TENET: Adaptive Service Chain Orchestrator for MEC-enabled Low-latency 6DoF Virtual Reality

Alisson Medeiros[1], Antonio Di Maio[1], Torsten Braun[1], Augusto Neto[2,3]
[1] Institute of Computer Science, University of Bern, Bern, Switzerland
[2] Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte, Brazil
{alisson.medeiros, antonio.dimaio, torsten.braun}@unibe.ch, augusto@dimap.ufrn.br

***Abstract*—The next generation of Virtual Reality (VR) applications is expected to provide advanced experiences through Six Degrees of Freedom (6DoF) content, which requires higher data rates and ultra-low latency. In this article, we refactor 6DoF VR applications into atomic services to increase the computing capacity of VR systems aiming to reduce the end-to-end (E2E) of 6DoF VR applications. Those services are chained and deployed across Head-Mounted Displays (HMDs) and Multi-access Edge Computing (MEC) servers in high mobility scenarios over real-edge network topologies. We 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. The DSCP problem is $\mathcal{NP}$-hard. We provide an integer linear program to model the system, along with a heuristic, namely disTributed sErvice chaiN orchEstraTor (TENET), which is one order of magnitude faster than optimally solving the DSCP problem. 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 observe a significant reduction of E2E latency and gains in more advanced video resolution selection and accepted context service migrations when using TENET's deployment strategy on VR services.**

***Index Terms*—Mobile Virtual Reality, End-to-end Latency, Six Degrees of Freedom Videos, Multi-access Edge Computing, Service Function Chaining, Service Offloading, Service Migration and Quality of Service.**

I. Introduction

Virtual Reality (VR) systems artificially render a virtual environment with cognitive and sensorimotor characteristics, providing an advanced immersive reality through Six Degrees of Freedom (6DoF) videos to support both body and head motion, where the viewing direction and position can change [1]. Although VR systems have attracted considerable attention in recent years, it is infeasible to meet the requirements to support 6DoF videos by processing 6DoF content on Head-Mounted Displays (HMDs) [2], [3]. 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 massive adoption of 6DoF VR depends on the processing capability of HMDs to support unprecedented low latency and ultra-high throughput requirements.

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 end-to-end (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 [4], [5]. 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 [6]. Even considering that extreme communication requirements, e.g., latency and throughput, will be achieved by 6G networks, the constrained computation and energy impose restrictions on processing 6DoF content on VR HMDs [7].

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 to reduce energy consumption. However, this strategy significantly restricts VR technology's application domain by limiting the user's mobility range, particularly for *tethered* HMDs. Introducing wireless communications in VR systems dramatically extends the applications of VR for mobile users, e.g., VR Automotive Video Streaming (AVS), as it unleashes VR's true potential by enabling Mobile VR (MVR) to provide user experience from anywhere at any time [8], [9]. However, wireless VR also raises several technical challenges to supporting Mobile Virtual Reality (MVR) applications [10]. For example, wireless (*standalone*) HMDs must rely on a constrained onboard computing capability and limited energy supply for their operation merely by HMD processing [11]. Consequently, 6DoF VR content is most likely restricted to *edge streaming* scenarios due to its high computing power demands [12].

Since it is impractical to use specialized hardware platforms to support VR use cases with high mobility features, e.g., VR-AVS, due to the limited processing capacity and battery constraints of HMDs, Multi-access Edge Computing (MEC) arises to support VR technical limitations by deploying computing and service delivery at the network edge to process VR-intensive computing services [13], [14]. However, coordinating such a plethora of VR services, especially during user mobility, yields several challenges. How to distribute VR services, e.g., decoders and mobility tracking, across the MEC infrastructure to reduce E2E latency of VR applications? 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? How does E2E latency reduction affect the selection of video resolutions for VR systems?

To address the challenges mentioned above, we propose a disTributed 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. Besides, TENET is developed according to an optimization problem that jointly minimizes latency and energy consumption. TENET also optimizes the selection of better video resolutions for VR systems. TENET is an extension of our previous work, which addresses the trade-off between E2E latency and power consumption [15]. However, TENET extends our previous work [15] by considering new algorithms, architecture, and Key Performance Indicators (KPIs). Our contributions are as follows.

- We define the *Distributed Service Chain Problem* (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 (Section III).
- 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 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 (Section IV).
- We use a physical 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 TENET simulation environment (Section V-A).
- We evaluate the performance of Meta HMD[1] applications in terms of frame rate, computing latency, and power usage to model service workloads (Section V-B).
- 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 orchestration algorithm with traditional approaches that provide service migration 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 independent VR services over MEC infrastructures.

II. Related Works

Previous studies [16], [17], [18], [19] have shown that edge computing enables advanced VR Six Degrees of Freedom (6DoF) experiences by supporting the deployment of compute-intensive services. Chakareski et al. [16] investigate edge-based 6DoF VR streaming over millimeter-Wave to offer high available spectrum and data rates for VR HMDs. Hou et al. [17] consider motion prediction and pre-rendering services at the edge network to enable low latency 6DoF VR. Pan et al. [18] propose an edge-assisted metaverse algorithm to reduce the computational latency of 6DoF videos. Jeong et al. [19] propose a viewport-dependent high-efficiency video coding-compliant tiled streaming for immersive 6DoF videos. 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 [20], [21], [22], [23] study the behavior of the E2E latency and other Quality of Service (QoS) metrics of VR applications when their services are deployed on the MEC infrastructure. Wang et al. [20] investigate offloading of Mobile Augmented Reality (MAR) services to edge networks, where each service comprises a chain of dependent services, i.e., services that require inputs from other services. Our previous work [21] proposes a solidarity resource allocation approach to ensure the deployment of high-priority services in MEC servers. Alencar et al. [22] investigate dynamic microservice allocation in 5G networks to optimize QoS based on latency. Santos et al. [23] propose a constrained-based heuristic to minimize the delay of VR services deployed over edge networks while meeting resource requirements. 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.

Likewise, some other works [24], [25], [26], [27] study how different policies for distributing services between mobile devices and MEC infrastructure impact the VR applications' QoS metrics, such as latency. Authors in [24] 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. [25] investigate the feasibility of enabling high-quality VR smartphone applications by employing a framework running on both the smartphone and the edge server. Younis et al. [26] propose a framework to minimize network latency by optimizing service placement through computation-offloading decisions on MEC infrastructure. Akhtar et al. [27] investigate the chain management of application functions over multi-technology edge infrastructure to provide higher data rates and ultra-low latency for VR applications. 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 [28], [29], [30], [31], [32], [33] study either latency reduction or energy consumption optimization in edge networks. Liu et al. [28] propose deploying VR services on MECs. They provide the trade-off among link adaptation, transcoding-based chunk quality adaptation, and viewport rendering offloading. Zheng et al. [29] investigate the

[1]https://www.meta.com/ch/en/quest/products/quest-2/

TABLE I. Comparison of TENET algorithm to related works. 1 = Offloading, 2 = Migration, 3 = E2E latency, 4 = Power consumption, 5 = MEC-supported, 6 = HMD-supported, SFC = 7.

Solutions	Characteristic						
(References)	1	2	3	4	5	6	7
Chakareski et al. [16]					✓	✓	
Pan et al. [18]					✓		
Jeong et al. [19]	✓					✓	
Wang et al. [20]	✓				✓		✓
Medeiros et al. [21]					✓		
Alencar et al. [22]		✓					✓
Santos et al. [23]	✓			✓			✓
Ruan et al. [24]	✓			✓			
Lai et al. [25]	✓			✓			
Younis et al. [26]	✓		✓		✓		
Akhtar et al. [27]				✓			✓
Liu et al. [28]	✓		✓	✓			
Zheng et al. [29]		✓	✓	✓			
Santos et al. [30]	✓	✓	✓	✓			✓
Doan et al. [31]							✓
Mandal et al. [32]	✓	✓					
Zheng et al. [33]							✓
TENET (present work)	✓	✓	✓	✓	✓	✓	✓

scenario of multi-tiles-based wireless VR video service with the aid of MEC, where the primary objective is to analyze the trade-off between energy consumption and latency. Santos et al. [30] propose the orchestration of VR services in fog-cloud infrastructures. The evaluation of realistic VR container-based service chains shows that deploying VR components hosted in a fog-cloud infrastructure can satisfy the 20 ms latency boundary. Doan et al. [31] formulate a novel subchain-aware service placement optimization model that accounts for the configuration cost for stitching together reused network functions to a Service Function Chaining (SFC) and strives to reuse existing subchains of consecutive network functions while accounting for the recovery cost of network functions with limited reliability. Mandal et al. [32] analyze the network service availability considering deploying network services using multiple host nodes, single host nodes, and mixed-mode. Besides, authors compare the availability and reliability of network services considering those placement strategies. Zheng et al. [33] introduce a novel augmented graph to address the parallel relationship constraint among SFCs. Besides, authors propose a novel problem called parallelism-aware SFC and embedding. 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. 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.

Table I compares the main characteristics of the related works concerning service offloading, service migration, E2E latency, power consumption, MEC-supported, HMD-supported, and SFC. Table I shows that none of the considered solutions can support all our claimed requirements towards E2E latency reduction. Motivated by the limitations of the approaches presented in this section, we propose TENET, as described in Sections III and IV.

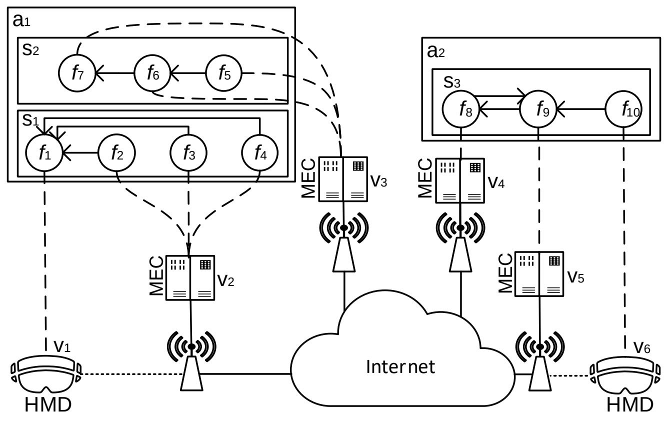


Fig. 1. Example of service chain graph deployment on the network. The solid lines indicate wired connectivity, the dotted lines indicate wireless connectivity, and the dashed lines represent the connectivity between HMDs and MEC servers hosting the offloaded VR services.

III. System Model and Problem Formulation

A. 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. Table II presents the symbols used in the system model and problem formulation sections.

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 Central Processing Unit (CPU) cycles per second $C_i \in \mathbb{R}$ and the maximum Graphics Processing Unit (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

TABLE II. Table of symbols

Symbol	Description
$\mathbb{G}$	Graph representing the network infrastructure
$\mathbb{V}$	Set of computing devices (MEC servers and HMDs)
$\mathbb{E}$	Set of paths between computing devices
$\mathbb{H}$	Set of HMDs
B_j	Maximum achievable data throughput on path e_j
C_i	Maximum CPU cycles per second of device v_i
G_i	Maximum GPU cycles per second of device v_i
$\mathbb{F}$	Set of all services
$\mathbb{F}_i$	Set of services deployed on device v_i
R_m	CPU and GPU cycles required for service f_m
A_m	CPU and GPU cycles allocated to service f_m
ω_n	Maximum data throughput between services in chain s_n
$\mathbb{S}$	Set of all service chains
b_n	Allocation resource vector of service chain s_n
W	Set of all maximum data throughputs
W^*	Set of all possible data throughput sets
Θ_n	VR application QoS for application a_n
ϵ_m	Power required to run service f_m
Ψ	Average system-wide power consumption per HMD
p_m	Computational latency of service f_m
P_i	Computational latency of service chain s_i
P_n^*	Maximum computational latency of application a_n
k_m	Latency to transmit data from service f_m to the next service
K_i	Network latency of service chain s_i
K_n^*	Maximum network latency of application a_n
L_n	Total end-to-end latency of application a_n
L	Average system-wide end-to-end latency
α	Power sensitivity coefficient
U	Cost function to be minimized
φ_n	Upper bound on end-to-end latency for application a_n
Δ_n	QoS accuracy constraint for application a_n

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

$$R_m = (R_m^c, R_m^g) \in \mathbb{R}^2 \tag{1}$$

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

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

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

$$A_m \geq R_m \tag{3}$$

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 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 services 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 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 1 shows an example containing 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 chain $s_3 = (f_8, f_9, f_{10})$. 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

$$\Theta_n = \sigma_n - \varrho_n \in \mathbb{R}, \forall a \in \{1, \ldots, |\mathbb{A}|\} \tag{4}$$

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.,

$$\Psi = \frac{1}{|\mathbb{A}|} \sum_{\{i:v_i \in \mathbb{H}\}} \sum_{\{m:f_m \in \mathbb{F}_i\}} \epsilon_m \tag{5}$$

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.,

$$P_i = \sum_{\{m:f_m \in s_i\}} p_m \tag{6}$$

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.,

$$P_n^* = \max_{\{i:s_i \in \mathbb{S}_n\}} P_i \tag{7}$$

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 consecutive services along the chain, i.e.,

$$K_i = \sum_{\{m:f_m \in s_i\}} k_m \tag{8}$$

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.,

$$K_n^* = \max_{\{i:s_i \in \mathbb{S}_n\}} K_i \tag{9}$$

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.,

$$L_n = K_n^* + P_n^*, \forall a \in \{1, \dots, |\mathbb{A}|\} \tag{10}$$

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

$$L = \frac{1}{|\mathbb{A}|} \sum_{n=1}^{|\mathbb{A}|} L_n \tag{11}$$

B. 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 $v_i \in \mathbb{V}$. Thus, the algorithm complexity depends on the number of combinations specified in the optimization problem. 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. Separately optimizing latency and energy may lead to different solutions to the optimization problem. 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

$$U = \alpha L + (1 - \alpha)\Psi \tag{12}$$

to be minimized by exploring the set of all possible allocation resource vectors, subject to a set of network operation constraints listed in Optimization Problem 13.

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

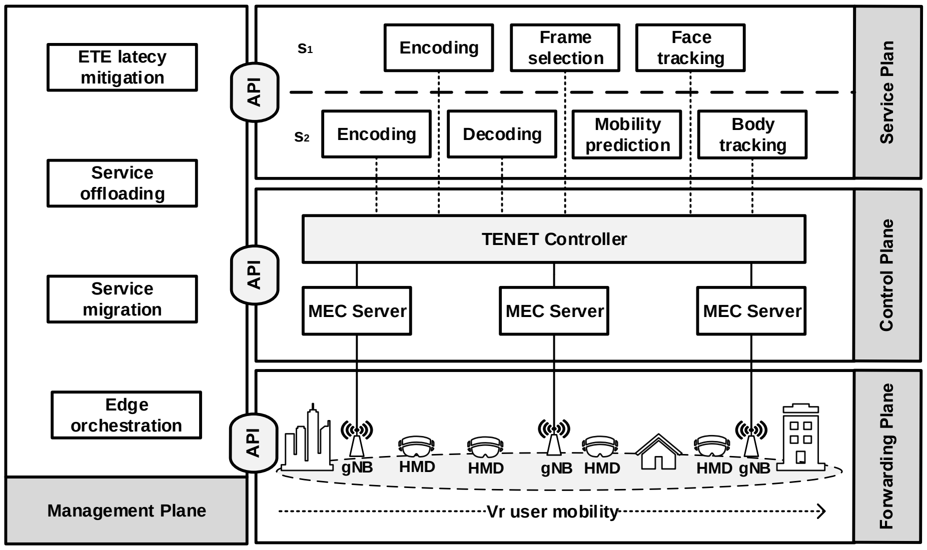


Fig. 2. TENET architecture.

$$
\begin{aligned}
&\underset{B \in B^*}{\text{minimize}} \quad U = \alpha L + (1-\alpha)\Psi && && \text{(13a)}\\
&\text{subject to} && &&\\
&L_n \leq \varphi_n && \forall n \in \{1, \ldots, |\mathbb{A}|\}, && \text{(13b)}\\
&\Theta_n \geq \Delta_n \cdot \sigma_n && \forall n \in \{1, \ldots, |\mathbb{A}|\}, && \text{(13c)}\\
&\sum_{n=1}^{|\mathbb{S}|} \omega_n \cdot c_n(e_j) \leq B_j && \forall j \in \{1, \ldots, |\mathbb{E}|\}, && \text{(13d)}\\
&\sum_{\{m: f_m \in \mathbb{F}_i\}} A_m^c \leq C_i && \forall i \in \{1, \ldots, |\mathbb{V}|\}, && \text{(13e)}\\
&\sum_{\{m: f_m \in \mathbb{F}_i\}} A_m^g \leq G_i && \forall i \in \{1, \ldots, |\mathbb{V}|\} && \text{(13f)}
\end{aligned}
$$

each application (constraint 13b). 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 13c). 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 13d). For each MEC server v_i, the sum of the CPU resources A_m^c and GPU resources A_m^g 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 13e and 13f).

IV. 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.

A. Architecture

To achieve the visions of TENET, we developed an architecture to be deployed in the MEC servers and VR HMDs. Figure 2 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 Software-Defined Networking (SDN) controller to manage the network resources to ensure the acceptable latency for MVR applications. In the following, we describe the TENET architecture in more detail.

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. The TENET controller also provides the trade-off analysis between the offloading and the migration, which considers the latency aspects of both procedures.
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. The more computationally intensive a service is, the higher is its prioritization.

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* [21]. *REACT* is a solidarity-based elastic service resource allocation strategy for service deployment over MEC servers with service prioritization.

B. 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. 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.

C. Managing Dependencies among SFCs

Section III points out that an application may have different service chains. Each service chain follows specific criteria to maintain the acceptable E2E latency for each application. 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. For instance, a content aggregator service depends on decoded video parts from other decoder services before sending the VR video to HMD. This type of dependency can be a bottleneck for the entire VR application, as any failure can cause a higher delay in a given service chain. 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*.

Algorithm 1 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 $\alpha$                              ▷ $0 \leq \alpha \leq 1$
 5:     if not $\alpha$ 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 2 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[$v$] $\leftarrow \varnothing$
    ▷ init dist[$s$] with $s$' computing latency
 3: dist [$s$] $\leftarrow s_p$
    ▷ split $\mathbb{G}$ into zones $Z$
 4: $Z \leftarrow \mathbb{G}$
    ▷ search in $\mathbb{Z}$ where $a_n$ is connected
 5: while $Z \neq \varnothing$ do
 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
```

Algorithm 3 SCG management for VR applications

```
    Input: $h_i$, $\alpha$, $\mathbb{F}$
 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($\rho_n$)
 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($v_i$, $h_i$, $s_n$, $f_m$)
```

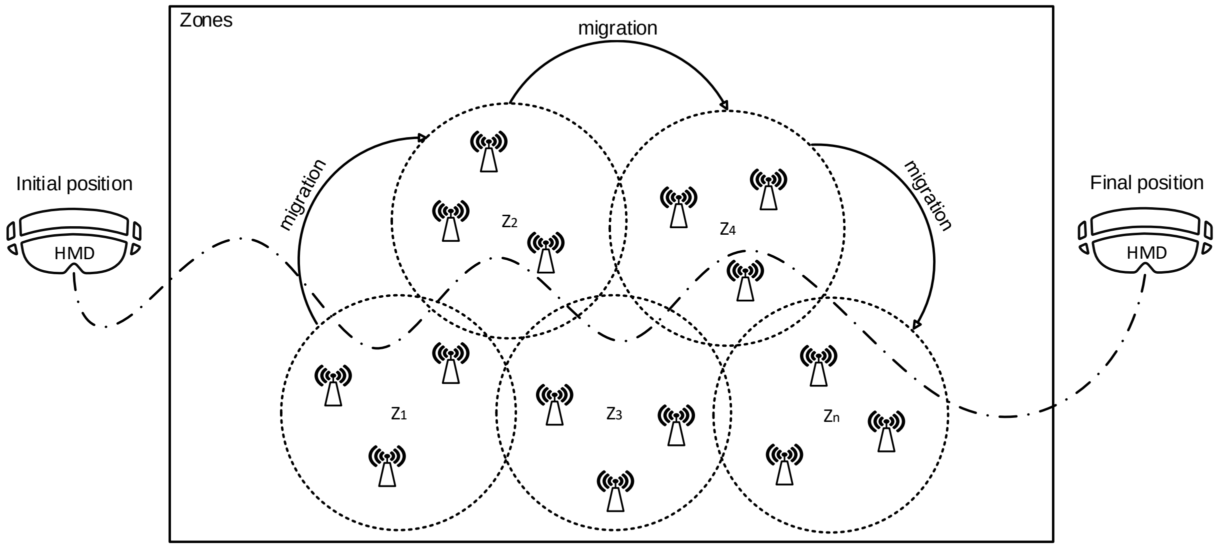


Fig. 3. TENET's zones scheme.

D. Latency and Energy Trade-off Procedure

Algorithm 1 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 1 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).

E. Path Calculation based on E2E Latency

Algorithm 2 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 2 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 2 only considers the 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 2 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. Figure 3 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 adjacent 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 (line 10).

F. Ensuring Acceptable E2E Latency for Mobile VR Services

Algorithm 3 describes the practical implementation of TENET. We shuffle the order in which services are processed in each iteration of Algorithm 3 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, 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 2 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 3 is equivalent to that of the Dijkstra's Algorithm, namely $\mathcal{O}((\mathbb{V}+\mathbb{E})log(\mathbb{V})) = \mathcal{O}(\mathbb{E}log(\mathbb{V}))$.

V. Performance Evaluation

A. Experiment Setup

We compare the latency and energy performance of our proposed method, TENET, in a simulated environment against a set of state-of-the-art methods that tackle the same problem.

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 4.80 Gbit/s 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 applications, we model the workloads for each VR application.

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 Frames per Second (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 1.20 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, 1.01 kJ, 1.27 kJ, and 2.57 kJ 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.

3) VR Users Mobility: We use Mininet-WiFi to simulate a realistic network scenario and user mobility. We use ONOS[2] 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.

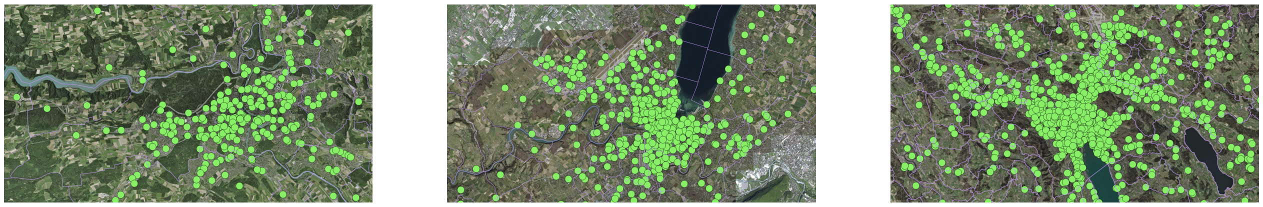
The VR users' mobility follows the Random Direction Model, in which users move along a straight line with a constant speed selected from a uniform distribution with a 100 mm/s average. We assume that mobile VR users connect to the base station whose signal is received with the highest Signal-to-Noise 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^c_m) and GPU (i.e., R^g_m) are randomly extracted from two uniform distributions with averages of 1.77 GHz for the CPU and 440 MHz for the GPU, based on the typical requirements of Meta HMD applications.

4) Edge Network Graphs: We use real 5G edge network topologies for three cities, Bern (BE), Geneva (GE), and Zurich (ZH) [34] in our simulation environment. The original 5G network infrastructures are shown in Figure 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 51.60 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, 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 500 µs and 1 ms. In each city, the base stations are located at positions illustrated in Figure 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 6 shows the generated topologies with network and computing latencies from Bern 5G network topology. We show only a random selection of 8% of total edges to improve visualization quality.

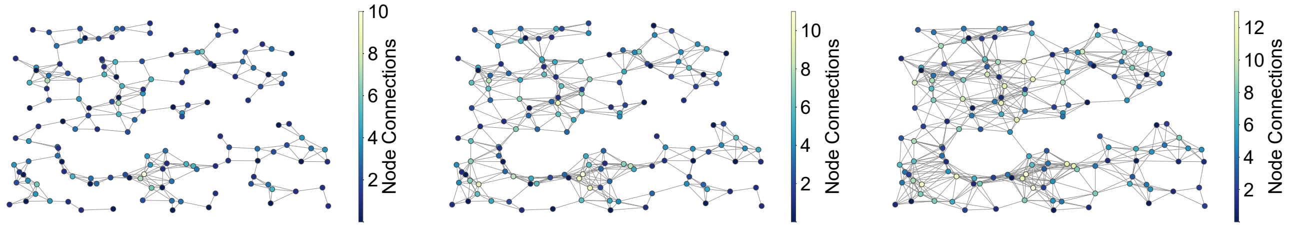
5) Performance Metrics: The performance of Meta HMD is evaluated by executing it in the simulated scenario for 10 hours and measuring the average E2E latency and power

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

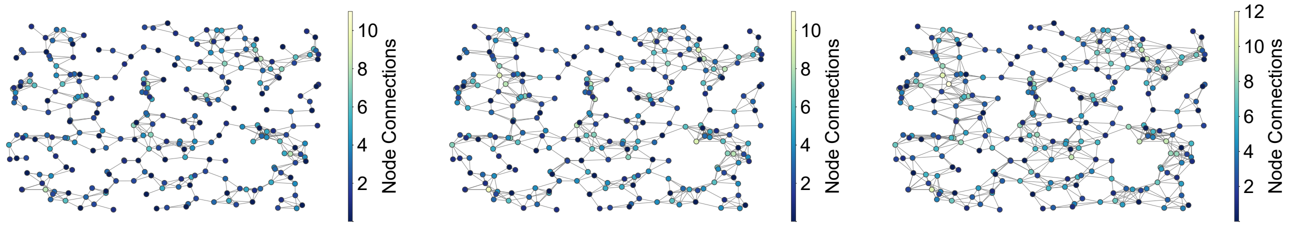


(a) Bern 5G base station locations (b) Geneva 5G base station locations (c) Zurich 5G base station locations

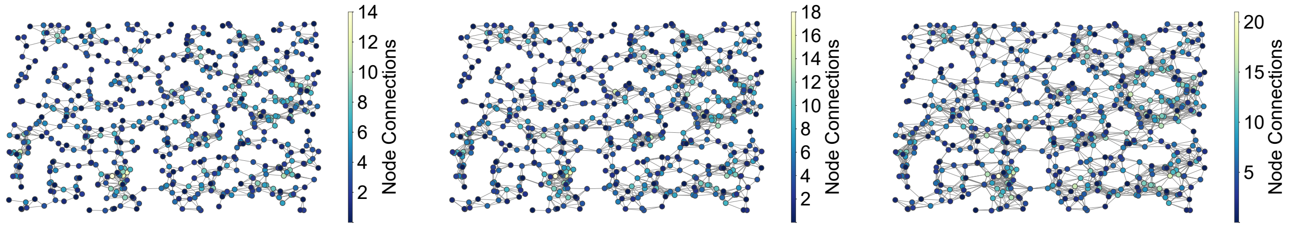
Fig. 4. Physical 5G network infrastructure map of the cities of Geneva, Bern and Zurich.



(a) Node connections established through radius 0.18. (b) Node connections established through radius 0.23. (c) Node connections established through radius 0.29.



(d) Node connections established through radius 0.15. (e) Node connections established through radius 0.17. (f) Node connections established through radius 0.19.



(g) Node connections established through radius 0.16. (h) Node connections established through radius 0.19. (i) Node connections established through radius 0.22.

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

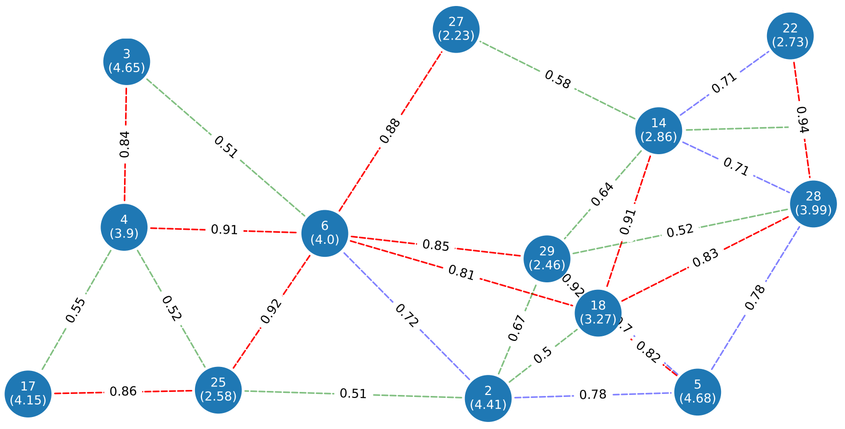


Fig. 6. Generated topology from part of the topology of the city of Bern with network and computing latencies. Green, blue, and red lines represent low, average, and high link latency.

consumption for the user applications. We assume that time is partitioned in a series of consecutive time windows of duration $T = 12\,\text{s}$, 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 so that the experiment yields 3.00×10^3 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 over 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, the more power and lower latency are required to process the video stream set to this resolution.

The *average accepted 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 at 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.

Computing latency for HMDs		
Minimum (p_m)	**Average (p_m)**	**Maximum (p_m)**
5×10^{-3} [s]	7.50×10^{-3} [s]	10×10^{-3} [s]
Computing latency for MECs		
Minimum (p_m)	**Average (p_m)**	**Maximum (p_m)**
3×10^{-3} [s]	4×10^{-3} [s]	5×10^{-3} [s]
Link latency for all topologies		
Minimum (k_m)	**Average (k_m)**	**Maximum (k_m)**
5×10^{-3} [s]	7.50×10^{-3} [s]	10×10^{-3} [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 III. Simulation parameters.

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 [35], [36]. It is worth noting that the video resolution selection is derived from the E2E latency provided by each algorithm during the VR user mobility.

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.

7) Simulation Parameters: For each topology described in Section V-A4, 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 by 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 infrastructures, directly impacting the latency. All simulation parameters are described in Table III.

B. Results

To validate the approach presented in this paper, we implemented a prototype of TENET, available at [37]. Our evaluation focuses on two major sets of results. We first assess the QoS for both Echo VR and Elixir[3] 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.

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 to allow VR users to use their hands in place of VR controllers. Elixir supports a refresh rate of 72 Hz.

a) Frame analysis: Figures 7a, 7b, and 7c compare both games in terms of overall *frame rate*, *stale frames*, and *early frames* over different *refresh rates*. While the frame rate is the number of images an HMD sends to its display every second, the refresh rate refers to how fast the display shows those frames.

Figure 7a 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. 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. However, these adjustments could harm the performance of produced FPS for a VR application.

Figure 7b 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. 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 7c 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 7a, 7b, and 7c 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.

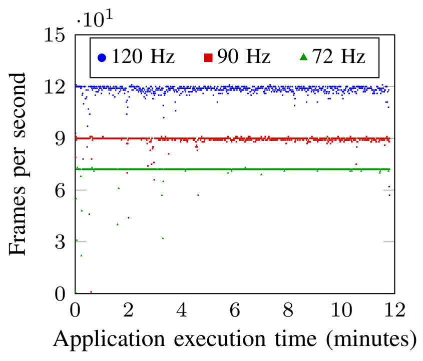
b) Computing latency analysis: VR systems have different sources of latency, e.g., the time between pressing a button and when the VR system detects it or when a frame is rendered until it appears on the VR HMD's screen. We focus on the time from when the VR system requests the user head orientation until the frames are rendered on the HMD. Figure 7d compares the computing latencies for each phase of a loop on Meta HMD.

Figure 7d 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 3.60 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.

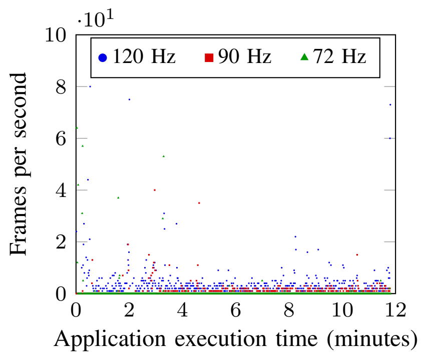
Figure 7d shows how much time the Asynchronous Time-Warp (ATW) spends to apply distortions and displays the scenes on Meta HMD for both games. ATW is a software component that transforms stereoscopic images based on the latest head-tracking information to reduce the motion-to-photon latency, shifting the rendered image to adjust for changes in head movement. Echo VR demanded 2.30 ms (120 Hz) and 2.60 ms (90 Hz) during the ATW phase. On the contrary, Elixir demanded 2.10 ms during the ATW phase.

Figure 7d shows the E2E latency of Meta HMD. 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 not specified in Meta HMD API. The mean E2E latency for Echo VR is about 30.50 ms (120 Hz) and 34.10 ms (90 Hz),

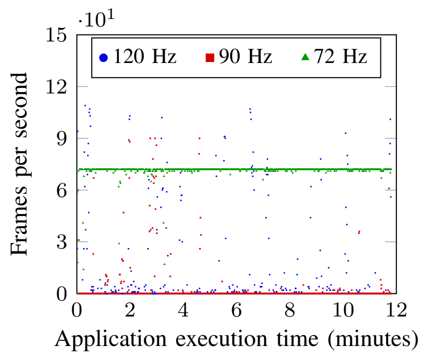
[3]https://www.oculus.com/experiences/quest/



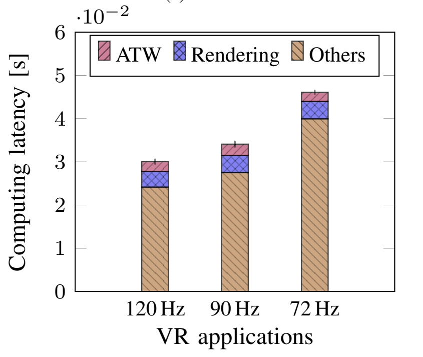
(a) Frame rate



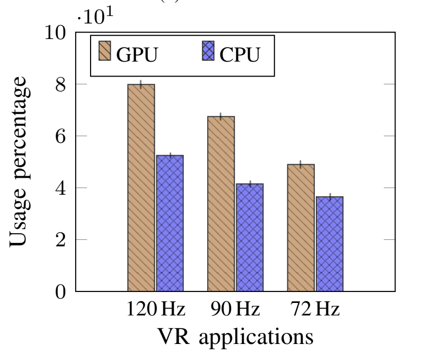
(b) Stale frames



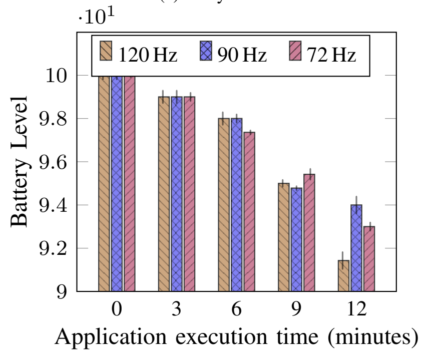
(c) Early frames



(d) Computing latency of different tasks running of Meta.



(e) CPU and GPU usage of Meta.



(f) Energy consumption of Meta.

Fig. 7. Frames, computational latency, GPU, CPU, and power consumption benchmarking of Echo VR and Elixir games running on Meta HMD.

respectively. Nevertheless, Elixir reached a mean E2E latency of 46.90 ms, representing 53.77% more than Echo VR E2E latency. Noticeably, Echo VR offers lower E2E latency than Elixir.

Other findings from Figure 7d 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.

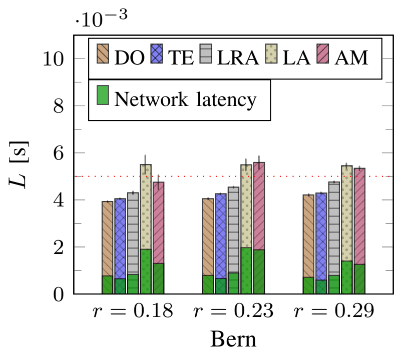
c) CPU usage, GPU usage, and Energy Consumption: Figure 7e compares both Echo VR and Elixir games' *GPU usage*, *CPU usage*, and *energy consumption*. GPU and CPU utilization are important to understand if a VR application is GPU or CPU bound. In particular, GPU utilization is more valuable than CPU 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 7e 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.

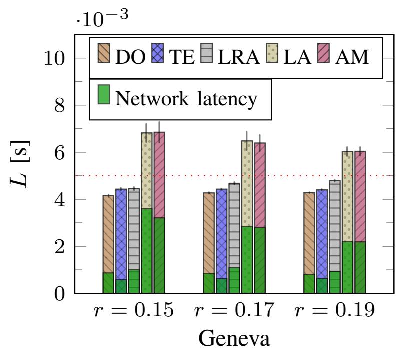
Figure 7e also provides the CPU usage, which considers 8 CPU cores available in Meta HMD. In practice, it is infeasible for an HMD only to have a powerful GPU, because a powerful CPU is required to reach frame rate stability. Thus, both GPU and GPU need to have a balance in terms of computing power. In most cases, VR applications will have a balance with favoring GPU over CPU due to graphical requirements. Results from Figure 7e 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.

VR applications require complex processing, which quickly drains the battery. Figure 7f represents the power consumption for both Echo VR and Elixir. While Echo VR drained 9% and 6% of its battery, respectively, Elixir drained 7% of its battery. This result reveals that VR-intensive computing applications may drive higher energy consumption in HMDs due to their need for computational resources. As a consequence, the energy constraints impair the user experience. Moreover, we observe that Elixir consumes more energy than Echo VR due to hand-tracking features to enhance the game experience.

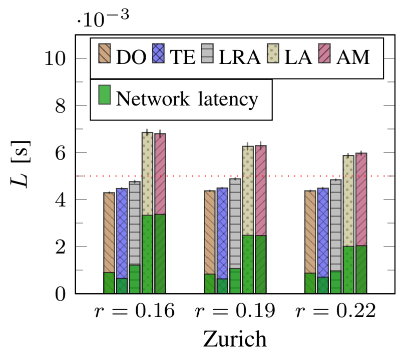
In a nutshell, Meta HMD QoS analysis can be described according to the following observations. (*i*) Higher refresh rates allow for more immersive experiences at the cost of



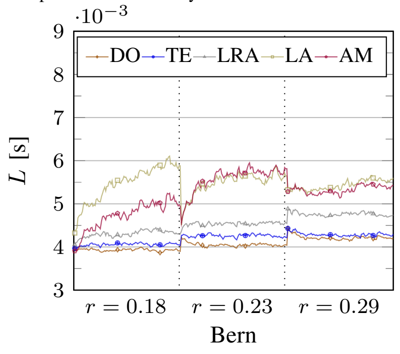
(a) Service average E2E latency L over different radii r for the city of Bern. Each bar shows computation latency on top of network latency.



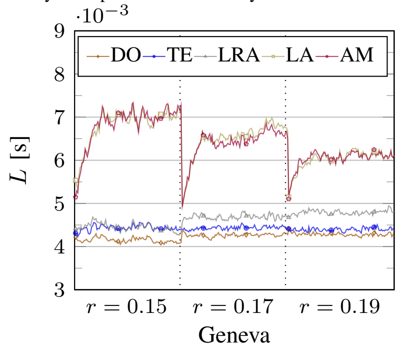
(b) Service average E2E latency L over different radii r for the city of Geneva. Each bar shows computation latency on top of network latency.



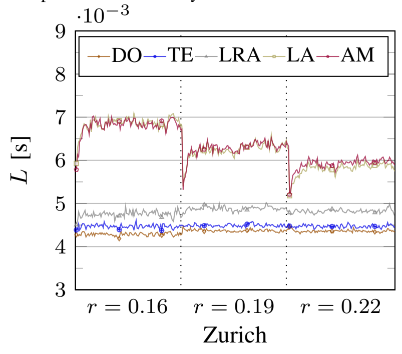
(c) Service average E2E latency L over different radii r for the city of Zurich. Each bar shows computation latency on top of network latency.



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

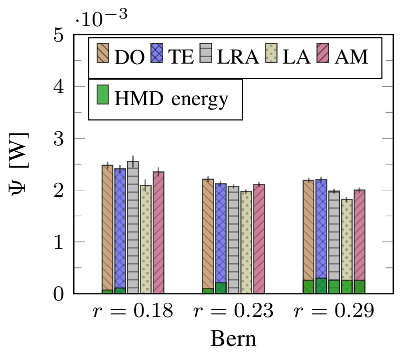


(e) Service E2E latency L over time over different radii r for the city of Geneva.

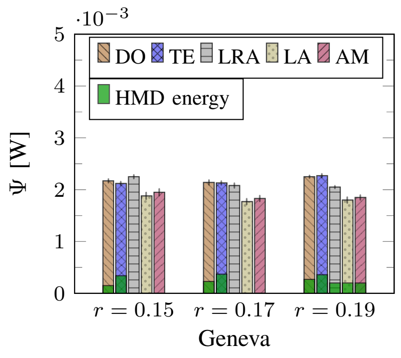


(f) Service E2E latency L over time over different radii r for the city of Zurich.

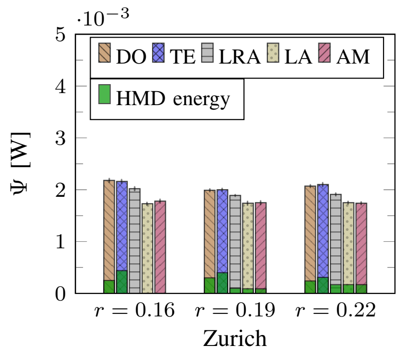
Fig. 8. Performance evaluation of end-to-end latency and its convergence for the topologies of Bern, Geneva, and Zurich. DO indicates the global optimum latency in each iteration. Error bars indicate 95% confidence intervals.



(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.

Fig. 9. Performance evaluation of HMDs power consumption for the topologies of Bern, Geneva, and Zurich . Error bars indicate 95% confidence intervals.

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.

2) *TENET Simulation Evaluation:*

a) E2E latency: Figures 8a, 8b, and 8c 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 (on average 35%) more services 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

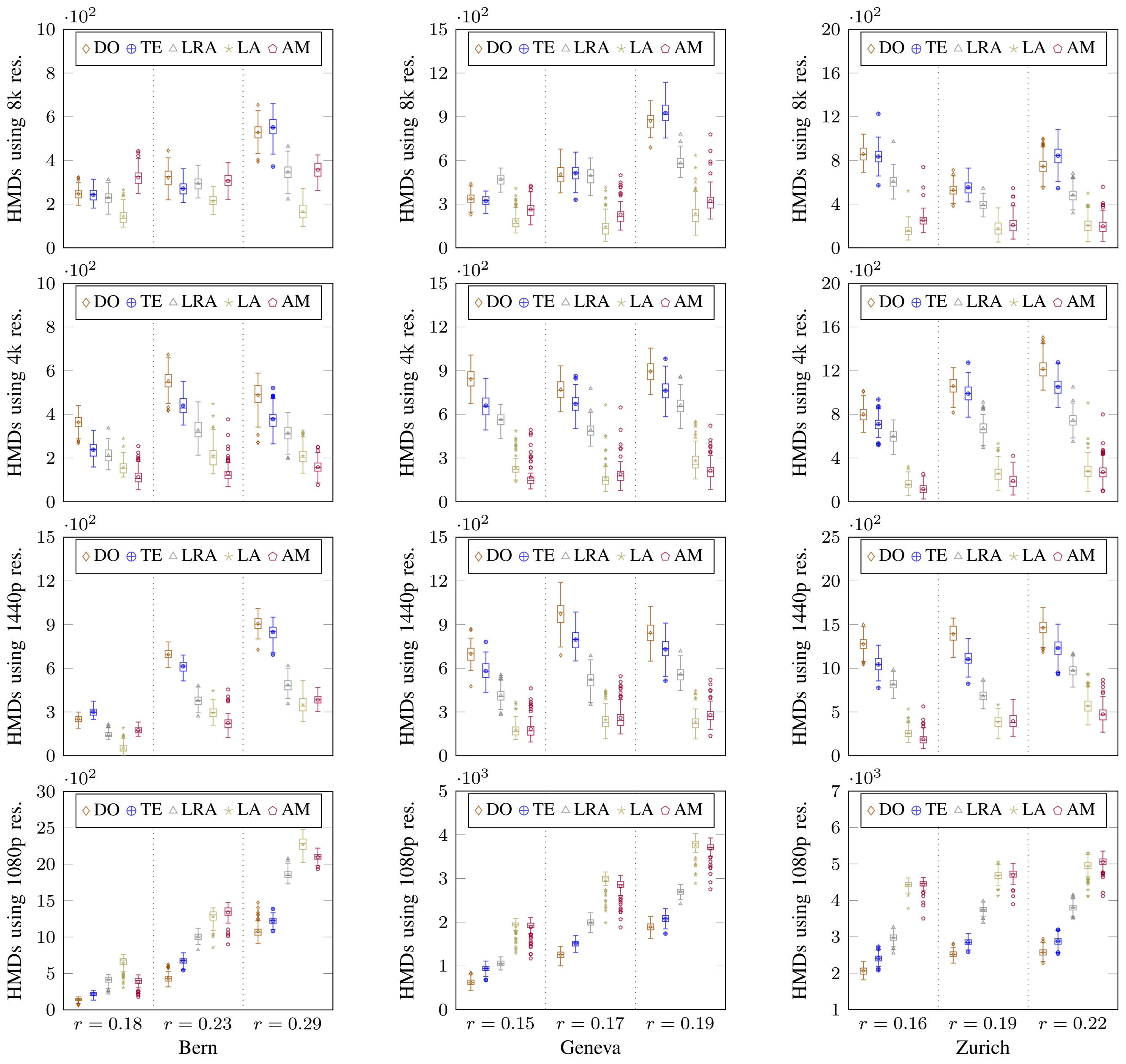


Fig. 10. Average of total HMDs using resolutions 8k, 4k, 1440p, and 1080p over different radii r for the cities of Bern, Geneva, and Zurich.

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 8d, 8e, and 8f 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 8f 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. 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 the 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.

b) Power consumption: Figure 9 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 al-

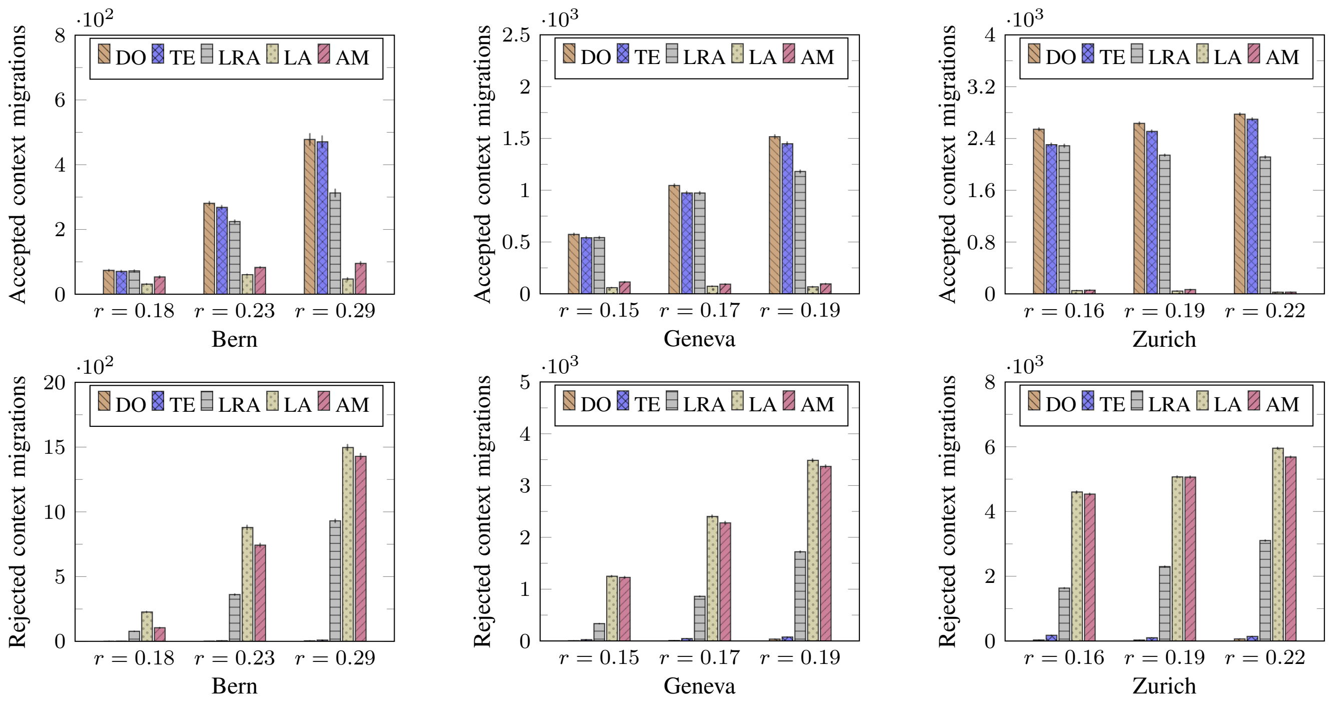


Fig. 11. Average application context acceptance and rejection migrations over different radii r for the cities of Bern, Geneva, and Zurich.

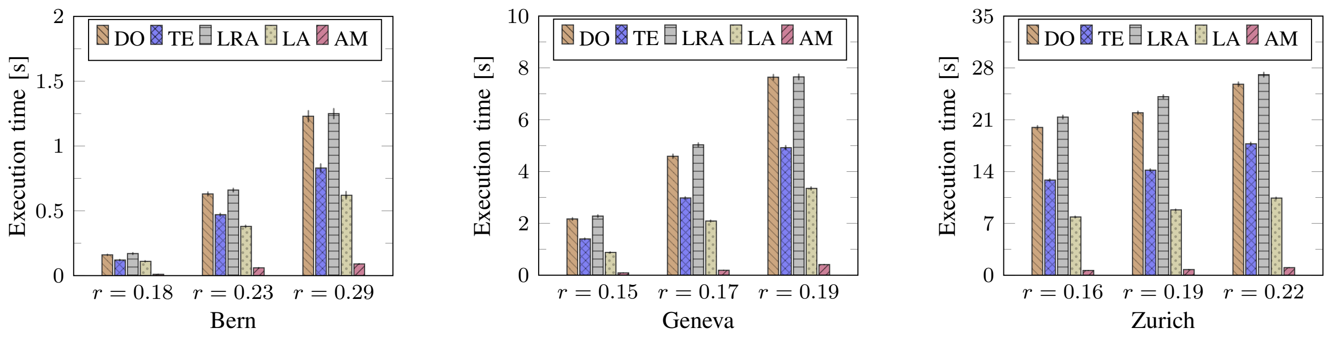


Fig. 12. Average of total execution time to provide placement for all services over different radii r for the cities of Bern, Geneva, and Zurich.

gorithms 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 8), 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 9a (r = 0.29), 9b (r = 0.19), and 9c (r = 0.19 and 0.22), where the entire MEC infrastructure was overloaded due to the number of users and available MEC servers. Whenever all MEC servers become unavailable for service deployment, services remain on HMD. 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.

c) Video resolution selection: Figure 10 shows 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. However, this 20% difference drops significantly to 5% as more complex scenarios with more users are considered. 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. Therefore, the TENET solution is promising for supporting VR applications running high-definition videos as it reduces the E2E latency compared to the LRA, LA, and AM mechanisms.

d) Context service migrations: Figure 11 shows the *average accepted* 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. We found that TENET provides a higher acceptance context migration than all other algorithms, except for DO, in all scenarios. We observe that TENET keeps its 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. Therefore, DO, TENET, and LRA provide a higher acceptance context migration ratio and lower rejected context migration ratio, while LA and AM have the opposite behavior.

e) Execution time: Figure 12 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 occurs because AM always tries to migrate the context to the MEC server attached to the base station where the HMD is connected. This result suggests that in a real scenario with many more users, 5G base stations, and MEC servers, DO has 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 $\mathcal{O}(\mathbb{E}log(\mathbb{V}))$.

VI. Conclusions

In this article, we have proposed a novel strategy to minimize the E2E latency for the next generation of 6DoF 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. We propose TENET, a fast heuristic to solve the DSCP problem. TENET manages 6DoF VR services by distributing them over edge networks and HMDs to avoid increased latencies. Through network simulations, we show 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 show that TENET can reduce the decision time on where to place the services while ensuring the performance of 5 ms. Therefore, TENET can satisfy the E2E latency requirements for 6DoF VR applications while providing higher video resolutions to improve the VR user experience.

References

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

[2] AT&T. (2017) Enabling mobile augmented and virtual reality with 5g networks. [Online]. Available: https://about.att.com/content/dam/innovationblogdocs/EnablingMobileAugmentedandVirtualRealitywith5GNetworks.pdf.

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

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

[5] 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.

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

[7] G. Berardinelli, P. Baracca, R. O. Adeogun, S. R. Khosravirad, F. Schaich, K. Upadhya, D. Li, T. Tao, H. Viswanathan, and P. Mogensen, "Extreme communication in 6g: Vision and challenges for 'in-x'subnetworks," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 2516–2535, 2021.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, 2019.

[14] Q.-V. Pham, F. Fang, V. N. Ha, M. J. Piran, M. Le, L. B. Le, W.-J. Hwang, and Z. Ding, "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.

[15] A. Medeiros, A. Di Maio, T. Braun, and A. Neto, "Service chaining graph: Latency-and energy-aware mobile vr deployment over mec infrastructures," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022, pp. 6133–6138.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] C. Wang, S. Zhang, Z. Qian, M. Xiao, J. Wu, B. Ye, and S. Lu, "Joint server assignment and resource management for edge-based mar system," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2378–2391, 2020.

[21] 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. 101380, 2021.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] S. Confederation. (2022) Maps of switzerland. [Online]. Available: https://map.geo.admin.ch/.

[35] Z. Rejiba, X. Masip-Bruin, and E. Marín-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.

[36] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 163–174, 2014.

[37] A. Medeiros, "Service Chaining Graph Release V1," Aug. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.7004077

Alisson Medeiros is a Ph.D. candidate at the Communication and Distributed Systems (CDS) group, Institute of Computer Science, University of Bern, Switzerland. He received his MSc from the Federal University of Rio Grande do Norte (UFRN), Brazil, and obtained his BSc in computer science at the State University of Paraíba, Brazil. His research interest includes Virtual Reality, Augmented Reality, Mixed Reality and Edge Computing.

Antonio Di Maio is a postdoctoral researcher in mobile networks with the Communication and Distributed Systems (CDS) group at the University of Bern, Switzerland. He obtained his Ph.D. degree in Computer Engineering from the University of Luxembourg in 2020, with a thesis on routing and content dissemination in software defined vehicular networks. His current research interests fall within the areas of network modeling, scheduling, routing, and channel access.

Torsten Braun is currently director at the Institute of Computer Science, University of Bern, where he has been a full professor since 1998. He got the Ph.D. degree from University of Karlsruhe (Germany) in 1993. From 1994 to 1995, he was a guest scientist at INRIA Sophia-Antipolis (France). From 1995 to 1997, he worked at the IBM European Networking Centre Heidelberg (Germany) as a project leader and senior consultant. He has been a vice president of the SWITCH (Swiss Research and Education Network Provider) Foundation from 2011 to 2019. He has been a Director of the Institute of Computer Science and Applied Mathematics at University of Bern between 2007 and 2011, and from 2019 to 2021. He received best paper awards from LCN 2001, WWIC 2007, EE-LSDS 2013, WMNC 2014, and the ARMS-CC-2014 Workshop as well as the GI-KuVS Communications Software Award in 2009. He also received several best poster awards at Bern Data Science day 2022 and 2021, and the best poster award from Adhoc-Now 2019. In the scope of EU funded projects, he was leading WPs of FP6-EUQOS and FP7-MCN. Moreover, he coordinated national projects such as SNSF Swiss Sense Synergy and SNSF CONTACT.

Augusto Neto is Associate Professor at the Informatics and Applied Mathematics Department(DIMAp) of the Federal University of Rio Grande do Norte(UFRN), member of the Instituto de Telecomunicações (Aveiro, Portugal), researcher of the National Council for Technological and Scientific Development (CNPq), and leader of the Research Group on Future Internet Service & Applications(REGINA), working mainly in the fields of Computer Networks and Distributed Systems. He got his Ph.D. at the University of Coimbra (2008), and coordinates/participates in research projects funded by national and international development agencies. He is active member of the IEEE Communication Society and participates as editor/reviewer of important journals in the field of computer networks, and has authored and co-authored more than 150 papers. His current main research interest are Future Internet, 5G, SDN/NFV, Cloud/Edge/Fog Computing, IoT, and Smart Cities.