A Novel Request Handler Algorithm for Multi-access Edge Computing Platforms in 5G

Gayan Dilanka*, Lakshan Viranga†, Rajitha Pamudith ‡, Tharindu D. Gamage§, Pasika Ranaweera¶
Indika A. M. Balapuwaduge‖, Madhusanka Liyanage**
*†‡§¶‖Department of Electrical and Information Engineering, University of Ruhuna, Galle, Sri Lanka
¶**School of Computer Science, University College Dublin, Ireland
**Centre for Wireless Communications, University of Oulu, Finland
Email: *gayandilankak@gmail.com, †lakshanviranga0000@gmail.com, ‡pamudithrajitha467@gmail.com, §tharindu@eie.ruh.ac.lk
¶pasika.ranaweera@ucdconnect.ie, ‖mendis@eie.ruh.ac.lk, **madhusanka@ucd.ie, **madhusanka.liyanage@oulu.fi

***Abstract*—Multi-access Edge Computing (MEC) is envisaging a storage and processing infrastructure at the edge of the mobile network to guarantee ultra-low latency and higher bandwidths for the provisioning services emanated by Internet of Things (IoT) devices. To achieve these dynamic requirements, MEC is adopting virtualization technologies that form a cost effective automated infrastructure ideal for 5G and beyond networks. Orchestration is the paramount task of such virtual platforms to manage and control the virtual entities autonomously. Service request handling is one such key orchestration function that handles the incoming requests to the orchestrator in case of a service initiation. However, existing service request handling procedures in MEC are still in a trivial stage. Thus, this paper proposes an advanced service request handling strategy for MEC orchestrator which can consider several factors such as service priority levels, feasibility, and resource availability. The performance of the proposed strategy is analyzed in a simulated environment and its feasibility is demonstrated using a prototype MEC infrastructure.**

***Index Terms*—Edge computing, MEC, Orchestration, Request handling, Virtualization**

I. Introduction

With the continuously growing demand for mobile services, the survival of the network service providers is reliant on their ability to develop a network infrastructure capable of launching the emerging use cases of the imminent 5^{th} Generation (5G) mobile networks. Though, launching these emerging use cases such as autonomous vehicles, Unmanned Aerial Vehicles (UAV), or Augmented Reality (AR) require an autonomous and dynamic service provisioning infrastructure. Cloud computing offers the service providers the opportunity to integrate these novel requirements to support enterprise-level and public customers in launching high-end services. Employing the cloud infrastructure as a service is having the drawbacks of unsolvable latency, bottlenecks in to ingressing traffic routes and locational unawareness due to its centralized nature [1]. Consequently, the concept of edge computing came on to the field of mobile communications to overcome these prescribed limitations.

Considering the use cases and applications of MEC, existing applications are enhanced while a wide range of novel and innovative applications are enabled with necessary authorizations from third parties to utilize local services and caching capabilities [2]. Orchestration is the function of auto-configuring, managing, coordinating and monitoring the virtualized service instances deployed in a virtual platform. The orchestrator has the visibility over the resources and capabilities of the holistic mobile edge network including a catalog of available applications. In the MEC system, Mobile Edge Orchestrator (MEO) is performing the holistic orchestration while Mobile Edge Platform Manager (MEPM) is managing the edge infrastructure. In the MEC perspective, there are various challenges and issues for reusing the existing orchestration tools or strategies due to its extremely dynamic nature and the requirement for complete automation [3], [4].

This paper proposes a novel service request handling approach based on request priority and resource availability of the edge platform, that leads to optimal orchestration of virtual and physical resources. The performance of the proposed mechanism is evaluated with both simulations and an emulated test-bed, that was developed in accordance to the proposed orchestration architecture specified in Section III. The rest of the paper is structured as follows. State of the art advancements related to the research directive is explicated in Section II. Section III presents the proposed orchestrator and its functionality while the proposed request handling strategy is specified in Section IV. Section V presents the simulation results while in Section VI, we have shown the implementation of this proposed algorithm on our developed virtual platform. Finally, the paper is concluded in Section VII.

II. Related Work

There is a need for a well-defined, comprehensive architecture to provide cloud infrastructure at the edge of the mobile network which can be easily updated, modifiable, and maintainable. Due to the heterogeneous nature of the different cloud-based services, provision of services at the edge of the mobile network has been challenging since it needs the

consideration of all the aspects of multi-tenancy support and networking. A deep investigation on the different existing orchestrators and their orchestration scope is done in [3]. The orchestration objectives also differ in each orchestrator such as Multi-Cloud solution for automating App deployment and Network Function as a service. In [5], authors depicts the network services and resources orchestration for vehicular communication by considering the requirements of 5G communication networks such as ultra-low-latency and higher capacity accompanied with the enormous requirements of the cloud, service providers and vendors. The authors in [5] have evaluated several open source orchestration tools and their performances against several parameters, specially against the latency. Even-though there are several orchestrators has been developed, still the infrastructure automation and dynamic resource allocation has not been approached.

Designing proper orchestration strategies and algorithms is critical in the scope of attaining ultra-low latency standards with higher bandwidth utilization. And also the technology and standards used for designing phase affect to the complete orchestration development process. In [6], authors provide a general orchestration architecture with the functional modules. The flow of the service request is also specified so that the the request will be initially processed at the edge of the mobile network. But in here the initial request handling refers to checking the network service catalogue for the service availability at the edge and allocating the resources. Therefore specific methodology related to resources availability and feasibility checking at the edge of the mobile network is not considered in the request handling process. In [7], authors develop a mathematical model to depict the request routing in the multi access edge computing. Therein, it considers the arrival of the requests to the systems as a stochastic process and then consider about the computational power and the bandwidth requirement for service placement. But it does not consider any orchestration process or other factors such as Service Level Agreements (SLAs) to initiate a service. This is a lagging point here and in our case we have included the orchestration process with the above mentioned features. In the paper [8], authors consider about the service placement in the edge of the mobile network by considering the mobility of the users. In the developed system model they have considered the Computational capacity and the feasibility factors in the edge servers. And [9] also considers the migration of the services from the cloud to the edge, and they have highly considered about the maximum computational power utilization along with services prioritization to have a faster migration process. Table I, compares the above mentioned request handling processes in edge computing based on the Service prioritization, Resource check and Feasibility check. Here Y indicates feature is available and N indicates not available.It can be seen that in our algorithm all the features have been achieved.

III. Proposed MEC Orchestrator

We propose a general architecture for the MEO, to streamline the service delivery at the edge. Therefore, the main attributes of the proposed MEC service orchestration are as follows,

Table I
Comparison of various features of our proposed request handling approach with existing request handling methods

Feature	[7]	[8]	[9]	Ours
Service Prioritization	N	N	Y	Y
Resources Check	Y	Y	Y	Y
Feasibility Check	N	Y	N	Y

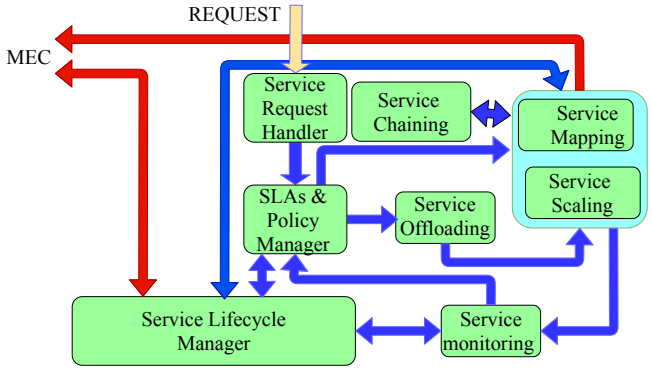

Figure 1. Proposed Internal Architecture of the MEO.

- **Resource allocation:** Compute the available resources such as CPU, memory, storage, and network bandwidth along with network altercation.
- **Service placement:** Placing of services in MEC servers to provide maximum QoE to the users.
- **Edge selection:** Allocating the proximate and resource enabled MEC server that offers the requested service in order to provision highest performance.

The composition of the proposed MEO is illustrated in Fig. 1. The MEO consists of eight components and the role of each component is mentioned below.

- **Service Request Handler** - This entity is responsible for handling the requests prior to service instigation.
- **SLA and Policy Manager** - SLAs and Policies will be checked and accordingly, services will be altered against SLAs.
- **Service Chaining and Service Mapping modules** - This module will be able to design/create service chains and map the created services to the available resources in the MEC by taking the information exposed from local repositories about the existing resources and virtual network functions.
- **Service Life-cycle Manager** - Instantiating, modification, operation, and termination of the service life-cycle is handled by this module.
- **Services Monitoring Module** - This will continuously monitor the services during the service life-time. E.g. monitor performance and security level of the service.
- **Service Scaling module** - This module performs the optimal scaling of the resources against dynamic demands from the users after service initiation.

IV. Proposed Request Handling Strategy

Service initiation in the orchestrator is conducted via three stages: service request, service provisioning, and service activation [6]. Once a service request arrived to the orchestrator, security and feasibility checks are followed in accordance to the prescribed policies. The processing of the request will continue if the service request passes these checks. Otherwise, the request will be dropped. The proposed strategy is based on the verification of resource availability and service priority level.

A. Orchestrator on the Edge Server Platform

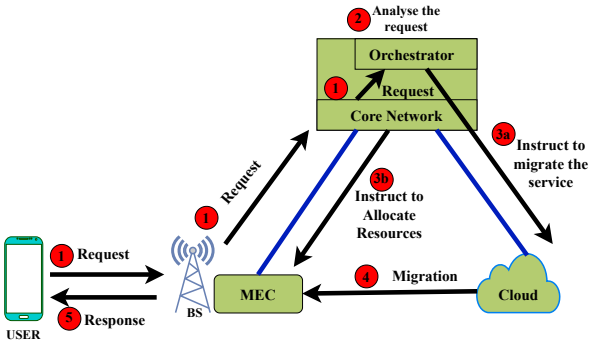

Figure 2. Formation of the Request Handling Process.

As depicted Fig. 2, The service request is sent from the user device. This would reach the orchestrator via the service provider network, and would be subjected to analysis through inspection. Then, a resource availability check, and a feasibility check would be performed to instigate the request. If the service can be provisioned at the MEC edge, instructions would conveyed towards the cloud for migrating the service. Concurrently, instructions would be sent towards the MEC edge to allocate the resources (i.e. CPU, RAM, and storage). Thus, service will be migrated from the cloud to the MEC. Finally, a service confirmation request is sent to the user from the MEC.

B. Proposed Service Request Handler Algorithm

Fig. 3 depicts the proposed service request handling process in a diagrammatic manner. The process of each block is explained below. The numbering corresponds to the each processing block.

1) This block will put all the incoming requests to a queue in order to select the request to be processed.
2) This block will select the requests to be started first based on a specific selection method. These methods are,
   a) First In First Out (FIFO).
   b) Preparing queue according to the service priority and take the highest priority service to migrate first (i.e. service weights according to its priority).
3) Resource availability of the MEC will be checked here. When checking the resources it will filter the MEC servers which has minimum remaining resources, and

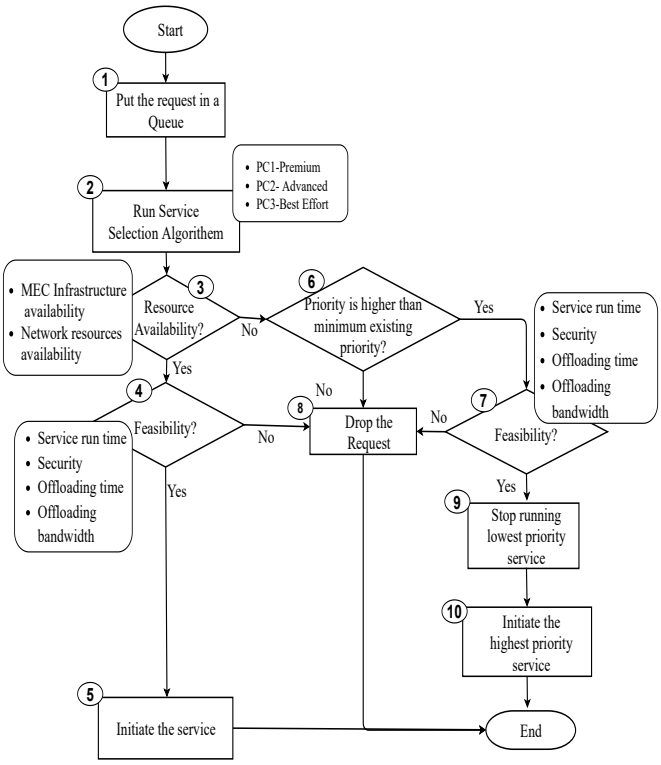

Figure 3. The Service Request Handling Process.

the remaining amount is sufficient to initiate corresponding services. Considered factors on resource availability are,

   a) MEC infrastructure:- CPU/ RAM / Storage.
   b) Network resources:- Inspects available bandwidth. If the MEC edge has adequate resources to cater the request, it will be forwarded to block 4; else it will be passed to block 6.
4) The feasibility of launching the request in terms of service run time, security, offloading time and bandwidth, and migration cost is evaluated. If the request is feasible, it will succeed to block 5.
5) The service will be migrated from the cloud to MEC edge.
6) The request will be passed to this block if the resources are insufficient in the MEC to migrate the service. In this block, the priority level of the request will be inspected. If the priority level exceeds the minimum priority level of the serving service instances, it will be passed to block 7. Unless, the request will be dropped.
7) Feasibility factors similar to process block 4 will be checked here to initiate the high priority service.
   a) Security : Here it compares the security level of the arriving request and the maximum security level that can be provided by the MEC. Let the required security level of the service be S_i and the maximum security that can be provided by MEC be S_{max}: if $S_{max} > S_i$,
   then security feasibility check is passed,

Algorithm 1: Service Selection Algorithm

```
Initialization;
Requiring: Q, a Queue
Requiring: S, a new service
Requiring: R, an incoming request
while true do
    instructions;
    if R is arrived then
        Update the priority of all requests in Q;
        Add R in Q;
        Order the Q based on the priority of the requests;
    else
        Discard Request;
    end
    if Q is not empty then
        Rp = Highest priority request in the Q;
        Add R in Q;
        Order all the requests Q according to their priority;
        if Rp can be initiated then
            Remove Rp from Q;
            Initiate Rp;
        else
            Discard Request;
        end
    else
        Discard Request;
    end
end
```

else security feasibility check fails.

b) Migration cost : The cost is evaluated for launching the service, either in the MEC or in the cloud.

8) This block will drop the requests when there is a false of checks.
9) A lowest priority service will be terminated to cater the service at play.
10) The requested service will be initiated to be launched.

V. Simulation Results

The Algorithm 1 was developed, and was evaluated via custom-built simulations using the discrete events implemented in MATLAB. To simulate the request handling process, we define an arrival process following a Poisson process with the parameter λ, which is the mean arrival rate. Moreover, the service times for services are exponentially distributed, with mean service rate μ. In order to bind the RAM, CPU, and security requirements of each and every request, we randomly generate the requirements and bonded them with the corresponding request.

In this simulation model, the user requests are equivalent to the requests arriving at the orchestrator which follows the Poisson process described previously. Once an arrival event happens (a service request arrives), all the service requirements and the time instants of the arrival are recorded in a matrix $\mathcal{A}$, including the requirement of the CPU, RAM and other requirements. A row of this generated matrix represents a service request arrived at a time instant. Another matrix, $\mathcal{M}$, is generated to represent the status of MEC server platform at a particular time instant. One row of the matrix $\mathcal{M}$ represents one server in the MEC platform. Different columns of the matrix $\mathcal{M}$ indicate the server occupancy status, total RAM, total CPU, available RAM, available CPU, and remaining time for the running services. When the simulation is running, $\mathcal{M}$ is continuously updated according to the arrivals and departures of services. During the simulation run, a variable is set to count the number of request arrivals as well as the number of requests successfully served by the MEC platform. Accordingly, the blocking probability of service requests denoted as P_B, is defined as,

$$P_B = \frac{N_{block}}{N_{tot}} \tag{1}$$

where N_{block} and N_{tot} denote the total number of blocked requests and the total number of service requests respectively. A blocking of a request means that it will not be served at a MEC server. Lower the blocking probability, higher a request being served at a MEC server. Here we have considered four cases to evaluate the performance of our simulation in terms of the blocking probability. Simultaneously, three other simulation were developed corresponding to the existing request handling processes. In one simulation it does only checking of the resources availability on a randomly selected MEC server. If there are enough resources service will be migrated to that MEC server. Another set of simulations are conducted to evaluate the scenario where we migrate the service to the MEC server which has the highest resources availability. The next simulations is performed to investigate the scenario when the migration of the service to the MEC server which has the higher feasibility. For each Case the algorithms were executed 50 times and the average of the observed results were used for the graphical representation. Table II contains the simulation parameters corresponding to the each case. Experiment duration of every case is set to 1000s

Table II
Importance of Simulation parameters

	Simulation Parameters		
Experiment Scenarios	λ	μ	MECs
Case 1: Impact of Arrival Rate (λ)	Variable	25	16
Case 2: Impact of Number of MECs	80	25	Variable
Case 3: Impact of Service Rate (μ)	80	Variable	16
Case 4: Impact of Priority	80	25	16

the requests with accumulated service run time. In this case arrival rate and the number of MEC servers was kept constant.

In Fig. 4, P_B corresponds to the results of Case 1. Moreover, results for Case 2 and Case 3 are shown in Fig. 5. In Fig. 6, in relation to Case 1, it can be seen that blocking probability of service provision at the edge is increasing when the arrival rate of the requests increases. The proposed algorithm has the lowest blocking probability compared to other algorithms. The reason for having a lower probability in the proposed algorithm

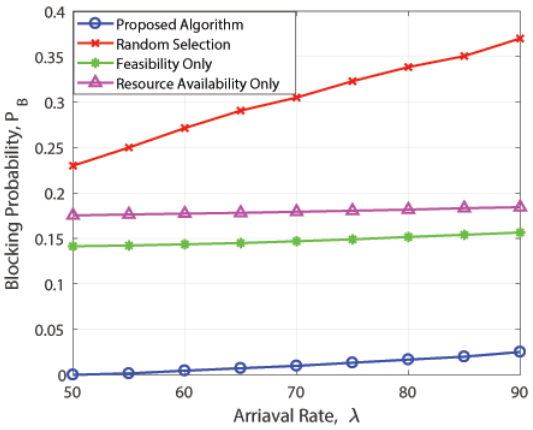

Figure 4. Blocking Probability vs Arrival Rate.

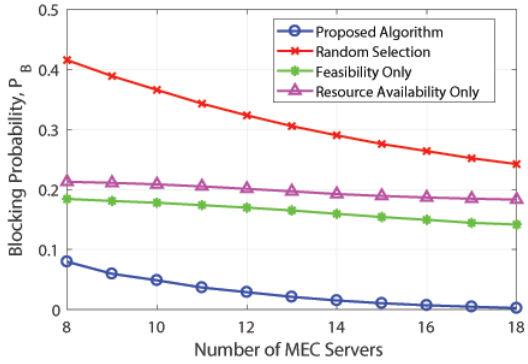

Figure 5. Blocking Probability vs Number of MEC Servers.

is due to the consideration of all the factors as shown in Fig. 3, and allocating computational resources in an optimal way as mentioned in the subsection IV-B.

In Case 2, blocking probability decreases when the number of MEC servers increases. It is observable that the blocking probability decreasing continuously in the proposed algorithm when the number of MEC servers increase. The reason for this result is when increasing the number of MEC servers, the computational resources in the edge increase. Thus the probability of migrating to the edge increases. And the proposed algorithm has the minimum blocking probability because when the computation resource pool in the edge increases serving a request in the edge is higher due to the allocation of resources in an optimal way.

Blocking probability decreases in Case 3 with increasing service rate of the MEC servers. Similarly, proposed algorithm has a lower blocking probability outperforming to other algorithms. The reason for this is, due to considering all the factors and optimum resource utilization, the number of requests that can be served in a unit time is higher compared to other algorithms. As the case 4 importance of the service prioritization was checked by considering the behaviour of the proposed algorithm and the existing algorithms. Here three priority levels were used and the blocking probability of the each level were checked corresponding to the each algorithm. Table III shows the resultant blocking probabilities and it

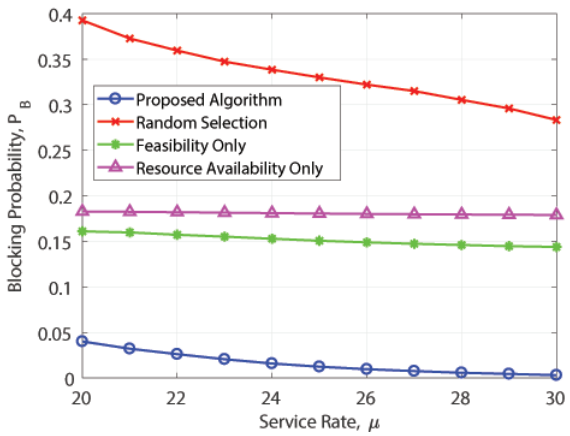

Figure 6. Blocking Probability vs Service Rate μ_P.

Table III
IMPORTANCE OF SERVICE PRIORITIZATION

	Priority Level		
Scenario	High	Medium	Low
Our	0.0390	0.0563	0.0783
Feasibility Only	0.1824	0.1815	0.1851
Resource Only	0.2117	0.2154	0.2125
Priority	0.0609	0.0615	0.0636
Random	0.4194	0.4168	0.4214

can be seen that proposed algorithm has the lowest blocking probability for each priority level because of optimum resource utilization.

Considering these results, it is observable that the developed algorithm performs effectively, when increasing the request arrival rate. And when increasing the number of MEC servers, and average serving time of the MEC servers, P_B is exponentially alleviating.

VI. IMPLEMENTATION

The proposed MEO architecture was implemented in a testbed environment employing VMware ESXI bare metal hyper-visor; as our virtual hardware infrastructure. Docker was used to develop the MEC edge infrastructure, while services were launched as docker containers. For establishing the communication among the entities: orchestrator, MEC server platform, user, and the cloud, we employed socket communication. The orchestrator, cloud, and the users were implemented as Virtual Machines (VMs). Here VMware ESXI environment 6.5 was used with Ubuntu 6.5 Virtual Machines. Docker 19.03 was used for running services as containers and python based socket programming was used to connect the Virtual Machines.

User requests are generated using a python script and requirements of CPU, RAM, Security level, and other requirements are bounded with the request. Once a user sends a request, it follows the flow depicted in Fig. 2 and the orchestrator runs the algorithm until a response will be sent to the user. Fig. 7 illustrates the placement of the entities

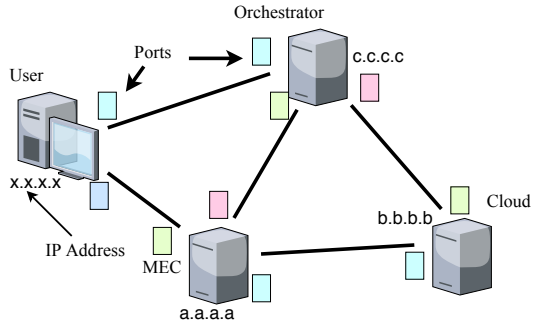

Figure 7. Socket based Connectivity Origination and Termination.

(User, Orchestrator, MEC, Cloud) in the emulated virtual environment. When a transaction occurs between two entities, they follow the client-server model. As shown in the protocol diagram in Fig. 8, user connects with the orchestrator using an IP address and a port number. Then the interaction between the orchestrator, cloud, and the MEC conducts until a response has been sent; regardless of a service ready, or termination notification.

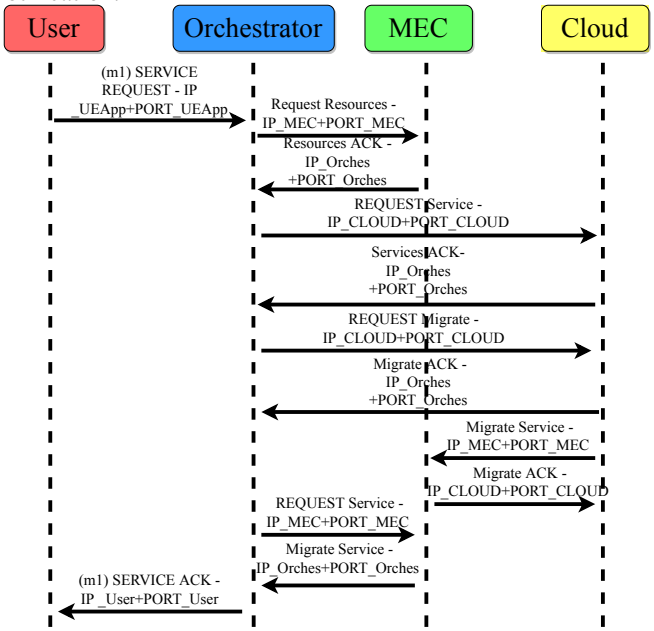

Figure 8. Proposed Protocol for User Request Handling.

According to our implementation, 1.7 ms is measured as the average response time for requests which has different priority levels, and infrastructure requirements, conveyed to the orchestrator. The average response time of Amazon EC2 and Microsoft Azure is 32 ms and 21 ms [10]. Therefore, our implementation measured time is comparatively lower than the response time of public cloud deployment.

VII. Conclusions and Future Work

In the conviction of providing ultra-low latency and better QoS to the end-users, telecommunication and networking service providers are trending to use more and more edge computing instead of cloud computing. Under this context, it is required to have an advanced orchestrators to manage the edge resources and dynamically deploy services at the edge for different IoT devices. In this paper, a novel request handler mechanism was proposed for such MEC orchestrators. The proposed mechanism considers the key factors such as resource availability at edge servers, feasibility (i.e. service run time, security, offloading parameters and migration cost) and the priority of the request, to handle the request. The simulation results verified that the developed algorithm performs effectively when increasing the request arrival rate and able to offer rights-of-way for high-priority requests. Moreover, the prototype implementation showed that the response time for a request conveyed to the orchestrator in 1.7 ms which comparatively lower than the response time of cloud computing.

In future, the proposed request handler will be further developed and integrated with other components of the MEC orchestrator to perform advanced tasks such as dynamically allocate the resources based on the demands of the IoT devices in the 5G systems.

Acknowledgement

This work is partly supported by Academy of Finland in 6Genesis (grant no. 318927) project.

References

[1] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on Multi-access Edge Computing for Internet of Things Realization," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2961–2991, 2018.

[2] F. Giust, V. Sciancalepore, D. Sabella, M. C. Filippou, S. Mangiante, W. Featherstone, and D. Munaretto, "Multi-access Edge Computing: The Driver Behind the Wheel of 5G-connected Cars," *IEEE Communications Standards Magazine*, vol. 2, no. 3, pp. 66–73, 2018.

[3] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On Multi-access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[4] P. Ranaweera, A. Jurcut, and M. Liyanage, "MEC-enabled 5G Use Cases: A Survey on Security Vulnerabilities and Countermeasures," *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–37, 2021.

[5] N. Slamnik-Kriještorac, E. Municio, H. C. Resende, S. A. Hadiwardoyo, J. M. Marquez-Barja *et al.*, "Network Service and Resource Orchestration: A Feature and Performance Analysis within the MEC-Enhanced Vehicular Network Context," *Sensors*, vol. 20, no. 14, p. 3852, 2020.

[6] S. Peng, J. O. Fajardo, P. S. Khodashenas, B. Blanco, F. Liberal, C. Ruiz, C. Turyagyenda, M. Wilson, and S. Vadgama, "QoE-oriented Mobile Edge Service Management Leveraging SDN and NFV," *Mobile Information Systems*, vol. 2017, 2017.

[7] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 10–18.

[8] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2333–2345, 2018.

[9] I. Sarrigiannis, E. Kartsakli, K. Ramantas, A. Antonopoulos, and 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)*. IEEE, 2018, pp. 1–6.

[10] G. T. Itichaa and T. F. Gedefa, "Selective Testing As A Service for Cloud Computing," *DOI:10.9790/0661-2103027085*, 2019.