prev[v] \leftarrow u$, **break if** $dist[v] \leq L_n$
11: **return** *dist*, *prev*

Figure 4.2 shows the zone scheme. u contains the vertex with a minimum distance value from Z (line 6). For each distance $dist[v]$ (line 7), the weight $w[e]$ of its adjacency nodes considers the network latency $[e_k]$ to reach node e and the computing latency $[e_p]$ to process service f_m in node e (line 8). Moreover, *dist* contains the current distances from s to other vertices (line 9), and *prev* contains pointers to previous-hop nodes on the shortest path from s to the given vertex (line 10).

4.3.5 Ensuring Acceptable E2E Latency for Mobile VR Services

Algorithm 5 describes the practical implementation of TENET. We shuffle the order in which services are processed in each iteration of Algorithm 5 to ensure fairness for all services during their processing. First, TENET discovers the information of each service chain s_n (lines 1-3). The next step is to iterate over all services and get the E2E latency of each service to evaluate if a particular service needs to be migrated or be redeployed on the HMD (lines 7-9). Then,

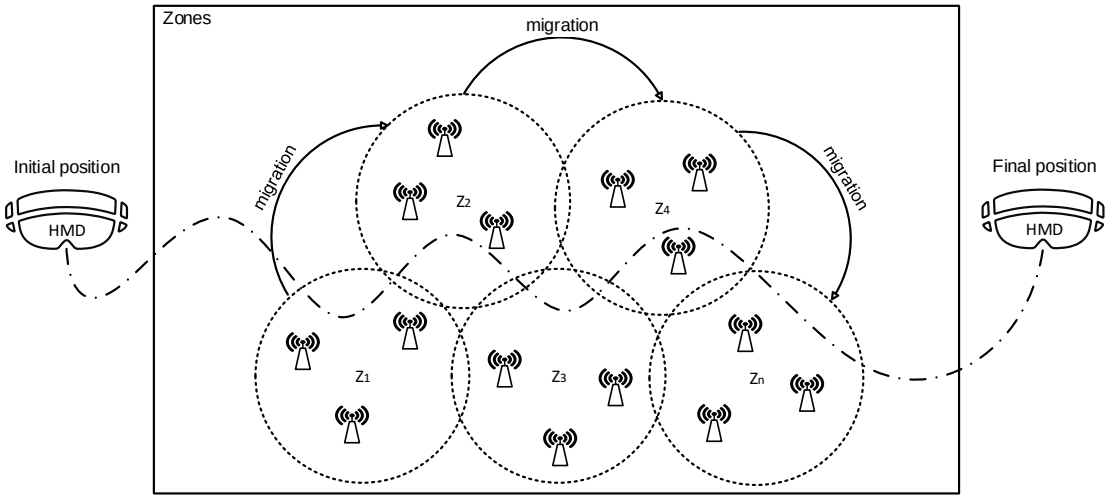


FIGURE 4.2: TENET's zones scheme.

TENET constructs the path ρ_n, allocates the bandwidth B_j, and deploys the services (lines 4-6). Whenever the h_i location changes, the algorithm checks whether the E2E latency L_n has increased (line 10). If so, the shortest path is calculated using Algorithm 4 and a new MEC server v_i is discovered to host service $f_m \in s_n$ (lines 11-15). If the current E2E latency $L_n > \alpha$ (line 16), reverse offloading brings the service f_m back to h_i (line 17). Otherwise, the service f_m is migrated to a nearby MEC server v_i (line 18). Compared to DSCP problem that has an exponential worst-case complexity of $\mathcal{O}(2^n)$, the cyclomatic complexity of Algorithm 5 is equivalent to that of the Dijkstra's Algorithm, namely $\mathcal{O}((\mathbb{V}+\mathbb{E})log(\mathbb{V})) = \mathcal{O}(\mathbb{E}log(\mathbb{V}))$.

Algorithm 5 SCG management for VR applications

Input: h_i, α, $\mathbb{F}$
1: **for** f_m **in** $\mathbb{F}$ **do**
2: **if** $f_m \in h_i$ **then**
3: $s_n \leftarrow \mathbb{F} \cup \{f_m\}$
4: $\rho_n \leftarrow$ *construct_path*(s_n)
5: $B_j \leftarrow$ *allocate_bandwidth*(ρ_n)
6: SERVICEDEPLOYMENT(h_i, s_n)
7: **while** True **do**
8: **for** $f_m \in s_n$ **do**
9: $L_n \leftarrow$ GETE2ELATENCY(f_m)
10: **if** h_i *location* changed **and** $L_n > \alpha$ **then**
11: *dist*, *prev* $\leftarrow$ GETSHORTESTPATH(s_n, f_m)
12: **while** *dist* $\neq \varnothing$ **and** *dist* $> \alpha$ **do**
13: $v_i \leftarrow$ DISCOVERMEC(h_i)
14: EXTRACTSERVICE(f_m)
15: EXTRACTNODE(*dist*, *prev*)
16: **if** $L_n > \alpha$ **then**
17: **return** REVERSEOFFLOADING(h_i, f_m)
18: **return** SERVICEMIGRATION(v_i, h_i, s_n, f_m)

4.3.6 TENET architecture

To achieve the visions of the TENET, we developed an architecture to be deployed in the MEC servers and VR HMDs. Figure 4.3 describes the TENET framework architecture. The main features of the TENET architecture are QoS analysis, migration of E2E latency, offloading, migration, and orchestration of edge resources. The architecture is composed of the TENET controller, the TENET VR agent, and the TENET MEC agent. Additionally, we consider a SDN controller to manage the network resources to ensure the acceptable latency for MVR applications. In the following, we describe the TENET architecture in detail.

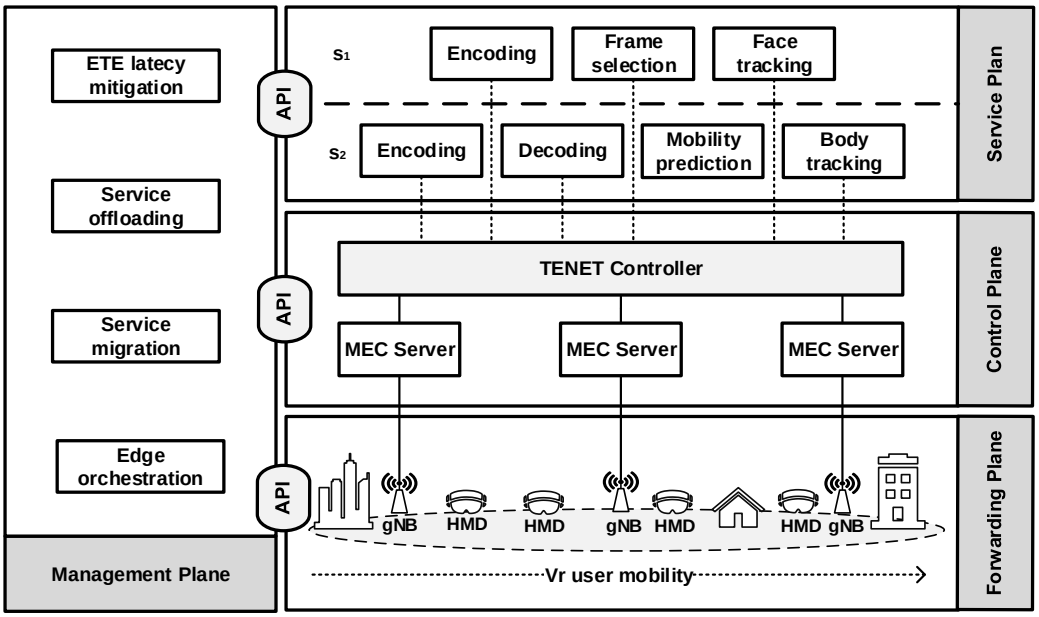


FIGURE 4.3: TENET architecture.

1. **TENET Controller** prepares the deployment by discovering the nearby MEC servers to offload VR services. Before the VR service offloading, the TENET controller requests the computing resources and bandwidth allocation to the TENET MEC agent and SDN controller, respectively. Furthermore, the TENET controller identifies whether a service migration must be performed whenever the user is in mobility.

2. **TENET VR Agent** is implemented onto VR HMDs, which interacts with the TENET controller by sending a set of services offloaded to the MEC infrastructure. The TENET VR agent chooses which services will be offloaded and prioritizes each service during this offloading process. To provide the refactoring process for VR services deployed on VR HMDs, the TENET VR agent prioritizes the services that should be offloaded according to its latency requirements.

3. **TENET MEC Agent** checks the resource availability at the MEC servers and allocates computing and network resources for VR services. The TENET MEC agent provides the resource allocation for VR services in MEC infrastructures via *REACT* [1]. *REACT* is a solidarity-based elastic service resource allocation strategy for service deployment over MEC servers with service prioritization support.

4.4 Experiment Setup

4.4.1 Testbed Configuration

First, to set the simulation parameters to realistic quantities, we perform an energy and latency benchmark on commercial devices in a real VR testbed composed of a Meta Quest 2 VR HMD (Qualcomm Snapdragon XR2 Platform CPU, Qualcomm Adreno 650 GPU, and 6 GB RAM) connected to a MEC server (Intel Core i9-10885H, 32 GB RAM, NVIDIA RTX 3000). The VR HMD and the MEC server are bridged by an access point, which simulates the role of the 5G Radio Access Network (RAN) access point. The access point is a TP-Link Archer AX6000, which supports Wi-Fi 6 (802.11ax) with a transmission rate of 4.8 Gbps at 5 GHz.

To get Meta HMD monitoring metrics, we use the OVR Metrics Tool, which provides performance information about a running application. OVR provides access to the information from an on-device application rather than the command line. After each session, the data will be stored in a CSV file on Meta HMD. To install the OVR metric tool on Meta HMD we use Android debug bridge, which is included in the Android software development kit. Based on the data extracted from Meta HMD's applications, we model the workloads for each VR application.

4.4.2 VR Application and Service Workloads

Since we cannot refactor Meta HMD applications into services, we estimate the realistic wireless link latency and the realistic average power needed for running a service on our HMD through the following benchmarking process. We deploy a video decoding service on our HMD and stream 360^o videos from a MEC server to the HMD for 600 s. During the video streaming, the HMD measures its total power consumption through on-board sensors and measures the latency to receive and decode videos. We repeat the benchmark five times and average their results for each of four video resolutions, namely 1080p, 1440p, 4 K, and 8 K running at 60 FPS. When no service is running on the HMD (*standby* mode), the consumed energy is 720 J over 600 s, which means an average power of 1.2 W.

We can now define the power needed to run a decoding service on the HMD as the difference between the measured power and the standby power. The outcome of the energy benchmark process is that the average energy consumption required by a video decoding service for 1080p, 1440p, 4K, and 8K resolutions are 978 J, 1014 J, 1272 J, and 2568 J over 600 s, respectively, which correspond to an average power consumption of 1.63 W, 1.69 W, 2.12 W, and 4.28 W. The realistic latency and power consumption measured in the benchmarking process are used as parameters of the simulation described hereafter.

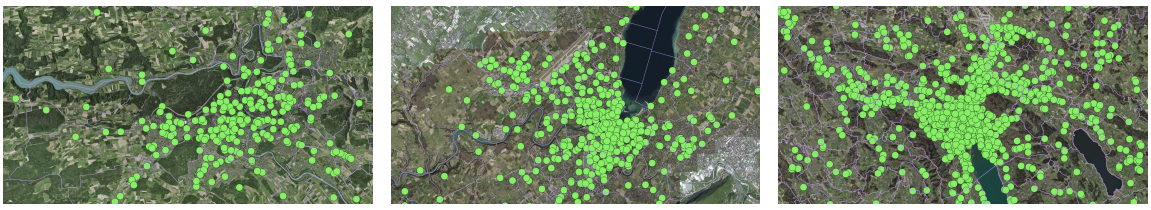
4.4.3 VR Users Mobility

We use Mininet-WiFi to simulate a realistic network scenario and user mobility. We use ONOS[1] SDN controller to provide flow control, bandwidth allocation, and mobility

[1] https://opennetworking.org/onos/

management for the simulated VR services. The simulated scenario covers the area of the cities of Bern, Geneva, and Zurich. Besides, each network topology contains a variable number of mobile VR users that can connect to the RAN via their 5G interface. We assume that each VR user runs exactly one VR application. The base stations transmit signals with a 50 dBm power, decaying according to the Free Space Path Loss model.

The VR users' mobility follows the Random Direction Model, in which users move along a straight line with a constant speed selected from a uniform distribution with a 0.1 meters per second average. We assume that mobile VR users connect to the base station whose signal is received with the highest Signal-to-Noise Ratio (SNR). We assume that each VR user executes a single 6DoF VR application made of decoding services with a power requirement as assessed in the real-testbed benchmark. For each 6DoF VR application we uniformly distribute between 3 and 10 decoders to observe how different sizes of service chains affect system performance. Furthermore, each 6DoF VR application contains a service to aggregate the chunks of VR video decoded by each decoder service. For each service f_m in the system, its equivalent requirements in terms of CPU (i.e., R_m^c) and GPU (i.e., R_m^g) are randomly extracted from two uniform distributions with averages of 1770 MHz for the CPU and 440 MHz for the GPU, based on the typical requirements of Meta HMD applications.



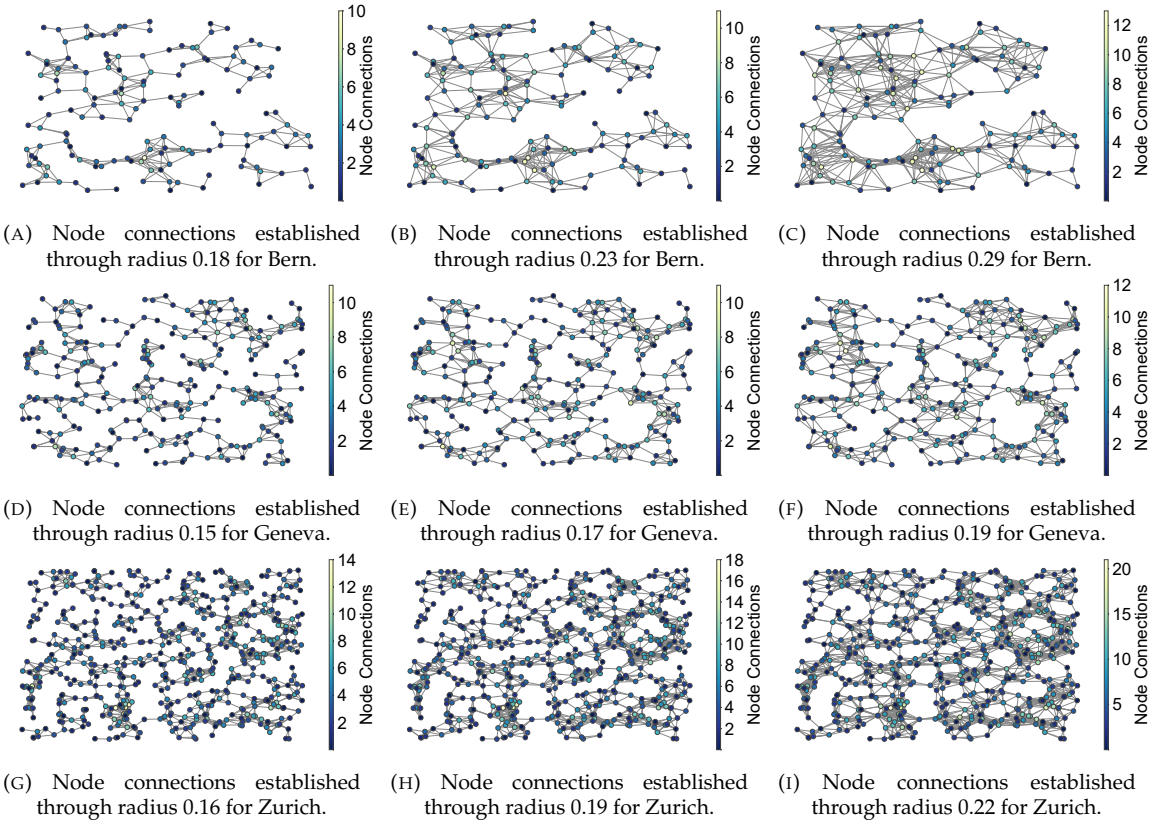
(A) Bern 5G base stations. (B) Geneva 5G base station. (C) Zurich 5G base stations.

FIGURE 4.4: Physical 5G network infrastructure map of the cities of Geneva, Bern and Zurich.

4.4.4 Edge Network Graphs

We use different types of edge network topologies for the simulation. We use real 5G edge network topologies for three cities, Bern (BE), Geneva (GE), and Zurich (ZH) [138]. The original 5G network infrastructures are shown in Figure 4.4. Geneva has an area of 15.93 km^2 with 269 nodes and a node density of 16.88 nodes/km^2. Bern has an area of 51.6 km^2 with 147 nodes and a node density of 2.84 nodes/km^2. Zurich has an area of 87.88 km^2 with 586 and a node density of 6.66 nodes/km^2.

Each generated topology is based on a cartesian plane, where the nodes are distributed between the coordinates $(0, 0)$ and $(1, 1)$. We define the area of coverage of each base station as radius r. Therefore, if the coverage area between two base stations overlaps, then we generate a link between them. The links between base stations are established whenever the Euclidean distance between any two base stations in the scenario is not greater than a radius r. The latency of each established link between two base stations is uniformly distributed between 0.5 ms and 1 ms. In each city, the base stations are located at positions



(A) Node connections established through radius 0.18 for Bern.

(B) Node connections established through radius 0.23 for Bern.

(C) Node connections established through radius 0.29 for Bern.

(D) Node connections established through radius 0.15 for Geneva.

(E) Node connections established through radius 0.17 for Geneva.

(F) Node connections established through radius 0.19 for Geneva.

(G) Node connections established through radius 0.16 for Zurich.

(H) Node connections established through radius 0.19 for Zurich.

(I) Node connections established through radius 0.22 for Zurich.

FIGURE 4.5: Generated 5G network infrastructure connectivity of the cities of Bern, Geneva, and Zurich over different radii.

illustrated in Figure 4.4. We define the aforementioned latency distribution according to latency measurements carried out in the University of Bern's local network infrastructure. In our scenario, 70% of the base stations of each topology are directly attached to MEC servers, which offer different GPU, CPU, memory, storage, and bandwidth resources. Around 80% of the MEC servers have a GPU. Figure 4.5 shows the generated topologies and their links.

4.4.5 Performance Metrics

The performance of Meta HMD is evaluated by executing it in the simulated scenario for 10 hours and measuring the average E2E latency and power consumption for the user applications. We assume that time is partitioned in a series of consecutive time windows of duration $T = 5\,\text{s}$, and we will measure a value of latency and energy per window. This choice for T gives sufficient time for our optimization algorithm to converge and is so that the experiment yields 3000 measurements for latency and energy over the 10 simulated hours.

The *average E2E latency* L is computed as follows. During a time window, each user executes an ICMP `ping` command along the core-network part of the service chain and uses the collected data to compute the average core network latency. During the same time window, for each user, we measure the average computing latency as the sum of the computing latency of each of its services deployed on MEC servers or HMD. Each user has an average E2E latency for that time window, which is the sum of the average core network latency, the average

computing latency, and the benchmarked wireless latency. The average E2E latency is the average E2E latency for the time window across all users. The average E2E latency L value is the average across all time windows of the window-based average E2E latency.

The *average power consumption per user* Ψ of VR HMDs is computed as follows. The window-based average power consumption is the product between the average number of services running on an HMD in the system during the time window and the benchmarked power consumption of a service corresponding to each user's selected video resolution. The value of the average power consumption Ψ is the average across all time windows of the window-based average power consumption.

The *video resolution selection* is performed as follows. We assume that each application in the system selects a video resolution based on the E2E latency among those we benchmarked, according to the average latency at each time window. The application maintains the resolution constant for the whole window duration. In the next time window, the resolution is selected according to the available E2E latency provided by the system. Therefore, the higher the resolution is, the more power and lower latency are required to process the video stream set to this resolution.

The *average acceptance and rejection ratio of service context migrations* measure the performance of each algorithm to find suitable MEC servers to either offload from HMD to a particular MEC server or to support the application context migration between MEC servers. We do not consider the migration of the entire software stack that supports a VR service, e.g., Virtual Machine (VM) or container. Instead, we consider that the VR application context migration, e.g., VR video streaming, is migrated between MEC servers. Then services depending on that context, e.g., decoder, depth estimation, image semantic understanding, 3D scene reconstruction, are enabled in advance in the target MEC server.

The *execution time* measures the time each algorithm takes to compute the decision on where the service has to be placed, which does not include any additional step, e.g., context migration time, time to enable services on the target MEC server, time to get E2E latency. This metric is highly impacted by the *average rejection ratio of service context migrations* since the more migration requests are rejected, the more time is needed to exploit an alternative solution.

4.4.6 Service Migration Algorithms

We compare the average latency, latency over time, energy, video resolution selection, accepted and rejected migrations, and execution time performance of TENET with DSCP implementation and those of three widely used solutions, which provide service migration among MEC servers under rapidly changing user mobility conditions, detailed hereafter [24]. It is worth noting that the video resolution selection is derived from the E2E latency provided by each algorithm during the VR user mobility.

Computing latency for HMDs		
Minimum (p_m)	**Average (p_m)**	**Maximum (p_m)**
0.005 [s]	0.0075 [s]	0.01 [s]
Computing latency for MECs		
Minimum (p_m)	**Average (p_m)**	**Maximum (p_m)**
0.003 [s]	0.004 [s]	0.005 [s]
Link latency for all topologies		
Minimum (k_m)	**Average (k_m)**	**Maximum (k_m)**
0.005 [s]	0.0075 [s]	0.01 [s]
Bern topology		
Radius (r)	**Users (u)**	**Average links per vertex (μ)**
0.18	1000	2.19
0.23	2000	3.36
0.29	3000	4.79
Geneva topology		
Radius (r)	**Users (u)**	**Average links per vertex (μ)**
0.15	2500	2.49
0.17	3500	3.02
0.19	4500	3.64
Zurich topology		
Radius (r)	**Users (u)**	**Average links per vertex (μ)**
0.16	5000	3.25
0.19	5500	4.28
0.22	6000	5.5

TABLE 4.1: Simulation parameters.

1. **DSCP-Optimal (DO)** provides a service migration strategy based on DSCP implementation, always aiming to find the optimal service placement of VR services, analyzing all deployment possibilities to achieve the lowest E2E.

2. **Network Latency Awareness (LA)** provides a service migration strategy based on network latency awareness. LA considers the base station to which the user is connected and the nearby MEC server with lowest latency. LA implements a method to discover candidate MEC servers to host the migrated service.

3. **Network Latency and Resource Awareness (LRA)** supports all features provided by LA. However, LRA can identify the optimal MEC server with lower network latency to host a VR service considering the resource availability of the selected MEC server.

4. **Always Migrate (AM)** considers the VR user's location to enable migration. The user's handover triggers this strategy. The service is always migrated to the MEC server attached to the base station where the user is connected. Unlike LA, AM is consistently restricted to the MEC server attached to the base station where the VR user is connected.

4.4.7 Simulation Parameters

For each topology described in Section 4.4.4, we choose a different radius r to increase the network topology connectivity, impacting the number of congested links and, consequently, the network latency. The radius selected for the experiments is chosen as follows. The minimum radius r for each topology is defined according to the smallest radius r possible to generate a connected graph. The maximum radius r for each topology is determined based on the analysis that a higher value than the maximum radius r does not provide a lower E2E latency performance in the experiments. Therefore, a topology running with maximum radius r has lower latency than the same topology running with minimum radius r. The higher the radius r is, the more VR users are considered for that scenario because more paths are available with less congested links, which improves the network latency. However, the increased number of users impacts the available resources in both network and MEC servers. All simulation parameters are described in Table 4.1.

4.5 Performance Evaluation

To validate the approach presented in this chapter, we implemented a prototype of TENET, available at [139]. Our evaluation focuses on two major sets of results. We first assess the QoS for both Echo VR and Elixir[2] games and take their workloads as a baseline to model service workloads used in TENET evaluation. Second, we provide a simulated environment to assess the capability of TENET to manage several VR services in a distributed edge environment, where each service has different requirements and workloads.

4.5.1 Meta HMD Evaluation

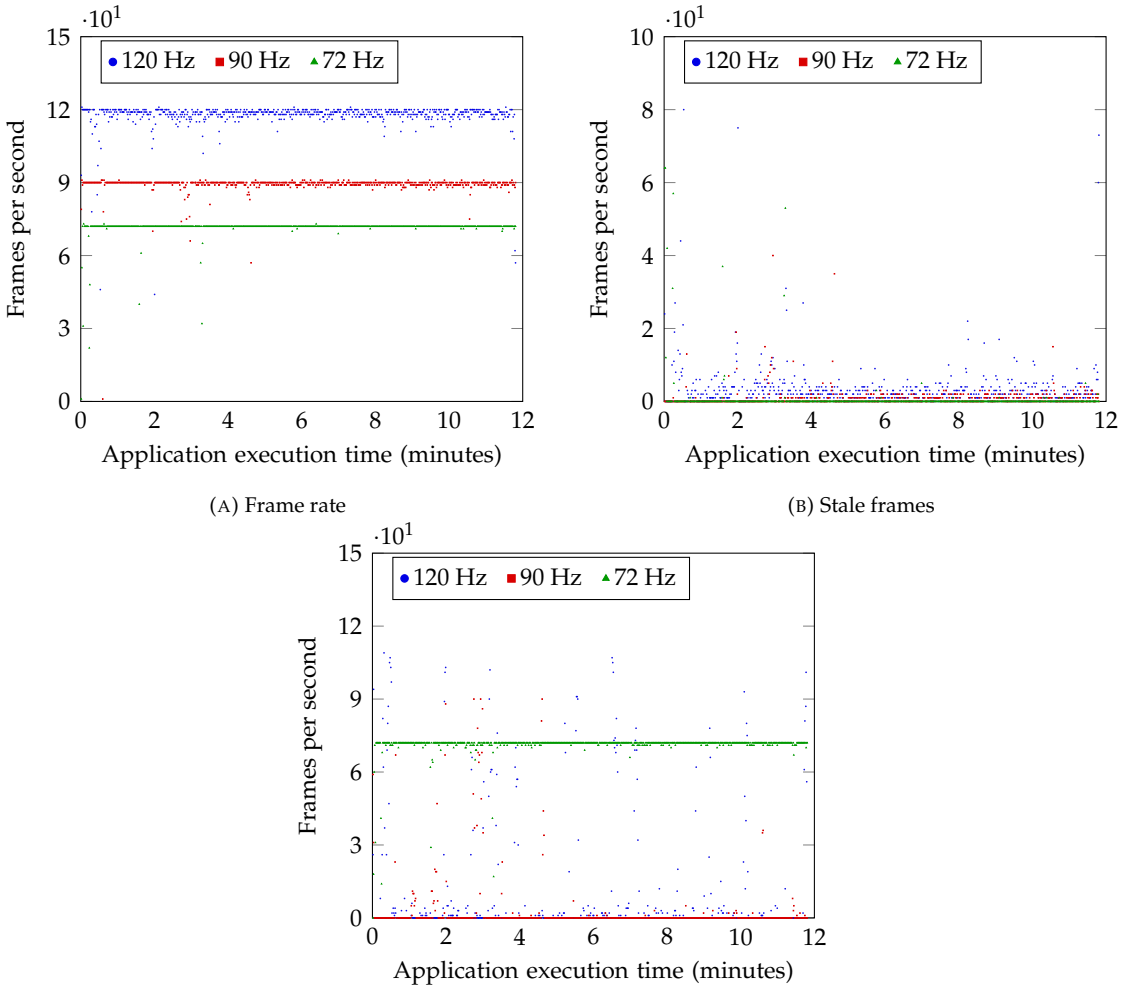
We measured the QoS of VR applications based on frame analysis with different refresh rates and the computational latency over Meta HMD. To understand the impact of different refresh rates on VR systems, we analyze two VR games, Echo VR and Elixir. Echo VR is a multiplayer game, which supports refresh rates of 90 Hz and 120 Hz. Besides, Elixir game supports hand tracking. Elixir supports a refresh rate of 72 Hz.

Frame Analysis

Figure 4.6 compares both games in terms of overall *frame rate*, *stale frames*, and *early frames* over different *refresh rates*. While frame rate is the number of images an HMD sends to its display every second, refresh rate refers to how fast the display shows those frames.

Figure 4.6a shows frame rate results, where the frames produced are measured in FPS. We discovered that the higher the refresh rate is, the fewer frames are produced. While this behavior is expected, Echo VR has far fewer frames because its refresh rate has been set to provide a higher realism. We observe that 26.87% of FPS were produced for Echo VR operating at 120 Hz, while the same game operating at 90 Hz achieved 65.91% of FPS. For

[2]https://www.oculus.com/experiences/quest/



(A) Frame rate

(B) Stale frames

(C) Early frames

FIGURE 4.6: Frames benchmarking of Echo VR and Elixir games running on Meta HMD.

Elixir, 96.81% of FPS were produced. Echo VR provides a considerably lower frame rate than Elixir despite the configuration of its refresh rate. This result suggests that increasing the refresh rate and rendering resolution improves the visual quality.

Figure 4.6b compares stale frames, the most important metric for evaluating the QoS of a VR application. A frame is considered stale if it is not ready to be displayed in time on the HMD, which forces the VR application to reuse an old frame that is now outdated. In most cases, if the application misses a frame, the stale frame rate increases, and the frame rate decreases. In most cases, if the application misses a frame, the stale frame increases, and the frame rate decreases. This result indicates that peaks with higher stale FPS can negatively impact the immersion provided by a VR system, which creates a less smooth in-VR experience.

Figure 4.6c shows the early frames, which represents the capability of delivering frames before they are needed. If the application does render quickly, the frame will be considered early, but the visual quality will look smooth. Elixir produced 98% of early frames. Despite the higher number of early frames, this result indicates that Elixir can be optimized to save computing resources and battery life.

Other findings from Figures 4.6a, 4.6b, and 4.6c are summarized as follows. Traditional games designed for conventional displays, e.g., using 30 FPS or 60 FPS, allow a small number of missed frames to go undetected by the user, mainly because the camera is decoupled from the display. However, missing frames in a VR environment trigger significant consequences for user experiences whenever the virtualized world does not match the real world in terms of image quality or even latency. As a consequence, the immersion provided by VR is compromised. A solution to increase the frame rate and decrease the stale frames would be to use a more powerful GPU on the HMD.

Computing Latency Analysis

VR systems have different sources of latency, e.g., the time between pressing a button and when the VR system detects it or when a frame is rendered until it appears on the VR HMD's screen. We focus on the time from when the VR system requests the user head orientation until the frames are rendered on the HMD. Figure 4.7 compares the computing latencies for each phase of a loop on Meta HMD.

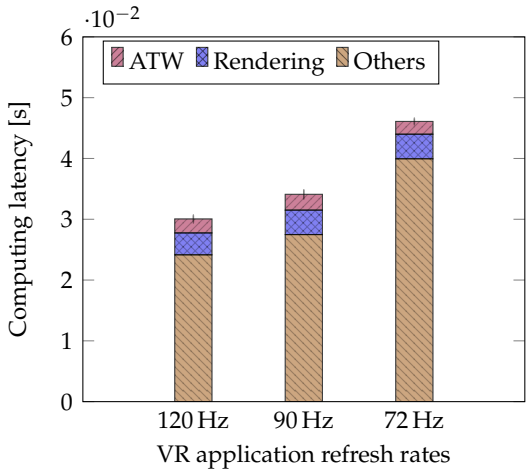


FIGURE 4.7: Computational latency benchmarking of different tasks for Echo VR and Elixir games running on Meta HMD.

Figure 4.7 shows the latency required by both applications to render the frames, e.g., *refresh time*. Refresh time is the duration of time for which one frame or image occupies the display. While Echo VR reached a mean of 3.6 ms (120 Hz) and 4.02 ms (90 Hz), Elixir reached a mean of 4.04 ms to render the frames. This result suggests that the higher the frame rate is, the faster these frames should be processed. However, higher frame rates introduce the need for more computing resources. Therefore, this result provides insight into how much headroom remains on the GPU, enabling analysis of compute-intensive objects running on the GPU.

Figure 4.7 shows how much time the Asynchronous TimeWarp (ATW) spends to apply distortions and displays the scenes on Meta HMD for both games. ATW is a software component that transforms stereoscopic images based on the latest head-tracking information to reduce the motion-to-photon latency, shifting the rendered image to adjust for changes in head movement. Echo VR demanded 2.3 ms (120 Hz) and 2.6 ms (90 Hz) during the ATW

phase. On the contrary, Elixir demanded 2.1 ms during the ATW phase. Furthermore, we analyze the *maximum rotational speed* in degrees per second because it impacts the latency on the ATW. The maximum rotation speed specifies the fastest speed the Meta HMD has rotated. Echo VR reached a maximum rotational speed mean of 38.8 (120 Hz) and 41.2 (90 Hz) degrees per second, respectively, while Elixir reached a maximum rotational speed mean of 57.4 degrees per second. We found out that lower rotations do not trigger higher latency as it slightly impacts the ATW performance.

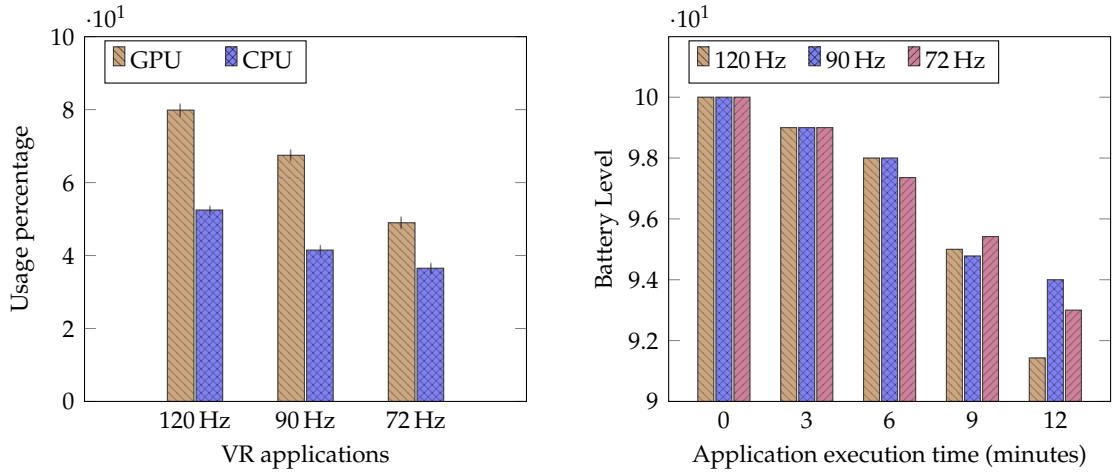
Figure 4.7 also shows the E2E latency of each application running on Meta HMD. In this context, E2E latency of each application is the sum of the latencies of all the tasks. This metric represents the time when an application does query the pose before rendering and the time the frames are displayed on the VR HMD. Besides, the *others* represent the task latencies that are not specified in Meta HMD API. The mean E2E latency for Echo VR is about 30.5 ms (120 Hz) and 34.1 ms (90 Hz), respectively. Nevertheless, Elixir reached a mean E2E latency of 46.9 ms, representing 53.77% more than Echo VR E2E latency. Noticeably, Echo VR offers lower E2E latency than Elixir.

Other findings from Figure 4.7 are summarized as follows. Different refresh rates impact the computing latency, e.g., a display operating at 72 Hz, 90 Hz, or 120 Hz takes up to 13.88 ms, 11.11 ms, and 8.33 ms to update the images, respectively. The higher the refresh rate is, the faster the display renders frames. However, more resources are needed to handle higher refresh rates, e.g., battery and GPU. As a result, increased power consumption in mobile HMDs leads to a poor user experience, and increasing the consumption of computational resources facilitates VR applications to run out of resources. Hence, higher refresh rates provide more realism for VR applications at the cost of higher refresh time, affecting battery usage and increasing the number of *stale frames*, which can break VR immersion.

CPU Usage, GPU Usage, and Energy Consumption

Figure 4.8 compares both Echo VR and Elixir games' *GPU usage*, *CPU usage*, and *energy consumption*. GPU and CPU utilization are important to understand if a VR application is GPU or CPU bound. In particular, GPU utilization is more valuable than CPU utilization as VR applications require more graphical features. From the GPU and CPU usage analysis, it is possible to evaluate the power consumption of an application.

Figure 4.8a indicates that both games are GPU bound as they used more GPU resources than CPU. We observe that Echo VR (120 Hz) has a peak of 88% of GPU utilization. Performance issues may occur if the GPU utilization is over 90%. This benchmark indicates that GPU can run out of resources for a more advanced game, potentially triggering a bottleneck for the application, especially the QoS. Moreover, the computing latency is highly influenced by the computing power of the GPU.Figure 4.8a also provides the CPU usage, which considers 8 CPU cores available in Meta HMD. In practice, it is infeasible for an HMD only to have a powerful GPU, because a powerful CPU is required to reach frame rate stability. Thus, both GPU and GPU need to have a balance in terms of computing power. In most cases, VR



(A) CPU and GPU usage of Meta.

(B) Energy consumption of Meta.

FIGURE 4.8: GPU, CPU, and power consumption benchmarking of Echo VR and Elixir games running on Meta HMD.

applications will have a balance with favoring GPU over CPU due to graphical requirements. Results from Figure 4.8a indicate that additional services running on the HMD to improve user experience, e.g., 3D scene reconstruction or scene depth estimation, would lead to more CPU utilization, which could easily reach 100% of CPU utilization on Meta HMD.

In a nutshell, Meta HMD QoS analysis can be described according to the following observations. (*i*) Higher refresh rates allow for more immersive experiences at the cost of reduced QoS; (*ii*) The QoS performance of VR applications can be significantly reduced if the HMD does not have enough computing power to handle high refresh rates; (*iii*) Energy consumption increases drastically whenever higher refresh rates are enabled, and (*iv*) Latency benchmarks indicate that we may be a long way from meeting the computing latency requirements for VR systems.

4.5.2 TENET Simulation

Trade-off Figure 4.9 shows the trade-off between the average E2E latency and the average power consumption for 1000 users when TENET runs with different values of the power sensitivity coefficient α. In line with our expectations, we observe that increasing values of α correspond to decreasing values of the average latency and increasing values of average power consumption. This relation exists because as α grows, TENET deploys more and more services on HMDs, reducing network latency but increasing the HMDs' power consumption. For applications that do not require stringent latency requirements, TENET can decrease α to reduce the system-wide power consumption. Regardless of the values of α, we observe that the E2E latency is always $\leq$ 5 ms because of the constraint in the TENET optimization problem, showing that TENET can always provide VR-compatible latency in high-mobility scenarios.

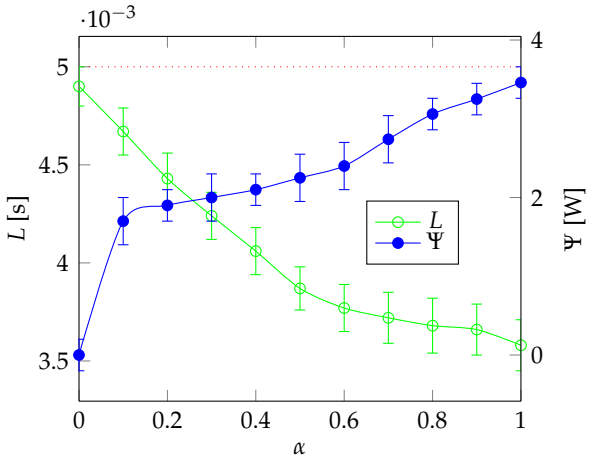
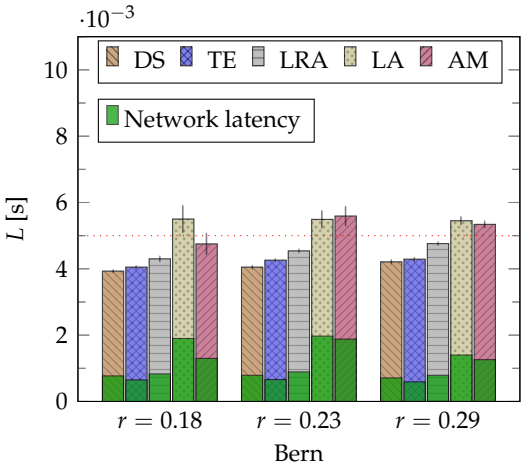


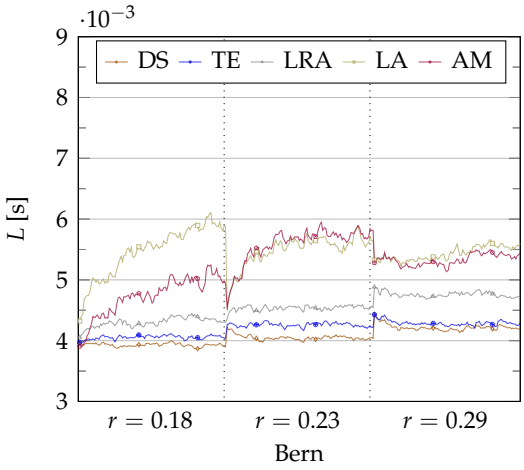
FIGURE 4.9: Trade-off between average E2E latency L and average power consumption Ψ.

E2E latency Figures 4.10a, 4.10c, and 4.10e show the average E2E latency as the sum of computation and network latency for the five evaluated schemes over three topologies, each with different radius and user densities when latency minimization has the highest priority ($\alpha = 1$). DO always performs the optimal E2E latency for all topologies at the cost of execution time. However, TENET provides the lowest E2E latency for all topologies compared to LRA, LA, and AM because it can deploy services in a way that minimizes network and computing latency. We observe that TENET deploys more services (on average 35%) on HMDs for all topologies because, when $\alpha = 1$, the TENET's cost function tends to minimize latency without considering power consumption on HMDs. This explains why TENET's network latency is lower and indicates that deploying services can improve the system-wide E2E latency onto HMDs. As the number of users in the scenario increases, more and more services need to be deployed on the MEC servers, leading to their saturation. For high user densities, services might be deployed on MEC servers that are topologically far from the HMD, resulting in increased network latency, as we observed.

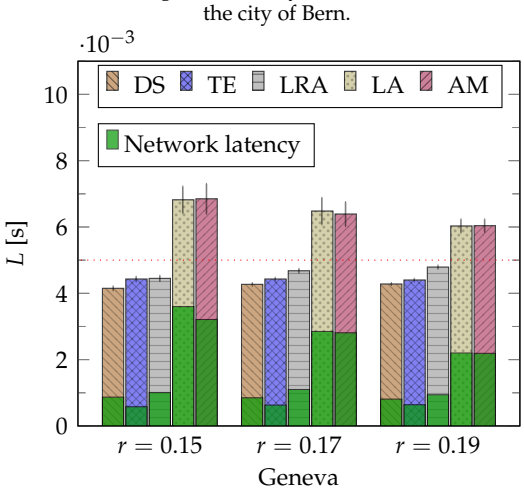
Figures 4.10b, 4.10d, and 4.10f show all algorithms' E2E latency over time. The DO algorithm indicates the global optimum latency in each iteration. We observe that the E2E latency increased for algorithms DO and LRA in all scenarios whenever the number of users has increased. Although LRA provides average E2E latency under φ_n, Figure 4.10f shows that LRA reached more than φ_n in all topologies. In contrast, TENET maintained its stable E2E latency for the topologies of GA and ZH. In contrast, for the topologies of GA and ZH, TENET maintained its E2E stable. This indicates that the higher radius r is, the lower is the network latency. DO and LRA highly depend on the number of users on the system to provide better latency performance. Therefore, using the zones scheme, TENET better distributes the services along MEC infrastructure, improving the average E2E latency even when more users are deployed in the same scenario. The same behavior does not occur in BE topology since it has fewer nodes than GA and ZH, which limits the possibility of exploiting a better service placement strategy. Algorithms LA and AM decrease E2E latency whenever a higher radius r is used.



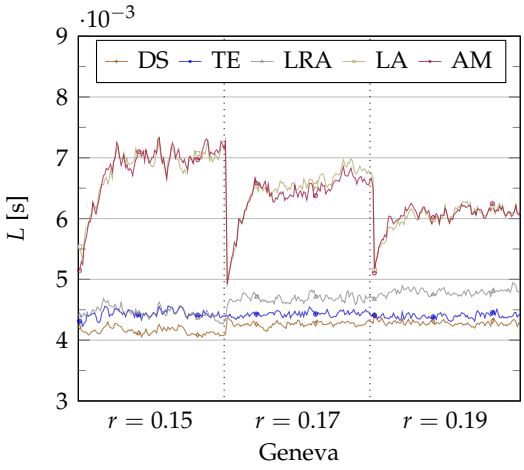
(A) Service average E2E latency L over different radii r for the city of Bern.



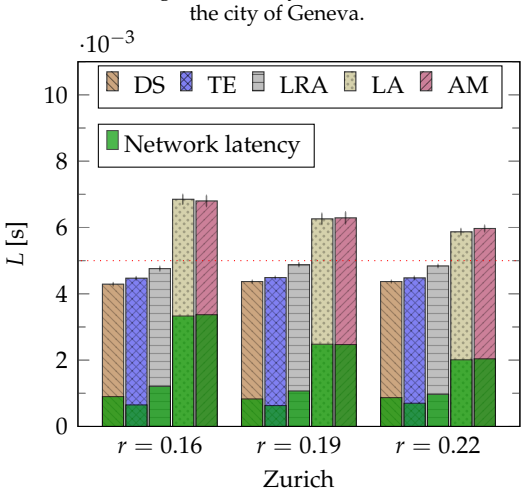
(B) Service E2E latency L over different radii r for the city of Bern.



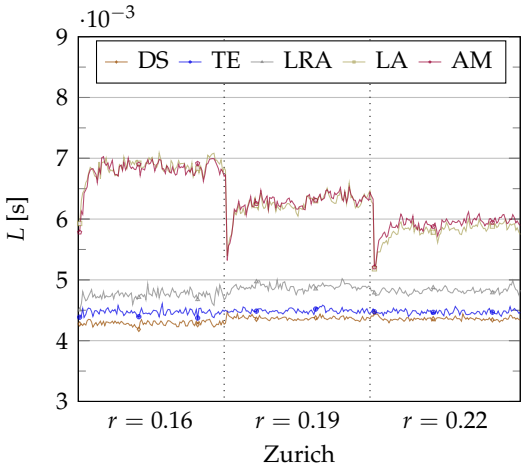
(C) Service average E2E latency L over different radii r for the city of Geneva.



(D) Service E2E latency L over different radii r for the city of Geneva.

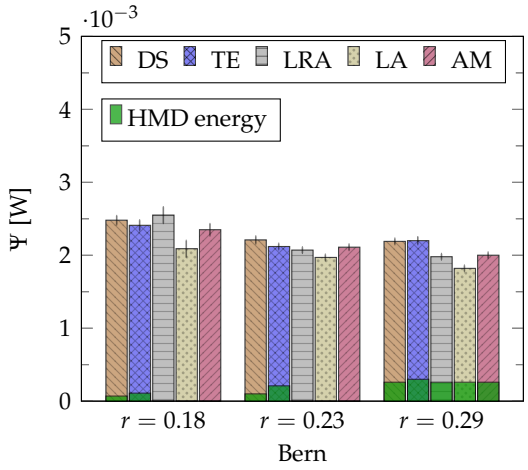


(E) Service average E2E latency L over different radii r for the city of Zurich.

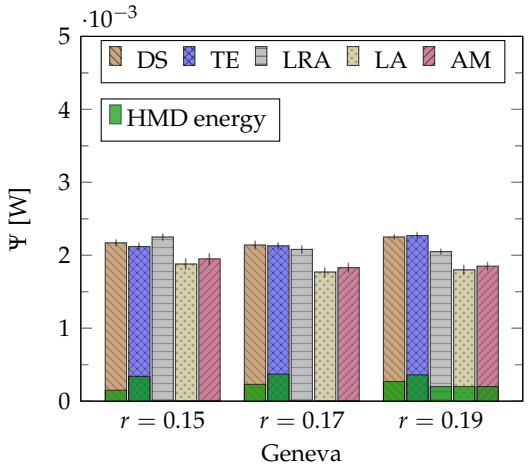


(F) Service E2E latency L over different radii r for the city of Zurich.

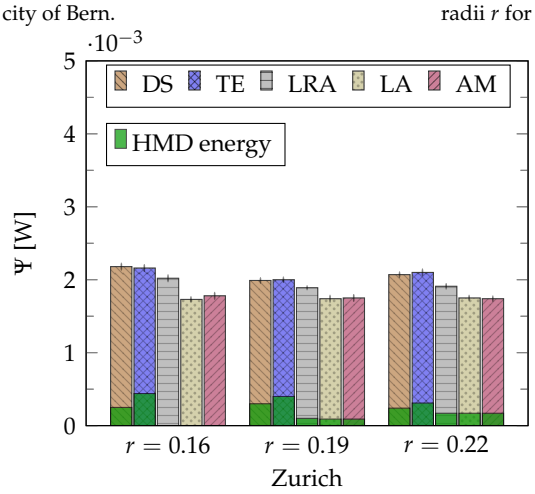
FIGURE 4.10: Performance evaluation of end-to-end latency and its convergence for the topologies of Bern, Geneva, and Zurich.



(A) Service average power consumption Ψ over different radii r for the city of Bern.



(B) Service average power consumption Ψ over different radii r for the city of Geneva.



(C) Service average power consumption Ψ over different radii r for the city of Zurich.

FIGURE 4.11: Performance evaluation of HMDs power consumption for Bern, Geneva, and Zurich.

Power consumption Figure 4.11 shows the average power consumed by each service as the sum of the average power consumed by the HMDs and the MEC infrastructure for the five evaluated schemes when energy minimization has the lowest priority ($\alpha = 1$). Although DO achieves a lower E2E latency than TENET, on average, DO consumes more power than TENET in all scenarios. The DO and TENET algorithms consume more HMD power than all other algorithms because, in some situations, when services move from a MEC server to an HMD, their E2E latency decreases (as shown in Figure 4.10), consequently demanding higher video resolutions that generate higher power consumption. Since TENET and DO are the only algorithms that can deploy services on HMDs, both are the only ones showing power consumption on HMDs, except in Figures 4.10a (r = 0.29), 4.10c (r = 0.19), and 4.10e (r = 0.19 and 0.22), where the entire MEC infrastructure was overloaded due to the number of users and available MEC servers. In contrast, the other compared algorithms only show infrastructure power consumption. This result motivates the need to deploy services based on a trade-off between latency and power consumption, which the TENET's design addresses.

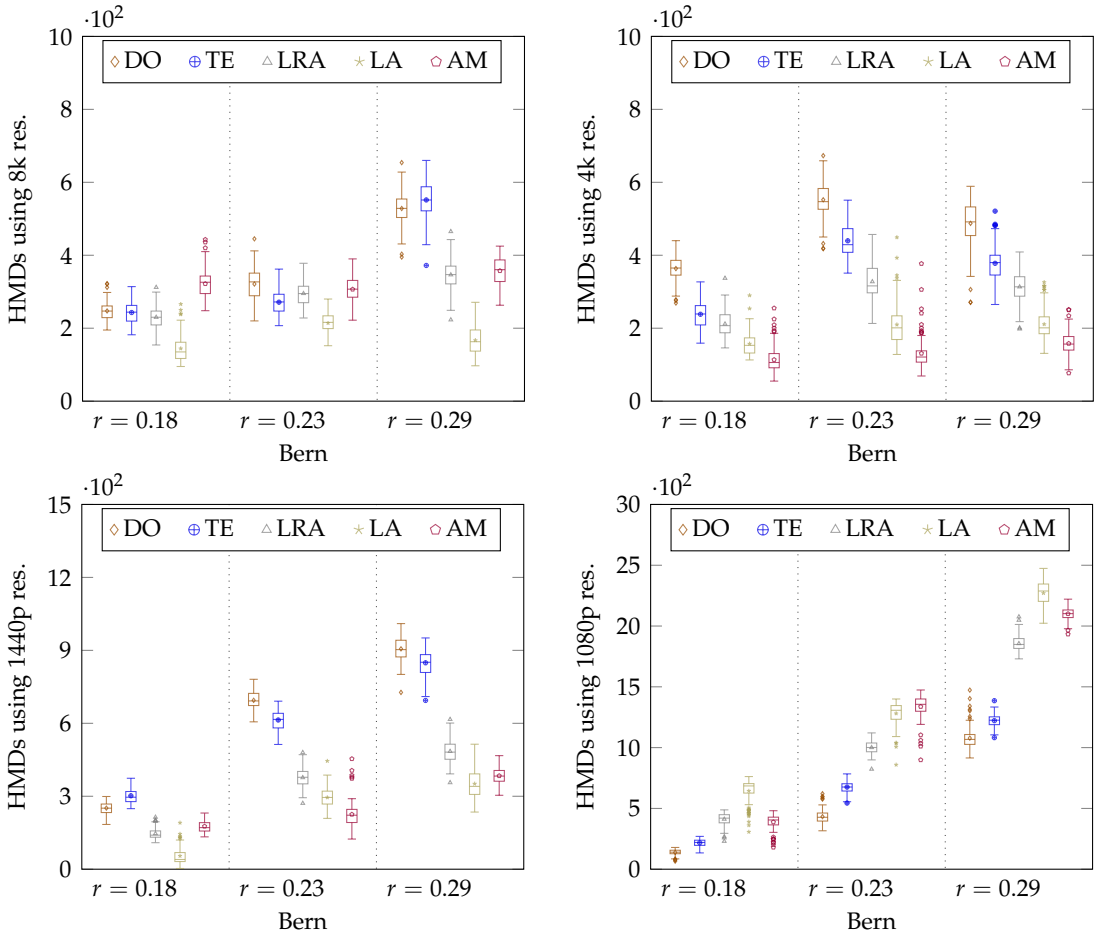


FIGURE 4.12: Average of total HMDs using resolutions 8k, 4k, 1440p, and 1080p over different radii r for the city of Bern.

Video resolution selection Figures 4.12 and 4.13 show the average of total HMDs using resolutions 8k, 4k, 1440p, and 1080p over different radii for all topologies. Each video resolution is selected based on the E2E latency provided by each algorithm. Although DO achieves 3% lower E2E latency performance on average than TENET, this greatly impacts the number of HMDs (on average 20% more) running at 8k and 4k resolutions in scenarios with fewer users. These results indicate that TENET can support videos at high resolutions at about the same rate as DO. Furthermore, TENET supports more HMDs running at 8k and 4k resolutions than LRA, LA, and AM. Thus, we show that no matter how slight the average E2E latency variation is, there is always a significant impact on the number of HMDs running high-resolution videos.

Context service migrations Figure 4.14 shows the *average acceptance* and *rejection service context migrations* over different radii for Bern, Geneva, and Zurich. The migration ratio is a crucial metric because frequent service migrations may introduce service interruption, leading to the migration process depending on network status, even if only the transfer of the service context is performed. Thus, fewer service migrations are expected to achieve better E2E latency performance. In all scenarios, we found that TENET provides a higher acceptance context migration than all other algorithms, except for DO. We observe that TENET keeps its

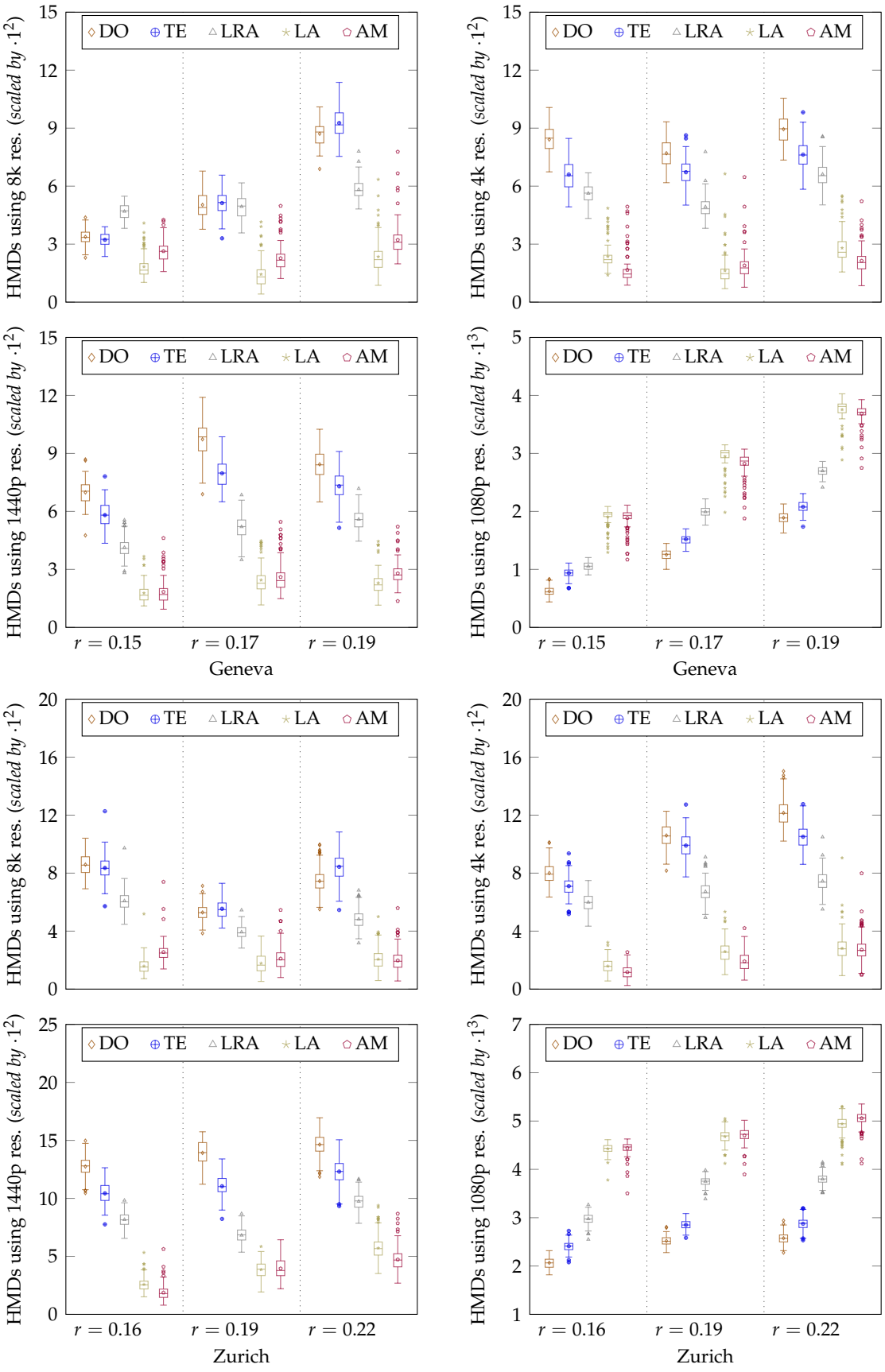


FIGURE 4.13: Average of total HMDs using resolutions 8k, 4k, 1440p, and 1080p over different radii r for the cities of Geneva and Zurich.

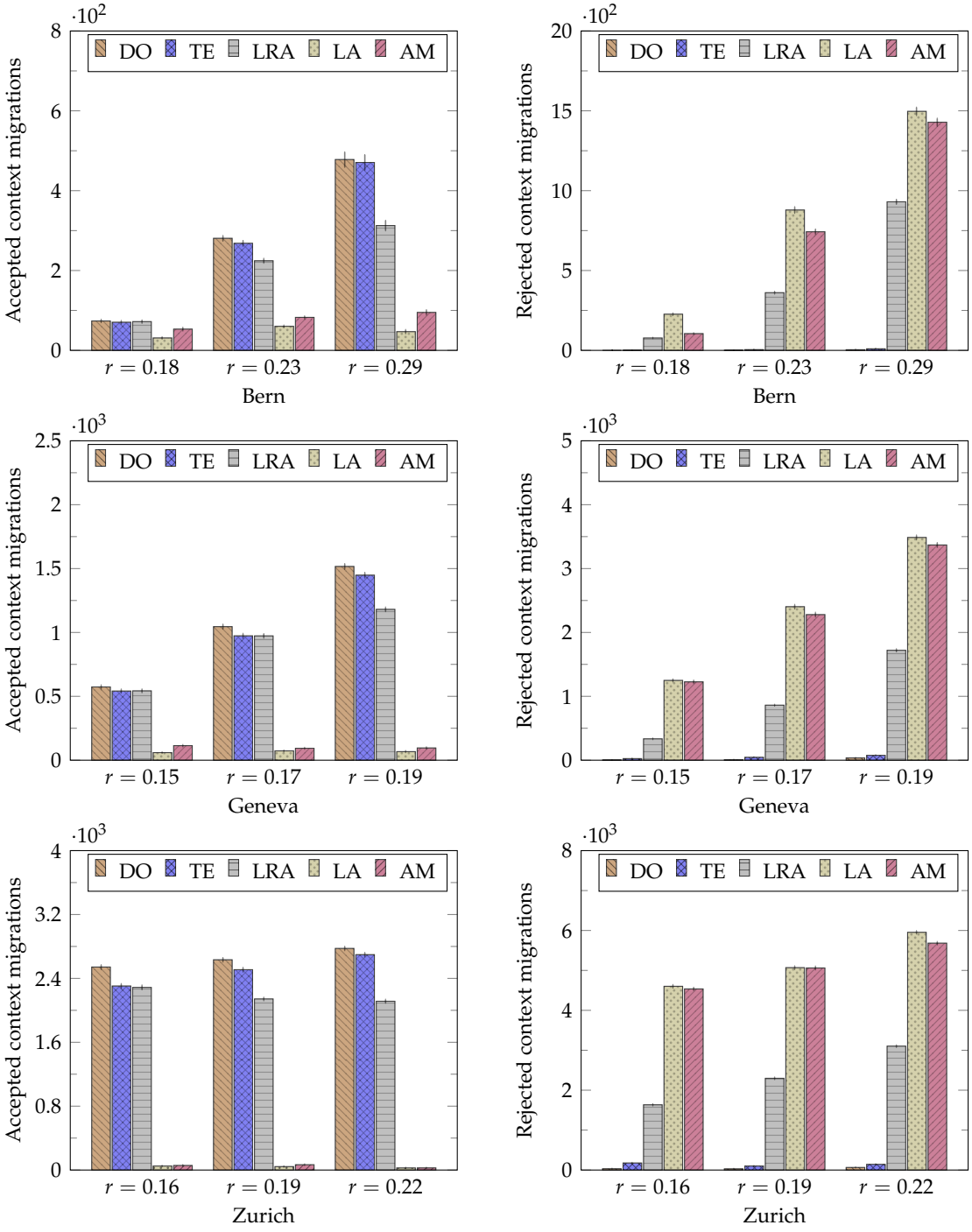


FIGURE 4.14: Average application context acceptance and rejection migrations over different radii r for the cities of Bern, Geneva, and Zurich.

performance constant, which does not occur for LRA in scenarios with more users. LA and AM provide a lower context acceptance ratio whenever more users are considered in each scenario. This occurs because the context migration ratio can be affected by the available MEC servers, i.e., whenever an algorithm chooses a server to host the service, and this server does not have available resources.

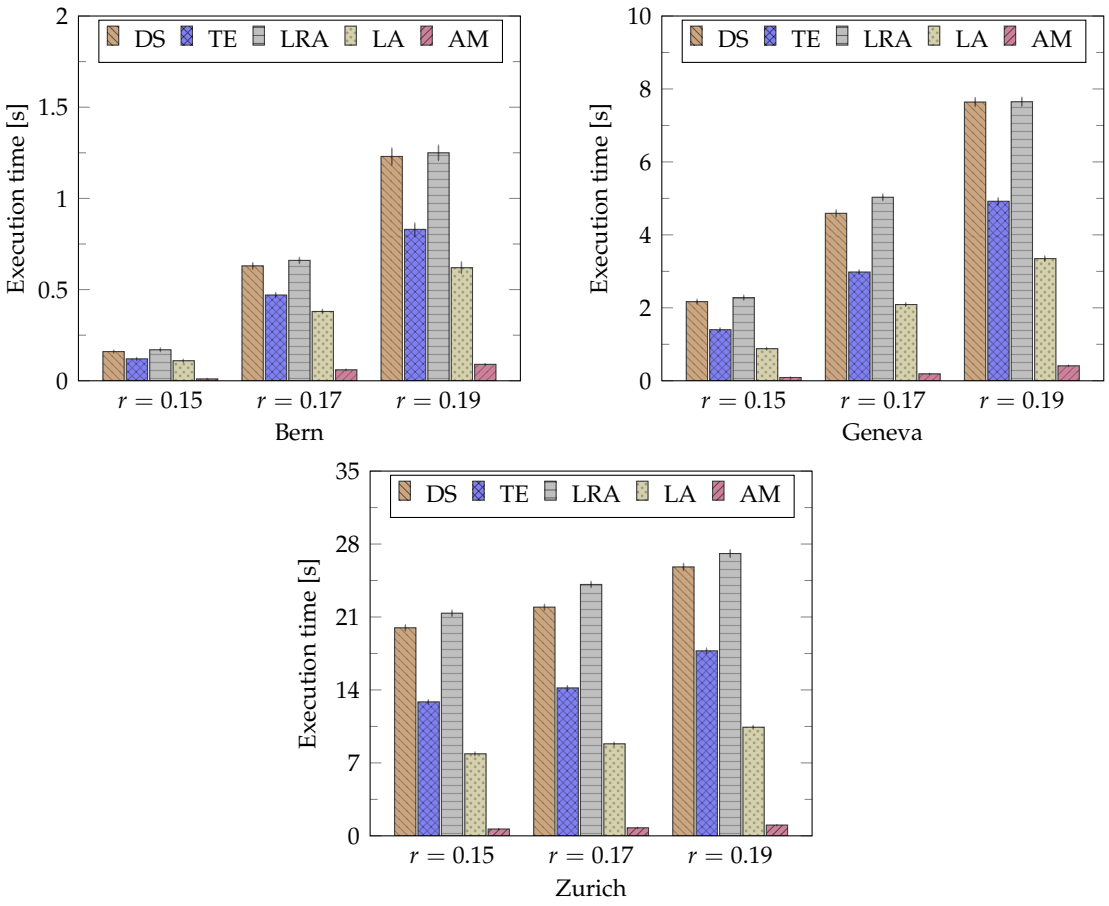


FIGURE 4.15: Average of total execution time to provide placement for all services over different radii r for the cities of Bern, Geneva, and Zurich.

Figure 4.15 shows the average total **execution time** to provide placement for all services over different radii for the cities of Bern, Geneva, and Zurich. Although DO performs the lowest E2E latency, it also performs a higher execution time than TENET. We observe that DO execution time grows fast whenever more users are included in the system, while TENET remains stable. In the Zurich topology, DO execution time is almost double the TENET execution time. Besides, even LRA provides a higher execution time than DO due to the number of rejected context migration requests. Although AM performed a high rejected context migration ratio, it achieved the lowest execution time due to not discovering a target MEC server whenever a context migration is needed. This result suggests that in a real scenario with many more users, 5G base stations, and MEC servers, DO performs an exponential execution time $\mathcal{O}(2^n)$ to find out the optimal placement of all services running in the system. At the same time, the heuristic provided by TENET can achieve acceptable E2E latency performance with a logarithmic execution time $\mathcal{O}(\mathbb{E}log(\mathbb{V}))$.

4.6 Chapter Conclusions

In this chapter, we have proposed a novel strategy to minimize the E2E latency for the next generation of 6DoF VR applications. We addressed the ***Research Question 2.1*** by showing how to distribute VR services across MEC servers to reduce the E2E latency for VR applications. The optimal solution is formulated through an integer linear programming problem (DSCP) whose objective is to find the optimal service placement of services from a service chain with varying capacity requirements of decoder services while satisfying 6DoF VR application ultra-low latency requirements of 5 ms. We show that DSCP implementation is unfeasible whenever there are too many VR users and network nodes. Thus, we have shown that new heuristics must be developed to get around the DSCP problem.

We propose TENET, a fast heuristic to solve DSCP problem. TENET manages 6DoF VR services by distributing them over edge networks and HMDs to avoid increased latencies. We addressed the ***Research Question 2.2*** by providing the trade-off between E2E latency and energy consumption through network simulations over three high-mobility scenarios compared to widely used service-migration strategies. We have developed algorithms for path calculation based on E2E latency and management of VR applications to ensure acceptable E2E latency and TENET architecture. We also developed a workload model to VR 6DoF virtualized applications based on Meta HMD application workloads.

Our evaluation demonstrates the benefits of TENET in managing VR 6DoF services by distributing them over MEC servers. We have shown that for varying user densities in an urban scenario, TENET outperforms other widely adopted mechanisms in terms of E2E latency in exchange for a moderate increment in power consumption. Moreover, we observe significant gains of TENET in selecting higher video resolutions for 6DoF VR applications based on E2E latency. TENET also provides more accepted context migrations than traditional service migration algorithms. Finally, we have shown that TENET reduces the decision time to perform service placement. Therefore, we addressed the ***Research Question 2.3*** by showing that TENET deployment strategy impacts the E2E latency of VR applications and the selection of better video resolutions for latency-sensitive VR applications.

The findings and contributions in this chapter provide insightful directions on advancing VR service deployment over edge networks, considering the restrictions of the edge infrastructure, the resource limitations of HMDs, and the stringent latency requirements of 6DoF VR applications. However, only considering the E2E latency optimization in a highly dynamic edge infrastructure scenario cannot ensure the reduction of the overall E2E latency in a real network scenario where several users are deployed on the network. This occurs because the network resource availability can negatively impact the overall E2E latency for all applications. Thus, optimizations on the network infrastructure to minimize the latency and, therefore, the E2E latency are required. Therefore, in Chapter 5, we address the issues of latency-sensitive routing for 6DoF VR applications, which focus on reducing the overall E2E latency by optimizing the routing process with an E2E latency awareness approach that also considers the impact of path allocations on other flows deployed on the network.

Chapter 5

Latency Sensitive Routing Algorithm for VR

5.1 Introduction

One of the significant advancements in VR technology is the adoption of 6DoF technology to support both body and head motion [5]. 6DoF VR applications require low-latency communications and processing to guarantee an immersive and reactive experience. Thus, the network infrastructure must provide ultra-low latency of less than 1 ms 1 ms [7]. However, traditional Internet routing protocols, designed primarily for throughput, may not meet the stringent low latency requirements of 6DoF VR applications [39].

Conventional routing protocols like IGRP and EIGRP consider factors such as throughput, latency, and network load when selecting paths within a network. Typically, they select paths with the highest throughput capacity while minimizing latency [40]. However, this strategy fails to optimize overall application throughput while providing low latency for all VR users on the network. It overlooks how assigning a path to a specific flow affects the network latency for other flows during periods of network congestion, in which alternative paths prioritizing network throughput are often chosen, leading to increased network latency [41].

Hence, allocating paths from VR flows to a cloud server can influence network congestion. Thus, alternative paths are required to accommodate the unique demands of different VR applications. As shown in chapter 4, only considering the latency optimization cannot ensure the reduction of the overall E2E latency because the network latency must also be optimized. Therefore, managing VR flows, each with different deployment policies and requirements while guaranteeing E2E latency and throughput in large-scale networks is challenging.

In this chapter, we design and investigate a network routing strategy to support the latency requirements for 6DoF VR applications. The following sections discuss the content described in **Paper [4]**. Therefore, Section 5.1 describes the research questions addressed in this chapter and discusses the contributions of the new latency-sensitive routing. Section 5.2 formulates the system model. Section 5.3 discusses the new latency-sensitive routing strategy. Section 5.4 describes the experiment setup. Section 5.5 discusses the evaluation results. Finally, section 5.6 concludes the study in this chapter.

5.1.1 Research Questions Addressed

The contributions of this chapter aim to answer the following research questions described in Section 1.3.3.

Research Question 3.1: How does the decision on the path of a 6DoF VR flow impact network latency and throughput for subsequent application flows in a large-scale network?

Research Question 3.2: How can overall network latency and throughput be optimized for applications deployed on the network?

Research Question 3.3: How does processing latency impact the decision on which path must be selected to fulfill the requirements of 6DoF VR applications?

5.1.2 Chapter Contributions

To address the challenges mentioned above, we propose in **Paper [4]** FLATWISE, a novel intra-domain routing algorithm with latency guarantees, which considers the E2E latency requirements of 6DoF VR applications. We address the ***Research Question 3.1*** by exploiting how FLATWISE approximates the E2E latency of the calculated path with the E2E latency required by each 6DoF VR application while it considers the impact of path assignment on other 6DoF VR applications. We address the ***Research Question 3.2*** by showing that FLATWISE minimizes the overall E2E latency performance for all 6DoF VR applications deployed on the network. We address the ***Research Question 3.3*** by showing that FLATWISE provides an adaptive routing approach that can squeeze or relax the path calculation based on the E2E latency requirement of 6DoF VR applications. Our contributions are as follows.

- We define the Joint Flow Allocation (JFA) problem to find suitable paths for all flows in a network such that it determines the optimal paths in terms of throughput and latency to reduce the overall latency for all flows. We use Mixed Integer Linear Programming (MILP) to model the JFA objective and constraints (Section 5.2).
- The JFA problem is $\mathcal{NP}$-hard, i.e., computationally expensive. Therefore, we propose a heuristic (FLATWISE) that is one order of magnitude faster than the JFA (Section 5.3). We provide algorithms for path allocation based on E2E latency awareness and for the source node selection based on the E2E latency awareness. We show the FLATWISE implementation and compare its performance with WSP and SWP approaches.
- We assess FLATWISE performance in a realistic simulated 5G network map of Bern, Geneva, and Zurich. Based on those topologies, we model both network and computing latencies used in the FLATWISE simulation environment (Section 5.4). We implement and compare the algorithms WSP and SWP against FLATWISE in a highly dynamic and realistic network environment where there is no flow prioritization, and the network link availability changes over time. We also evaluate the performance of FLATWISE, WSP, and SWP by analyzing the VR in-game communication as a reference use case. We consider the KPIs flow network latency, path latency, over-provisioned latency, E2E latency, flow network throughput, frame rate, video resolutions, and execution time (Section 5.5).

5.2 System Model and Problem Formulation

5.2.1 System Model

Our scenario contains a set of users, each provided with an HMD that executes a VR application, e.g., VR games. We assume that each user can move around in the scenario at speeds ranging from pedestrians to vehicles and is always connected to the Internet via a 5G base station. In our scenario, the most challenging use case is ***VR in-game communication***, in which HMDs are connected to a cloud server and require low-latency video streaming to communicate with other players. Figure 5.1 shows the system model representation, where it represents the base stations, network links, MEC servers attached to each base station, the deployment of VR video decoders, VR HMDs, SDN controller, and the cloud server to host the VR application logic.

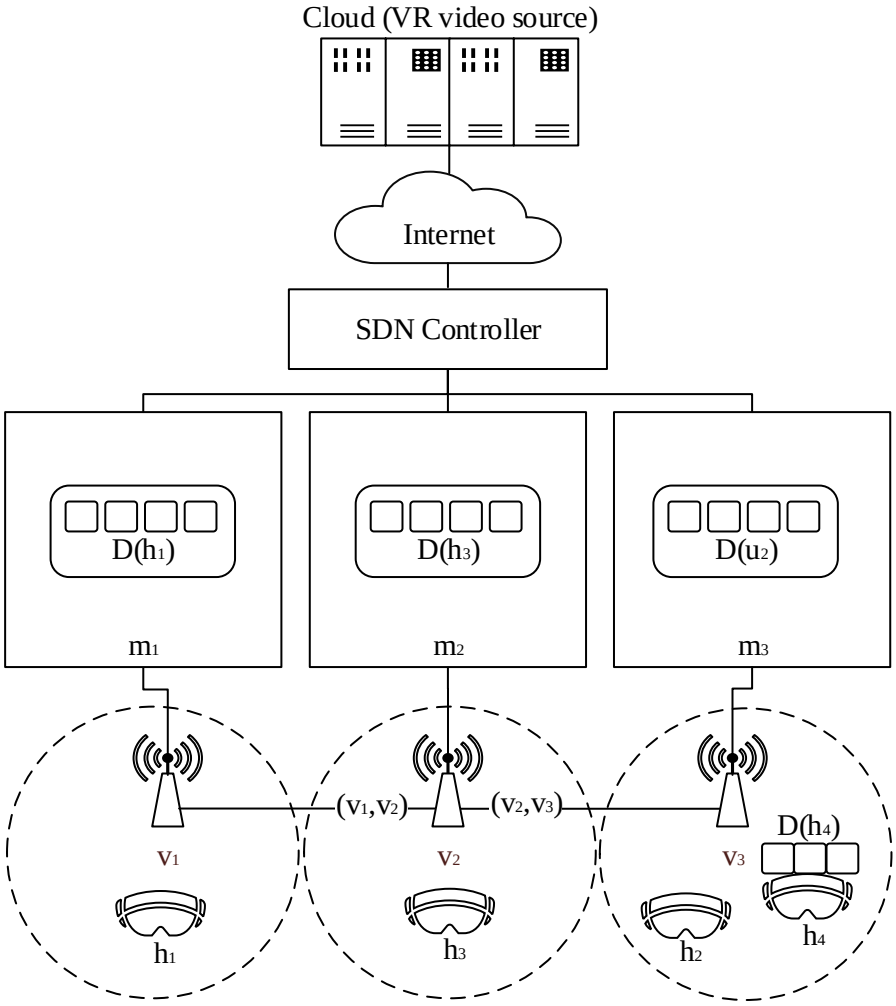


FIGURE 5.1: FLATWISE System Model Representation.

The network infrastructure is defined as a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$. Let us denote the set of base stations as $\mathbb{V} = \{v_1, \ldots, v_{|\mathbb{V}|}\}$ and the set of edges as $\mathbb{E} = \{(i,j)|i,j \in \mathbb{V}\}$, where each ordered pair $(i,j) \in \mathbb{E}$ represents the connection between the base stations i and j. Let us denote the *throughput capacity* of edge $(i,j) \in \mathbb{E}$ as $c_{ij} \in ([0,+\infty)$ and the *link latency* of edge $(ij) \in \mathbb{E}$ as

$l_{ij} \in (0, +\infty)$. Each base station v_i is equipped with one MEC server. Let us denote the set of all MEC servers as $M = \{m_1, \ldots, m_{|M|}\}$, each with potentially different physical resources. Let us denote $c(m_i)$ as the number of CPUs and $g(m_i)$ as the number of GPUs available at MEC server m_i. The processing latency of MEC server m_i is denoted by $p(m_i)$, representing the time MEC server m_i takes to process a given task CPU and GPU workloads φ_c and φ_g, respectively, defined as the task's required CPU and GPU computation rates expressed in [cycles/s]. Besides, $p(m_i)$ on MEC server m_i also depends on CPU and GPU cycle speeds $\Phi(m_i)$ and $\sigma(m_i)$, expressed in [cycles/s], as well as the time it takes for the VR task to run on the CPUs and GPUs, defined as $t_{c(m_i)}$ and $t_{g(m_i)}$, respectively. We assume that each task can be run parallel over all CPUs and GPUs available at the MEC server m_i. Therefore, we use the following equation to denote the processing latency $p(m_i)$ of a VR task, expressed in seconds.

$$p(m_i) = \max \left\{ \frac{\varphi_c \cdot t_{c(m_i)}}{\Phi(m_i) \cdot c(m_i)}, \frac{\varphi_g \cdot t_{g(m_i)}}{\sigma(m_i) \cdot g(m_i)} \right\} \tag{5.1}$$

We consider the set of VR HMDs $H = \{h_1, \ldots, h_{|H|}\}$ deployed on the graph, where each HMD runs a real-time 6DoF VR application. Each HMD h_i is connected to a base station $\in \mathbb{V}$, and its VR application has several decoder services. The set of decoder services of HMD h_i is denoted by $D(h_i) = \{d_1, \ldots, d_{|D(h_i)|}\}$, where $|D(h_m)|$ represents the number of decoder services available for each user h_i. The decoder services $D(h_i)$ can be deployed on the HMD h_i or any MEC server available at the edge infrastructure. We denote $m(h_i)$ to represent the MEC server hosting the decoders of HMD h_i. If the decoder services $D(h_i)$ are deployed on HMD h_i, then they generate a processing latency on the HMD h_i, denoted by $p(h_i)$. Thus, $p(h_i)$ also depends on both CPU cycle speed $\Phi(h_i)$ and GPU $\sigma(h_i)$ cycle speed on HMD h_i, as well as the time it takes for the VR task to run on the CPU and GPU, defined as $t_{c(h_i)}$ and $t_{g(h_i)}$, respectively. We also assume that each task can be run parallel over all CPUs and GPUs available at the HMD h_i. Therefore, we use the following equation to denote the processing latency HMD $p(h_i)$, expressed in seconds.

$$p(h_i) = \max \left\{ \frac{\varphi_c \cdot t_{c(h_i)}}{\Phi(h_i) \cdot c(h_i)}, \frac{\varphi \cdot_g t_{g(h_i)}}{\sigma(h_i) \cdot g(h_i)} \right\} \tag{5.2}$$

Each HMD h_i transmits the user's movements to a cloud server and receives a VR video with updates of other users' positions from that cloud server, where the VR video runs at a particular resolution and operates at a specific frame rate. Therefore, each HMD h_i communicates with the cloud server through a network flow. We define the set of all possible flows on the network as $F = \{f_1, \ldots, f_{|F|}\}$. The i-th flow is defined as $f_i = (s, t, \overline{\delta}, \overline{\omega})_i \in \mathbb{V} \times \mathbb{V} \times \mathbb{R}_+ \times \mathbb{R}_+$, where $s_i \in \mathbb{V}$ is the base station (source node) where the HMD h_i is connected, $t_i \in \mathbb{V}$ is the flow destination node where the cloud server is connected, $\overline{\delta}_i$ is the maximum latency requirement of flow f_i, and $\overline{\omega}_i$ is the minimum throughput requirement of flow f_i. We model $\overline{\omega}_i$ to represent the average bitrate for flow f_i and define it as $\overline{\omega}_i = \frac{\xi_i \varrho_i \Theta_i}{\varsigma_i}$, where ξ_i denotes the number of pixels in the images to be transmitted within the VR video in flow f_i, ϱ_i represents the number of bits per pixel in flow f_i, Θ_i represents for

the frame rate of the VR video in flow f_i. Finally, ς_i indicates the variable compression ratio in flow f_i, which depends on the network conditions.

5.2.2 Problem Formulation

Each flow has a path associated with it. We define the set of all possible paths on the network graph as $R = \{r_1, \ldots, r_{|R|}\}$. The path associated with flow f_i is defined as $r_i = \{(s, v_1), (u_2, v_2), \ldots, (u_{|r|}, t_i)\}$, with intermediate nodes u_k and v_k, such that $v_k = u_{k+1}$ for all $k \in \{1, \ldots, |r_i| - 1\}$. The vector containing one path for each flow in the system is represented by the *path vector* $\boldsymbol{r} = (r_1, \ldots, r_{|F|}) \in R^{|F|}$. Let us define flow i's throughput as $\omega^{(i)}$. We can now define the throughput vector $\boldsymbol{\omega} = (\omega^{(1)}, \ldots, \omega^{(|F|)}) \in \mathbb{R}^{|F|}$, where each component holds the throughput of flow i. The delay associated with a given path $r \in R$ is defined as $\delta(r) = \sum_{(i,j)\in r} l_{ij} + p(h_s) + p(m_t)$, where h_s represents the HMD h_i connected to source base station s and m_t is the cloud server connected to destination node t.

Our objective is to compute an optimal path vector $\boldsymbol{r}^*$ and a corresponding optimal throughput vector $\boldsymbol{\omega}^*$ associated with the flow set F such that a utility function $U(\boldsymbol{\omega}, \boldsymbol{r})$ is minimized. This is equivalent to jointly allocating all system flows on optimal paths that achieve the best global system performance. We design a utility function $U(\boldsymbol{\omega}, \boldsymbol{r}) = \sum_{k=1}^{|F|} \alpha\delta(r_k) - (1-\alpha)\omega^{(k)}$ that assigns a high value to those paths and throughput vectors that balance global throughput and delay depending on a *sensitivity coefficient* $\alpha \in [0, 1]$. This sensitivity coefficient weighs the relative importance between throughput and latency for the final application according to the policymaker's preference, tuning the value of α closer to 0 if throughput is more important and closer to 1 if delay is more important to provide QoS to the final application. We formulate the joint flow allocation problem in Optimization Problem 5.3.

$$\underset{\boldsymbol{\omega}\in\mathbb{R}^{|F|}, \boldsymbol{r}\in R^{|F|}}{\text{minimize}} \quad \sum_{k=1}^{|F|} \alpha\delta(r_k) - (1-\alpha)\omega^{(k)} \tag{5.3}$$

subject to

$$\delta(r_k) \leq \overline{\delta}_k \quad \forall k \in \{1, \ldots, |F|\} \tag{5.3a}$$

$$\omega^{(k)} \geq \overline{\omega}_k \quad \forall k \in \{1, \ldots, |F|\} \tag{5.3b}$$

$$\sum_{k\in\{1,\ldots,|F|\}} \omega^{(k)} \mathbb{1}_{(i,j)\in r_k} \leq c_{ij} \quad \forall (i,j) \in \mathbb{E} \tag{5.3c}$$

Constraints 5.3a and 5.3b ensure that the selected paths and throughputs provide the minimum latency and throughput performance levels required by the flows. Constraint 5.3c ensures that the sum of all flows on any edge does not surpass its capacity c_{ij}. The symbol $\mathbb{1}_q$ represents an indicator function that returns 1 if the statement q is true and returns 0 otherwise.

The formulated Optimization Problem 5.3 (JFA) is an NP-hard Mixed Integer Linear Programming problem, with exponential worst-case time complexity. Therefore, it is essential to devise polynomial-complex heuristics to solve this problem in a feasible time scale, even though sub-optimally.

5.3 Calculating Paths for VR Flows with FLATWISE

This section introduces FLATWISE, a novel heuristic method to provide an approximate solution to the JFA problem while reducing the required computational requirement. Furthermore, this section describes the process of calculating paths based on flow latency and throughput requirements, the E2E latency awareness for source node selection, and FLATWISE dynamic path calculation compared to WSP and SWP approaches. We also describe in detail how the selection of the source node impacts the path searching and the E2E latency for a VR flow.

5.3.1 Latency-aware Adaptive Path Allocation with FLATWISE

Algorithm 6 aims to calculate different paths r_i based on the latency requirements $\overline{\delta}(f_i)$ of each flow f_i. Furthermore, each path $r_i \in \mathbf{r}$ is always calculated respecting the throughput requirements of each flow $\overline{\omega}(f_i)$. Therefore, the main objective is to approximate the calculated path latency $\delta(r_i)$ to the latency required by each flow $\overline{\delta}(f_i)$.

First, Algorithm 6 initializes the variables *dist*, and *prev*. Then, Algorithm 6 uses *candidates* to store the nodes during the search (lines 1-5). Algorithm 6 uses a priority queue (Fibonacci heap) to efficiently find a suitable path that satisfies the latency $\overline{\delta}(f)$ and throughput $\overline{\omega}(f)$ requirements of the flow f (line 7). Algorithm 6 starts removing the source node s from *candidates* and starts the search in its neighbors while updating each neighbor's distance and the previous node to reach it (lines 9-14). Whenever a shorter distance weight to reach each neighbor is found based on the function *LatencyEst*, then the neighbor's distance is updated accordingly, and the function *LatencyEst* is used to calculate the weight of each neighbor before including it in the priority queue (lines 15-19). Finally, Algorithm 6 returns *prev* and *dist*, where *prev* stores the previous node used to reach each node in the path $r_i \in \mathbf{r}$, and *dist* stores the distance (latency) from the source node s to each node n (line 20). At the end of the process, a path r_i is traced for flow f_i, in which its cost is less than or equal to the flow latency requirement $\overline{\delta}(f)$ and also satisfies the flow throughput requirement $\overline{\omega}(f)$. However, Algorithm 6 approximates the cost of the traced path r_i to the flow latency requirement $\overline{\delta}(f)$.

The function *LatencyEst* calculates the latency weight between a current traversed node n and its predecessor p in a graph to define the priority of p in the routing procedure, where it considers the application flow latency requirement $\overline{\delta}(f)$, the achieved latency to reach the currently traversed node n, the Euclidean distance, edge latency l_{np} between n and p, and edge congestion between n and its predecessor p (lines 22-27). The FLATWISE heuristic guides the search based on the closest latency achieved at each iteration compared to the flow latency. Thus, the function *LatencyEst* adapts its search criteria according to the latency

Algorithm 6 FLATWISE Operation

Input: $G = (\mathbb{V}, \mathbb{E})$, s, t, f ▷ graph, source, target, flow
Output: *prev*, *dist* ▷ distance, previous nodes

1: $dist$, $prev \leftarrow \{\}$
2: **for** $n \in \mathbb{V}$ **do**
3: dist[n] $\leftarrow +\infty$
4: dist[s] $\leftarrow 0$
5: candidates $\leftarrow$ [(dist[s] + LATENCYEST(s, t, $\varnothing$, f, $\varnothing$), s)]
6: **while not** candidates $= \varnothing$ **do**
7: $c_{node} \leftarrow$ HEAPPOP(candidates) ▷ current node(c_{node})
8: **if not** $c_{node} = t$ **and** $c_{node} \in \mathbb{V}$ **then**
9: ▷ Remove c_{node} from unvisited nodes set
10: $\mathbb{V} \leftarrow \mathbb{V} \setminus \{c_{node}\}$
11: neighbors $\leftarrow$ GETNEIGHBORS(c_{node})
12: **for** n **in** neighbors **where** $b_{c_{node},n} \geq \overline{\omega}(f)$ **do**
13: $new_{dist} \leftarrow$ dist[c_{node}] + $l_{c_{node},n}$
14: **if** $new_{dist} <$ dist[n] **then**
15: dist[n] $\leftarrow new_{dist}$
16: prev[n] $\leftarrow c_{node}$
17: $n_{weight} \leftarrow$LATENCYEST(n, t, c_{node}, f,dist[n])
18: $candidate_{dist} \leftarrow new_{dist} + n_{weight}$
19: HEAPPUSH(candidates, $candidate_{dist}$, n)
20: **return** *prev*, *dist*

21: **function** LATENCYEST(n, t, p, f, n_{dist})
22: $latency_{weight} \leftarrow 1$
23: **if not** $p = \varnothing$ **then**
24: $l_{np} \leftarrow$ GETEDGELATENCY(n, p)
25: ▷ Compute residual capacity and latency
26: $r_c \leftarrow c_{np} - \overline{\omega}(f)$
27: $r_l \leftarrow \left| \overline{\delta}(f) - \left(\frac{l_{np}}{\overline{\delta}(f)} \right) \right|$
28: $latency_{weight} \leftarrow r_l + r_c + \frac{\overline{\delta}(f)}{1.05 n_{dist}}$
29: $(x_n, y_n) \leftarrow$ GETNODEPOSITION(n)
30: $(x_t, y_t) \leftarrow$ GETNODEPOSITION(t)
31: ▷ Return weighted Euclidean distance
32: **return** $latency_{weight} \cdot \sqrt{(x_n - x_t)^2 + (y_n - y_t)^2}$

of each neighbor of the current traversed node n. We use the Euclidean distance to guide FLATWISE's search, which is always calculated from n to the destination node t. In the first iteration of Algorithm 6, there is no predecessor p, because n is the source node s. Therefore, the weight for this particular node is the Euclidean distance between s and t. To calculate the weight of n, the function *LatencyEst* considers the flow latency $\overline{\delta}(f)$ and the edge congestion level r_c between n and p. We calculate the edge congestion level r_c according to its current throughput capacity c_{np} and the flow f_i throughput requirement $\overline{\omega}(f)$. The desired latency is calculated through the absolute value considering the flow latency requirement $\overline{\delta}(f)$ and the edge latency l_{np} between n and p (line 28). After that, the latency weight is calculated using the desired latency, edge congestion, flow latency $\overline{\delta}(f)$, and the current distance of n (29-31).

Here, the higher the edge latency l_{np} is, the greater is the impact on the latency weight. Finally, the Euclidean distance is multiplied by the latency weight and then returned by the function *LatencyEst* (line 32). Algorithm 6 shows that the proposed optimization problem can be solved in linear time. The Algorithm 6 has a time complexity of $\mathcal{O}((\mathbb{V}+\mathbb{E})log(\mathbb{V})) = \mathcal{O}(\mathbb{E}log(\mathbb{V}))$, where $|\mathbb{E}|$ represents the number of edges and $|\mathbb{V}|$ represents the number of vertices in the graph. The algorithm's time complexity scales linearly with the number of edges and has a logarithmic dependence on the number of vertices in the graph.

5.3.2 E2E Latency Awareness for Source Node Selection

The E2E latency awareness plays a crucial role in routing for latency-sensitive applications. Section 2.2.3 shows that only a few works address routing with E2E latency awareness. Therefore, it is essential to consider the latency of the edge infrastructure supporting the deployment of VR services because 6DoF VR applications are very latency-sensitive. Any reduction in the E2E latency can improve the content delivery while maintaining the desired QoS. Unlike related works and regular routing approaches, FLATWISE provides an adaptive routing scheme, where it can squeeze or relax the selection of the traversed node's n neighbor in each iteration based on the achieved latency of the path r_i to reach node n and the target latency $\overline{\delta}(f_i)$ required by flow f_i.

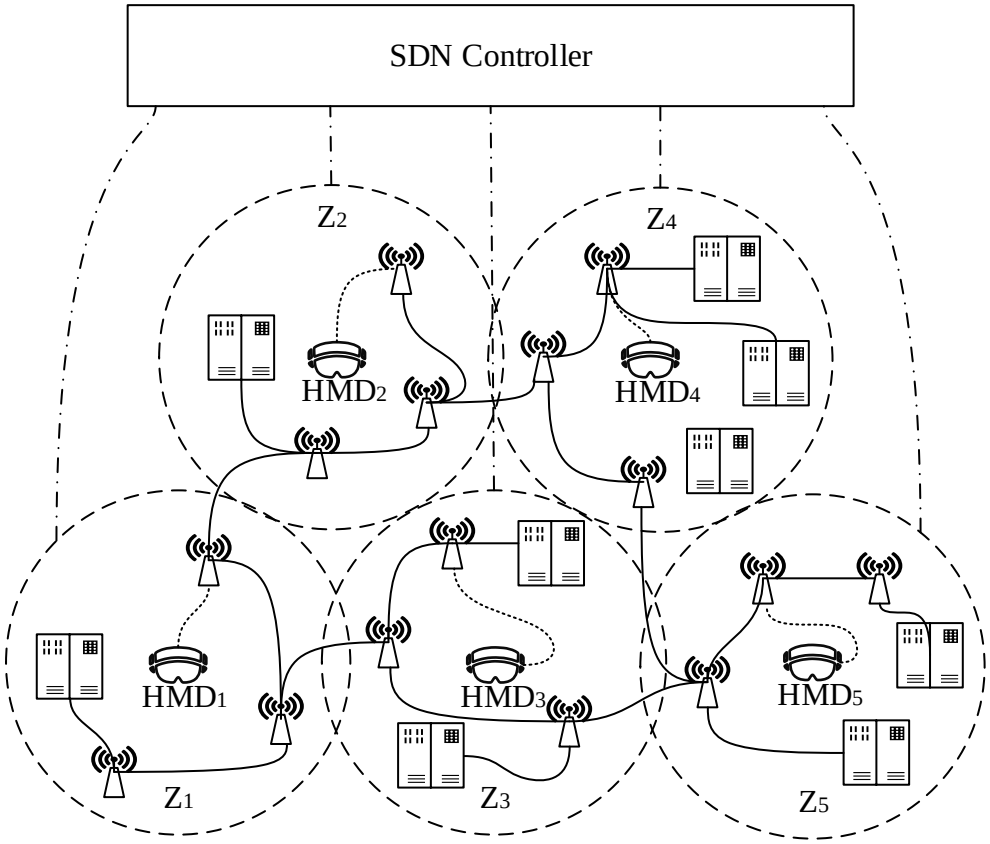


FIGURE 5.2: FLATWISE zone scheme to support routing with E2E latency awareness to select the optimal source node, from which the MEC server attached to it supports the offloading of VR services while minimizing the E2E latency.

To optimize such an adaptive routing scheme, FLATWISE sets an *anchor* to a nearby MEC server m_i, where the anchor hosts the VR services for HMD h_i. To select the anchor, FLATWISE considers the MEC server with the lowest computing latency $p(m_i)$ with sufficient computing resources, e.g., CPU, GPU, RAM, and storage, to support the deployment of VR services from HMD h_i. Instead of establishing the path between the base station where HMD h_i is connected and the cloud server m_j hosting the VR application logic, FLATWISE establishes the path between base station v_i, where the anchor is attached, and the cloud server.

For each HMD h_i, we consider all MEC servers attached to base stations v_i within twice the HMD's coverage area to ensure a higher heterogeneity of available MEC servers. Therefore, we define the coverage area mentioned above as a zone z_n. Figure 5.2 shows the proposed zone scheme for each HMD, where the selection of the optimal source node is provided according to the zone z_n specified for each HMD h_i. After defining the zone scheme shown in Figure 5.2, FLATWISE selects the MEC servers considering their resource availability, computing latency $p(m_i)$, HMD's h_i location, HMD's h_i mobility patterns, and network congestion. Therefore, FLATWISE calculates a path from the base station v_i and the cloud server m_j hosting the 6DoF VR back-end, where the chosen MEC server m_i (*anchor*) is attached to base station v_i. After that, FLATWISE establishes the path from v_i to the base station where the HMD h_i is connected. Thus, FLATWISE ensures lower E2E latency compared to state-of-the-art routing approaches while providing network load balancing through a heterogeneity path selection for VR flows.

5.3.3 FLATWISE Dynamic Path Calculation compared to Widest Shortest and Shortest Widest Paths

One of the unique features of FLATWISE is its ability to provide different paths between the same source s and destination t, regardless of network conditions. FLATWISE provides different paths based on the latency requirements $\overline{\delta}(f_i)$ of each 6DoF VR application. Thus, FLATWISE can relax or squeeze the search for new paths between s and t based on latency $\overline{\delta}(f_i)$. For instance, the lower the latency $\overline{\delta}(f_i)$ is, the tighter is the search for low-latency paths, following the same logic as shortest path-based approaches. On the other hand, the higher the latency $\overline{\delta}(f_i)$ is, the more relaxed the search is, which means that paths with a higher number of hops can be used for a 6DoF VR flow. As a result, considering the features described in Section 5.3.2, FLATWISE establishes a dynamic on-demand path calculation based on the E2E latency for each VR flow, where it can provide different paths between any source s and destination t according to the flow latency requirement $\overline{\delta}(f_i)$.

Let the network graph in Figure 5.3 be an explanatory example, where $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ is the network graph and each edge (i, j) has a maximum network throughput b_{ij} and network latency l_{ij}. Let us consider node $s \in \mathbb{V}$ as the source node and $t \in \mathbb{V}$ as the destination node, where $t \neq s$. Tables 5.1 and 5.2 show two distinct scenarios where the baseline routing allocation algorithms WSP and SWP are compared to the FLATWISE approach. Besides, both tables show the flow f_i being processed, flow latency requirement $\overline{\delta}(f_i)$, flow throughput

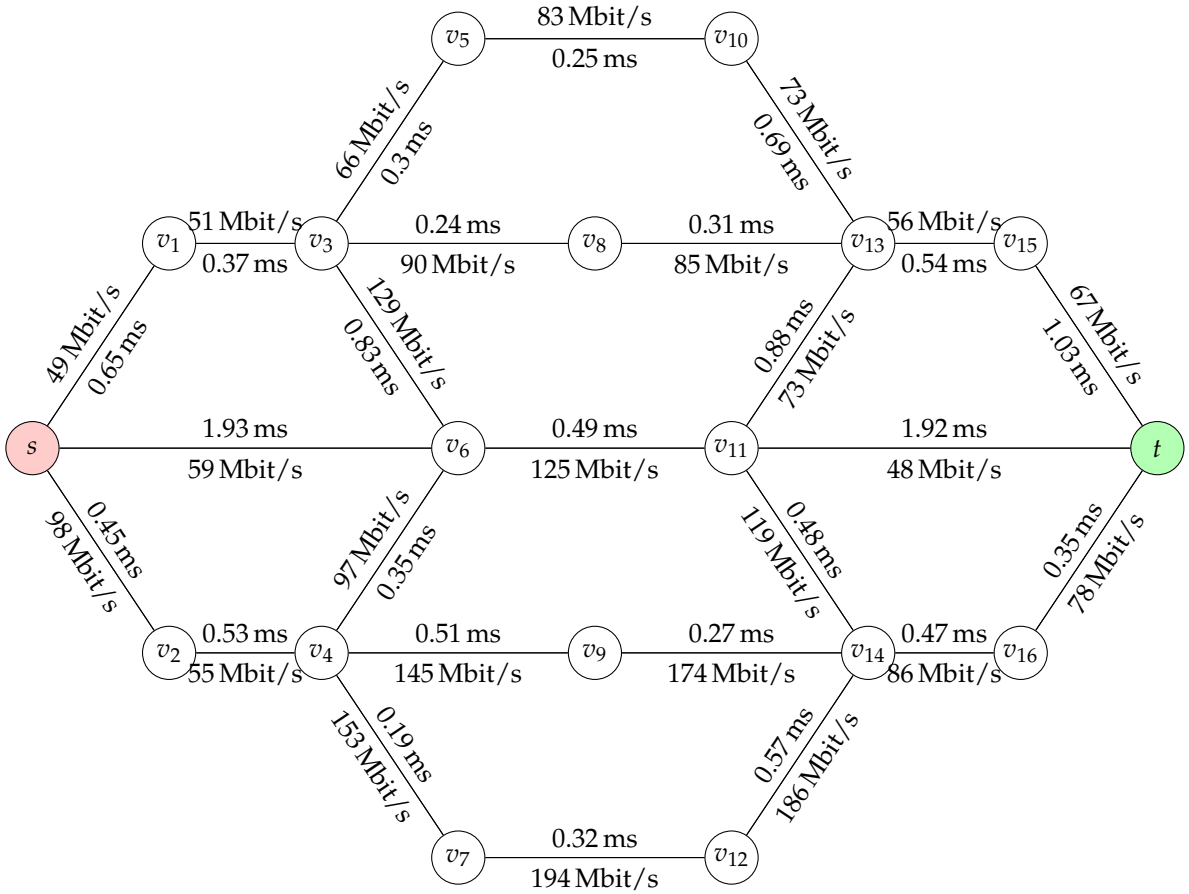


FIGURE 5.3: Network graph with network latencies and throughputs for each edge $(i, j) \in \mathbb{E}$, where s and t represent the source and destination nodes.

Routing algorithm	f_i	$\overline{\delta}(f_i)$	$\overline{\omega}(f_i)$	$\delta(r_i)$	$\omega(r_i)$	Path (r_i)
WSP	f_1	$\varnothing$	30 Mbit/s	2.58 ms	54 Mbit/s	$s \to v_2 \to v_4 \to v_9 \to v_{14} \to v_{16} \to t$
SWP	f_1	$\varnothing$	30 Mbit/s	3.88 ms	59 Mbit/s	$s \to v_6 \to v_4 \to v_9 \to v_{14} \to v_{16} \to t$
FLATWISE	f_1	3 ms	30 Mbit/s	2.58 ms	54 Mbit/s	$s \to v_2 \to v_4 \to v_9 \to v_{14} \to v_{16} \to t$
FLATWISE	f_1	4 ms	30 Mbit/s	2.88 ms	54 Mbit/s	$s \to v_2 \to v_4 \to v_7 \to v_{12} \to v_{14} \to v_{16} \to t$
FLATWISE	f_1	5 ms	30 Mbit/s	3.12 ms	54 Mbit/s	$s \to v_2 \to v_4 \to v_6 \to v_{11} \to v_{14} \to v_{16} \to t$
FLATWISE	f_1	6 ms	30 Mbit/s	3.14 ms	49 Mbit/s	$s \to v_1 \to v_3 \to v_8 \to v_{13} \to v_{15} \to t$
FLATWISE	f_1	7 ms	30 Mbit/s	3.64 ms	49 Mbit/s	$s \to v_1 \to v_3 \to v_6 \to v_{11} \to v_{14} \to v_{16} \to t$
FLATWISE	f_1	8 ms	30 Mbit/s	3.72 ms	59 Mbit/s	$s \to v_6 \to v_{11} \to v_{14} \to v_{16} \to t$
FLATWISE	f_1	9 ms	30 Mbit/s	4.34 ms	48 Mbit/s	$s \to v_6 \to v_{11} \to t$

TABLE 5.1: Widest Shortest Path, Shortest Widest Path, and FLATWISE flow processing of flow f_1, where $f_1(s, t, \varnothing, 30\,\text{Mbit/s})$.

requirement $\overline{\omega}(f_i)$, path latency $\delta(r_i)$, minimum throughput along the path $\omega(r_i)$, and the calculated path r_i provided by each algorithm.

In particular, Table 5.1 describes the first scenario, which considers the current status of the network, as shown in Figure 5.3. Let us consider flow f_1, where $\overline{\delta}(f_1)$ and $\overline{\omega}(f_1)$ represent the flow latency and throughput requirement, respectively, where $\overline{\delta}(f_1) \leftarrow \varnothing$ and $\overline{\omega}(f_1) \leftarrow 30\,\text{Mbit/s}$. Therefore, there is no flow latency requirement in this example. WSP, SWP, and FLATWISE calculate and allocate the path for flow f_1 merely based on throughput $\overline{\omega}(f_1)$, the current approach adopted in Internet routing protocols. While WSP and SWP only calculate one single path or multiple paths with the same latency or throughput weight, FLATWISE provides different paths for flow f_1 according to the flow latency $\overline{\delta}(f_1)$ specification. We show that for flow f_1 with latency requirement $\overline{\delta}(f_1)$ ranging from 3 ms to 9 ms, FLATWISE ensures the flow throughput $\overline{\omega}(f_1)$ of 25 Mbit/s while providing different suitable paths that can be allocated according to the requirement of each flow in the network.

Routing algorithm	f_i	$\overline{\delta}(f_i)$	$\overline{\omega}(f_i)$	$\delta(r_i)$	$\omega(r_i)$	Path (r_i)
WSP	f_1	$\varnothing$	25 Mbit/s	2.58 ms	55 Mbit/s	$s \rightarrow v_2 \rightarrow v_4 \rightarrow v_9 \rightarrow v_{14} \rightarrow v_{16} \rightarrow t$
WSP	f_2	$\varnothing$	56 Mbit/s	4.87 ms	56 Mbit/s	$s \rightarrow v_6 \rightarrow v_{11} \rightarrow v_{13} \rightarrow v_{15} \rightarrow t$
SWP	f_1	$\varnothing$	25 Mbit/s	3.88 ms	59 Mbit/s	$s \rightarrow v_6 \rightarrow v_4 \rightarrow v_9 \rightarrow v_{14} \rightarrow v_{16} \rightarrow t$
SWP	f_2	$\varnothing$	55 Mbit/s	4.28 ms	55 Mbit/s	$s \rightarrow v_2 \rightarrow v_4 \rightarrow v_6 \rightarrow v_3 \rightarrow v_8 \rightarrow v_{13} \rightarrow v_{15} \rightarrow t$
FLATWISE	f_1	6 ms	25 Mbit/s	3.14 ms	49 Mbit/s	$s \rightarrow v_1 \rightarrow v_3 \rightarrow v_8 \rightarrow v_{13} \rightarrow v_{15} \rightarrow t$
FLATWISE	f_2	3 ms	55 Mbit/s	2.88 ms	55 Mbit/s	$s \rightarrow v_2 \rightarrow v_4 \rightarrow v_7 \rightarrow v_{12} \rightarrow v_{14} \rightarrow v_{16} \rightarrow t$

TABLE 5.2: Widest Shortest Path, Shortest Widest Path, and FLATWISE flow processing with throughput guarantees for flows f_1 and f_2, where $f_1(s, t, 6\,\text{ms}, 25\,\text{Mbit/s})$ and $f_2(s, t, 3\,\text{ms}, 55\,\text{Mbit/s})$.

Table 5.2 describes the second scenario, representing a 6DoF VR scenario where flows require latency and throughput requirements. This second scenario considers the current status of the network, as shown in Figure 5.3. Let us consider flows f_1 and f_2, where flow f_1 arrives first in the network and flow f_2 arrives after flow f_1. WSP, SWP, and FLATWISE must calculate the path r_i for flows f_1 and f_2 and allocate the throughput for each path r_i, where the latency and throughput requirements for flow f_1 are $\overline{\delta}(f_1) \leftarrow 6\,\text{ms}$, $\overline{\omega}(f_1) \leftarrow 25\,\text{Mbit/s}$ and for flow f_2 are $\overline{\delta}(f_2) \leftarrow 3\,\text{ms}$ and $\overline{\omega}(f_2) \leftarrow 55\,\text{Mbit/s}$. It is worth noting that WSP and SWP calculate the paths only considering the flow throughput, whereas FLATWISE considers both flow latency and throughput requirement to calculate a path for each flow.

First, WSP, SWP, and FLATWISE process f_1 and allocate 25 Mbit/s for flow f_1, making some edges unavailable to support the allocation of 55 Mbit/s for flow f_2. WSP and SWP allocate a path for flow f_1 with latency of 2.58 ms and 3.88 ms, respectively. However, flow f_1 requires a network latency of 6 ms. Besides, after processing flow f_2, algorithms WSP and SWP found a path with 4.87 ms and 4.28 ms, respectively. This indicates that WSP and SWP did not

allocate a suitable path for flows f_1 and f_2 in terms of network latency. Therefore, flow f_2 was impaired, as there were no available paths to meet its latency requirements because algorithms WSP and SWP allocated a path with a lower latency than the latency required by flow f_1. In contrast, FLATWISE allocates suitable paths to each flow based on their latency requirements. In practice, FLATWISE allocates paths for flows f_1 and f_2 with 3.14 ms and 2.88 ms, respectively. This indicates that the latency awareness during the path-searching decision process makes FLATWISE suitable for 6DoF VR scenarios that require both latency and throughput guarantees. Therefore, this practical example shows that using WSP and SWP always to calculate the shortest or widest path does not necessarily optimize the overall network latency for all flows deployed in the network. This happens because allocating a path for a specific flow f_i with latencies much lower than that requested by the flow can harm flows arriving at the network later and not finding available paths that meet the latency requirements for a given flow.

5.4 Experiment Setup

This section describes the experiment setup used in FLATWISE evaluation. We discuss the network requirements for support VR applications, including video resolutions, frame rates, latency, throughput, and VR services. Furthermore, we describe the edge network graphs, the evaluation scenarios, the VR flow management, the baseline routing algorithms used in the evaluation, the VR users' mobility, and the VR use case. Finally, we describe the performance metrics used in our evaluation.

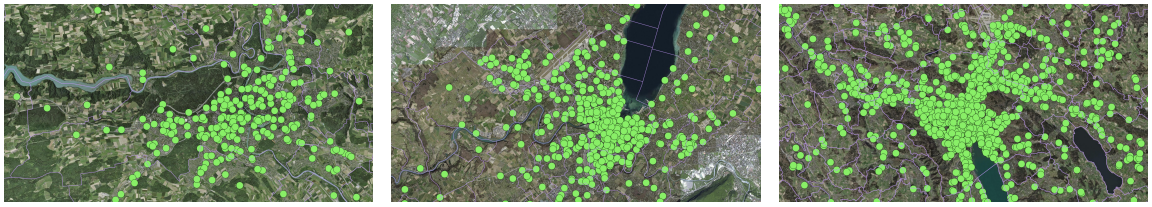
5.4.1 Network Requirements to support VR applications

The development of VR is centered around experience, especially improvement in profile, interactive, and immersive experiences. The compatibility between transmission and network technologies determines the level of immersive experience that 6DoF VR can deliver. In this scenario, network requirements vary along with interaction modes, where the network infrastructure must meet the requirements of VR services to ensure a high interaction experience. *Weak-interaction* VR places high requirements on throughput, e.g., Image Maximum Theatre, 360° panoramic video, and VR live broadcast, while *strong-interaction* VR places high requirements on both throughput and latency, e.g., VR games, VR home fitness, and VR social networking. We follow the four VR stages proposed in [7]. Table 5.3 reports the recommended experience duration, VR video resolutions, color depths, frame rates, bitrates, network throughput, and latency that we consider for video services deployed in our scenario.

5.4.2 Edge Network Graphs

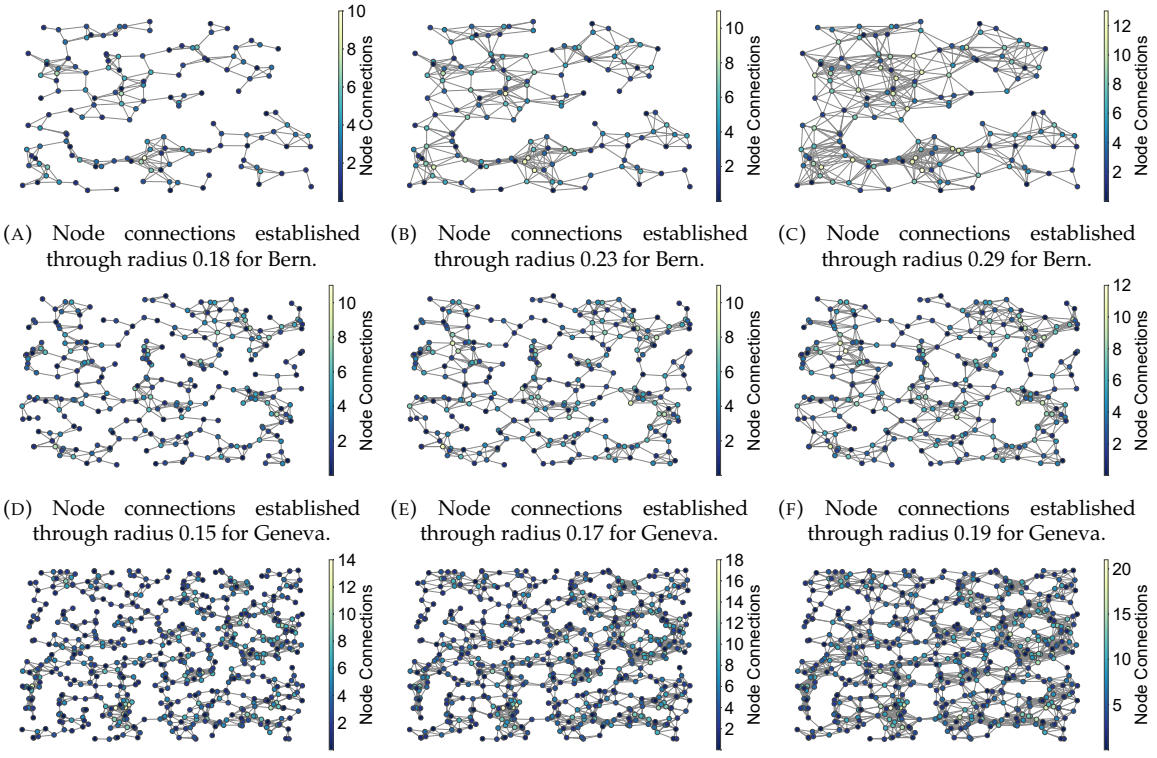
We use different edge network topologies for the simulation. We use real 5G edge network topologies for three cities, Bern (BE), Geneva (GE), and Zurich (ZH) [138]. The original 5G network infrastructures are shown in Figure 4.4. Bern has an area of 51.6 km^2 with 147 nodes

and a node density of 2.84 nodes/km^2. Geneva has an area of 15.93 km^2 with 269 nodes and a node density of 16.88 nodes/km^2. Zurich has an area of 87.88 km^2 with 586 and a node density of 6.66 nodes/km^2.



(A) Bern 5G base stations. (B) Geneva 5G base station. (C) Zurich 5G base stations.

FIGURE 5.4: Physical 5G network infrastructure map of the cities of Geneva, Bern, and Zurich.



(A) Node connections established through radius 0.18 for Bern. (B) Node connections established through radius 0.23 for Bern. (C) Node connections established through radius 0.29 for Bern.

(D) Node connections established through radius 0.15 for Geneva. (E) Node connections established through radius 0.17 for Geneva. (F) Node connections established through radius 0.19 for Geneva.

(G) Node connections established through radius 0.16 for Zurich. (H) Node connections established through radius 0.19 for Zurich. (I) Node connections established through radius 0.22 for Zurich.

FIGURE 5.5: Generated 5G network infrastructure connectivity of the cities of Bern, Geneva, and Zurich over different radii.

Each generated topology is based on a Cartesian plane, where the nodes are distributed between the coordinates $(0,0)$ and $(1,1)$. We define each base station's coverage area as a *radius*. Therefore, if the coverage area between two base stations overlaps, we generate a link between them. The links between base stations are established whenever the Euclidean distance between any two base stations in the scenario is not greater than a specific radius. The latency of each established link between two base stations is uniformly distributed between 0.5 ms and 1 ms, where the link latency can vary between 1% and 5% over time. In each city, the base stations are located at positions illustrated in Figure 5.4. We define the

aforementioned latency distribution according to latency measurements in the University of Bern's internal network infrastructure. In our scenario, 70% of the base stations of each topology are directly attached to MEC servers, which offer different GPU, CPU, memory, storage, and throughput resources. Around 80% of the MEC servers have GPUs. Figure 5.5 shows the generated topologies and their links. Unlike the topologies in Figure 4.5, we consider more radii and use the network throughput and latency, as shown in Figure 5.3.

5.4.3 Evaluation Scenarios

We consider the network topology of the cities of Bern, Geneva, and Zurich. We use different radii to provide the node connectivity to set different user densities under varying network conditions and traffic load patterns for each network topology. Thus, for each topology, there are different levels of connectivity, throughput availability, and latency according to the radius used. Therefore, we defined a different number of users for each topology according to their number of nodes. The topology of Bern has 1000 6DoF VR applications with a radius ranging from 0.12 to 0.20. The topology of Geneva has 3000 6DoF VR applications. We use the radius ranging from 0.09 to 0.17. The topology of Zurich has 5000 6DoF VR applications, with a radius ranging from 0.06 to 0.14 for this topology. The topologies used in our evaluation scenario are important to understand the limitations of each routing algorithm and scalability issues.

5.4.4 VR Flow Management

We have implemented a flow management mechanism that aims to provide fairness to process all flows in the network during the simulations. Before starting each simulation, we assigned a random resolution to all the flows deployed in the network according to the requirements and resolutions described in Table 5.3. When the simulations are started, the routing algorithms try to improve the resolution of the flows over time. When the network cannot support a more advanced resolution than the one the HMD h_i runs, a resolution is configured according to the available network resources, such as throughput and latency. The flow management is handled via the SDN controller in all evaluation scenarios described in Section 5.4.3. The flows are shuffled at each iteration to provide fairness during the flow-processing procedure. Flow shuffling is essential because we congest the network in all evaluation scenarios to allocate alternative paths. Thus, we can examine the behavior of the algorithms during a network overload scenario. Therefore, the order in which flows are processed depends on the shuffling procedure in each iteration.

5.4.5 Baseline Routing Algorithms

For the above scenarios, we consider two approaches to allocating paths for VR applications in which HMDs use a throughput-based algorithm to increase the bitrate of 6DoF videos.

- SWP: The widest path is determined first. Assume there are multiple paths between a source and a destination. In that case, the second attribute of the additive cost is applied to

TABLE 5.3: Network KPI requirements in different phases of VR implementation.

Standard		Pre-VR	Entry-Level VR	Advanced VR	Ultimate VR
Experience duration		$\leq$ 20 min.	$\leq$ 20 min.	20 to 60 min.	over 60 min.
Video resolution		Full-view 4K 2D video (resolution 3840 x 1920)	Full-view 8K 2D/3D video (resolution 7680 x 3840)	Full-view 12K 3D video (resolution 11520 x 5760)	Full-view 24K 3D video (resolution 23040 x 11520)
Color depth		8 bits	8 bits	10 bits	12 bits
Frame rate		30 to 90 FPS	30 to 90 FPS	60 to 120 FPS	120 to 200 FPS
Week interaction VR Services	(Bitrate)	16 Mbit/s	FOV 42 Mbit/s	FOV 220 Mbit/s	FOV 1.56 Gbit/s
	(Throughput)	25 Mbit/s	FOV 63 Mbit/s	FOV 340 Mbit/s	FOV 2.34 Gbit/s
	(RTT)	30 ms	20 ms	20 ms	10 ms
Strong interaction VR Services	(Bitrate)	18 Mbit/s	FOV 60 Mbit/s	FOV 390 Mbit/s	FOV 1.68 Gbit/s
	(Throughput)	50 Mbit/s	FOV 200 Mbit/s	FOV 1.40 Gbit/s	FOV 3.36 Gbit/s
	(RTT)	10 ms	10 ms	5 ms	5 ms

determine the list cost path among the multiple widest paths [140]. We consider the latency as the additive cost.

- WSP: The shortest path is determined first. Assuming multiple shortest paths exist, we choose the one with the largest width. Combining the shortest and widest paths into SWP or WSP is possible if a communication network needs to consider multiple metrics, such as QoS routing, where it is essential to consider two cost attributes, one additive and the other non-additive [140], [141].

5.4.6 VR Users Mobility

We use Mininet-WiFi to simulate a realistic network scenario and user mobility [142]. We use ONOS[1] SDN controller to provide flow control, bandwidth allocation, and mobility management for the simulated VR services. The simulated scenario covers the area of the cities of Bern, Geneva, and Zurich. Besides, each network topology contains a variable number of mobile VR users that can connect to the RAN via their 5G interface. We assume that each VR user runs exactly one VR application. The base stations transmit signals with a 50 dBm power, decaying according to the Free Space Path Loss model. The VR users' mobility follows the Random Direction Model, in which users move along a straight line. We

[1]https://opennetworking.org/onos/

assume that mobile VR users connect to the base station whose signal is received with the highest Signal-to-Noise Ratio (SNR). We assume that each VR user executes a single 6DoF VR application made of decoding services. For each 6DoF VR application, we uniformly distribute between 5 and 10 decoders to observe how computing requirements affect system performance. Furthermore, each 6DoF VR application contains a service to aggregate the chunks of VR video decoded by each decoder service. For each decoder in the system, its equivalent requirements in terms of CPU and GPU are randomly extracted from two uniform distributions with averages of 1770 MHz for the CPU and 440 MHz for the GPU, based on the typical requirements of regular HMDs available at the market.

5.4.7 VR In-Game Communication Use Case

We evaluate SWP and WSP against FLATWISE in the *VR in-game communication use case*, where a central cloud node is processing the inputs from all players. In in-game communications, players in an online virtual game hear and talk to the players within their immediate vicinity inside the virtual game. Using an HMD, the rendering can be controlled by head tracking so that the positions of other players are updated as the user turns the player head [76]. Essentially, the evaluated algorithms allocate paths from the base station v_i in which HMD h_i is connected and the cloud server m_j hosting the VR application back-end. HMD users may change their location in each iteration, and a new path must be allocated. Otherwise, the current path of HMD h_i can be used to transmit and receive the data from cloud server m_j.

5.4.8 Performance Metrics

The performance of each routing algorithm is evaluated by executing it in the simulated scenario for 15 hours and measuring the average E2E latency for the user applications. We assume that time is partitioned in a series of consecutive time windows of duration $T = 5\,\text{s}$, and we will measure a value of latency per window. This choice for T gives sufficient time for all algorithms to converge so that the experiment yields 10 000 measurements for latency over the 15 simulated hours. We consider the following KPIs in our evaluation scenario:

- *E2E latency*. Each user executes an ICMP `ping` command along the core network part and uses the collected data to compute the average core network latency during a time window. During the same time window, for each user, we measure the average computing latency as the sum of the computing latency of each of its services deployed on MEC servers or HMD. Each user has an average E2E latency for that time window, the sum of the average core network latency, the average computing latency, and the benchmarked wireless latency. The average E2E latency is the average E2E latency for the time window across all users.
- *Flow network latency* measures the network latency requirement of a specific flow based on the video resolution and frame rate configured for the flow.
- *Flow path latency* measures the time data packets travel from the source node s to the destination node t across a network. A path is allocated for each flow based on the *flow network latency* KPI, where *flow path latency* $\leq$ *flow network latency*.

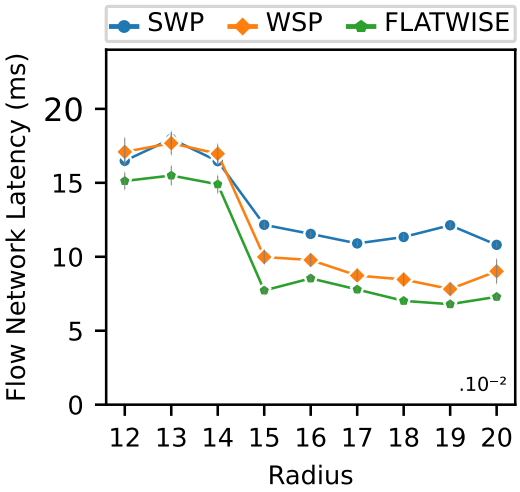
- *Over-provisioned latency* measures the difference between *flow network latency* and *flow path latency*. In most cases, the *flow network latency* KPI is way higher than the *flow path latency*, indicating how much latency optimization can be provided to all flows by allocating the most suitable route according to the *flow network latency*.
- *Network throughput* refers to the distribution of network resources for data transfer. It measures how much throughput is allocated to different flows on the network.
- *Frame rate* measured in *Frames per Second* (FPS) is used to quantify the smoothness of HMD video playback or animation. It represents the number of frames displayed per second in a video or animation sequence.
- *Video resolution*. We assume that each application in the system selects a video resolution based on the E2E latency, according to the average latency at each time window. The application maintains the resolution constant for the whole window duration. In the next time window, the resolution is selected according to the available E2E latency provided by the system. Therefore, the higher the resolution is, the lower is the latency required to process the video set to this resolution.
- *Execution time* measures the time taken to calculate and allocate paths for each flow, considering the number of edges in each network scenario and the number of flows.

5.5 Performance Evaluation

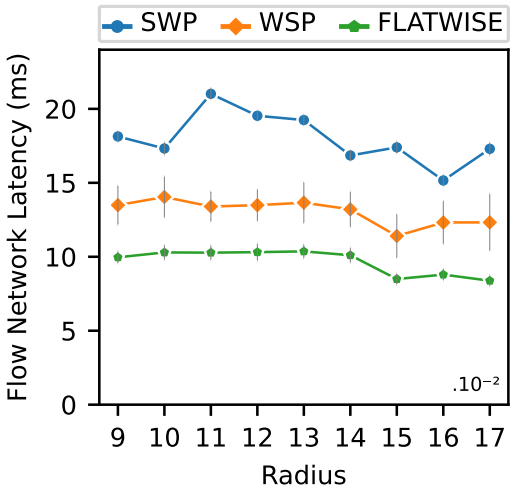
We start by investigating the impact on the Internet network infrastructure when existing routing protocols prioritize throughput as the primary metric for resource allocation. For this purpose, we employ a CDN scenario where a central node is responsible for distributing 6DoF VR videos to all users, reflecting the current content delivery approach employed by the Internet. Thus, we can analyze how different routing protocol approaches, operating under various network conditions and traffic load patterns, influence key factors such as overall network throughput allocation, network congestion, network latency, average video resolution FPS, and the achieved video resolutions for each VR application. These factors are evaluated considering that network conditions change dynamically and are driven by the decisions made by different path selection approaches.

5.5.1 Flow Network Latency

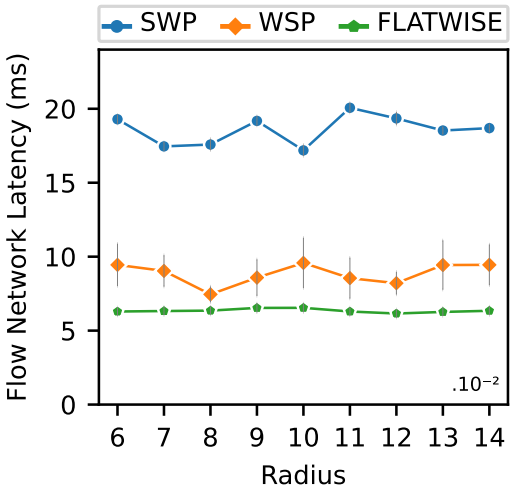
Flow Network Latency is a critical performance metric that measures the network's ability to satisfy the latency requirements of specific flows, considering the video resolution and frame rate configured for each flow. Figure 5.6 shows the average flow network latency performance evaluation for Bern, Geneva, and Zurich topologies over different radii. In Bern topology (Figure 5.6a), on average, FLATWISE provides lower flow network latencies than SWP and WSP. The results for Geneva and Zurich (Figures 5.6b and 5.6c) show a similar trend to Bern, where FLATWISE consistently provides lower flow network latencies than SWP and WSP.



(A) Average Flow Network Latency for Bern.



(B) Average Flow Network Latency for Geneva.

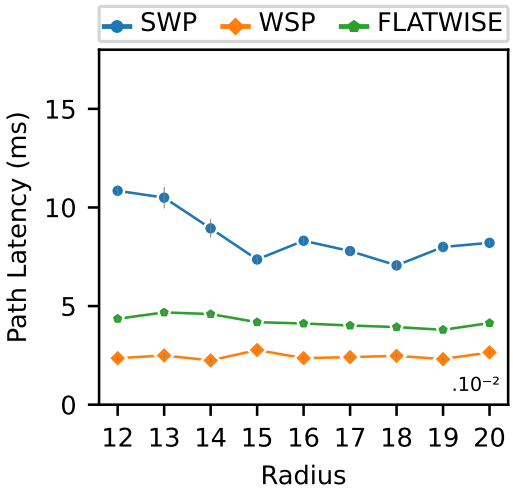


(C) Average Flow Network Latency for Zurich.

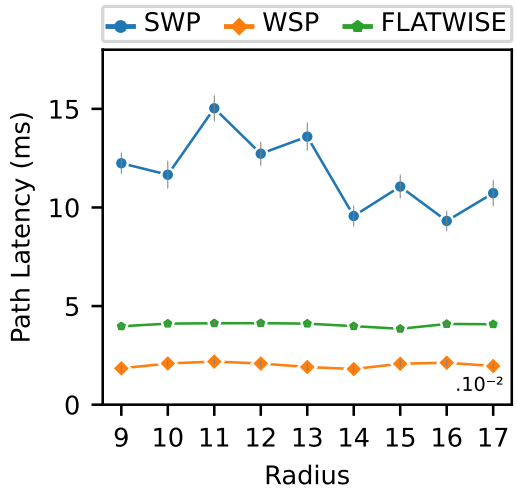
FIGURE 5.6: Performance evaluation of average flow network latency for Bern, Geneva, and Zurich topologies over different radii.

This suggests that FLATWISE efficiently allocates paths with latencies that closely match the requirements of each VR application.

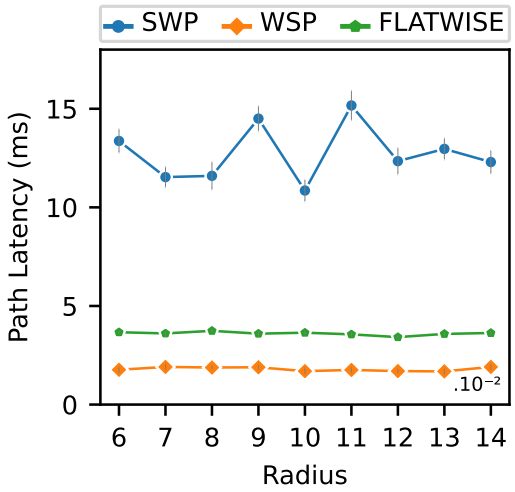
In summary, the flow network latency analysis across the three topologies highlights the effectiveness of the FLATWISE routing algorithm in minimizing latency for 6DoF VR applications. As the radius increases, the latencies generally decrease for all algorithms, showcasing improved network conditions. These findings provide valuable information on optimizing path allocation for 6DoF VR applications, as always selecting the shortest path tends to congest these paths, which leads to more flows competing for the throughput resources available on this low-latency path. In this way, more flows tend to compete for the resources of the path with the shortest distance, which prevents some flows from operating at higher transmission rates. To address this problem, FLATWISE provides a more dynamic path search, which always provides different routes according to each latency requirement.



(A) Average Flow Path Latency for Bern.



(B) Average Flow Path Latency for Geneva.

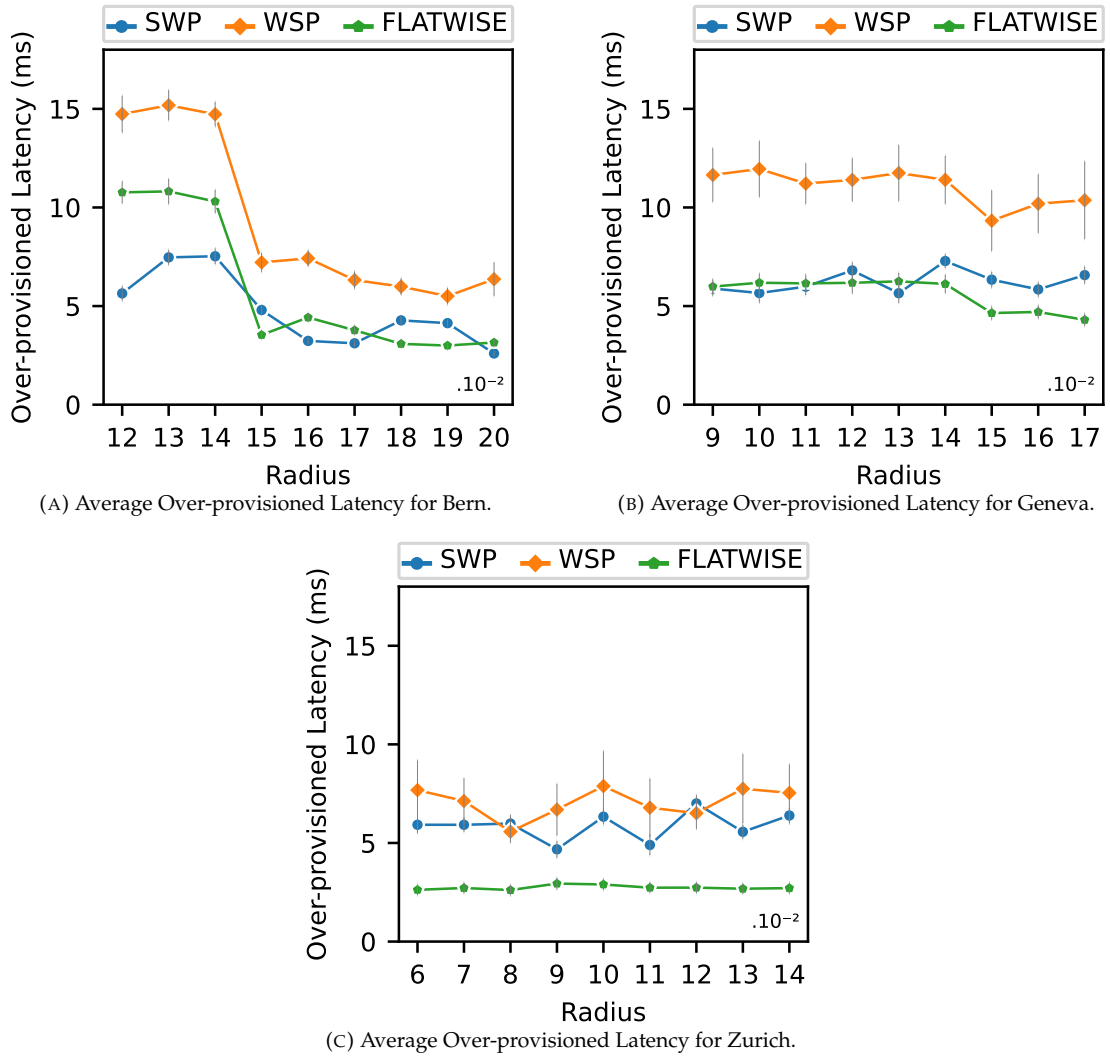


(C) Average Flow Path Latency for Zurich.

FIGURE 5.7: Performance evaluation of path latency for Bern, Geneva, and Zurich topologies over different radii.

5.5.2 Path Latency

Path latency is a crucial performance metric that quantifies the time data packets traverse from the source node to the destination node across the network, encompassing factors such as routing algorithm, network conditions, and traffic load. Figure 5.7 shows the performance evaluation of path latency for Bern, Geneva, and Zurich topologies over different radii. In Figure 5.7a, WSP provides the lowest average path latency across all radii for the Bern topology. These results suggest that WSP allocates the shortest path available for each flow arriving at the network while prioritizing the widest available path. This strategy highly depends on the requirements of the flows arriving in the network, where the first flows must have lower latency requirements than consecutive flows to guarantee the success of this approach. FLATWISE, on the other hand, balances latency and throughput allocation trade-offs. It offers competitive performance with lower latencies than SWP, emphasizing the importance of considering latency constraints in path allocation decisions. Figures 5.7b and 5.7c show that WSP consistently outperforms SWP and FLATWISE regarding average



(A) Average Over-provisioned Latency for Bern.

(B) Average Over-provisioned Latency for Geneva.

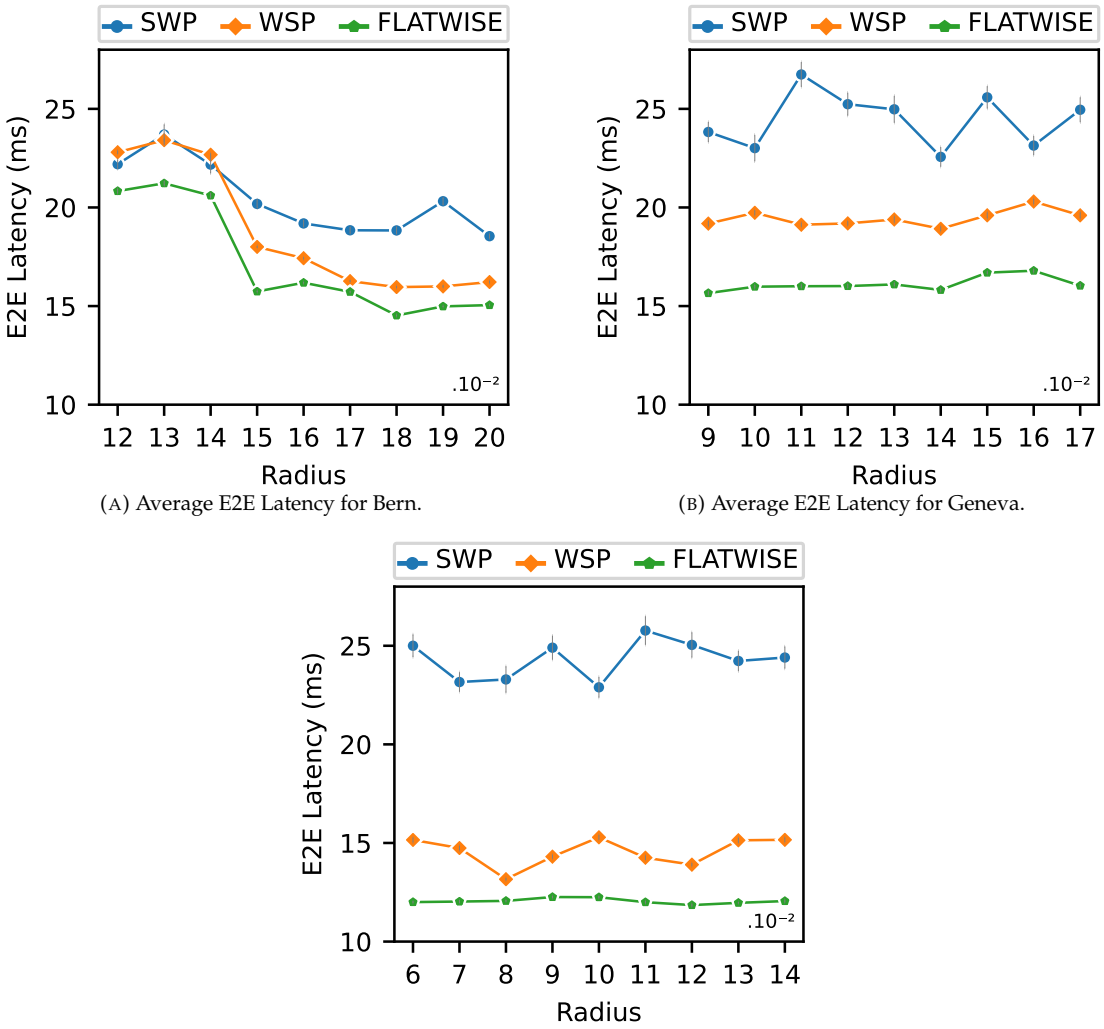
(C) Average Over-provisioned Latency for Zurich.

FIGURE 5.8: Performance evaluation of over-provisioned latency for Bern, Geneva, and Zurich topologies over different radii.

path latency for Geneva and Zurich topologies. The path latency analysis across the three topologies consistently highlights the superior performance of the WSP routing algorithm in allocating routes with lower latency. However, this does not indicate that WSP supports flows with higher flow network latency than FLATWISE because WSP efficiency highly depends on flows with lower latency requirements arriving in the network first. Otherwise, WSP will allocate ultra-low latency paths for flows that demand way higher latencies than those path latencies allocated.

5.5.3 Over-provisioned Latency

Over-provisioned latency represents the difference between the flow latency and the latency of the selected path. It shows the efficiency of all routing algorithms to allocate paths with latency as close as the latency required by the flow. The higher the over-provisioned latency is, the higher is the difference between the latency of the path allocated for the flow and the latency required by the flow. Figure 5.8 shows the performance evaluation



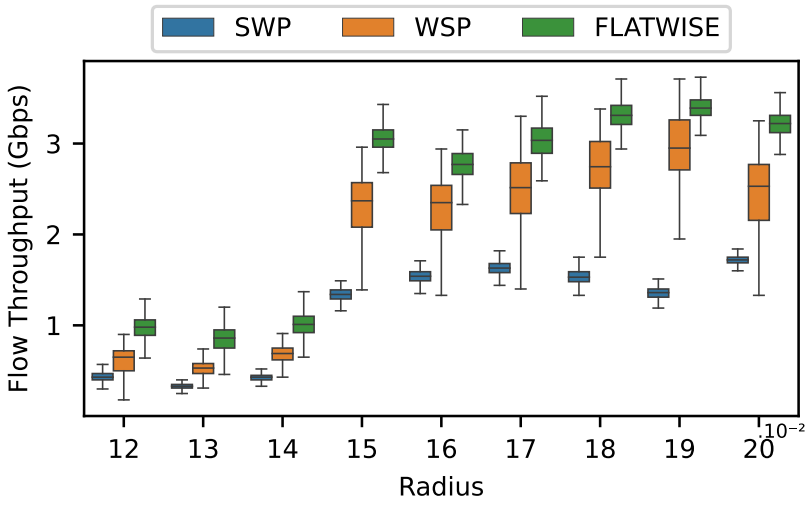
(A) Average E2E Latency for Bern.

(B) Average E2E Latency for Geneva.

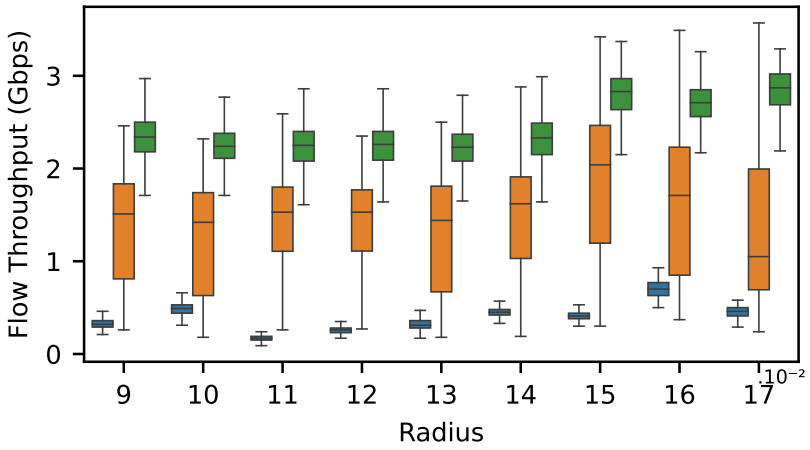
(C) Average E2E Latency for Zurich.

FIGURE 5.9: Performance evaluation of E2E latency for Bern, Geneva, and Zurich topologies over different radii.

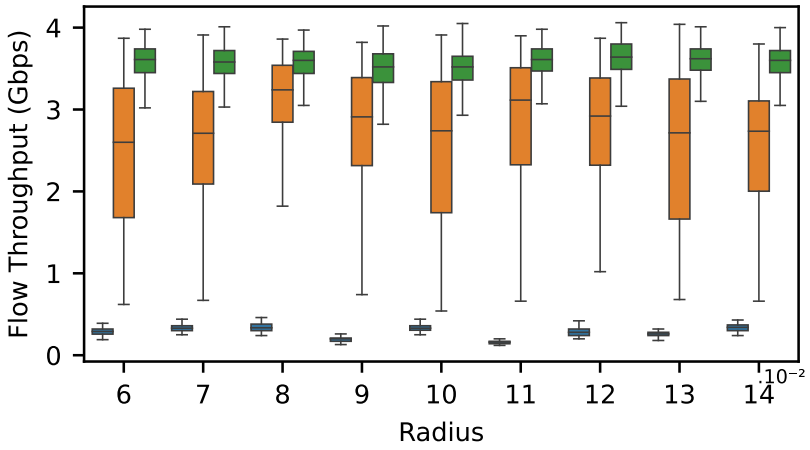
of over-provisioned latency for Bern, Geneva, and Zurich topologies over different radii. Figure 5.8a shows WSP provides the highest over-provisioning latency. With fewer edges on the Bern network topology, which is the case of a radius ranging from 0.12 to 14, FLATWISE provides higher over-provisioning latency than SWP because fewer paths with lower latency are available. Therefore, SWP selects more paths with higher latency than FLATWISE and WSP. Figure 5.8b shows that FLATWISE and SWP behave similarly when the Geneva network has fewer edges. However, FLATWISE provides a lower over-provisioned latency in scenarios with more edges. Results in Figure 5.8c suggest that FLATWISE consistently provides the lowest over-provisioned latencies across different radii for Zurich topology. Therefore, FLATWISE's ability to find paths with latency close to the flow network latency requirement contributes to reducing the over-provisioned latency. The trends observed across all cities reinforce the value of considering latency when selecting routing paths.



(A) Average Flow Network Throughput for the city of Bern.



(B) Average Flow Network Throughput for the city of Geneva.



(C) Average Flow Network Throughput for the city of Zurich.

FIGURE 5.10: Performance evaluation of average network throughput for the cities of Bern, Geneva, and Zurich over different radii.

5.5.4 E2E Latency

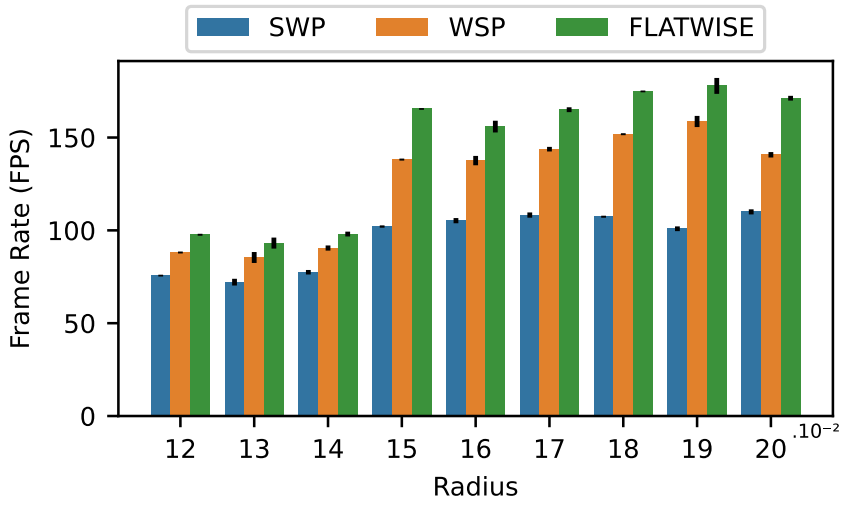
E2E latency is the sum of the flow latency, the node where the decoders are deployed, and the wireless latency of the base station where the HMD is connected. Figure 5.9 shows the performance evaluation of E2E latency for Bern, Geneva, and Zurich over different radii. The E2E latency is an important KPI for 6DoF VR applications since they require latency guarantees at different layers of the entire VR pipeline, for instance, in the network core and the computing node processing the VR video. FLATWISE achieves lower E2E latency in all topologies over different radii. Besides, the flow latency is crucial in achieving the desired E2E latency. There is a strong correlation between the flow network latency analyzed in Figure 5.6 and Figure 5.9 because the lower the flow network latency, the lower the E2E latency. Therefore, the capability of FLATWISE to optimize the flow selection by finding suitable paths with latency closest to the latency required by the flow provides lower flow network latency and E2E latency in all scenarios evaluated.

5.5.5 Flow Throughput

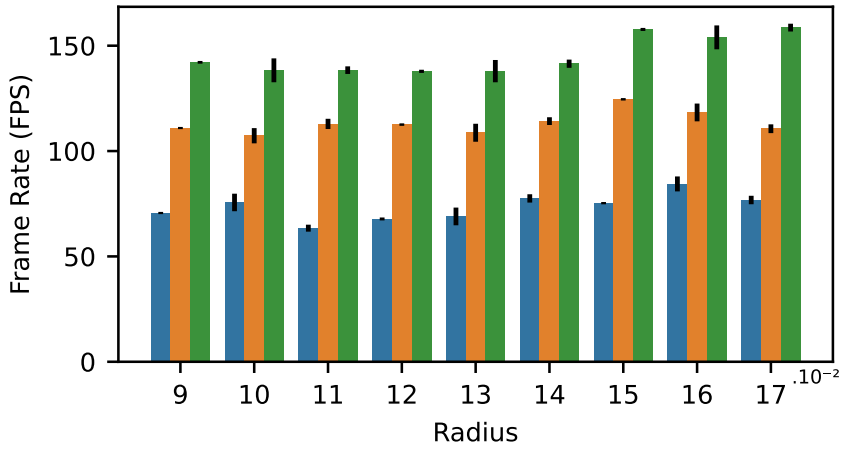
Flow throughput is a crucial performance metric determining the network's capacity to handle data traffic efficiently. Figure 5.10 shows the performance evaluation of average network throughput for the cities of Bern, Geneva, and Zurich over different radii. Wider sections in Figure 5.10 represent a higher probability that a flow is configured with the throughput specified on the y-axis of the plot, whereas the skinnier sections represent a lower probability of the flow being configured with the throughput shown in the y-axis. Figure 5.10 shows that FLATWISE achieves the highest average throughput across different radii and topologies. In particular, Figure 5.10a shows more stable average throughput results for all the algorithms. This is because fewer users are deployed in the Bern, and fewer edges are available, indicating that the scenario in Figure 5.10a is less overloaded than those in Figures 5.10b and 5.10c.

5.5.6 Frame Rate

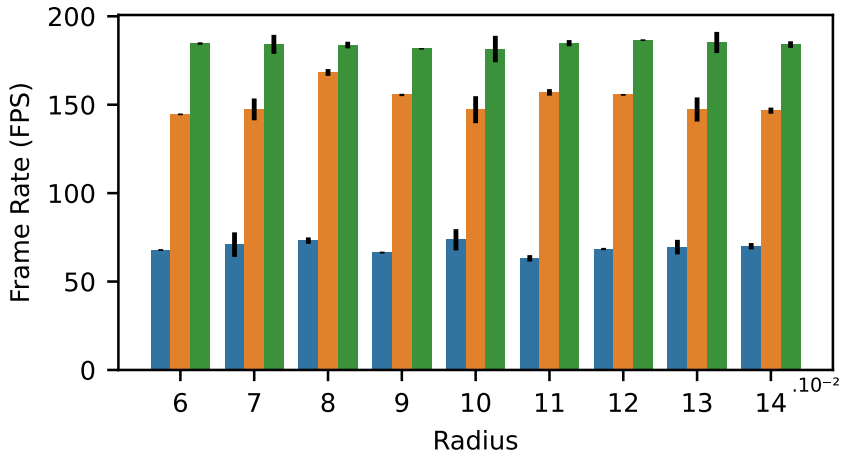
Frame rate assesses the smoothness of HMD video playback by indicating the number of frames displayed per second (FPS) within a video sequence. This is the most important KPI to measure the QoE perceived by VR users. The lower the average latency and the higher the throughput of the flow is, the higher is the frame rate of each flow, as it depends heavily on these two parameters. Figure 5.11 shows the performance evaluation of the average frame rate for the cities of Bern, Geneva, and Zurich over different radii. As expected from the flow latency and throughput analysis in Figures 5.6 and 5.10, FLATWISE outperforms both WSP and SWP by providing higher FPS in all topologies over different radii. In Figure 5.11a, the difference in FPS performance across all algorithms is more balanced. This can be seen in the flow latency and throughput analysis in Figures 5.6a and 5.10a. However, when Bern topology gets more edges, e.g., a higher radius, the results are more expressive in terms of FPS. Figures 5.11b and 5.11c show that FPS is higher for Geneva and Zurich topologies over different radii. These results indicate that even a slight improvement in flow latency can significantly affect the FPS configured for each VR application.



(A) Average Frame Rate for the city of Bern.

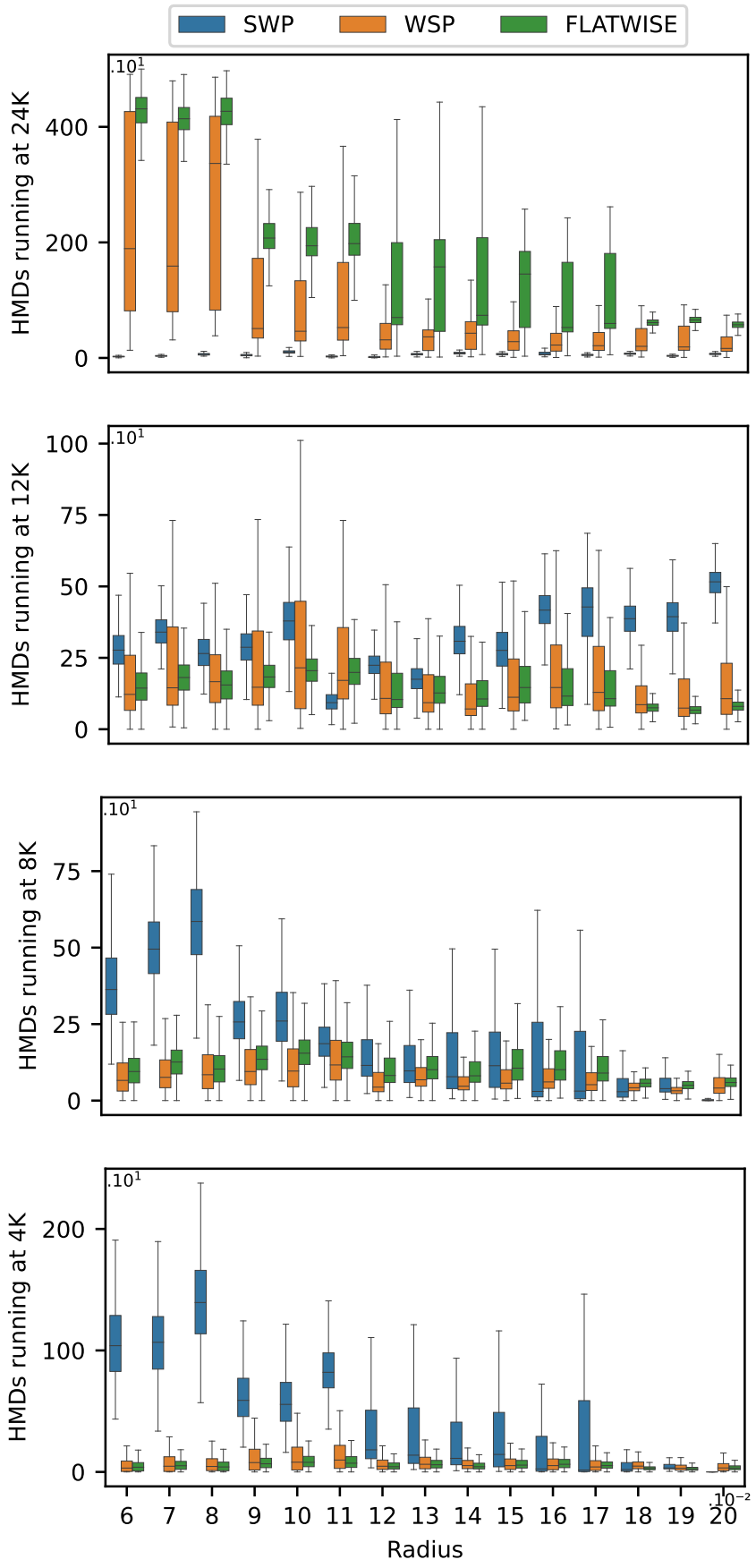


(B) Average Frame Rate for the city of Geneva.



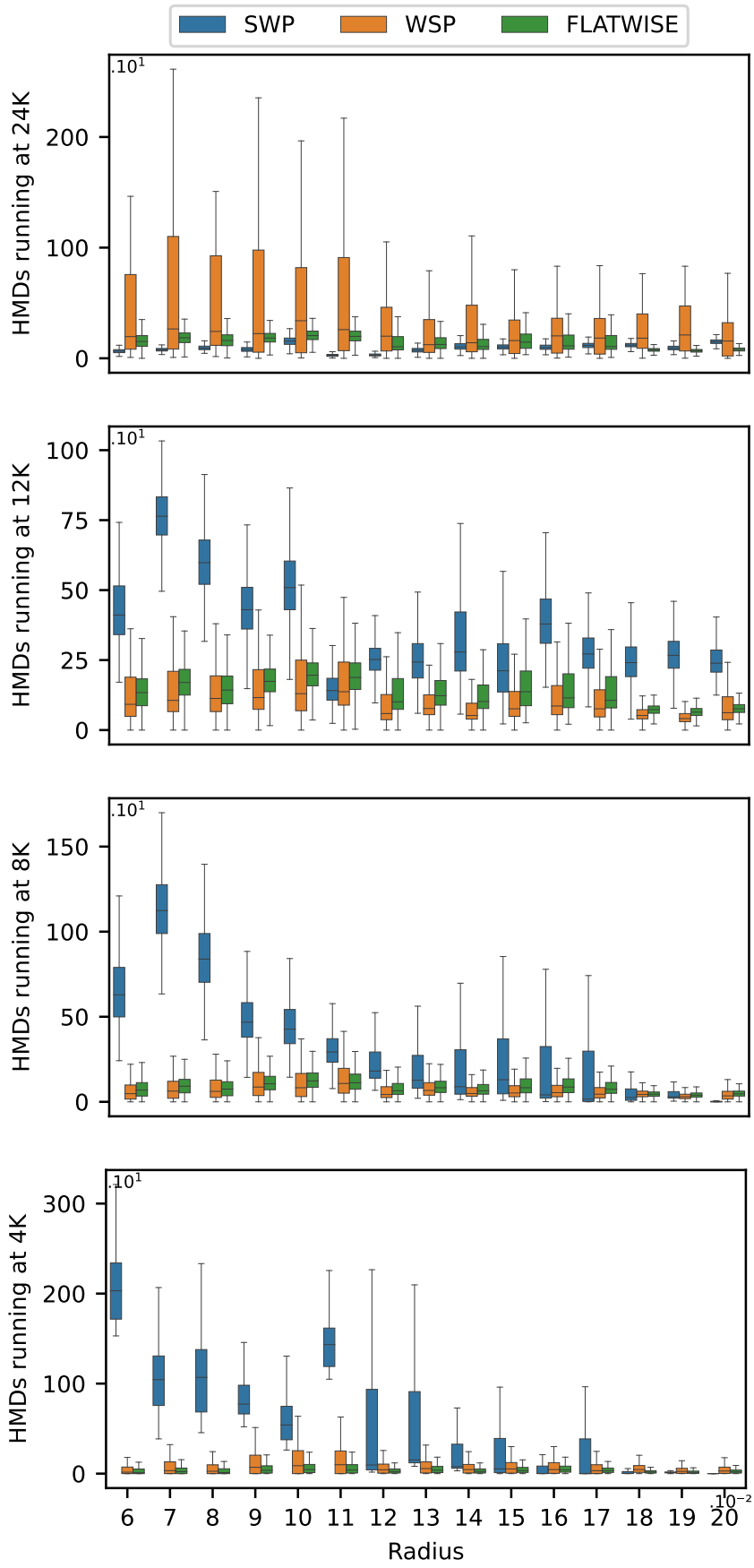
(C) Average Frame Rate for the city of Zurich.

FIGURE 5.11: Performance evaluation of average frame rate for the cities of Bern, Geneva, and Zurich over different radii.



(A) Video resolutions for strong-interaction VR services.

FIGURE 5.12: Performance evaluation of video resolutions for strong-interaction VR services for Bern, Geneva, and Zurich over different radii.



(A) Video resolutions for weak-interaction VR services.

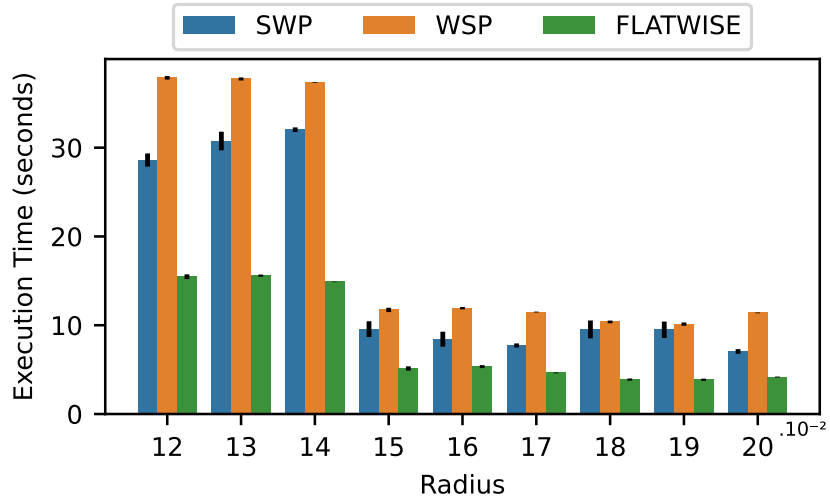
FIGURE 5.13: Performance evaluation of video resolutions for weak-interaction VR services for Bern, Geneva, and Zurich over different radii.

5.5.7 Video Resolutions

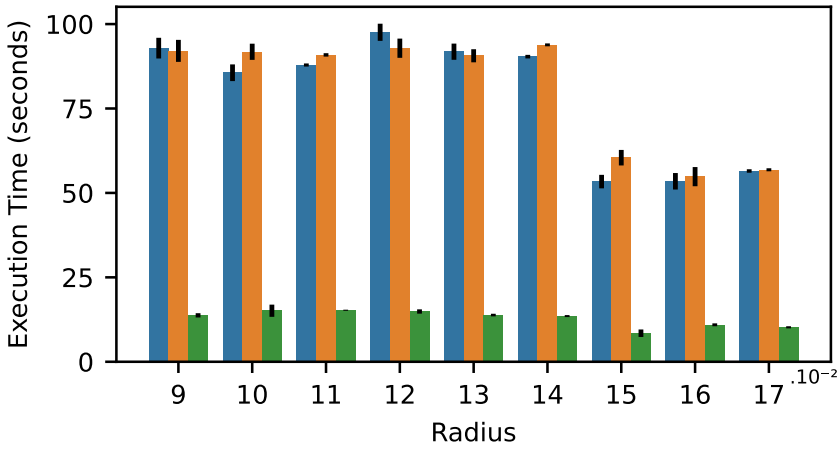
Video resolutions for 6DoF VR applications require ultra-high throughput and ultra-low latency requirements to set higher frame rates to offer VR users better experiences. Therefore, we consider the E2E latency and network throughput to switch the video resolution for each HMD during our simulations. Thus, E2E latency and throughput requirements must be met before we switch from the current resolution to a more advanced one. We categorize the video resolutions used in our evaluation into strong- and weak-interaction VR services, as described in Table 5.3. Strong-interaction VR services require higher frame rates than weak-interaction VR services, even for the same resolution. Consequently, strong-interaction VR services require lower latency and higher throughput than weak-interaction services. Figures 5.12 and 5.13 show the performance evaluation of video resolutions configured in each HMD considering strong- and weak-interaction VR services for Bern, Geneva, and Zurich topologies over the radius configured for each topology. The video resolution configured in each HMD highly depends on the flow network throughput and E2E latency. Figure 5.12 shows that FLATWISE supports strong-interaction VR services to configure more HMDs with resolution 24K than WSP and SWP, whereas WSP supports weak-interaction VR services for more HMDs running at resolution 24k. On the other hand, SWP supports more HMDs running at resolutions 12K, 8K, and 4K for both strong- and weak-interaction VR services. These results can be derived by analyzing the flow network latency, the E2E latency, and flow throughput. Therefore, the results indicate that FLATWISE's ability to provide path balancing between WSP and SWP is beneficial for providing video resolutions with higher frame rates.

5.5.8 Execution Time

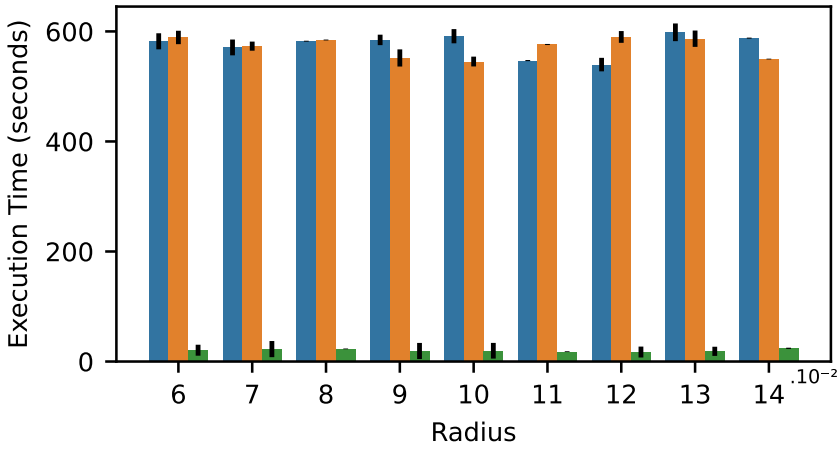
Execution time is the duration required to compute and assign paths for individual flows, accounting for the count of edges in each network scenario and the number of flows. Figure 5.14 shows the performance evaluation of the average algorithm execution time for the cities of Bern, Geneva, and Zurich over different radii. SWP and WSP are greedy solutions that always process all network nodes before calculating a route based on latency or throughput parameters, while FLATWISE is a heuristic that approximates the latency of the calculated path according to the latency specified for each flow. Thus, FLATWISE does not need to visit every node in the network to guarantee better latency, throughput, and FPS results in scenarios with latency and throughput guarantees simultaneously. The results show that FLATWISE needs less search time than WSP and SWP in all topologies. The larger the topology is, the greater is the processing time required by WSP and SWP algorithms. This situation can change depending on the number of users deployed on the network and the network's capacity to support these users. Therefore, FLATWISE achieves lower latency and higher throughput results than WSP and SWP and offers low processing costs in all the network scenarios analyzed.



(A) Average Algorithm Execution Time for the city of Bern.



(B) Average Algorithm Execution Time for the city of Geneva.



(C) Average Algorithm Execution Time for the city of Zurich.

FIGURE 5.14: Performance evaluation of average algorithm execution time for the cities of Bern, Geneva, and Zurich over different radii.

5.6 Chapter Conclusions

In this chapter, we investigated how to optimize the overall E2E latency for 6DoF VR applications. These applications demand stringent latency and throughput guarantees to deliver immersive experiences. We have investigated the JFA problem, which aims to find suitable paths for all flows in a network such that it determines the optimal paths in terms of throughput and latency to reduce the overall latency for all flows. We used Mixed Integer Linear Programming to model the JFA objective and constraints. We have shown that the JFA problem is $\mathcal{NP}$-hard, i.e., computationally expensive. Thus, it is impractical to implement an optimal solution to solve the JFA in real network environments. Therefore, we have proposed FLATWISE, a new heuristic that is one order of magnitude faster than the JFA problem.

FLATWISE is a novel intra-domain routing algorithm with throughput guarantees for minimizing the overall E2E latency performance for all 6DoF VR applications SDN infrastructures. FLATWISE provides an adaptive routing approach that can squeeze or relax the path calculation based on the E2E latency requirement of 6DoF VR applications. We addressed the ***Research Question 3.1*** by showing that FLATWISE primary characteristic is to approximate the E2E latency of the calculated path with the E2E latency required by each 6DoF VR application while it considers the impact of path assignment on other 6DoF VR applications. We provided algorithms for path allocation based on E2E latency awareness, and we also have shown the practical implementation of FLATWISE and compare its performance with SWP and WSP implementations.

We have assessed FLATWISE performance in a realistic simulated 5G network map of Bern, Geneva, and Zurich. Based on those topologies, we have modeled both network and computing latencies used in the FLATWISE simulation environment. We have implemented and compared the algorithms WSP and SWP against FLATWISE in a highly dynamic and realistic network environment where there is no flow prioritization while the network link availability changes over time. We have evaluated the performance of FLATWISE, WSP, and SWP by analyzing the VR in-game communication as a reference use case. We have considered the KPIs flow network latency, path latency, over-provisioned latency, E2E latency, flow network throughput, frame rate, video resolutions, and execution time. We have addressed the ***Research Question 3.2*** by showing the optimization of the overall network latency and throughput for all VR applications deployed on the network.

The importance of FLATWISE is highlighted through extensive simulations, showcasing its potential to reduce flow latency, over-provisioned latency, and E2E latency. It also enhances flow throughput, frame rate, and algorithm execution time compared to existing approaches. We addressed the ***Research Question 3.3*** by showing FLATWISE's ability to provide different routes for the same source-destination pairs based on E2E latency requirements, which demonstrates its adaptability to diverse network conditions. FLATWISE seamlessly integrates 6DoF VR applications into network infrastructures, ensuring low latency, high throughput, and a truly immersive user experience. As VR technologies continue to advance, solutions like FLATWISE will play a pivotal role in shaping the future of VR.

Chapter 6

Conclusions and Future Work

In this chapter, we first summarize the contributions of this thesis in Section 6.1, in which we provide an overview of the contributions described in Chapters 3, 4, and 5. Then, in Section 6.2, we briefly discuss future research directions towards latency minimization for future generation of VR systems running 6DoF content.

6.1 Summary of Contributions

In this thesis, we have investigated solutions to minimize the E2E latency for 6DoF VR applications. However, no viable solution covers all aspects of E2E latency for VR systems that demand ultra-low latencies and bandwidths over 1 Gbit/s. Thus, we considered various aspects of the E2E pipeline of VR systems, which can minimize latency for VR applications. Therefore, we mainly focused on optimization at the edge network and the network infrastructure, but always considering the limitations of current VR HMDs and the requirements of advanced VR applications running 6DoF content. We also focused on the impact on the E2E latency performance for VR systems in a highly dynamic network environment where over-provisioned resources, service migration, service offloading, and path allocation are performed.

First, our research investigated a resource provisioning mechanism. This mechanism was specifically designed to ensure the availability of resources while prioritizing them for the optimal delivery of real-time VR services deployed at the network edge. Second, as a crucial component of this research, we have developed an edge framework with the primary goal of efficiently orchestrating VR services. This framework incorporates advanced strategies, such as offloading and migration, carefully tailored to address the demands of DoF VR applications. The main objective is to reduce the E2E latency while maintaining the seamless functionality of these applications. Finally, we present a novel network routing strategy. This strategy represents a novel approach that aims to decrease the latency performance of 6DoF VR applications. To achieve such a goal, the new approach computes routing paths considering the unique E2E latency requirements of each 6DoF application. The contributions of this thesis are summarized below in the order in which they occurred.

6.1.1 Resource Provisioning Mechanism for VR Services

In Chapter 3, we investigated a new edge resource provisioning mechanism to address the research questions described in Section 1.3.1. We studied the problem of resource scarcity on resource-constrained MEC infrastructures and how to overcome it in the context of VR services deployment. To address this problem, we have proposed REACT, a resource provisioning mechanism that leverages resource provisioning among different services running on a shared edge environment. We have shown that REACT guarantees resource availability and prioritization for real-time VR services deployed on the network edge.

We addressed the ***Research Question 1.1*** by exploiting how REACT adopts an adaptive and solidarity-based auto-scaling strategy to redistribute resources from over-provisioned services (VR services) to under-provisioned services (not sensitive to latency) in edge environments. We highlighted that REACT is an alternative strategy to avoid service migration due to resource scarcity in MEC servers. We have shown that the core idea of the REACT is prioritizing resource provisioning for real-time VR applications. With such prioritization, REACT enhances the performance of high-priority VR services, especially when the edge infrastructure resources become scarce.

We addressed the ***Research Question 1.2*** by considering a resource-constrained MEC scenario to evaluate our approach against Kubernetes, a reactive algorithm baseline approach to provide resource provisioning in MEC servers. We optimized the resource provisioning in edge computing infrastructures by reducing the amount of over-provisioning resources. We also reduced the overall service outages whenever MEC resources become unavailable. We have demonstrated that REACT minimized the harmful effects of service migration while keeping more services running over the same MEC server.

Our evaluation assesses REACT's and Kubernetes's performance on a real testbed. Testbed results demonstrated the superior performance of REACT over Kubernetes in terms of accomplishing up to 18% more elasticity events, reducing service outages by up to 95%, reducing elasticity attempts by up to 95%, and reducing over-provisioned resources by up to 33%, 38%, and 73% for CPU cycles, RAM and bandwidth resources, respectively. Finally, REACT reduced response time by up to 15%. We have shown that REACT, compared to Kubernetes, provided several improvements. As a result, we enhanced the resource provisioning requests for *high-priority* VR services in edge infrastructures with resource scarcity situations, thus addressing the ***Research Question 1.3***.

We have provided evidence that our findings and contributions underscore REACT's substantial benefits in providing resource provisioning for VR services in MEC environments, making it a valuable contribution to edge-enabled VR deployment. Therefore, we demonstrated the potential for advancing the capabilities of edge networks in supporting latency-sensitive VR applications. By effectively managing resources and prioritizing the deployment of VR services, the research presented in Chapter 3 supports VR services deployed on edge networks while providing seamless, low latency, and responsive VR experiences for VR applications.

6.1.2 Edge Framework for VR Services Orchestration

In Chapter 4, we have investigated a new edge framework to orchestrate VR services, which aims to address the research questions described in Section 1.3.2. We studied the DSCP problem to find the optimal service placement of services from a service chain such that its E2E latency does not exceed 5 ms. We have shown that DSCP is $\mathcal{NP}$-hard, i.e., computationally expensive. Therefore, we propose a heuristic (TENET) that is one order of magnitude faster than DSCP. TENET is a novel edge framework to orchestrate VR services through offloading and migration strategies considering the requirements of 6DoF VR applications to minimize the overall E2E latency.

We implemented TENET as a SFC orchestrator, which supports offloading, migration, and orchestration of VR services deployed across HMDs and MECs to ensure acceptable E2E latency for VR applications. We addressed the ***Research Question 2.1*** by providing algorithms to split VR applications into indivisible services and deploy them across HMDs and MEC servers according to an optimization problem that jointly minimizes latency and energy consumption. We also provided algorithms for path calculation based on E2E latency and management of VR applications to ensure acceptable E2E latency.

We evaluated the performance of Meta HMD applications in terms of frame rate, computing latency, and power usage to model VR service workloads. We use those application metrics to model 6DoF VR service workloads in a simulated environment to evaluate system scalability, E2E latency, energy consumption, video resolution selection, context migrations, and execution time. We used a physical 5G network infrastructure map of the cities of Bern, Geneva, and Zurich. Based on those topologies, we modeled both network and computing latencies used in the TENET simulation environment. To address the ***Research Question 2.2***, in our evaluation, we provided the trade-off between E2E latency and energy consumption over three high-mobility scenarios compared to widely used service-migration strategies.

We have shown that for varying user densities in an urban scenario, TENET outperforms other widely adopted mechanisms regarding E2E latency in exchange for a moderate increment in power consumption. Moreover, we observed significant gains of TENET in selecting higher video resolutions for 6DoF VR applications based on E2E latency. TENET also provides more accepted context migrations than traditional service migration algorithms. Finally, we have shown that TENET can reduce the decision time on where to place the services while ensuring the performance of 5 ms. Therefore, we addressed the ***Research Question 2.3*** by showing that TENET deployment strategy impacts the E2E latency of VR applications and the selection of better video resolutions for latency-sensitive VR applications.

Through our findings and contributions, we have provided insightful directions on advancing VR service deployment over edge networks, considering the restrictions of the edge infrastructure, the resource limitations of HMDs, and the stringent latency requirements of 6DoF VR applications. Therefore, in the research presented in Chapter 4, we have shown that managing VR services deployed on edge networks brings benefits to reduce E2E latency, which has the potential to increase the QoS of 6DoF VR applications.

6.1.3 Adaptive Latency-aware Routing Mechanism for VR

In Chapter 5, we investigated a new latency-aware routing mechanism to improve the routing allocation for VR flows, which aims to address the research questions described in Section 1.3.3. We studied the Joint Flow Allocation (JFA) problem to find suitable paths for all flows in a network such that it determines the optimal paths in terms of throughput and latency to reduce the overall latency for all flows. We used Mixed Integer Linear Programming to model the JFA objective and constraints. We have shown that JFA problem is $\mathcal{NP}$-hard, i.e., computationally expensive. Therefore, we proposed a heuristic (FLATWISE) that is one order of magnitude faster than JFA.

FLATWISE is a new intra-domain routing strategy to support ultra-low latency requirements for 6DoF VR applications. FLATWISE provides throughput guarantees for minimizing the overall E2E latency performance for all 6DoF VR applications deployed on the network. Besides, we have shown that FLATWISE optimizes the latency performance for 6DoF VR applications by calculating paths based on their latency requirements. FLATWISE provides an adaptive routing approach that can squeeze or relax the path calculation based on the E2E latency requirement of 6DoF VR applications. We addressed the ***Research Question 3.1*** by showing that FLATWISE primary characteristic is to approximate the E2E latency of the calculated path with the E2E latency required by each 6DoF VR application by analyzing the impact of path assignment on other 6DoF VR applications.

We assessed FLATWISE performance in a realistic simulated 5G network map of Bern, Geneva, and Zurich. Based on those topologies, we modeled both network and computing latencies used in the FLATWISE simulation. We implemented and compared the algorithms WSP and SWP against FLATWISE in a highly dynamic and realistic network environment where there is no flow prioritization and the network link availability changes over time. We evaluated the performance of FLATWISE, WSP, and SWP by analyzing the VR in-game communication as a reference use case. We considered the KPIs network latency, path latency, over-provisioned latency, E2E latency, network throughput, frame rate, video resolutions, and execution time. We addressed the ***Research Question 3.2*** by showing the optimization of the overall network latency and throughput for VR applications deployed on the network.

Our findings and contributions offered important insights into improving current routing approaches to reduce the overall network latency for applications that demand stringent latency requirements and high throughput, such as 6DoF VR applications. We addressed the ***Research Question 3.3*** by showing FLATWISE's ability to provide different routes for the same source-destination pairs based on E2E latency requirements, which demonstrates its adaptability to diverse network conditions. Therefore, in the research resented in Chapter 5, we provided evidence that the design of a tailored routing approach for applications is more beneficial than always providing the shortest path (latency) or the widest path (throughput), which reduced the latency and improved the throughput for all flows on the network. As a result, FLATWISE seamlessly integrated 6DoF VR applications into network infrastructures, ensuring low latency, high throughput, and a truly immersive user experience.

6.2 Future Work

In this section, we outline potential avenues for future research and development based on the contributions made in this thesis. The primary focus of these contributions is to further enhance E2E latency reduction for 6DoF VR applications and address the challenges presented in the current research. Building upon these foundations, we propose several avenues for future work in this field that were elaborated based on the open issues of our contributions in this thesis. Below, we discuss the potential open questions for each chapter this thesis presents.

For chapter 3, an open issue is the development of *dynamic resource allocation* schemes based on machine learning or artificial intelligence techniques to predict resource demands and allocate resources in real time, considering the resource usage patterns of VR applications. Another open issue is the *restriction of the edge network environment*. We could consider extending the research in chapter 3 to consider hybrid cloud-edge environments, e.g., fog computing, edge computing, and cloud computing. Therefore, we could develop resource provisioning strategies that seamlessly span both cloud and edge resources, optimizing the E2E latency for VR applications that may utilize resources from both domains. Another open issue is the exploration of *resource allocation mechanisms based on energy efficiency*. Such mechanisms should not only reduce energy consumption but also consider latency reduction.

For chapter 4, an open issue is the *user mobility and handover*. We could investigate how our proposed edge orchestrator can adapt to user mobility and seamless handover between edge nodes in a more realistic environment, where the number of users can change according to certain criteria, such as more users connecting to the network at night. This is particularly relevant for VR applications used in scenarios with high user mobility. Another issue is using machine learning and artificial intelligence to *predict user mobility*. Through this prediction, VR services could be placed on MEC servers more efficiently, avoiding overloading and the forced migration of some services due to a scarcity of resources on some MEC servers.

For chapter 5, a potential issue is the *lack of inter-domain routing features*. We could extend the proposed routing mechanism to encompass inter-domain routing scenarios. We could also investigate how to optimize the routing of VR traffic between multiple domains while ensuring low-latency delivery. Another issue is the *consideration of QoE metrics during routing decisions*. We could consider incorporating QoE metrics into the routing strategy, as they are critical for VR applications. Besides, we could develop methods to adjust routes dynamically based on real-time user experience feedback. Another issue would be *addressing different traffic categories* in the experiments, not just VR traffic. This would make the scenario even more realistic and complex.

In conclusion, the contributions made in this thesis provide a solid foundation for further advancements in reducing E2E latency for 6DoF VR applications. Future research can build upon these contributions to create more robust, efficient, and user-friendly VR experiences in a variety of network environments.

Bibliography

[1] A. Medeiros, T. Braun, A. Di Maio, and A. Neto, "REACT: A Solidarity-based Elastic Service Resource Reallocation Strategy for Multi-access Edge Computing," *Physical Communication*, p. 101 380, 2021.

[2] A. Medeiros, A. Di Maio, T. Braun, and A. Neto, "Service Chaining Graph: Latency-and Energy-aware Mobile VR Deployment over MEC Infrastructures," in *Global Communications Conference*, IEEE, 2022, pp. 6133–6138.

[3] ——, "TENET: Adaptive Service Chain Orchestrator for MEC-enabled Low-latency 6DoF Virtual Reality," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, 2023.

[4] A. Medeiros, A. Di Maio, and T. Braun, "FLATWISE: Flow Latency and Throughput Aware Sensitive Routing for 6DoF VR over SDN," *IEEE Transactions on Network and Service Management*, 2023.

[5] G. S. for Mobile Communications (GSMA). "Cloud AR/VR whitepaper." (2019), [Online]. Available: https://www.gsma.com/futurenetworks/wiki/cloud-ar-vr-whitepaper.. (accessed: 26.07.2023).

[6] Nokia. "5G for Mission Critical Communication." (2016), [Online]. Available: http://www.hit.bme.hu/~jakab/edu/litr/5G/Nokia_5G_for_Mission_Critical_Communication_White_Paper.pdf..

[7] H. iLab. "Cloud VR Bearer Networks." (2017), [Online]. Available: https://www-file.huawei.com/-/media/corporate/pdf/ilab/cloud_vr_oriented_bearer_network_white_paper_en_v2.pdf. (accessed: 26.07.2023).

[8] Qualcomm. "Augmented and Virtual Reality: the First Wave of 5G Killer Apps." (2017), [Online]. Available: https://www.qualcomm.com/media/documents/files/augmented-and-virtual-reality-the-first-wave-of-5g-killer-apps.pdf.).

[9] AT&T. "Enabling mobile augmented and virtual reality with 5g networks." (2017), [Online]. Available: https://about.att.com/content/dam/innovationblogdocs/Enabling%20Mobile%20Augmented%20and%20Virtual%20Reality%20with%205G%20Networks.pdf.. (accessed: 26.02.2021).

[10] 5. P. A. W. Group, "View on 5G Architecture - Version 3.0," 5G PPP, Tech. Rep., Feb. 2020. DOI: 10.5281/zenodo.3265031.

[11] S. M. LaValle, *Virtual reality*. Cambridge university press, 2023.

[12] I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat, and H. Dai, "A survey on low latency towards 5g: Ran, core network and caching solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3098–3130, 2018.

[13] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[14] H. Abdah, J. P. Barraca, and R. L. Aguiar, "Qos-aware service continuity in the virtualized edge," *IEEE Access*, vol. 7, pp. 51 570–51 588, 2019.

[15] T. Chen, R. Bahsoon, and X. Yao, "A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–40, 2018.

[16] 5.-P. S. N. W. Group, "From webscale to telco, the cloud native journey," 5G PPP, Tech. Rep., Jul. 2018.

[17] S. Kekki, W. Featherstone, Y. Fang, *et al.*, "Mec in 5g networks," *ETSI white paper*, vol. 28, no. 2018, pp. 1–28, 2018.

[18] 5.-P. S. N. W. Group, "Cloud native and 5g verticals services," 5G PPP, Tech. Rep., Feb. 2020.

[19] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[20] A. Yousefpour, C. Fung, T. Nguyen, *et al.*, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, vol. 98, pp. 289–330, 2019.

[21] Q.-V. Pham, F. Fang, V. N. Ha, *et al.*, "A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art," *IEEE Access*, vol. 8, pp. 116 974–117 017, 2020.

[22] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–37, 2019.

[23] F. Zhang, G. Liu, X. Fu, and R. Yahyapour, "A survey on virtual machine migration: Challenges, techniques, and open issues," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1206–1243, 2018.

[24] Z. Rejiba, X. Masip-Bruin, and E. Marin-Tordera, "A survey on mobility-induced service migration in the fog, edge, and related computing paradigms," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–33, 2019.

[25] F. Spinelli and V. Mancuso, "Towards enabled industrial verticals in 5g: A survey on mec-based approaches to provisioning and flexibility," *IEEE Communications Surveys & Tutorials*, 2020.

[26] M. Kumar, S. Sharma, A. Goel, and S. Singh, "A comprehensive survey for scheduling techniques in cloud computing," *Journal of Network and Computer Applications*, 2019.

[27] F. S. D. Silva, M. O. Lemos, A. Medeiros, *et al.*, "Necos project: Towards lightweight slicing of cloud federated infrastructures," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, IEEE, 2018, pp. 406–414.

[28] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: State of the art and research challenges," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 430–447, 2018.

[29] 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.

[30] I. SG05, "Draft new report itu-r m.[imt-2020. tech perf req]-minimum requirements related to technical performance for imt-2020 radio interface (s)," *ITU-R SG05 Contribution*, vol. 40, 2017.

[31] Qualcomm. "Making Immersive Virtual Reality Possible in Mobile." (2016), [Online]. Available: https://www.qualcomm.com/media/documents/files/whitepaper-making-immersive-virtual-reality-possible-in-mobile.pdf.).

[32] W. Saad, M. Bennis, and M. Chen, "A vision of 6g wireless systems: Applications, trends, technologies, and open research problems," *IEEE network*, vol. 34, no. 3, pp. 134–142, 2019.

[33] C. Perfecto, M. S. Elbamby, J. Del Ser, and M. Bennis, "Taming the latency in multi-user vr 360°: A qoe-aware deep learning-aided multicast framework," *IEEE Transactions on Communications*, vol. 68, no. 4, pp. 2491–2508, 2020.

[34] F. Hu, Y. Deng, W. Saad, M. Bennis, and A. H. Aghvami, "Cellular-connected wireless virtual reality: Requirements, challenges, and solutions," *IEEE Communications Magazine*, vol. 58, no. 5, pp. 105–111, 2020.

[35] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylinattila, "A survey on mobile augmented reality with 5g mobile edge computing: Architectures, applications and technical aspects," *IEEE Communications Surveys & Tutorials*, 2021.

[36] G. Berardinelli, P. Baracca, R. O. Adeogun, *et al.*, "Extreme communication in 6g: Vision and challenges for 'in-x'subnetworks," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 2516–2535, 2021.

[37] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications surveys & tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[38] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[39] B. Halabi, S. Halabi, and D. McPherson, *Internet routing architectures*. Cisco press, 2000.

[40] J. T. Moy, *OSPF: anatomy of an Internet routing protocol*. Addison-Wesley Professional, 1998.

[41] C. Huitema, *Routing in the Internet*. Prentice-Hall, Inc., 1995.

[42] Meta. "Meta oculus 2." (2021), [Online]. Available: https://www.meta.com/ch/en/quest/products/quest-2/. (accessed: 11.27.2023).

[43] L. Kong, J. Tan, J. Huang, *et al.*, "Edge-computing-driven internet of things: A survey," *ACM Computing Surveys*, vol. 55, no. 8, pp. 1–41, 2022.

[44] L. Bréhon–Grataloup, R. Kacimi, and A.-L. Beylot, "Mobile edge computing for v2x architectures and applications: A survey," *Computer Networks*, vol. 206, p. 108 797, 2022.

[45] F. S. Abkenar, P. Ramezani, S. Iranmanesh, *et al.*, "A survey on mobility of edge computing networks in iot: State-of-the-art, architectures, and challenges," *IEEE Communications Surveys & Tutorials*, 2022.

[46] C. Feng, P. Han, X. Zhang, B. Yang, Y. Liu, and L. Guo, "Computation offloading in mobile edge computing networks: A survey," *Journal of Network and Computer Applications*, vol. 202, p. 103 366, 2022.

[47] S. Vlahovic, M. Suznjevic, and L. Skorin-Kapov, "A survey of challenges and methods for quality of experience assessment of interactive vr applications," *Journal on Multimodal User Interfaces*, vol. 16, no. 3, pp. 257–291, 2022.

[48] R. Uhlig, G. Neiger, D. Rodgers, *et al.*, "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48–56, 2005.

[49] J. Halpern and C. Pignataro, "Service function chaining (sfc) architecture," Tech. Rep., 2015.

[50] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, IEEE, 2017, pp. 405–410.

[51] G. C. Burdea and P. Coiffet, *Virtual reality technology*. John Wiley & Sons, 2003.

[52] E. Cuervo, K. Chintalapudi, and M. Kotaru, "Creating the perfect illusion: What will it take to create life-like virtual reality headsets?" In *Proceedings of the 19th international workshop on mobile computing systems & applications*, 2018, pp. 7–12.

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

[54] Z. Tan, Y. Li, Q. Li, Z. Zhang, Z. Li, and S. Lu, "Supporting mobile vr in lte networks: How close are we?" *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 1, pp. 1–31, 2018.

[55] M. Chen, W. Saad, and C. Yin, "Virtual reality over wireless networks: Quality-of-service model and learning-based resource management," *IEEE Transactions on Communications*, vol. 66, no. 11, pp. 5621–5635, 2018.

[56] M. Hu, X. Luo, J. Chen, Y. C. Lee, Y. Zhou, and D. Wu, "Virtual reality: A survey of enabling technologies and its applications in iot," *Journal of Network and Computer Applications*, vol. 178, p. 102 970, 2021.

[57] J. Chakareski, M. Khan, T. Ropitault, and S. Blandino, "6dof virtual reality dataset and performance evaluation of millimeter wave vs. free-space-optical indoor communications systems for lifelike mobile vr streaming," in *2020 54th Asilomar Conference on Signals, Systems, and Computers*, IEEE, 2020, pp. 1051–1058.

[58] S. Lee, J.-B. Jeong, and E.-S. Ryu, "Efficient group-based packing strategy for 6dof immersive video streaming," in *2022 International Conference on Information Networking (ICOIN)*, IEEE, 2022, pp. 310–314.

[59] M. Broxton, J. Flynn, R. Overbeck, *et al.*, "Immersive light field video with a layered mesh representation," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 86–1, 2020.

[60] J.-B. Jeong, S. Lee, and E.-S. Ryu, "Rethinking fatigue-aware 6dof video streaming: Focusing on mpeg immersive video," in *2022 International Conference on Information Networking (ICOIN)*, IEEE, 2022, pp. 304–309.

[61] J.-B. Jeong, S. Lee, I.-W. Ryu, T. T. Le, and E.-S. Ryu, "Towards viewport-dependent 6dof 360 video tiled streaming for virtual reality systems," in *Proceedings of the 28th ACM International Conference on Multimedia*, 2020, pp. 3687–3695.

[62] Y. Cai, X. Gao, W. Chen, and R. Wang, "Towards 6dof live video streaming system for immersive media," *Multimedia Tools and Applications*, vol. 81, no. 25, pp. 35 875–35 898, 2022.

[63] X. Hou and S. Dey, "Motion prediction and pre-rendering at the edge to enable ultra-low latency mobile 6dof experiences," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 1674–1690, 2020.

[64] W. (MPEG), "MPEG Strategic Standardisation Roadmap," International Organisation for Standardisation, Tech. Rep., Jun. 2016.

[65] B. Van Schewick, *Internet architecture and innovation*. Mit Press, 2012.

[66] T. Koponen, M. Chawla, B.-G. Chun, *et al.*, "A data-oriented (and beyond) network architecture," in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, 2007, pp. 181–192.

[67] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: An overview," 1994.

[68] A. Clemm, M. F. Zhani, and R. Boutaba, "Network management 2030: Operations and control of network 2030 services," *Journal of Network and Systems Management*, vol. 28, no. 4, pp. 721–750, 2020.

[69] C. Han, Y. Wu, Z. Chen, *et al.*, "Network 2030 a blueprint of technology, applications and market drivers towards the year 2030 and beyond," *International Telecommunication Union*, 2018.

[70] O. G. de Dios, R. Casellas, F. Cugini, and J. A. Hernandez, "Beyond 5g domainless network operation enabled by multiband: Toward optical continuum architectures," *arXiv preprint arXiv:2302.08244*, 2023.

[71] M. Ruiz, J. A. Hernández, M. Quagliotti, *et al.*, "Network traffic analysis under emerging beyond-5g scenarios for multi-band optical technology adoption," *Journal of Optical Communications and Networking*, vol. 15, no. 11, F36–F47, 2023.

[72] R. Li *et al.*, "Towards a new internet for the year 2030 and beyond," in *Proc. 3rd Annu. ITU IMT-2020/5G Workshop Demo Day*, 2018, pp. 1–21.

[73] S. Bryant, U. Chunduri, and A. Clemm, "Preferred Path Routing Framework," Internet Engineering Task Force, Internet-Draft draft-chunduri-rtgwg-preferred-path-routing-03,

Nov. 2022, Work in Progress, 25 pp. [Online]. Available: https://datatracker.ietf.org/doc/draft-chunduri-rtgwg-preferred-path-routing/03/.

[74] H. X. Wireless. "The future of mobile broadband." (2017), [Online]. Available: https://www-file.huawei.com/~/media/CORPORATE/PDF/mbb/huawei-mbb-report-final.pdf.. (accessed: 10.23.2023).

[75] M. Warburton, M. Mon-Williams, F. Mushtaq, and J. R. Morehead, "Measuring motion-to-photon latency for sensorimotor experiments with virtual reality systems," *Behavior Research Methods*, pp. 1–21, 2022.

[76] *Virtual reality profiles for streaming applications (3gpp ts 26.118 version 16.2.1 release 16)*, https://www.etsi.org/deliver/etsi_ts/126100_126199/126118/16.02.01_60/ts_126118v160201p.pdf.

[77] A. Hazarika and M. Rahmati, "Towards an evolved immersive experience: Exploring 5g-and beyond-enabled ultra-low-latency communications for augmented and virtual reality," *Sensors*, vol. 23, no. 7, p. 3682, 2023.

[78] I. Wohlgenannt, A. Simons, and S. Stieglitz, "Virtual reality," *Business & Information Systems Engineering*, vol. 62, pp. 455–461, 2020.

[79] G. Sayfan, *Mastering kubernetes*. Packt Publishing Ltd, 2017.

[80] B. Burns, J. Beda, K. Hightower, and L. Evenson, *Kubernetes: up and running*. " O'Reilly Media, Inc.", 2022.

[81] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *2012 IEEE Network Operations and Management Symposium*, IEEE, 2012, pp. 204–212.

[82] D. Balla, C. Simon, and M. Maliosz, "Adaptive scaling of kubernetes pods," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2020, pp. 1–5.

[83] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of grid computing*, vol. 12, no. 4, pp. 559–592, 2014.

[84] M. Ghobaei-Arani, A. Souri, and A. A. Rahmanian, "Resource management approaches in fog computing: A comprehensive review," *Journal of Grid Computing*, pp. 1–42, 2019.

[85] Q. Yuan, X. Ji, H. Tang, and W. You, "Toward latency-optimal placement and autoscaling of monitoring functions in mec," *IEEE Access*, vol. 8, pp. 41 649–41 658, 2020.

[86] F.-H. Tseng, M.-S. Tsai, C.-W. Tseng, Y.-T. Yang, C.-C. Liu, and L.-D. Chou, "A lightweight autoscaling mechanism for fog computing in industrial applications," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4529–4537, 2018.

[87] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, "Enorm: A framework for edge node resource management," *IEEE transactions on services computing*, 2017.

[88] R. da Rosa Righi, L. Andrioli, V. F. Rodrigues, C. A. da Costa, A. M. Alberti, and D. Singh, "Elastic-ran: An adaptable multi-level elasticity model for cloud radio access networks," *Computer Communications*, vol. 142, pp. 34–47, 2019.

[89] C. Li, H. Sun, Y. Chen, and Y. Luo, "Edge cloud resource expansion and shrinkage based on workload for minimizing the cost," *Future Generation Computer Systems*, vol. 101, pp. 327–340, 2019.

[90] A.-F. Antonescu and T. Braun, "Simulation of sla-based vm-scaling algorithms for cloud-distributed applications," *Future Generation Computer Systems*, vol. 54, pp. 260–273, 2016.

[91] R. K. Naha, S. Garg, A. Chan, and S. K. Battula, "Deadline-based dynamic resource allocation and provisioning algorithms in fog-cloud environment," *Future Generation Computer Systems*, vol. 104, pp. 131–141, 2020.

[92] C. Li, C. Wang, and Y. Luo, "An efficient scheduling optimization strategy for improving consistency maintenance in edge cloud environment," *The Journal of Supercomputing*, pp. 1–28, 2020.

[93] G. Castellano, F. Esposito, and F. Risso, "A distributed orchestration algorithm for edge computing resources with guarantees," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, IEEE, 2019, pp. 2548–2556.

[94] A. G. Tasiopoulos, O. Ascigil, I. Psaras, and G. Pavlou, "Edge-map: Auction markets for edge resource provisioning," in *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, IEEE, 2018, pp. 14–22.

[95] J. Guo, C. Li, Y. Chen, and Y. Luo, "On-demand resource provision based on load estimation and service expenditure in edge cloud environment," *Journal of Network and Computer Applications*, vol. 151, p. 102 506, 2020.

[96] I. Sarrigiannis, K. Ramantas, E. Kartsakli, P.-V. Mekikis, A. Antonopoulos, and C. Verikoukis, "Online vnf lifecycle management in an mec-enabled 5g iot architecture," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4183–4194, 2019.

[97] N. Akhtar, I. Matta, A. Raza, L. Goratti, T. Braun, and F. Esposito, "Managing chains of application functions over multi-technology edge networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 511–525, 2021.

[98] J. Son and R. Buyya, "Latency-aware virtualized network function provisioning for distributed edge clouds," *Journal of Systems and Software*, vol. 152, pp. 24–31, 2019.

[99] H. Alipour, Y. Liu, and A. Hamou-Lhadj, "Analyzing auto-scaling issues in cloud environments," in *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '14, Markham, Ontario, Canada: IBM Corp., 2014, pp. 75–89.

[100] Y. Pan, C. Wang, Y. Liu, C. Xu, Y. Liu, and L. Zhang, "5g mobile edge assisted metaverse light field video system: Prototype design and empirical evaluation," *Available at SSRN 4106315*, 2022.

[101] C. Wang, S. Zhang, Z. Qian, *et al.*, "Joint server assignment and resource management for edge-based mar system," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2378–2391, 2020.

[102] D. Alencar, C. Both, R. Antunes, H. Oliveira, E. Cerqueira, and D. Rosário, "Dynamic microservice allocation for virtual reality distribution with qoe support," *IEEE Transactions on Network and Service Management*, 2021.

[103] H. Santos, D. Rosario, E. Cerqueira, and T. Braun, "Multi-criteria service function chaining orchestration for multi-user virtual reality services," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*, IEEE, 2022, pp. 6360–6365.

[104] S. Mangiante, G. Klas, A. Navon, Z. GuanHua, J. Ran, and M. D. Silva, "VR is on the edge: How to deliver 360 videos in mobile networks," in *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, 2017, pp. 30–35.

[105] J. Ruan and D. Xie, "Networked vr: State of the art, solutions, and challenges," *Electronics*, vol. 10, no. 2, p. 166, 2021.

[106] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, N. Dai, and H.-S. Lee, "Furion: Engineering high-quality immersive virtual reality on today's mobile devices," *IEEE Transactions on Mobile Computing*, vol. 19, no. 7, pp. 1586–1602, 2019.

[107] A. Younis, B. Qiu, and D. Pompili, "Latency-aware hybrid edge cloud framework for mobile augmented reality applications," in *2020 17th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, IEEE, 2020, pp. 1–9.

[108] Y. Liu, J. Liu, A. Argyriou, and S. Ci, "Mec-assisted panoramic vr video streaming over millimeter wave mobile networks," *IEEE Transactions on Multimedia*, vol. 21, no. 5, pp. 1302–1316, 2018.

[109] C. Zheng, S. Liu, Y. Huang, and L. Yang, "Mec-enabled wireless vr video service: A learning-based mixed strategy for energy-latency tradeoff," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, IEEE, 2020, pp. 1–6.

[110] J. Santos, J. van der Hooft, M. T. Vega, T. Wauters, B. Volckaert, and F. De Turck, "Efficient orchestration of service chains in fog computing for immersive media," in *2021 17th International Conference on network and service management (CNSM)*, IEEE, 2021, pp. 139–145.

[111] T. V. Doan, G. T. Nguyen, M. Reisslein, and F. H. Fitzek, "Sap: Subchain-aware nfv service placement in mobile edge cloud," *IEEE Transactions on Network and Service Management*, vol. 20, no. 1, pp. 319–341, 2022.

[112] P. Mandal, "Comparison of placement variants of virtual network functions from availability and reliability perspective," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 860–874, 2022.

[113] D. Zheng, G. Shen, X. Cao, and B. Mukherjee, "Towards optimal parallelism-aware service chaining and embedding," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2063–2077, 2022.

[114] R. Mohammadi, S. Akleylek, A. Ghaffari, and A. Shirmarz, "Taxonomy of traffic engineering mechanisms in software-defined networks: A survey," *Telecommunication Systems*, vol. 81, no. 3, pp. 475–502, 2022.

[115] R. Venkatasai, U. Prabu, and S. Ch, "A survey and analysis of qos-based routing techniques in software-defined networks," in *2023 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*, IEEE, 2023, pp. 1459–1464.

[116] D. Wu, Z. Yang, P. Zhang, R. Wang, B. Yang, and X. Ma, "Virtual-reality inter-promotion technology for metaverse: A survey," *IEEE Internet of Things Journal*, 2023.

[117] G. S. Let, C. Pratap, D. Jagannath, D. Dolly, and L. D. Evangeline, "Software-defined networking routing algorithms: Issues, qos and models," *Wireless Personal Communications*, pp. 1–31, 2023.

[118] S. H. A. Kazmi, F. Qamar, R. Hassan, K. Nisar, and B. S. Chowdhry, "Survey on joint paradigm of 5g and sdn emerging mobile technologies: Architecture, security, challenges and research directions," *Wireless Personal Communications*, pp. 1–48, 2023.

[119] G. Y. Handler and I. Zang, "A dual algorithm for the constrained shortest path problem," *Networks*, vol. 10, no. 4, pp. 293–309, 1980.

[120] Z. Wang and J. Crowcroft, "Bandwidth-delay based routing algorithms," in *Proceedings of GLOBECOM'95*, IEEE, vol. 3, 1995, pp. 2129–2133.

[121] ——, "Quality-of-service routing for supporting multimedia applications," *IEEE Journal on selected areas in communications*, vol. 14, no. 7, pp. 1228–1234, 1996.

[122] Y. Yang, J. K. Muppala, and S. T. Chanson, "Quality of service routing algorithms for bandwidth-delay constrained applications," in *Proceedings Ninth International Conference on Network Protocols. ICNP 2001*, IEEE, 2001, pp. 62–70.

[123] M. N. Soorki and H. Rostami, "Label switched protocol routing with guaranteed bandwidth and end to end path delay in mpls networks," *Journal of Network and Computer Applications*, vol. 42, pp. 21–38, 2014.

[124] S. Tomovic and I. Radusinovic, "Fast and efficient bandwidth-delay constrained routing algorithm for sdn networks," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, IEEE, 2016, pp. 303–311.

[125] H. Li, A. Osmani, and A. S. A. Aziz, "A fuzzy-based fast routing algorithm with guaranteed latency-throughput over software defined networks," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 10, pp. 8221–8233, 2022.

[126] C. Wu, Y. Zhang, N. Li, and A. Rezaeipanah, "An intelligent fuzzy-based routing algorithm for video conferencing service provisioning in software defined networking," *Telecommunication Systems*, pp. 1–12, 2023.

[127] J. Gong and A. Rezaeipanah, "A fuzzy delay-bandwidth guaranteed routing algorithm for video conferencing services over sdn networks," *Multimedia Tools and Applications*, pp. 1–30, 2023.

[128] J. Cheng, X. Zhu, and S. Abedi, "A fuzzy based routing approach for improving online conferencing services in software defined networking," *Cybernetics and Systems*, pp. 1–23, 2023.

[129] L. Zhao, Z. Yin, K. Yu, *et al.*, "A fuzzy logic-based intelligent multiattribute routing scheme for two-layered sdvns," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4189–4200, 2022.

[130] A. Alidadi, M. Mahdavi, and M. Hashmi, "A new low-complexity qos routing algorithm for mpls traffic engineering," in *2009 IEEE 9th Malaysia International Conference on Communications (MICC)*, IEEE, 2009, pp. 205–210.

[131] H. Yang, W. Liu, J. Li, and T. Q. Quek, "Space information network with joint virtual network function deployment and flow routing strategy with qos constraints," *IEEE Journal on Selected Areas in Communications*, 2023.

[132] A. Alidadi, S. Arab, and T. Askari, "A novel optimized routing algorithm for qos traffic engineering in sdn-based mobile networks," *ICT Express*, vol. 8, no. 1, pp. 130–134, 2022.

[133] P. Kamboj, S. Pal, S. Bera, and S. Misra, "Qos-aware multipath routing in software-defined networks," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 2, pp. 723–732, 2022.

[134] Y. Wang, R. Kang, L. Guo, S. Yang, J. Zhou, and C. Zhang, "Optimal flow and capacity allocation in multiple joint quickest paths of directed networks," *Computers & Operations Research*, vol. 150, p. 106 053, 2023.

[135] A. Ali, S. Tariq, M. Iqbal, *et al.*, "Adaptive bitrate video transmission over cognitive radio networks using cross layer routing approach," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 3, pp. 935–945, 2020.

[136] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[137] A. Medeiros, *REACT Prototype*, 2020. [Online]. Available: https://bitbucket.org/alissonpmedeiros/elasticity/src/master/.

[138] S. Confederation. "Maps of switzerland." (2022), [Online]. Available: https://map.geo.admin.ch/.. (accessed: 10.08.2023).

[139] A. Medeiros, *Service Chaining Graph Release V1*, version v1, Aug. 2022. DOI: 10.5281/zenodo.7004077. [Online]. Available: https://doi.org/10.5281/zenodo.7004077.

[140] D. Medhi and K. Ramasamy, *Network routing: algorithms, protocols, and architectures*. Morgan kaufmann, 2017.

[141] E. DIJKSTRA, "A note on two problems in connexion with graphs.," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959. [Online]. Available: http://eudml.org/doc/131436.

[142] R. R. Fontes, S. Afzal, S. H. Brito, M. A. Santos, and C. E. Rothenberg, "Mininet-wifi: Emulating software-defined wireless networks," in *2015 11th International Conference on Network and Service Management (CNSM)*, IEEE, 2015, pp. 384–389.