Computer Networks 240 (2024) 110168

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Joint multi-objective MEH selection and traffic path computation in 5G-MEC systems☆

Prachi Vinod Wadatkar [a,b], Rosario Giuseppe Garroppo [b], Gianfranco Nencioni [a,*], Marco Volpi [b]

[a] University of Stavanger, Stavanger, 4021, Norway
[b] University of Pisa, Pisa, 56126, Italy

ARTICLE INFO

Keywords:
5G
Multi-access Edge Computing
Multi-objective Dijkstra Algorithm
Pareto-optimal
MEH selection
Traffic path computation
AdvantEDGE

ABSTRACT

Multi-access Edge Computing (MEC) is an emerging technology that allows to reduce the service latency and traffic congestion and to enable cloud offloading and context awareness. MEC consists in deploying computing devices, called MEC Hosts (MEHs), close to the user. Given the mobility of the user, several problems rise. The first problem is to select a MEH to run the service requested by the user. Another problem is to select the path to steer the traffic from the user to the selected MEH. The paper jointly addresses these two problems. First, the paper proposes a procedure to create a graph that is able to capture both network-layer and application-layer performance. Then, the proposed graph is used to apply the Multi-objective Dijkstra Algorithm (MDA), a technique used for multi-objective optimization problems, in order to find solutions to the addressed problems by simultaneously considering different performance metrics and constraints. To evaluate the performance of MDA, the paper implements a testbed based on AdvantEDGE and Kubernetes to migrate a VideoLAN application between two MEHs. A controller has been realized to integrate MDA with the 5G-MEC system in the testbed. The results show that MDA is able to perform the migration with a limited impact on the network performance and user experience. The lack of migration would instead lead to a severe reduction of the user experience.

1. Introduction

Some of the use cases defined for 5G-and-beyond systems [1,2] require high performance, in terms of latency, reliability, throughput, etc. [3]. Multi-access Edge Computing (MEC) is one of the fundamental technologies committed to satisfying these requirements. MEC consists in the computing platforms located near the users. MEC deployments in proximity to users often demand additional computing resources and infrastructure to ensure optimal performance. This additional effort is largely compensated by the achieved performance (not only low latency, but also reduction of network congestion and increased data rate) and the enabling of new advanced services (which can also rely of context awareness). As described in [4], the main functions of 5G and MEC architecture interact as shown in Fig. 1.

In particular, the user requests an application to the Application Function (AF). The MEC may be seen as an AF, and if it is trusted it may directly interact with the 5G Network Functions (NFs). In the case of untrusted MEC (as a general untrusted AF), the interaction MEC-5G system is mediated by the Network Exposure Function (NEF) that is in charge of securely exposing the network capabilities and events to untrusted AFs. A key element of the MEC system level is the MEC Orchestrator (MEO), which provides centralized functions for orchestrating the computing resources and operation of the MEC hosts (MEHs). On one side, the MEO interacts with the different MEHs to acquire information on the available resources and Apps. On the other side, MEO interacts with the NEF (or directly with the 5G NFs), to acquire capability and state information about the 5G Core Network (CN) services. The MEO uses these data to select the MEH for instantiating a new application request (or migrating a running application) by considering the performance requirements (e.g., latency, throughput, packet loss, etc.), the MEH available resources, and the MEH and network performance.

During the service, through the NEF (or directly interrogating the 5G NFs in case of trusted MEC), the MEO acquires UE-related events of interest to decide if some actions (e.g. change the access point, modify the traffic path) on the 5G domain needs to be performed to satisfy the required QoS. On the MEC infrastructure, the MEO monitors the performance given by the MEH. Problems in the network domain or in

☆ This work was supported by the Norwegian Research Council through the 5G-MODaNeI project (no. 308909), and by the Italian Ministry of Education and Research (MIUR) in the framework of the FoReLab project (Departments of Excellence).

* Corresponding author.
E-mail address: gianfranco.nencioni@uis.no (G. Nencioni).

https://doi.org/10.1016/j.comnet.2023.110168
Received 10 July 2023; Received in revised form 18 November 2023; Accepted 27 December 2023
Available online 3 January 2024

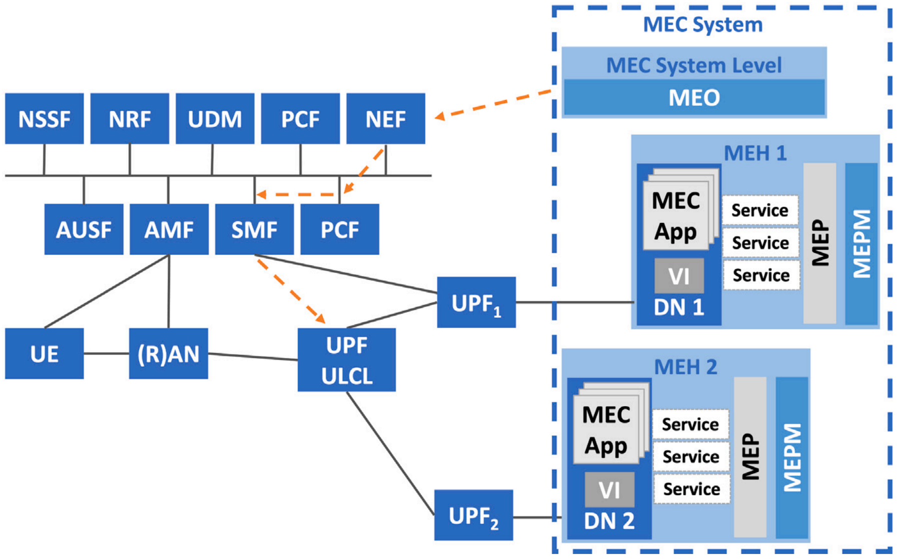


Fig. 1. Interaction 5G-MEC functions for supporting services.

the MEC resources can lead the MEO to modify the traffic steering rules and/or migrate the service to another MEH.

Intuitively, to maintain high performance, a user request should be served by the nearest MEH that runs the required application. Empirical studies confirm this intuition [5]. Maintaining application proximity for mobile users poses significant challenges. Among these, two fundamental challenges regarding the achievement of a desirable trade-off between QoS and migration cost are related to the questions: (1) How are the applications migrated [6,7]? and (2) When/where are the applications migrated [8]? This paper considers this second question by analyzing two important problems: the dynamic selection of the MEH providing the resources for running the application requested by a UE (*MEH selection*) and the computation of the traffic path between the selected MEH and the UE (*traffic path computation*). Both problems aim to maintain the requested QoS as the UE moves. Indeed, a major challenge in 5G MEC is related to UE mobility. To ensure the desired QoS parameters, including low latency and minimal packet loss, it may become necessary to modify the traffic path and, in certain situations, switch the MEH, leading to what is known as an MEH handover. This process can be due to the deterioration of performance provided by the MEH and/or the traffic path leading to the source MEH. In the latter scenario, performance degradation may arise from handovers between different base stations or an increase in traffic load along this path.

1.1. Problem definition and contribution

The paper jointly considers the *MEH selection* problem and the *traffic path computation* problem in 5G-MEC systems by defining a multi-objective optimization problem. The heterogeneity of the 5G-MEC applications leads to defining new control algorithms aimed to consider different criteria simultaneously. For example, some applications require the minimization of the latency, while others require minimizing packet loss, increasing the security of the data transfer and of the elements supporting the application, minimizing the energy consumption, etc.. Hence, to account for most requirements the optimization problems cannot consider a single criterion. The complexity of the multi-objective optimization problem is obviously higher than the single criterion one. For example, for the routing problem, algorithms, such as Dijkstra, allow the computation of the single criterion minimum cost path in an efficient way. In contrast, different studies have been focused on finding efficient approaches for solving the complex multi-objective counterpart [9]. The most common approach is to define an optimization function which is a weighted combination of all criteria. In this manner, the multi-criteria problem can be treated as a single-criterion problem, i.e. with reduced complexity. However, this approach implies that the weights need to be established a priori. Before the problem solution begins, the relative "importance" of the different criteria needs to be set. The solution is optimal only for the selected weighting. The heterogeneity of the applications of today networks does not allow finding a weighting providing good results for all.

The multi-criteria optimization problem gives a set of Pareto optimal solutions (the so-called Pareto front), which can be dynamically selected depending on the requirements of the particular application. The MEO determines the most suited solution for the application needs, analyzing the Pareto front. Then, MEO decides where to locate (or re-locate) the application and how to steer the traffic. Although multi-criteria optimization is NP-hard, recent works proposed efficient algorithms for typical-size networks that work well in practice.

The main contribution can be summarized in the following points.

- To jointly consider the problems of MEH selection and traffic path computation, the paper proposes a procedure for defining a graph able to account for both the network-layer and application-layer performance. In this manner, the two problems are mapped in the single problem of finding optimized paths between the user and the MEHs.
- To solve the problems using the defined graph, the paper proposes the Multi-objective Dijkstra Algorithm (MDA) for computing the Pareto front of the Multi-Objective Shortest Path (MOSP) model. Furthermore, the paper shows how the Pareto front is used for supporting applications with different requirements.
- To evaluate the performance of MDA, the paper implements a hybrid testbed. The testbed includes simulative, emulative and experimental parts.
- To integrate the MDA with the 5G-MEC system in the testbed, the paper realizes a controller to retrieve the input of MDA and to apply the output of MDA for both network layer and application layer.

The paper is organized as follows. Section 2 summarizes the MOSP problem and the MDA algorithm, while Section 3 describes the proposed approach for jointly solving the MEH selection problem and the traffic path computation problem, using the MDA algorithm. Sections 4 and 5 introduce extensive evaluations in an emulative/experimental environment. Section 6 presents the related works and summarizes the novelties of this paper. Finally, Section 7 concludes the paper.

Table 1
List of used symbols.

Symbols	Definition
$G = (V, A)$	Directed graph without parallel arcs, composed by the set V of nodes and the set A of arcs
$N = \|V\|$	Number of nodes
$M = \|A\|$	Number of arcs
d	Number of attributes (e.g. delay, energy consumption, jitter, packet loss, etc.) considered by the MOSP
c_a	Vector of values assumed by the attributes in the arc a, $c_a(c_{a1}, c_{a2}, \dots, c_{ad})$
$\delta^+(v)$	Set of outgoing arcs in v
$\delta^{--}(v)$	Set of incoming arcs in v
(v_1, v_k)-path, $P_{(v_1,v_k)}$	Set of arcs necessary to reach node $v_k \in G$ from node $v_1 \in G$. It can be represented by the sequence of traversed nodes, i.e. $P_{(v_1,v_k)} = (v_1, v_2, \dots, v_k)$
$\mathbf{c}_{(v_i,v_{i+1})}$	Cost vector related to the arc $(v_i, v_{i+1}) \in G$
$\mathbf{c}(P_{(v_1,v_k)})$	Cost of the (v_1, v_k)-path. In the case of summable attributes, it can be calculated as $\mathbf{c}(P_{(v_1,v_k)}) = \sum_{i=1}^{k-1} \mathbf{c}_{(v_i,v_{i+1})}$
$P <_D Q$	Path P dominates path Q

2. MOSP problem and MDA algorithm

This section summarizes the key definitions and features of the MOSP problem and of the MDA algorithm.

2.1. MOSP problem

The technical details on the MOSP problem definition and on the MDA algorithm can be found in [10]. The definitions of the different symbols used in this summary are shown in Table 1.

The solution of the MOSP problem requires the definition of the concept dominated-path, $P_{(v_1,v_k)}$.

Let us consider two paths $P_{(s,t)}$ and $Q_{(s,t)}$ with the same origin node s and the same destination node t. Let $c_j(P)$ and $c_j(Q)$ the jth component of the cost vector of the two paths. Then $P <_D Q$

$$\begin{aligned} &If \quad c_j(P) \le c_j(Q), \quad \forall j \in [1,d] \\ &\exists i \in [1,d] \quad : \quad c_i(P) < c_i(Q) \end{aligned} \tag{1}$$

It is worth noting that (1) considers only the cost vectors of the two paths. Hence, in general given two cost vectors, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, (1) establishes the dominance between two cost vectors, i.e. if $\boldsymbol{\alpha} <_D \boldsymbol{\beta}$.

Given a (s,t)-path P with cost vector $\mathbf{c}(P)$, $\mathbf{c}(P)$ is called a *non-dominated vector* if there is no other (s,t)-path having a cost vector dominating $\mathbf{c}(P)$. In this case, P is denoted as *efficient path*. There is no (s,t)-path in G that dominates P.

The MOSP problem aims at finding a path optimizing simultaneously different attributes, which often are conflicting. An improvement in one objective can lead to the worsening of at least one other objective. In this scenario, the solution of the MOSP problem is represented by the minimum complete set of efficient paths between a node s and every node $v \in V$. For each node pair (s,v), this set contains exactly one efficient path per non-dominated cost vector. This definition refers to the so-called one-to-all variant. If a target node t is given as input, the one-to-one variant of MOSP aims at finding the minimum complete set of efficient paths connecting the nodes s and t.

2.2. MDA algorithm

The MDA is a recent label-setting algorithm [10] for solving the MOSP problem. A more recent paper presents further improvements aimed at reducing the solving time of the one-to-one version of MOSP [11]. The performance comparison among various one-to-many MOSP algorithms demonstrates the superiority of MDA, particularly on graphs comprising approximately 300,000 nodes and 800,000 links [12]. Notably, these graph sizes are often larger than real-world networks. Like other label-setting algorithms, the MOSP uniquely represents a path P as a sequence of node labels. For each node v, a label is composed by three values $l_v = (v, \mathbf{c}_{l_v}, l_{pred})$, where:

- v is the identity of the node owner of the label
- $\mathbf{c}_{l_v}$ is the cost vector of the (s,v)-path
- l_{pred} is the pointer to a label of a predecessor node $u \in \delta^-(v)$. This label refers to the (s,u)-subpath of the (s,v)-path.

The defined labels can be compared to each other. The labels comparison and the one-to-one correspondence between paths and labels allow finding the efficient paths analyzing the labels. In other words, the efficient paths can be found through the list of non-dominated labels of the nodes. The definition of non-dominated labels is as follows.

Given two labels, $l_1(v)$ and $l_2(v)$ corresponding to two alternative (s,v)-paths P_1 and P_2

- $l_1(v)$ dominates $l_2(v)$, indicated as $l_1(v) \le_D l_2(v)$, and both are non-equivalent if and only if $\mathbf{c}_{l_1(v)} <_D \mathbf{c}_{l_2(v)}$ and $\mathbf{c}_{l_1(v)} \ne \mathbf{c}_{l_2(v)}$

For each node v, the MOSP commonly leads to having a set of labels, indicated as L_v, that are non-dominated. To compare a new label $l_{new}(v)$ with L_v the following definition is necessary.

- L_v dominates $l_{new}(v)$ or $L_v \le_D l_{new}(v)$ iff there is a label $l_\alpha \in L_v$ s.t. $l_\alpha \le_D l_{new}(v)$.

The lexicographic order of labels is defined as follows. Let $l_1(v)$ and $l_2(v)$ be two labels corresponding to two alternative (s,v)-paths. The label $l_1(v)$ is lexicographically smaller than $l_2(v)$, denoted as $l_1(v) <_{lex} l_2(v)$, iff $\mathbf{c}_{l_1(v)}$ is lexicographically smaller than $\mathbf{c}_{l_2(v)}$, and this is true if $\mathbf{c}_{l_{1,k}(v)} < \mathbf{c}_{l_{2,k}(v)}$ for the first index $k \in \{1, \dots, d\}$ such that $\mathbf{c}_{l_{1,k}(v)} \ne \mathbf{c}_{l_{2,k}(v)}$.

Algorithm description

In the following, a brief description of the MDA algorithm is presented. Further details can be found in [10]. The algorithm defines the following set of lists and vectors:

- H: List storing the tentative labels in lexicographical order. The tentative labels correspond to paths explored during the algorithm but for which it is not yet decided if they are non-dominated paths. At any point during the algorithm, H stores at most one label per node, i.e. the size of H is bounded N.
- L_v: For each node $v \in V$, a list L_v contains the non-dominated labels.
- **lastProcessedLabel**: This vector contains the pointer to the last processed label for each arc in the graph.

Algorithm 1 shows the pseudocode of MDA. The initialization phase sets up the empty list H and the N empty lists L_v. The L_v lists are grouped into a list L. Then, there is the initialization of the vector $lastProcessedLabel$ composed of M values equal to zero. Then the algorithm generates a label for the origin node s which is $l_s = (s, (0, \dots, 0), NULL)$ and inserts them in the list H.

Algorithm 1 Pseudocode of MDA

```
1: Input: Graph $G = (V, A)$ with cost vectors $\mathbf{c}_(v_i, v_(i+1)) \in \mathbb{R}^d$, Node $s \in V$
2: Output: set $L_v$ of non-dominated labels $\forall v \in V$
3: Priority Queue $H \leftarrow 0$
4: for $v \in V$ do
5:     Efficient Labels $L_v \leftarrow 0$
6: end for
7: $L \leftarrow \bigcup_{v \in V} L_v$
8: for $a \in A$ do
9:     lastProcessedLabel$[a] \leftarrow 0$
10: end for
11: Label $l_s \leftarrow [s, (0, \cdots, 0)$ NULL$]$
12: $H \leftarrow H.insert(l_s)$
13: while $H \neq 0$ do
14:     $l_v^* \leftarrow H.extract\_lexmin()$
15:     $v \leftarrow l_v^*.node$
16:     $L_v.push\_back(l_v^*)$
17:     $l_v^{new} \leftarrow$ nextCandidateLabel($v$,lastProcessedLabel, $\delta^-(v), L$)
18:     if $l_v^{new} \neq$ NULL then
19:         $H.insert(l_v^{new})$
20:     end if
21:     for $w \in \delta^+(v)$ do
22:         $H \leftarrow$ propagate($l_v^*, w, H, L$)
23:     end for
24: end while
25: return $L_v$ for all $v \in V$
```

After this phase, the `while` loop begins and lasts until the list H becomes empty. An iteration of the `while` loop starts with the extraction of the lexicographical smallest label l_v^* from the list H, where v is the node associated with the label. It is worth recalling that in H the labels are lexicographically sorted, then the label l_v^* is surely non-dominated. Thus the label can be added to the end of the list of non-dominated labels of the node v, i.e. L_v. This is the only way a label is added to the list L_v, i.e. the label gets permanent. As a consequence, the sets L_v $v \in V$ are also lexicographically sorted. Each iteration carries out two main tasks. The first is to find the next *tentative/candidate* label for node v that can be added to H. This task is necessary because only one label per node v is present in H. Hence, when this label is extracted, a new tentative label for v must be found (if it exists) and added to H. The search of the new label, l_v^{new}, is performed extending existing non-dominated labels at the predecessor node $u \in \delta^-(v)$ along the arc (u,v). In particular, l_v^{new} must be lexicographically the smallest and the non-dominated one among the all possible extension, i.e.

$$l_v^{new} = arglexmin_{l,u}\{lv = (v, c_l + c_{uv}, l) | Lv \not\prec_D l_v\}$$

where $l \in L_u$ and $u \in \delta^-(v)$.

This part allows maintaining a single tentative label for each node, which represents an important characteristic that differentiates the MDA from the classical label-setting MOSP algorithms, which keep a set of tentative labels for the same node.

The second task is to propagate the extracted l_v^* to the successor nodes $w \in \delta^-(v)$. Let $l_w = (w, c_{l_v^*} + c_{vw}, l_v^*)$ be such a propagated tentative label. If l_w is dominated by any label in L_w, it is discarded. Otherwise, if there is no label for w in H, l_w is inserted. On the contrary if there is a label, a comparison between l_w and the label of w already present in H is performed. The lexicographically smaller will be in H, the other will be discarded. The details and the pseudocode of these two tasks can be found in [10].

The running time performance of MDA is deeply analyzed in [10] with large synthetic and real-world graph instances. In the one-to-all case, the results show that for $d \geqslant 3$ the running time is $O(d \cdot (\log(N) \cdot \sum_{v \in V} |L_v| + M \cdot (\max_{v \in V} |L_v|)^2))$. To the best of the authors' knowledge, MDA is the fastest among any MOSP algorithm known so far.

3. Proposed multi-layer graph

To jointly address the MEH selection problem and the traffic path computation problem, the approach of incorporating information on the MEH performance at the application layer in a graph is proposed. In general, the two problems are considered separately. As an example, the graph considering only the network-layer information is built taking into account the network topology. Routing algorithms, such as MDA, can be used to compute the traffic path from the user to the chosen MEH supporting the requested application. Before starting the traffic path computation, the MEH needs to be selected. The MEH selection can be done using algorithms such as [13]. The problem with this disjoint approach is that the algorithm used for the MEH selection considers only aspects related to the MEH platform, such as the constraints on the computation and communication capacity. The goal of the algorithm is to maximize the number of served requests while assuring that the service placement cost is within the budget. The quality of the service experienced by the application user is not taken into account. Indeed, bad quality is experienced if there is no traffic path between UE and selected MEH able to guarantee the network performance required by the application.

The proposed approach is based on the idea of enhancing the network graph by incorporating MEH performance information. For each MEH, a node and an arc are introduced to the graph. For each MEH, a node and an arc are introduced to the network graph. The added node represents the application layer of the MEH, while the arc captures the application layer performance of the MEH in processing data exchanged with the network layer. The attributes of this arc reflect the MEH performance from the perspective of the running application and can be obtained through monitoring tools implemented within the MEH.

The remaining part of the graph considers the available alternative network paths between the client and each MEH supporting the required application. The values of the attributes of each arc of the graph are estimated on each link of the network. In some cases, such as on the wireless network interface, the metrics are estimated from the values reported in some APIs defined by the MEC architecture.

Fig. 2 shows an example of the graph obtained when the user (node UE) asks for a service that can be offered by four different MEHs located in nodes 5, 6, 7 and 8. In each of these nodes, the MEH is represented by the extra node in red (i.e. MEH5A, MEH6A, MEH7A and MEH8A). The added extra arcs representing the MEH performance at the application layer are in red. The UE has three alternative Points of Access (PoAs) for accessing the network: PoA1, PoA2 and PoA3. The black arcs of the graph refer to the network layer. The vector of each arc gives the values of the considered attributes. In the example, the selected attributes are respectively the packet loss probability, the jitter and the latency. However, other attributes can be used, such as security, energy consumption, etc.

This graph serves as the input to the MDA, which generates the Pareto front of paths from the client to each candidate MEH, considering the application-level performance. The output of the MDA is utilized to select the MEH and determine the corresponding path.

3.1. Metrics

There are different algorithms for finding the entire set of Pareto-optimal paths in the graph presented above. These algorithms, such as MDA, are designed for sum and bottleneck-type metrics. In the case of bottleneck-type metrics, a simple strategy is to prune from the graph the arcs that do not satisfy the constraints on the bottleneck metric. For example, if the service requires minimum datarate, the simple

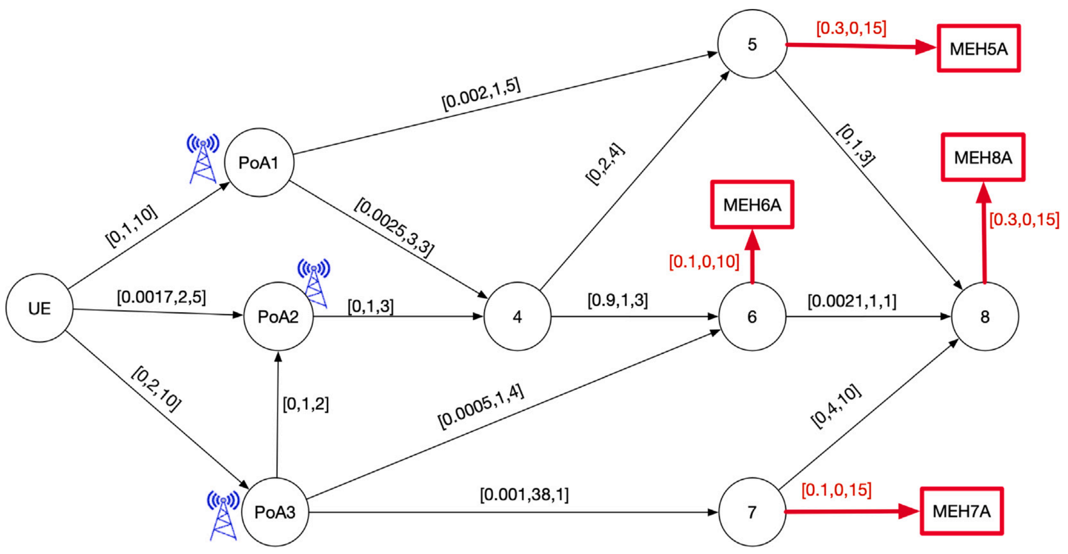


Fig. 2. An example of an extended graph. The nodes labeled as MEH5A, MEH6A, MEH7A, and MEH8A represent different MEHs, while the nodes PoA1, PoA2, and PoA3 represent the available PoAs for the user UE. Node 4 is exclusively utilized for traffic routing, while the remaining nodes, apart from traffic routing, fulfill the functions of the MEH. Each link is associated with a cost vector denoting packet loss probability, jitter, and latency. The black arcs represent network-layer links, while the red arcs indicate internal links connecting the application layer of the MEH with the network layer. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

”pruning” of the arcs non-satisfying this constraint can be applied to the original graph before running the algorithm. Furthermore, information on the maximum datarate given by a particular path can be obtained considering that the maximum datarate offered by a path corresponds to the lowest datarate observed in its arcs. In [14], the authors suggest an alternative approach to deal with a bottleneck-type metric: they suggest converting it into a sum-type by using reciprocals. For example, in the case of the available datarate, this bottleneck-type metric can be converted into a sum-type defining the optimal goal as $f^p = \sum_{a\in A_p}(1/datarate(a))$, where p is a path and A_p is the set of arcs of p. Referring to the presented example, it is important to note that latency is a sum-type metric, while the other two metrics, namely packet loss and jitter, require additional assumptions and manipulation.

Regarding packet loss, it is necessary to assume the independence of the loss process across consecutive arcs and, more generally, across diverse arcs. The packet loss probability P_{loss} on a path L, composed by two consecutive arcs (s,u) and (u,v) with independent packets losses, can be computed by means of the probability of the complementary event “correct packet delivery” (CPD), i.e. $P_{loss} = (1 - P_{CPD})$. The independence of the packet loss on the links (s,u) and (u,v) composing the path L leads that the CPD probability on L is given by the product of the CPD probability on both links of the path, i.e. $P_{CPD_L} = P_{CPD_{s,u}} P_{CPD_{u,v}}$. Hence, P_{loss} of the path L can be calculated with the relation $P_{loss_L} = 1 - P_{CPD_{s,u}} P_{CPD_{u,v}}$.

This result can be easily generalized to a generic (s,d)-path P composed by k arcs:

$$P_{loss_P} = 1 - \prod_{i=1}^{k-1} P_{CPD_i}. \tag{2}$$

The minimization of P_{loss_P} can be found maximizing $\prod_{i=1}^{k-1} P_{CPD_i}$ or, in other words, minimizing $(-\prod_{i=1}^{k-1} P_{CPD_i})$. Using the logarithms property: $-\log(\prod_{i=1}^{k-1} P_{CPD_i}) = -\sum_{i=1}^{k} \log_{10}(P_{CPD_i})$. Hence, in the case of the packet loss attribute, the sum-type metric for the arc i is represented by $-\log_{10} P_{CPD_i}$. Using this metric for each link of the graph, the MDA will find the path with the minimum value of $-\sum_{i=1}^{k} \log_{10}(P_{CPD_i})$, which corresponds to the path with the minimum P_{loss}.

Two important assumptions are required when considering jitter. Firstly, it is assumed that jitter is defined as the variance of latency. Secondly, the latency in each link is modeled as a Gaussian random process, and the processes associated with different links are assumed to be independent. These assumptions enable the estimation of jitter as the variance of the Gaussian model of latency in each link. Furthermore, by assuming the independence of the processes modeling latency in diverse arcs, the jitter can be transformed into a sum-type metric. Indeed, these assumptions lead to calculating the jitter of a path L, composed by two consecutive independent arcs (s,u) and (u,v), by means of the sum of the jitter of each arc:

$$jitter_L = jitter_{s,u} + jitter_{u,v}. \tag{3}$$

3.2. Graph without MEH state information

This case refers to the disjoint approach, wherein the selection of the MEH is determined using dedicated algorithms, while the MDA is solely utilized for defining the traffic path between the user and the chosen MEH. Referring to the example of Fig. 2, MDA receives as input the subgraph $G = (V, A)$ derived from the graph in the figure after removing all the red arcs and red nodes, which correspond to the application layer information.

Let us assume that the dedicated algorithm has selected the MEH in node 8. In this scenario, the MDA will output the set of all non-dominated paths between the UE and 8. In general, it provides all non-dominated paths between UE and any node of the graph. In each node, the labels related to the alternative non-dominated paths are lexicographically ordered. The output is presented in Table 2: each row represents the list of non-dominated labels between the UE and each node. For sake of clarity, each label includes the node owner, the cost vector, and the link to the previous node. The link is represented as a pair, where the first element is the identifier of the previous node and the second is the identifier of the label in the previous node. This information is necessary because each node can have multiple non-dominated labels, and each label may correspond to a different path. It is worth noting that in the destination node 8, the MDA provides 6 distinct non-dominated labels. Therefore, selecting one path from the non-dominated set requires defining a strategy that considers the application requirements. For example, when the strategy aims to minimize P_{loss}, the path UE-PoA3-7-8 is selected. On the other hand, if the goal is to achieve minimum latency with $P_{loss} \leq 0.01$, the available paths are UE-PoA3-4-5-8 (with lower P_{loss}) and UE-PoA3-6-8 (with lower jitter).

Table 2
Output of the MDA algorithm for the example "Graph without MEH State Information".

Node	Non-dominated Labels
UE	[**UE**, [0, 0, 0], ['NULL', 'NULL']]
PoA1	[**UE**, [0.0, 1, 10], [**UE**, 1]]
PoA2	[**PoA2**, [0.0, 3, 12], [**PoA3**, 1]], [**PoA2**, [0.0017, 2, 5], [**UE**, 1]]
PoA3	[**PoA3**, [0.0, 2, 10], [**UE**, 1]]
4	[**4**, [0.0, 4, 15], [**PoA2**, 1]], [**4**, [0.0017, 3, 8], [**PoA2**, 2]]]
5	[**5**, [0.0, 6, 19], [**4**, 1]], [**5**, [0.0017, 5, 12], [**4**, 2]], [**5**, [0.002, 2, 15], [**PoA1**, 1]]
6	[**6**, [0.0005, 3, 14], [**PoA3**, 1]], [**6**, [0.9002, 4, 11], [**4**, 2]]
7	[**7**, [0.001, 40, 11], [**PoA3**, 1]]
8	[**8**, [0.001, 44, 21], [**7**, 1]], [**8**, [0.0017, 6, 15], [**5**, 2]], [**8**, [0.002, 3, 18], [**5**, 3]], [**8**, [0.0026, 4, 15], [**6**, 1]], [**8**, [0.9004, 5, 12], [**6**, 2]]

Table 3
Output of the MDA algorithm for the example "Graph without MEH State Information" with the constraints $P_{loss} \leq 0.002$ and $jitter \leq 40$ ms.

Node	Non-dominated Labels
UE	[**UE**, [0, 0, 0], ['NULL', 'NULL']]
PoA1	[**UE**, [0.0, 1, 10], [**UE**, 1]]
PoA2	[**PoA2**, [0.0, 3, 12], [**PoA3**, 1]], [**PoA2**, [0.0017, 2, 5], [**UE**, 1]]
PoA3	[**PoA3**, [0.0, 2, 10], [**UE**, 1]]
4	[**4**, [0.0, 4, 15], [**PoA2**, 1]], [**4**, [0.0017, 3, 8], [**PoA2**, 2]]], [**4**, [0.002, 2, 15], [**PoA1**, 1]]
5	[**5**, [0.0, 6, 19], [**4**, 1]], [**5**, [0.0017, 5, 12], [**4**, 2]], [**5**, [0.002, 4, 19], [**4**, 3]]
6	[**6**, [0.0005, 3, 14], [**PoA3**, 1]]
7	[**7**, [0.001, 40, 11], [**PoA3**, 1]]
8	[**8**, [0.0017, 6, 15], [**5**, 2]], [**8**, [0.002, 5, 22], [**5**, 3]]

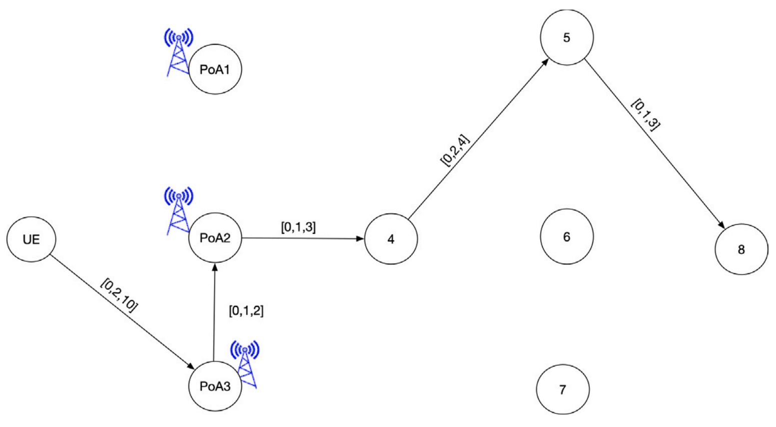


Fig. 3. Best $(UE, 8)$-path with minimum P_{loss} chosen among the set of Table 3.

The upper bounds of some attributes, such as P_{loss} or jitter, can be directly considered in the MDA. The label-setting procedure can be modified by eliminating labels that do not meet the constraints on the upper bounds. For instance, let us consider the case of an application that requires $P_{loss} \leq 0.002$ and $jitter \leq 40$ ms. The modified MDA gives the output shown in Table 3. This revised output contains a reduced number of non-dominated labels due to the inclusion of upper bounds on P_{loss} and jitter. The path with the minimum P_{loss} obtained from this table is shown in Fig. 3. Furthermore, among this set, for instance, the path with the minimum latency is **UE-PoA2-4-5-8**.

3.3. Graph with MEH state information

In the second scenario, the MDA is employed on the entire graph depicted in Fig. 2, which includes the information regarding the performance of MEHs at the application layer. The output of the MDA consists of the collection of all non-dominated paths between the user (represented by node UE) and the MEHs that can support the desired application, namely nodes MEH5A, MEH6A, MEH7A, and MEH8A. For each candidate MEH, the cost vector is derived by augmenting the non-dominated set, as presented in Table 3, with the additional costs associated with the application layer performance (represented by the red links in Fig. 2). The outcomes of this process are illustrated in Table 4.

Depending on the specific application requirements, the selection of the MEH considers various parameters. For instance, in the case of an application prioritizing minimum P_{loss}, the chosen MEH is MEH6A. The $(UE, MEH6A)$-path selected from the non-dominated set provided by MDA is depicted in Fig. 4. In the scenario where the application prioritizes minimum latency while also favoring low P_{loss}, the selected MEH remains MEH6A.

4. Experimental performance evaluation

This section thoroughly presents the experimental setup of the testbed. This emphasis is motivated by the need to ensure transparency and reproducibility of our work. Moreover, the experimental setup is instrumental in elucidating the pivotal role of the proposed controller, which stands as one of the key contributions of this paper. To evaluate the developed MDA-based controller, a hybrid (simulative-emulative-experimental) testbed based on AdvantEDGE has been implemented.

AdvantEDGE platform provides an emulated and experimental environment with edge-enabling technologies [15]. The platform runs on Docker and K8s, and provides experimentation with MEC deployment

Table 4
Cost and non-dominated paths for each MEH - Case “Graph with MEH State Information” with the constraints $P_{loss} \leq 0.002$ and $jitter \leq 40$ ms.

MEH	Non-dominated Labels to achieve the network layer of the MEH
MEH5A	[**5**, [0.3, 6, 34], [**4**, 1]], [**5**, [0.3017, 5, 27], [**4**, 2]], [**5**, [0.302, 4, 34], [**4**, 3]]
MEH6A	[**6**, [0.1005, 3, 24], [**PoA3**, 1]]
MEH7A	[**7**, [0.101, 40, 26], [**PoA3**, 1]]
MEH8A	[**8**, [0.3017, 6, 30], [**5**, 2]], [**8**, [0.302, 5, 37], [**5**, 3]]

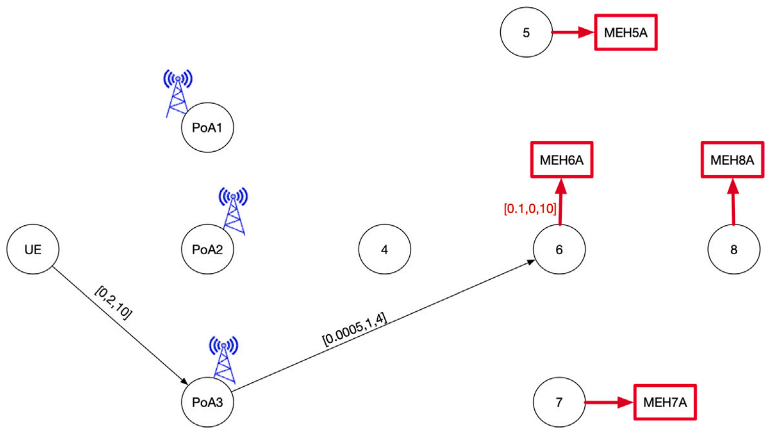


Fig. 4. The $(UE, MEH6A)$-path derived from the graph shown in Fig. 2, while adhering to the constraints $P_{loss} \leq 0.002$ and $jitter \leq 40$ ms. The path with lowest P_{loss} is chosen from the set of non-dominated paths shown in Table 4.

models along with their applications and services. The emulation platform supports several standardized APIs and edge services standardized by the ETSI MEC, including ETSI MEC 013 Location [16], ETSI MEC 012 Radio Network Information [17], ETSI MEC 028 WLAN Information [18], ETSI MEC 011 Edge Platform Application Enablement [19], and ETSI MEC 021 Application Mobility [20]. AdvantEDGE allows the mobility of the UEs within the network by using its own APIs to evaluate the impact on the application performance. The platform allows for configuring the network layer performance in the 5G-MEC architecture through the AdvantEDGE API, which includes settings for latency, jitter, throughput, and packet loss for each link in the emulated network scenario. The platform allows mobility events, UE movement and mapping of the geo-location of each element. The UE movement can be monitored and visualized by using the Geospatial Subsystem. Furthermore, AdvantEDGE supports the inclusion of MEHs that are not emulated but running on separate devices.

To address performance at the application layer, the scenario incorporates MEHs capable of running and migrating applications as needed. These MEHs form a unified cluster of K8s nodes. Transport-related data is obtained from the AdvantEDGE platform, while application-related information is acquired from the MEC. Interaction with AdvantEDGE allows for the retrieval of transport information, primarily utilizing the location API [16]. During the experimentation phase, the location API is leveraged to track the physical location of UEs within the network, acquiring the necessary graph information as input for MDA. In the presented experiments, the VLC application [21] is utilized, where the VLC client operates on the UE, while the VLC server is hosted on an MEH.

Fig. 5 shows the physical testbed with logical connectivity of the involved elements. The testbed is composed of 3 GIGABYTE(32/512) Intel i7 NUCs. `NUC1` is responsible for running the emulated network scenario implemented with AdvantEDGE, as well as hosting the VLC client of the UE. The AdvantEDGE platform is installed on a single K8s node running Ubuntu 20.04.4 LTS Operating System (OS). Access to the AdvantEDGE platform GUI is achieved through the IP address 192.168.64.68, enabling the configuration and deployment of the emulated network scenario.

`NUC2` is designated as the MEH responsible for the initial deployment of the MEC App, specifically the VLC server. Conversely, `NUC3` serves as an alternative MEH to which the MEC App can migrate. The choice between `NUC2` and `NUC3` as the migration destination depends on the geographical position of the UE. In the scenario configuration, the external MEC App is associated with the MEH by using the IP address and the port number of the related `NUC2` and `NUC3`. This enables AdvantEDGE to provide support for conducting experiments involving external nodes and applications.

4.1. Migrating the MEC App

The K8s cluster, shown in Fig. 6, comprises a master node and one or more worker nodes. The master node has the control plane functions necessary to coordinate and orchestrate the activities of the worker nodes of the cluster. It plays a crucial role in managing the overall cluster state, scheduling pods, and exposing the API for cluster interactions. The main components of a master node, shown in the figure, are the following:

- The Kube API Server acts as the central control point for interacting with the K8s cluster. It exposes the K8s API, allowing users, administrators, and other components to communicate with the cluster.
- The Controller Manager is responsible for maintaining the desired state of the cluster. It continuously monitors the cluster resources and ensures that the current state matches the desired state defined in the cluster configuration.
- The Scheduler is responsible for assigning pods to nodes in the cluster based on resource requirements, node constraints, and other policies.
- The etcd is a distributed key–value store used by K8s to store the cluster configuration data, including information about nodes, pods, services, and other objects.
- Kube-proxy is a component that runs on each node of the K8s cluster and is responsible for implementing the necessary network routing for services based on instructions received from the control plane. Specifically, when a pod is migrated, Kube-proxy updates the network configuration of local node to ensure that client traffic destined for the service virtual IP address is correctly

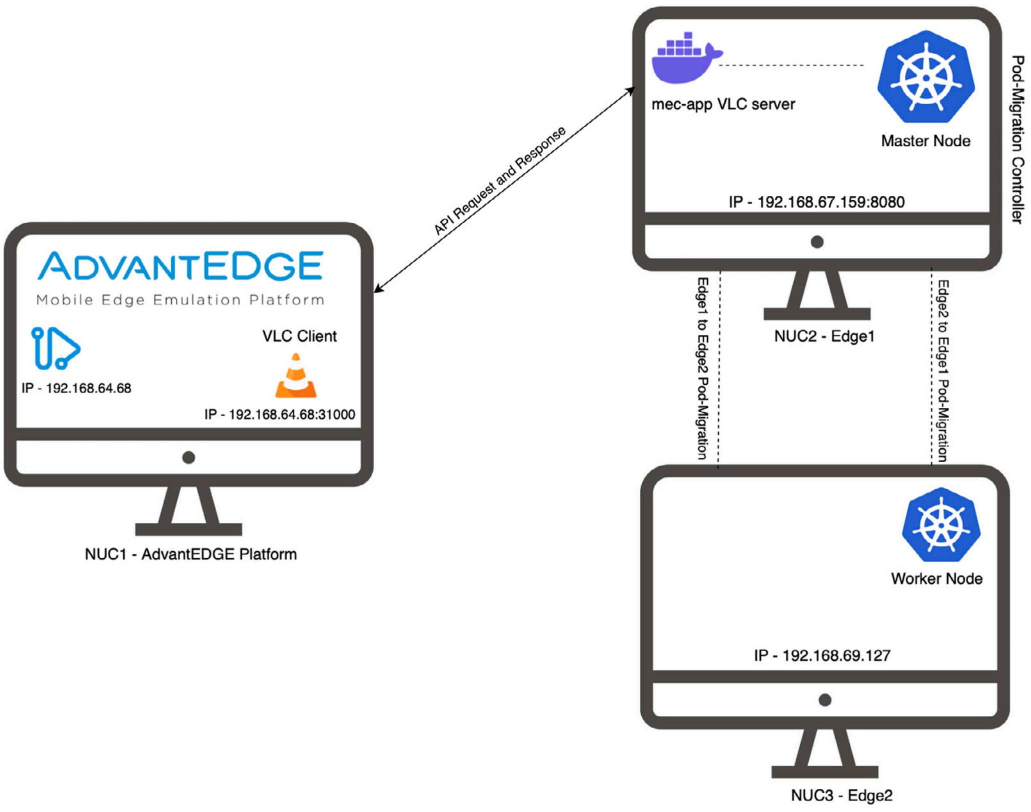


Fig. 5. The testbed.

routed to the appropriate pods backing the service. A service is a method used in K8s for exposing a network application that runs as one or more pods.

The worker node is responsible for running the actual workloads, including containers and pods. It contributes to the overall execution and management of workloads in the K8s cluster, allowing applications to run and scale efficiently across the distributed environment. To achieve this goal, a worker node implements the following functions shown in the figure:

- Kubelet is an agent that receives instructions from the master node, schedules pods onto the node, and monitors their health and resource usage. It is responsible for managing the state of the node and ensuring that the containers and pods on the node are running as expected.
- The Container Runtime is responsible for running and managing containers. It provides the environment for running application containers and manages their lifecycle, including image management, container creation, starting, stopping, and resource isolation. An example of a container runtime is `containerd`.
- Kube-proxy enables the communication between services and pods within the cluster. It manages network routing, load balancing, and service discovery, allowing applications to communicate with each other and access services seamlessly.
- The Pod is the fundamental unit of deployment in K8s. Pods encapsulate one or more containers and share the same network namespace, storage volumes, and scheduling constraints. The worker node ensures that the containers within the pods are running and handles their lifecycle, resource allocation, and networking.

Referring to this general architecture of a K8s cluster, the `NUC2` implements both control plane functions of a master node and data plane functions of a worker node, while `NUC3` is only a worker node.

To enable manual pod migration in K8s, additional elements and configurations need to be incorporated into the cluster. By default, K8s does not support manual pod migration between nodes initiated by the operator. However, it offers built-in mechanisms for pod migration in specific scenarios, such as node draining or when constrained by other factors. In this case, an extended version of K8s is utilized, as referenced in [22], which incorporates the essential features and functionalities needed for smooth and efficient pod migration operations.

In this architecture, the MEC App is implemented in a Docker container and deployed using a pod referred to as `video pod` that supports the VLC server application. The runtime migration is implemented by using the extended K8s version, allowing the runtime pod migration to use the CRIU tool. CRIU enables the checkpointing and live migration of running containers from one node to another within a K8s cluster. When a pod migration is initiated, CRIU captures the state of the running containers, including their memory, file system, and network connections. This captured state is subsequently transferred to the target node, where it is used to restore the containers, ensuring uninterrupted execution. In the testbed, an important aspect is preserving the state of the VLC server by retaining the last transmitted frame of the video stream for each individual user.

Fig. 7 shows the details of the architecture responsible for the `video pod` migration. `NUC2` and `NUC3` implement the extended K8s version necessary for application migration. This extended version of K8s (in specific the `pod migration API server`) is an element that provides support to the `kubectl-migrate` and `kubectl-checkpoint` commands [22], which have been integrated into `kubectl`, the command-line interface (CLI) tool used to interact with the K8s cluster. It acts as a control plane client and allows users to manage and control various aspects of the cluster. The `pod migration API server` facilitates information exchange between the `pod migration controller`, UE App (i.e., the VLC client), and the K8s nodes. It directs the pod migration from `NUC2` to `NUC3` and vice versa.

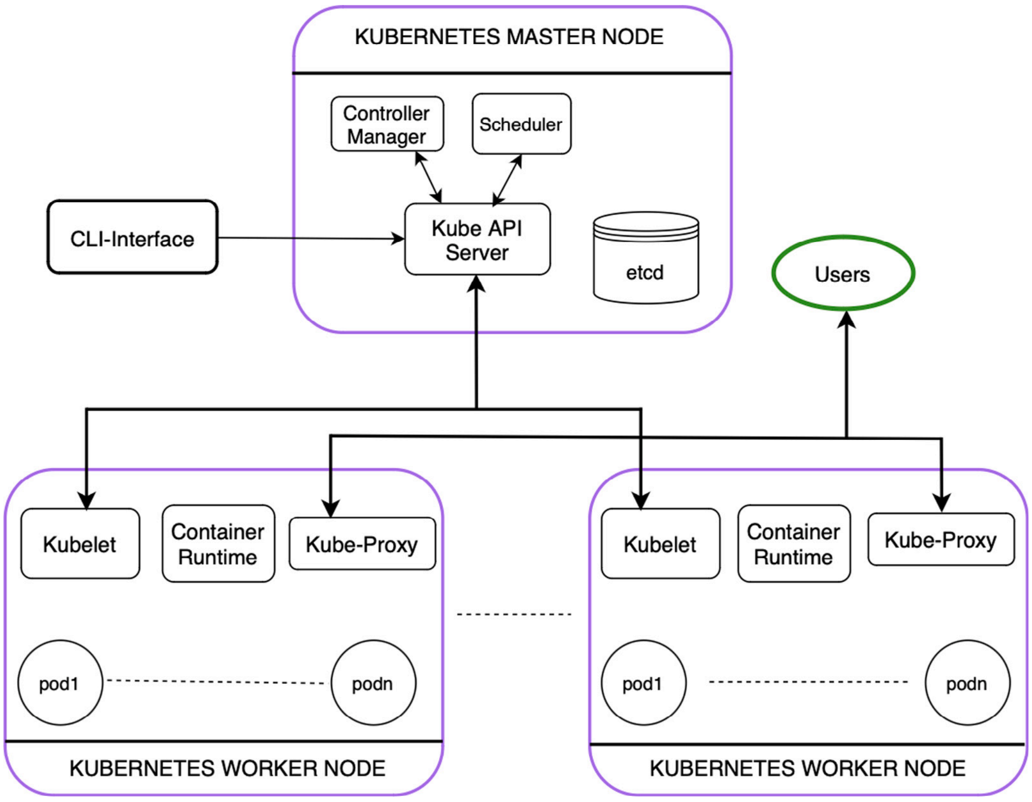


Fig. 6. K8s cluster and components.

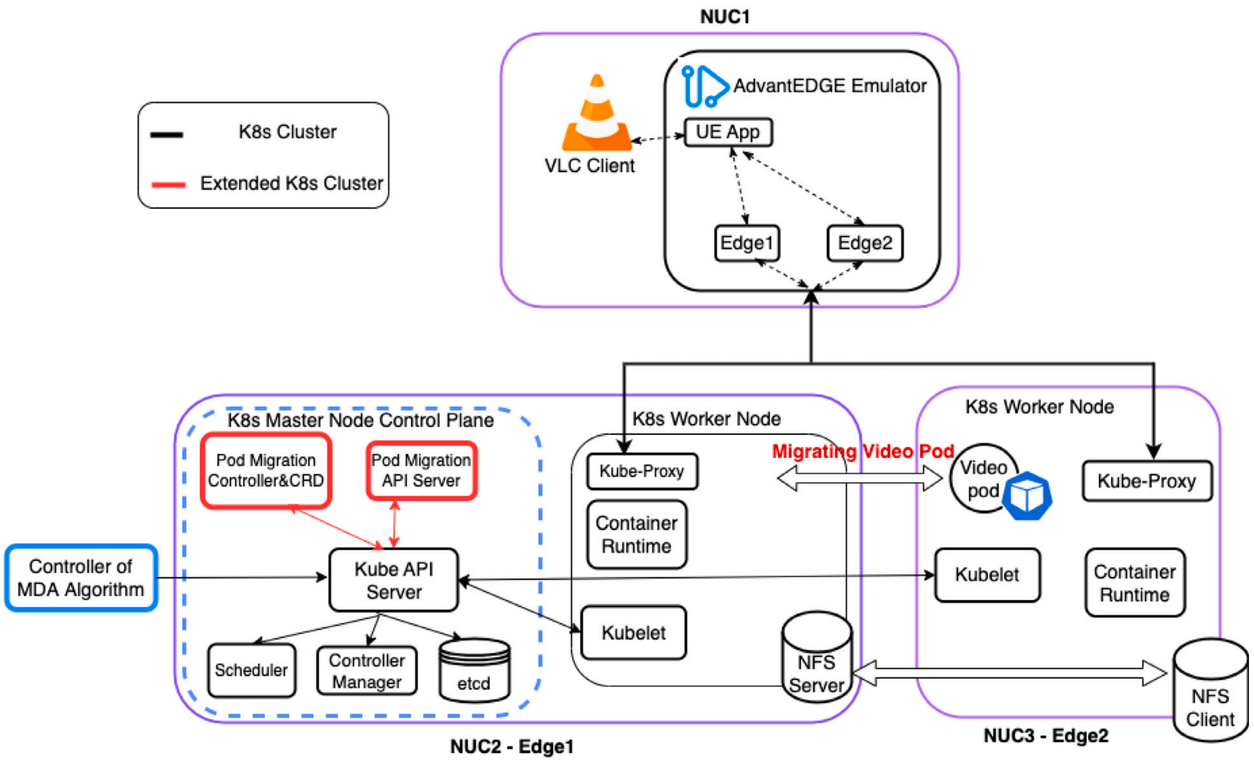


Fig. 7. Architecture for video pod migration in the experimental testbed.

During the experiments, the controller developed for testing MDA runs on NUC1 and interacts with AdvantEDGE. The controller utilizes the APIs of ETSI MEC specifications provided by the platform to acquire UE and network information. This acquired information is utilized by the MDA to determine the need for triggering the migration of the MEC App. When migration becomes necessary, the developed controller sends a request to the Kube API Server, which initiates the migration by sending the request to the `pod migration controller`. This controller generates two essential commands, namely `kubectl-checkpoint` and `kubectl-migrate`, for the migration process. The `kubectl-checkpoint` command is responsible for creating a checkpoint of the running container and saving its state as files on the respective nodes. Conversely, the `kubectl-migrate` command facilitates the migration of the application within the nodes.

The `pod migration controller` is a component that facilitates the migration of pods from one node to another within a K8s cluster. It manages the process of moving pods while ensuring minimal disruption to the running applications. It is worth noting that this element is not directly connected to the Controller Manager, which primarily manages the deployment and health of nodes and pods. The `pod migration controller` used in this architecture includes Customized Resource Definition (CRD) and a custom controller to monitor the pod migration within the K8s cluster. The CRD mechanism supports user-defined data types in K8s and allows the design of the required state, which will be transferred to the target node by the `pod migration controller`. When the migration is triggered, this controller orchestrates the migration process by following a set of steps:

- Designate a specific pod migration controller that will assume the responsibility of overseeing the entire pod migration process.
- Identify the pod to be migrated.
- Prepare the target node by ensuring that it has the necessary resources and dependencies to accommodate the migrated pod. This may involve allocating resources, setting up networking, and preparing the environment.
- Capture the state of the pod on the source node, including its network connections, attached volumes, and other relevant information (in the considered framework, this is done by checkpointing the application). This state information is crucial for preserving the pod functionality during the migration.
- Transfer the captured state from the source node to the target node through the network connecting the two NUCs. This step ensures that the pod state is replicated on the target node.
- Initiate the restoration process on the target node. Once the state transfer is complete. The controller ensures that the pod containers are started, network connections are established, and any necessary dependencies are met. The pod resumes execution on the target node, seamlessly continuing its tasks.
- Update the Kube API server, reflecting the changes in the pod location and status.

The source node updates its state by removing the migrated pod and reclaiming any previously allocated resources. The `Kube API server` maintains the logs of each step in the application migration process, including the time of the migration request and its completion. It also manages the synchronization between the nodes to ensure proper coordination throughout the migration process.

A shared NFS folder, `/var/lib/kubectl/migrate`, is created and utilized to facilitate the sharing of recorded checkpoint information acquired in the MEH where the MEC App is running. In the testbed, NUC2 is configured as the NFS server, while the NUC3 worker node acts as the NFS client.

The successful migration of the MEC App depends on the coordination and interaction of these components, as well as the compatibility and appropriate configuration of the extended K8s version. The use of CRIU and the integration of the extended K8s version enable the checkpointing and migration of running containers, ensuring a smooth transition of the MEC App between nodes within the K8s cluster.

The implementation of MEC and MDA may require a high level of technical expertise and resources. However, in our testbed we do not require any customization of the MEC architecture but the developed controller, which can be seen as part of the MEO, allows to easily integrate MEC and MDA and hides the MDA complexity. The controller collects the system information by exploiting the standard MEC APIs. This information includes data such as UE location, Radio Signal-to-Noise Indicator (RSNI), link performance, and other telemetry data. The information is then delivered to the MDA, which consequently computes a solution. The controller then applies the MDA solution in the system. Additionally, in general ETSI simplifies the implementation of MEC by giving the option of reusing elements of Network Function Virtualization (NFV) to implement the MEC architecture [23].

4.2. Network scenario

The design of the network scenario is motivated by the imperative need to conduct a functional experimental evaluation of the proposed controller proof-of-concept. The chosen scenario is intentionally crafted to encompass a dynamic environment with multiple handovers and mec-app migrations. This deliberate scenario construction aims to stress-test the capabilities and responsiveness of the proposed controller under conditions simulating real-world challenges in contemporary wireless networks.

Fig. 8 shows the starting point of the network scenario as depicted in the AdvantEDGE GUI. The scenario consists of a single UE (ue1) ocated in zone1, while zone2 and zone3 contain the emulated MEHs edge1 and edge2 respectively. In the testbed, edge1 is deployed in NUC2, whereas edge2 is deployed in NUC3. Each zone is equipped with a different network access technology: zone1–WiFi, zone2–5G, and zone3–4G. Depending on its location, the ue1 can establish a connection to the MEHs through one of the two PoAs available within each zone. The MEHs are represented by blue boxes in the figure, while the brown boxes indicate the locations of the application elements. In this particular scenario, the application elements consist of the Video Pod (mec-app) running in MEH edge1, and the VLC client (vlc1) running on ue1. The physical UE is depicted as a green box, while the antennas represent the PoAs. In the figure, Operator1 is the ISP, which provides the IP connectivity through the three access technologies and the IP services supported by the MEC architecture.

The AdvantEDGE platform facilitates the assignment of physical locations for each element depicted in the scenario. The three distinct networking technologies of the PoAs are geographically mapped in different locations, as illustrated in Fig. 9. The geographical setting represents the area surrounding the Arno River in Pisa. The WiFi PoAs have a coverage radius of 200 m (indicated in red), while the 5G PoAs span 500 m (in orange) and the 4G PoAs extend up to 1000 m (in purple). The blue line in the figure indicates the path followed by ue1 for the experimental analysis.

4.3. Applying MDA

The graph of the considered network scenario is shown in Fig. 10. In this case as well, the attributes assigned to each link are the packet loss probability, jitter, and latency. The links connecting op1 with the various zones are assumed to operate at a datarate of 2 Mbps. The black links and wireless connections between the PoAs and ue1 within the coverage range have a datarate of 100 Mbps. Furthermore, the datarate for the connection between the application layer of the MEH and the corresponding network layer of the supporting node is 1 Gbps, as indicated by the red link in the figure.

During the experimental sessions, the GIS API (`getGeoDataByName`) is utilized to acquire information regarding the physical position of ue1, which is necessary for calculating the distance between ue1

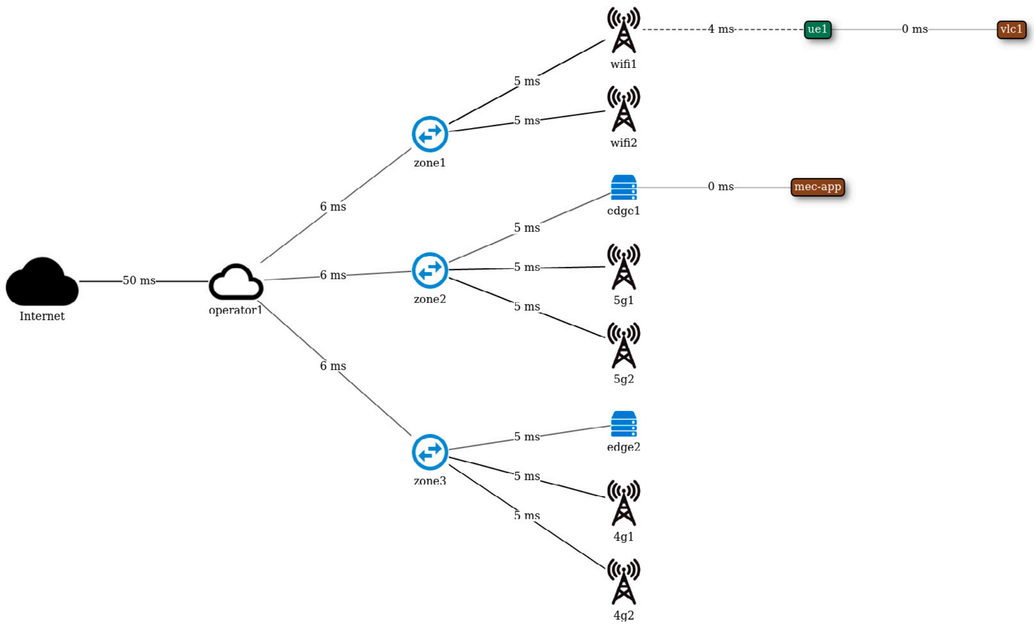


Fig. 8. Network scenario described by the AdvantEDGE GUI. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

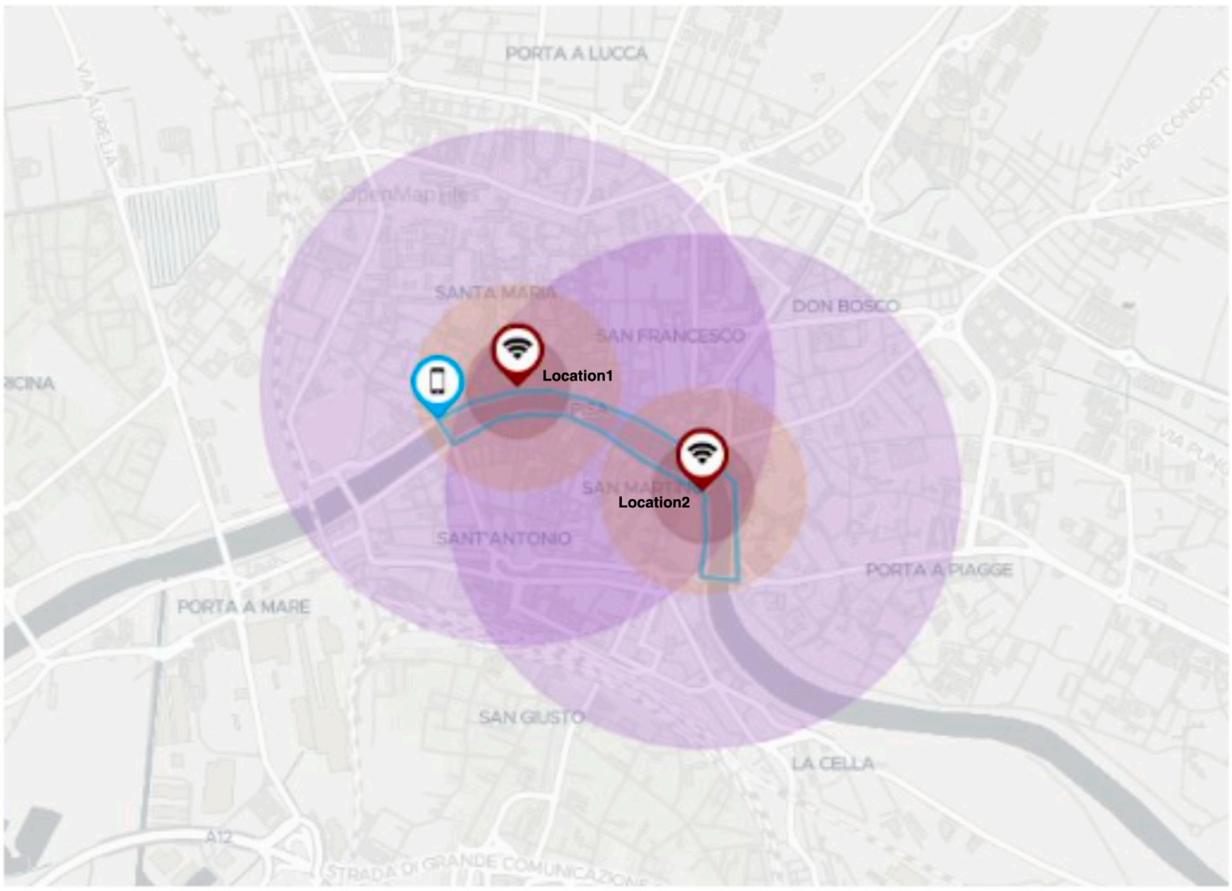


Fig. 9. Map of the scenario considered in the experimental analysis with AdvantEDGE platform. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

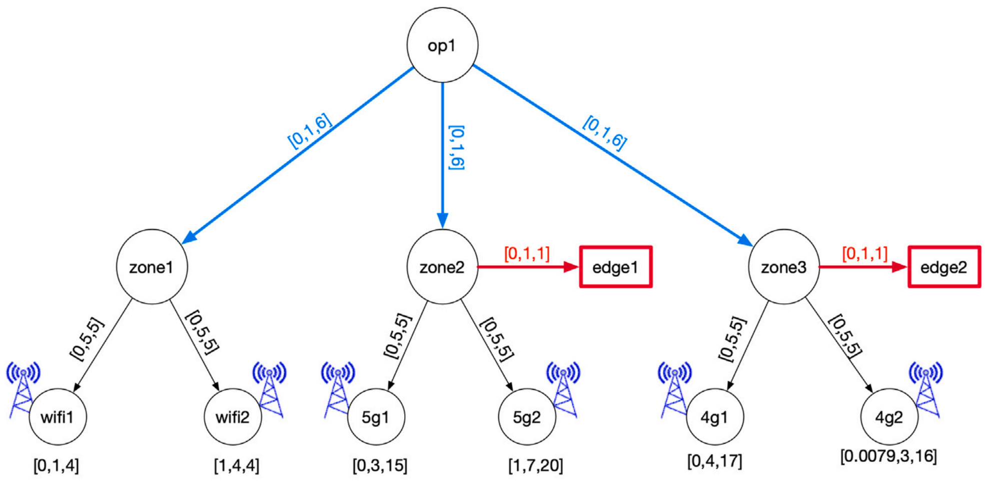


Fig. 10. Graph depicting the network scenario being analyzed, with each link accompanied by a vector representing the metrics of packet loss probability, jitter, and latency. The blue links operate at a datarate of 2 Mbps, while the black links and wireless connections between the PoAs and `ue1` within the coverage range operate at a datarate of 100 Mbps. The red links connecting the application layer of the MEH and the corresponding network layer have a datarate of 1 Gbps. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

and the various PoAs. This information is employed to determine the set of PoAs capable of providing connectivity to `ue1`. Given the distance between the PoAs and the UE, AdvantEDGE lacks the capability to dynamically compute the datarate of the corresponding wireless links. However, it does provide information regarding the available PoAs, i.e., PoAs that have the UE within their coverage range. When a PoA is available, the datarate is set to the value manually configured by the user during the setup of the network scenario. The Sandbox API (sendEvent) of AdvantEDGE enables the capability to change the PoA to which `ue1` is connected (i.e. performing the PoA handover), allowing for PoA handover, during runtime. The data required for constructing the graph, including nodes, arcs and their attribute values, can be obtained at runtime by using the APIs of the MEC architecture. To simplify the experimental tests, the attribute values of the arcs are assumed to remain constant throughout the experiments. Therefore, these values have been manually configured in AdvantEDGE.

The test is based on the application of MDA to the dynamic graph generated by using AdvantEDGE information. It is worth noting that the attributes of the link are assumed constant, but the mobility of `ue1` varies the alternative links available to the `ue1` to reach the available MEHs. Depending on `ue1` location, the MDA output gives the selected MEH and the serving PoA of `ue1`, which establish the traffic path in the considered scenario.

Table 5 presents the performance parameters of the available paths from the UE to the MEH for all available PoAs and MEHs. The values presented in the table indicate the performance obtained from the MDA output when the UE is located within the coverage range of the respective PoA. The table highlights that the dominant solution is to connect to `5G-1` and utilize MEH `edge1`. Consequently, when the UE is near location 1 and within the coverage range of `5G-1`, `5G-1` and `edge1` are selected. If `5G-1` is not within range, then the predominant solution is `4G-2` with the utilization of MEH `edge2`.

It is worth noting that AdvantEDGE does not consider the control plane procedures for executing handovers between PoAs of the same technology or between different technologies (i.e. multi-Radio Access Technology (multi-RAT) handover). As a result, the delay and certain performance issues (such as packet loss or increased jitter) induced by these procedures are neglected.

4.4. Test execution

The simulation involves the interaction of multiple components: AdvantEDGE, the developed controller with MDA, and K8s. The logic of the developed controller is described by the flow chart shown in Fig. 11.

The figure illustrates the initial configuration of the network scenario in AdvantEDGE, including the placement of PoAs, zones, MEH, and the performance parameters of the links. Additionally, the settings for the initial position, speed, and application-related parameters of `ue1` are required. Subsequently, the movement of `ue1` is initiated using the automation feature in the AdvantEDGE GUI, allowing `ue1` to access its application. In this specific case, the VLC client begins displaying the video stream provided by the `mec-app` (i.e., the VLC server) hosted in `edge1`. All this information is loaded by using the AdvantEDGE-specific API known as Sandbox API. The test is prepared for execution once the developed controller subscribes to the ETSI-specific MEC (013) location service API. This subscription is essential for obtaining real-time data necessary to generate the network graph, which serves as input to the MDA algorithm. Specifically, the location of `ue1` is monitored every second, enabling the application of the MDA.

In general, the output of the MDA determines the MEH that will support the UE service `mec-app` and the path to connect `ue1` with the selected MEH. The first aspect may require the migration of the `mec-app`, while the second aspect can modify the path. Due to the simplified network configuration in AdvantEDGE, the traffic path can only be modified through a PoA handover. As previously mentioned, a PoA handover entails the migration of the `mec-app` and vice versa, as indicated by the values in Table 5. Therefore, the flow chart in Fig. 11 solely includes the control for the required PoA handover, as this change also involves the migration of the `mec-app`. In general, two separate controls, one for the PoA (or path in the general case) and one for the MEH, are necessary to account for the other cases: i) no PoA handover and no App migration, and (ii) PoA handover but no App migration.

In the case where the MDA triggers a PoA handover, two actions are required. First, the PoA handover is posted through the Sandbox API of AdvantEDGE. This action is necessary to establish the new traffic path between `ue1` and the `mec-app`.

Secondly, if the new path leads to a different MEH, the `mec-app` migration request is sent to the `pod migration controller` via `Kube API server`. Upon receiving this request, the `pod migration controller` creates and assigns a dedicated video controller pod for the migration process. The video controller pod verifies the information of the `mec-app` pod, including the pod name and its running status. Once the `mec-app` pod name is obtained, the `pod migration controller` migrates the pod to the designated MEH using the `kubectl`

Table 5
Performance parameters of the path for the experimental tests as a function of the PoA and MEH.

`edge1`-UE	WiFi-1	4G-1	5G-1	WiFI-2	4G-2	5G-2
PLoss (%)	0	0	0	1	0.0079	1
Jitter (ms)	9	12	9	12	11	13
Latency (ms)	22	35	21	22	27	26
Min. Data Rate (Mbps)	2	2	100	2	2	100
`edge2`-UE	WiFi-1	4G-1	5G-1	WiFI-2	4G-2	5G-2
PLoss (%)	0	0	0	1	0.0079	0.1
Jitter (ms)	9	10	11	12	9	15
Latency (ms)	22	23	33	22	22	38
Min. Data Rate (Mbps)	2	100	2	2	100	2

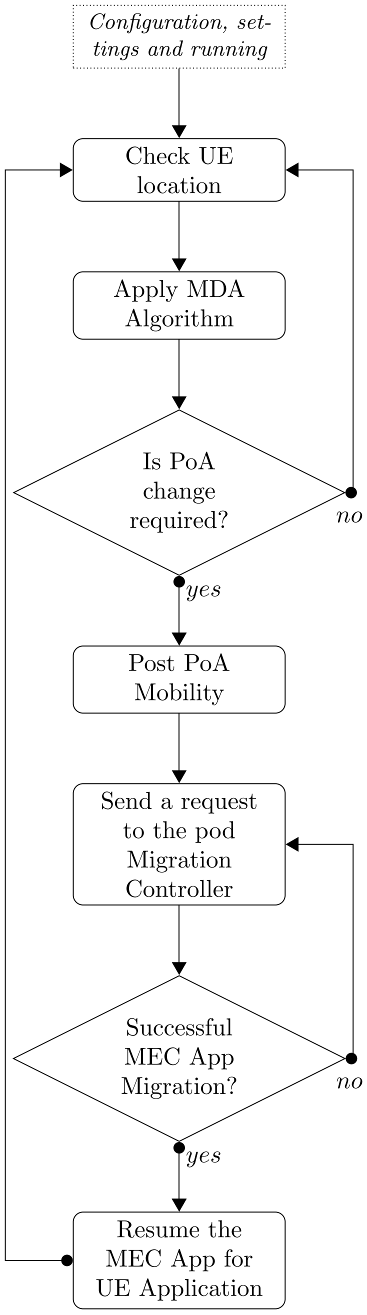

Configuration, settings and running
Check UE location
Apply MDA Algorithm
Is PoA change required?
no
yes
Post PoA Mobility
Send a request to the pod Migration Controller
Successful MEC App Migration?
no
yes
Resume the MEC App for UE Application

Fig. 11. Flow chart depicting the logic of the developed controller for the network scenario under consideration. The migration is triggered by the PoA handover, as indicated by the MDA results presented in Table 5.

Table 6
Features of transmitted video - Audio and video bitrate refer to the average values.

Codec	Video bitrate	Audio bitrate	Width	Height
H.264	3060 kbps	256 kpbs	1920 px	1080 px

commands. A new name is assigned to the migrated `mec-app` pod based on the MEH selected by the MDA output. Subsequently, the video controller pod verifies the updated information of the new `mec-app` pod, such as its running status, and relays this information to the `pod migration controller`. Finally, the `pod migration controller` deletes the associated video controller pod and checks the status of the `mec-app` pod in the new MEH using K8s.

The verification process may encounter errors due to various reasons, such as the non-existence or non-running state of the `mec-app` pod in the new MEH, or the absence of the current application state in the shared NFS folder, among others. In such cases, the strategy involves sending a new request to the `pod migration controller` to attempt another migration of the same `mec-app`.

On the contrary, if the control confirms the successful completion of the pod migration, the `mec-app` service is resumed. It is important to note that the information regarding the restart of the `mec-app` service is limited to the VLC server, as K8s cannot verify if `vlc1` maintains the service session. During the experimental tests, it is observed that when the `mec-app` restarts, the `vlc1` application maintains the session, and user experiences minimal service degradation.

5. Experimental results

Experimental tests are conducted on the same testbed, exploring two distinct network scenarios. These scenarios primarily differ in the presence or absence of the MEH `edge2`, while MEH `edge1` is always present. In the first scenario, the aim is to assess the benefits of service migration between the two MEHs when such migration becomes necessary to maintain the desired service quality. In the second scenario, the focus shifts towards observing the degradation in QoS when the service relies only on MEH `edge1`. In this case, the mobility of `ue1` leads to the PoA handover to maintain the network connection, but the traffic path from the new PoA to MEH `edge1` does not satisfy the QoS requirements.

In both cases, as detailed in Section 4.1, a videostreaming service is considered, implemented by means of a VLC server supported by a pod of the K8s framework. The VLC server streams the video by using the MPEG Transport Stream (MPEG TS) protocol, defined in the ISO/IEC standard 13818-1 [24], over TCP. The video shows "Big Buck Bunny" movie and lasts $597s$. The main features of the video are summarized in Table 6.

The traffic generated by the VLC server is delivered to the `UE-app` (i.e., the VLC client) through the network emulated by AdvantEDGE. The playout buffer of the VLC client is set to $1s$. On the data plane, AdvantEDGE adds packet loss, delay jitter, latency and controls each packet transmission time (related to the available link data rate) according to the network characteristics set in the scenario. As a consequence, the quality of the streamed video might be affected to various degrees by the performance of the used network links.

5.1. Performance parameters

During each test, two different classes of performance parameters are collected and analyzed. The first class refers to the parameters obtained from the Grafana dashboard [25] of AdvantEDGE, which are used to acquire data during the emulation. Grafana retrieves metrics such as latency, UL/DL throughput, UL/DL packet loss, and handover events from the InfluxDB database. AdvantEDGE deploys InfluxDB as a pod, creating a dedicated database for the mentioned metrics for each scenario deployment. This information remains available during runtime and until the scenario is redeployed. The data visualized in Grafana can be exported in `csv` format at the end of the experiment. The following parameters are considered:

- **Latency** is the time a packet takes to be transferred from the ingress to the egress point of AdvantEDGE. Considering a path composed of more than one arc, the overall latency is calculated as the sum of latency values across each arc. AdvantEDGE generates the packet latency values by using a random variable following a Gaussian distribution. Referring to the parameters configured for each link of the AdvantEDGE network scenario, the mean of the distribution represents the latency parameter, while the standard deviation corresponds to the jitter. Each second, Grafana shows the latency obtained by averaging the latency observed by the packets arriving at the egress point in consecutive and non-overlapping time windows of 1 ms.
- **Throughput** is the maximum amount of data that can be transmitted in one second. In AdvantEDGE, the data rate can be configured for each link in the emulated network scenario. The reported value is determined by monitoring the throughput of the designated traffic flow at the egress node of AdvantEDGE. In the considered analysis, the chosen node is `ue1`.

The second class refers to the subjective QoE that is observed by the user during the service. The considered parameter is the **Mean Opinion Score** (MOS), which represents the mean of the absolute score given by the customers according to their satisfaction during the visualization of the video. As recommended by the ITU-T P800 standard [26], an Absolute Category Rating (ACR) is used to score the experience by using a five-point category-judgement, from 1 (Bad) to 5 (Excellent).

5.2. Results: Scenario with two MEHs

In this scenario, the data rate available between the two NUCs implementing the two MEHs is set to 1 Gbps. During the experimental run, the selection of the PoA and MEH used by `ue1` is determined by the MDA algorithm, which receives input from the MEC APIs of AdvantEDGE. The selection process considers the geographical position of `ue1`. Table 5 summarizes the performance parameters of the computed path by the MDA when `ue1` is within the coverage of different PoAs. Upon observing the table, it can be noted that `ue1` connects to either `5G-1` or `4G-2` based on its geographical position. Furthermore, when `ue1` is connected to `5G-1`, the MDA recommends utilizing `edge2` as the associated MEH. Instead, if the `ue1` is connected to `4G-2`, the MDA suggests utilizing `edge1` as the preferred option. Given these particular MDA results, during the movement of `ue1`, both the PoA handover and the MEH service migration occur simultaneously. Fig. 12(a) displays the end-to-end latency obtained from Grafana, along with the moving average of this time series using a window of 20 samples. As explained in the legend, the vertical lines represent the PoA handovers of `ue1` (e.g., `5G-1` or `4G-2`), which coincide with the initiation of the `mec-app` migration. The figure presents a specific time period of the experimental test to illustrate the latency between the MEH and UE, as well as the occurrences of handover events. Table 7 displays the maximum, minimum, and average latency values calculated over the entire run.

Table 7
MEH-UE latency.

Average	Minimum	Maximum
36.35 ms	19.32 ms	61.06 ms

Table 8
Statistics on observed `mec-app` migration time in ms.

95% C.I.	Median	Min	Max	95th percentile
3015.46 ± 134.533	2866.5	1459	6789	3903.7

Fig. 12(b) shows the measured throughput at `ue1`, along with the traffic generated by the VLC server and the PoA handover events. The dashed blue lines represent the traffic generated by the VLC server, which was measured using Wireshark [27] under ideal network conditions with no packet loss and high data-rate in each network link.

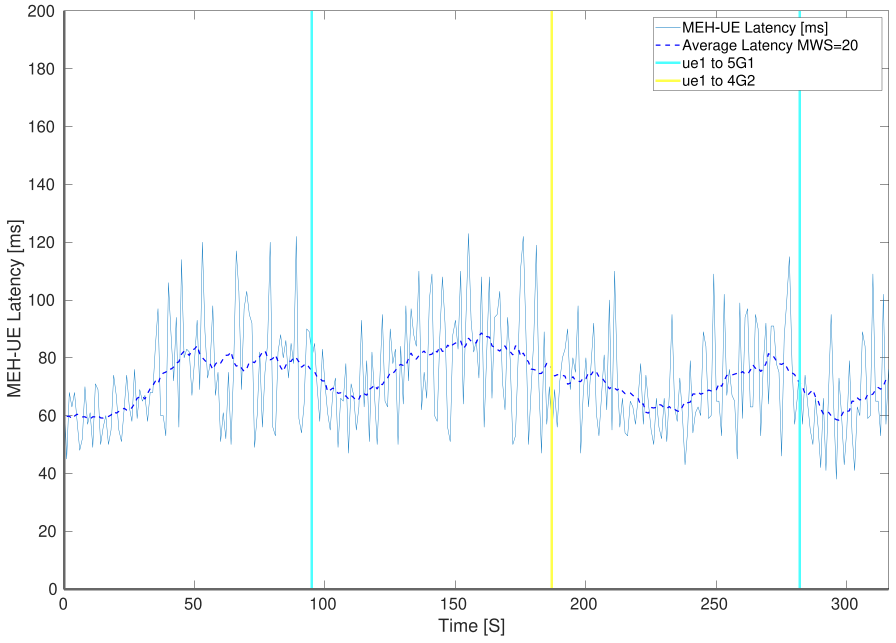
During the experimental analysis, the `ue1` follows the circular track shown in Fig. 9, with each lap lasting approximately 3 min. Within each lap, two migrations occur between MEH `edge1` and MEH `edge2`, and vice versa. The test concludes after observing 100 `mec-app` migration events. The collected data is then analyzed to evaluate the performance of the K8s platform in terms of `mec-app` migration time. Table 8 presents the statistical parameters for the observed migration times. It can be observed that approximately 5% of the migrations require more than 3.9 s, while the minimum values are below 2 s. By setting the playout buffer of the VLC client to 1 s, the degradation of the video experienced by the end-user is mitigated.

During the migration of the `mec-app`, often the videostreaming service experiences a temporary freeze on a single image until the migration process is completed and the VLC server is restored. In particular, in 11 out of the 100 migration events, the freezing phenomenon of the streamed video was not detected by the user. The video continued to play without any noticeable issues, such as momentary image freezing caused by the need to refill the playout buffer. This result can be primarily attributed to the low migration time and to the size of the playout buffer of the VLC client, which allows for the absorption of video data loss for a period of 1 s. In the remaining migration events, a temporary freezing of the video stream for a few seconds was observed, but the image remained of high quality and free from artifacts. Fig. 13 presents a sample video image captured during a service migration. The yellow bar and the play button, highlighted by the red ellipses at the bottom of the figure, indicate that the VLC client is buffering the video and the image is temporarily frozen. However, in the worst case, the video restarts after approximately 5 s. In summary, the experimental tests result in high-quality video with a MOS score of 4.

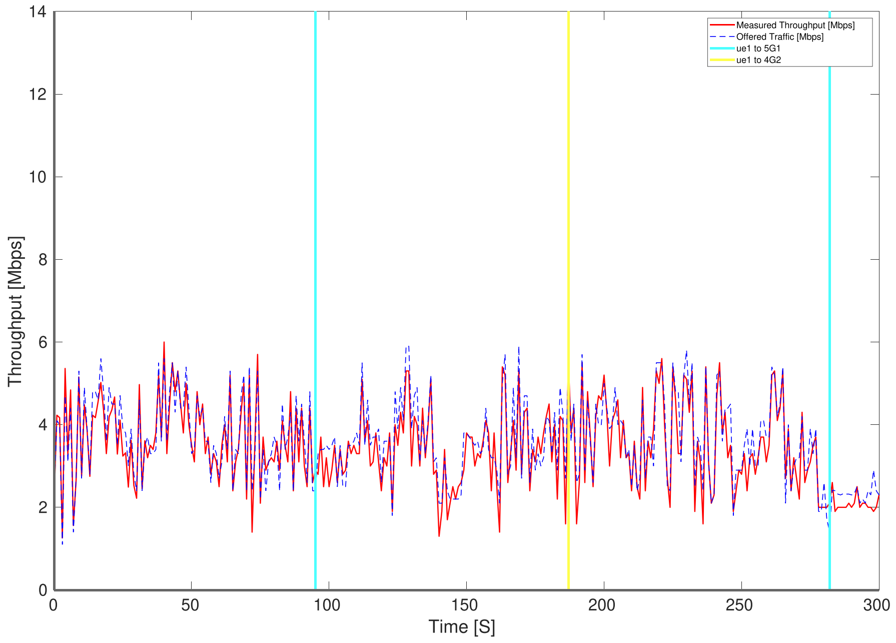
During the experiment, the viability of the migration strategy used in the developed testbed is analyzed. The first viability analysis assesses whether the application can initiate and function properly on the target MEH after the migration. The experimental analysis demonstrates that the developed testbed successfully supports application migration while preserving the necessary user state, resulting in a migration process that is nearly transparent to the end user.

The second viability analysis assesses whether the `mec-app` migration can be completed within a time period that allows the application to remain in the target MEH without requiring another migration. To achieve this objective, the interarrival times between consecutive service migration commands are analyzed. The `ue1` follows a circular path with a distance of approximately 0.611 km between two MEHs. Analysis of the data reveals that the migration command is generated with interarrival times ranging from 85 to 87 s. These values exceed the maximum observed `mec-app` migration time, which is approximately 6.8 s, as presented in Table 8. Thus, the migration process is completed before a new migration is triggered.

The size of the data containing the app state information of the client at the VLC server is on the order of kilobytes. These data are



(a) Measured latency, and PoA handover events - Scenario with two MEHs.



(b) Measured throughput, offered traffic, and PoA handover events - Scenario with two MEHs.

Fig. 12. Performance parameters observed in the scenario with two MEHs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

crucial for resuming the streaming at the new MEH after the migration, starting at the exact point (in the considered experiment, the same image and audio) where it was paused in preparation for the migration.

5.3. Results: Scenario with one MEH

In this scenario, the VLC service is exclusively provided by the MEH `edge1`. Referring to Table 5, the quality of the paths associated with the three available PoA options in location 2 is relatively similar, except for the minimum data rate. Considering this parameter, the strategy that takes into account the constraint on the minimum data rate required to support the application suggests using `5G-2` in this location. MDA can incorporate the constraint on the minimum data rate requirement by excluding all links from the actual graph that have a data rate lower than the minimum requirement. However, for the purpose of comparing the results of this scenario with the previous ones, the selection of PoA `4G-2` is maintained.

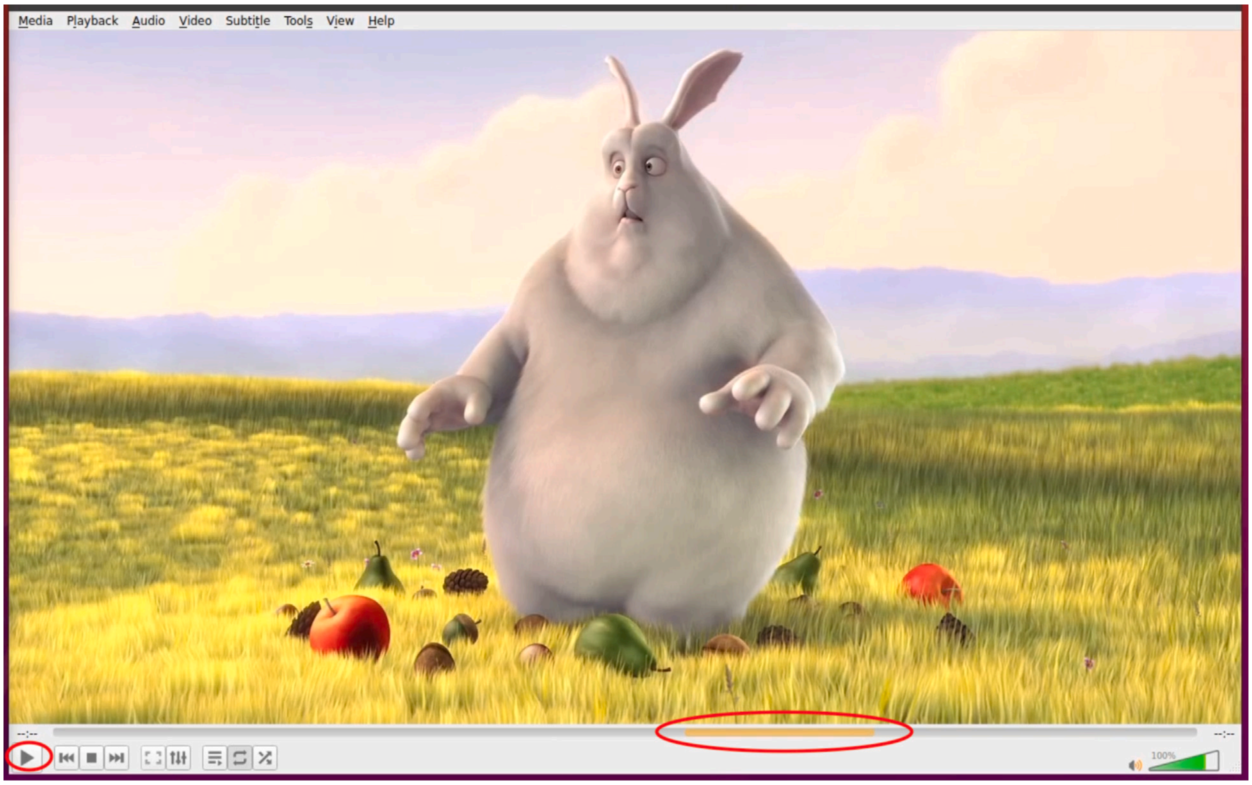


Fig. 13. Paused image during the mec-app migration.

Figs. 14(a) and 14(b) show the latency and traffic curves, along with the PoA handover events, for a single run of 200 s. During this period, three PoA handover events occurred. As observed in the experiment, when the ue1 is connected to 4G-2, which is further away from edge1, the latency exhibits significant oscillations with very high values (around 1 s). Additionally, the figure reveals that during certain periods, the latency appears to be constant. However, no packets are arriving at the egress point of AdvantEDGE during these periods. This phenomenon may be attributed to a link datarate in the network scenario that is lower than the offered traffic. Consequently, the K8s framework of AdvantEDGE may experience packet loss in the network queuing systems, causing the interruption of data acquisition for latency measurements. Further investigation is required to understand this phenomenon. In these periods, the video quality is severely degraded, with the VLC client displaying a frozen frame of poor quality, as depicted in Fig. 15(b). Instead, when the ue1 is connected to 5G-1, the traffic arrives consistently. AdvantEDGE provides the observed latency between the MEH and the ue1, which fluctuates based on the link configuration of the scenario. In such cases, the video resumes after a brief period, exhibiting good image quality. Referring to the throughput curves in Fig. 14(b), when ue1 is connected to PoA 4G-2, the measured throughput (red curve) is bounded by 2 Mbps. However, there are periods when the offered traffic exceeds this limit, resulting in the very high latency values observed above. When the ue1 position allows for the use of PoA 5G-1, the latency returns to the tens of milliseconds range, and the measured throughput at the VLC client aligns with the traffic offered by the VLC server.

To illustrate the impact on the observed quality from the user perspective, Fig. 15 is presented. This figure enables a comparison of the image quality when the ue1 is located in location 2. As depicted in Fig. 15(b), the image quality is poor. When comparing this figure with Fig. 15(a), obtained during the test with two MEHs, the enhancement in quality observed by the end-user becomes apparent upon the execution of the mec-app migration.

6. Related works and novelties

Numerous recent works analyze various technical challenges of MEC. Recent surveys, such as [28–30], summarize the results on three key aspects of the 5G-MEC integrated scenario: security, dependability and performance. Other surveys, such as [31], review the works related to resource allocation in 5G-MEC systems.

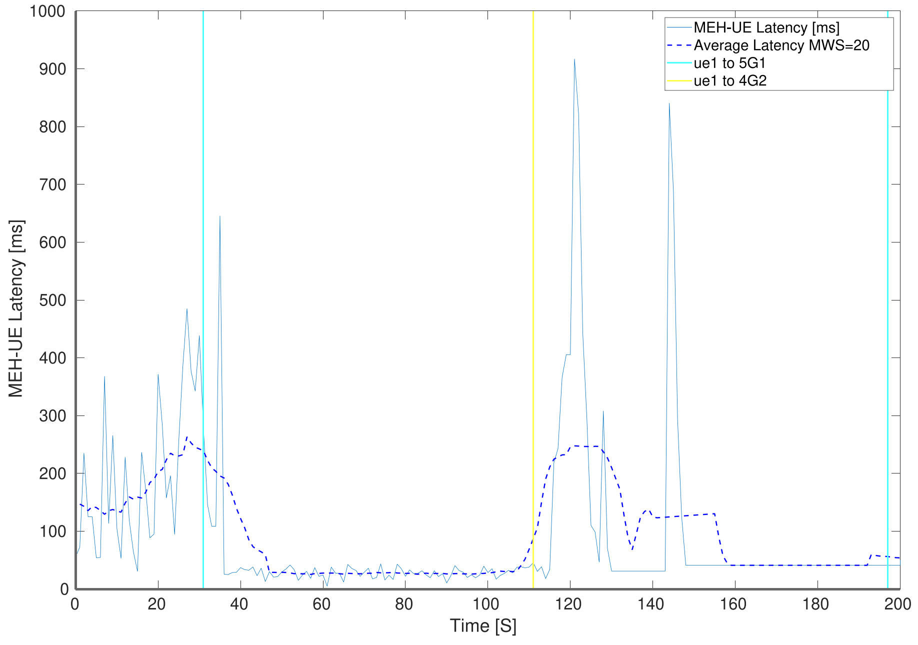
The related works can be categorized into two primary classes, although some, like this paper, consider aspects related to both:

- *Protocols and Architecture*: These works focus on presenting architecture solutions, introducing new elements and protocols aimed at reducing service downtime during migration. They also compare VM-based approaches with container-based ones and define strategies to minimize service downtime. The focus is on the protocol and architecture for service migration.
- *Optimization Algorithms*: These works define optimization problems and algorithms considering specific use-cases, with limited consideration of the protocols and network architecture necessary for deploying the proposed solution.

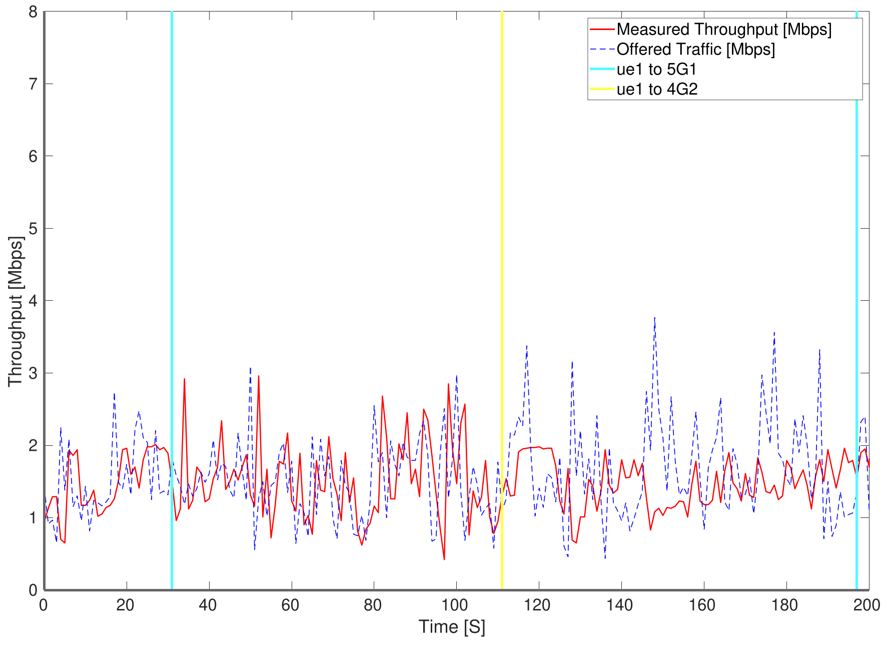
The methodology employed for performance analysis varies across different works. Some present simulation studies based on ad-hoc models, while others use emulation tools or adopt an experimental approach, integrating a proof-of-concept implementation into a simplified network scenario. This paper utilizes a hybrid (simulative-emulative-experimental) approach. A summary of the novelties of this paper compared to related works, along with a comparison with a selected set of related works, is provided in Section 6.3.

6.1. Protocols and architecture

Focusing on the MEC handover process, Plachy et al. [32] consider that computational resources in the edge are represented by VMs. Thus, the MEC handovers are performed through VM migration (and in general of edge applications migration). This migration implies a rerouting



(a) Measured latency and PoA handovers - Scenario with one MEH.

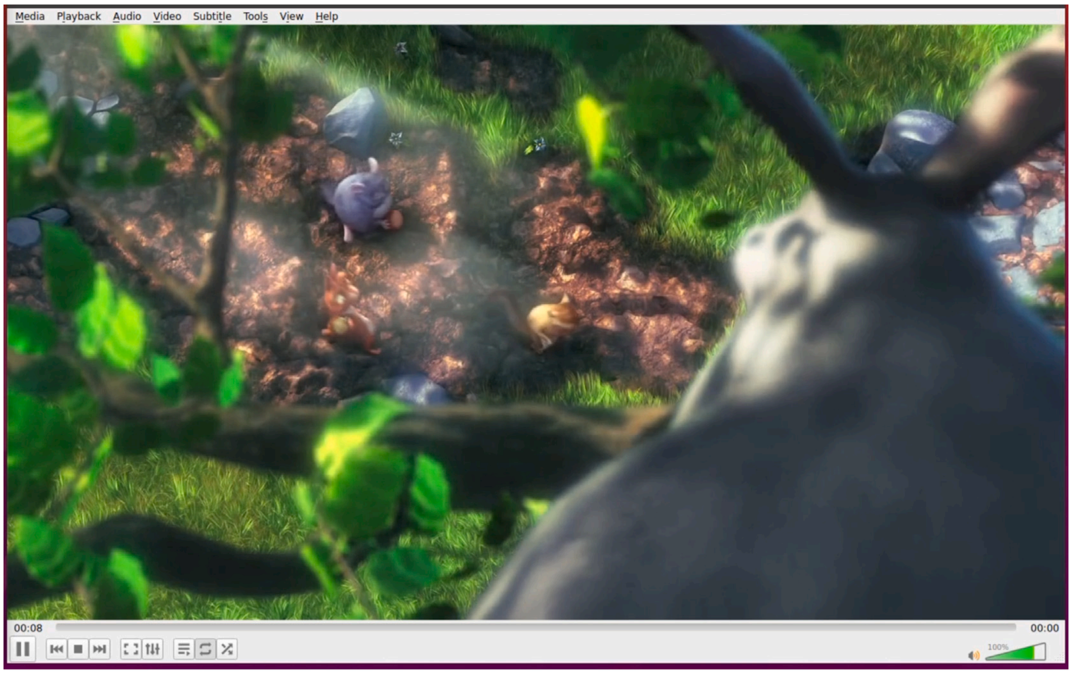


(b) Measured throughput, offered traffic and PoA handovers - Scenario with one MEH.

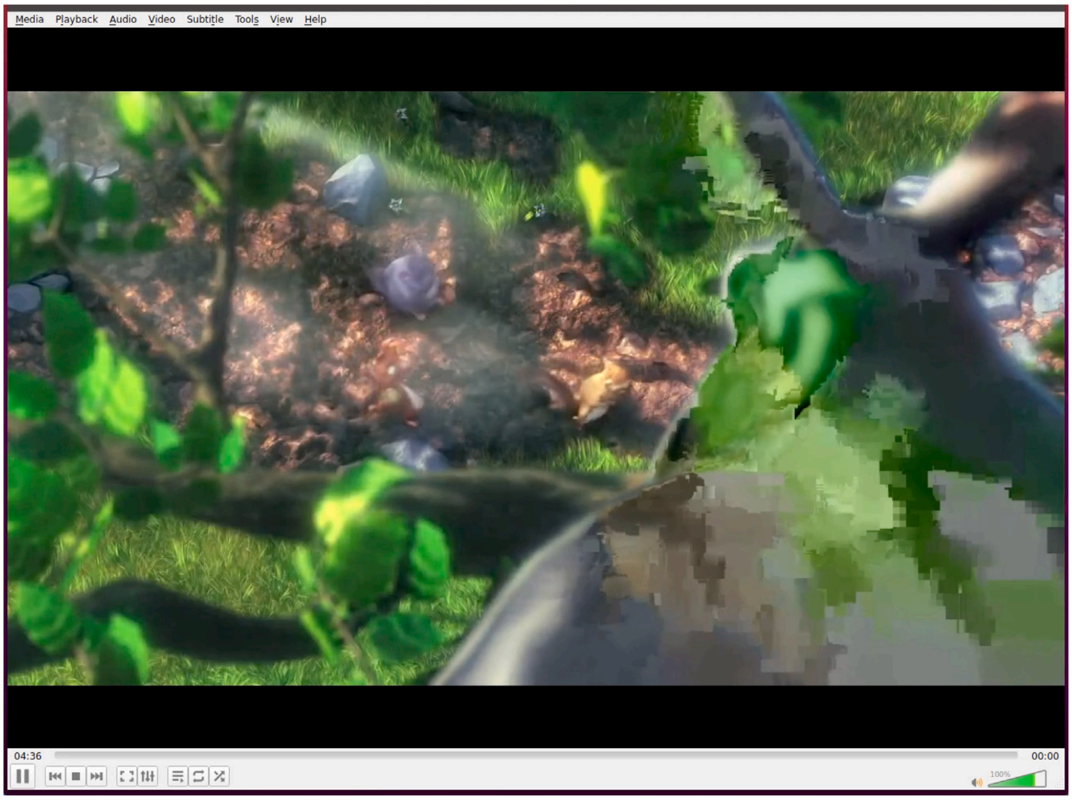
Fig. 14. Performance parameters observed in the scenario with one MEH.

of the traffic to reach the new service location and, in some cases, an exchanging of traffic between the MEHs. Sharghivand et al. [33] propose an Online Service Handoff Mechanism (OSHM) to provide an efficient path dynamically for transferring VM/container from the current serving cloudlet to a nearby cloudlet at the destination of a mobile user. Sarrigiannis et al. [34] explore the implementation of application and VNF migration within an MEC-enabled 5G framework to improve resource optimization and accommodate application-specific demands. Application migration is triggered when MEC resources are depleted, involving the relocation of VMs to better utilize computing resources. VNF migration complements this by reconfiguring network components to address heightened 5G application requirements, especially during periods of increased network traffic.

The volume of traffic exchanged between the source and target MEH depends on the type of migrations (stateless or stateful [35]) and proactive strategies aimed at minimizing the time required to complete the MEC handover. For instance, Machen et al. [35] introduce



(a) Scenario with two MEHs.



(b) Scenario with one MEH.

Fig. 15. Comparison of image quality observed with MEH migration and with one MEH.

a layered framework for migrating active service applications encapsulated in either VMs or containers. This layered approach significantly reduces service downtime. However, automated orchestration is crucial to implement mechanisms that deploy applications at optimal locations and, when necessary, relocate them to meet QoS requirements. Fondo et al. [36] describe an architecture implemented in an experiment demonstrating how Open Source MANO (OSM) can automate the relocation of a video processing application aiding drivers in recalling the latest traffic sign viewed. They propose to add two new components: the first one maintains the state of applications when deployed at a new location, and the second one enables OSM to manage the Open Network Edge Services Software (OpenNESS) edge platform. Wadatkar et al. [37] present a performance evaluation study of a migration technique based on Docker and K8s.

The migration of containerized MEC applications is a key research challenge for supporting low-latency services and for efficient network resource utilization. The migration is necessary to maintain service proximity, reducing the distance between the end user and the application. Different works propose migration strategies [38,39] and experimental comparisons of them [7]. However, for migrating MEC applications, pod migration might be preferred over container migration due to the orchestration and management functions given by K8s. Indeed, K8s provide application portability within the cluster that neglects the complexity of the underlying infrastructure and network condition configurations by offering services such as load balancing, service discovery and fault tolerance. K8s has a default behavior of rescheduling or evicting pods to healthy nodes in the event of a node failure, ensuring the continuity of applications. However, K8s does not inherently handle the check-pointing or preservation of the application state during this process. Consequently, when a pod is replaced by a new instance, the new pod starts somewhere else and does not retain any information or state from the previous pod. To cope with this issue, Schrettenbrunner [40] introduces a migration controller and provides a prototype implementation that demonstrates the feasibility of the proposed approach. Junior et al. [41] present a pod migration mechanism in K8s that enables seamless migration of pods between nodes within a geo-distributed environment. The migration process involves stopping and check-pointing the pod memory state and system-level resources, transferring the checkpoint data to the destination node, and restarting a new pod from the checkpoint. The mechanism implies that the application needs to modify how it handles and persists in-memory state to ensure its preservation during pod migration. Tran et al. [42] proposes a stateful service migration mechanism that extends the capabilities of K8s by leveraging the Checkpoint/Restore In Userspace (CRIU) project [43] and the Container Runtime Interface (CRI) extension [44]. Their work focuses on enhancing fault tolerance and ensuring the high availability of containerized services by considering both the storage state and in-memory state during migration.

Shah et al. [45] propose an architecture integrating SDN and MEC, capitalizing on SDN for end-to-end mobility and QoS. The architecture was validated through V2X simulations using Mininet-WiFi and Docker emulators. The work addresses service migration issues between MEC and introduces DRS for relocating MEC applications with minimal downtime. Concerning application state migration, Docker volumes are employed, primarily focusing on local container data storage, potentially lacking orchestration for multi-node sharing capabilities. The lack of orchestration could lead to extended periods of downtime. The evolution of this work is presented in [46], where the authors suggest a centralized network and MEC server resource coordinator, utilizing SDN orchestration to manage limited resources in highly mobile environments like V2X. Fondo et al. [47] explore the implementation of an SDN solution for dynamically and transparently relocating communication endpoints using containers within cellular networks. This approach ensures session continuity and reduced latency, especially in congested network conditions.

6.1.1. Kubernetes role in ETSI-MEC framework

K8s stands out as the primary container orchestration platform in the contemporary networking landscape. Numerous initiatives are currently engaged in deploying and evolving K8s to meet the functionalities outlined by the ETSI-MEC framework. CAICT et al. [48] extensively discuss the capabilities of the EdgeInfra solution, which manages applications through K8s to enhance Container Network Interfaces (CNIs) and ensures service isolation in Telecom. They underscore the importance of clearly defining the requisites for various industry applications, ideally as K8s templates, to guarantee deterministic 5G capabilities and resource allocation. Other solutions, like EdgeGallery [49] and OpenSigma, also leverage K8s as their edge infrastructure. Martínez-Casanueva, et al. [50] propose an edge computing design based on K8s and Helm, offering function blocks and APIs as defined by ETSI. The prototype demonstrates the feasibility of a lightweight MEC platform. ETSI proposes two white papers. The first [51] emphasizes the importance of application packaging and runtime environments, such as VMs, Docker containers, or K8s templates within the MEC system. The other [52] discusses MEC system support for edge-native designs, with the Edge Multi Cluster Orchestrator (EMCO) of Linux Foundation proposing the use of K8s clusters for scaling geo-distributed applications and network functions for telco solutions. Escaleira et al. [53] demonstrate the efficiency and viability of integrating K8s within the MEC system, following the ETSI standardized framework. The need for the rapid development of a fully operational MEC infrastructure by seamlessly scaling K8s cluster nodes showcases the potential for K8s to play a pivotal role in MEC architecture. [54] proposes different existing solutions for function mapping within the ETSI MEC framework, where K8s may assume the responsibilities of VIM. Slamnik-Krijestorac et al. [55] conducted a study based on container-based service deployment and established a benchmark for MANO solutions in the MEC context. Barrachina et al. [56] propose the MARSAL MEC framework, which leverages a subset of ETSI MEC/NFV where K8s assumes the role of VIM. Bolettieri et al. [57] demonstrate a novel slicing architecture where orchestration and slicing for MEC applications benefit from using K8s and Helm technologies.

6.2. Optimization algorithms

As concerning the description of algorithms for service placement and routing, Poularakis et al. [64] focus on joint service placement and request routing problem in a MEC multi-cell scenario with multiple constraints, aiming to minimize the load of the centralized cloud. A robustness-aware VNF placement and request scheduling scheme is presented in [65], while a model considering the resource allocation for services, the traffic management of requests, and the path arrangement for data delivery is presented in [66]. The goal of the model is to investigate and quantify the relationship between the performance and cost of the edge-based service provisioning system. Considering the multi-service-provider MEC system, the joint service placement and request routing problem has been analyzed in [67]. In this case, the authors consider that a service provider prefers to use edge servers deployed by itself instead of others, which not only improves service quality but also reduces processing costs. The service placement and request scheduling strategies directly affect the revenue of service providers. A recent evolution of this work proposes to federate geographically proximate edge servers to form a logically centralized resource pool [68]. However, the optimization of such systems is challenging. Gohar et al. [69] and Sarah et al. [70] consider an intermediate entity, slice broker, that buys virtual resources from infrastructure providers and sells network slices to slice tenants. In a MEC-cloud scenario, the broker selects the MEH and datacenters (and the related resource configuration) and allocates the virtual network functions composing the network slice. Mason et al. [71,72] consider that the selection of MEH and datacenters has been already made and they dynamically select the data and

Table 9
Comparison with selected existing works.

Ref	Methodology	Contribution	Comparative analysis with our work
[45,46]	Algorithm and Architecture. SUMO Simulation with Mininet-WiFi and Docker	SDN-MEC integration and four modules address mobility, including performance monitoring, service classification, and SDN-assisted traffic steering.	+ Address diverse service requirements and resource management effectively – Lack support for multi-access technology – Overlook multiple objectives (prioritizing latency and bandwidth) – Limited applicability of migration and orchestration to customized MEC environments – Potential portability issues with Docker volume usage for transferring application state
[58]	Algorithm based on ML	Deployment policy based on k-means clustering and particle swarm optimization	+ Utilizes multiple agents for cloudlet deployment and selection – No consideration for network selection and application migration – No implementation and experimental analysis
[59]	Algorithm. Offline and Online stage	Edge server placement considering server heterogeneity and response time fairness	+ Considers edge server placement – No relation to the ETSI-MEC framework – Addresses server selection during user mobility but overlooks application migration – Only simulation analysis, no implementation in a testbed
[60]	Algorithm. Mininet-WiFi and Floodlight SDN controller	MEC state transfer optimization by using FAST framework with Tabu search algorithm	– No consideration of ETSI MEC APIs – No consideration for multi RAT, network selection, and handover – No defined scheme for the execution of application state transfer
[47]	Experiment with ONOS SDN controller and Docker	SDN solution for dynamic relocation of communication endpoints	– Migration of containerized applications from edge to core without edge host selection – Not aligned with the ETSI MEC framework
[34]	Experiment using Openstack, Linux container, OSM and OAI	Network and MEC resource allocation	– No consideration for network handover, multi-objective selection, MEC path selection, and ETSI-MEC support – Possible hypervisor dependencies with VM migration
[61]	Algorithm. Simulation	Resolve scalability for large MEC network and Multi-objective Optimization	– Selection of the shortest dynamic path for the edge server and router only accounting delay and bandwidth – The selection process overlooks migration based on simulated analysis and is not in line with ETSI MEC APIs
[62]	Algorithm. SUMO Simulator and Reinforcement learning	Development of a robust controller for CACC through MEC	+ Addresses placement issue – No consideration for multi RAT – Q-learning algorithm overlooks multi-objective path selection – Simulated application migration lacks practical validation and decision impact demonstration
[63]	Algorithm with COMA reinforcement learning	Distribute task migration based on multi-agent policy	+ Considers a multi-users scenario – Exclusively designed for task migration, neglects optimal path selection for MEC, lacks network handover – No practical validation for task migration

network resources to allocate to each slice in order to maximize the user experience.

The need of adjusting the service placement and request scheduling is discussed by Farhadi et al. [13], referring to data-intensive applications, such as video analytics, machine learning (ML) tasks. The authors show that, due to time-varying demands, the code and data placement need to be adjusted over time, which raises concerns about system stability and operation cost. They address these issues by proposing a two-time-scale framework that jointly optimizes service (code and data) placement and request scheduling while considering storage, communication, computation, and budget constraints. Anwar et al. [61] propose distributed traffic steering by distinguishing between two distinct types of network elements, namely MEHs and routers. They establish the equivalence between solving the Shortest Path problem, which minimizes the cost derived from the sum of latency and the inverse of available bandwidth, and the Pareto optimal path obtained through multi-objective minimization of latency and the inverse of available bandwidth. The proposed approach yields improved results, incorporating Pareto optimality considerations for minimizing various criteria.

Rodrigues et al. [58] propose a deployment policy for edge servers based on k-means clustering and particle swarm optimization to reduce operational costs and service delays. On the other hand, Cao et al. [59] investigate the edge server placement problem, considering the heterogeneity of servers and the fairness of response time. They propose an approach with both offline and online stages. The work by Doan et al. [60] focuses on optimizing MEC state transfer, considering factors such as optimality, latency, and communication awareness. The goal is to minimize migration costs under various constraints. They introduce the FAST (Flexible And Low-Latency State Transfer) framework, applicable to large-scale networks, and efficiently implement a Tabu search algorithm. This algorithm iteratively explores the search space, avoids revisiting solutions using a Tabu list, and updates solutions while considering constraints and an aspiration criterion. Ayimba et al. [62] focus on developing a robust controller for Cooperative Adaptive Cruise Control (CACC) of connected cars through MEC in cellular networks. This addresses low-latency connection maintenance and platoon switching. The study introduces a Q-Learning algorithm for network adaptation and evaluates the performance of the migration scheme in comparison to a state-of-the-art scheme. Liu et al. [63] tackle the task migration problem in MEC, introducing a distributed task migration algorithm using Counterfactual Multi-Agent (COMA) reinforcement learning. The primary objective of this algorithm is to minimize the average task completion time while adhering to a migration energy budget.

6.3. Summary of the novelties

Regarding the architecture aspect, the novelty of this paper lies in the utilization of the pod migration framework [22] to conduct an experimental performance analysis of the proposed MDA. To accomplish this, a controller based on the MDA approach is developed, enabling seamless interaction with the components of the pod migration framework. This integration facilitates the implementation of the MDA decision for application migration within the testbed, thereby enabling the experimental evaluation of performance at both the network layer and the application layer.

Regarding the optimization algorithm, the novelty of the study proposed in this paper is that the proposed multi-objective technique can take into account more than two metrics simultaneously. The result is more general because packet loss, bandwidth, latency, and other performance metrics (e.g. power consumption, security level etc.) can be considered simultaneously. Furthermore, the proposed scheme for generating the graph containing information on the performance at the application layer and at the network layer allows to jointly find the solutions to both the traffic path computation problem and the MEH selection problem. Moreover, this paper explores the performance at the network layer as well as the enhancement of user experience through the utilization of the proposed scheme in a hybrid testbed with an ad-hoc controller.

Table 9 offers a comparison analysis of a selected set of works in the field related to our contribution. The table outlines the methodology of the study, the contribution, and the pros and cons with respect to our work. Except for the works in the first row, the others do not adhere to ETSI-MEC architecture.

7. Conclusions

The paper addressed the joint selection of the MEH and the path between the UE and the MEH. The paper found the solution to the addressed problem by using MDA through a proposed procedure to create a graph that is able to consider both network-layer and application-layer metrics. The performance of MDA has been evaluated by implementing a hybrid testbed, which is able to migrate a VideoLan application between two MEHs. The testbed is based on AdvantEDGE, which is able to simulate the UE mobility and the radio link, to emulate the network and the MEC APIs, and to experiment the VLC client. Moreover, the testbed includes K8s, which is used to support the migration of the VLC server pod between two actual MEHs. Finally, the paper added in the testbed a controller in order to integrate MDA with the 5G-MEC system. Two evaluations have been performed in the testbed: one with two MEHs; another one with only one MEH. In the evaluations, two network performance parameters (latency and throughput) and the user experience (MOS) have been considered. The results of the scenario with two MEHs demonstrate that MDA is able to perform the migration with a limited impact on the network performance and user experience. Instead, the results of the scenario with only one MEH emphasize that the absence of migration would result in a significant decline in the user experience.

CRediT authorship contribution statement

Prachi Vinod Wadatkar: Conceptualization, Software, Writing – original draft, Writing – review & editing. **Rosario Giuseppe Garroppo:** Conceptualization, Formal analysis, Supervision, Writing – original draft, Writing – review & editing. **Gianfranco Nencioni:** Conceptualization, Funding acquisition, Supervision, Writing – review & editing. **Marco Volpi:** Conceptualization, Software.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Gianfranco Nencioni reports financial support was provided by Research Council of Norway. Rosario G. Garroppo reports financial support was provided by Government of Italy Ministry of Education University and Research. Prachi V. Wadatkar reports financial support was provided by Research Council of Norway.

Data availability

No data was used for the research described in the article.

References

[1] NGMN, 5G white paper, 2015, https://ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf. (Accessed 18 November 2023).

[2] NGMN, 6G drivers and vision v.1.0, 2021, https://www.ngmn.org/wp-content/uploads/NGMN-6G-Drivers-and-Vision-V1.0_final.pdf. (Accessed 18 November 2023).

[3] 3GPP, 3GPP TS 22.261 V17.9.0, "3rd generation partnership project; technical specification group services and system aspects; service requirements for the 5G system; stage 1 (release 17), 2021.

[4] ETSI, MEC in 5G networks - white paper no. 28, 2018.

[5] K. Ha, Y. Abe, Z. Chen, W. Hu, B. Amos, P. Pillai, M. Satyanarayanan, Adaptive VM Handoff Across Cloudlets, Technical Report CMU-CS-15-113, Computer Science Department, Carnegie Mellon University, 2015.

[6] K. Ha, Y. Abe, T. Eiszler, Z. Chen, W. Hu, B. Amos, R. Upadhyaya, P. Pillai, M. Satyanarayanan, You can teach elephants to dance: Agile VM handoff for edge computing, in: Proceedings of the Second ACM/IEEE Symposium on Edge Computing, SEC '17, Association for Computing Machinery, New York, NY, USA, 2017, http://dx.doi.org/10.1145/3132211.3134453.

[7] M.A. Hathibelagal, R.G. Garroppo, G. Nencioni, Experimental comparison of migration strategies for MEC-assisted 5G-V2X applications, Comput. Commun. 197 (2023) 1–11, http://dx.doi.org/10.1016/j.comcom.2022.10.009, URL https://www.sciencedirect.com/science/article/pii/S0140366422003978.

[8] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, K.K. Leung, Dynamic service migration in mobile edge-clouds, in: 2015 IFIP Networking Conference, (IFIP Networking), 2015, pp. 1–9, http://dx.doi.org/10.1109/IFIPNetworking.2015.7145316.

[9] R.G. Garroppo, S. Giordano, L. Tavanti, A survey on multi-constrained optimal path computation: Exact and approximate algorithms, Comput. Netw. 54 (17) (2010) 3081–3107.

[10] P.M. de las Casas, A. Sedeño-Noda, R. Borndörfer, An improved multiobjective shortest path algorithm, Comput. Oper. Res. (2021) 105424.

[11] P.M. de las Casas, L. Kraus, A. Sedeño-Noda, R. Borndörfer, Targeted multiobjective dijkstra algorithm, 2021, arXiv:2110.10978.

[12] T. Kurbanov, M. Cuchý, J. Vokřínek, Fast one-to-many multicriteria shortest path search, IEEE Trans. Intell. Transp. Syst. (2023) 1–10, http://dx.doi.org/10.1109/TITS.2023.3282069.

[13] V. Farhadi, F. Mehmeti, T. He, T.F.L. Porta, H. Khamfroush, S. Wang, K.S. Chan, K. Poularakis, Service placement and request scheduling for data-intensive applications in edge clouds, IEEE/ACM Trans. Netw. 29 (2) (2021) 779–792, http://dx.doi.org/10.1109/TNET.2020.3048613.

[14] Z. Li, J.J. Garcia-Luna-Aceves, A distributed approach for multi-constrained path selection and routing optimization, in: Proceedings of the 3rd International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks, QShine '06, Association for Computing Machinery, New York, NY, USA, 2006, pp. 36–es, http://dx.doi.org/10.1145/1185373.1185420.

[15] InterDigitalInc, Advantedge on github, 2023, https://github.com/InterDigitalInc/AdvantEDGE/tree/1e63a66e8820f0882c998f1cbc6d200bcd14f41. (Accessed 18 November 2023).

[16] ETSI, GS MEC 013 V2.1.1: Multi-access edge computing (MEC); location API, 2019.

[17] ETSI, GS MEC 012 V2.1.1: Multi-access edge computing (MEC); radio network information API, 2019.

[18] ETSI, GS MEC 028 V2.1.1: Multi-access edge computing (MEC); WLAN information API , 2020.

[19] ETSI, GS MEC 011 V2.2.1: Multi-access edge computing (MEC); edge platform application enablement, 2020.

[20] ETSI, GS MEC 021 V2.1.1: Multi-access edge computing (MEC); application mobility service API, 2020.

[21] V.L. organization, Video LAN website, 2023, https://www.videolan.org/index.it.html. (Accessed 18 November 2023).

[22] SSU-DCN, Podmigration-operator, 2023, https://github.com/SSU-DCN/podmigration-operator/blob/main/init-cluster-containerd-CRIU.md. (Accessed 18 November 2023).

[23] ETSI, ETSI GS MEC 003 V2.2.1, Tech. rep., ETSI, 2020, (Accessed 18 November 2023).

[24] ISO, ISO/IEC 13818-1, 2023, https://www.iso.org/standard/75928.html. (Accessed 18 November 2023).

[25] T. Ödegaard, 2023, https://grafana.com/oss/grafana/. (Accessed 18 November 2023).

[26] ITU-T, Methods for objective and subjective assessment of speech and video quality - P.808 (06/21), 2023, https://www.itu.int/rec/T-REC-P.808-202106-I/en. (Accessed 18 November 2023).

[27] Wireshark, Wireshark website, 2023, https://www.wireshark.org/. (Accessed 18 November 2023).

[28] T.W. Nowak, M. Sepczuk, Z. Kotulski, W. Niewolski, R. Artych, K. Bocianiak, T. Osko, J.-P. Wary, Verticals in 5G MEC-use cases and security challenges, IEEE Access 9 (2021) 87251–87298, http://dx.doi.org/10.1109/ACCESS.2021.3088374.

[29] G. Nencioni, R.G. Garroppo, R.F. Olimid, 5G multi-access edge computing: A survey on security, dependability, and performance, IEEE Access 11 (2023) 63496–63533, http://dx.doi.org/10.1109/ACCESS.2023.3288334.

[30] Q. Pham, F. Fang, V.N. Ha, M.J. Piran, M. Le, L.B. Le, W. Hwang, Z. Ding, A survey of multi-access edge computing in 5G and beyond: Fundamentals, technology integration, and state-of-the-art, IEEE Access 8 (2020) 116974–117017, http://dx.doi.org/10.1109/ACCESS.2020.3001277.

[31] A. Sarah, G. Nencioni, M.M.I. Khan, Resource allocation in multi-access edge computing for 5G-and-beyond networks, Comput. Netw. 227 (2023) 109720, http://dx.doi.org/10.1016/j.comnet.2023.109720, URL https://www.sciencedirect.com/science/article/pii/S1389128623001652.

[32] J. Plachy, Z. Becvar, E.C. Strinati, Dynamic resource allocation exploiting mobility prediction in mobile edge computing, in: 2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications, (PIMRC), 2016, pp. 1–6, http://dx.doi.org/10.1109/PIMRC.2016.7794955.

[33] N. Sharghivand, L. Mashayekhy, W. Ma, S. Dustdar, Time-constrained service handoff for mobile edge computing in 5G, IEEE Trans. Serv. Comput. 16 (3) (2023) 2241–2253, http://dx.doi.org/10.1109/TSC.2022.3208783.

[34] I. Sarrigiannis, E. Kartsakli, K. Ramantas, A. Antonopoulos, C. Verikoukis, Application and network VNF migration in a MEC-enabled 5G architecture, in: 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, (CAMAD), 2018, pp. 1–6, http://dx.doi.org/10.1109/CAMAD.2018.8514943.

[35] A. Machen, S. Wang, K.K. Leung, B.J. Ko, T. Salonidis, Live service migration in mobile edge clouds, IEEE Wirel. Commun. 25 (1) (2018) 140–147, http://dx.doi.org/10.1109/MWC.2017.1700011.

[36] P. Fondo-Ferreiro, A. Estévez-Caldas, R. Pérez-Vaz, F. Gil-Castiñeira, F.J. González-Castaño, S. Rodríguez-García, X.R. Sousa-Vázquez, D. López, C. Guerrero, Seamless multi-access edge computing application handover experiments, in: 2021 IEEE 22nd International Conference on High Performance Switching and Routing, (HPSR), 2021, pp. 1–6, http://dx.doi.org/10.1109/HPSR52026.2021.9481834.

[37] P.V. Wadatkar, R.G. Garroppo, G. Nencioni, MEC application migration by using advantedge, in: S. Yu, B. Gu, Y. Qu, X. Wang (Eds.), Tools for Design, Implementation and Verification of Emerging Information Technologies, Springer Nature Switzerland, 2023, pp. 104–118.

[38] C. Campolo, A. Iera, A. Molinaro, G. Ruggeri, MEC support for 5G-v2x use cases through docker containers, in: 2019 IEEE Wireless Communications and Networking Conference, (WCNC), IEEE, 2019, pp. 1–6.

[39] F. Barbarulo, C. Puliafito, A. Virdis, E. Mingozzi, Extending ETSI MEC towards stateful application relocation based on container migration, in: 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks, (WoWMoM), IEEE, 2022, pp. 367–376.

[40] J. Schrettenbrunner, Migrating Pods in Kubernetes (Ph.D. thesis), 2020, http://dx.doi.org/10.13140/RG.2.2.31821.97762.

[41] P.S. Junior, D. Miorandi, G. Pierre, Good shepherds care for their cattle: Seamless pod migration in geo-distributed kubernetes, in: 2022 IEEE 6th International Conference on Fog and Edge Computing, (ICFEC), 2022, pp. 26–33, http://dx.doi.org/10.1109/ICFEC54809.2022.00011.

[42] M.-N. Tran, X.T. Vu, Y. Kim, Proactive stateful fault-tolerant system for kubernetes containerized services, IEEE Access 10 (2022) 102181–102194, http://dx.doi.org/10.1109/ACCESS.2022.3209257.

[43] CRIU, A project to implement checkpoint/restore functionality for linux, 2023, https://github.com/checkpoint-restore/criu. (Accessed 18 November 2023).

[44] J. Schrettenbrunner, Containerd-cri, 2023, https://github.com/schrej/containerd-cri. (Accessed 18 November 2023).

[45] S.D.A. Shah, M.A. Gregory, S. Li, R.D.R. Fontes, SDN enhanced multi-access edge computing (MEC) for E2E mobility and QoS management, IEEE Access 8 (2020) 77459–77469.

[46] S.D.A. Shah, M.A. Gregory, S. Li, R.d.R. Fontes, L. Hou, SDN-based service mobility management in MEC-enabled 5G and beyond vehicular networks, IEEE Internet Things J. 9 (15) (2022) 13425–13442, http://dx.doi.org/10.1109/JIOT.2022.3142157.

[47] P. Fondo-Ferreiro, D. Candal-Ventureira, F. Gil-Castiñeira, F.J. González-Castaño, D. Collins, Experimental evaluation of end-to-end flow latency reduction in softwarized cellular networks through dynamic multi-access edge computing, in: 2021 IEEE 32nd Annual International Symposium on Personal, Indoor and Mobile Radio Communications, (PIMRC), 2021, pp. 1310–1315, http://dx.doi.org/10.1109/PIMRC50174.2021.9569590.

[48] C.A. of Information, C.T. (CAICT), et al., Edge native technical architecture white paper, 2021, https://www-file.huawei.com/-/media/corporate/pdf/news/edge%20native%20technical%20architecture%20white%20paper.pdf?la=en.

[49] Edgegallery, 2019–2021, URL https://github.com/edgegallery. (Accessed 18 November 2023), GitHub.

[50] I.D. Martínez-Casanueva, L. Bellido, C.M. Lentisco, D. Fernández, An initial approach to a multi-access edge computing reference architecture implementation using kubernetes, in: H. Gao, R. J. Durán Barroso, P. Shanchen, R. Li (Eds.), Broadband Communications, Networks, and Systems, Springer International Publishing, Cham, 2021, pp. 185–193.

[51] D. Sabella, V. Sukhomlinov, L. Trang, S. Kekki, P. Paglierani, R. Rossbach, X. Li, Y. Fang, D. Druta, F. Giust, L. Cominardi, W. Featherstone, B. Pike, S. Hadad, Developing Software for Multi-Access Edge Computing, White Paper No. 20, ETSI, 2019, URL https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp20ed2_MEC_SoftwareDevelopment.pdf.

[52] D. Sabella, A. Li, H. Lee, L. Cominardi, Q. Huang, E. Pateromichelakis, V. Kashyap, C. Costa, F. Granelli, W. Featherstone, F. Naim, X. Yang, H. Ding, S. Nadathur, L. Chen, D. Druta, Q. Tang, B. Gazda, G. Chen, MEC Support Towards Edge Native Design, White Paper No. 55, ETSI, 2023, URL https://www.etsi.org/images/files/ETSIWhitePapers/ETSI-WP55-MEC_support_towards_Edge_native.pdf.

[53] P. Escaleira, M. Mota, D. Gomes, J.P. Barraca, R.L. Aguiar, Multi-access edge computing as a service, in: Proceedings of the 18th International Conference on Network and Service Management, CNSM '22, International Federation for Information Processing, Laxenburg, AUT, 2023.

[54] D. Sabella, A. Alleman, E. Liao, M. Filippou, Z. Ding, L.G. Baltar, S. Srikanteswara, K. Bhuyan, O. Oyman, G. Schatzberg, N. Oliver, N. Smith, S.D. Mishra, P. Thakkar, S. Shailendra, Edge Computing: From Standard to Actual Infrastructure Deployment and Software Development, White paper, Intel, 2019, URL https://networkbuilders.intel.com/docs/networkbuilders/edge-computing-from-standard-to-actual-infrastructure-deployment-and-software-development.pdf, (revised August 2021).

[55] N. Slamnik-Kriještorac, M. Peeters, S. Latré, J.M. Marquez-Barja, Analyzing the impact of VIM systems over the MEC management and orchestration in vehicular communications, in: 2020 29th International Conference on Computer Communications and Networks, (ICCCN), 2020, pp. 1–6, http://dx.doi.org/10.1109/ICCCN49398.2020.9209636.

[56] S. Barrachina, M. Payaró, P. Kokkinos, P. Soumplis, K. Kontodimas, E. Varvarigos, J. Vardakas, R. González, G. Siracusano, D. Sanvito, R. Bifulco, Deliverable D4.2: Initial report on elastic MEC platform design and data-driven orchestration and automation, Deliverable D4.2, MARSAL Project, 2022, Grant Agreement No. 101017171, Acronym: MARSAL, Full Title: Machine learning-based, networking and computing infrastructure resource management of 5G and beyond intelligent networks, URL http://www.marsalproject.eu.

[57] S. Bolettieri, D.T. Bui, R. Bruno, Towards end-to-end application slicing in multi-access edge computing systems: Architecture discussion and proof-of-concept, Future Gener. Comput. Syst. 136 (2022) 110–127, http://dx.doi.org/10.1016/j.future.2022.05.027, URL https://www.sciencedirect.com/science/article/pii/S0167739X22001984.

[58] T.K. Rodrigues, K. Suto, N. Kato, Edge cloud server deployment with transmission power control through machine learning for 6G internet of things, IEEE Trans. Emerg. Top. Comput. 9 (4) (2021) 2099–2108, http://dx.doi.org/10.1109/TETC.2019.2963091.

[59] K. Cao, L. Li, Y. Cui, T. Wei, S. Hu, Exploring placement of heterogeneous edge servers for response time minimization in mobile edge-cloud computing, IEEE Trans. Ind. Inform. 17 (1) (2021) 494–503, http://dx.doi.org/10.1109/TII.2020.2975897.

[60] T.V. Doan, G.T. Nguyen, M. Reisslein, F.H.P. Fitzek, FAST: Flexible and low-latency state transfer in mobile edge computing, IEEE Access 9 (2021) 115315–115334, http://dx.doi.org/10.1109/ACCESS.2021.3105583.

[61] M.R. Anwar, S. Wang, M.F. Akram, S. Raza, S. Mahmood, 5G-enabled MEC: A distributed traffic steering for seamless service migration of internet of vehicles, IEEE Internet Things J. 9 (1) (2022) 648–661, http://dx.doi.org/10.1109/JIOT.2021.3084912.

[62] C. Ayimba, M. Segata, P. Casari, V. Mancuso, Driving under influence: Robust controller migration for MEC-enabled platooning, Comput. Commun. 194 (2022) 135–147, http://dx.doi.org/10.1016/j.comcom.2022.07.014, URL https://www.sciencedirect.com/science/article/pii/S0140366422002626.

[63] C. Liu, F. Tang, Y. Hu, K. Li, Z. Tang, K. Li, Distributed task migration optimization in MEC by extending multi-agent deep reinforcement learning approach, IEEE Trans. Parallel Distrib. Syst. 32 (7) (2021) 1603–1614, http://dx.doi.org/10.1109/TPDS.2020.3046737.
[64] K. Poularakis, J. Llorca, A.M. Tulino, I. Taylor, L. Tassiulas, Service placement and request routing in MEC networks with storage, computation, and communication constraints, IEEE/ACM Trans. Netw. 28 (3) (2020) 1047–1060, http://dx.doi.org/10.1109/TNET.2020.2980175.
[65] J. Fang, G. Zhao, H. Xu, H. Tu, H. Wang, Reveal: Robustness-aware VNF placement and request scheduling in edge clouds, Comput. Netw. (2023) 109882.
[66] Z. Xiang, Y. Zheng, Z. Zheng, S. Deng, M. Guo, S. Dustdar, Cost-effective traffic scheduling and resource allocation for edge service provisioning, IEEE/ACM Trans. Netw. (2023).
[67] Z. Lei, H. Xu, L. Huang, Z. Meng, Joint service placement and request scheduling for multi-SP mobile edge computing network, in: 2020 IEEE 26th International Conference on Parallel and Distributed Systems, (ICPADS), 2020, pp. 27–34, http://dx.doi.org/10.1109/ICPADS51040.2020.00014.
[68] W. Chu, X. Jia, Z. Yu, J.C. Lui, Y. Lin, Joint service caching, resource allocation and task offloading for MEC-based networks: A multi-layer optimization approach, IEEE Trans. Mob. Comput. (2023).
[69] A. Gohar, G. Nencioni, Minimizing the cost of 5G network slice broker, in: IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops, (INFOCOM WKSHPS), 2021, pp. 1–6, http://dx.doi.org/10.1109/INFOCOMWKSHPS51825.2021.9484590.
[70] A. Sarah, G. Nencioni, Resource allocation for cost minimization of a slice broker in a 5G-mec scenario, Comput. Commun. (2023) in press.
[71] F. Mason, G. Nencioni, A. Zanella, A multi-agent reinforcement learning architecture for network slicing orchestration, in: 2021 19th Mediterranean Communication and Computer Networking Conference, (MedComNet), 2021, pp. 1–8, http://dx.doi.org/10.1109/MedComNet52149.2021.9501279.
[72] F. Mason, G. Nencioni, A. Zanella, Using distributed reinforcement learning for resource orchestration in a network slicing scenario, IEEE/ACM Trans. Netw. 31 (1) (2023) 88–102, http://dx.doi.org/10.1109/TNET.2022.3187310.

Prachi V. Wadatkar is a Ph.D. student with the University of Stavanger, Norway, and with the University of Pisa, Italy, from 2021. She received the M.Tech. degree in Communication Networks and Software and B.Tech. degree in Electronics and Telecommunication from the University of Pune in 2019 and 2016 respectively. She was a TRIL Research Fellow in International Center of Theoretical Physics, Trieste, Italy from 2019 to 2020. Her research activity includes emerging networking technologies (e.g. Multi-access Edge Computing, 5G , NFV). Her past research activity has been focused on low power wide area network technologies (e.g. LoRa and Sigfox).

Rosario G. Garroppo is Associate Professor at the Dipartimento di Ingegneria dell'Informazione of the University of Pisa. His expertise is on networking and his main research activities are focused on experimental measurements and traffic modeling in broadband and wireless networks, MoIP systems, traffic control techniques for multimedia services in wireless networks, network optimization, and green networking. On these topics, he has published more than 100 peer-reviewed papers in international journal and conference proceedings, and won a Best Paper Award at the 4th International Workshop on Green Communications (2011). He served as Technical Program Committee member of several international conferences on wireless networks, and as referee for several international journals and conferences. He was co-creator and co-organizer of the international IEEE Workshop on advanced EXPerimental activities ON WIRELESS networks and systems (EXPONWIRELESS), held in conjunction with IEEE WoWMoM since 2006 until 2009.

Gianfranco Nencioni is Associate Professor with the University of Stavanger, Norway, from 2018. He is received the M.Sc. degree in telecommunication engineering and the Ph.D. degree in information engineering from the University of Pisa, Italy, in 2008 and 2012, respectively. In 2011, he was a visiting Ph.D. student with the Computer Laboratory, University of Cambridge, U.K. He was a Post-Doctoral Fellow with the University of Pisa from 2012 to 2015 and the Norwegian University of Science and Technology, Norway, from 2015 to 2018. He is currently the head of the Computer Networks (ComNet) research group and leader of the 5G-MODaNeI project funded by the Norwegian Research Council. His research activity regards modeling and optimization in emerging networking technologies (e.g., SDN, NFV, 5G, Network Slicing, Multi-access Edge Computing). His past research activity has been focused on energy-aware routing and design in both wired and wireless networks and on dependability of SDN and NFV.

Marco Volpi received the M.Sc. degree in telecommunication engineering from the University of Pisa, Italy, in 2021.