

QoS Provisioning for MEC Application Slices in Beyond 5G Networks

I. Vilà¹, O. Sallent^{1,2}, J. Pérez-Romero¹

¹Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, ²i2CAT Foundation, Barcelona, Spain.
irene.vila.munoz@upc.edu, sallent@tsc.upc.edu, jordi.perez-romero@upc.edu

Abstract— Beyond 5G and 6G mobile networks will have to deal with services of low latency and high computational requirements, such as immersive communications or artificial intelligence (AI)-based applications. This requires the integration of edge computing capabilities in the mobile network and the assurance of Quality of Service (QoS) guarantees across both network and edge domains. This paper presents a framework that integrates QoS provisioning mechanisms for Multi-Access Edge Computing (MEC) Application (App) slices based on the ETSI MEC architecture. The QoS provisioning mechanisms, particularized for the case of leveraging Kubernetes features, are empirically evaluated in an edge-enabled 5G testbed where gaming and buffered video services are deployed. Results demonstrate that the effective configuration of these mechanisms is key to ensuring service availability while enhancing the overall Quality of Experience (QoE).

Keywords— *Beyond 5G; 6G; Edge Computing; MEC App Slices; QoS; Kubernetes; Testbed.*

I. INTRODUCTION

While the 5G rollout is advancing, with 5G subscriptions representing around one-third of all mobile subscriptions by the end of 2025 [1], several research programs have been launched to conceptualize the 6G system, identifying the potential service requirements and key technologies [2]. The framework and objectives of the International Mobile Telecommunications (IMT)-2030 that will constitute the future 6G system were published in [3]. Use cases for Beyond 5G (B5G)/6G are expected to expand those for 5G into broader use, including immersive communications, artificial intelligence (AI) applications or hyper-reliable and low-latency communication. Behind these use cases, there are services such as eXtended Reality (XR), autonomous driving, holographic representations, etc., with stringent Quality of Service (QoS) requirements (e.g., extremely low latency and high reliability). Moreover, these applications will require extensive computational capabilities to handle large data volumes, execute AI and machine learning (ML) algorithms, and perform complex tasks like image rendering, etc.

To deal with the large computational loads and real-time computing requirements, edge computing has emerged as a key enabler that provides compute and storage capabilities at the edge of operators' networks, e.g., by co-locating computing servers with network base stations, transport nodes, etc. Many research efforts have been made in the area of edge computing, as reflected in survey papers, such as [4] and [5]. Similarly, standardization efforts have been mostly driven through the European Telecommunications Standards Institute (ETSI) Industry Specification Group (ISG) Multi-access Edge Computing (MEC) and the Third Generation Partnership Project (3GPP) [6]. While 3GPP has defined an architecture focused on

enabling and controlling edge computing within the 5G system, ETSI MEC has focused on a generic edge-cloud platform with agnostic access technology. In parallel to these efforts, the AI-RAN alliance also envisages an unprecedented convergence between communication and computing for the upcoming 6G system to the point that hardware used for running Radio Access Network (RAN) functionalities can also be leveraged to support AI-based applications running at the edge [7]. This concept is referred to as AI-on-RAN and envisages the execution of, among others, AI applications and Generative AI XR applications [8]. This vision is key to providing new revenue options for network infrastructure providers, enabling them to offer computing services in addition to communication services. To take advantage of this, multiple services must be provided concurrently at both the network and edge computing domains, ensuring the required QoS guarantees for each of them.

In the network domain, network slicing has matured as a key technology to provide optimized service connectivity across network nodes. It allows sharing a common network infrastructure by creating multiple end-to-end logical networks on top of it, denoted as network slices, each one optimized to meet the connectivity requirements of a specific application domain. This technology has been studied profusely in the literature [9], and its standardization work has been mainly driven by 3GPP [10], covering architectural and management aspects as well as techniques for satisfying QoS requirements of services supported on network slices.

In the edge computing domain, ETSI MEC has introduced the concept of MEC Application Slice (MEC App slice) to manage and optimize the set of applications, referred to as MEC Apps, associated with a service running at the MEC system [11]. To materialize this concept, ETSI MEC has also defined a set of functionalities and procedures for the management of MEC App slices. However, being a concept still in an early development stage, only a few recent works have explored it [12]–[16]. The ETSI MEC architecture to support the MEC App slice concept and related functionalities is elaborated in [12]. In parallel to the ETSI MEC progress, some other works have also proposed architectures and frameworks for managing edge slices or application slices, which are similar concepts to MEC App slices. Among them, a mobile edge slice broker is proposed in [13] to orchestrate the provisioning of mobile edge slices that include network functions, services and edge applications, tackling the problem from a business perspective. The work in [14] presents a lightweight orchestrator that supports the lifecycle management (LCM) of edge slices, specifying their requirements. Instead, the architecture proposed in [15] covers the LCM of end-to-end slices, encompassing network slices and application slices, identifying high-level roles and studying the requirements of LCM of edge sub-slices, introducing a template

for ETSI MEC to accommodate the management of such slices. From an implementation perspective, the work in [15] presents a preliminary proof-of-concept of deploying MEC App slices based on Kubernetes. Similarly, a broader implementation of the MEC system is proposed in [16], including MEC App slices. However, MEC App slices in [15] and [16] are considered part of a network slice, which deviates from the ETSI MEC view of considering them as separate entities from network slices.

This paper focuses on the QoS provisioning for MEC App slices, a critical aspect in edge computing environments with limited computing resources. Only a few works in the literature have covered this topic to some extent [13][15]. The framework proposed in [13] considers QoS requirements during the MEC App slices design phase and QoS monitoring during MEC App slices execution, but it does not specify the mechanisms to support these functions. In turn, the proof of concept in [15] uses Kubernetes tools for enforcing isolation between MEC App slices rather than for actual QoS provisioning. Moreover, the actual radio connectivity is not included in their testbed.

Advancing the state-of-the-art, this paper presents an architectural framework for QoS provisioning of MEC App slices. The main contributions of the paper are: (i) The presented framework integrates different QoS provisioning mechanisms at different levels of the ETSI MEC architecture. These mechanisms are particularized by leveraging native Kubernetes capabilities. (ii) The integrated QoS-aware mechanisms are empirically evaluated on an edge-enabled 5G testbed. To the authors' best knowledge, this constitutes the first work to assess MEC App slices in a testbed with 5G connectivity. This allows assessing the performance not only in terms of computing resource metrics, as in previous works, but also in terms of end-to-end latency and user Quality of Experience (QoE).

The remainder of this paper is organized as follows. Section II presents the concept of MEC App slices and their operation and management according to ETSI MEC, while Section III proposes the integration of two types of QoS provisioning mechanisms for MEC App slices with the ETSI MEC architecture. Section IV includes the empirical validation, and Section V closes with the conclusions and future work.

II. MEC APPLICATION SLICES

A. Concept

The concept of MEC App slice has been introduced by the ETSI MEC standardization group in [11] based on some initial works conducted in [17]. It is defined as a "logical MEC application service environment, which provides specific MEC application functions and related MEC services characteristics" [11]. The motivation behind this concept is that, from a customer-oriented perspective, a virtualized application (e.g., XR, AI) requiring computations at the MEC system cannot be considered solely as a part of the network slice. The applications' requirements go beyond those of the network (e.g., data rate, latency, reliability) to encompass others such as computing resources, application-level isolation, or virtualization approach (e.g., application deployment as virtual machines or containers). With this vision, to provide a service that demands computation at the MEC system, an end-to-end slice is required, composed of a network slice that provides the network connectivity and a MEC App slice that provides the computational power at the edge [12]. The MEC App slice will be composed of a number of MEC Apps according to the

architecture of the application [15]. For instance, to support a holographic service, the MEC App slice can include a MEC App for rendering and another one for managing multi-user streams. Also, if multiple MEC Apps of different users of the same application need to be instantiated, all of them can be included in the same MEC App slice.

Fig. 1 illustrates the concept with an example of the provision of XR and holographic communications services, each with different QoS requirements and offloading computationally intensive tasks to the edge. Each service is supported by a different network slice, comprising Radio Access Network (RAN), Transport Network (TN), and Core Network (CN) subnet slices, and by a MEC App slice at a MEC host co-located with the base station, i.e., the gNode-B (gNB) in 5G. For each service, the User Equipment (UE) sends requests for task computation at the edge to the gNB via the corresponding RAN slice. Then, the gNB forwards these requests to the MEC host through a local User Plane Function (UPF). For instance, requests for rendering tasks of a user of the XR service are sent to the gNB through RAN slice #i and then forwarded to the MEC host, where they are processed by the rendering MEC App of MEC App slice #x.

B. MEC App slices operation and management

Fig. 2 illustrates the relevant entities of the ETSI MEC architecture for supporting MEC App slices. MEC App slices run at the *MEC host*, which contains the computing, storage, and network resources used by the MEC Apps. These resources are part of the *Virtualization Infrastructure*, which also includes the data plane that executes traffic rules and routes traffic between applications, the network, etc. The *MEC host* also contains the *MEC platform*, which is responsible for controlling the traffic rules applied in the data plane, configuring the Domain Name System (DNS) server, and offering certain MEC services that can be consumed by the MEC Apps or by the MEC platform itself. Examples of these services are the Radio Network Information Service (RNIS) that provides MEC Apps with radio network-related information, and the QoS-aware services proposed in this paper that consider the QoS requirements of MEC App slices for traffic prioritization purposes, as will be explained later in Section III. While the ETSI MEC entities might support different access technologies, Fig. 2 shows the particular case of 5G by depicting on the left side (in blue) the UE, the RAN, and the 5G core (5GC), including the local UPF.

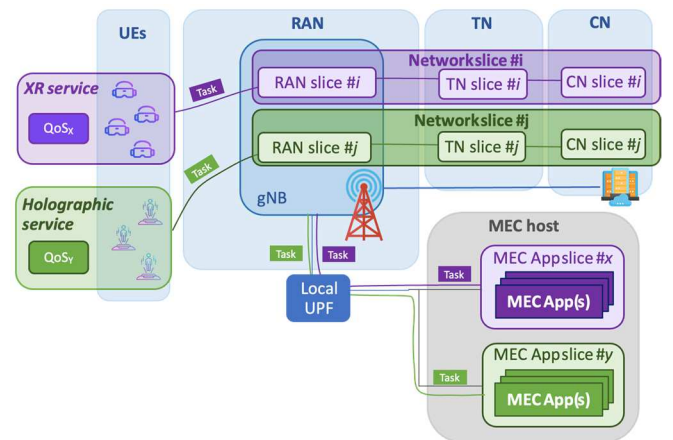


Fig. 1. System overview with network slices and MEC App slices.

Fig. 2 also includes ETSI MEC functionalities essential for managing MEC App slices. These are distributed between the system and host levels. At the system level management, the *Operations Support System* (OSS) manages requests to instantiate or terminate MEC Apps coming, e.g., from third-party customers. The OSS includes the MEC App slice Communication Service Management Function (MAS-CSMF) that translates high-level service-related QoS requirements into MEC App slice requirements [12]. The OSS also includes the MEC App slice Management Function (MAS-MF), responsible for the design of the MEC App slice Template (MAST), which is a collection of parameters that define an information model of the MEC App slice (e.g., identifier, name, description). The MAS-MF also performs the LCM of the MEC App slices according to the requirements specified by the MAS-CSMF. This is done by interacting with the *MEC Orchestrator* (MEO), which has an overall view of all the deployed MEC hosts and is responsible for onboarding application packages, selecting MEC hosts, and triggering the instantiation, termination, or relocation of MEC Apps. In this way, the MAS-MF requests the MEO to create, update, and terminate MEC App slice-related elements (i.e., MEC Apps and associated MEC services if required) and their packages and images. For creation and update operations, the MEO establishes the resource requirements (e.g., CPU, memory, network).

At the host level, the management is conducted by the *MEC platform manager* (MEPM), responsible for the LCM of MEC Apps within the MEC host, and by the *Virtualisation Infrastructure Manager* (VIM), which allocates, manages and releases the virtualized resources. Upon the request to instantiate a MEC App slice from the MEO, the MEPM proceeds with the requested allocation of computing resources, which is negotiated with the VIM, and with the MEC platform services configuration, if required. To ensure the QoS of MEC App slices, the MEPM enforces QoS-aware policies and authorises MEC App slices to use QoS-aware services available in the MEC platform, as it will be detailed in the following section.

III. QoS PROVISIONING MECHANISMS FOR MEC APP SLICES

Although ETSI MEC specifies the entities and procedures required to manage MEC App slices and support their LCM, the specific mechanisms for ensuring their QoS fall outside the scope of the standard, as they are implementation-dependent. These mechanisms are particularly relevant when multiple MEC App slices simultaneously run on a MEC host with limited resources. While the MEO can relocate the MEC App slice to a different MEC host with more resources available or to the cloud, this might not always be possible or convenient due to, e.g., extremely low-latency requirements, hardware constraints, or security measures. Therefore, QoS provisioning mechanisms are needed to efficiently distribute the computational resources in a MEC host, considering the QoS of MEC App slices. Fig. 2 identifies two of these mechanisms, coloured in red and labeled as A and B, respectively.

The first mechanism is the definition of QoS-aware policies at the MEC platform manager level (see label A in Fig. 2). These policies consider the QoS requirements of MEC App slices to manage the allocation of computational resources and resolve conflicts when there are not enough resources. These QoS-aware

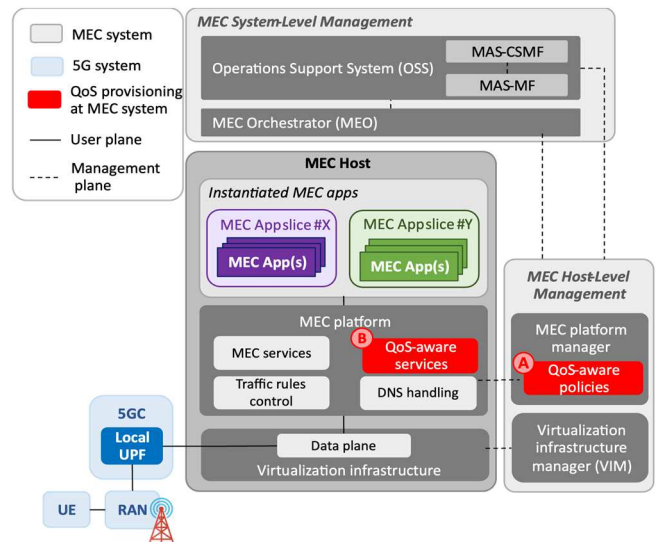


Fig. 2. Management and operation of MEC App slices.

policies depend on the platform used to deploy the MEC system. In the case of Kubernetes, which is a commonly used platform for MEC (see e.g. [18]), MEC Apps of a MEC App slice will be deployed as pods (i.e., the smallest deployable units of computing that can be created and managed in Kubernetes).

The QoS-aware policies can leverage the following Kubernetes features:

- **QoS classes:** Three classes of pods are defined, namely *guaranteed*, *burstable*, and *best-effort*, each one involving a different allocation of computing resources. Guaranteed pods are provided with exactly the requested computing resources, burstable pods are provided with the minimum requested resources but can use more if available, and best-effort pods use the remaining resources.
- **Pod priority and preemption:** In Kubernetes, once a pod is requested to be created, it is included in a queue. Each pod can be given a priority that determines how it is scheduled and served in relation to the other pods in the queue. Also, a preemption capability can be configured that determines if the pod can be terminated whenever another pod with higher priority is scheduled.
- **Bandwidth shaping:** It is possible to define rules that limit or prioritize the amount of bandwidth available in the ingress and egress interfaces between the UPF and specific pods.

Moreover, to achieve a higher level of isolation between MEC App slices, they can be deployed as *namespaces*. In a namespace, all the pods (i.e., the MEC Apps) of a MEC App slice are treated as a group. Then, resource quotas and bandwidth shaping policies can be defined at the *namespace* level using features like ResourceQuota and LimitRange.

The second mechanism identified in Fig. 2 (see label B) is the use of QoS-aware services of the MEC platform, such as load balancers (e.g., a Kubernetes ingress service using NGINX controller), which can be deployed at the MEC host. When the MEC host receives the user plane packets of the UEs requesting the execution of computation tasks, these requests are first forwarded to the load balancer, which prioritizes them based on their QoS requirements and forwards them to their destination MEC App slice. In this way, the requests associated with services with higher priority will be processed first.

IV. EMPIRICAL VALIDATION OF QoS PROVISIONING MECHANISMS FOR MEC APP SLICES

A. Testbed implementation

This section presents an edge-enabled 5G testbed to showcase the relevance of incorporating QoS provisioning mechanisms within the MEC platform to manage the computing resources among MEC App slices. Fig. 3a shows the deployed testbed, which consists of an Amarisoft Callbox Classic 5G network equipment [19], an edge server, and a switch connecting them via a Local Area Network (LAN). The setup includes two UEs equipped with programmable Subscriber Identity Module (SIM) cards supporting 5G and the monitors to operate the Amarisoft and edge server equipment.

Fig. 3b depicts the testbed architecture highlighting the functionalities of each component. The Amarisoft equipment integrates both the gNB and the 5GC into a single device and supports both Standalone (SA) and Non-Standalone (NSA) 5G modes. For the validation in this paper, the equipment is configured as SA 5G with a single New Radio (NR) cell operating in the n78 band with 20 MHz bandwidth using Time Division Duplex (TDD). 5G connectivity is provided to two Crosscall CORE-Z5 UEs.

The edge server, which includes the MEC Host and MEC management functionalities as shown in Fig. 3b, features an Intel(R) Core(TM) i5-6400, 4 CPUs, 8 GB of RAM, and 256 GB of hard disk. The operating system is Ubuntu 22.04.5 LTS with Docker (version 28.0.4) and Kubernetes installed on top of it. Docker enables the creation and running of containers, while Kubernetes allows the management and automation of the deployment, scaling, and operation of containers in the environment. The Minikube (version v1.36.0) Kubernetes distribution [20] has been installed since it allows testing the required MEC App slices in a local environment with reduced complexity of installation and operation. In this setup, the Docker engine and the Kubelet component of Minikube act as virtualization infrastructure and VIM of Fig. 2, respectively. The Kubernetes Control Plane serves as the MEPM, providing automated MEC App LCM, resource scheduling, and policy enforcement, including the QoS-aware policies. The current testbed implementation supports QoS-aware policies, while the

incorporation of QoS-aware services is left as future work.

On top of Kubernetes and Docker, the applications associated with different services are deployed in different MEC App slices in the form of pods. These are instantiated using MASTs in the form of YAML manifests, which also include the parameters of the QoS-aware policies to be applied to each of them, as explained in section IV.B. In addition, Prometheus and Grafana pods are deployed for monitoring the CPU, memory usage, and pod status when running the different MEC App slices in the different tests.

B. Validation use case

Two concurrent services are deployed in the testbed. *Service #1* corresponds to the Minecraft game [21] and *Service #2* to the VLC media player [22], as an example of a buffered video service. At the edge server, they are supported by MEC App slices #1 and #2, respectively. Each one includes a MEC App with the server-side application. The client-side application of each service is installed on a different UE.

The main challenge of running the two MEC App slices concurrently is ensuring a consistent user-perceived QoE, particularly when the edge server is congested. To deal with these situations, it is key to implement and properly configure QoS provisioning mechanisms to provide sufficient computing resources to MEC App slices. To demonstrate this, the following QoS-aware policies are tested, allowing the systematic study of the impact of each Kubernetes feature:

- **Policy #1:** Serves as a baseline, considering the QoS class of MEC App slices #1 and #2 as *best-effort* and without using pod priority and preemption.
- **Policy #2:** Configures the *guaranteed* QoS class for both MEC App slices. Resource guarantees are set at 0.6 CPUs and 1.5 GB of memory for MEC App slice #1, and 0.2 CPUs and 1 GB of memory for MEC App slice #2. The pod priority and preemption capability is also configured: MEC App slice #1 is assigned a higher-priority class with preemption disabled, and MEC App slice #2 is assigned a lower-priority class with preemption enabled.
- **Policy #3:** Configures the *guaranteed* QoS class for both MEC App slices but with lower resource guarantees than in Policy #2. In this case, resource guarantees are set at 0.5

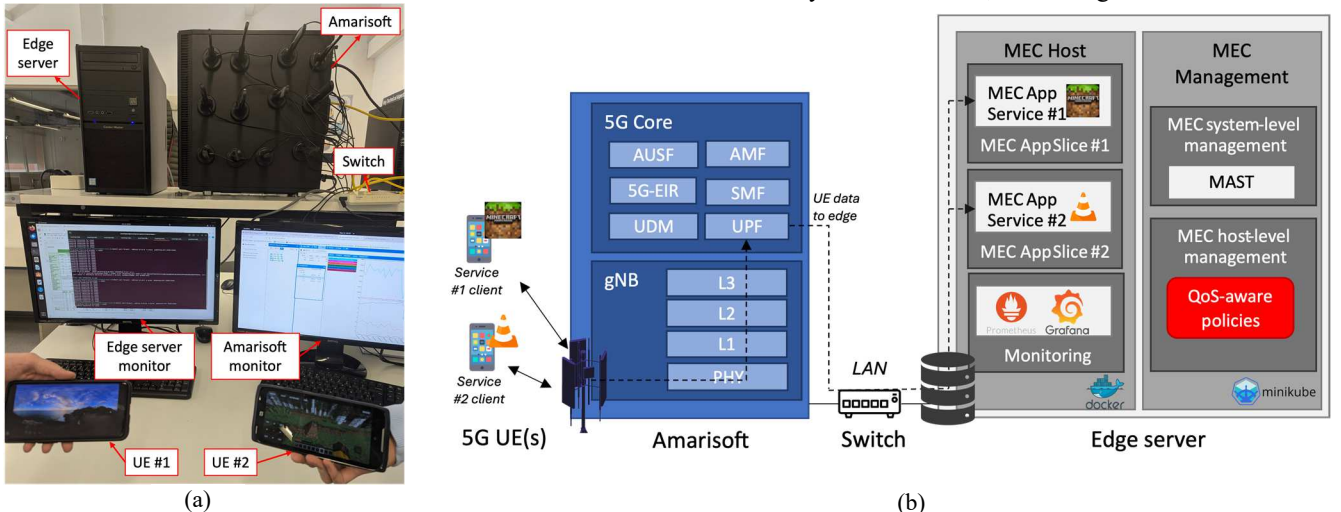


Fig. 3. Implemented testbed: (a) Testbed deployment; (b) Testbed architecture.

CPUs and 1.5 GB of memory for MEC App slice #1 and 0.1 CPUs and 1 GB of memory for MEC App slice #2. The pod priority and preemption capability is also configured as in Policy #2.

- **Policy #4:** Configures the *burstable* QoS class for both MEC App slices. The minimum requested resources for MEC App slice #1 are 0.3 CPUs and 0.7 GB of memory, while for MEC App slice #2, they are 0.01 CPUs and 35 MB of memory. The pod priority and preemption configuration is the same as for Policy #2.

To evaluate the policies under high-load edge server conditions, a stress-ng pod generating a constant load of 2.5 CPU cores and 1 GB of memory is deployed. Since radio conditions between the Amarisoft and the UE also impact the QoE, tests are conducted under two scenarios. *Scenario A* denotes favourable radio conditions, in which both UEs experience high radio channel quality. In this scenario, the measured average Reference Signal Received Power (RSRP) is -84.6 dBm, and the average Channel Quality Indicator (CQI) equals the maximum value 15 for both UEs. *Scenario B* represents poorer radio conditions with degraded radio channel quality for both UEs. In this case, the measured average RSRP is -113.4 dBm and the average CQI is 8.20, with a standard deviation of 0.92.

C. Performance evaluation

Table I includes the performance metrics achieved with each policy and scenario in terms of the average CPU and memory usage, average latency, and standard deviation of the latency. CPU and memory usage have been measured through a Grafana dashboard, fed by Prometheus. Presented values are the average of 5 test executions. Latency of Service #1 is measured through an internal latency metric of the Minecraft application at the client side. For Service #2, it is based on the application response time obtained by inspecting the packets exchanged between the VLC server and the client. Average and standard deviation values are obtained using a total of 50 samples of latency per service (i.e., 10 samples per test execution).

Results show that Service #1 has greater CPU and memory usage than Service #2. The latency is significantly higher for Service 2, but this difference mostly comes from the different nature of the traffic generated by each service. Regarding the impact of policies, Policy #1 results in lower average resource usage for both services. This leads to higher average latency values with high variability, as services share the resources without guarantees, resulting in bursty performance. Under Policy #2, since the requested guarantees cannot be met by the available computing resources at the edge server, the Kubernetes system is only able to deploy MEC App slice #1, which has higher priority, while MEC App slice #2 remains in pending status. This illustrates the impact of the pod priority and preemption configuration that favors MEC App slice #1. This policy leads to the lowest values of average latency and standard deviation of Service #1. Under Policies #3 and #4, both MEC App slices #1 and #2 are deployed as the requested resources are less stringent. However, this is at the expense of a certain latency increase of Service #1 with respect to Policy #2. It is also observed that Policy #4 yields lower latencies than

TABLE I. SERVICES PERFORMANCE METRICS FOR POLICIES #1 - #4

Service	Policy	Scenario	CPU usage	Memory Usage (MB)	Av. Latency (ms)	Latency st. dev. (ms)
#1	#1	A	0.223	690.3	38.3	19.5
		B	0.455	797.6	45.8	15.6
	#2	A	0.515	813.2	29.1	6.3
		B	0.491	797.6	32.5	9.8
	#3	A	0.48	779.9	33.2	10.6
		B	0.46	790.4	39.7	12.3
	#4	A	0.559	874.2	31.5	8.4
		B	0.530	898.6	34.6	8.1
#2	#1	A	0.019	35.5	141.3	112.6
		B	0.070	48.4	178.6	117.2
	#2	A	N/A	N/A	N/A	N/A
		B	N/A	N/A	N/A	N/A
	#3	A	0.016	65.3	140.5	102.7
		B	0.022	63.0	163.9	235.8
	#4	A	0.021	62.0	123.4	80.6
		B	0.025	42.0	154.4	148.4

Policy #3, as no explicit resource limits are established for the MEC App slices, which allows using more computing resources than the minimum requested if they need them, accelerating the computing operations, and thus, reducing the latency.

Regarding the impact of the different radio conditions captured in Scenarios A and B, a first observation is that the comparative behavior of the policies described above remains consistent for both scenarios. However, the radio conditions impact some of the obtained metrics. Particularly, it is found that the average latencies in Scenario B exceed those of Scenario A for both services and all policies, due to the degraded radio conditions of UEs in Scenario B. The latency increase in Scenario B is around 15% for Service #1 and 20% for Service #2, indicating that Service #2 is a bit more sensitive to radio conditions. Moreover, the latency variability reflected in the standard deviation is also higher in Scenario B than in Scenario A. This is because the degraded channel quality in Scenario B results in a burstier and more variable channel. In terms of CPU and memory usage, results for both scenarios are generally similar. Only under Policy #1, MEC App slices #1 and #2 experience higher CPU and memory usage in Scenario B, as the adverse radio conditions in this scenario induce the MEC App slices to manage packet retransmissions and the synchronization of fragmented data streams. For the rest of the policies, this effect has less impact on the results because these policies provide better control of the CPU and memory usage for each MEC App slice. This reflects that Policies #2-#4 are also beneficial in minimizing the impact of radio conditions on computing resource usage.

The QoE has also been collected for both services and scenarios in the different test executions. This metric has been subjectively assessed by the human testers executing the two applications and considering the following grading scale: (1) does not work, (2) works briefly but crashes; (3) degraded operation; (4) acceptable operation with sporadic interruptions; (5) optimal operation. Fig. 4 presents the resulting average QoE

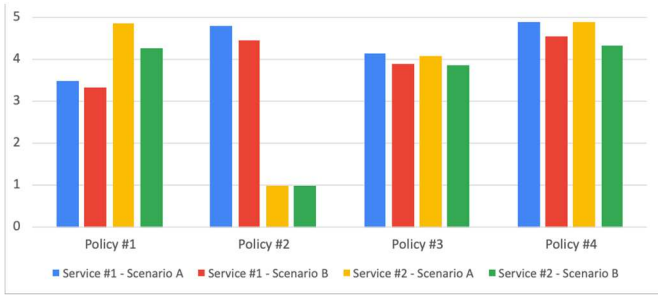


Fig. 4. QoE of both services for Policies #1-#4 and Scenarios A and B.

for each policy. Results show that policy configuration has a significant impact on Service #1 QoE, as the lack of computing resources and increased latencies of Policy #1 lead to a very bad gaming experience close to 3. For Policies #2 and #4, in which Service #1 resource usage is higher and latencies are lower, the QoE is very high. In turn, Service #2 QoE is less sensitive to the applied policies, with values above 3.8 as long as the corresponding MEC App slice #2 can be deployed, which occurs with all policies but Policy #2. Regarding the impact of radio conditions, the QoE is degraded in Scenario B compared to Scenario A for all the policies. This degradation is more pronounced for Service #2, which is a bit more sensitive to radio conditions.

Overall, the testbed results have empirically showcased the benefits of incorporating and properly configuring QoS-aware policies for managing MEC App slices that run concurrently on an edge server, allowing for consistent QoE of the services. However, results have also shown that, despite using QoS-aware policies at the edge, the radio conditions can still compromise the QoE. Therefore, a coordinated QoS provisioning that takes into account both the radio and the edge conditions is identified as a key challenge to be addressed in future work.

V. CONCLUSION AND FUTURE WORK

This paper has addressed the need for QoS provisioning mechanisms for the MEC Application slice concept in ETSI MEC. Considering the management and operation view of MEC App slices in the standard, two QoS provisioning mechanisms are proposed to be integrated in the MEC architecture: (i) QoS-aware policies, which can leverage the QoS class, Pod Priority & preemption, and bandwidth shaping features of Kubernetes, and (ii) QoS-aware services enabled by the MEC platform (e.g., load balancer). The QoS-aware policies for MEC App slices have been validated empirically through an edge-enabled 5G testbed, on which two services and their MEC App slices have been deployed under different radio conditions. Results have showcased the trade-offs of the different applied QoS-aware policies. It has been shown that flexible configurations with minimum resource guarantees to run the MEC Applications and without strict resource limits are the ones that provide higher availability of the MEC App slices with enhanced QoE.

As part of the future work, we envisage the validation of QoS-aware services in the implemented testbed, as well as assessing the benefits of using both mechanisms together. Moreover, motivated by experimental results presented in this work, we intend to study the coordination between the QoS

provisioning mechanisms for network slices and for MEC App slices, considering both the radio and the edge domains.

ACKNOWLEDGMENT

This work was supported in part by the TRAINER-6G project (PID2023-146748OB-I00) funded by MCIN/AEI/10.13039/501100011033 and by ERDF/EU, and in part by the EU's HE research and innovation program HORIZON-JU-SNS-2023 under the 6G-PATH project (Grant No. 101139172). The authors would like to acknowledge Gerard Carreras and Aleix Teruel for their assistance with the testbed setup and experiments.

REFERENCES

- [1] Ericsson, "Ericsson Mobility Report", November, 2025.
- [2] P. Yang et al., "6G Wireless Communications: Vision and Potential Techniques," in *IEEE Netw.*, vol. 33, pp. 70-75, July 2019.
- [3] ITU-R, "Framework and objectives of future development of IMT for 2030 and beyond", ITU-R M.2160, November 2023.
- [4] M. Caprolu, et al., "Edge Computing Perspectives: Architectures, Technologies, and Open Security Issues," 2019 *IEEE International Conference On Edge Computing (EDGE)*, Milan, Italy, 2019.
- [5] N. Abbas, et. al, "Mobile Edge Computing: A Survey," in *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450-465, February 2018.
- [6] N. Sprecher et al, "Harmonizing standards for edge computing," *ETSI White paper #36*, France, 2020.
- [7] AI-RAN Alliance, "AI-RAN Alliance Vision and Mission White Paper," White Paper, February 2024.
- [8] AI-RAN Alliance, "AI-RAN Architecture Overview and Component Definitions," White Paper, V. 1.2, June 2024.
- [9] F. Debbabi, et al., "Algorithmics and Modeling Aspects of Network Slicing in 5G and Beyonds Network: Survey," *IEEE Access*, vol. 8, pp. 162748-162762, 2020.
- [10] 3GPP TS 23.501 v19.0.0 "5G; System architecture for the 5G System (5GS)" (Release 19).
- [11] ETSI GR MEC 044 v3.1.1 "Study on MEC Application Slices", September 2024.
- [12] H. Zhu, et al., "MEC Application Slice and its collaboration with 5G network slice," 2022 *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, Bilbao, Spain, 2022.
- [13] S. Boutalbi, et al., "Mobile Edge Slice Broker: Mobile Edge Slices Deployment in Multi-Cloud Environments," 2023 *IEEE 7th International Conference on Fog and Edge Computing (ICFEC)*, India, 2023.
- [14] S. Arora, et al, "Lightweight edge Slice Orchestration Framework," *ICC 2022 - IEEE International Conference on Communications*, Seoul, Republic of Korea, 2022.
- [15] S. Bolettieri, et al, "Towards end-to-end application slicing in Multi-access Edge Computing systems: Architecture discussion and proof-of-concept", in *Future Generation Computer Systems*, vol. 136, pp. 110-127, November 2022.
- [16] Y. C. Chan, et al., "MEC Implementation in NFV Architecture Supporting 5G End-to-End Network Slicing," 2023 *2nd International Conference on 6G Networking (6GNet)*, Paris, France, 2023.
- [17] ETSI GR MEC 038 v3.1.1 "MEC in Park Enterprises deployment scenario", November 2022.
- [18] V. Kjørveziroski, C. Bernad, K. Gilly, S. Filiposka, "Implementing Multi-Access Edge Computing with Kubernetes," in *ICT Innovations 2022*.
- [19] Amarisoft, AMARI Callbox Classic, Available at: <https://www.amarisoft.com/test-and-measurement/device-testing/device-products/amari-callbox-classic>.
- [20] Minikube, "Welcome!". Available at: <https://minikube.sigs.k8s.io/docs/>
- [21] Minecraft. "Minecraft". Available at: <https://www.minecraft.net>.
- [22] VideoLAN, "VLC media player". Available at: <https://images.videolan.org/vlc/index.an.html>.