Enhancing QoS for IoT Devices through Heuristics-based Computation Offloading in Multi-access Edge Computing

Marouane Myyara, Oussama Lagnfdi, Anouar Darif, and Abderrazak Farchane

***Abstract*—Multi-access Edge Computing (MEC) networks, particularly with the advent of 5G, aim to reduce latency and increase speed to meet the demands of resource-intensive applications in the Internet of Things (IoT), such as private wireless networks, online gaming, industry, and remote healthcare. These applications require guaranteed performance. However, while Quality of Service (QoS) management is well established in the Cloud, improving it remains a challenge in MEC en- vironments. This study addresses this challenge by proposing heuristic computation offloading algorithms for IoT-intensive devices in MEC networks. These algorithms aim to minimize service time while maximizing the QoS, taking into account tasks and resource characteristics to determine the optimal execution location for IoT device applications. We evaluated our approach using the EdgeCloudSim simulator, and the results demonstrate its superiority over existing solutions. Our approach significantly improves QoS by reducing the service time of IoT application tasks. This research fills a gap in efficient QoS improvement and contributes to advances in computation offloading strategies in MEC environments. It paves the way for enhanced performance of IoT applications in these networks.**

***Index Terms*—MEC, IoT, Computation Offloading, Quality of Service, Heuristic Algorithms, EdgeCloudSim.**

I. INTRODUCTION

The Internet of Things (IoT) is a rapidly expanding ecosystem of diverse physical objects connected through various networks, both wired and wireless [1]. It enhances internet utilization by linking mobile devices and sensors. However, IoT faces significant challenges, including high network latency, availability, and mobility, often mitigated through Cloud Computing [2]. Resource-constrained IoT devices struggle with limited processing power, memory, and battery life, complicating the execution of complex tasks. Additionally, the surge in IoT devices can overload networks accessing Cloud servers, hindering low-latency and high-capacity applications. To address these issues, edge computing paradigms, such as Mobile/Multi-access Edge Computing (MEC), have emerged.

MEC, introduced by the European Telecommunications Standards Institute (ETSI) [3], enhances edge intelligence and boosts processing and storage capabilities [4]. By bringing cloud functionalities closer to the Radio Access Network (RAN), it provides ultra-low latency and network context awareness. ETSI identifies IoT as a key use case for MEC [4], emphasizing the mutual benefits of their integration [2]. From the MEC perspective, IoT expands MEC services to various devices, while from the IoT viewpoint, MEC architecture offers computing resources closer to users. This integration significantly aids resource-constrained IoT devices by providing access to powerful computing at the network edge, enabling efficient task execution and improved service quality. According to [5], this integration provides three main benefits: reduced infrastructure traffic, lower application latency, and scalable network services. The key advantage is decreased latency through MEC, which shortens distances and transmission times between resources, facilitating efficient resource provision for processing IoT applications [6].

Despite growing interest in MEC for IoT, research on computation offloading remains limited. Recent studies focus on reducing latency in MEC networks [7], [8], [9], but offloading for resource-constrained IoT devices has received insufficient attention. Effective offloading strategies can lower latency, enhance service quality, and reduce reliance on centralized cloud systems. Optimizing MEC resources involves managing limited server capacity to minimize execution delays and improve user experience. Fair allocation mechanisms are essential due to the diverse interests of IoT users and edge servers. The dynamic nature of these systems highlights the need for ongoing research to develop robust resource allocation methods that maximize edge computing's potential and foster innovation.

This study aims to develop a heuristic-based offloading strategy that optimizes task execution time and improves QoS. This paper presents a novel heuristic-based offloading strategy that addresses the specific challenges faced by IoT devices. The paper's structure includes a review of related work (Section II), the system model and problem formulation (Section III), the proposed heuristic-based offloading strategy (Section IV), performance evaluation through simulation results (Section V), and a conclusion with future research perspectives (Section VI).

II. RELATED WORK

The integration of IoT with MEC provides significant benefits, such as ultra-low latency, real-time data analytics, improved resource management, increased capacity, and enhanced scalability. By localizing computing capabilities, MEC reduces bandwidth needs and reliance on central Clouds, minimizing data transfers to remote data centers.

M. Myyara, O. Lagnfdi, A. Darif, and A. Farchane are affiliated with the Laboratory of Innovation in Mathematics, Applications, and Information Technology, Polydisciplinary Faculty, Sultan Moulay Slimane University, Beni Mellal, 23000, Morocco. (E-mails: marouane.myyara@usms.ac.ma, lagnfdi.o@gmail.com, anouar.darif@gmail.com, a.farchane@gmail.com)

Manuscript received April 19, 2005; revised August 26, 2015.

DOI: 10.36244/ICJ.2024.4.2

Researchers have explored collaborative computation offloading and resource allocation schemes to enhance task processing efficiency in MEC systems. MEC facilitates efficient computation, load balancing, and latency reduction by migrating tasks to resource-rich infrastructures [10]. Recent studies include dynamic offloading algorithms that optimize user experience by minimizing service time and balancing workloads [11], and a deadline-aware scheduling algorithm that reduces execution time for critical tasks while considering task type and weight [12]. While many studies focus on optimizing task processing time, they often neglect the broader concept of service time, which encompasses both processing and transmission delays. This paper advances the field by introducing a novel algorithm that incorporates each task's deadline and latency tolerance to minimize service time while meeting QoS requirements.

Several research efforts have focused on optimizing computation offloading in MEC to enhance IoT device performance. One approach is the Lagrange duality resource optimization algorithm [13], which improves task offloading and resource allocation compared to traditional methods like random offloading and load balancing. This highlights the importance of efficient processing for real-time IoT applications, addressing service time and QoS requirements. A notable study [14] presents a collaborative computing framework that enables devices to partially process tasks across terminals, edge servers, and the Cloud using a pipeline-based offloading scheme. Additionally, various models have been developed to reduce latency and improve system efficiency, including an algorithm specifically designed to minimize execution latency [15].

Effective joint resource management between MEC and the central Cloud is crucial for meeting the service demands of IoT applications, particularly given the limited capabilities of edge devices compared to Cloud infrastructures [16]. Recent research has focused on MEC network workload orchestration and resource allocation strategies to improve IoT application performance [17]. For instance, [18] explores computation offloading and bandwidth distribution in IoT networks using graph-based models for resource optimization. Heuristic methods, such as the iterative heuristic mobile edge computing resource allocation algorithm [19], aim to enhance efficiency and minimize latency.

Despite advancements in MEC and IoT integration, challenges specific to resource-constrained IoT devices remain unaddressed. Our primary objective is to minimize task execution time while considering computing resource constraints and application requirements. Unlike prior studies, our research focuses on optimizing QoS for end IoT devices within an MEC framework. We propose a heuristic-based strategy for offloading and resource allocation that adheres to constraints while maximizing task execution efficiency, reducing latency, and effectively utilizing computing resources.

III. SYSTEM MODEL AND PROBLEM FORMULATION

A. System Model

The MEC system model, illustrated in Figure 1, features a three-tier architecture: central cloud, MEC servers, and wirelessly connected IoT devices. IoT devices can offload resource-intensive computations to MEC servers or the cloud for efficient processing. Each MEC server is linked to a wireless access point or base station, covering a specific area and serving IoT users. These servers, equipped with sufficient hardware resources, connect to the Edge Orchestrator (EO) via a backhaul link, which manages infrastructure resources, server states, and capacities. Positioned close to users, the servers connect to the EO through a MAN. The central cloud consists of high-capacity servers provided by cloud service providers, accessible via a WAN network linking the EO to the upper layer. The offloading process begins with the device and is guided by the EO, considering workload distribution, computing resources, and network conditions.

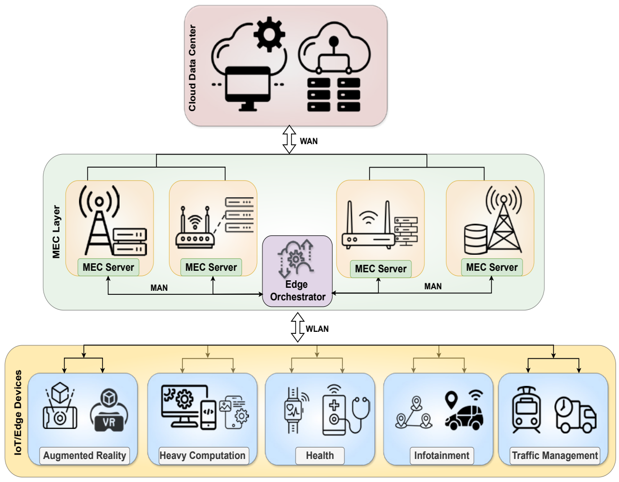

Fig. 1: Multi-layer Multi-access Edge Computing architecture.

B. Notation and Variables

Let $\mathcal{M}$ represent the set of MEC nodes, $\mathcal{C}$ denote the Central Cloud, and $\mathcal{D}$ the set of IoT devices, with IoT_i indicating device i. Each MEC node $m \in \mathcal{M}$ is associated with a set of Virtual Machines (VMs). The computing capacity of each VM, denoted by $\mathcal{F}$, is measured in MIPS (Millions of Instructions Per Second). Each IoT device i has one or more computation tasks, represented by $\mathcal{T}_i$, with each task $\tau_{i,j}$ characterized by a length $\mathcal{L}_{i,j}$, indicating the data generated for the task. The computational capacity required for task j from device i is $C_{i,j}$. These parameters are defined in an XML file in the EdgeCloudSim simulator [20], allowing for customization based on application characteristics.

C. Computation Offloading Model

In a multi-layer MEC environment, computation tasks can be executed locally on the IoT device, offloaded to MEC servers, or offloaded to central Cloud servers. The computation task offloading involves two sets of optimization variables:

- Binary Variable $\mathcal{A}$: This variable is defined as $\mathcal{A} = \{\alpha_{i,j} \mid i \in \mathcal{D}, j \in \mathcal{T}_i\}$. Here, $\alpha_{i,j}$ takes a value of 0 if task data $\tau_{i,j}$ is offloaded, and 1 if it is executed locally.

- Binary Variable $\mathcal{B}$: This variable is defined as $\mathcal{B} = \{\beta_{i,j} \mid i \in \mathcal{D}, j \in \mathcal{T}_i\}$. Here, $\beta_{i,j}$ equals 1 if the task is offloaded to an MEC server, and 0 if it is offloaded to the Cloud.

D. Task Computation Model

In this section, we focus on the task computation models in the MEC architecture, providing estimations for determining local processing times and remote processing times.

1) Local Computation: Local computation processes tasks directly on the user's IoT device, resulting in low latency since data does not need to be transferred to a remote server. However, if the task size is large or the device's computing capacity is limited, offloading may be necessary. Let $\mathcal{F}_{IoT_i}$ denote the computing power of the IoT device i in MIPS. The overall service time for local computation, denoted as $T^{IoT}_{\tau_{i,j}}$, can be calculated as:

$$T^{IoT}_{\tau_{i,j}} = \frac{\mathcal{L}_{i,j}}{\mathcal{F}_{IoT_i}} \tag{1}$$

The local computation delay depends on task size and computing power. Larger tasks or weaker computing power result in longer delays. Local computation has no transmission delays as data is processed on the user's IoT device.

2) Computation in MEC Servers: Computation in MEC servers leverages proximity to end-users to achieve low latency. Offloading tasks to nearby MEC servers typically reduces service time compared to local computation. However, task execution on MEC servers incurs delays due to wireless transmission from IoT devices. The total execution time on an MEC server includes both transmission and computation delays. Let $\mathcal{F}_{\text{MEC}_m}$ denote the computing power of the m-th MEC server. The execution time of task $\tau_{i,j}$ on MEC server m can be expressed as:

$$T^{m}_{\text{Com}(i,j)} = \frac{\mathcal{L}_{i,j}}{\mathcal{F}_{\text{MEC}_m}} \tag{2}$$

The total delay, considering both transmission and computation times, for transmitting the input data and receiving the computation results from the IoT layer to the MEC server through the wireless channel can be expressed as:

$$T^{\text{MEC}_m}_{\tau_{i,j}} = T^{m}_{\text{Com}(i,j)} + T^{m}_{\text{Up}(i,j)} + T^{m}_{\text{Dw}(i,j)} \tag{3}$$

where $T^{m}_{\text{Com}(i,j)}$ represents the computation time for task j on MEC server m, $T^{m}_{\text{Up}(i,j)}$ represents the upload data transfer time, and $T^{m}_{\text{Dw}(i,j)}$ represents the download data transfer time from the MEC server back to the IoT device.

3) Computation in Cloud Servers: Computation in Cloud servers offers high processing power and storage capacity but incurs higher latency due to the distance of data transfer. When MEC servers cannot process offloaded tasks promptly, they are sent to the Cloud server over the wireless network. The total delay in Cloud server computation consists of transmission and processing delays.

The computation time in the Cloud is given by:

$$T^{cloud}_{\text{Com}(i,j)} = \frac{\mathcal{L}_{i,j}}{\mathcal{F}_{\text{Cloud}}} \tag{4}$$

Where $\mathcal{F}_{\text{Cloud}}$ represents the computing power of the Cloud server in MIPS. Similar to the MEC server case, there is a transmission delay for uploading and downloading data, denoted as $T^{cloud}_{\text{Up}(i,j)}$ and $T^{cloud}_{\text{Dw}(i,j)}$ respectively. The total delay for offloading a task to the Cloud server is the sum of the computation time and the transmission delay:

$$T^{\text{Cloud}}_{\tau_{i,j}} = T^{cloud}_{\text{Com}(i,j)} + T^{cloud}_{\text{Up}(i,j)} + T^{cloud}_{\text{Dw}(i,j)} \tag{5}$$

Figure 2 illustrates task computation models for local computation, MEC servers, and Cloud servers, indicating task offloading and allocation in the MEC architecture. It's important to note that transmission delay in the Cloud server case is typically longer due to distance and potential network congestion.

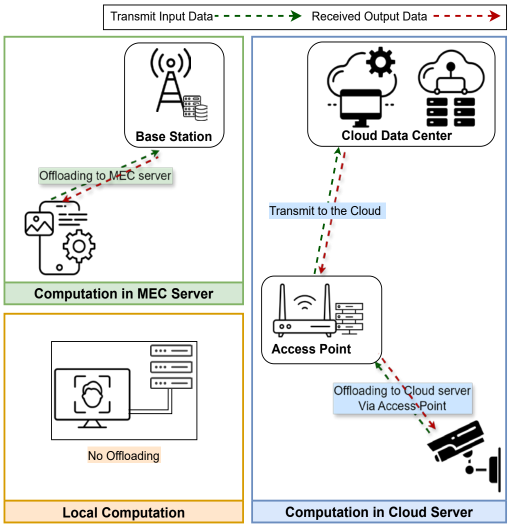

Fig. 2: An illustration of Computation Offloading type in MEC networks

E. Problem Formulation

The objective of this study is to minimize the execution time of computing tasks in an MEC system by optimizing task offloading and resource allocation. We consider application constraints, available computing capacities, and resources. Our problem is formulated as a minimization problem with an objective function that incorporates task constraints, resource capacities, and offloading decision variables:

$$\text{Minimize} \quad \mathcal{P} = \sum_{i=1}^{\mathcal{D}} \sum_{j=1}^{\mathcal{T}_i} \sum_{m=1}^{\mathcal{M}} \alpha_{i,j} T^{IoT}_{\tau_{i,j}} + (1-\alpha_{i,j}) \times \left(\beta_{i,j} T^{\text{MEC}_m}_{\tau_{i,j}} + (1-\beta_{i,j}) T^{\text{Cloud}}_{\tau_{i,j}}\right) \tag{6}$$

Subject to:

$$\alpha_{i,j}, \beta_{i,j} \in \{0,1\} \quad ,\forall i \in \mathcal{D}, \forall j \in \mathcal{T}_i \tag{6a}$$

$$\sum_{i=1}^{\mathcal{D}} \sum_{j=1}^{\mathcal{T}_i} (1-\alpha_{i,j}) \times \beta_{i,j} C_{i,j} \leq \sum_{m=1}^{\mathcal{M}} \mathcal{F}_{\text{MEC}_m} \tag{6b}$$

$$\sum_{i=1}^{\mathcal{D}} \sum_{j=1}^{\mathcal{T}_i} (1-\alpha_{i,j})(1-\beta_{i,j}) C_{i,j} \leq \mathcal{F}_{\text{Cloud}} \tag{6c}$$

$$\sum_{i=1}^{\mathcal{D}} \sum_{j=1}^{\mathcal{T}_i} \alpha_{i,j} C_{i,j} \leq \sum_{i=1}^{\mathcal{D}} \mathcal{F}_{IoT_i} \tag{6d}$$

$$\sum_{i=1}^{\mathcal{D}} \sum_{j=1}^{\mathcal{T}_i} \sum_{m=1}^{\mathcal{M}} (1-\alpha_{i,j}) \beta_{i,j} \mathcal{F}_{\text{MEC}_m} \leq \sum_{m=1}^{\mathcal{M}} Th_{\text{MEC}_m} \tag{6e}$$

The problem is subject to the following constraints: Constraint (6a) ensures that the decision variables $(\alpha_{i,j}, \beta_{i,j})$ are binary. Constraint (6b) guarantees that the server's computing capacity is sufficient if a task is offloaded to an MEC server. Constraint (6c) verifies that the capacity is adequate for tasks offloaded to the cloud server. Constraint (6d) ensures the device's computing capacity is sufficient for local execution. Finally, constraint (6e) restricts MEC server resource utilization to a specified threshold Th_{MEC_m}, typically set around 80%.

By addressing this optimization challenge, we can identify effective task offloading and resource allocation strategies that minimize total execution time while adhering to MEC system constraints. This problem can be formulated as an Integer Linear Programming (ILP) model, which provides optimal solutions [21] but faces high computational complexity, complicating real-time implementation for large-scale applications. To overcome this, heuristic strategies have been developed, balancing solution quality with computational efficiency. These methods enable real-time offloading decisions and resource optimization associated with ILP, ensuring system agility in managing dynamic task arrivals.

IV. HEURISTIC-BASED OFFLOADING STRATEGY FOR QOS IMPROVEMENT

We propose a **H**euristic-based **O**ffloading strategy to enhance **QoS** (HOQoS) in MEC environments. Our goal is to optimize offloading decisions for computing tasks while considering time constraints, resource limitations, and application requirements. Utilizing greedy heuristics, we explore the solution space to make optimal decisions for each task, factoring in predicted execution times across different architecture layers. By assessing the impact of offloading on overall service time, we aim for significant QoS improvement.

A key component of our strategy is the Offloading Decision Variable Identification (ODVI) algorithm (Algorithm 1), which determines the best offloading decision $(\alpha_{i,j}, \beta_{i,j})$ for each incoming task $\tau_{i,j}$. The algorithm initializes the minimum time as infinite and the decision variables as zero. For each task, it calculates the local execution time $T^{IoT}_{\tau_{i,j}}$ and estimates delays on MEC servers $T^{\text{MEC}_m}_{\tau_{i,j}}$ and in the Cloud $T^{\text{Cloud}}_{\tau_{i,j}}$. The task is assigned to the server with the shortest delay T_{min}, while respecting the maximum acceptable delay constraint. The decision variables $(\alpha_{i,j}, \beta_{i,j})$ are then based on the chosen optimal location, focusing on individual tasks rather than overall execution time.

Algorithm 1 Offloading Decision Variable Identification (ODVI)

Require: $(\mathcal{M}, \mathcal{C}, \mathcal{D}, \mathcal{T}_i)$
Ensure: $(\alpha_{i,j}, \beta_{i,j})$ Offloading decision variables

$T_{\text{Min}(i,j)} \leftarrow \infty$
Decision variables $(\alpha_{i,j}, \beta_{i,j}) \leftarrow \emptyset$
for $i \in \mathcal{D}$ **do**
  for $j \in \mathcal{T}_i$ **do**
    for $m \in \mathcal{M}$ **do**
      $T_{\text{Min}(i,j)} \leftarrow \min(T_{\text{Min}(i,j)}, T^{IoT}_{\tau_{i,j}}, T^{\text{MEC}_m}_{\tau_{i,j}}, T^{\text{Cloud}}_{\tau_{i,j}})$
    end for
    if $T_{\text{Min}(i,j)} = T^{IoT}_{\tau_{i,j}}$ **then**
      $\alpha_{i,j} \leftarrow 1$
    else
      if $T_{\text{Min}(i,j)} = T^{\text{Cloud}}_{\tau_{i,j}}$ **then**
        $\alpha_{i,j} \leftarrow 0$
        $\beta_{i,j} \leftarrow 0$
      else
        $\alpha_{i,j} \leftarrow 0$
        $\beta_{i,j} \leftarrow 1$
      end if
    end if
  end for
end for
return $(\alpha_{i,j}, \beta_{i,j})$

The ODVI algorithm, while effective for individual task offloading, has limitations in global optimization and resource contention management. Its focus on single tasks can lead to suboptimal overall system performance and does not address resource contention among multiple tasks. To overcome these issues, we introduce the Virtual Machine Selection for Execution (VMSE) algorithm (Algorithm 2).

VMSE tackles the global optimization problem by considering overall system resource allocation when selecting a virtual machine for each incoming task $\tau_{i,j}$. It implicitly manages resource contention by prioritizing VMs with lower utilization. The algorithm evaluates the characteristics of incoming tasks against the current utilization of MEC servers and the Cloud, selecting the least utilized VM that meets the task's performance, resource, and time constraints. This integration significantly enhances QoS through efficient resource utilization and improved task execution efficiency.

To optimize task offloading and resource allocation, we developed the HOQoS algorithm (Algorithm 3), which integrates the features of the ODVI and VMSE algorithms. First, the ODVI algorithm evaluates available execution environments by considering MEC server performance and network latency, generating decision variable pairs based on task characteristics such as size, complexity, and time constraints. These pairs

Algorithm 2 Virtual Machine Selection for Execution (VMSE)

Require: Incoming task $\tau_{i,j}$, Node($R_{opt}[\tau_{i,j}]$)
Ensure: VM satisfying task's ($\tau_{i,j}$) resource requirements.

task $\leftarrow \tau_{i,j}$
selectedVM $\leftarrow$ null
bestUtili $\leftarrow \infty$
for host $\in$ getInfrastructureHostList(Node($R_{opt}[\tau_{i,j}]$)) **do**
    vmList $\leftarrow$ getVmList(host)
    for vm $\in$ vmList **do**
        if task.Utilization() $\leq$ vm.getCapacity() **then**
            currentUtili $\leftarrow$ vm.getCurrentUtilization()
            if currentUtili $<$ bestUtili **then**
                selectedVM $\leftarrow$ vm
                bestUtili $\leftarrow$ currentUtili
            else if currentUtili == bestUtili **then**
                selectedVM $\leftarrow$ SelectVM(selectedVM,vm)
            end if
        end if
    end for
end for
return selectedVM

represent potential execution options for each task. Next, the VMSE algorithm selects the optimal execution location by assessing task characteristics and resource availability on MEC servers and in the Cloud. It evaluates each virtual machine's performance and chooses the one that best meets the task's requirements regarding performance, resource availability, and time constraints.

Algorithm 3 Heuristic-based offloading for improving quality of service (HOQoS)

Require: $((\alpha_{i,j}, \beta_{i,j}), \mathcal{M}, \mathcal{C}, \mathcal{D}, \mathcal{T}_{i,j})$
Ensure: Optimal Resource Allocation $R_{\text{opt}}[\tau_{i,j}]$ and selected VM $VM_{\text{selected}}[\tau_{i,j}]$

$R_{\text{opt}}[\tau_{i,j}] \leftarrow$ null
$VM_{\text{selected}}[\tau_{i,j}] \leftarrow$ null
for $i \in \mathcal{D}$ **do**
    for $j \in \mathcal{T}_i$ **do**
        $(\alpha_{i,j}, \beta_{i,j}) \leftarrow$ Call Algorithm 1
        if $\alpha_{i,j} = 1$ **then** ▷ Local allocation
            $R_{\text{opt}}[\tau_{i,j}] \leftarrow$ LocalNode
        else if $\beta_{i,j} = 1$ **then** ▷ MEC allocation
            $R_{\text{opt}}[\tau_{i,j}] \leftarrow$ MECNode
        else ▷ Cloud allocation
            $R_{\text{opt}}[\tau_{i,j}] \leftarrow$ CloudNode
        end if
        $VM_{\text{selected}}[\tau_{i,j}] \leftarrow$ Call Algorithm 2
    end for
end for
return R_{opt} and $VM_{\text{selected}}[\tau_{i,j}]$

The HOQoS algorithm employs a sequential task processing strategy, handling tasks individually rather than in batches. This approach is crucial for achieving system agility and effectively managing dynamic task arrivals. By adapting resource allocation in real time, the algorithm ensures a rapid and optimized response, even with irregular or unpredictable task arrival rates. Thus, sequential processing is integral to the algorithm's ability to efficiently manage dynamic environments.

V. PERFORMANCE EVALUATION

In the performance evaluation section, we conducted simulations using the EdgeCloudSim simulator [20]. We analyze the performance of our MEC architecture using various measurement and evaluation methods.

A. Simulation Parameters

The simulation parameters used for our evaluations are summarized in Table I. The simulations are implemented in Java, and the generated plots are visualized using Matlab. The experiments are conducted on a computer with 4 Intel Core i7-9600U processors clocked at 2.59 GHz and 8 GB of RAM.

TABLE I
SIMULATION PARAMETERS.

Parameter	IoT	MEC	Cloud
Number of devices	100 - 2,300	14	1
Number of hosts	1	1	1
Number of VMs per host	1	8	4
Number of Cores per VM	1	2	4
VM CPU Speed (in MIPS)	4,000	10,000	100,000

EdgeCloudSim utilizes four distinct application types to realistically simulate diverse scenarios, as detailed in [20]. Table II summarizes the characteristics of each application type: Heavy applications are characterized by high data transmission, high computational intensity, and low sensitivity to delays. Infotainment applications generally require moderate data transmission, high computational intensity, and low sensitivity to delays. AR/VR applications involve high data transmission volumes, moderate computational intensity, and high sensitivity to delays. Health applications typically feature moderate data transmission, moderate computational intensity, and moderate sensitivity to delays. Each application is assessed based on several criteria, including usage percentage, which indicates the share of each application in overall usage, and Cloud selection probability, which demonstrates the tendency to use the Cloud for their operation. The volumes of data uploaded and downloaded, as well as the task length, emphasize the requirements for bandwidth and processing.

TABLE II
APPLICATION CHARACTERISTICS.

Characteristics	Heavy	Infotainment	AR/VR	Health
Task Length (GI)	45	15	9	3
Delay Sensitivity	0.1	0.3	0.9	0.7
Max. Delay Req. (s)	2	1.5	1	0.5
Data Upload (KB)	2500	25	1500	20
Data Download (KB)	200	1000	25	1250

In addition to the characteristics of the applications presented in Table II, it is important to specify the technical parameters of the simulated environment. The bandwidth of the wireless local area network (WLAN) is set at 200 Mbps, while the bandwidth of the wide area network (WAN) is 15 Mbps. The propagation delay on the WAN is 0.1 seconds. These parameters also influence system performance and should be taken into account when evaluating the results of the simulation.

B. Simulation Results

This section presents the results of simulations conducted to evaluate the performance of different resource management approaches and computation offloading strategies in a high-density IoT environment. The studied approaches are:

- Only MEC: All tasks are executed on the MEC server. This approach is straightforward but may be limited in terms of resources and processing capacity.
- Only Cloud: All tasks are executed on the Cloud server. This approach offers high processing capacity but can lead to significant latency due to the distance between IoT devices and the cloud.
- Random: Tasks are randomly distributed between the MEC and Cloud servers. This implementation is simple but may not be optimal in terms of performance.
- DCOA-ST: A dynamic computation Offloading approach based on the service time [11]. This approach considers the current system state to decide where to execute tasks, potentially improving performance. It provides a systematic approach to task assignment but may lack flexibility in dynamic environments.
- LCDA*: The LCDA approach [12], which aims to minimize the execution time of delay-sensitive tasks while ensuring no deadline violations. Their algorithm selects servers and schedules tasks to optimize service time in a dynamic environment. While LCDA effectively adapts to changing workloads, it may not fully account for task sensitivity or latency constraints.

We can evaluate their relative advantages and limitations by comparing these methodologies with our HOQoS algorithm. HOQoS integrates factors such as execution time, task sensitivity to delays, latency requirements, resource utilization, and server processing capabilities. Through rigorous evaluations and performance comparisons, we illustrate the enhanced effectiveness of HOQoS in optimizing computation offloading and resource allocation within MEC environments, ultimately improving the quality of service for IoT applications.

1) Simulation Based on Average Service Time: We evaluate the average service time, a critical factor influencing service quality and user experience. Figure 3 illustrates the impact of the number of IoT devices on service times for different resource management approaches. For 100 devices, HOQoS and LCDA* show shallow service times of 0.8 seconds, while Only MEC and DCOA-ST display times of 0.9 seconds. At 900 devices, HOQoS maintains a service time of 1 second, while Only Cloud shows an alarming increase to 5.1 seconds, confirming its inadequacy in high-load scenarios. Other approaches, such as Random and DCOA-ST, exhibit times of 2.1 seconds and 1.15 seconds, respectively. For 1500 devices, HOQoS slightly increases to 1.2 seconds, while Only Cloud reaches 7.1 seconds. Finally, at 2300 devices, HOQoS reports a service time of 1.5 seconds, whereas Only Cloud remains high at 7.2 seconds.

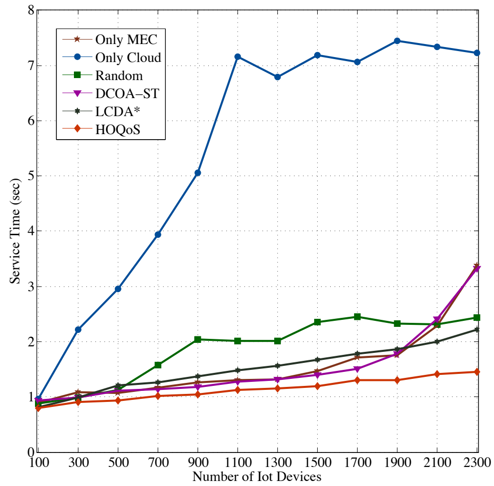

Fig. 3: Service time as a function of the number of IoT devices.

2) Simulation Based on Task Failure Rates: In assessing task execution success, task failure rates are key indicators influenced by factors such as excessive virtual machine usage and insufficient network bandwidth. Figure 4 illustrates how failure rates vary across algorithms with different numbers of IoT devices. The HOQoS approach consistently maintains low failure rates, even as device numbers increase, highlighting its robustness in high-density IoT environments. For 300 devices, HOQoS and Only MEC have low failure rates of 1%, while LCDA* has the highest at 10%. At 900 devices, Only

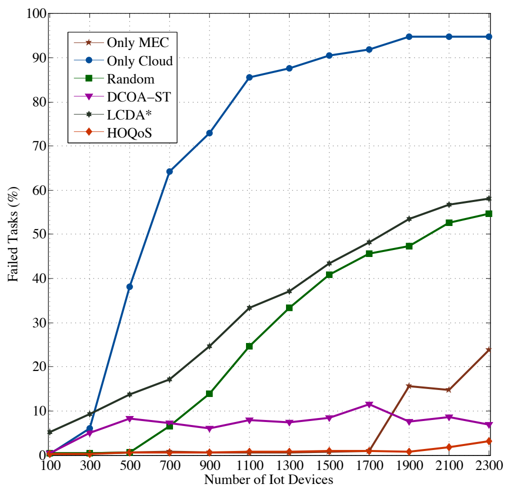

Fig. 4: Task failure rate as a function of the number of IoT devices.

Cloud shows a concerning 74% failure rate, whereas HOQoS maintains a reasonable 1.5%. With 1500 devices, Only Cloud's failure rate rises to 91%, confirming its inadequacy in high-load scenarios, while HOQoS remains at 2%. Finally, at 2300 devices, all failure rates increase, with Only Cloud reaching a critical 95% and HOQoS at 4%.

3) Simulation Based on VM Utilization Rates: Figure 5 highlights the impact of the number of IoT devices on the utilization of VMs at the MEC level. For 300 devices, HOQoS shows a utilization rate of 1%, while Only MEC reaches 3%. Other approaches, such as Random and DCOA-ST, exhibit utilization rates of 2% and 4.5%, respectively, while LCDA* reaches 5%. At 1100 devices, HOQoS maintains a rate of 2.5%, while Only MEC climbs to 12%. The Random and DCOA-ST approaches also show increases, reaching 5% and 13%, respectively. When the number of devices reaches 1700, HOQoS increases to 5%, while Only MEC rises to 23%, with LCDA* at 19% and DCOA-ST at 23%. Finally, at 2300 devices, HOQoS presents a utilization rate of 9.8%, contrasting sharply with Only MEC, which reaches 58%. Other approaches show increased utilization, with Random at 12% and DCOA-ST at 27%.

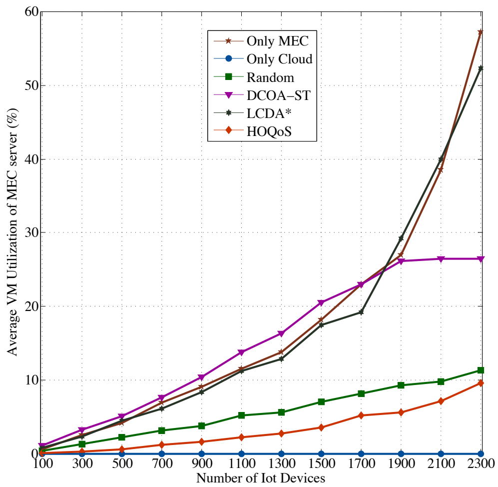

Fig. 5: MEC server utilization as a function of the number of IoT devices.

The utilization of VM at the Cloud level is presented in Figure 6, exhibiting slightly lower rates compared to MEC servers. For 500 devices, HOQoS shows a VM utilization rate of only 0.1%, while Only Cloud reaches 1.2%. Other approaches, such as Random and DCOA-ST, exhibit utilization rates of 0.8% and 1.2%, respectively, while LCDA* reaches 0.4%. At 1300 devices, HOQoS maintains a utilization rate of 0.1%, while Only Cloud climbs to 1.1%. The Random and DCOA-ST approaches also show increases, reaching 1% and 3.7%, respectively. When the number of devices reaches 1900, HOQoS increases to 0.2%, while Only Cloud rises to 1.1%, with LCDA* at 2.7% and DCOA-ST at 5.8%. Finally, at 2300 devices, HOQoS presents a utilization rate of 0.3%, contrasting sharply with Only Cloud, which reaches 1.4%. Other approaches show increasing utilization, with Random at 1% and DCOA-ST at 11.9%.

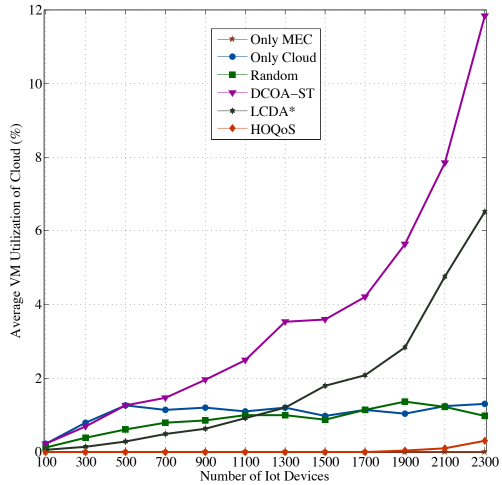

Fig. 6: Cloud server utilization as a function of the number of IoT devices.

The results demonstrate that HOQoS effectively manages both MEC and cloud server resources under increasing load. In contrast, other approaches, particularly Only MEC and Only Cloud, exhibit a significant rise in VM utilization, which could lead to challenges concerning performance and energy consumption. It is important to note that Only Cloud does not utilize MEC servers, while Only MEC does not utilize cloud servers; therefore, their VM usage is not considered in this analysis.

VI. Conclusion and Future Perspectives

This study aimed to improve service quality in Multi-access Edge Computing by offloading computational loads and allocating resources near IoT devices. The findings emphasize the importance of offloading tasks from IoT devices with limited computing capabilities to locations with adequate resources, thereby reducing latency. The proposed heuristic algorithms for task offloading and resource allocation consider task characteristics and resource availability, resulting in decreased service times and task failure rates. Future research will focus on developing an autonomous system that utilizes reinforcement learning and machine learning techniques to optimize task execution locations in complex MEC environments. Enhancing the algorithms' adaptive capabilities to respond dynamically to changes in network conditions and device capabilities will be a priority. Additionally, integrating advanced predictive analytics could enable proactive optimization of resource allocation and offloading strategies by forecasting future resource needs and user demands. These advancements aim to further enhance the performance and efficiency of MEC systems.

References

[1] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke," Middle-ware for Internet of Things: A Survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, Feb. 2016.
doi: 10.1109/JIOT.2015.2498900.

[2] 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. **DOI**: 10.1109/COMST.2018.2849509.

[3] European Telecommunications Standards Institute (ETSI), "Mobile-edge computing (MEC); Framework and reference architecture," ETSI GS MEC 003, 2019.

[4] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal, et al., "Mobile-edge computing introductory technical white paper," *White paper, mobile-edge computing (MEC) industry initiative*, vol. 29, pp. 854– 864, 2014.

[5] 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. **DOI**: 10.1109/COMST.2017.2705720.

[6] A. P. Jafari Pozveh and H. S. Shahhoseini, "IoT Integration with MEC," in *Mobile Edge Computing, Springer, Cham*, 2021, pp. 111–144. **DOI**: 10.1007/978-3-030-69893-5_6.

[7] M. Myyara, O. Lagnfdi, A. Darif, and A. Farchane, "A New Approach Based on Genetic Algorithm for Computation Offloading Optimization in Multi-Access Edge Computing Networks," *IAES International Journal of Artificial Intelligence (IJ-AI)*, vol. 13, no. 4, pp. 4186–4194, 2024. **DOI**: 10.11591/ijai.v13.i4.pp4186-4194.

[8] C. Shu, Z. Zhao, Y. Han, G. Min, and H. Duan, "Multi-User Offloading for Edge Computing Networks: A Dependency-Aware and Latency-Optimal Approach," *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 1678–1689, Mar. 2020. **DOI**: 10.1109/JIOT.2019.2943373.

[9] C. Chen, B. Liu, S. Wan, P. Qiao, and Q. Pei, "An Edge Traffic Flow Detection Scheme Based on Deep Learning in an Intelligent Transportation System," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1840–1852, Mar. 2021. **DOI**: 10.1109/TITS.2020.3025687.

[10] A. Naouri, H. Wu, N. A. Nouri, S. Dhelim, and H. Ning, "A novel framework for mobile-edge computing by optimizing task offloading," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 13 065–13 076, 2021. **DOI**: 10.1109/JIOT.2021.3064225.

[11] M. Myyara, O. Lagnfdi, A. Darif, and A. Farchane, "Quality of Expe- rience Improvement and Service Time Optimization through Dynamic Computation Offloading Algorithms in Multi-access Edge Computing Networks," *International Journal of Computer Network and Information Security (IJCNIS)*, vol. 16, no. 4, pp. 1–16, 2024. **DOI**: 10.5815/ijcnis.2024.04.01.

[12] H. Choi, H. Yu, and E. Lee, "Latency-classification-based deadline-aware task offloading algorithm in mobile edge computing environments," *Applied Sciences*, vol. 9, no. 21, p. 4696, 2019. **DOI**: 10.3390/app9214696.

[13] C. Eang, S. Ros, S. Kang, I. Song, P. Tam, S. Math, and S. Kim, "Offloading Decision and Resource Allocation in Mobile Edge Computing for Cost and Latency Efficiencies in Real-Time IoT," *Electronics*, vol. 13, no. 7, p. 1218, 2024. **DOI**: 10.3390/electronics13071218.

[14] C. Kai, H. Zhou, Y. Yi, and W. Huang, "Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 2, pp. 624–634, 2020. **DOI**: 10.1109/TCCN.2020.3018159.

[15] S. Wan, R. Gu, T. Umer, K. Salah, and X. Xu, "Toward Offloading Internet of Vehicles Applications in 5G Networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4151–4159, Jul. 2021. **DOI**: 10.1109/TITS.2020.3017596.

[16] I. Sarrigiannis, K. Ramantas, E. Kartsakli, P.-V. Mekikis, A. Antonopoulos, and C. Verikoukis, "Online VNF Lifecycle Management in an MEC-Enabled 5G IoT Architecture," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4183–4194, May 2020. **DOI**: 10.1109/JIOT.2019.2944695.

[17] M. Myyara, O. Lagnfdi, A. Darif, and A. Farchane, "A Resource Allocation Strategy to Enhance User Experience for IoT Devices in Multi-access Edge Computing," in *2024 Sixth International Conference on Intelligent Computing in Data Sciences (ICDS)*, 2024, pp. 1–7. **DOI**: 10.1109/ICDS62089.2024.10756444.

[18] Z. Ding, J. Xu, O. A. Dobre, and H. V. Poor, "Joint Power and Time Allocation for NOMA–MEC Offloading," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, pp. 6207–6211, Jun. 2019. **DOI**: 10.1109/TVT.2019.2907253.

[19] Z. Ning, P. Dong, X. Kong, and F. Xia, "A Cooperative Partial Computation Offloading Scheme for Mobile Edge Computing Enabled Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4804–4814, Jun. 2019. **DOI**: 10.1109/JIOT.2018.2868616.

[20] C. Sonmez, A. Ozgovde, and C. Ersoy, "Fuzzy workload orchestration for edge computing," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 769–782, 2019. **DOI**: 10.1109/TNSM.2019.2901346.

[21] P. W. Khan, K. Abbas, H. Shaiba, A. Muthanna, A. Abuarqoub, and M. Khayyat, "Energy efficient computation offloading mechanism in multi- server mobile edge computing—An integer linear optimization approach," *Electronics*, vol. 9, no. 6, p. 1010, 2020. **DOI**: 10.3390/electronics9061010.

Marouane Myyara received his B.Sc. in Electronic and Telecommunication Engineering in 2019 and M.Sc. in Telecommunication Systems and Computer Networks in 2021 from Sultan Moulay Slimane University, Beni Mellal, Morocco. He is currently a Ph.D. candidate at the Laboratory of Innovation in Mathematics, Applications, and Information Technology (LIMATI), Polydisciplinary Faculty, Sultan Moulay Slimane University, Morocco. His current research focuses on improving the performance of Multi-access Edge Computing networks (MEC), Cloud Computing, Computation Offloading, and the Internet of Things (IoT).

Oussama Lagnfdi received his B.Sc. in Physical Matter Science in 2020 and M.Sc. in Telecommunications Systems and Computer Networks in 2022 from Sultan Moulay Slimane University, Beni Mellal, Morocco. Currently, he is a Ph.D. candidate at the Laboratoire d'Innovation en Mathématiques et Applications et Technologies de l'Information (LIMATI), Polydisciplinary Faculty, Sultan Moulay Slimane University, Morocco. His ongoing research is focused on enhancing the performance of Internet of Things (IoT) and Mobile Edge Computing (MEC), Artificial Intelligence, Deep Learning, and Fuzzy Logic.

Anouar Darif received the bachelor in IEEA (Informatique Électrotechnique, Électronique and Automatique) from Dhar El Mahraz Faculty of Sciences at Mohamed Ben Abdellah University Fez, Morocco in 2005. He received the Diplôme d'Etudes Supérieures Approfondies in Computer Sciences and Telecommunications from the Faculty of Sciences Rabat in 2007. He received the Ph.D. degree in Computer Sciences and Telecommunications from the Faculty of Sciences of Rabat in 2015. He is currently a Research and Teaching Associate in the Multidisciplinary Faculty at the University of Sultan Moulay Slimane Beni Mellal, Morocco. His research interests include Wireless Sensor Networks (WSN), Mobile Edge Computing (MEC), Internet of Things (IoT), Cloud Computing, and Neural Networks.

Abderrazak Farchane received his B.Sc. in Computer Science and Engineering in June 2001 and M.Sc. in Computer Science and Telecommunication from the University of Mohammed V Agdal, Rabat, Morocco, in 2003. He obtained his Ph.D. in Computer Science and Engineering at ENSIAS, Rabat, Morocco. He is currently an Associate Professor of Computer Science in the Polydisciplinary Faculty, at Sultan Moulay Slimane University, Morocco. His areas of interest are Information Coding Theory, Cryptography, and Security.