Distributed Resource Distribution and Offloading for Resource-Agnostic Microservices in Industrial IoT

Amit Samanta, Tri Gia Nguyen, *Senior Member, IEEE*, Thao Ha, and Shahid Mumtaz, *Senior Member, IEEE*

***Abstract*—Due to increase in real-time mobile applications and Industrial Internet-of-Things (IIoT) devices, the edge computing paradigm provides a systematic and eccentric platform for real-time Internet-of-Things applications. Though the paradigm provides an effective infrastructure, however the resource requirements of IIoT devices change radically with time, which is described as a resource-agnostic property. Therefore, the estimation of resource requirements of IIoT devices is a critical and resilient assignment. In addition, it requires an extensive amount of resources to process the data traffic flows and microservice offloading. Hence, we present RAISE, a novel resource-agnostic microservice offloading scheme for mobile IIoT devices. RAISE efficiently estimates the resource-agnostic nature of IIoT devices to maximize their resource utilization in the network. Based on the estimated resource requirement, we propose a resource-agnostic microservice offloading scheme to maximize the success rate. Extensive experiments show that RAISE provides better performance in terms of network throughput and Quality-of-Service (QoS) than the other existing methods, SDTO and DTOS, in terms of cost and reliability.**

***Index Terms*—Mobile edge computing, Internet of Things, Industrial IoT, resource-agnostic, microservice offloading.**

I. Introduction

THE rapid growth of the industrial revolution has directed us to amalgamate different advanced fabrication methods with IoT to develop an acute and efficient network manufacturing system. The main motive of such advanced system is to enable an elevated automation system by merging heterogeneous technologies like IIoT [1]–[3] and edge computing to authorize the formation of interdependent, reactive, and smart inventing system. Therefore, Mobile Edge Computing (MEC) [4]–[7] plays important to role to envision such IIoT networks. MEC has emerged as a principal and essential paradigm for real-time IIoT applications. This kind of paradigm shifts the offloading mechanism to the network edge instead of offloading to a centralized infrastructure, i.e. a cloud platform. The advancement of MEC unites the cloud resource potential of the IIoT devices. Thus, such combination systematically takes out the core centralized compute capabilities to edge [8]–[12]. Such a kind of platform provides a productive and distinct unification of the network functionalities of the cloud platform and the access network. The edge platform provides a plethora of composite value-added microservices to distributed mobile applications [13]–[16], while providing a set of new functionalities for mission-critical applications. Expansion of MEC is mainly concentrated on performance improvement in terms of flexibility, microservice latency, and power consumption over the typical cloud computing platform.

The IIoT devices are typically resource-constrained in nature. Therefore, in an emergency situation, they do not get a fair amount of resources to process and offload their computational microservices [17]–[19] with heterogeneous applications to edge servers in real-time. Apart from this, the high network load, shared radio access network, and growing demand for network bandwidth affect the data transmission process in edge computing, which eventually makes the IIoT devices more resource-constrained. Furthermore, the resource requirement of IIoT devices changes dynamically with time, such kind of behavior for IIoT devices is generally described as the resource-agnostic property. Therefore, it is very important to capture such behaviour in order to provide fair resources for all the contending IIoT devices. Hence, we develop a resource-agnostic microservice offloading method for IIoT devices. Our scheme accurately fulfils the resource requirements of IIoT devices for different mobile applications.

A. Motivation

We discuss the underlying challenges and implications of designing a novel resource-agnostic optimal computational microservice offloader for the MEC platform. To accomplish efficient computation offloading for the MEC platform, we need to address two fundamental questions:

- How should a scheme efficiently estimate the resource requirements of IIoT devices accurately and provide a higher priority to edge microservices over others?
- How efficiently the computational microservices is to be offloaded on the edge platform, while providing the scalability to the platform with higher accuracy?

These two fundamental questions are comprised of several sub-questions, which we need to address. Hence, a resource-agnostic resource distribution and offloading scheme must address these four primary challenges:

- ***Optimal microservice identification***: In general, incoming microservices (i.e., microservice pools) are a mash-up of various mobile applications; thus, identifying different mobile microservices is a critical task. In real-life, the applications of several mobile IIoT devices are generally mixed into different heterogeneous microservices. Some

Amit Samanta is with the School of Computing, University of Utah, USA. (Email: amit.samanta049@gmail.com)

Tri Gia Nguyen and Thao T. Ha are with FPT University, Da Nang 50509, Vietnam. (Email: tri@ieee.org, thaoht32@fpt.edu.vn)

Shahid Mumtaz is with Nottingham University, Nottingham, UK. (Email: Dr.shahid.mumtaz@ieee.org)

Corresponding author: Tri Gia Nguyen (tri@ieee.org).

of the examples of different microservices are: critical (augmented reality (AR)), normal (general sensor data), and background (updates). Due to the lack of proper identification, the microservice delay of the network increases and also the system overhead increases invariably.

- ***Estimation of resource requirements of IIoT devices***: Estimation of resource requirements of edge microservices for a wide range of microservices generated from different applications is very challenging and important work for the MEC platform. As each of the microservices has a resource-agnostic property, it is tough to estimate the requirements accurately and offload them efficiently to edge servers. Therefore, improper estimation of resource requirements of IIoT devices leads to an increase in microservice delays and also decreases in system utilization.
- ***Design of optimal microservice offloader***: After the estimation of the resource requirements of IIoT devices, the computational microservices need to be offloaded efficiently to increase the network throughput. In order to do that, we need to design an optimal microservice offloading mechanism, which can efficiently offload the microservices to nearby servers.
- ***Microservice prioritization***: To offload the microservices efficiently, it is very important to prioritize the microservices based on their resource requirements and their microservice types. To prioritize the microservices, we implement the First-In-First-Out (FIFO) queue for simplified microservice prioritization. The starvation is a problem for FIFO because of the heavy hitters. Hence, we use the preemption for long running microservices, which helps in case of starvation. Also, FIFO is easy to apply and each microservice with the right amount resources in the beginning of the queue can be executed fairly without any manipulation.

B. Contributions

We observe that some existing solutions provide some resource allocation schemes for IIoT devices. However, they frequently assume that the resource requirements of IIoT devices are known to the system, which is not always the case due to the devices' resource-agnostic nature. They apparently ignore this fact. Therefore, it is necessary to consider this resource-agnostic fact for efficient computational microservice offloading of IIoT devices first, while minimizing the microservice delay. Hence, the *important* contributions can be outline as follows:

- We design a resource-agnostic resource distribution scheme for edge computing to distribute legitimate resources to IIoT devices.
- We also design an optimal microservice offloading method for efficient computational microservice offloading of IIoT devices to edge servers based on their different application requirements and computational powers.
- We justify the RAISE's efficiency through a series of extensive experiments and evaluate the impact on resource utilization, system overhead and microservice delay.

The paper is organized as described. Section II describes the related work. In Section III, we study a simple problem scenario and system model for MEC platform. A resource-agnostic microservice offloading scheme is investigated in Section IV. Then, an optimization framework is designed for microservice offloading in Section V. Section VI depicts the simulation experiments to validate the performance. Section VII concludes the paper with the future research problems.

II. Related Work

We provide a synopsis of existing work concerning the aspects of offloading mechanisms in MEC and IIoT networks.

A. Offloading Mechanisms in Edge Computing

First, we focus on investigating the offloading mechanisms in edge computing. In [20], Gao *et al.* focus on the offloading scheme among multiple user-equipments and servers in MEC. First, each user-equipment determines the amount of workload that needs to be offloaded to the server to optimize the overall task execution delay. The authors then proposed an aggressive game that takes into account instant load billing to improve the server's processing efficiency. In [21], Hou *et al.* integrates MEC nodes and fixed edge computing nodes to support both compute-intensive and delay-stringent services in Internet of vehicles (IoV). The objective of the approach is to provide better and more successful execution probability of services. In [22], Wong *et al.* proposed a cooperative edge computing scheme with AI techniques for IoT networks. The fused heterogeneous edge public data for training a basic AI model of cloud-scale are implemented by the cloud. The basic model and the migration model based on edge-private data are decompressed and reconstructed by the edge while extending learning with newly generated data. With this approach, the authors implemented a predictable task offloading architecture and caching algorithm at the network edge. In [23], Li *et al.* focused on securing and distributed outsourcing problems of modular exponentiation with fixed base and variable exponent in IoT networks based on edge computing. First, the authors proposed an algorithm for load balancing by the edge nodes' precomputation and solving the modular exponentiation. Then, the authors adopted another logical division approach and load balance among edge nodes using a segmentation approach. In [24], Li *et al.* designed an algorithm to optimize offloading scheme with multiple servers in MEC. First, the authors formulated three problems of multi-variable optimization, i.e., minimizing response time, power consumption, and cost-performance ratio. The authors then developed numerical algorithms to solve the three problems mentioned above. An online anticipatory proactive network association scheme for mobile devices in IoT networks with MEC is presented by Cui *et al.* [25]. The aim of their work is to optimize the stringent task latency with optimal energy consumption constraint. First, the authors design a delay model for a mobility-aware edge platform. The authors then propose an algorithm for online decisions with two-stage Markov decision processes (MDP) and the Lyapunov optimization technique.

Table I: A brief comparison between relevant exiting related works

Literature	Resource-agnostic	Mobility	Latency	Completion time	Ordering
Gao *et al.* [20], Hou *et al.* [21]	✗	✗	✓	✗	✗
Gong *et al.* [22], Li *et al.* [23]	✗	✓	✓	✗	✗
Cui *et al.* [25], Fantacci *et al.* [26], Xu *et al.* [27], Qi *et al.* [28]	✗	✗	✓	✓	✓
Chekired *et al.* [29], Coutinho *et al.* [30]	✗	✗	✗	✗	✗
Lucas *et al.* [31], Li *et al.* [24]	✗	✓	✓	✗	✗
Proposed scheme (**RAISE**)	✓	✓	✓	✓	✓

B. Resource Allocation and Offloading in IIoT

Here, we focus on investigating the offloading mechanisms in IIoT networks. In [26], Fantacci and Picano focus on IIoT networks with a combined edge-cloud computing architecture. The authors propose an approach to pursuing a preferable deployment of virtual-machine replica-copies of the edge services residing on the edge servers, combined with acceptable IIoT devices assigned to the edge servers. In [27], Xu *et al.* design the uplink joint power allocation and the scheduling problem of IoT devices providing optimal fairness for IIoT over cognitive heterogeneous NOMA networks. They have used dual decomposition and convex approximation methods to solve the above problems. A compressed and private data sharing (Cpds) network to enable efficient and private data sharing for IIoT by using blockchain is presented by Qi *et al.* [28]. This framework considers an off-chain method to compress and encrypt product data before being submitted to the blockchain system. In [29], Chekired *et al.* designed a fog platform for IIoT applications considering multi-tier servers. To solve the problem of scheduling IIoT devices, the authors formulated a workload assignment algorithm as an integer programming system. Further, the solution is accumulated in different layers by applying the simulated annealing method. In [30], an optimal method of a shared 5G small cell caching system for content delivery in smart industry and connected cars applications is presented by Coutinho and Boukerche. The authors consider the content request characteristics and the distinct content catalogs of the different applications of IIoT and vehicle networks to develop the mathematical framework. Lucas-Estan and Gozalvez [31] presented a load balancing method which dynamically control the amount of data to be transmitted by each node. The authors design the scheme to predict the spatio-temporal variations of data in IIoT and reduce the number of reconfigurations of wireless links. The scheme also supports the deployment of self-organizing and reliable industrial wireless networks.

Synthesis : Most of the existing literature focuses on computational offloading techniques and the minimization of microservice delay in the MEC. None of the existing literature considers the resource-agnostic property of the IIoT devices and, moreover, they have not proposed any optimal and flexible resource distribution scheme for IIoT devices to provide better microservices. Although there are many resource distribution techniques in the literature, they are mostly limited to general wireless and cellular networks. However, these techniques are not deemed suitable for microservices at the edge, as microservices differ from traditional monolithic services in several ways. Such as monolithic services are very hard to scale based on the demand, whereas microservices are easy to scale. Similarly, it is very hard to deploy the monolithical services compared to microservices. Also, easier to provide better isolation for microservices based on the application requirements. Therefore, we design a microservice offloading scheme for the edge platform while considering the optimal delay constraint.

III. RAISE Design and Description

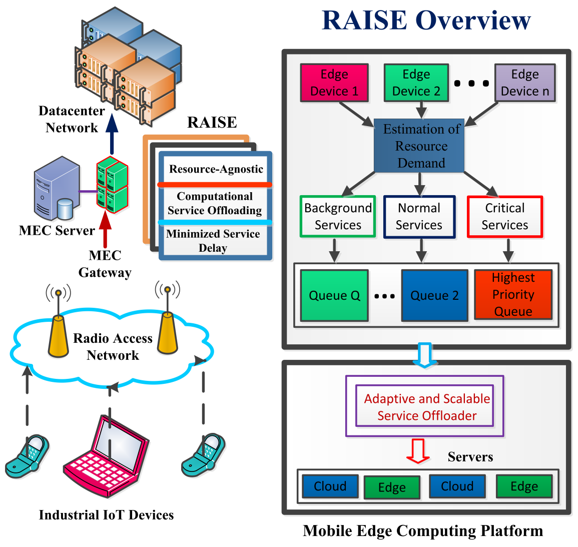


Figure 1: RAISE Architecture

This section discusses the overall architectural view of RAISE, as shown in Figure 1. The left part of the figure 1 shows the overall architecture of MEC, in which the IIoT devices offload their computational microservices to servers through an edge gateway using the radio-access network. We deploy our design system RAISE at the edge gateway in order to provide an optimal microservice offloading scheme between IIoT devices and servers. The right part of the figure shows the detailed design specifications of RAISE, which are basically comprised of four modules. In the first module (1), the IIoT devices estimate their resource requirements/demands, while considering the resource-agnostic property. In the second module (2), we identify the different microservice classes based on the estimated resources. Here, we considered three types of microservice classes - critical edge microservices, normal microservices and background microservices. Thereafter, in the third module (3), we design several priority queues to offload the microservices efficiently. Thus, we assign different priorities to the microservices based on their classes. After the assignment of different priorities, the computational microservices are offloaded to edge servers in order to minimize the microservice latency using the optimal microservice offloader. Hence, in the last module (4), the estimated resource requirements are sent to the optimal resource distributor. The optimal resource distributor assigns the required resources to IIoT devices according to their requirements. Then, the IIoT devices offload their computational microservices to edge servers.

A. Resource-Agnostic Microservices

As previously discussed, it is essential to allocate an equitable amount of resources to resource-hungry applications running on mobile IIoT devices. In order to assign the resources fairly, we need to estimate the resource requirements of IIoT devices efficiently to fulfill their quality-of-microservice (QoS). However, the estimation of the resource requirements of IIoT devices is very challenging, as the IIoT devices are resource-agnostic in nature. To identify the requirements accurately, we consider flow, community, and application level properties. After the identification of accurate resource requirements, we offload the microservices of different IIoT devices optimally to optimize the microservice delay. To achieve these goals, RAISE relies on the following four important steps:

1) ***Identification of Resource-agnostic Property***: First, we identify the resource-agnostic property of IIoT devices accurately. The microservice flow from IIoT devices can be characterized by a set of different attributes. Thus, the selection of useful attributes is an important step for the identification of resource-agnostic property of IIoT devices. Here, RAISE explores explicit and implicit attributes and heuristics on multiple levels for identification instead of considering a black-box approach.
2) ***Efficient Resource Distribution***: Following the identification of resource-agnostic property, we estimate the optimal resource requirements of IIoT devices to offload their computational microservices efficiently. Hence, given all the attributes, RAISE calculates optimal microservice utility for IIoT devices to capture the microservice flow relationships. Here, the different flows belonging to the same microservice class will have a small difference in the resource response rate. The key challenge here is to design a proper metric which reflects the importance of resource-agnostic property. RAISE employs previous resource demand to estimate the current demand response of the IIoT devices.
3) ***Optimal Microservice Offloading***: RAISE employs an optimal microservice offloading mechanism to offload the microservices to servers. The offloading decision is dependent on the total load on servers, capacity and flow constraints. Following the optimization problem, we also design an optimal microservice offloading algorithm.
4) ***Microservice Prioritization***: After the estimation of resource requirements, we design a microservice prioritization queue to avoid the starvation problem in the network. Critical microservices should be given higher priority, while normal microservices should be given lower priority.In the presence of all normal microservices, we follow a first-in-first-out (FIFO) queue.

B. Identification of Resource-Agnostic Property

We consider a set of microservice-level features that might be useful for the identification of resource-agnostic property of IIoT devices. First and foremost, to identify the resource-agnostic property of a microservice class, we apply different microservice-level features. These features are very important for knowing the microservice types and their fundamental attributes. They will help us to know the data packet size and arrival time of different microservices. Hence, we rely on a largely-used microservice-level features [32]–[34], such as:

- $\mathcal{S}_t$: start time of a microservice
- $\mathcal{F}_{si}$: mean data packet size in a microservice flow
- $\mathcal{V}_{siz}$: variance of data packet sizes in a microservice flow
- M_{int}: average data packet inter-arrival time

Here, we have excluded microservice flow size and duration as they cannot be obtained until a microservice flow completes. They eventually will not affect the performance as we have some fundamental features of the microservices. In order to formalize our approach, we define the problem of estimating the resource usage for a given computational microservice to be provisioned. We assume that the provisioning IIoT device has the following information about the computational microservices, which is discussed as:

- The type of microservices to be provisioned $\mathcal{S}$.
- A vector $\mathcal{B}$ of input data for task $\mathcal{S}$

Furthermore, given all the information about the microservices, a list of resources for each IIoT device is to be estimated.

For our system architecture, we assume that a microservice flow is comprised of k sub-microservice flows as $\mathcal{S} = \{S_1, S_2, \cdots, S_k\}$. The microservice flows $\mathcal{S}$ of IIoT devices should proceed through the platform in an optimal order while entering in S_1 and leaving in S_k. The order of microservice flows should be decided depending on the criticality factor of microservices, in the beginning we consider that microservice flows $\mathcal{S}$ should be ordered partially. An IIoT device consists of heterogeneous microservices and it's configurations. We consider that such set of microservices as *microservice configuration*. Further, we consider the set of feasible configurations $\mathbb{C}$ of the i^{th} microservice can be represented as $\mathbb{C}_i = \{\mathbb{C}_{i1}, \mathbb{C}_{i2}, \cdots, \mathbb{C}_{i|\mathbb{C}_i|}\}$. Following this, the microservice-level system configurations should be represented as $\mathbb{C}_{ij} = \{\mathbb{X}_{j1}, \mathbb{X}_{j2}, \cdots, \mathbb{X}_{j|\theta|}\}$, where $\mathbb{X}_{j|\theta|}$ is the count of microservice instances θ and θ illustrates the set of available microservice instances $\theta = \{\theta_1, \theta_2, \cdots, \theta_{|\theta|}\}$. Each microservice instance θ_i is connected to a tuple of h dimensions considering the allotted heterogeneous resources (e.g. storage, network, CPU cores, $\cdots$). As an example, suppose a microservice (instance) requires multiple resources in a order of CPU, I/O, memory and network. So, in this case, the microservice would first need CPU then I/O and then memory and network. Basically, it would be connected to resource types in a staged fashion (top-down). We denote the resources for the l^{th} microservice instance as $\theta_l = \langle \theta_l^1, \theta_l^2, \cdots, \theta_l^h \rangle$. The configuration setups $\mathbb{C}_i$ should be ordered in top-down fashion with it's resource necessity. Hence, we design a set for all microservice configurations as $\mathcal{S}$ as $\mathcal{C} = \{\mathbb{C}_i | i \leq k\}$. It should be noted that we can choose any kind of configuration for microservice instance without any fundamental limitation. The configuration refers to a process by which microservices should be deciding which resource types it wants and which order. On such system configuration in the load balancer can handle multiple microservices with same kind, on the other hand, microservice of different kinds needs different configurations. As per as the resource requirements is concerned, we define the minimum aggregated resource requirements of type r for a system configuration $\mathbb{C}_{ij}$ as $\mathcal{R}_i^{min}(r) = \sum_{l=1}^{|\theta|} \mathbb{X}_{jl}\theta_l^h$. The required resources have upper bound of u^r, it considers the maximum resources of type r available for microservice $\mathcal{S}$. Further, with respect to the microservice, we define the microservice utilization of $\mathbb{C}_{ij}$ as $\mathbb{W}_{ij}$ the function of the microservice being executed and the average capacity of a microservice.

IV. Resource Distribution Framework

Let us consider, the edge network has n IIoT devices $\mathcal{N} = \{1, 2, \cdots, n\}$ and they are connected to m edge severs $\mathcal{M} = \{1, 2, \cdots, m\}$. To deal with the time-critical microservices, we divide time into $\mathcal{T}$ time slots, defined as $T = \{1, 2, \cdots, \mathcal{T}\}$. In the edge platform, we have a set of predefined resource capacity, denoted as $\mathcal{R}$ with size $|\mathcal{R}|$. Their offloading capacities are denoted as $\mathcal{E} = \{\mathcal{E}_1, \mathcal{E}_2, \cdots, \mathcal{E}_n\}$ in terms of the number of microservices it can offload in one time unit. Here, to handle the resource-agnostic nature of IIoT devices in MEC, there is an agnostic factor for each IIoT device $\gamma = \{\gamma_1, \gamma_2, \cdots, \gamma_n\}$, which is defined as:

$$\gamma_i = \left(\mathcal{R}_i \frac{\mathcal{H}_i^t}{\sum_{i \in n} \sum_{t \in T} \mathcal{H}_i^t} \right) \tag{1}$$

where $\mathcal{R}_i^t$ denotes the resource requirement, T denotes the total time period and $\mathcal{H}_i^t$ denotes the changing resource response rate of IIoT devices. For the resource requirement, first we calculate the resource requirements of microservices over each timestep in a time period, then we created a profile based on it. We start with the maximum CPU allocation and do a binary search on the CPU values to estimate resource requirement. If the profiled point resulted in a better resource utilization that is less than a fixed threshold (say 50%), then we continue binary search on the lower half of CPU values, else we profile more points on the upper half. The idea here is to empirically profile CPU regions that show significant difference in resource utilization, while skip those regions with little to no improvement in resource utilization.

Based on the resource-agnostic factor, we design a resource demand matrix for each IIoT device $\mathcal{G}_{it}$, where $i \in n, t \in T$. We specify an offloading strategy by solving a offloading rate $\mathcal{S}_{ij}^t$, $i \in n, j \in m$. For resource-agnostic scheme, we consider three tuple $\mathcal{R}_i^t, \mathcal{D}_i, \mathcal{P}_i$ to provide fair and optimal resource distribution among IIoT devices. Here, $\mathcal{D}_i$ denotes the delay requirement and $\mathcal{P}_i$ denotes the priority level of of each edge microservice, respectively. The RAISE, basically, distributed the resources to each of the devices. Based on few important metrics, we design a utility for IIoT devices and the utility of i IIoT device at time t is expressed by $\mathbb{U}_i(\mathcal{R}_i, \mathcal{D}_i, \mathcal{P}_i, t)$. We get a fixed set to interpret a feasible utility set of IIoT devices, which is denoted by $\mathbb{U}$. The set $\mathbb{U}$ is represented as the joint utility set, such as, $\mathbb{U} = \{\mathbb{U}_1, \mathbb{U}_2, \cdots, \mathbb{U}_n\} \in \mathbb{R}^n$. Each IIoT device has a minimum resource requirement to process their computational microservices. The minimum requirement of resources for IIoT device i is represented as, $\mathcal{R}^{min} = \{\mathcal{R}_1^{min}, \mathcal{R}_2^{min}, \cdots, \mathcal{R}_n^{min}\}$. The utility function $\mathbb{U}_i(\mathcal{R}_i, \mathcal{D}_i, \mathcal{P}_i, t)$ for the IIoT device i at time instant t is calculated as:

$$\mathbb{U}_i(\mathcal{R}_i, \mathcal{D}_i, \mathcal{P}_i, t) = \left[\gamma_i \Theta_i^t \mathcal{P}_i \frac{\mathcal{C}_{\mathcal{R}_i^t}}{\mathcal{C}_{\mathcal{R}_i^t}^{max}} ln(\mathcal{R}_i^t - \mathcal{R}_i^{min,t}) \right] \tag{2}$$

where Θ_i^t denotes the profit level of IIoT devices, $\mathcal{C}_{\mathcal{R}_i^t}$ and $\mathcal{C}_{\mathcal{R}_i^t}^{max}$ denote the delay-sensitive actual resource and maximum cost for IIoT devices, $\mathcal{R}_i^t$ denotes the actual resource requirement of IIoT devices. The profit level Θ_i^t for the IIoT devices is mathematically described as:

$$\Theta_i^t = \left(\sigma_i^{net} \frac{\mathcal{I}_i(t)}{\mathcal{L}_i(t)} \right) \tag{3}$$

where $\mathcal{I}_i(t)$ and $\mathcal{L}_i(t)$ denote the corresponding microservice rate and size of the microservice at time t, respectively. σ_i^{net} denotes the unit profit factor. The actual delay-sensitive resource cost $\mathcal{C}_{\mathcal{R}_i^t}$ is described as the total cost incurred by the mobile IIoT devices for delay-sensitive microservices. Mathematically,

$$\mathcal{C}_{\mathcal{R}_i^t} = \left(\eta(\mathcal{W}_{\mathcal{R}_i}(t) \Upsilon_{\mathcal{R}_i}(t) - \mathcal{V}_{\mathcal{R}_i}(t)) \right) \tag{4}$$

where η denotes the pricing factor, $\mathcal{W}_{\mathcal{R}_i}(t)$ signifies the cost of actual resources allotted to devices for delay-stringent microservices, $\mathcal{V}_{\mathcal{R}_i}(t)$ denotes the basic cost require to get minimum resources. The basic cost $\mathcal{V}_{\mathcal{R}_i}(t)$ is mathematically expressed as:

$$\mathcal{V}_{\mathcal{R}_i}(t) = \left(\mathcal{F}(t) \frac{\mathcal{D}_i}{\mathcal{D}_{max}} \right) \tag{5}$$

where $\mathcal{F}(t)$ denotes the unit delay-sensitive cost for IIoT devices, $\mathcal{D}_i$ and $\mathcal{D}_{max}$ denote the actual and maximum delay faced by microservices, respectively. It should be considered that the demand rate of resource requirements for IIoT devices changes with time, therefore the demand response rate $\Upsilon_{\mathcal{R}_i}(t)$ is dynamic in nature. This is modeled as an ordinary differential equation. The demand response at time t depends on the demand before t. Thus, demand rate $\Upsilon_{\mathcal{R}_i}(t)$ of resource requirements for IIoT devices is expressed as:

$$\Upsilon_{\mathcal{R}_i}(t) = \left(\frac{d^n \Upsilon_{\mathcal{R}_i}(t-1)}{dt^n} + \Upsilon_{\mathcal{R}_i}(t-1) \right) \tag{6}$$

The accurate prediction of demand rate is crucial for microservices to get fair amount of resources.

The resource distribution among IIoT devices tries to find an optimal solution with fair resources for IIoT devices to optimize the throughput and resource utilization. To find the optimal resource utilization, the distributed resource-agnostic scheme need to be maximized. The utility function for resource maximization problem is expressed as:

$$\text{Max} \ \ \mathbb{U}_i(\mathcal{R}_i, \mathcal{D}_i, t) = \left[\gamma_i \Theta_i^t \mathcal{P}_i \frac{\mathcal{C}_{\mathcal{R}_i^t}}{\mathcal{C}_{\mathcal{R}_i^t}^{max}} ln(\mathcal{R}_i^t - \mathcal{R}_i^{min,t})\right] \quad (7)$$

$$\text{subject to} \quad \sum_{i=1}^{n} \mathcal{D}_i < \mathcal{D}_{max}, i \in \mathcal{N} \quad (8)$$

$$\sum_{i=1}^{n} \mathcal{R}_i^t \geq \mathcal{R}_i^{min,t}, i \in \mathcal{N}, t \in T \quad (9)$$

$$\sum_{i=1}^{n} \Theta_i^t \geq \Theta_{th}, i \in \mathcal{N}, t \in T \quad (10)$$

We illustrate the optimization problem in detail. (7) demonstrates the primary optimization problem and (8) signifies that the delay-requirement of microservices $\mathcal{D}_i$ should be less than the maximum delay requirement $\mathcal{D}_{max}$. The resource requirement $\mathcal{R}_i^t$ should be greater than the minimum resource requirement $\mathcal{R}_i^{min,t}$ as defined in (9). (10) demonstrates that the profit factor Θ_i^t should be greater than the threshold profit level Θ_{th}. We take into consideration of Lagrangian multiplier to get the solution of the objective function. By applying the method, we get the following equation,

$$\Theta_{\mathbb{U}} = \sum_{i=1}^{n} \frac{1}{\mathbb{U}_{th}} \mathbb{U}_i(\mathcal{R}_i, \mathcal{D}_i, t) - \xi_1 \left(\sum_{i=1}^{n} \mathcal{D}_i - \mathcal{D}_{max} \right)$$
$$- \xi_2 \left(\sum_{i=1}^{n} \mathcal{R}_i^t - \mathcal{R}_i^{min,t} \right) - \xi_3 \left(\sum_{i=1}^{n} \mathcal{Q}_i - \mathcal{Q}_{th} \right)$$

where ξ_1, ξ_2 and ξ_3 are the balancing factors for Lagrangian method. Our primary intention is to get the optimal value of $\mathbb{U}_i$ by following the Lagrangian method. After getting the Lagrangian equation, we solve it using the gradient descent method. The Lagrangian mathematical expressed as:

$$\mathcal{L}_{\Theta_{\mathbb{U}}} = \sum_{i=1}^{n} \frac{1}{\mathbb{U}_{th}} \mathcal{L}_{\Theta_{\mathbb{U}}} \Big(\mathbb{U}_i(\mathcal{R}_i, \mathcal{D}_i, t) \Big) - \xi_1 \left(\sum_{i=1}^{n} \mathcal{D}_i - \mathcal{D}_{max} \right)$$
$$- \xi_2 \left(\sum_{i=1}^{n} \mathcal{R}_i^t - \mathcal{R}_i^{min,t} \right) - \xi_3 \left(\sum_{i=1}^{n} \mathcal{Q}_i - \mathcal{Q}_{th} \right)$$

We consider the gradient decent method to optimize $\mathcal{L}_{\mathcal{U}}$. Thus, we get the subsequent equations.

$$\frac{\delta \mathcal{L}_{\Theta_{\mathbb{U}}}}{\delta \mathcal{D}_i} = \sum_{i=1}^{n} \frac{1}{\mathbb{U}_{th}} \frac{\delta \mathcal{L}_{\Theta_{\mathbb{U}}} \Big(\mathbb{U}_i(\mathcal{R}_i, \mathcal{D}_i, t) \Big)}{\delta \mathcal{D}_i} \quad (11)$$

$$\frac{\delta \mathcal{L}_{\Theta_{\mathbb{U}}}}{\delta \mathcal{R}_i^t} = \sum_{i=1}^{n} \frac{1}{\mathbb{U}_{th}} \frac{\delta \mathcal{L}_{\Theta_{\mathbb{U}}} \Big(\mathbb{U}_i(\mathcal{R}_i, \mathcal{D}_i, t) \Big)}{\delta \mathcal{R}_i^t} \quad (12)$$

Following the upper mentioned equations, we acquire the optimal value of $\mathcal{L}_{\Theta_{\mathbb{U}}}$ for IIoT devices.

We elaborate the algorithm for microservice utility maximization for IIoT devices. As illustrated in Algorithm 1, in the beginning, we requisite to supply four inputs – IIoT devices $\mathcal{N}$, edge servers $\mathcal{M}$, microservice flows $\mathcal{S}$ and total time $\mathcal{T}$. To feed the resource demand to IIoT devices, we present microservice utility maximization scheme to optimize the microservice latency and resource demand. To begin with, we specify the waiting time τ_{wait} to 0. Afterward, we continue the algorithm for several rounds (i.e., 200 runs) for each IIoT device i. If the cumulative time is greater than or equal to 0, i.e., $\mathcal{T} \geq 0$, in such situation we approximate the total microservice latency $\mathcal{D}_i$ and determine the start time of a microservice $\mathcal{S}_t$. Thereafter, we approximate the mean data packet size in a microservice flow $\mathcal{F}_{si}$ and determine the variance of data packet sizes in a microservice flow $\mathcal{V}_{siz}$. Also, approximate the average data packet inter-arrival time M_{int} and determine the resource agnostic factor for each IIoT device γ_i. Following, we approximate the resource requirement $\mathcal{R}_i$ and variation in resource response rate of IIoT devices $\mathcal{H}_i^t$. Following the determined parameters, we formulate a utility function $\mathbb{U}_i$ for resource-agnostic microservice offloading. While the utility value $\mathbb{U}_i$ is higher than the average threshold value $\mathbb{U}_{th}$, followed by a degradation in the IIoT devices $\bar{\mathcal{N}} = \mathcal{N} \cap i$. Further, we upgrade the time τ_{wait}. Applying the solution for the optimization problem, we attain optimal resource demand $\Upsilon_{\bar{\mathcal{R}_i}}(t)$. The algorithm is discontinued, if the waiting time meets a set maximum waiting time τ_{wait}^{max}. We get the optimal utility value from Equation (7) using the Lagrangian method.

Algorithm 1: Microservice Utility Maximization

```
Input: Set of IIoT devices ($\mathcal{N}$), set of edge servers ($\mathcal{M}$), set of
       sub-microservice flows $\mathcal{S}$ and time period $\mathcal{T}$.
Output: Accurate resource demand $\Upsilon_{\bar{\mathcal{R}_i}}(t)$ and $\tau_{wait}$.
1  Specify $\tau_{wait} = 0$;
2  Specify $\mathcal{N} = n$, $\mathcal{M} = m$, $\mathcal{S} = k$;
3  for consider IIoT device i in edge platform do
4      Consider the microserice flow for each IIoT device;
5      if $\mathcal{T} \geq 0$ & $\tau_{wait} \leq \tau_{wait}^{max}$ then
6          Determine the average microservice delay $\mathcal{D}_i$;
7          Determine start time of a microservice $\mathcal{S}_t$;
8          Approximate mean data packet size in a microservice flow $\mathcal{F}_{si}$;
9          Calculate variance of data packet sizes in a microservice flow $\mathcal{V}_{siz}$;
10         Approximate average data packet inter-arrival time $M_{int}$;
11         Determine resource agnostic factor for each IIoT device $\gamma_i$;
12         Approximate resource requirement $\mathcal{R}_i$;
13         Changing resource response rate of IIoT devices $\mathcal{H}_i^t$;
14         Design utility function $\mathbb{U}_i$;
15         if $\mathbb{U}_i \geq \mathbb{U}_{th}$ then
16             Upgraded set of IIoT devices $\bar{\mathcal{N}} = \mathcal{N} \cap i$;
17             Optimal resource demand $\Upsilon_{\bar{\mathcal{R}_i}}(t)$;
18             Upgrade waiting time $\tau_{wait} = \tau_{wait}$. else
19                 Upgraded set of IIoT devices $\bar{\mathcal{N}} = \mathcal{N}$;
20                 Non-optimal price cost $(\widehat{\Upsilon_{\mathcal{R}_i}(t)})$;
21                 Upgrade waiting time $\tau_{wait} = \tau_{wait} + 1$;
22 Return $\Upsilon_{\bar{\mathcal{R}_i}}(t)$ and $\tau_{wait}$;
```

V. Optimal Microservice Offloading

After distributing the resources among IIoT devices, the microservices should be offloaded to the edge servers for further processing. Hence, in this section, we design an optimal microservice offloading mechanism. The microservice offloading mechanism collects the information from the resource-agnostic model to make the decision about offloading with RAISE. The offloading decision mechanism would design an optimization problem if the microservice flow completion remarks and execution time could be precisely estimated immediately after the mobile IIoT devices start their applications. The optimization depends on the total load of edge server and the microservice flow rate between the IIoT devices and the edge servers. The offloading decision metric $\mathcal{O}_{ij}^t$ is defined as:

$$\mathcal{O}_{ij}^t = \left(\mathcal{Y}_j^t - \frac{\mathcal{A}_{ij} \mathcal{S}_{ij}^t f_{ij}(t)}{\mathcal{E}_j \mathcal{J}_j^{cap}} \right) \tag{13}$$

where $\mathcal{Y}_i^t$ denotes the total load of server i at time t, $\mathcal{A}_{ij}$ denotes the microservice flow constraint, $\mathcal{S}_{ij}^t$ denotes the microservice flow rate at time t, $\mathcal{J}_j^{cap}$ denotes the capacity of server j and $f_{ij}(t)$ signifies the dedicated flows. Hence, the optimization problem is defined as:

$$\text{Maximize} \quad \mathcal{O}_{ij}^t = \left[\mathcal{Y}_i^t - \frac{\mathcal{A}_{ij} \mathcal{S}_{ij}^t f_{ij}(t)}{\mathcal{E}_j \mathcal{J}_j^{cap}} \right] \tag{14}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \mathcal{Y}_i^t \geq \mathcal{Y}_{th}^t, i \in \mathcal{N}, t \in T \tag{15}$$

$$\sum_{i=1}^{n} \mathcal{S}_{ij}^t \geq \mathcal{S}_{th}^t, i \in \mathcal{N}, t \in T \tag{16}$$

$$\sum_{j=1}^{m} \mathcal{J}_j^{cap} \geq \mathcal{J}_{th}^{cap}, j \in \mathcal{M} \tag{17}$$

$$\sum_{i=1}^{n} \mathcal{A}_{ij} \geq \mathcal{A}_{th}, j \in \mathcal{M} \tag{18}$$

We illustrate the optimization problem in detail. (14) demonstrates the primary optimization problem. (15) signifies that the present load of the server $\mathcal{Y}_i^t$ should be greater than the threshold load $\mathcal{Y}_{th}^t$. The microservice flow rate $\mathcal{S}_{ij}^t$ should be greater than the threshold microservice flow rate $\mathcal{S}_{th}^t$ as depicted in (16). (17) demonstrates that the capacity of server $\mathcal{J}_j^{cap}$ should be greater than the threshold capacity of server $\mathcal{J}_{th}^{cap}$. (18) demonstrates that the microservice flow rate $\mathcal{A}_{ij}$ should be greater than the threshold flow rate $\mathcal{A}_{th}$. We take into consideration of Lagrangian multiplier to get the solution of the objective function. By applying the method, we get the following equation,

$$\begin{aligned} \nabla_{\mathcal{O}} = &\sum_{i=1}^{n} \sum_{j=1}^{m} \frac{\Psi_i}{\mathcal{O}_{th}} \mathcal{O}_{ij}^t \left(\mathcal{Y}_i^t, \mathcal{A}_{ij}, \mathcal{S}_{ij}^t, f_{ij}(t), \mathcal{E}_j, \mathcal{J}_j^{cap} \right) \\ &- \xi_1^{''} \left(\sum_{i=1}^{n} \mathcal{Y}_i^t - \mathcal{Y}_{th} \right) - \xi_2^{''} \left(\sum_{i=1}^{n} \sum_{j=1}^{m} \mathcal{S}_{ij}^t - \mathcal{S}_{th}^t \right) \\ &- \xi_3^{''} \left(\sum_{j=1}^{m} \mathcal{J}_j^{cap} - \mathcal{J}_{th}^{cap} \right) - \xi_4^{''} \left(\sum_{i=1}^{n} \sum_{j=1}^{m} \mathcal{A}_{ij} - \mathcal{A}_{th} \right). \end{aligned}$$

here $\xi_1^{''}$, $\xi_2^{''}$, $\xi_3^{''}$ and $\xi_4^{''}$ illustrate the balancing elements for Lagrangian method. Ψ_i signifies the priority values of microservices in IIoT devices. The crucial motive is to optimize $\mathcal{O}_{ij}^t$. After getting the Lagrangian equation, we solve it using the gradient descent method. The Lagrangian mathematical expressed as:

$$\begin{aligned} \mathcal{L}_{\nabla_{\mathcal{O}}} = &\sum_{i=1}^{n} \sum_{j=1}^{m} \frac{\Psi_i}{\mathcal{O}_{th}} \mathcal{L}_{\mathcal{O}} \left(\mathcal{Y}_i^t, \mathcal{A}_{ij}, \mathcal{S}_{ij}^t, f_{ij}(t), \mathcal{E}_j, \mathcal{J}_j^{cap} \right) \\ &- \xi_1^{''} \left(\sum_{i=1}^{n} \mathcal{Y}_i^t - \mathcal{Y}_{th} \right) - \xi_2^{''} \left(\sum_{i=1}^{n} \sum_{j=1}^{m} \mathcal{S}_{ij}^t - \mathcal{S}_{th}^t \right) \\ &- \xi_3^{''} \left(\sum_{j=1}^{m} \mathcal{J}_j^{cap} - \mathcal{J}_{th}^{cap} \right) - \xi_4^{''} \left(\sum_{i=1}^{n} \sum_{j=1}^{m} \mathcal{A}_{ij} - \mathcal{A}_{th} \right). \end{aligned}$$

We focus on to get the optimal value of $\mathcal{L}_{\mathcal{O}}$ using Lagrange method. Thus, we get the subsequent equations.

$$\frac{\delta \mathcal{L}_{\nabla_{\mathcal{O}}}}{\delta \mathcal{Y}_i^t} = \sum_{i \in n} - \frac{\Psi_i \mathcal{L}_{\mathcal{O}} \left(\mathcal{Y}_i^t, \mathcal{A}_{ij}, \mathcal{S}_{ij}^t, f_{ij}(t), \mathcal{E}_j, \mathcal{J}_j^{cap} \right)}{\mathcal{Y}_i^{t^2}}$$

$$\frac{\delta \mathcal{L}_{\nabla_{\mathcal{O}}}}{\delta \mathcal{J}_j^{cap}} = \sum_{j \in m} \frac{\Psi_i}{\mathcal{O}_{th}} \frac{\delta \mathcal{L}_{\mathcal{O}} \left(\mathcal{Y}_i^t, \mathcal{A}_{ij}, \mathcal{S}_{ij}^t, f_{ij}(t), \mathcal{E}_j, \mathcal{J}_j^{cap} \right)}{\delta \mathcal{J}_j^{cap}}$$

$$\frac{\delta \mathcal{L}_{\nabla_{\mathcal{O}}}}{\delta \mathcal{S}_{ij}^t} = \sum_{i \in n, j \in m} \frac{\Psi_i}{\mathcal{O}_{th}} \frac{\delta \mathcal{L}_{\mathcal{O}} \left(\mathcal{Y}_i^t, \mathcal{A}_{ij}, \mathcal{S}_{ij}^t, f_{ij}(t), \mathcal{E}_j, \mathcal{J}_j^{cap} \right)}{\delta \mathcal{S}_{ij}^t}$$

$$\frac{\delta \mathcal{L}_{\nabla_{\mathcal{O}}}}{\delta \mathcal{A}_{ij}} = \sum_{i \in n, j \in m} \frac{\Psi_i}{\mathcal{O}_{th}} \frac{\delta \mathcal{L}_{\mathcal{O}} \left(\mathcal{Y}_i^t, \mathcal{A}_{ij}, \mathcal{S}_{ij}^t, f_{ij}(t), \mathcal{E}_j, \mathcal{J}_j^{cap} \right)}{\delta \mathcal{A}_{ij}}$$

Following the equations, we attain the optimal value of $\mathcal{L}_{\nabla_{\mathcal{O}}}$ to obtain the offloading decision metric for IIoT devices. We now propose a microservice offloading algorithm – *RAISE* for IIoT devices.

We elaborate the optimal microservice offloading algorithm for MEC as depicted in Algorithm 2. In the beginning, each IIoT device approximated their resource requirements and also their delay requirements. Following, each IIoT device determines their utility function. If the determined utility function $\mathbb{U}_i(\mathcal{R}_i, \mathcal{D}_i, \mathcal{P}_i, t)$ is greater than the threshold utility function $\mathbb{U}_{th}(\mathcal{R}_i, \mathcal{D}_i, \mathcal{P}_i, t)$. Then, the load of edge server is computed and also offloading decision metric $\mathcal{O}_{ij}^t$ is determined. If the offloading decision metric $\mathcal{O}_{ij}^t$ is less then the threshold offloading decision metric $\mathcal{O}_{th}^t$. Then, the microservice offloading time is computed, else we solve the objective problem to obtain the optimal offloading matrix $\mathcal{O}^*$.

Algorithm 2: Optimal Microservice Offloading

```
Input: Number of IIoT devices ($i \in \mathcal{N}$), Number of edge servers ($j \in \mathcal{M}$),
       Offloading time $\mathcal{T}_{low}$.
Output: Optimal offloading matrix ($\mathcal{O}^*$).
1  IIoT device estimate their resource requirement $\mathcal{R}_i$;
2  Approximate the resource-constrained factor $\gamma_i$ of $i^{th}$ device;
3  Determine the delay requirement $\mathcal{D}_i$ of $i^{th}$ device;
4  Determine the utility function $\mathbb{U}_i(\mathcal{R}_i, \mathcal{D}_i, \mathcal{P}_i, t)$ for device $i$;
5  for $i = 1$ to $N$ do
6      if $\mathbb{U}_i(\mathcal{R}_i, \mathcal{D}_i, \mathcal{P}_i, t) \geq \mathbb{U}_{th}(\mathcal{R}_i, \mathcal{D}_i, \mathcal{P}_i, t)$ then
7          Determine the load of the each edge server $\mathcal{Y}_i^t$;
8          Determine the offloading decision metric $\mathcal{O}_{ij}^t$;
9          if $\mathcal{O}_{ij}^t \leq \mathcal{O}_{th}^t$ then
10             Upgrade offloading time, $\mathcal{T}_{wait}^* = (\mathcal{T}_{low} + 1)$;
11         else
12             Solving the optimization problem, we obtain the optimal
               offloading matrix ($\mathcal{O}^* = \{\mathcal{O}_{ij}^t\}^*$) ;
13     Return when $\mathcal{T}_{wait}^* = \mathcal{T}_{tot}$;
```

1) Microservice Prioritization: After the estimation of resource requirements, we design a microservice priority queue to provide different priorities Ψ_i to IIoT devices in order to offload their computational microservices. The priorities are defined based on their microservice classes to minimize the effect of the starvation problem. Here, we have designed three simple priority queues. Critical microservices (i.e., real-time edge microservices) are given the highest priority, normal microservices are given a moderate priority, and background microservices are given the lowest priority. The queue in the higher phase gets to execute their microservices first, whereas the queue in the lower phase gets to execute their microservices after completion of the higher queues. The continuous priorities derived from these priority matrices can provide fair resource sharing and also provide optimal microservice offloading to microservices. On the other hand, contentious priority extraction increases the microservice overhead, which provides extra challenges to system designers. Therefore, we need to minimize the microservice overhead without requiring excessive extraction of priorities. Moreover, we elaborate on a microservice prioritization framework to eventually design an optimal microservice offloader.

Theorem 1. *The complexity of RAISE is $O(\mathcal{Z}n)$, n signifies the count of IIoT devices.*

Proof. In RAISE, the IIoT devices attempt to offload the microservices to edge servers to optimize the offloading latency. In the beginning, the IIoT devices approximate the resource and delay requirements. Moreover, these steps do not require any computational power. Thereafter, the IIoT devices check the conditions of the utility functions to offload the microservices. This step is performed for n times. Hence, the worst-case computation complexity to offload the computational microservices is $O(\Xi_1 n \log n)$. In addition, the complexity of utility maximization algorithm is $O(\Xi_2 n)$. Therefore, combining the complexity of optimal microservice offloading scheme, we have, $T(n) = \Xi_1 \mathcal{Z} n \log n + \Xi_2 n$. We calculate the total complexity of RAISE, i.e., $O(\mathcal{Z}n)$ and n signifies the IIoT devices. □

Table II: Simulation Parameters

Parameter	Value
Bandwidth of base stations	1.4 MHz
CPU cycles of microservice tasks [35]	1,500 Megacycles
Compute latency threshold	2.5 ms
Microservice completion time	[400, 600] ms
Resource requirement of microservices	[25, 30] units
Tran-energy of IIoT device [36]	90 mWatts
IIoT device compute competence [37]	0.85 GHz
MEC server compute competence	150 GHz
Incoming rate of microservices [36]	[0, 10] unit/sec
Microservice size [36]	150 Mbits
Execution rate of microservices	[50, 80] ms
Spin-up time of microservices	180 ms
Energy coefficient factor [38]	5^{-24}

VI. Experimental Results

We have implemented RAISE as a middleware at the network edge and here we describe each of the components in detail.

A. Experimental Setup

We explain the experimental setup for performance evaluation of RAISE in Table II. With align to existing literature [39], [40], we consider Arduino and Intel i7 CPU as example of IIoT device to acquire the practical values with regard to compute proficiency. We evaluate RAISE while considering 50 IIoT devices dispersed over a region of 4 km x 4 km. The edge devices communicate through different base stations and MEC servers are mainly collocated near to base stations (BSs) [41]. The compute competence of MEC server is 10^{12} CPU cycles/s and the compute competence of IIoT device is 0.85 GHz. The edge devices use several wireless channel to offload the microservices to edge servers. Here, we consider a distributed framework, where the edge platform are connected to each other using a backhaul network. The backhaul network rate is set to 0.0005 sec/KB [42]. The computing time of microservices for edge IIoT devices is set within $20 - 30$ ms. The compute fie size of a microservice is assign in the range of 500 KB to 1 MB. The latency factor of a IIoT device is defined in the range $1 - 1.5$ s. We get the real-time traffic of IIoT devices [43] using a D-ITG traffic generator [44]. We generate two types of topologies – 1) AttMpls and 2) Goodnet, while collecting different information from the Internet Topology Zoo [45].

The time-slot of offloading microservices are considered to be 1 ms. The data flow rate of IIoT devices is considered to be with the range of $400 - 1400$ kbps. To provide the latency guarantee to IIoT applications, the threshold latency factor is assigned to 2.5 ms. The minimum offloading rate is considered to be 1.5 MBps or more. The offloading data size of a microservice task is set to in the range of $200-1200$ KB. The CPU cycles of microservice tasks is set to 1,500 Megacycles. The transmission power of IIoT device is 90 mWatts. The deadline of microservices is set to $0.2-0.7$s and delay tolerant capability is set to 0.4s for critical microservices.

B. Workload

We implement RAISE in 15 edge server machines and each of them is enabled with Linux OS in an Intel core-i5 having CPU clock size of 2.8 GHz. In RAISE, we generate a pair of traffic work patterns, 1) delay-stringent (i.e., basically microservices) and 2) delay-lenient (i.e., normal applications) workloads. The maximum preference should be set for latency-stringent applications rather than for latency-lenient applications. Thus, for the RAISE framework, we operate critical microservices with maximum preference compared to normal microservices. To create the aforementioned workloads, we investigated three types of probability distributions: uniform, normal, and power, to see how they affected the RAISE's performance. To illustrate the effectiveness of RAISE, the power distribution shows the highly screwed workload typically used for online social networks considering power law [46].

C. Evaluation Metrics

We discuss the evaluation metrics used to observe the performance of RAISE. First, we consider the resource utilization

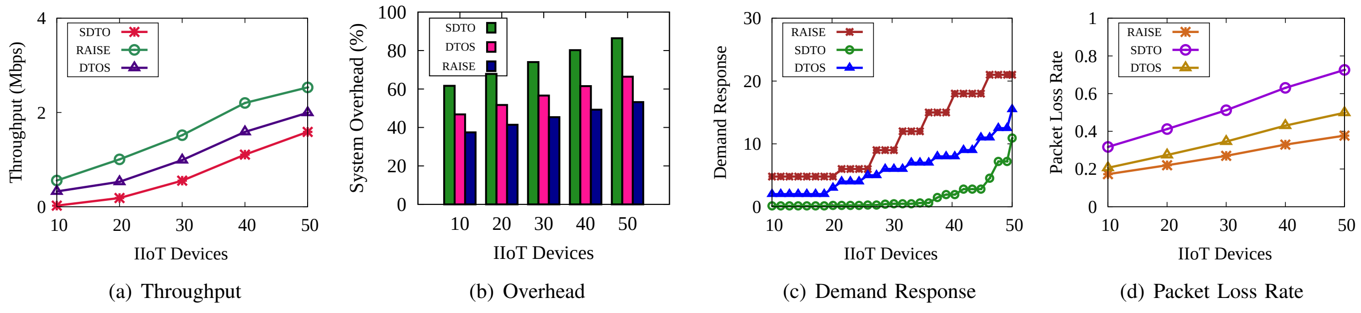


Figure 2: Analysis of throughput, system overhead, resource demand and loss rate

metrics for microservice offloading. The resource utilization metric can be designed as the ratio of resources assigned and the average resource requirement of IIoT devices. Along with that, we also design a success rate (SR) metric. It is dependent on the average count of offloaded microservices and the total count of microservices to be offloaded.

$$SR = \frac{\text{Average \# of microservices offloaded}}{\text{Total microservices to be offloaded}}$$

The metric with the higher quantitative values signifies the superior performance of RAISE, but it does not provide fair performance for microservices. We also compare RAISE to other state-of-the-arts in terms of reliability at a particular time t. Here, the QoS stands for goodput of the systems and the system overhead stands of how much time it takes to run the algorithms and its coverage time.

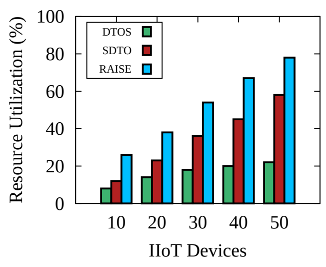
D. Benchmarks for RAISE

For performance evaluation of RAISE, we study two existing benchmarks - SDTO [47] and DTOS [48]. SDTO [47] leveraged the concept of SDN, they considered a task offloading method in ultra-dense network focusing on optimizing the delay and power consumption. Offloading tasks is difficult due to the compute resources on the edge platform and the power consumption of the edge devices.They considered a task offloading mechanism as NP-hard integer programming. To solve it, they rearranged it by dividing it into a pair of sub-categories, i.e., task assignment and resource allotment. Considering both the sub-categories, they proposed an optimal offloading scheme. DTOS [48] proposed a joint task offloading scheme (mapping from tasks to applications) and scheduling (execution order) scheme in order to deal the heterogeneity of tasks (with distinct resources, delay, etc.) and limited MEC capabilities. This scheme is regarded as a challenging combinatorial problem. They then decided on the resource allocation problem for heterogeneous applications and formulated it as a Dynamic Task Offloading and Scheduling problem. They mathematically formulated the problem and, due to its complexity, later devised a novel, thoughtful decomposition based on the Logic-Based Benders Decomposition technique.

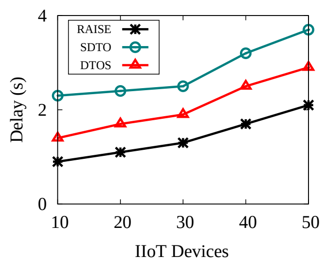
On the other hand, RAISE, we design a resource-agnostic microservice offloading with delay constraint for IIoT devices at the edge, while taking into account – delay, priority, completion time and resource-agnostic property, according to Algorithm 1 and 2. We try to find the optimal value while considering different local values in several rounds.

E. Discussion on Simulation Results

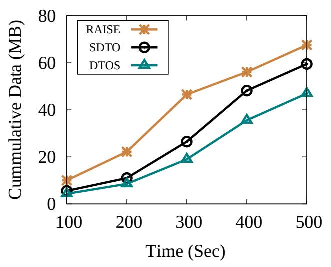
1) Impact on Network Throughput: We analyze the impact of network throughput in RAISE. Figure 2(a) signifies the network throughput for RAISE. As far as the figure is concerned, we note that network throughput is higher while there is an increase in the density of IIoT devices. As the number of IIoT devices has grown, the demand response of the mobile IIoT devices also increases as shown in Figure 2(c). Therefore, in such a situation, RAISE optimally distributes the resources among IIoT devices and also provides an optimal offloading scheme, which eventually increases the network throughput. As it can distribute the resources fairly, hence edge servers can execute more number of microsevices, which eventually increase the throughput. Hence, RAISE achieves better throughput the other methods DTOS and SDTO by 12%-15%. Figure 2(b) depicts the system overhead of RAISE. Following the figure, we can consider that system overhead is higher as more IIoT devices join the platform. The overhead for RAISE is low as its works in a online fashion and converge it relatively faster. By comparing with other methods, DTOS and SDTO, we can conclude that RAISE has less system overhead. The overhead of using RAISE is comparatively less than other methods with performance gain 13%-19%. Figure 2(c) illustrates that the demand response of IIoT devices is higher, as IIoT applications are generating multiple microservices. The microservice-level characteristics helps us to estimate the resource requirements of IIoT devices accurately and helps to keep a track of demand change. Hence, RAISE provides better peerofmance while estimating resource demand. However, RAISE outperforms the existing DTOS and SDTO methods. Hence, the resource demand of IIoT devices for RAISE is higher than other methods with performance gain 18%-28%. Figure 2(d) shows the packet loss rate for the IIoT devices. As the IIoT devices execute their microservices on optimal servers through the radio access networks (generally long-term-evolution). Hence, as the IIoT devices share the radio access channel for computational offloading, the packet loss is higher if more IIoT devices rejoin the network. However, it provides a lower packet loss rate than the other approaches. Hence, RAISE achieves less packet loss than the other methods, DTOS and SDTO with performance gain 8%-14 %.



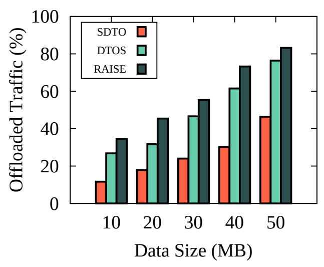
(a) Resource Utilization



(b) Delay

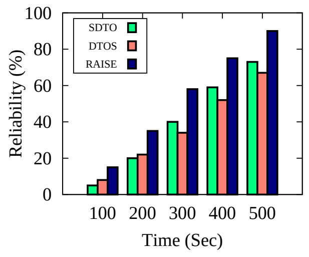


(c) Cumulative Data

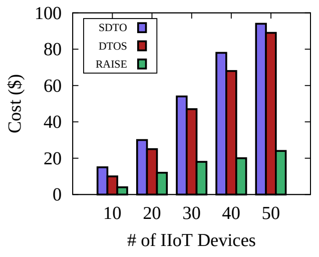


(d) Offloaded Traffic

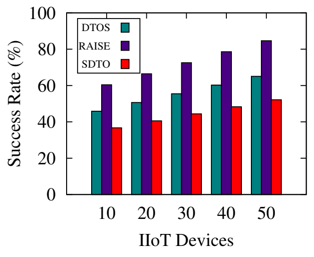
Figure 3: Analysis of resource utilization, delay, cumulative data and offloaded traffic



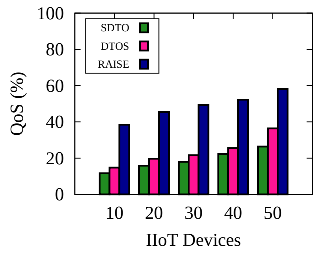
(a) Reliability



(b) Cost



(c) Success Rate



(d) QoS

Figure 4: Analysis of reliability, cost, success rate and QoS

Figure 3(a) represents the resource utilization for RAISE. We see in the figure that the resource utilization is higher with more number of IIoT devices. Here, RAISE efficiently estates the resource requirements of IIoT devices. Therefore, it can allocate resources to IIoT devices optimally according to their requirements. Hence, resource utilization has increased for the proposed scheme. The resource utilization of IIoT devices for RAISE is optimal and better than other methods with performance gain 13%-25%.

2) Effectiveness of Optimized Latency: Figure 3(b) represents the microservice delay in the platform. We observe that the microservice delay of the network increases if more IIoT devices try to execute their microservices. As the number of microservices increase in the network, then the IIoT devices contending for resources face the congestion, it automatically increases the microservice delay. However, we evaluate RAISE against other existing benchmarks and RAISE achieves less delay the other methods with performance gain 13% and 17%.

3) Impact on Microservice Offloading: Figure 3(c) shows the total cumulative data to be offloaded over a span of time for the proposed scheme - RAISE. The figure shows that the cumulative data on the network increases over time. As the total time increases, then the total number of applications running at mobile IIoT devices also increases during the peak hours. On the other hand, RAISE manages to provide resources for IIoT devices optimally. Thus, the total data being offloaded from IIoT devices also increases. Therefore, in such a situation, our proposed scheme - RAISE, provides higher cumulative data from each IIoT device than the other approaches. The existing approaches failed to offload the optimal microservices to optimal servers efficiently, hence the offloaded traffic of the proposed scheme - RAISE increases over other approaches. Hence, the cumulative data of the proposed scheme - RAISE is higher by 8%-10% than existing approaches - DTOS and SDTO. RAISE's advantage over the baseline does not increase that much over time, as like RAISE, both the existing schemes efficiently capture the data packets from IIoT devices. Figure 3(d) shows the total offloaded traffic for the varying data sizes in the network. The figure shows that the offloaded traffic increases as the data size of the network increases. RAISE manages to provide a lower packet loss rate to IIoT devices, hence the total offload traffic also increases. As the data size increases, then the IIoT devices offload more microservices to edge servers. Hence, RAISE outperforms the other approaches by 8%-17%.

4) Impact on Reliability: Figure 4(a) shows the reliability of the network for a particular duration of time. Here, the reliability is defined as the percentage of successful computational microservice offloading. From the figure, we see that the reliability of the network increases over time. RAISE offloads the computational microservices optimally, hence the packet loss rate decreases in the network as shown in 2(d). Therefore, the reliability of the network increases. The existing approaches failed to provide optimal microservices to IIoT devices in terms of packet loss rate, hence the reliability of the network decreases. However, RAISE provides higher reliability by 6%-8% than the other approaches.

5) Cost Analysis: Figure 4(b) shows the cost incurred for the allocated resources among IIoT devices. The cost is calculated using the Equation 4. From the figure, we see that the cost incurred is much less than in the other approaches - DTOS and SDTO. Because the proposed approach - RAISE optimized the cost of resource allocation, the incurred cost by the proposed approach - RAISE is significantly lower.

However, the existing approaches - DTOS and SDTO are mainly focused on computation microservice offloading. They have failed to provide optimal resources and optimized costs for IIoT devices. Hence, the RAISE provides better performance than other existing methods with performance gain DTOS and SDTO by 35% and 40%, respectively. Figure 4(c) shows the success rate while reckoning more IIoT devices. As RAISE efficiently allocates the resource to IIoT devices and optimally offloads the microservices to edge servers. Hence, the success rate of the proposed scheme - RAISE increases over other approaches. The other approaches did not consider any resource-agnostic property for IIoT devices, hence the computational offloading mechanism may not be optimal in such conditions. Hence, the RAISE increases the success rate by 12%-20% than the other schemes - DTOS and SDTO. Figure 4(d) shows the QoS of RAISE for the varying number of IIoT devices. We can notice from the figure that as the number of IIoT devices increases the QoS of RAISE also increases. The proposed approach - RAISE optimally quantifies the resource requirements of IIoT devices, hence the computation microservice offloading becomes more efficient. As it can effectively map the microservices to right edge servers, hence it can effectively execute more number of microservies. Therefore, the QoS of the network increases. The other approaches - DTOS and SDTO have failed to provide optimal resources to IIoT devices, hence it can execute the microrservices effectively. Therefore, the QoS of the network is lower than the proposed approach - RAISE. Hence, the QoS of RAISE increases by 11%-13% than other approaches. For our scheme RAISE, the variance of error is very low, it is within the range of $\pm 1.8\%$. Hence, it's effect in QoS not much.

VII. Conclusion

In this paper, we present a novel scheme - RAISE for microservices at the edge, to supply fair resources to IIoT devices. RAISE proposes a resource-agnostic method for predicting the requirements of IIoT devices in order to maximize network throughput. Thereafter, we propose an optimal microservice offloading method to optimize the microservice delay of IIoT devices. The proposed scheme - RAISE efficiently offloads the computational microservices to edge servers to increase the system utilization. Rigorous and extensive simulation results show that RAISE achieves better network throughput than SDTO and DTOS. Though RAISE can efficiently estimate the resource requirements of microservices which assists the operator to derive a better offloading policy, however it generates additional costs, such as higher energy consumption cost. Also, RAISE can not handle any kind of failures, be it server-side or device-side. Hence, it important to design an fault-tolerant mechanism for microservices in edge platform while improving the overall performance. In the future, we would like to design a load balancing scheme for MEC. We also propose to have an optimal and salable microservice offloading scheme for MEC with fault-tolerance. Consequently, we propose to have a privacy-aware microservice offloading scheme for mobile IIoT devices to provide secured communication.

References

[1] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[2] A. Samanta and Y. Li, "Cost-effective microservice scaling at edge: poster," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 326–328.

[3] T. G. Nguyen, T. V. Phan, D. T. Hoang, T. N. Nguyen, and C. So-In, "Federated deep reinforcement learning for traffic monitoring in sdn-based iot networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 4, pp. 1048–1065, 2021.

[4] G. Castellano, F. Esposito, and F. Risso, "A distributed orchestration algorithm for edge computing resources with guarantees," in *IEEE INFOCOM*, 2019.

[5] B. Gao, Z. Zhou, F. Liu, and F. Xu, "Winning at the starting line: Joint network selection and service placement for mobile edge computing," in *IEEE INFOCOM*, 2019, pp. 1459–1467.

[6] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *IEEE INFOCOM*, 2019, pp. 514–522.

[7] A. Samanta, F. Esposito, and T. G. Nguyen, "Fault-tolerant mechanism for edge-based iot networks with demand uncertainty," *IEEE Internet of Things Journal*, vol. 8, no. 23, pp. 16 963–16 971, 2021.

[8] L. Tong, Y. Li, and W. Gao, "A Hierarchical Edge Cloud Architecture for Mobile Computing," in *IEEE INFOCOM*, 2016, pp. 1–9.

[9] V. Farhadi, F. Mehmeti, T. He, T. L. Porta, H. Khamfroush, S. Wang, and K. S. Chan, "Service placement and request scheduling for data-intensive applications in edge clouds," in *IEEE INFOCOM*, 2019.

[10] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing," in *IEEE INFOCOM*, 2019, pp. 2287–2295.

[11] P. Jin, X. Fei, Q. Zhang, F. Liu, and B. Li, "Latency-aware vnf chain deployment with efficient resource reuse at network edge," in *IEEE Conference on Computer Communications (INFOCOM)*, 2020, pp. 267–276.

[12] T. G. Nguyen, T. V. Phan, D. T. Hoang, H. H. Nguyen, and D. T. Le, "Deepplace: Deep reinforcement learning for adaptive flow rule placement in software-defined iot networks," *Computer Communications*, vol. 181, pp. 156–163, 2022.

[13] A. Samanta and J. Tang, "Dyme: Dynamic microservice scheduling in edge computing enabled iot," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6164–6174, 2020.

[14] G. Yu, P. Chen, and Z. Zheng, "Microscaler: Cost-effective scaling for microservice applications in the cloud with an online learning approach," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2020.

[15] X. Chen, Y. Bi, X. Chen, H. Zhao, N. Cheng, F. Li, and W. Cheng, "Dynamic service migration and request routing for microservice in multi-cell mobile edge computing," *IEEE Internet of Things Journal*, pp. 1–1, 2022.

[16] F. Liu, J. Chen, Q. Zhang, and B. Li, "Online mec offloading for v2v networks," *IEEE Transactions on Mobile Computing*, pp. 1–13, 2022.

[17] D. Alencar, C. Both, R. Antunes, H. Oliveira, E. Cerqueira, and D. Rosário, "Dynamic microservice allocation for virtual reality distribution with qoe support," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2021.

[18] M. Abdullah, W. Iqbal, J. L. Berral, J. Polo, and D. Carrera, "Burst-aware predictive autoscaling for containerized microservices," *IEEE Transactions on Services Computing*, pp. 1–1, 2020.

[19] Y. Niu, F. Liu, and Z. Li, "Load balancing across microservices," in *IEEE Conference on Computer Communications (INFOCOM)*, 2018, pp. 198–206.

[20] M. Gao, R. Shen, J. Li, S. Yan, Y. Li, J. Shi, Z. Han, and L. Zhuo, "Computation offloading with instantaneous load billing for mobile edge computing," *IEEE Transactions on Services Computing*, pp. 1–1, 2020.

[21] X. Hou, Z. Ren, J. Wang, W. Cheng, Y. Ren, K. Chen, and H. Zhang, "Reliable computation offloading for edge computing-enabled software-defined iov," *IEEE Internet of Things Journal*, pp. 1–1, 2020.

[22] C. Gong, F. Lin, X. Gong, and Y. Lu, "Intelligent cooperative edge computing in the internet of things," *IEEE Internet of Things Journal*, pp. 1–1, 2020.

[23] H. Li, J. Yu, H. Zhang, M. Yang, and H. Wang, "Privacy-preserving and distributed algorithms for modular exponentiation in iot with edge computing assistance," *IEEE Internet of Things Journal*, pp. 1–1, 2020.

[24] K. Li, “Computation offloading strategy optimization with multiple heterogeneous servers in mobile edge computing,” *IEEE Transactions on Sustainable Computing*, pp. 1–1, 2019.

[25] Q. Cui, J. Zhang, X. Zhang, K. Chen, X. Tao, and P. Zhang, “Online anticipatory proactive network association to mobile edge computing for iot,” *IEEE Transactions on Wireless Communications*, pp. 1–1, 2020.

[26] R. Fantacci and B. Picano, “A matching game with discard policy for virtual machines placement in hybrid cloud-edge architecture for industrial iot systems,” *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2020.

[27] L. Xu, W. Yin, X. Zhang, and Y. Yang, “Fairness-aware throughput maximization over cognitive heterogeneous noma networks for industrial cognitive iot,” *IEEE Transactions on Communications*, pp. 1–1, 2020.

[28] S. Qi, Y. Lu, Y. Zheng, Y. Li, and X. Chen, “Cpds: Enabling compressed and private data sharing for industrial iot over blockchain,” *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2020.

[29] D. A. Chekired, L. Khoukhi, and H. T. Mouftah, “Industrial iot data scheduling based on hierarchical fog computing: A key for enabling smart factory,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4590–4602, 2018.

[30] R. W. L. Coutinho and A. Boukerche, “Modeling and analysis of a shared edge caching system for connected cars and industrial iot-based applications,” *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 2003–2012, 2020.

[31] M. C. Lucas-Estañ and J. Gozalvez, “Load balancing for reliable self-organizing industrial iot networks,” *IEEE Transactions on Industrial Informatics*, vol. 15, no. 9, pp. 5052–5063, 2019.

[32] T. T. Nguyen and G. Armitage, “A Survey of Techniques for Internet Traffic Classification using Machine Learning,” *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[33] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, “CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark,” in *ACM SIGCOMM*, 2016, pp. 160–173.

[34] A. Samanta and Y. Li, “Deserve: Delay-agnostic service offloading in mobile edge clouds: Poster,” in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–2.

[35] A. Samanta, L. Jiao, M. Mühlhäuser, and L. Wang, “Incentivizing microservices for online resource sharing in edge clouds,” in *IEEE ICDCS*, 2019.

[36] A. Samanta and Z. Chang, “Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint,” *IEEE Internet of Things Journal*, 2019.

[37] A. Samanta, Z. Chang, and Z. Han, “Latency-Oblivious Distributed Task Scheduling for Mobile Edge Computing,” in *IEEE GLOBECOM*, 2018, pp. 1–7.

[38] N. Dao, T. Nguyen, M. Luong, T. Nguyen-Thanh, W. Na, and S. Cho, “Self-calibrated edge computation for unmodeled time-sensitive iot offloading traffic,” *IEEE Access*, pp. 1–1, 2020.

[39] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, “On reducing iot service delay via fog offloading,” *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 998–1010, 2018.

[40] D. Xu, A. Samanta, Y. Li, M. Ahmed, J. Li, and P. Hui, “Network coding for data delivery in caching at edge: Concept, model, and algorithms,” *IEEE Transactions on Vehicular Technology*, 2019.

[41] A. Samanta, Y. Li, and F. Esposito, “Battle of microservices: Towards Latency-Optimal heuristic scheduling for edge computing,” in *IEEE NetSoft*, 2019.

[42] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, “Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks,” *IEEE Access*, 2016.

[43] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, “Characterizing and classifying iot traffic in smart cities and campuses,” in *IEEE INFOCOM Workshops*, 2017.

[44] A. Botta, A. Dainotti, and A. Pescapé, “A tool for the generation of realistic network workload for emerging networking scenarios,” *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, 2012.

[45] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, “The internet topology zoo,” *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[46] L. Wang, L. Jiao, J. Li, and M. Mühlhäuser, “Online Resource Allocation for Arbitrary User Mobility in Distributed Edge Clouds,” in *IEEE ICDCS*, 2017.

[47] M. Chen and Y. Hao, “Task offloading for mobile edge computing in software defined ultra-dense network,” *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[48] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, “Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing,” *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 668–682, 2019.