Machine Learning based Timeliness-Guaranteed and Energy-Efficient Task Assignment in Edge Computing Systems

Tanmoy Sen and Haiying Shen
Department of Computer Science
University of Virginia
Email: {ts5xm, hs6ms}@virginia.edu

***Abstract*—The proliferation in the use of the Internet of Things (IoT) and Machine Learning (ML) techniques in edge computing systems have paved the way of using Intelligent Cognitive Assistants (ICA) for assisting people in working, learning, transportation, healthcare, and other activities. A challenge here is how to schedule application tasks between the three tiers in the edge computing system (i.e., remote cloud, fog and edge devices) according to several considered factors such as latency, energy, and bandwidth consumption. However, the state-of-the-art approaches for this challenge fall short in providing a schedule in real time for critical ICA tasks due to complex calculation phase. In this paper, we propose a novel ReInforcement Learning based Task Assignment approach, *RILTA*, that ensures the timeliness guaranteed execution of ICA tasks with high energy efficiency. We first formulate the task-scheduling problem in the edge computing systems considering timeliness and energy consumption in ICA applications. We then propose a heuristic for solving the problem and design the reinforcement model based on the output of the proposed heuristic. Our simulation results show that *RILTA* can reduce the task processing time and energy consumption with higher timeliness guarantee in comparison to other existing methods by $13-22\%$ and $1-10\%$ respectively.**

I. Introduction

The rapid development of Internet of Things (IoT) devices (e.g., smartphones, wearable smart devices, sensors, autonomous vehicles) and the artificial intelligence (AI) and machine learning (ML) techniques has paved the way to a future of using Intelligent Cognitive Assistant (ICA) applications to assist people in working, learning, transportation, healthcare and other activities. These ICA applications with their adaptability will augment human performance and productivity and, thus provide helpful digital hands in facing everyday challenges. Alexa from Amazon, SARA (Socially Aware Robot Assistant) from CMU, Siri from Apple, etc. bear the testimony of the emergence of such ICA applications.

In many cases, ICA needs to assist people in real time, and some even have life-or-death consequences that could put their human counterpart at risk. For example, medical IoT devices in a home must be able to predict a heart attack of the residents in time and urge the patient or his/her doctor quickly. An autonomous car must be able to detect a person crossing the street in front of the car and decide to stop in real time. Meanwhile, edge computing systems prove increasingly crucial for such real-time or low-latency ICA tasks. Unlike the cloud computing that stores and processes end-users' data in remote and centralized datacenters, which generates certain delay due to data transmission and network congestion, edge computing can help process data from IoT devices and provide real-time local data analysis. It brings the provision of services closer to the end-users by pooling the available resources at the edge of the network (e.g., smartphones, tablets, smart cars, base station, and routers) to provide more user-aware, resource-efficient, scalable and low-latency services [1]. Apple announced in 2017 that it would put machine learning accelerators into its top-end iPhone [2] and based on this announcement Gartner predicted that 80% of smartphones would be AI equipped by 2022. It is also predicted that almost 50% of IoT-created data would be stored, processed, analyzed and acted upon close to or at the edge of the network [3].

In spite of the many challenges in leveraging the edge computing to realize the ICA future, one critical challenge is achieving timeliness-guaranteed and energy-efficient edge processing [4]. As indicated in [4], there are still gaps in state-of-the-art edge system architectures that fail to address information delivery overload for ICA due to lacking timeliness. Moreover, IoT devices are constrained by limited power resources while ICA tasks may involve intensive computation and communication, which imposes formidable challenges of achieving low-power edge processing. In the edge computing domain, several task assignment schemes have been proposed that decide which tier (remote cloud, fog, and edge devices) to assign a task to reduce task latency and (or) energy consumption. These methods can be categorized into two groups based on their goals: (1) energy-aware task assignment [5–11] and (2) bandwidth-aware task assignment [12–16]. However, these schemes involve complex calculation (e.g., solving a dynamic programming problem), which make them fall short in providing a schedule in real time (or with timeliness guarantee) with energy efficiency for critical ICA tasks. Further, finding an assignment schedule that simultaneously consider multiple goals such as task latency and energy efficiency makes the calculation more complex, which generates higher latency and energy consumption in finding the assignment schedule.

To provide timeliness-guaranteed and energy-efficient edge processing, in this paper, we use machine learning to find a solution for the task-assignment problem for ICA applications.

978-1-7281-2365-3/19/$31.00 ©2019 IEEE

We first formulate the task assignment problem considering the timeliness and energy consumption related to the data transmission and task computation for ICA applications. We then propose a heuristic solution and finally based on the all possible outcomes from that solution we devise a ReInforcement Learning (RL) based Task Assignment approach (called *RILTA*) depending on the features of data and the computation environment. The features include the urgency of the computation tasks, the amount of the data, data transmission latency, task computation latency and resource demand of a task. Based on these features, *RILTA* determines the assignment locations (edge, fog or cloud) for tasks with the objectives to ensure timeliness guarantee and power efficiency in real time. Our contribution in this paper is as follows:

- We formulate a task-assignment problem addressing the issues of deadline awareness and energy efficiency as per requirement for ICA applications.
- To solve the formulated task assignment problem, we propose a heuristic solution where the nodes (edge, fog or cloud) gain a profit for executing an ICA application. This profit scheme ensures the node that gains the highest profit is responsible for guaranteeing deadline aware execution of the tasks with different priorities in an energy efficient way.
- Finally, we propose a ReInforcement Learning system *RILTA* based on the outcomes generated using our proposed heuristic in a simulated environment. *RILTA* takes advantage of independent learning of the environment by multiple agents in RL to reduce the running time of the assignment task itself.
  To the best of our knowledge, this is the first task assignment scheme in edge computing platform that uses RL based approach for making task assignment decision in near real time with high energy efficiency.

The rest of the paper is organized as follows. Section II presents the related work of task assignment in fog/cloud in edge computing. In Section III, we formulate the task assignment into a mathematical problem focusing the features of ICA applications. In Section IV, we propose our region based deadline aware and energy efficient solution heuristics for the formulated problem and *RILTA*. Section V presents the experimental evaluation of *RILTA* in comparison with other methods in a simulated environment. Finally, Section VI presents conclusion and discusses the future work.

II. Related Work

With the development of IoT and mobile-based application systems in recent years, task offloading to fog/cloud has received much attention. Many state-of-the-art works propose different task-offloading schemes with varying objectives alongside reducing computational latency. Based on the objectives, the previous works can be categorized into two groups based on their goals: (1) Energy-aware task assignment [5–11], (2) Bandwidth-aware task assignment [12–16].

A. Energy-aware Task Assignment

Chun et al. [5] proposed CloneCloud that uses a combination of static analysis and dynamic profiling to offload partial applications from the edge devices while optimizing execution time and energy use related to computation and communication in the edge-cloud environment. CloneCloud runs code analysis on the incoming applications and then depending on the expected workload and execution conditions (CPU speeds, network performance) decide on the part of application's execution to be offloaded to the cloud. Similarly, Kao et al. [6] formulated a task assignment problem among edge devices in a network and provided an online learning based algorithm, Hermes, to find the optimal strategy to minimize energy consumption of the edge devices with a certain cost guarantee which is defined as a function of execution and transmission latency in the edge-cloud environment. Similarly in the edge-cloud environment, Wang et al. [8] and Munoz et al. [7] proposed schemes suitable for applications such as face-detection which run over a fixed set of images where the amount of data processed is known beforehand, and the execution can be parallelized into multiple processes. In these schemes, they consider part of the application is offloaded to the cloud, with the objective of optimization of the amount of energy consumed by the edge devices caused by transmission of an application. Sardellitti et al. [9] studied application offloading in multiple devices with multiple input-output systems to a shared cloud server. The offloading scheme aims at minimizing the energy consumption of edge devices under latency and energy budget constraints in an edge-cloud scenario. For the same environment, Lyu et al. [10] proposed an offloading decision scheme to a nearby cloud server to maximize the quality of experience (QoE) in perspective of task completion time and energy reduction of the system. Furthermore, Dong et al. [11] developed two offloading algorithms within an energy efficient framework for smartphones, which decide the cloud server for offloading the tasks to jointly optimize guaranteed response time performance within a certain confidence interval and energy efficiency. Although all of the above approaches try to reduce task latency and (or) improve the energy efficiency of the edge devices, they involve a significant amount of computation in calculating the task assignment schedule itself, which makes them unsuitable for the ICA tasks with real-time and energy-efficiency requirements. The trade-off between energy consumption for local computation and remote communications is discussed in [17–19].

B. Bandwidth-aware Task Assignment

Messaoudi et al. [12] proposed a framework for offloading computation tasks from a mobile edge to a fog node with the objective of minimization of total task completion time for low-latency computation-offloading based applications in the edge-fog scenario. In their proposed decision mechanism jointly consider estimated Round-Trip-Time (RTT) (which measures available bandwidth) and energy consumption of edge devices for offloading a task to a candidate fog node with high CPU availability. Their system includes a service that continuously estimates the expected RTT based on change of bandwidth to feed the device decision to offload parts of an applications computing tasks. Yang et al. [14] studied optimizing task offloading decision for the tasks from multiple edge devices and fog nodes to the cloud with consideration of available bandwidth to achieve either high processing speed or throughput. Fan et al. [13] proposed an offloading mechanism for the edge-fog-cloud environment that calculates profit based on the transmission delay of the nodes located in different environment level with primary target to ensure completing every task within its deadline. In this approach, the profit is a revenue given to a computation entity (edge device, fog node or remote datacenter) for executing a task within its deadline and they define their transmission latency on the available network bandwidth in different layers of their environment. However, they used a pheromone value based genetic algorithm for making the assignment decision which does not ensure finding a solution in real time. Wang et al. [15] proposed a task scheduling approach called *HealthEdge* that sets different processing priorities for different tasks in an edge-cloud environment based on the collected data on human health status. In their proposed approach they estimate the total waiting time in a queue and transmission delay as a function of current available bandwidth of the network and try to reduce the total processing time in the order of task priority. Choudhari et al. [16] proposed a task scheduling algorithm for the edge-fog environment that efficiently prioritize tasks according to their delay tolerance levels, which results in decreasing the average response time and the total cost of task offloading. Here, they define the cost of data transfer by dividing the amount of data transferred by the available bandwidth in the assignment phase. However, none of these approaches take both the deadline-awareness of a task and limited energy of the edge devices into consideration.

III. Task Assignment Strategy for ICA Applications

ICA run intelligent applications to assist people in many areas such as healthcare and intelligent transport [4]. These applications usually are context-aware in a sense they need to make decisions accordingly based on the gathered information from the environment at a specific moment. Usually, a collected data item is used for decision making in multiple ICA applications. For example, road congestion data is used by multiple ICA applications regarding route recommendation, optimal velocity calculation and so on. Furthermore, data can also be shared among multiple tasks within one application. As an example, we can say in healthcare service, data from breathing rate abnormality can be used for prediction of both heart and asthma attack. For such data sharing, transferring data among the devices will generate a massive amount of traffic and increase task latency and energy consumption caused by data transmission. However, depending on the decision-making process most of these tasks demand a quick response. To address these issues, we formulate a task assignment problem with an objective to ensure timeliness guarantee with power efficiency. Our proposed system considers a three-tier cloud-fog-edge environment similar to [13] that includes a client or edge device layer, fog layer and cloud layer. We consider that each tier except the cloud layer is comprised of multiple devices, which we term as nodes in this paper. In this section, we present our task assignment problem using Generalized Assignment Problem (GAP) [20] after introducing the characteristics related to an incoming task and resources considered for the problem formulation. In our approach, we assume that the system has knowledge of:

- the mapping between bottom tier nodes and the upper-tier nodes. That is, each edge node knows the list of fog nodes or one cloud datacenter where it can offload the tasks. Similarly, the fog nodes know the list of edge nodes where they need to send the computation results and so on.
- the existing network delays between nodes of different tiers. Such knowledge can be obtained by deploying software that monitors the state of the fog nodes, latencies, and application instances, as proposed in [21].

A. Task Characteristics

In this subsection, we introduce the task characteristics we consider for our problem. In the three-tier edge-fog-cloud environment the tasks are generated from different ICA applications running on edge devices at the bottom-tier. Most of these tasks have a specific deadline because of providing support to people in certain decision making as per their service level agreement (SLA). Each task has the following characteristics:

1) Arrival time of the task.
2) Deadline of the task.
3) Data size. The data size refers to the amount of data each edge node needs to process. It is used for calculating the estimated time and energy consumption for both computation and transmission of a task.
4) Energy consumption demand
5) Demand for the processor of a task for execution.

B. Resource Characteristics

In our task allocation system, we only consider three types of resources: processor, network-bandwidth, and battery life. Each node keeps track of its processor to decide whether, with its available processing resources, it can execute a task. Furthermore, the node estimates the required energy to execute or transmit a task to make the processing energy efficient. Each node also keeps track of the available bandwidth and the reliability of the path connecting it to the fog or the cloud. This resource information is used to estimate the transmission delay for both the data and the task. If we have the individual failure probability for the network link between a node to the fog or cloud as P_{fail}, then the reliability (denoted by R_{Net}) of the network link in case of the existence of n nodes and links is $R_{net} = 1 - P_{fail}^{n}$.

In our model, we distinguish the path from the edge node to fog as one network and from fog to cloud as another network.

The path reliability of these two networks is denoted with R_F and R_C respectively. The concept of path reliability is helpful for both capturing any sort of jitter in the network and any noise introduced due to the mobility of the edge devices.

C. Task Assignment Problem

In this section, we formulate timeliness guaranteed and energy efficient computation task assignment problem for the ICA applications. Based on different characteristics introduced regarding tasks and resources, we formulated a problem to decide whether a task should be executed by edge device itself or it should be offloaded to any of the fog node/cloud.

D. Task Assignment Model

Profits Estimation Recall that our task assignment system considers both timeliness guarantee and energy efficiency. Based on the importance of these two factors, the system can give different weight denoted by β and $1-\beta$, respectively. Here, β $(0 \leq \beta \leq 1)$ stands for the reward weight associated for an node for ensuring timeliness guarantee. In general terms, β controls the tradeoff between delay and energy efficiency. The trade-off between the delay and energy efficiency is based on the assumption that edge devices are more energy constrained than the fog nodes and the cloud datacenter. If a task is to be executed within its deadline, it is better to be executed in the edge node itself as it avoids the transmission latency. However, if a task is executed in the edge node consumes significant energy due to the computation. Moreover, as only the edge nodes are constrained by the energy, we define a parameter n_{type}. n_{type} represents whether the node, n is constrained by energy, i.e., n is a cloud or fog dataceneter or not. We consider $n_{type} = 0$ if n is a cloud or fog datacenter, otherwise $n_{type} = 1$. Consequently, n_{type} decides the factor of $1-\beta$ should be considered while calculating the profit.

If task, t arrives at time $a(t)$ with deadline d(t) then on the basis of estimated end time of the task ($\widetilde{T}_{end}(t)$), estimated energy consumption of node n for running the task ($\widetilde{E}(t,n)$) and energy stored in a node at the beginning of the task ($E_{stored}(n)$), the profit of task t for its assignment on node n is calculated as:

$$p(t,n) = (\beta \times \frac{(d(t) - \widetilde{T}_{end}(t))}{d(t) - a(t)} \\ +(1-\beta) \times n_{type} \times \frac{E_{stored}(n)}{\widetilde{E}(t,n)}) \tag{1}$$

Here, β is a reward factor for the nodes to finish a task in the earliest possible time. $d(t) - \widetilde{T}_{end}(t)$ means the time duration task t finishes before its deadline. The larger the difference, the better in terms of the timeliness guarantee. The denominator $d(t) - a(t)$ means the total allowed duration to complete task t. The ratio $\frac{(d(t) - \widetilde{T}_{end}(t))}{d(t) - a(t)}$ refers how much faster task t is executed by node n relative to its limited allowed duration. $\frac{E_{stored}(n)}{\widetilde{E}_{exe}(t,n)}$ refers to the how much energy is consumed by a node n for a task t relative to the node's stored energy. The lower the estimated energy consumption $\widetilde{E}_{exe}(t,n)$, the higher the reward for energy efficiency. And, the higher the stored energy $E_{stored}(n)$, the higher the reward in order not to drain the node's energy.

The estimated time $\widetilde{T}_{end}(t)$ is dependent on three factors: task t's arrival time $a(t)$, task $t's$ estimated execution time $\widetilde{T}_{exe}(t)$ and the estimated transmission time $\widetilde{T}_{trans}(t,n)$:

$$\widetilde{T}_{end}(t) = \tau_c + \widetilde{T}_{exe}(t) + \widetilde{T}_{trans}(t,n) \tag{2}$$

Here, τ_c is the current timestamp whereas $\widetilde{T}_{exe}(t)$ and $\widetilde{T}_{trans}(t,n)$ represents estimated execution time of task t and estimated transmission time interval of task t from or to node n respectively.

According to [13], $\widetilde{T}_{exe}(t)$ can be calculated using execution time for processing one unit of data based on the one unit of computing resource ($T_{exe}(1)$) and CPU demand by the task t ($c(t)$) as per follows:

$$\widetilde{T}_{exe}(t) = \frac{T_{exe}(1) \times D(t)}{c(t)} \tag{3}$$

Here, $D(t)$ indicates the t's dataset size which needs to be transmitted during task execution after the assignment to a node of any layer.

The estimation of transmission delay, $\widetilde{T}_{trans}(t,n)$ is the round-trip time required to transmit the task itself and its associated dataset from task-generator node to the task-assigned node, and to transmit the computation result from the latter to the former node. This estimated time is dependent on the dataset size, bandwidth and path reliability of the network between the source and destination nodes and associated transmission path delays induced by the routers or switches of the network [22]. We consider the bandwidth and reliability of the network between the edge nodes and fog nodes as B_F and R_F respectively. Furthermore, considering the bandwidth and reliability of the network between the fog and cloud as B_C and R_C respectively we adopt the equation for calculating the transmission delay in [13].

$$\widetilde{T}_{trans}(t,n) = \begin{cases} 0; \text{if } n \text{ is the task-generator edge node} \\ \frac{D(t)}{B_F} \times \frac{1}{R_F} + T_F, \text{ if } n \text{ is fog} \\ \frac{D(t)}{B_F} \times \frac{1}{R_F} + \frac{D(t)}{B_C} \times \frac{1}{R_C} + T_F + T_D, \\ \text{if } n \text{ is cloud} \end{cases} \tag{4}$$

where the delay between edge to fog and fog to cloud layer are represented by T_F and T_C respectively. If a task is assigned to a second-tier fog, it needs to be transmitted to the fog node and the transmission time is primarily dependent on the size of the data ($D(t)$) and the available bandwidth of the link from edge node to the fog (B_F) subject to the reliability (R_F) of the network. Besides, it considers the path delay (T_F) induced by the network devices, e.g., routers and switches. In the three-tiered edge-fog-cloud environment, any transmission from an edge node to the cloud needs to pass through the second-tier fog [13, 23]. As a result, while n is a cloud datacenter, we need to take into account both the path delay and data transfer delay from an edge node to fog and from fog to the cloud [13, 23, 24]. Therefore, when a task is transferred from the edge node to the cloud, we need to additionally consider the transmission time from fog to cloud dependent on the available bandwidth of the link from fog to the cloud (B_C) subject to the reliability (R_C) with additional path delay (T_D) induced by the network devices.

$\widetilde{E}_{exe}(t,n)$ is only dependent on two factors: the transmission time of the dataset and the execution time of a task.

$$\widetilde{E}(t,n) = \widetilde{T}_{trans}(t,n) \times E_{trans}(1) + \widetilde{T}_{exe}(t) \times E_{exe}(1)) \quad (5)$$

In the Equation above, $E_{trans}(1)$ and $E_{exe}(1)$ represents energy consumption for transmitting for one unit time and energy consumption for processing one unit of data based on the one unit of computing resource respectively. In Equation (1), we try to maximize the overall profit gained for assigning a task, t to a node and the profit should be reflective of our goal of timeliness guarantee and energy efficiency. The first fractional part ensures the assigned node is responsible for finishing a task before the deadline. A higher task estimated value $\widetilde{T}_{end}(t)$ than the deadline $d(t)$ will end-up assigning a node negative profit which means penalizing a node for violating a timeliness guarantee. By dividing by the $d(t) - \widetilde{T}_{end}(t)$ the node gets higher profit for finishing a task early as possible. Similarly, in the second fraction of Equation (1), we ensure the node which ensures higher energy efficiency gets higher profit. The lower energy a node, n consumes for executing a task, t in respect to its current stored energy it means the more energy efficient these nodes for this task. Consequently, $\frac{E_{stored}(n)}{\widetilde{E}_{exe}(t,n)}$ ensures the node, n receives higher profit for being energy efficient.

Problem Formulation The objective of this task assignment is to maximize the profits by ensuring timeliness guarantee with energy efficiency. Given three sets i) nodes working at the bottom layer (N_{edge}) ii) nodes working at the fog layer (N_{fog}) and iii) a datacenter in the cloud N_{cloud} we can define a larger Node Set, $\mathbb{N} = \{N_{edge}\} \cup \{N_{fog}\} \cup \{N_{cloud}\}$. Now for pending task sets (T_p) and running task sets (T_r) using GAP, our task assignment problem can be formulated as follows:

$$\text{Maximize} \sum_{t \in T_p} \sum_{n \in N} p(t,n) \times x(t,n) \quad (6)$$

$$\text{Subject to: } \forall t \in T_p, \forall n \in \mathbb{N} \; x(t,n) \in \{0,1\} \quad (7)$$

$$\forall t \in T_p \cup T_r, \sum_{n \in \mathbb{N}} x(t,n) = 1 \quad (8)$$

$$\forall n \in \mathbb{N}, \sum_{t \in T_p} c(t) \times x(t,n) + \sum_{t \in T_r} c(t) \leq C(n) \quad (9)$$

$$\forall t \in T_r \cap n \in \mathbb{N} \cap x(t,n) = 1, \widetilde{T}_{end}(t) \leq d(t) \quad (10)$$

Among the Equations (6) to (10), $x(t,n) = 1$ if task t is assigned to node n otherwise it is set to zero. And $C(n)$ represents the amount of computing resources of any node $n \in \mathbb{N}$ $(N_{edge}, N_{fog}, N_{cloud})$ Finally, Equation (6) represents the GAP problem formulation to our required scenario for ICA applications. Equation (7) and (8) cumulatively denotes the constraint that each task is assigned to only one node. Equation (9) assures that after the task assignment no node exceeds its available resources. Our goal is to provide timeliness guarantee for type of tasks which need both tight-time bound and flexible time bound. Equation (6) ensure this by providing a profit for timeliness guaranteed task assignment. In this case, we add an additional constraint from GAP problem as Equation (10) to ensure that estimated time never exceeds the deadline of a task to ensure the term $d(t) - \widetilde{T}_{end}(t)$ to be non-negative.

IV. Proposed Problem Solution

A. *Profit based Assignment*

As this task assignment problem is NP-hard [13], in case of large-scale applications involving lots of nodes, the problem-solving time may be unacceptable. Considering the urgency of some ICA applications, the task assignment must be done in near-realtime. To reduce the cost of profit calculation, we consider partitioning the problem based on the geographical region. After partitioning the region, the centralized cloud uses the proposed heuristic for profit estimation from the possible set states and actions to generate a centralized table using ReInforcement Learning. Later this table distributed among the edge nodes which they use for making the assignment decision of an incoming task. As a first step of building the RL model, we estimate the profit for all the possible task assignment nodes for an incoming task. Based on the deadline and resource requirement of the task it estimates the task processing time and energy consumption of each option using Equation (2) and (5) respectively. After the estimation, it calculates the profit that can be achieved by the node as per Equation (1), and it assigns the task to the node that gives the maximum profit which can be the edge node itself, fog or the cloud. We consider this profit-based assignment as an optimal solution to the task assignment problem.

B. *Machine Learning based Task Assignment*

For making real-time assignment decisions, we propose RL based Task Assignment approach (RILTA). We use the profit as the reward to build the RL model using *Q-learning* [25]. RL considers a general setting in which an agent interacts with an environment. At each time step τ, the agent observes some state s_τ and chooses an action a_τ. After applying the action, the state changes to $s_{\tau+1}$ and the agent receives a reward r_τ. This process forms a Markov Dynamic Process (MDP). The goal of this learning is to maximize the expected cumulative reward by maintaining a policy. In *RILTA*, a centralized server in the cloud trains the RL model using the collected state and reward information from individual edge nodes for their decisions on task assignment. Finally, *RILTA* builds a Q-value table that stores the cumulative reward achieved by an edge node for taking action dependent on its foreseen state space following the policy at each state. The Q-value table is distributed to the edge nodes, which they use for task assignment to itself, fog within their regions or the cloud based on the state. The cloud server keeps updating the Q-value table based on the task assignment activities of the edge nodes. Figure 1 illustrates the adaption of RL for our task assignment problem.

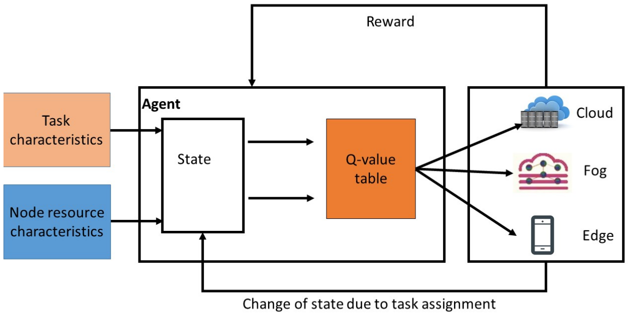
Reward
Agent
Task characteristics
Node resource characteristics
State
Q-value table
Cloud
Fog
Edge
Change of state due to task assignment

Fig. 1: Applying RL to task assignment

The mapping between Q-learning elements to our task assignment for ICA applications is as follows:

- **State space (S):** When an edge node has a task, the state includes all node resource characteristics (including available bandwidth, processor capacity, stored energy) of the edge node itself, the fog and the cloud, and task characteristics (Section III-A). For this purpose, the system discretize the continuous space of node resource characteristics (e.g., processor capacity and available bandwidth) by dividing their value range in different equi-width levels. For simplicity, assume the state space only consists of processor capacity and available bandwidth, if we consider l different levels of processor capacity and m levels for available bandwidth, the state space for task, t can be represented as a vector in the form as: $s_{(i,j)} = \{s_{(1,1)}, s_{(1,2)} \cdots s_{(1,m)}, s_{(2,1)}, s_{(2,2)}, \cdots s_{(2,m)}, \ldots s_{(l,1)}, s_{(l,2)}, \ldots s_{(l,m)}\} \in S$, where $s_{(i,j)}$ express the state by i and j that represents the level of CPU capacity and available bandwidth.
- **Action space (A):** Action space is the set of actions an edge node performs based on the state. The action space of a task can be represented by a vector in the form of $a_\tau = \{a_1, a_2, a_3\} \in A$, which means that an edge node assigns a task to itself, fog within its region or the cloud.
- **Immediate reward:** A reward function defines the immediate reward that is gained by an edge node for performing an action depending on its observation of the state. In *RILTA*, an edge device gains a reward (calculated by Equation (1)) from one of the chosen actions upon observation of the states at the arrival of a task.
- **Policy model:** Upon observing a node's state s_τ, an edge node needs to take an action a_τ that corresponds to the action of assigning an incoming task to itself, fog or the cloud. By referring to the Q-value table, the edge node selects the action based on a policy following distribution over actions defined as $Q : Q(s_\tau, a_\tau)$ that returns the future expected reward of that action at that state [25]. That is, the node selects the action that leads to the maximum future expected reward. Based on the MDP built from the environment observation, Q-learning finds the optimal action for each specific state that maximizes the cumulative rewards using the following function:

$$Q(s_\tau, a_\tau) = Q(s_\tau, a_\tau) + \alpha \times [r_{\tau+1} + \gamma \times max_{a_{\tau+1}} Q(s_{\tau+1}, a_{\tau+1}) - Q(s_\tau, a_\tau)] \tag{11}$$

where α is a learning rate parameter that facilitates convergence to the true Q-values in the presence of noisy or stochastic rewards and state transitions, and the discount rate denoted by γ guarantees the updated reward convergence for all the incoming tasks for an edge node.

This policy model is used to generate the Q-value table in the training phase. In the training phase, each edge node chooses an action on the observation of the current state based on the current Q-value estimates $Q(s_\tau, a_\tau)$. That is, it chooses the action corresponding to the highest $Q(s_\tau, a_\tau)$ value. It performs the chosen action and observe the outcome state $(s_{\tau+1})$ and the reward $r_{\tau+1}$. Based on these observations, the Q-value table is updated.

1) Q-value Table Creation in RILTA: In *RILTA*, the cloud server gathers information regarding the incoming task and node resource characteristics from all the edge nodes mentioned in Section III. Using the information, the cloud does the RL training by deriving the state corresponding to the task and node resource characteristics and calculates the reward associated with the actions for the state to build the Q-value table. The cloud server distributes this Q-value table to each edge node. Upon arrival of a task, at an edge node, it checks the state s_τ and finds the cumulative maximum reward $Q(s_\tau, a_\tau)$ from its Q-value table for that particular state and chooses action a_τ. That is, the edge node decides where to run its task: itself, fog or cloud using this mapped action. The edge nodes report the task assignment activities to the cloud server, which keeps updating the Q-value table. By using *RILTA*, for each edge node, upon observing the state, the task assignment problem gets limited to lookup the Q-value table and assign the task to the node with the highest reward. This lookup operation reduces the complexity of the task assignment problem and reduces the solution time to near real-time.

2) Generating the Q-Value Table in Experiments: The Q-value table should be created based on the edge nodes' real task assignment activities initially. In our experiment, in the whole Q-value table formation process, for each task, we derive the states and rewards regarding all possible task assignment locations calculated offline. For building the best possible policy to be adapted by the edge nodes using Q-learning, we create a simulated environment with 100 edge nodes, 12 fog nodes, and one cloud data center. The details of the settings for the node resource capacity and task characteristics are presented in Section V. Next, each edge node creates 100 tasks sequentially with an interval of 5 seconds. The Q-value table is initialized with zero. When an incoming task in an edge node, initially, we calculate the reward (using Equation (1)) for the task assignment to the edge node itself, fog and the cloud respectively. Based on the states, actions and rewards in the task assignments, using the policy formulation by Equation(11) with learning rate $\alpha = 0.01$, we build and update a Q-value table. After the Q-value table is formed, rather than trying all possible task assignment locations, observing the state at that timestep, we choose an action using ϵ-greedy policy [25] where ϵ is the exploration rate. Using this policy, we either select a random action with ϵ probability or select an action with $1 - \epsilon$ probability that gives the maximum reward in the given state. Based on the chosen action, we calculate the immediate reward using Equation (1) for performing the action. We also track the change of state for performing this action to build the MDP in Q-learning. Accordingly, in our training process, initially, we set the value of ϵ to 1 because we do not know anything about the values in the Q-value table. Then we reduce its value progressively as we get more Q values with the incoming tasks. Ultimately, this Q-value table records the maximum expected future reward an edge node can achieve for a certain action taken at a certain state. By referring to the Q-value table, when a task arrives, an edge node observes the current state and chooses the action corresponding to the highest cumulative reward. This whole Q-value table generation process is run off-line.

V. Performance Evaluation

A. Experiment Setup

We conducted our experiments on *iFogSim* [26] for testing the performance of our task assignment system. *iFogSim* is a toolkit that models and simulates IoT and Fog Computing environments together with the cloud. This simulator considers parameters such as network latency, bandwidth utilization, cost measurement, and energy consumption.

We conducted all our experiments using a three-tier edge-fog-cloud architecture. To simulate the nodes located in different tiers using iFogSim, we differentiated the processor capacities of the nodes in different tiers by setting the unit rate per MIPS of the processor as 1, 2.1 and 3.9 GHz to mimic edge devices, fog, and cloud respectively. In our experimentation, we used one cloud data-center, 12 fog nodes and varied the number of edge devices from 100 to 500.

In our experimentation, we assumed that edge devices could generate tasks at a certain arrival rate following a Poisson distribution [13]. Here, task arrival rate defines the number of tasks generated by an edge device in a unit time. Besides, the deadlines of the arriving tasks were set by adding a random value from Uniform distribution, $U(5, 125)$, to the arrival time of the individual tasks. For each set of experiment, we varied the processor demands of a task following a uniform distribution $U(50, 250)$ [13]. For preparing the state space of our proposed model *RILTA*, we divided the available bandwidth between different layers into three equi-width levels for the bandwidth value mentioned for each tier in Table I. Similarly, we divided the processor capacity for different nodes into three equi-width levels dependent on the different MIPS value of the nodes mentioned above. In general, all other simulation parameters are set according to the Table I. In our evaluation, we varied the number of tasks generated from each node in the range [100,500]. The execution of a task is halted whenever the task processing time exceeds its deadline. The whole simulation stops when there are no more tasks to execute in each edge node. The default values for the number of edge nodes, the number of tasks generated by each edge node, and the task arrival rate are 100, 500 and 200, respectively, unless otherwise specified.

TABLE I: Simulation parameters.

Parameters	Values
Bandwidth from edge nodes to fog	512 Mbps
Bandwidth from fog to cloud	1024 Mbps
Network delay between edge nodes to fog	50 milliseconds
Network delay between fog to cloud	100 milliseconds
Deadline of a task	$T_a + U(5, 125)$ in seconds
Dataset size of a task	Uniform distribution: U(100,1000) bytes
Failure probability of a node link	0.01
$T_{exe}(1)$	5 milliseconds
$E_{exe}(1)$	1.17 Joules
$E_{trans}(1)$	0.63 Joules
Reward Factor, β	Uniform distribution: $U(0.2, 0.8)$

B. Metrics

To evaluate our task assignment scheme, we used the following metrics:

1) **Timeliness guarantee ratio:** This is the percentage of tasks that finish within their deadlines.
2) **Percentage of deadline misses due to task assignment:** This is the percentage of the tasks that have higher task assignment time than their deadlines. This metric is used to show the performance of fast task assignment.
3) **Average energy consumption:** This is the average energy consumed by an edge device at the bottom-tier of the edge-fog-cloud architecture calculated from the statistics of total energy consumption using iFogSim.
4) **Task running time:** We calculate the running of all the tasks throughout the simulation process. We track the run time by the total simulation time starting at arrival of the first task and end of completion of the last task.
5) **Task allocation time:** This is the time spent on task allocation. We calculate the individual allocation time for each of the incoming tasks.

C. Compared Methods

We compared the performance of our proposed *RILTA* with two recently proposed assignment approaches SpanEdge [27] and Suspension-and Energy-Aware offloading algorithm

(*SEA*) [11]. We also show the performance of our heuristic solution as the optimal solution for reference.

- *SpanEdge* divides all the tasks into two groups: local and global tasks. A local task refers to the task which needs data only from its local edge node while a global task refers to the task that requires data from a set of local tasks spread among multiple edge nodes. *Spanedge* focuses on reducing the bandwidth consumption and response latency by assigning local tasks to the edge node or fog while assigning the global tasks in the distant cloud. Only when an edge node does not have enough resources to run the assigned task, the task is assigned to the fog. *Spanedge* identifies global tasks by processing a task dependency graph, which takes a long time, and also energy-efficiency is not its focus.
- The primary objective of *SEA* is to reduce energy consumption alongside guaranteeing bounded response time. This algorithm profiles on network and energy and decides on offloading the task to a cloud server or execute it on the edge device itself. For this purpose, *SEA* first finds various combinations of offloading decisions (that offloads different amount of workload in a task) with different energy consumptions, and then find the optimal solution that consumes lower energy over a period of time. However, the task assignment process has long latency.

D. Experimental Results

For recording the experimental plots, we run each set of experiment ten times. In each set of the run, we keep the simulation environment and parameters the same for all the compared methods. For each plot of our experiments, we show lower and upper error bar at 10^{th} and 90^{th} percentile.

1) Timeliness Guarantee: Figure 2 plots the timeliness guarantee ratio with the change of task arrival rate. For each task arrival rate, β is set based on a uniform distribution: $U(0.2, 0.8)$. From the figure, we observe in terms of ensuring timeliness guarantee ratios, the order of performance follows *Optimal* > *RILTA* > *SEA* >*SpanEdge*. *RILTA* performs $13-22\%$ better than *SEA* and *SpanEdge* and it performs $2-3\%$ worse than the optimal solution. The percentage is calculated by $|RILTA - method|/method$. *RILTA* performs better than others due to having lower task allocation time as it is limited to only finding an entry in the Q- value table. *SpanEdge* offloads all the global tasks to the cloud which contributes to its increased task completion time. The increase in task completion time occurs due to high transmission delay from an edge node to the cloud. *SEA* involves a complex computation phase as it needs to find a solution with different combinations to find the optimal solution. The optimal solution only needs to calculate the profits for allocating a task to the edge node, fog, and the cloud and chooses the option with the highest reward, which has lower computational complexity than *SEA*. However, its reward calculation process still contributes to its higher timeliness guarantee ratio than *RILTA*. However, *RILTA* cannot always find the optimal solution in all the cases due to the limitation in finding the best policy using *Q-learning*, which also causes its high variance in the error bars. From Figure 2, we can also observe that timeliness guarantee ratios decrease with the increase of task arrival rate. With the increase in the arrival rate, more tasks cannot be scheduled with limited resource constraints. However, *RILTA* still shows better performance than *SEA* and *SpanEdge* with the increasing arrival rate. In our evaluation, we also plot the percentage of tasks that missed their deadline due to high task allocation time. Figure 3 shows the percentage of the tasks missing deadlines out of all the tasks due to task assignment. From the figure, we can observe that *RILTA* performs $63-68\%$ better than *SpanEdge* and *SEA*. However, *RILTA* only performs $4-12\%$ better than the optimal solution. High task allocation time or decision-making time contributes to missing deadline of tasks in most of the cases for *SEA* and *SpanEdge*. *RILTA* is more efficient than other methods due to having task allocation time limited to finding an entry in the Q-value table to find the optimal task assignment action. *SpanEdge* has higher task allocation time because it needs to wait for the completion of the task dependency graph processing. On the other hand, *SEA* first finds various combinations of offloading decisions with different energy consumptions, and then find the optimal solution that consumes lower energy over a period of time, which contributes to its high task allocation time. On the contrary, the relatively lower allocation time of the optimal solution is caused by low computation complexity of the approach as explained previously. In Figure 3 we also see the increase in the percentage of deadline misses due to task assignment with the increase of arrival rate. This is caused by the same reason that more tasks cannot be scheduled with limited resource constraints. The lower task allocation time of *RILTA* leads to a lower percentage of deadline misses due to task assignment, thus enhancing the timeliness guarantee.

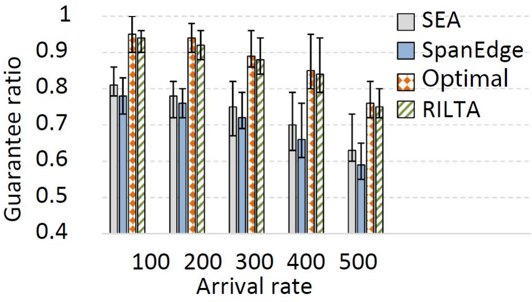

Fig. 2: Guarantee ratio.

2) Energy Consumption: Figure 4 shows the performance of *RILTA* with other methods in terms of average energy consumption versus the number of edge nodes. We decompose the result into two parts which are marked in the bar chart with different colors and patterns. The bottom part of each bar represents average energy spent by the edge nodes in the decision-making phase, and the upper part shows the amount of energy consumed in the task running phase. In terms of energy consumption, the result order is as follows: *RILTA* <*Optimal* < *SEA* <*SpanEdge*. In *RILTA*, an edge node

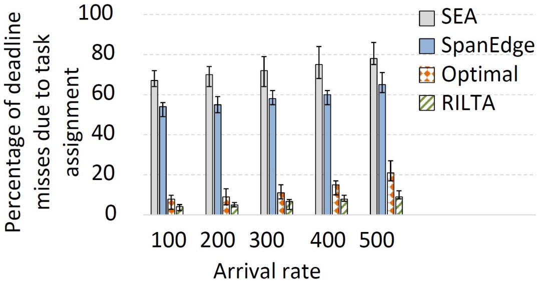

Fig. 3: Percentage of the tasks that missed the deadline due to task assignment.

consumes 0.5 to 1 Megajoules less energy on average than the other methods for overall simulation time. For *SpanEdge*, the energy consumption increases as the number of tasks increases because the ICA applications sometimes require data from different nodes; as a result, many of the tasks $(25\%$ of the total tasks) are scheduled to the cloud. It causes an increased network load and causes higher transmission time, which ultimately increases energy consumption as we see in Figure 4. Other three methods have a smaller consumption of energy for the edge nodes in comparison to *SpanEdge* due to considering the energy consumption in making the offloading decision. *RILTA* performs better than both *SEA* and the optimal solution because of having lower energy consumed at the decision making phase as it can be seen from the lower part of each bar in the figure, which is due to reduced computation in decision making. Finally, we can also observe that the average energy consumed by the edge nodes increase as the number of nodes increases. This phenomenon occurs due to an increase in candidate nodes which means more energy is consumed in the task allocation phase as the number of computation rises. As a conclusion, *RILTA* shows better performance than the other methods in terms of the amount of average energy consumption for each edge node.

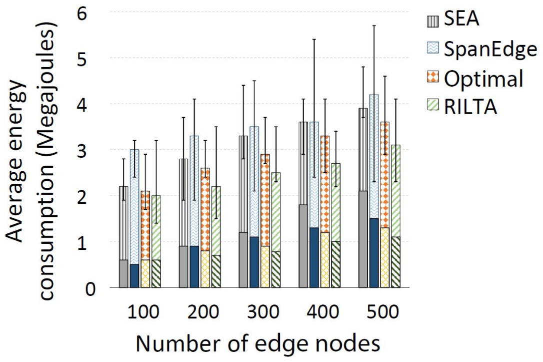

Fig. 4: Average energy consumption.

3) Task Running Time: Figure 5 shows the total task running time with respect to the number of tasks generated by each edge node. In the figure, we mark the amount of time spent on making the task assignment decision (lower part) and the actual task running time (upper part) in different colors and patterns. From the figure, we can see that the total task running time of all the methods are similar and only the time spent on decision makings for each method is different. To make the difference of decision making performance clear, we also drew Figure 6 to only show the time spent in task allocation. The figure shows that the order of performance in terms of task allocation time is: *RILTA* $<$ *Optimal* $<$ *SpanEdge* $<$ *SEA*. *RILTA* is comparatively $34-51\%$ faster than all other methods in terms of task allocation time. This is because *RILTA* only needs to decide by picking up the optimal action in the Q-value table. *SEA* spends more time on task allocation as they require multiple computation steps for finding the combinations of offloading decisions with different energy consumptions. Although *SpanEdge* makes the task allocation only depending on whether the task is local or global, it has a higher allocation time because the allocation of the global tasks is dependent on task dependency graph processing. However, *SEA* and *Optimal* generate longer assignment decision time but reduced total task running time because they ensure the time-bound response time of an incoming task. For both Figure 5 and 6, their corresponding Y-value changes with the increase of the number of tasks because with an increasing number of tasks generated from each edge device, both the task allocation and task running require more time.

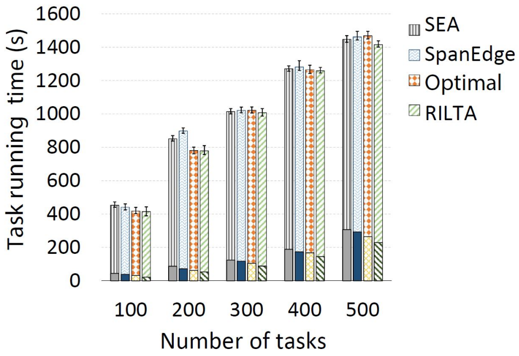

Fig. 5: Task running time.

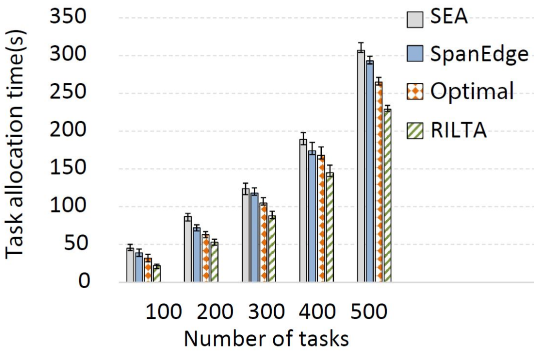

Fig. 6: Task allocation time.

VI. Conclusion

With the popularity of Intelligent Cognitive Assistants (ICA) using the edge computing systems, it is essential to solve the task assignment problem that determines the allocation of ICA application tasks to the three tiers in the edge computing system (i.e., remote cloud, fog and edge devices) considering task timeliness guarantee and energy constraints of the edge nodes. Despite many works on the task assignment, they fail to maintain timeliness guarantee due to complex computation phase of the task assignment process. In this paper, we present *RILTA*, a ReInforcement Learning based algorithm that can make the task assignment decision in significantly quick time Extensive evaluation results manifest that *RILTA* performs better than methods like the Spanedge and SEA algorithms in terms of timeliness guarantee and energy consumption alongside making the assignment decision in near real-time.

Acknowledgements

This research was supported in part by U.S. NSF grants NSF-1827674, CCF-1822965, OAC-1724845, ACI-1719397 and CNS-1733596, and Microsoft Research Faculty Fellowship 8300751. We would like to thank Mr. Liuwang Kang for his valuable discussions and comments.

References

[1] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coding for distributed fog computing," *CoRR*, vol. abs/1702.06082, 2017.

[2] K. Fogarty, "Semiconductor engineering, low power-high performance, processing moves to the edge," https://semiengineering.com/processing-moves-to-the-edge, 2018.

[3] A. Zone, "Is artificial intelligence a booster for edge computing?" https://dzone.com/articles/is-artificial-intelligence-a-booster-for-edge-comp, 2018.

[4] J. Oakley, "Intelligent cognitive assistants (ica) workshop summary and research needs - collaborative machines to enhance human capabilities," *Semiconductor Research Corporation (SRC)*, 2018.

[5] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proc. of EuroSys*, 2011.

[6] Y. H. Kao, B. Krishnamachari, M. R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Trans. on Mobile Computing*, 2017.

[7] O. Muoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Trans. on Vehicular Technology*, 2015.

[8] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. on Communications*, 2016.

[9] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. on Signal and Information Processing over Networks*, 2015.

[10] X. Lyu, H. Tian, C. Sengul, and P. Zhang, "Multiuser joint task offloading and resource optimization in proximate clouds," *IEEE Trans. on Vehicular Technology*, 2017.

[11] Z. Dong, Y. Liu, H. Zhou, X. Xiao, Y. Gu, L. Zhang, and C. Liu, "An energy-efficient offloading framework with predictable temporal correctness," in *Proc. of SEC*, 2017.

[12] F. Messaoudi, A. Ksentini, and P. Bertin, "On using edge computing for computation offloading in mobile network," in *Proc. of GLOBECOM*, 2017.

[13] J. Fan, X. Wei, T. Wang, T. Lan, and S. Subramaniam, "Deadline-aware task scheduling in a tiered iot infrastructure," in *Proc. of GLOBECOM*, 2017.

[14] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *SIGMETRICS Perform. Eval. Rev.*, 2013.

[15] H. Wang, J. Gong, Y. Zhuang, H. Shen, and J. Lach, "Healthedge: Task scheduling for edge computing with health emergency and human behavior consideration in smart homes," in *Proc. of IEEE Big Data*, 2017.

[16] T. Choudhari, M. Moh, and T.-S. Moh, "Prioritized task scheduling in fog computing," in *Proc. of the ACMSE Conference*, 2018.

[17] W. Zhang, Y. Wen, and D. O. Wu, "Collaborative task execution in mobile cloud computing under a stochastic wireless channel," *IEEE Trans. on Wireless Communications*, 2015.

[18] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? the bandwidth and energy costs of mobile cloud computing," in *Proc. of INFOCOM*, 2013.

[19] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, 2013.

[20] D. R. Morales and H. E. Romeijn, *The Generalized Assignment Problem and Extensions*, 2005.

[21] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the suitability of fog computing in the context of internet of things," *IEEE Trans. on Cloud Computing*, 2018.

[22] R. Ramaswamy, N. Weng, and T. Wolf, "Characterizing network processing delay," in *Proc. of GLOBECOM*, 2004.

[23] A. Venčkauskas, N. Morkevicius, K. Bagdonas, R. Damaševičius, and R. Maskeliūnas, "A lightweight protocol for secure video streaming," *Sensors*, 2018.

[24] P. G. V. Naranjo, Z. Pooranian, S. Shamshirband, J. H. Abawajy, and M. Conti, "Fog over virtualized iot: New opportunity for context-aware networked applications and a case study," *Applied Sciences*, vol. 7, 2017.

[25] R. S. Sutton, A. G. Barto *et al.*, *Reinforcement learning: An introduction.* MIT press, 1998.

[26] H. Gupta, A. VahidDastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, 2017.

[27] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov, "Spanedge: Towards unifying stream processing over central and near-the-edge data centers," in *Proc. of SEC*, 2016.