Optimising Computational Offloading and Resource Management in Online and Stochastic Fog Computing Systems

A thesis submitted in partial fulfilment of the requirement for the degree of Doctor of Philosophy

Faten Shaiban Alenizi

March 2023

Cardiff University School of Computer Science & Informatics

Abstract

Fog computing is a potential solution to overcome the shortcomings of cloud-based processing of IoT tasks. These drawbacks can include high latency, location awareness, and security attributed to the distance between IoT devices and cloud-hosted servers. Although fog computing has evolved as a solution to address these challenges, it is known for having limited resources that need to be effectively utilised. This is because its advantages could be lost. Moreover, the increasing number of IoT devices and the amount of data they generate make optimising Quality of Service (QoS) in IoT applications, computational offloading, and managing fog resources more challenging. In this context, the problem of computational offloading and resource management is investigated in online and stochastic fog systems. To deal with dynamic online fog systems, we propose a combination of two algorithms: dynamic task scheduling (DTS) and dynamic energy control (DEC). These methods were applied with a fixed offloading threshold (i.e., the criteria by which a fog node decides whether tasks should be offloaded to a neighbour, and which neighbour, rather than executed locally) with the aim to minimise overall delay, improve the throughput of user tasks, and minimise energy consumption at the fog layer while maximising the use of resource-constrained fog nodes. The approach is further enhanced by applying a dynamic offloading threshold. Compared to other benchmarks, our approach could reduce latency by up to 95.4%, improve throughput by 41%, and reduce energy consumption by up to 55.7% in fog nodes. For stochastic fog systems, we address the computational offloading and resource management problem. This is with the aim to minimise the average energy consumption of fog nodes while meeting QoS requirements of tasks. We formulated the problem as a stochastic problem and decomposed it into two subproblems. In order to solve this problem, we have proposed a scheme called Joint Q-learning and Lyapunov Optimization (JQLLO). Using simulation results, we demonstrate that JQLLO outperforms a set of baselines.

Publications associated with this research.

- Alenizi, F., & Rana, O. (2020). Minimising delay and energy in online dynamic fog systems. *arXiv preprint arXiv:2012.12745*.
- Alenizi, F., & Rana, O. (2021). Dynamically controlling offloading thresholds in fog systems. *Sensors*, *21*(7), 2512.

Table of contents

List of figures

List of tables

Acknowledgements

I am grateful for the opportunity to pursue doctorate studies and complete my dissertation. My gratitude goes out to many individuals who have helped me along the way.

Throughout my PhD studies, I would like to thank my advisor, Prof. Omer Rana, for his advice, knowledge, and continuous support. His insightful comments and support were significant in defining my study and helping me to the successful completion of my thesis.

During my stay at Cardiff University, my colleagues and friends Dr. Dalia Alfarasani and Dr. Asma Alshuhail provided me with support, encouragement, and companionship. Their camaraderie and generosity have enriched the significance and enjoyment of our voyage.

All members of the School of Computer Science and Informatics are to be thanked for their helpful discussions, comments, feedback, events, and facilities.

Lastly, I would like to express my gratitude to my family for their love, support, and encouragement during my academic path. Their unshakable confidence in me has been an inexhaustible source of motivation and inspiration for me. Thank you for your assistance, direction, and encouragement. Without your help, this thesis would not have been feasible.

1 Introduction

1.1 Research Context

The Internet of Things (IoT) has recently grown to play a significant role in our day-to-day lives, from the wearables we use to keep track of our health to the gadgets we use to control our lights and thermostats. Furthermore, it enables real-time data collection and analysis of various urban systems, including transportation, water management, energy, and public safety, to develop smart cities. As the number of connected IoT devices continues to expand, it is predicted that by 2025 there will be one trillion connected devices [1]. Due to the growing number of connected devices worldwide, large amounts of data are being generated.

In this context, cloud computing and IoT applications are faced with significant challenges. Due to data transmission time, processing IoT tasks within the cloud can result in substantial latency and performance complications. Moreover, the centralized architecture of cloud computing can cause network congestion, further aggravating these issues. Thus, cloud computing may not be an appropriate solution for IoT tasks involving real-time analysis, data processing, and decision-making.

An emerging paradigm called fog computing is designed to address the challenges of processing IoT tasks in the cloud [2], the architecture involving the use of fog computing systems is in Figure 1.1. Fog computing is an intermediate layer that exists between the cloud and IoT devices, providing location awareness, low latency, and a wide geographical distribution for IoT devices. As a result of processing data within the fog infrastructure, processing times will be shortened, network congestion will be reduced, and reliability will be increased.

Further, fog computing allows for the efficient handling of large amounts of data, improved security, and the ability to perform real-time data analysis, processing, and decision-making. It consists of limited-resource devices called fog devices/nodes, providing storage, processing, and networking resources close to IoT devices where tasks are produced. One of the key challenges that should be addressed effectively when running IoT applications in a fog computing environment is computational offloading and resource management.

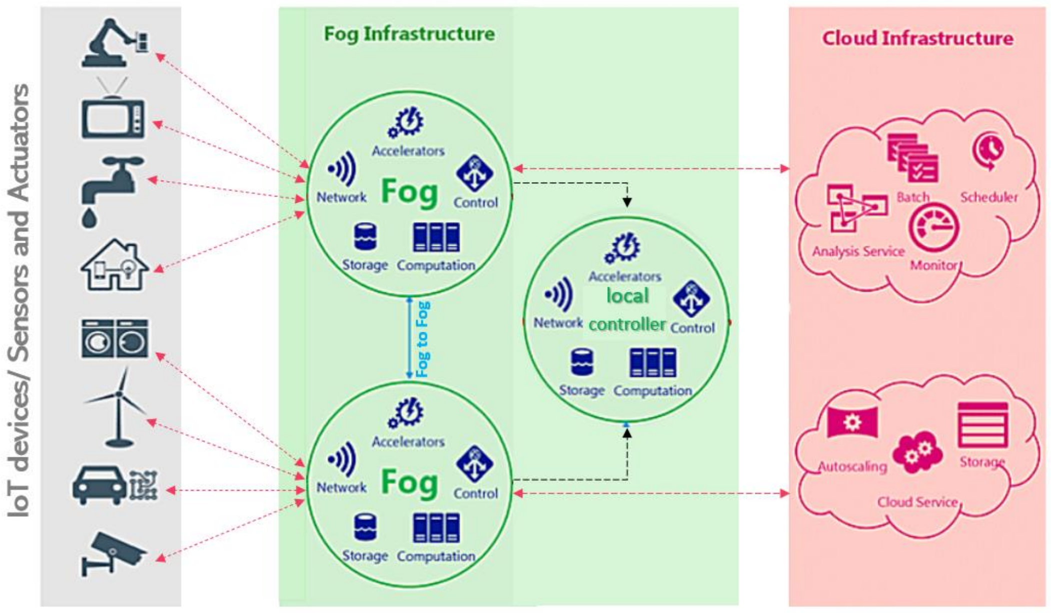


Figure 1.1: IoT-Fog-Cloud architecture.

1.1.1 Computational Offloading in Fog Computing

Computational offloading is an approach of resource management [3], it enables workload/computational tasks to be shared between IoT devices, fog nodes, and cloud servers. It is the process of transmitting tasks from resources with limited capacities to other resources with rich capacities to process these tasks within their Quality of Service (QoS) requirements [3]. When computational offloading occurs between fog nodes, this is called “fog cooperation” [4], in which overloaded fog nodes send part of their workload to other underloaded fog nodes. This is done to meet the QoS requirements of tasks such as processing tasks within their deadline and ensuring security.

Furthermore, the available fog system's computational resources can be exploited. In the process of computation offloading, primary fog nodes will decide where to process their received workloads. This decision will be made considering the status of the system and the requirements of these workloads.

Computational offloading can be accomplished by offline, online, and stochastic forms [5].

1. The offline setting addresses computational offloading before deploying the system in static environment. This is done by knowing the system's workload, network conditions and tasks requirements in advance [6], so that the decision on where to process tasks can be made before simulation begins [5].

2. In online settings, computational offloading is addressed online. This is accomplished while the system is running, and tasks are arriving in a dynamic environment. Online computational offloading is well suited to mimic real-world dynamics. The decision maker in this manner has no prior knowledge of the system. Additionally, the decision of where to process tasks is made considering the current requirements of the tasks and the current status of the system such as available computational resources, energy, and bandwidth [5, 6].

3. In stochastic settings, the computational offloading problem is analysed when uncertainty is involved. The characteristics of uncertainty can be influenced by workload, network conditions, or the status of entities. The process of stochastic offloading involves making decisions based on a probabilistic model of the state of the system.

1.1.2 Resource Management in Fog Computing

In fog computing, the resources are heterogeneous, dynamic, and constrained in that they are limited in terms of computation, power, and memory [7]. In light of this, resource management is one of the most challenging aspects of fog computing due to the complexity involved [7]. The effectiveness of fog computing requires efficient resource management. Effective resource management means maximizing capacity, availability, utilization, and cost of resources in the fog system. By considering these, we can ensure the optimal utilisation of the available computational, storage, and communication capabilities of fog devices. The concept of resource management can be addressed in several approaches, including task offloading, resource scheduling, resource allocation, and resource provisioning [3].

The concept of resource scheduling can be defined as a method or plan of calculating the number of resources required for IoT tasks, and the time when those resources will be needed [3]. The concept of resource allocation refers to allocating of available fog devices to perform IoT tasks while considering the requirements of a high quality of service, and other factors such as cost and energy [3]. Allocation of resources and resource scheduling are closely related and are frequently used together.

Scaling up or down available fog resources to maximise their energy effectiveness, cost, and response time is a process known as resource provisioning [3]. In resource provisioning approach, one of the main crucial issue that directly influences the lifespan of the resources, total expenses, and ultimately the effectiveness of the system and the resources is energy consumption [8]. Considering the importance of energy consumption, researchers have proposed different approaches to improve it; however, more research is needed in this area and it is still in its infancy [8].

In resource management, energy consumption poses a significant challenge since resource-constrained fog devices typically have limited energy resources [7]. The efficient management of resources can contribute to the reduction of energy consumption, the optimization of system performance, and the fulfilment of the requirements of QoS for IoT applications. Integrating computational offloading and resource management is essential to effectively utilise fog resources.

1.2 Research Gap

Most of the studies that addressed computational offloading problem to optimise system performance have focused on offline offloading [9-14] while online and stochastic offloading receives less attention. It would be beneficial to investigate and address these approaches further. This is because tasks that are generated by IoT applications in real-world scenarios are dynamic and unpredictable.

Furthermore, previous studies that investigated the problem of online and stochastic computational offloading considered different aspects of resource management in conjunction. These aspects are mostly related to resource scheduling and allocation [15-35]. However, resource provisioning in regard to energy saving at the fog infrastructures has not been considered. Saving energy in computing systems will lead to green computing, which helps to environmentally utilising computing resources in a user-friendly manner while ensuring overall system performance [36]. Ensuring green computing is a challenge for computing systems [36].

1.3 Research Aims

The work presented in this thesis aims to achieve the following.

- This study aims to explore the problem of computational offloading and resource management in online dynamic fog systems to minimise average delay and overall system's energy consumption.
- This study aims at investigating computational offloading problem and resource management with the purpose of minimising energy consumption under systems constraints in stochastic fog systems.

1.4 Research Objectives

- Identifying and listing the most important principles, hypotheses, and conclusions of existing research in the field.
- Developing a scheme that ensures minimising delay and energy in online dynamic fog systems with static offloading threshold.
- Addressing the impact of cooperation between fog nodes by increasing the number of neighbours on system performance in online dynamic fog computing systems.
- Developing and exploring the impact of dynamic offloading threshold in the joint problem of computational offloading and resource management on system performance in terms of throughput, average delay, and energy consumption in online fog systems.
- Formulating the minimisation energy consumption problem in stochastic fog systems as an optimisation problem to be solved as two subproblems and find a solution for each sub-problem.
- Designing and developing a learning approach that helps the learning agents/local controllers to learn about their environment faster and efficiently in order to manage energy resources in stochastic fog systems.
- Addressing the impact of various learning parameters in the efficiency of the learning process.
- Evaluating the proposed approach of minimising the average energy consumption in stochastic fog systems with other baselines and parameters to assess the feasibility and efficiency of the proposed approach. Thesis Outline

1.5 Research Questions

This thesis addresses the following research questions.

Q1. What is the significance of addressing the issue of threshold modification in online computational offloading and resource management in dynamic fog computing systems? How does this strategy compare to existing methods, and what implications does it have on delay, energy, and throughput?

Q2. What role does addressing stochastic computational offloading along with resource provisioning have in reducing the time-average energy consumption of fog devices in stochastic fog systems? How to formulate the problem considering both aspects?

Q3. How can cooperation between learning agents within stochastic fog systems be used to improve performance and scalability? How will this contribute to designing efficient dynamic resource management approaches within fog environments?

1.6 Research Questions: Narrative Logic and Connections

In this context, we provide a scenario of smart city applications like Public Transport and Smart Traffic Management that can be affected by highly congested events like a football match to highlight the logic and relationship between the stated research questions. Additionally, we demonstrated how addressing these research questions is important in optimising smart city operations.

Public Transport system is a vital application in the smart city, offering efficient and reliable transportation for its residents. It depends on a set of deployed sensor and data sources in order to ensure an efficient arrivals and departures. However, during peak times and busy periods such as football matches, relying on a static computational offloading and resource management scheme could fail to meet the demand or lead to suboptimal results.

Regarding the first research question, addressing threshold modification can help the smart city to dynamically adjusts computational offloading and resource management according to the anticipated traffic conditions. During highly congested events, fog devices, where this data is processed, can adapt their thresholds to distribute some of the data to neighbouring fog devices. This adjustment serves to reduce response times, maximise data processing, reduces energy consumption and ensures efficient Public Transport operations.

In terms of Smart Traffic Management, it depends on a set of sensors, real-time data sources, traffic cameras in order to optimise traffic flow. During unpredictable events, the system is challenged to perform effectively as the traffic patterns fluctuate significantly. The second research question plays a significant role in overcoming this problem. In order to control unforeseeable traffic flow at times of congestion, it is necessary to handle stochastic computational offloading and resource provisioning. During this congestion period, Smart Traffic Management can implement stochastic computational offloading scheme along with resource provisioning with the help of previous analysed data to respond in real time. This will lead to minimising traffic jam by smoothing traffic flow and reducing energy consumption.

In dynamic and complex environment, where congestion can occur, several smart city applications dealing with real-time data operate independently such as Public Transport system and Smart Traffic Management. Since these applications are affected by the same events, such as a football match, analysing their data separately can lead to delays in response, negatively impacting system performance. However, when these applications begin cooperating by sharing analysed information, this dilemma can be overcome, as cooperation will help to expedite response times. For example, the cooperation of Smart Traffic Management and Public Transport systems can help to reduce traffic and maximises routes around the football stadium. This will improve overall efficiency, scalability, and performance even when managing the challenges posed by large events. The third stated research question introduces a critical element to this scenario.

1.7 Research Hypotheses

In this thesis, the following hypothesis is investigated:

In online and stochastic computational offloading and resource management, varying an offloading threshold upon which fog devices start sharing their workloads with their neighbours will have a significant impact on optimising average delay, throughput, and energy consumption. Additionally, the implementation of dynamic offloading threshold, which adjusts in response to the availability of neighbouring fog devices, will effectively optimize the system performance.

1.8 Research Contributions

Following is a summary of the contributions of this thesis.

Presenting a review of state-of-the-art literature on computational offloading and resource management that aims to minimise delay or/and energy consumption to define the gap. This is achieved in chapter 2.

Designing and evaluating a model of a joint algorithm that aims at minimizing delay and optimising energy consumption in online fog systems. This joint algorithm is composed of two algorithms named Dynamic Task Scheduling (DTS) and Dynamic Energy Control (DEC). The method is examined in a simulated environment to evaluate its performance in a variety of settings. In addition, examine the performance of the joint algorithm when the offloading threshold is varied. Furthermore, investigate how the performance of the joint algorithm changes when the number of neighbours increases. This is accomplished in accordance with Question 1 and achieved in chapter 3.

Developing a dynamic offloading threshold that considers the workload status of the primary fog nodes and their neighbours in online dynamic fog systems. In addition, a simulation environment is utilised to compare the suggested solution against a set of baselines. The suggested threshold will be evaluated under various scenarios to establish its influence on average delay, throughput, and energy consumption. This is related to Question 1 and accomplished in chapter 3.

Establishing a novel stochastic optimization model that reduces energy consumption in stochastic fog systems. The optimization approach takes into account several stochastic factors, such as resource availability and network load, in order to produce an ideal solution. In addition, a decomposition strategy to solving the stated stochastic optimization issue is proposed. This strategy breaks the problem into two subproblems that are easier to tackle separately, and then combines the results to obtain the best solution to the original problem. The novel model presents a more effective and economical approach to tackling the energy consumption of fog computing systems. This is related to Question 2 and is achieved throughout chapter 4 and part of chapter 5.

Designing a dynamic resource management scheme based on Q-learning, and this is accomplished by two proposals. First, a cooperative Q-learning (CQL) method is used to develop a model that facilitates knowledge and experience sharing between agents. Second, proposing a scheme called Eliminating Unacceptable Actions (EUAs) to help speed up the learning process of agents in such stochastic fog systems. Moreover, the proposed approach (CQL-EUAs) is evaluated by conducting a series of experiments designed to measure the impact of potential parameters on the efficiency of the proposed scheme. This is related to Question 3 and is achieved in chapter 5.

Development of a novel optimization-based solution based on Lyapunov: this is accomplished by presenting a novel solution to the problem of computational offloading and processing decisions in stochastic fog systems. Using Lyapunov optimization, the proposed technique is intended to optimise energy consumption while meeting QoS requirements and system limitations. Using drift-plus-penalty theory, this approach decomposes the issue into three subproblems, which are addressed individually to achieve optimal solutions and reduce the complexity of the original problem. This is related to Question 2 and accomplished in chapter 5.

1.9 Thesis Structure

The structure of this thesis is as follows:

Chapter 2: Literature Review. This section provides an overview of the state-of-the-art. It is divided into two subsections. The first sub-section is on computational offloading problems and dynamic resource management in the area of fog computing. The second subsection is about investigating the joint problem of computational offloading and resource management in stochastic fog/edge computing systems. A comprehensive gap analysis concludes each subsection, describing how this research addresses the gap in the literature.

Chapter 3: Minimising delay and energy in online dynamic fog systems. This chapter proposes a joint dynamic approach with the aim to minimise average delay and energy consumption using fixed offloading threshold. It also shows the impact of increasing the number of neighbouring fog devices in overall system performance. Moreover, it proposes the use of dynamic offloading threshold upon which the fog node is determined congested and starts sharing its workload with its neighbours. The efficiency of the proposed scheme is evaluated with various baselines.

Chapter 4: Optimising the Energy Consumption in Stochastic Fog Computing Systems. The purpose of this chapter is to provide an overview of the system model and constraints, along with formulating the minimisation problem of energy consumption in stochastic fog system into two subproblems. In addition, it provides a brief description of the proposed solutions.

Chapter 5: Dynamic Resource Management based on Q-Learning Algorithm. In this chapter the minimisation problem of energy consumption in regard to dynamic resource management is formulated as a constrained optimisation problem, where the learning agent/local controllers aim to maximise, its average received rewards in the system. The proposed solution is called CQL-EUAs is based on combining two approaches named Cooperative Q-learning (CQL) and Eliminating Unacceptable Actions (EUA). This is used to solve the firs sub-problem. A set of experiments is conducted with different parameters and approaches. Additionally, this chapter aims at solving the second-sub problem regard finding the optimal computational offloading and processing decision in stochastic fog systems. The proposed approach is based on the framework of Lyapunov Optimisation and drift-plus-penalty theory. The results of the main optimisation problem are presented along with a set of benchmarks.

Chapter 6: Conclusion. This chapter Provides a summary and conclusion of the results and contributions of the thesis. A discussion of potential future work is also included.

2 Literature review and Background

2.1 Introduction

In this section, a literature review of subjects relevant to the dissertation is presented. The literature review chapter is organized into three sections. The first section focuses on online/offline computational offloading and dynamic servers' energy management in fog computing and related fields. The section discusses various research studies and findings related to these topics. It emphasizes their importance in improving the overall efficiency of optimising delay and energy consumption in fog computing and related fields. The second part of the literature review is dedicated to addressing the problem of computational offloading and resource management in stochastic fog computing systems and similar fields. The section provides an in-depth analysis of research studies and approaches that have been developed to optimise energy consumption, delay, and other factors in stochastic fog/edge computing systems. Finally, a background on Reinforcement Learning (RL), Q-learning, and cooperative Q-learning is provided. In this section, the importance of these approaches is discussed along with related works on the cooperative Q-learning approach. Overall, this section offers a comprehensive review of the literature related to the dissertation's research topics.

2.2 Online/offline computational offloading and dynamic servers’ energy management

This section is organised into three main parts. The first part focuses on computational offloading between entities within a specific system. A summary of relevant research studies is presented in Table 2.1. The second part addresses the impact of dynamically managing servers to enhance power efficiency in computing systems. This part provides an in-depth analysis of research studies and findings related to this topic, highlighting their importance in minimising energy consumption. Finally, the section provides a comparison of the state-of-the-art approaches in fog computing and related fields. This comparison highlights the strengths and weaknesses of different approaches and provides insights into the future direction of research in this area.

2.2.1 Computational Offloading

In this section, we focused on studies that addressed the offline and online computational offloading problem. In the literature, online and offline offloading sometimes are referred to as dynamic and static offloading [37, 38]. The main differences between online and offline offloading problem in distributed systems are summarised below. In regard to the timing of the offloading decision and system knowledge, in offline offloading, decision is made before the execution of the application and we have complete system information, such as the arrival of tasks and the availability of computing devices. In online offloading, decisions are made in real-time during the execution of the application, and the system doesn't have complete knowledge of the task sequence or future states while making offloading decision. Regarding adaptability, in situations when the environment is stable or where events in the future can be predicated, offline offloading is more appropriate. In real-world applications with dynamic conditions, offline offloading may not be as adaptable to changes in the system state compared to online offloading. Online offloading is more adaptive to changing condition, including varying workloads, network latency, and resource availability. When classifying the work in the literature as to address online or offline offloading problem, few works have specified the type of offloading problem they have considered [39], while the majority did not specify that. To define the type of the addressed offloading problem in the literature, we analyse their model assumptions, the process of the offloading decisions, and the use cases. In more details, the problem is considered as an offline offloading problem if the following aspects is considered. The model assumes a stable or predictable environment with complete knowledge of variables like task arrival, resource availability, and network conditions. Moreover, if the described scenario or the use case is static or dealing with batch processing.

Furthermore, If the offloading decision is made collectively for a batch of tasks all at the same time and distribute them to available computing devices without considering the impact of these tasks on the availability of the computing devices, then the problem is defined as offline offloading problem. On the other hand, the offloading problem is considered an online problem if the following is satisfied. The used model deals with dynamic or unpredictable environment with incomplete system information. Additionally, if the used scenario or the use case is dynamic. Also, the offloading decision is made for each task upon its arrival considering the current system state. In offline implementation, all system information needed to make the offloading decision is previously known and is based on historical or predictive knowledge, such as the computational capabilities of fog nodes, the total number of IoT devices and their total workload (number of requests). This is applied during the system design stage. In online deployment, the computational offloading decision takes place at run-time and takes into account the current system status and process characteristics, such as the current waiting time and the current available computational resources, without prior knowledge of system inputs. **A number of studies have explored computational offloading in offline deployment [9-11, 13, 14, 40-44]**.

In IoT-Fog-Cloud architecture, Sun et al. [9] have investigated computational offloading and resource allocation in order to minimise the cost which represents the trade-off between the completion time for IoT tasks and the energy consumption for handling these tasks on cloud servers, fog servers, and IoT devices. the authors presented the "ETCORA" algorithm that consists of two parts. The first part aims to find the optimal offloading decision based on minimising delay and energy, and the second part optimises resource allocation in terms of transmission power allocation. Their proposed solution helps to minimise energy consumption and completion time of tasks compared to other schemes.

In [10], Wang et al. investigated the optimised offloading problem to minimise task completion time given tolerable delay and energy constraints. The optimisation problem has been formulated as a mixed integer nonlinear programming problem that jointly optimises the local computation capability for IoT devices, the computing resource allocation of fog nodes and the offloading decision. Wang et al. [10] decompose the problem into two independent sub problems to find the optimal amount of workload that should be processed locally at IoT devices and at fog nodes. A hybrid genetic-simulated annealing algorithm has been developed for this purpose. In terms of completion time and energy usage, their results demonstrate that their proposed solution achieves considerable benefits over a set of baselines.

Liu et al. [11] conducted research on a fog environment to address the multi-objective optimisation offloading problem, which aims to minimise execution delay, energy consumed at mobile devices, and offloading payment cost for using fog/cloud resources. In order to accomplish the specified goals, the authors attempted to determine the optimal offloading probability and transmission energy for each mobile device. The multi-objective problem is formulated into a single problem using the scalarization method, and proposed a solution based on the Interior Point Method (IPM) to solve it [11]. Based on simulation results, their suggested approach optimises delay, energy, and payment costs more effectively than existing schemes.

Xiao and Krunz [13] investigated the topic of task offloading in fog networks. Under a given power constraint, fog nodes in their system select whether to offload a portion of their workloads to nearby nodes or to process them locally. When considering fog offloading, the authors examine the trade-off between service quality and energy efficiency. The problem is defined as a non-convex optimization problem and split into smaller problems that each fog node may tackle. A distributed optimization technique has been presented to handle the optimum workload allocation issue in order to maximise QoS in terms of response time under energy limitations. Their findings demonstrate that their suggested method substantially enhances the performance of fog computing systems. The authors proposed that the response time of end-user tasks should be set to its highest tolerable point to optimise energy consumption at fog computing systems. This enables most of the tasks to be processed at end-user devices, avoiding any offloading.

Chen and Hao [14] studied offloading problem in dense software-defined networks, formulating this as a mixed-integer nonlinear problem that is decomposed into: (i) deciding whether the task is processed locally at the end-user device or offloaded to the edge device; (ii) determining the computational resources that are dedicated to each task. The authors developed an efficient software-defined task offloading scheme to solve these sub-problems. The results of their proposed scheme demonstrate the superiority of their approach at decreasing end user device energy consumption and overall task execution latency.

Mukherjee et al. [40] developed an offloading technique with a focus on jointly optimising the computing and communication resources at fog systems to reduce end-to-end latency. Their technique considers the trade-off between transmission delay and task execution delay when making the offloading decision. A fog node can obtain additional computational resources from either one of its neighbours or the cloud data centre to reduce task execution delay at the expense of the transmission delay. The authors transformed the optimisation problem into convex Quadratically Constraint Quadratic Programming and solved it using the CVX toolbar, a MATLAB-based modelling system for convex optimization. Their simulation results showed that their proposed solution minimises end-to-end latency compared to executing all tasks at end-user devices or executing all tasks at the primary fog nodes.

Zhu et al. [41] presented a task offloading strategy based on execution time and energy consumption. This method allows mobile devices to make an appropriate decision on whether to process their tasks locally or offload them to a fog node or the cloud. During the decision-making process, mobile devices calculate the execution time and energy consumption required for executing the task locally and compare it with the execution time and energy consumption required for offloading and receiving the processed task on a fog node, without considering the energy consumed by the fog nodes. The mobile device selects the option with the lowest cost, which is the sum of execution time and energy consumption. The simulation results demonstrate that their approach is better than Random, no offloading, and offloading based solely on execution time, as it optimizes the execution time and energy consumption of mobile devices.

Mukherjee et al. [42] formulated the offloading problem as an optimization problem with the goal of minimizing the total system cost, which is the sum of the total delay of end-users' tasks and the total energy consumed by end-users' devices due to local task processing and uploading tasks to the fog environment for processing. Their optimization problem considers both delay and energy constraints, and it has been converted into a Quadratically Constraint Quadratic Programming problem that has been solved using the semidefinite relaxation method. In a heterogeneous environment where fog nodes have varying computational resources, the proposed solution identifies the optimal amount of workload to be processed at end-user devices, primary and neighbouring fog nodes, and cloud servers. The offloading decision is based on the availability of computational resources. The authors noted that having higher computational resources at fog nodes can reduce the system cost. Additionally, an increase in the number of end-users leads to greater congestion at fog nodes, resulting in fog nodes preferring to send their workload to the cloud server for processing rather than to neighbouring fog nodes.

In radio access networks, Zhao et al. [43] examined the computational offloading problem to decrease the weighted sum of total offloading latency plus total energy consumption. To enhance the allocation of computation and radio resources, the authors formulated the problem as a non-linear, non-convex joint optimization problem, and their solution is more effective than mobile cloud computing (MCC) and mobile edge computing (MEC) because it employs a combination of available resources at the cloud and fog nodes, rather than just cloud resources as in MCC or edge computing resources as in MEC.

Hybrid-Computational offloading optimization problem has been investigated by Meng et al. [44]. Their research considered two types of models: cloud computational offloading and fog computational offloading. The authors aimed to reduce the energy consumption caused by processing and transmitting tasks on mobile terminals, fog, and cloud servers while adhering to deadline constraints. They introduced a new concept called computation energy efficiency, which is defined as the number of computation tasks offloaded by consuming a unit of energy. The proposed solution involves offloading tasks to both fog and cloud servers for execution, and simulation results demonstrate its effectiveness compared to offloading tasks to either cloud or fog resources alone.

The online deployment of computational offloading has not been extensively studied, as few studies have addressed this topic, including [4, 12, 39, 45, 46]. Al-Khafajiy et al. [4] have proposed an offloading mechanism that facilitates fog-to-fog collaboration in heterogeneous fog systems to minimize overall service latency. Their mechanism employs the FRAMES load balancing scheme to detect congested fog devices, determine the workload located at fog devices' queues that require offloading based on their deadline requirement, and select the best fog node with minimal service latency for the workload. They evaluated their mechanism using simulation, which showed that their proposed model is effective in minimizing overall latency compared to different algorithms.

In Fog-Cloud computing system, Gao et al. [12] investigate the issue of dynamic computational offloading and resource allocation. In order to reduce energy consumption and delay while having a stable queueing status, the authors formulate the problem as a stochastic network optimisation problem. They provide a predictive approach to computational offloading and resource allocation that depends on the trade-off between delay and energy use. Their approach implies that a delay reduction can be induced by increasing the allocation of computational resources at fog nodes, however, because of the processing of more tasks, energy consumption increases, and vice versa. Compared to other systems, the authors show the importance of their method.

Yousefpour et al. [45] proposed a delay-minimization approach to reduce overall service delay. Their approach utilizes the estimated queueing delay as the offloading threshold, which determines whether a fog node processes its incoming task(s) or offloads them to one of its neighbours or the cloud server. When the offloading threshold is reached, the best neighbouring fog node in its domain is selected to offload its upcoming tasks. The best neighbouring fog node is selected based on having the minimum total propagation delay and queuing delay. Their results showed the minimum average service delay compared to other models.

Yin et al. [39] determine where to process end user tasks into task scheduling and resource allocation problems, where tasks are either processed locally at end-user devices or offloaded to fog nodes or cloud servers. In an intelligent manufacturing environment, the authors introduced fog computing and utilised the concept of the container within the fog system intending to reduce overall delay and optimise the number of concurrent tasks for the fog node. In their online model, generated tasks by end-users are transmitted to the Request Evaluator which is located at a fog node that decides whether to accept or reject the task based on its deadline requirement. If the task is accepted, then the task is transmitted to Task Scheduler which determines whether the task is processed at fog nodes or cloud servers based on the available resources and the execution time of this task which involves computation and transmission time. Finally, the Resource Manager responsible for reallocating the required resources to process the task at fog nodes. Experimental results show the effectiveness of their approach compared to other benchmarks.

Mukherjee et al. [46] developed a scheduling strategy that manages to fulfil the deadline constraint of end-user tasks, taking into account computational resources. The deadline constraint of a given task and the availability of a neighbour, in their scheduling policy, help to decide on whether to place a given task in the fog node queue, e.g., in its high-priority queue or low-priority queue, or offload it to one of its neighbouring fog nodes. Their findings illustrate the efficacy of their suggested strategy as opposed to the no offloading and random schemes.

2.2.2 Dynamic Server Energy Management

Dynamic Server Energy Management has been implemented in the wireless local area network (WLAN) and the cloud, leading to improved power quality. However, this technique has not yet been widely used in the fog computing area. In WLANs, energy efficiency has been improved by placing access points (APs) in sleep mode or turning them off.

In [47], Marsan and Meo proposed a system in which a single AP in each community would remain active and service incoming clients, while all other APs would be turned off. This approach was shown to reduce energy consumption by up to 40%. Additionally, up to 60% of energy consumption could be saved by turning off all APs during idle periods, such as at night.

Li et al. [48] introduced an energy-saving method that involved an intermediary stage to reduce the frequency of AP state transformations, which can shorten an AP's service life. The authors also noted that the intermediary stage helps to prevent latency and energy overhead caused by frequently turning APs on and off.

It has been suggested that servers could be periodically switched off [49, 50] or placed into sleep mode [51-53] in cloud computing systems to conserve energy resources. In [49-53], the authors examined the issue of the placement of virtual machines (VMs) to save resources concerning energy and yet retain QoS. When underutilised data centres are detected, all VMs are migrated to other active data centres, and these underutilised data centres are placed in sleep mode according to [51-53] or shutdown as per [49, 50]. This is intended to reduce the consumption of energy at cloud computing systems and is called ‘VM consolidation’. Numerous VM migration approaches have been suggested to assess which virtual machines can be migrated from overloaded data centres. Moreover, in order to satisfy the QoS specifications of the system, a switched-off data centre may also be activated to handle the migrated VMs. According to Mahadevamangalam [51], the energy demand for an idle data centre is as 70% of the energy generated by a fully-utilized data centre. Thus, by switching off idle-mode data centres, up to 70% of the energy consumed can be saved in the cloud system.

2.2.3 Comparison of the State-Of-The-Art

Table 2.1 summarises the related research on computational offloading in fog computing systems, highlighting the architecture model such as IoT-Fog, which means that the local end-user tasks are either processed at IoT devices or offloaded to fog nodes. The table also mentions the use of fog cooperation, and the type of offloading threshold used to start offloading tasks from the primary fog nodes to their neighbours. Type of communication whether it is vertical or horizontal. The stated objectives of the work, e.g., delay, energy, throughput, and finally, the evaluation tools for both offline and online offloading decisions.

It can be seen that when addressing the computational offloading problem, most research focused on offline deployment such as [9-11, 13, 14, 40-44]. However, online deployment of computational offloading receives less attention as in [4, 12, 39, 45, 46]. It is essential to investigate online computational offloading problems as it mimics the dynamics of real-world applications.

In regard to the difference between the methodologies that applied in the literature to address offline and online offloading problems, we have noticed the following. In offline offloading problems, authors utilised optimisation techniques that have common characteristics. First, they require full knowledge of the systems' parameters in order to make the offloading decisions. Furthermore, the decision-making process is made for batch of tasks all at the same time given full system information. Moreover, the proposed algorithms are based on optimisation techniques that iterate until convergence to an optimal solution, this iteration is a time consuming. The features required to run these algorithms make them suit the nature of the offline offloading problem. In more details, in [10] authors proposed an approach that combines Genetic Algorithm (GA) and the Simulated Annealing (SA). Combining the two approaches can result in increased computational overhead as a result of combining the operations of both algorithms. Moreover, to ensure converges to an optimal or near-optimal solution, more time is required. These features do not suit the nature of online scenarios where quick decisions are required.

In [11], authors proposed a method based on Interior point method (IPM) to solve the optimisation problem. IPM is an iterative method and during each iteration, it solves a set of problems. The cost of computing this can be high, particularly for large-scale problems. Additionally, it starts with an initial feasible solution and iterates until convergence. A good starting point is not guaranteed, and it influences the final solution. Moreover, in real-time systems, there might not be enough time for the algorithm to iterate each time and finally converge to an optimal solution, especially if new data/tasks are arriving rapidly.

In [40, 42], the optimisation problem is transformed as Quadratically Constraint Quadratic Programming (QCQP) problem and solved using existing solvers such as CVX program. QCQP is known as computationally intensive. This is problematic for online offloading where real-time or near-real-time solutions are often desired. Moreover, transforming the problem into QCQP and then solving it, requires full knowledge about system’s parameters. In more details, the coefficients of the quadratic and linear terms, which is part of the problem's objective and constraints, must be defined in advance. Solving a QCQP may be more difficult if the system's parameters are constantly changing or can only be observed in real-time. In such cases, transforming a problem into QCQP and then solving it, is not practical in online environment. additionally, QCQP problems, particularly non-convex ones, can be NP-hard. It may not always be feasible to solve them optimally within the time constraints due to the computing demands.

Addition to that an optimisation problem in [43] is decoupled into four subproblems. Each of which is solved based on these approaches; closed-form solution, Interior point method, and dynamic programming. These proposed solutions are iterative and combining these together will increase the complexity of the proposed solution. Additionally, they require full knowledge of system parameters which in turn do not suite the nature of online offloading problems. The closed-form based solution is also adopted in [44]. Closed-form solutions are produced by deriving a thorough comprehension of the whole problem space as well as knowledge of all the coefficients, constraints, and variables in advance. Also, it is known as computationally intensive approach. Closed-form solutions are deterministic and might not adapt well to the dynamic nature of offloading scenarios where conditions can change rapidly.

Xiao and Krunz [13] proposed a solution based on distributed Alternating Direction Method of Multipliers (ADMM). The problem is divided into several sub-problems and solved accordingly. The proposed solution is based on finding the optimal solution that contains the amount of workload that can be processed locally at each fog node based on its local information. Then, the optimal solutions for all fog nodes are sent to the cloud to find the optimal coordination solution between fog nodes. ADMM in in its distributed form quite is complex and time-consuming. It requires knowledge about the global system in order to converge to an optimal solution. These features dop not suit the structure of online environment.

Regard online offloading problem, the proposed solutions in the literature that address this problem is based on Heuristic and Algorithmic methods. In online environments with dynamic and unpredictable nature, it's essential to have solution methods that can deliver timely and efficient results. Heuristic and algorithmic methods inherent the following characteristics that meet the challenges of online offloading problems. One of these futures are being fast in producing good-enough solutions considering the current information that is available in the system. Furthermore, they can adjust the offloading decision based on changing conditions such as amount of workload and availability of computing devices. They can operate effectively with partial or imperfect information. Being flexible and provide robust solution.

In more details, the authors in [4], Al-Khafajiy et al proposed a solution based on Algorithmic methods. The reason for classifying their proposed solution based on Algorithmic methods is because the proposed solution contains a set of procedures. Each procedure follows a systematic process, and the process of the decision making follows specific conditions such as checking the queue size, service arrival rate, and service rate. Furthermore, the behaviour of the proposed solution is deterministic given the current state.

In terms of applying Heuristic-based methods, the works conducted by [45, 46] have proposed solutions based on this approach. In more details, in [45] the proposed approach is said to be a heuristic-based method because of the following reasons. First, the primary fog node makes decisions based on the current state of its queue. Moreover, the decision process is relatively straightforward. Additionally, applying a threshold to decide whether to add a task in its queue or not and then selecting a neighbouring fog node with the least waiting time plus propagation delay are both heuristic strategies.

In [46], their proposed solution is said to be heuristic-based method because it appears to make an offloading decision based on constraints and thresholds. This behaviour is typical of heuristic approaches. Additionally, in their approach, instead of waiting for tasks to reach their completion time, it uses a predictive, rule-based strategy to determine resource requirements, and this is one of the characteristics of Heuristic methods.

Moreover, when optimising delay and energy consumption in the literature, previous studies have focused on minimizing energy usage at IoT devices while neglecting energy consumption at the fog level [14, 43, 54]. Others have explored the trade-off between delay and energy consumption within fog systems [9, 12, 13].

In the case of optimising the energy consumption of fog/edge servers along with delay, most studies aim to balance delay and energy consumption by processing tasks on IoT devices, fog nodes, or cloud servers, depending on the quality-of-service requirements. However, there may be underutilized or idle fog nodes that could be turned off to save energy. This would enable them to reap the benefits of the fog architecture, such as processing more tasks at fog nodes and thereby reducing delay. To the best of our knowledge none of the previous studies have addressed both delay and energy minimization simultaneously while implementing dynamic server energy management through the on/off switching of fog nodes in the fog system.

Our approach primarily focuses on the online approach, which has not been explored in detail in previous literature. Fog computing aims to deploy computational resources in close proximity to end-users to reduce delay. Users send their requests to the nearest fog node, but if it is overwhelmed, current methods offload some of the work to the cloud for processing. However, other underloaded fog nodes in the vicinity could also help to process the workload and further reduce delay. This concept is known as "fog cooperation," but it has received limited attention in the literature [12, 13, 45, 46]. In this context, we exploit the concept of cooperation between fog nodes in online dynamic system. This is to minimise the average delay in the system.

In addition to that and in order to conserve energy resources in a dynamic fog system we propose the use of switching on/off fog nodes, which has not been considered in the fog/edge computing systems.

Additionally, none of the existing fog computing models have studied the impact of varying the offloading threshold on system performance, highlighting the need for further research in this area. In addition, to the best of the authors' knowledge, no literature has addressed the dynamic offloading threshold that increases and decreases based on the status of the primary fog nodes and all their neighbours.

In summary, online computational offloading, and resource management in relation to energy conservation in dynamic fog systems have received limited attention so far. As a result, this sheds light on the importance of examining these aspects at the same time.

Table 2.1: Computational Offloading State-Of-Art Comparison.

Ref	**Offloading Deployment**	**Architecture Model**				**Fog Cooperation**	**Offloading Threshold**	**Com-muni-cation Type**		**Objectives**						**Evaluation Tool**
		IoT-Fog	**IoT-Fog-cloud**	**Fog-Cloud**	**Fog**			**Vertical**	**Horizontal**	**Delay**	**Energy**				**throughput**	
											Yes	**Relation with delay**	**Which Energy**	**Which device**		
Wang and Chen [10]	Offline	✓	-	-	-	X	static	✓	X	✓	✓	Energy constraint	Processing and transmitting	IoT devices and Fog nodes	X	Simulation
Liu et al [11]		-	✓	-	-	X	static	✓	X	✓	✓	Trade-off	Energy spent by local processing and transmitting tasks	mobile devices	X	Simulation
Mukherjee et al [40]		-	✓	-	-	✓	static	✓	✓	✓	X	X	X	X	X	Monte Carlo simulations
Zhu et al [41]		-	✓	-	-	X	static	✓	X	✓	✓	Offloading policy is designed to minimise task execution delay and to save energy	The energy spent by uploading and receiving tasks	Mobile devices	X	Simulation (MATLAB)
Mukherjee et al [42]		-	✓	-	-	✓	static	✓	✓	✓	✓	The goal of the study is to minimise total systems cost which includes total energy consumption plus total processing delay	Energy consumed by local computing and uploading tasks to fog nodes	End users' devices	X	MATLAB

Ref	Offloading Deployment	Architecture Model				Fog Cooperation	Offloading Threshold	Com-muni-cation Type		Objectives						Evaluation Tool
		IoT-Fog	IoT-Fog-cloud	Fog-Cloud	Fog			Vertical	Horizontal	Delay	Energy				throughput	
											Yes	Relation with delay	Which Energy	Which device		
Chen and Hao [14]		✓	-	-	-	X	static	✓	X	✓	✓	Battery capacity	Processing and transmitting	End users' devices	X	Simulation
Sun et al [9]		-	✓	-	-	X	static	✓	X	✓	✓	Selects the least overhead cost which involves total computational time plus total energy spent by either processing task at IoT devices or transmission tasks to fog or cloud	Processing and transmission power	IoT, fog nodes, cloud servers	X	iFogSim
Zhao et al [43]		-	✓	-	-	X	static	✓	X	✓	✓	Minimises the system cost which is the total offloading latency and the total energy consumption	transmission and processing energy	end users' devices, fogs, and cloud	X	Simulation
Meng et al [44]		-	✓	-	-	X	static	✓	X	✓	✓	Minimising energy given delay constraint	transmission + computational energy	Mobile Terminal, Fog servers and cloud servers	X	simulation
Xiao and Krunz [13]		-	-	✓	-	✓	static	✓	✓	✓	✓	Trade-off	Transmission energy	fog nodes	X	Simulation

		Architecture Model						**Com-muni-cation Type**		**Objectives**						
											Energy					
Ref	**Offloading Deployment**	**IoT-Fog**	**IoT-Fog-cloud**	**Fog-Cloud**	**Fog**	**Fog Cooperation**	**Offloading Threshold**	**Vertical**	**Horizontal**	**Delay**	**Yes**	**Relation with delay**	**Which Energy**	**Which device**	**throughput**	**Evaluation Tool**
Yousefpour et al [45]		-	✓	-	-	✓	static	✓	✓	✓	X	X	X	X	X	Simulation
Yin et al [39]		-	-	✓	-	X	static	✓	X	✓	X	X	X	X	✓	Simulation
Al-Khafajiy et al [4]		-	✓	-	-	✓	static	✓	✓	✓	X	X	X	X	X	MATLAB-based simulation
Gao et al [12]		-	-	✓	-	✓	static	✓	X	✓	✓	Trade-off	Processing and transmitting tasks between fog nodes	Fog nodes	X	Simulation
Mukherjee et al [46]	Online	-	-	-	✓	✓	static	X	✓	✓	X	X	X	X	X	simulation
Our proposed approach in chapter 3		-	-	✓	-	✓	static	✓	✓	✓	✓	X	POWERING ON and processing tasks	Fog nodes	✓	iFogSim
Our proposed approach in chapter 4		-	-	✓	-	✓	Dynamic	✓	✓	✓	✓	X	POWERING ON and processing tasks	Fog nodes	✓	iFogSim

2.3 Computational offloading and Resource management in stochastic fog computing system

Computing offloading and resource management are two of the main mechanisms of fog computing. This is because they help to overcome resource restrictions at fog devices. They also enhance scalability and flexibility, ensure efficient use of computing resources, and therefore enhance system performance and reliability. Having a stochastic environment where the elements of the system are random and not known in advance including the arrival of computational tasks, the state of the available bandwidth, and the available computational resources, makes it difficult to ensure the effectiveness of computational offloading and resource management in such an unstable environment [28].

In uncertain dynamic systems, several works addressed the computational offloading problem along with resource management to maximise the efficiency of the network resources and satisfy the QoS requirements in this regard, we conduct a comprehensive literature review that discusses the computational offloading problem and resource management in stochastic fog systems. Authors in this field mostly utilised one or both of the following two techniques to help solve such a problem: namely Lyapunov Optimisation technique and Reinforcement/Deep Reinforcement Learning. Based on the applied methodologies, relevant works are classified into the following. Works that applied Lyapunov Optimisation theory only, works that utilized Reinforcement and Deep reinforcement learning, and finally works that combined the two techniques. Following then, a state-of-the-art comparison of these works is provided.

A summary of all the stated related works is shown in table 2.2. Table 2.2 of the related works presents the main objectives, the type of resource management involved when addressing the computational offloading problem, the issue of cooperation among edge/fog servers to allocate the proper computational resources and share workload, where tasks are processed, the heterogeneity of tasks in terms of types and deadlines, whether priority-aware scheduling is considered, and finally the proposed approach used to solve the considered problem.

When addressing computational offloading and resource management in a stochastic fog system, several works have examined solutions based on Lyapunov Optimisation theory, including [15-24, 55]. In [15], the authors addressed the optimisation problem that aims at minimising the time average energy consumption caused by executing tasks on local mobile devices while imposing constraints regarding energy budget and over-exploiting the resources of other mobile devices that might affect users' collaborations. In a stochastic environment, where the current available connections between devices and computation resources are unpredictable due to users' mobility, the authors proposed an online task offloading algorithm based on Lyapunov theory that helps the network operator (e.g. base stations) to make the offloading decision for all users based on the current global network information. The proposed algorithm allows end users to share communication (bandwidth) and computation resources. As compared to local execution on mobile devices, around 40% of the energy is saved.

The authors of [16], proposed a green computing framework for IoT-Edge-Cloud queueing computing systems, aiming to minimize energy consumption at edge and cloud servers while meeting user deadlines. In a stochastic environment, where task arrival is unknown in advance, the authors discuss the optimal allocation of tasks between local edge nodes, neighbouring edge nodes and the cloud. Their developed algorithm is named delay-based workload allocation (DBWA) which is based on Lyapunov drift-plus-penalty theory. In a scenario of three IoT regions and three edge nodes and one cloud, their numerical results demonstrated the superiority of the proposed algorithm over benchmark schemes, which are cloud-only and edge-only systems, in terms of achieving minimum energy consumption and meeting deadline constraints. The scalability issue is such a concern in their work, as they tested their proposed solution with only having three edge nodes in the system. Furthermore, the selection process of the best neighbour was not clearly handled in their work.

In the IoT Fog computing system, Chang et al [17] studied dynamic allocation of computational and radio resources and computation offloading. The problem has been formulated as an optimisation problem. It considered both the allocation of resources and finding the optimal offloading strategy. This was aimed at minimising the system cost which consists of the total energy consumption of mobile devices and overall latency. The offloading decision involved either processing tasks locally on mobile devices or offloading them to fog nodes. To solve the optimisation problem, the authors decomposed the main problem into several subproblems that are addressed in each time step. They proposed a dynamic algorithm based on Lyapunov optimization to solve it. As a result of the results, the proposed scheme outperforms other approaches.

A study by Chen et al [18] examined peer offloading among MEC-enabled small-cell base stations (SBSs) in an edge environment. Based on energy budgets and deadline constraints, the optimisation problem is aimed at minimizing time average delays. An online algorithm based on Lyapunov theory is proposed. This algorithm does not require prediction of the future. Compared to other benchmarks that consider no peer offloading and only minimising delay, their results demonstrated that their developed algorithm shows high system performance without violating the energy budget constraint of SBSs.

In [19], the authors developed an online energy-efficient computation offloading framework based on Lyapunov optimisation theory to solve the stochastic optimisation problem. Under CPU-cycle frequency constraints of the mobile device and fog devices and transmission power constraints, the system aims to minimize time-averaged energy consumption while maintaining system network stability. In their framework, the processing of computation tasks is done either at the mobile terminal or are offloaded to its associated fog device that called helper through device-to-device communication technology or offloaded them through cellular network to a single edge server that is assumed to have enough computing capacity to process all the upcoming tasks. Compared with MEC, where all tasks are processed at the edge servers, their proposed scheme shows its effectiveness.

In [20], Ni et al minimised the overall latency of all mobile devices while keeping the energy usage of the mobile devices to a minimum given deadline and long-term energy consumption constraints. They have developed an online framework based on Lyapunov optimization theory. This allows them to design a control algorithm that jointly optimises task offloading and data caching considering only current system information. Compared to other benchmarks, the proposed scheme effectively reduces latency and maintains long-term energy consumption.

The problem of task offloading has been addressed along with service cashing in MEC-enabled dense cellular networks, aiming to minimise overall latency given the constraint of long-term energy consumption in stochastic systems [21]. The developed online algorithm, called OREO (Online service caching for mobile edge computing) which is based on Lyapunov optimization and Gibbs sampling that integrates edge server cooperation. Their proposed solution optimizes task offloading and dynamic service caching simultaneously. Simulation results show the effectiveness of the proposed scheme.

In an edge system consisting of one mobile device and one edge server, Zhang et al [22] conducted a study to investigate the trade-off between execution latency and energy consumption on the mobile device side. The authors proposed an online dynamic scheduling algorithm to solve the stochastic optimisation problem that aims to minimise energy consumption and delay under queue stability and battery level constraints. The algorithm proposed is based on the Lyapunov optimisation approach. The execution of tasks is done either locally on the mobile device or offloaded to the edge server. The authors stated that compared to other offloading schemes, the proposed scheme achieves the best results. However, based on their work, the scalability has not been investigated to prove the efficacy of their approach.

The problem of energy efficiency and delay trade-off is addressed in wireless powered mobile-edge computing (MEC) systems with multi-access schemes by Mao et al [23]. The problem has been formulated as a stochastic optimisation problem under the constraints of queue stability, maximum energy consumption, maximum communication, and computational resources. They have developed a Lyapunov optimization-based algorithm for joint computation offloading and resource allocation for multiple access schemes known as Time-Division Multiple Access and Frequency-Division Multiple Access. Simulation results demonstrate the trade-off between energy efficiency and delay. Additionally, their proposed scheme outperforms the results of the equal bandwidth allocation scheme.

Hazra et al [24] proposed an Energy-Efficient Task Offloading algorithm (EETO) based on the Lyapunov Optimisation technique. The authors aim to minimise the expected time-average energy consumption for each task considering the energy for uploading, processing, and downloading. The authors developed an optimal scheduling policy that considers the priority of tasks and the most efficient offloading decision that ensures minimum energy consumption in a stochastic fog system. The priority of tasks is evaluated upon arrival of these tasks at gateways by assigning them to proper queues. Then gateways make the optimal offloading decision on where to process them, locally at IoT device, fog devices or cloud servers. Their proposed scheme reduces energy consumption in the system by around 23.79% compared to a set of baselines. While their proposed scheme offers several advantages, it also has one notable disadvantage. All gateways make the optimal offloading decision simultaneously at the beginning of each time slot. This imposes a challenge in a stochastic fog system, as when making the decision based on the current information and status of the system, a single fog device that ensures the minimum energy consumption might be selected by all gateways to process tasks. As a result, QoS requirements considered in their work will be violated. This will result in exceeding the allowed waiting time threshold in queues and dropping tasks because their deadlines cannot be met. Further, the selected fog device does not possess sufficient computing and/or energy resources to process all the offloaded tasks.

Wang et al [55] aimed at minimising the long-term energy consumption of edge servers by developing an offloading strategy based on Lyapunov optimisation and a sleep control scheme. Their suggested strategy is based on the concept of putting edge servers in a sleeping cycle if the workload in their queues is below a threshold. Following that

Lyapunov optimisation is applied to determine the optimal offloading strategy. The authors stated that their proposed scheme saves more than 30% of total energy consumption compared to when no cooperation is involved, and no sleeping controller is included. Although the proposed scheme may have some potential benefits, there are also some concerns to consider, including: making a decision about sleep control. In more detail, a single edge server enters a sleeping cycle considering only its status and ignoring the status of all the neighbouring edge servers. This will lead to an outcome that violates QoS requirements and system stability. This will be noted if the system is extremely congested, and one edge server goes into a sleeping mode without helping in the process of the workload in other neighbours. Additionally, in a situation where all edge servers have workloads that are slightly below the sleeping threshold, all edge servers will enter a sleeping cycle. Therefore, no processing will occur.

In regard to the concept of Reinforcement/ Deep Reinforcement Learning, it has proven its effectiveness in dealing with uncertain dynamic environments, and it has been applied in a number of studies, including [25-32]. A study of industrial fog computing, where device-to-device offloading is used, by Wang et al [25] sought to minimise system cost which reflects the trade-off between energy consumption at mobile devices and service delay. In their work, reinforcement learning was exploited to dynamically make offloading decisions in each time slot. Their proposed scheme takes into consideration resource constraints, vehicle mobility, and offloading requirements. The authors developed two algorithms based on RL, namely the dynamic RL scheduling algorithm and deep dynamic scheduling to find the optimal offloading strategy. Comparing their proposed scheme to other benchmarks, their results show the effectiveness of their proposed algorithm in regard to total energy consumption and service delay.

A new paradigm was presented in [26], which combine digital twin networks with industrial Internet of Things (IIoT) in order to model network topology and mapping between physical entities and digital systems based on digital twin networks. With the aim to minimise the network efficiency which is the ratio of the long-term energy consumption caused by local execution and transmission of tasks to the corresponding long-term total executed tasks, the authors formulated the optimisation problem that addressed the stochastic offloading and resource allocation problems under the constraints of bandwidth allocation, transmission energy and computational resources. Using deep reinforcement learning (DRL), the proposed solution optimizes offloading decisions based on the current system states which include available computing resources, maximum transmission power, and bandwidth. Comparing the proposed algorithm with other benchmarks, their results demonstrate its effectiveness.

Li et al. [27] investigated cooperative MEC servers in an uncertain environment caused by vehicle mobility in dynamic MEC-enabled vehicular networks. The authors intend to reduce the service costs, which are represented by service delays and service failure penalties. The authors proposed a location-aware task offloading and computing strategy based on deep reinforcement learning to find the optimal offloading strategy. In their work, a neighbouring edge server that provides the least computing delay is a suitable candidate to help overloaded edge servers. As compared to other benchmarks, their results demonstrated that the proposed scheme achieves the lowest service cost in comparison with others.

In heterogeneous vehicular networks, Ke et al [28] addressed the computational offloading problem in MEC system and explored its efficiency in uncertain environment caused by unstable wireless channel and the availability of bandwidth. The authors aimed to reduce the total system cost which considers the trade-off between energy consumption of mobile devices, bandwidth allocation and transmission delay with continuous action space. An adaptive offloading method using deep reinforcement learning is proposed to determine whether the task should be processed locally at the vehicle or offloaded to the edge.

In a vehicular edge computing environment, Zhan et al [29] investigated the computational offloading problem aiming to minimise the system cost which is the trade-off between system delay and energy consumed by locally processing tasks at vehicle terminals and transmitting them. The authors model the stochastic optimisation problem as a Markov decision process (MDP) and then use deep reinforcement learning to solve it. The developed algorithm helps to find the optimal offloading scheduling policy that defines where and when to schedule and process each task. Compared to other baseline algorithms, their simulation results show the effectiveness of their introduced scheme.

In green industrial fog networks, service provisioning has been studied by Hazra et al [30]. In their work, Hazra et al analyse the optimization of energy and delay in conjunction with a weighting factor. Their proposed approach is a two-step scheme, consisting of task partitioning and intelligent service provisioning. In the task partitioning strategy, a large set of data is divided into a sequence of subtasks. To accomplish distributed workload optimization among computing devices, such as local IIoT, master fog nodes, near fog nodes, and cloud servers, an intelligent service provisioning model is built based on a DRL approach called Deep-Q Network (DQN). Simulation results demonstrate that the proposed framework minimises total energy consumption and delay by around 24.5% and 16.3% respectively compared to a set of benchmarks.

Chen et al [31] addressed the problem of stochastic computational offloading in a sliced radio access network (RAN). The problem is transformed into a Markov decision process (MDP) with the objective of minimizing the system's long-term utility. The system utility includes task execution delay, task queueing delay, task dropping, the penalty of execution failure, and the payment of accessing services in edge servers. The authors proposed an online deep state-action-reward-state-action-based reinforcement learning algorithm (Deep-SARL) that combines the use of the Q-function and double DQN. According to the results of simulations, the proposed framework is superior to existing offloading algorithms from a system utility perspective.

Yan et al [32] focuses on minimizing the weighted sum of task execution time and energy consumption of mobile devices in edge computing systems. To achieve this goal, the problem is formulated as a mixed integer optimization problem with the aim to optimise joint computational offloading and computation resource allocation. As part of the proposed solution, deep neural networks (DNNs) are used to learn from experiences and improve offloading policies. In addition to that the offloading actions in DRL are based on a novel method called Gaussian noise-added order-preserving quantization. Additionally, a heuristic approach named the one-climb policy is developed to speed up the learning process in DRL. Based on their simulation results, the proposed scheme achieves near-optimal results.

Moreover, few studies have considered applying solutions based on the combination of Lyapunov optimization and Reinforcement/Deep Reinforcement learning in stochastic environments, as in [33, 34] in order to address these problems.

In a 5G-based smart city, Xu et al [33] developed a cloud-edge-terminal model for the joint optimization problem of communication and computational resources under energy consumption constraints to minimise overall processing delay. The authors first formulated the problem as an online optimisation problem and proposed Lyapunov optimisation with drift plus penalty to solve it. The framework of Lyapunov optimisation theory is exploited to design virtual energy queues in mobile devices and edge servers to monitor the status of energy consumption. Additionally, applying Lyapunov optimisation helps to decompose the long-term optimisation problem into two sub-problems called task offloading and task migration. The task offloading problem is formulated as a MDP and solved using Q-learning in order to determine where the task should be processed locally at smart mobile devices, on edge nodes, or on cloud servers. Additionally, the task migration problem is solved based on the proposed Lagrange-based migration algorithm. In this method, heavy-loaded edge nodes migrate their workload to under-loaded edge nodes. This is done to balance workload and improve resource utilisation. Their results reflect the efficiency of the proposed algorithm as compared to other approaches.

Bae et al [34] developed a reinforcement learning approach to Lyapunov optimization to minimise the time-average penalty cost of energy consumption while maintaining queue stability in the system. The problem is transformed into MDP and solved based on the proposed scheme. In their proposed scheme, the reward function is created by taking into account the dynamic and penal functions that take into account the cost of energy consumption in edge servers and the transition from the edge to the cloud along with the queue stability based on a one-step difference. The proposed scheme helps to find the optimal task offloading between edge nodes and the cloud server when the arrival process of computation tasks is random. The authors stated that the efficiency of their formulated reinforcement learning approach to Lyapunov optimisation is attributed to the effective design of the reward function and the state and action spaces.

Applying a different scheme than Lyapunov theory and DRL, Zhang et al [35] investigated the problem of minimising delay in uncertain fog enabled IoT-eHealth networks where the movement of IoT-eHealth devices is random. In order to accomplish the stated objective, the authors propose a multi-stage stochastic programming approach that considers offloading decisions, resource allocation, and task migration between fog servers. In their work, the process of offloading consists of several stages that are carried out sequentially, the first stage starts when an eHealth device decides whether to offload its workload or not to the nearest fog server. This is based on its maximum energy consumption threshold. In the second stage, the fog server decides how much of its computational resources should be given and how much of workload can be processed. Finally, the associated fog server makes the task migration decision to a neighbouring fog node after processing the workload. This decision is made to return the processed workload to the eHealth mobile device based on its location. Compared to other baselines, their work achieves the least delay.

2.3.1 Comparison of the State-Of-The-Art

In table 2.2, an overview of all the cited relevant works is shown. It can be seen that when addressing the problem of stochastic computational offloading, previous works used techniques such as Reinforcement Learning (RL), Deep Reinforcement Learning (DRL), and Lyapunov over other Artificial Intelligence techniques such as Deep Learning (DL). The reason for that is attributed to the structure of these techniques that suits the features of stochastic environments and the problem of computational offloading. In more details, in stochastic environments where the input data such as the arrival of tasks is not known in advance, we don’t have labelled datasets. RL and DRL do not require labelled datasets as the agents are learning through the interaction with their environment. However, techniques such as DL requires labelled datasets for training which is challenging to have in a stochastic setting.

Additionally, the problem of computational offloading in stochastic environment, involves making a series of decisions that influence future states and rewards. So, the aim is to optimise a sequence of decisions over time. This can be accomplished through the use of RL/DRL as the learning agent will not only focus on maximising its current reward but also its future rewards. In DL, agent takes action based on the current inputs without taking into consideration the influence of this action in the future states.

Furthermore, based on the structure of these techniques, RL/DRL are better suited to scenarios in which an agent interacts with its environment, updates its learning policy, and make decisions over time to accomplish a goal. Examples of such environments include training a robot, playing a game, and learning to drive autonomous vehicles. This is different in DL, DL performs better in tasks that involve classification, pattern recognition, and prediction based on large datasets. This well-suited to environments such as Medical Imaging to help Identifying tumours, Image Classification, and Speech Recognition. In our environment, the agent needs to learn about its environment to make the optimal or near-optimal offloading decisions, which makes the use of RL more suitable.

After analysing related research studies concerning computational offloading and resource management in stochastic fog systems, several limitations were identified. It was determined that no scheme had addressed the optimization problem of energy consumption, delay, and security simultaneously. While various schemes deal with individual aspects of these challenges, such as energy efficiency and delay, a comprehensive approach that engages all three factors has yet to be developed [15-35, 55].

Moreover, research on computational offloading and resource management in stochastic fog systems has increasingly focused on resource allocation rather than resource provisioning in regard to server energy management [15-35]. While allocating resources for tasks is a crucial component of the decision-making process associated with computational offloading problems, resource optimization is also critical, especially in resource-constrained environments such as fog systems that can help conserve energy. In stochastic fog systems, both resource allocation and resource optimization are crucial factors to consider when dealing with computational offloading problems, as they can have a substantial impact on the system's performance.

Another important aspect that needs to be investigated in stochastic computing systems is the cooperation between fog/edge devices, where multiple devices work together to process workloads efficiently. While several studies have examined the benefits of such cooperation, it could negatively affect overall system performance. This is especially true when it comes to selecting the most suitable candidate to process a given workload in stochastic systems. Due to the inherent randomness and variability of the system, the main challenge is determining which server will provide the most effective and efficient processing [16, 18, 21, 27, 30, 33, 35, 55].

Cooperation in stochastic systems has previously been investigated by selecting the most suitable neighbouring fog/edge device in a distributed or centralized manner. However, some studies have resulted in multiple fog/edge devices selecting the same neighbour, violating QoS requirements in the system [16, 21, 33]. In other studies, a cooperative centralized approach was used by local controllers, but the details of how the framework operated were not provided [18, 27, 35, 55]. In one study, IoT devices sent their tasks to a master fog node, which was aware of the status of all traditional fog nodes and assigned workloads to the proper fog node for processing [30]. Future research efforts should focus on proposing an appropriate approach that addresses the limitations encountered when cooperation is considered in stochastic systems.

Prior research has mostly disregarded two crucial factors, namely priority scheduling and heterogeneous task characteristics, except for one study [24]. Considering these two factors can significantly influence the performance of stochastic fog systems as they closely resemble real-world scenarios. Although earlier research has made significant contributions to energy optimization and delay reduction, their methods may not be adequate for the distinctive requirements of stochastic fog systems. Hence, further exploration is necessary to address these challenges and improve the performance of fog systems.

To tackle all mentioned challenges, we propose a new approach, named Joint Q-learning and Lyapunov Optimisation (JQLLO) algorithm, which employs Reinforcement Learning and Lyapunov theory.

Table 2.2: Computational offloading and Resource management in stochastic fog computing system State-Of-Art Comparison.

Ref	**Objective**				**Resource allocation**	**Server power management**	**Fog Cooperation**		**Where tasks executed**	**Heterogeneous Tasks**	**Priority-aware Scheduling**	**Proposed solution**
		Energy										
	delay	**yes**	**which**	**Security**			**Yes**	**Selection criteria**				
Pu et al [15]	✗	✓	End users' devices	✗	✓	✗	✗	✗	• locally • nearby mobile user.	✓	✗	Lyapunov Optimisation
Guo et al [16]	✓	✓	• Edge nodes • Cloud server	✗	✓	✗	✓	✗	• Local edge node • Neighbours/cloud	✗	✗	
Chang et al [17]	✓	✓	• Mobile devices	✗	✓	✗	✗	✗	• Locally at mobile device • The Fog node	✗	✗	
Chen et al [18]	✓	✓	Edge servers	✗	✓	✗	✓	✗	• local server • neighbouring server	✗	✗	
Qingmin et al [19]	✓	✓	Mobile devices	✗	✓	✗	✗	✗	• Mobile device • Fog device (helper) • Edge device	✗	✗	
Ni et al [20]	✓	✓	Mobile device	✗	✓	✗	✗	✗	• mobile device • local edge server	✗	✗	
Xu et al [21]	✓	✓	Base stations where edge servers are located	✗	✓	✗	✓	✗	• local edge server • neighbouring edge server • remote cloud server	✗	✗	
Zhang et al [22]	✓	✓	mobile devices	✗	✓	✗	✗	✗	• Locally at the mobile device & Edge server	✗	✗	
Mao et al [23]	✓	✓	End user devices & edge servers	✗	✓	✗	✗	✗	• Locally at end user devices & Edge server	✗	✗	
Hazra et al [24]	✓	✓	end devices, fog nodes and cloud servers	✗	✓	✗	✗	✗	• local end devices • fog nodes/cloud servers	✓	✓	

Ref	Objective				Resource allocation	Server power management	Fog Cooperation		Where tasks executed	Heterogeneous Tasks	Priority-aware Scheduling	Proposed solution
	delay	Energy		Security			Yes	Selection criteria				
		yes	which									
Wang et al [55]	✓	✓	Edge servers	✗	✓	✓	✓	✓	• local fog node • neighbouring fog nodes	✗	✗	
Wang et al [25]	✓	✓	mobile devices	✗	✓	✗	✗	✗	• Locally at the mobile device • Nearby mobile device • An edge server	✗	✗	Reinforcement and Deep reinforcement learning
Dai et al [26]	✗	✓	• Local IIOT device Base station where edge server is located	✗	✓	✗	✗	✗	• locally at IIOT device • local edge server	✗	✗	
Li et al [27]	✓	✗	✗	✗	✓	✗	✓	✗	• Local MEC server • Neighbouring server	✗	✗	
Ke et al [28]	✓	✓	Vehicles	✗	✓	✗	✗	✗	• Locally at vehicles • local edge server	✗	✗	
Zhan et al [29]	✓	✓	Vehicles	✗	✓	✗	✗	✗	• Locally at vehicle terminal • At edge server	✓	✗	
Hazra et al [30]	✓	✓	IoT devices /fog servers /cloud server	✗	✓	✗	✓	✗	• IoT device • Master/ Nearby fog device • Cloud server	✗	✗	
Chen et al [31]	✓	✗	✗	✗	✓	✗	✗	✗	• Mobile device • edge server	✗	✗	
Yan et al [32]	✓	✓	Mobile device	✗		✗	✗	✗	• mobile device • edge server	✗	✗	

Ref	Objective				Resource allocation	Server power management	Fog Cooperation		Where tasks executed	Heterogeneous Tasks	Priority-aware Scheduling	Proposed solution
		Energy										
	delay	yes	which	Security			Yes	Selection criteria				
Xu et al [33]	✓	✓	Smart mobile devices Edge nodes	✗	✓	✗	✓	✗	• Smart mobile device • Local edge node • Neighbour edge node • Cloud server	✓	✗	Lyapunov optimization And reinforcement learning
Bae et al [34]	✗	✓	Edge node	✗	✓	✗	✗	✗	• Edge node • Cloud node	✓	✗	
Zhang et al [35]	✓	✗	✗	✗	✓	✗	✓	✗	• eHealth device • fog servers	✗	✗	Multi-Stage Stochastic Programming approach
Our proposed work in chapters 5, 6, and 7	✓	✓	Fog devices	✓	✓	✓	✓	✓	• Locally at primary fog nodes • Neighbouring fog nodes • Cloud server	✓	✓	Combination of Lyapunov Optimisation and Reinforcement Learning

2.4 Background on Reinforcement Learning (RL), Q-learning algorithm, and related works on Cooperative Q-learning

This section describes the main components of RL and Q-learning. Additionally, previous studies that considered cooperative Q-learning are analysed.

2.4.1 Reinforcement Learning (RL)

Reinforcement learning (RL) refers to a technique in machine learning that trains agents to behave according to the information they obtain from their environment. The basic goal of RL is to identify an optimum strategy that maximises cumulative rewards over time. RL algorithms accomplish this objective by interacting with their environment, and depending on their actions they obtain feedback as rewards or punishments. Then using this input to enhance their decision-making process [56]. Numerous applications, such as natural language processing, control systems, and robotics, have made extensive use of RL.

In uncertain and dynamic environments, RL has been applied as a solution to many resource-allocation and management problems and has proven its efficiency compared to other traditional approaches [57, 58]. RL problems with finite state spaces can be solved by three main approaches: Dynamic Programming (DP), Monte Carlo method (MC), and temporal-difference learning (TD) [59, 60].

DP, MC, and TD are three main methodologies in the field of reinforcement learning [61]. A DP is a technique for addressing optimization problems in a recursive manner by decomposing them down into smaller subproblems, in order to recursively solve them. DP requires a comprehensive understanding of the environment, including transition probabilities and rewards [62]. MC can be considered a model-free learning method in which value functions are estimated by sampling interactions with the surrounding environment. MC can be applied in applications where there is no need for knowledge about the environment and learning occurs from experience [62]. A TD is a model-free learning technique. Its value function is updated based on the difference between the current estimation and the previous estimation [62].

Since we are dealing with a stochastic environment where the transmission probability from one current state to another is not known in advance for the learning agent, we adopt the TD method and more specifically a model-free reinforcement learning algorithm called Q-learning [63-65]. The reason for selecting TD over DP is that the former does not require prior knowledge about the environment, e.g., a model, agent's reward, and next state probability distribution [59, 66]. In stochastic environments, this information is not provided beforehand. Also, comparing TD to MC, in MC, the agent only gains knowledge after completing an entire episode, while in TD, the agent gains knowledge after each time step [59, 66]; which speeds up the learning process and the agent in TD learning learns more and faster compared to the agent in MC method. In conclusion, TD combines DP and MC [59]. Similar to DP in the sense that the agent learns about its environment at every step and updates its policy accordingly. This is without having to wait for an entire episode to end [59]. In addition, the learning agent in the TD approach can learn from repeated experiences, similarly to MC, without needing to have a model of the environment [59].

2.4.1.1 Elements of Reinforcement Learning

The main elements of RL are *agent*, *model of the environment*, *policy*, a *reward signal*, and a *value function* [59].

- In RL, ***agents*** are learning/intelligent entities that have the capability of observing their environment and making decisions based on the observations they have made.
- A ***model*** of an environment is a model that mimics the properties of the environment that the agent needs to learn about in order to function effectively. The learning agent interacts with the environment which provides a state. The agent observes the state and then depending on its policy, the agent selects an

action. Following that and based on the selected action, the agent receives a reward, and a new state is encountered. Again, based on the received reward and the evolved state, the agent updates its policy and chooses an action. The process is continuous until the number of trainings is reached. The agent will learn based on trial and error in accordance with the received feedback from its performed actions.

- The ***policy*** represents how the agent's learning evolves, and it maps from a set of states to its most effective actions. The goal of reinforcement learning is determined by a reward signal that defines the goal of the problem.
- ***Reward signal***: The agent seeks to maximise its total long-term rewards, and rewards determine how successful the previously performed actions of the learning agent were in a previous state.
- ***Value function***: Rewards only shows how good or bad the action performed by the learning agent in the state; however, the value function not only determines how good the performed action was in the previous state but also how good it is in the upcoming states that could be encountered by the learning agent in the future.

The agent-environment interaction in RL is shown in Figure 2.1. If this is the first encounter between the learning agent and its environment, the learning agent observes the current state, s(t), and based on its policy, the agent will perform an action a(t). Following that, the agent will receive a reward from its environment based on how efficient its actions were during the previous time step. After receiving the reward and observing the new state, the agent will update its policy, and then will make an action according to its policy. The process is repeated until the agent develops its policy that helps the agent to perform its most effective actions in its environment at any encountered state.

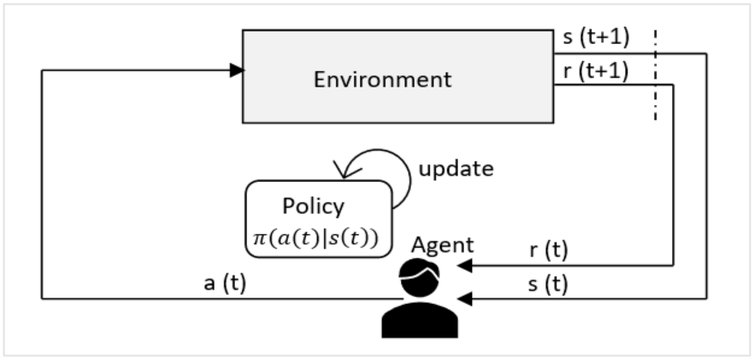


Figure 2.1: Environment-agent interaction in Reinforcement Learning.

2.4.1.2 Markov decision process (MDP)

To solve an optimised problem using RL, first the problem is transformed into MDP, which is a mathematical scheme used to design the decision-making process in the RL problem. In this context, MDP is used to describe an environment for reinforcement learning. An environment defined by MDP is denoted by $< S, A, P, R, \gamma >$ [66, 67], where S determines the set of states a learning agent observes when interacting with the environment, A is a set of actions available for the learning agent, P is represented as P (s, a, s') which is the probability distribution from the current state (s) to next state (s') after performing an action a which belongs to a set of actions A. R is the received reward after performing an action a by the learning agent. γ is a discount factor with a value between zero and one, and it shows the importance of future rewards compared to the current reward, if $\gamma = 0$, this means that the learning agent only concerns about maximising its current reward and when $\gamma = 1$, the agent only focuses on its future rewards.

In our environment, the probability distribution from the current state to the next state is not constant and it changes over time due to the dynamics in the system. This is called a non-stationary environment [68]. A non-stationary environment can be defined as a system in which the underlying properties, dynamics, or parameters are not constant over time. This type of environment poses a challenge for traditional RL techniques. This is because RL approaches are designed to operate in stationary environments where the underlying distributions and relationships remain constant. Applying RL approaches will help to find the optimal decisions in a stationary environment, however, in non-stationary environments, RL approaches can lead to sub-optimal decisions [68]. In this manner we design an algorithm to complement the RL approach so that it performs well in such non-stationary environments.

2.4.2 Q-learning

Q-learning is a model-free reinforcement learning technique and TD control algorithm that known as off-policy [66]. As the name implies, Q-learning does not require the use of a model to train a learning agent, nor does it apply the transition probability associated with MDP for transforming from one state to another. The definition of "off policy" requires an understanding of the difference between the behaviour policy and the update policy, because the two policies are involved in the process. The behaviour policy tells the agent how to act in a given situation. Based on a greedy algorithm, the agent will act randomly or select an action with the highest Q value in the state-action pair. In the update policy, the Q-value for the state-action pair is updated not only considering the current action taken, but also considering the expected suitable action in the next observed state [66]. In off-policy, the behaviour policy and update policy are different whereas they are the same in on-policy approach [66].

The following terms should be defined when solving an optimisation problem with Q-learning.

2.4.2.1 State Space:

The state space in Q-learning describes the collection of all potential states that the agent may experience in the environment [59]. Depending on the characteristics of the surrounding environment, the state space may be discrete or continuous. It plays a vital role in measuring the success of Q-learning in addressing reinforcement learning problems.

2.4.2.2 Action Space:

In Q-learning, the action space is the collection of all potential actions the agent may perform depending on the environment state [59]. Depending on the characteristics of the surrounding environment, the action space may be discrete or continuous. In order to find the optimal action-value function, the Q-learning method is used to map each state-action combination to a value that represents the projected reward that will result from a particular action taken from a particular state.

2.4.2.3 Epsilon-greedy strategy:

For the agent to select the optimal action in each state, the agent should first gain knowledge of its environment. As part of its analysis of the environment, the agent must take random actions in the current state and observe the reward and subsequent state. By doing so, it will be able to gain a better understanding of it. This is called **exploration** [59]. However, if the agent always explores the environment by taking random actions, the agent will not get the benefit of its previous knowledge of the environment. Therefore, the agent will not exploit what he has learnt, and this is called **exploitation**. Also, if the agent always relies on the knowledge he built and for each given state the agent selects the action with the highest Q-value, the agent will easily become stuck at a local optimum due to being trapped in a limited search area. In this manner, the learning agent should be capable of recognizing when to explore its environment and when to exploit it. In this context, several algorithms have been proposed to address the trade-off between the terms exploration and exploitation, and Epsilon-greedy is one of them [66].

At the beginning of the learning process, the learning agent has zero knowledge about the environment, and as the agent is interacting with its environment it builds its knowledge and its experience which help the agent to know its environment and act in an optimal manner. In the Epsilon-greedy strategy, we define a parameter called epsilon (ε) which determines the agent's action in each state. To put it another way, the agent will explore its environment by performing randomly, or exploit the knowledge it gains about its environment by selecting an action with the highest Q value based on the epsilon value. This is shown in the following equation.

$$Action\ a\ (t) = \begin{cases} \max \mathrm{Q}\ (s, a) & b > \varepsilon \\ random\ a & otherwise \end{cases}$$

At the beginning of the training, the epsilon value is set to one, and the agent at the beginning of each state, generates a random number (b) between zero and one. If the generated number is higher than the epsilon value, then the agent will take the action that has the highest Q-value stored in the Q-table for the observed state. Otherwise, the agent will choose any action randomly.

As Epsilon has a value of one, the agent will always choose random actions and observe the reward. At the end of each episode, the epsilon value decreases slightly. The epsilon keeps minimising each time as the agent learns about its environment until reaching a threshold value (e.g., 0.01).

2.4.2.4 Reward Function:

After taking an action in a particular state, the learning agent in Q-learning will receive a reward. The reward will determine how good or bad the performed action for the encountered state. In defining the reward function, it should be connected to the objective function [69, 70]. In RL, the learner aims to maximise the received reward.

Shaped Reward Function

To deal with non-stationary environments where state transition distributions is changing over time, a deterministic reward function may not help the learning agent know its environment, instead the agent will keep adjusting its policy, which makes the learning process difficult. Reward shaping helps to guide the learning agent during the learning process by incorporating the dynamic nature of its environment into its reward [71, 72]. Moreover, it is stated that a shaped reward function speeds up the learning process [73]. In recent works [74, 75], shaped reward functions have been considered for non-stationary environments It is also stated that reward shaping can be formulated as a weight that is adjusted during the learning process. This weight considers the original reward and the punishment/penalty if the state space and action space are limited [76].

2.4.2.5 Q-table

One of the main components in Q-learning is the Q-table [59]. Q-table contains the Q-values of the encountered pairs of states and actions, which are determined based on the received reward and discount factor of estimated received reward. It has the size of SXA. The Q-values of all state-action pairs are zero at the beginning of the system. An example of the Q-table is shown in table 2.3, which stores the Q value for the state-action pair, and it is updated after taking each action.

Table 2.3: Initial Q-table with zero Q-values

		States				
		<5,0,0,0,0,0,0>	**<4,1,0,0,0,0,0>**	**<0,0,5,0,0,0,0>**	**…….**	**<3,1,0,0,0,0,1>**
Actions	**0**	0	0	0	0	0
	1	0	0	0	0	0
	2	0	0	0	0	0
	3	0	0	0	0	0
	4	0	0	0	0	0
	5	0	0	0	0	0

2.4.2.6 Updating Q-table

At the beginning of the learning process, the Q-table initialised with zero Q-values as in table 2.3. After taking each action, the table is updated with the Q value associated to the (s, a) pair.

For a given state (s) after taking an action (a), observing the reward (r) and the following/transitioned/new state (s'), the Q-value for the pair state-action is updated in the Q-table following bellman equation below [59].

$$Q^{new}(s,a) = Q^{old}(s,a) + \alpha \left[r + \gamma \, max_{a'} \, Q(s', a') - Q^{old}(s,a) \right] \qquad (2.1)$$

In the above equation, $Q^{new}(s,a)$ is the new Q-value that we are calculating and $Q^{old}(s,a)$ refers to the stored Q-value in the Q-table for the pair state-action. **Learning rate α** is a parameter between the value one and zero. If it is zero it means that the agent will not update the Q-value for the state-action pair, hence no learning is occurring. If it is set higher, e.g., 0.9, it means that the agent is learning and updating its Q-value considering the new received reward along with the expected maximum reward in the following state (s+1). Parameter $\boldsymbol{\gamma}$ is the **discount factor** and it is value is between zero and one, it measures how important do we find future rewards versus immediate reward, it is usually set high, e.g., 0.90, this is to help the agent to select the best action in the current state that will not only consider maximising its current reward but also maximising the upcoming rewards. $max_{a'} \, Q(s', a')$ is the maximum Q-value that can be achieved in the next state (s') by selecting its best action a'.

2.4.3 Cooperative Q-learning (CQL) algorithm

In a multi-agent learning system, there are many agents interacting and learning individually about their environment to achieve a common objective. In our work, each learning agent will learn its environment in its cluster and the action taken by a single learner will not influence the environment of other learners. In such a system, it is logical that agents who learn individually will accomplish their objective more quickly if they start sharing their experiences while learning as compared to when sharing is not occurring [77], and this is called cooperation. Cooperation refers to the process when agents work together and sharing experiences to achieve a shared goal. When RL is applied as a learning tool in multi-agent systems, there are three possible approaches to applying cooperation [77]. First, by sharing instantaneous information such as the states encountered, the actions taken and the rewards received, which is referred to as sharing sensations [77]. Secondly, by exchanging episodes, where a single episode consists of state, action, and reward. Finally, by sharing the knowledge they gained represented by the learned policies.

The process when learning agents apply the Q-learning algorithm during the learning process of their environment, and share their knowledge and their learning outcomes is called cooperative Q-learning [78]. Cooperative Q-learning can be accomplished in two consecutive stages [78]. Firstly, each individual learner applies the Q-learning algorithm while interacting with its environment. During the learning process, each learner builds its experience represented by its Q-table while interacting with its environment. Secondly, the interaction phase where learners share their learning outcomes with others, and based on the selected cooperative approach, they update their Q-tables.

The impact of cooperation between agents while applying RL has been investigated by Tan [77]. The author addressed the problem of hunters whose goal is to chase preys. Moreover, the author investigated three cooperative approaches named sharing sensation, sharing episodes, and sharing learned policies which are mentioned above; and compared that to when cooperation is not considered. As a result of the experiments, the technique of sharing sensation appears to be beneficial if the information is related to the environment of other learners. Regarding the other two cooperative approaches, the results suggest that these tools are helpful for speeding up the learning process but at the expense of communication. Finally, having the same number of training episodes, cooperative agents can learn faster about their environment and converge sooner compared to non-cooperative agents.

Another study that shows the impact of cooperative learning was conducted by Abedalguni et al [78]. Using well-known cooperative Q-learning algorithms named BEST-Q, AVE-Q, PSO-Q, and WSS algorithms, Abedalguni et al [78] investigated the impact of these algorithms on the learning process and compared them to when cooperation was not involved. In their work, they classified the level of experience of learning agents based on the number of training episodes; for example, an agent who trained for 100 episodes is more expert than an agent who trained for 50 episodes. Their results show that all well-known cooperative Q-learning algorithms impact the system in the same manner. Additionally, the results when cooperation is considered outperform those when no cooperation is involved if the level of agents' experience is different. In more detail, the authors stated that if cooperative agents have the same level of experience, sharing their knowledge is not beneficial. The system will perform similar to when no cooperation was considered. However, the authors did not specify the number of the level of experience upon which cooperation is not beneficial. This is because the agent could have the same small number of training episodes, for example 2 episodes, and cooperation in this case is still useful. Additionally, the authors did not give more information about the sequence of states different agents encounter, in terms of being the same sequence or not, and whether or not the same action is performed by these agents if they encounter the same state. Accordingly, we do not believe that we can make a definitive statement regarding whether or not cooperation is beneficial based solely on the level of experience agents have. This is represented by the number of training episodes. As cooperation is helpful even if agents have the same level of training, as in our experiment. Therefore, further clarification is needed in this regard.

There are other factors that would impact on cooperation among agents while training. One of them is the sequence of events each agent has encountered in the environment, and how each agent acts towards these events. If all training agents are facing the same sequence of events, e.g., same states, and they perform different actions on these events, cooperation is still beneficial. Some agents would benefit from the experience of other agents who performed different actions from them and see how their environment acted. On the other hand, if agents are encountering the same sequence of events and they all perform the same actions, cooperation in this case could not be helpful.

One of the well-known cooperative Q-learning algorithms is called BEST-Q approach and it has been investigated in [78-80]. BEST-Q algorithm has been proposed by Iima and Kuroe [80] along with other cooperative approaches called AVE-Q and PSO-Q algorithms. In BEST-Q algorithm, the highest Q-value for each state-action pair is determined, and then all agents will update the Q-value of the same state-action pair in their Q-tables by replacing their old value with the best Q-value. At the end of the updating process, all learning agents will have the same Q-table [78]. Among all cooperative Q-learning approaches, we used the BEST-Q algorithm. Abed-Alguni in [81] states that it performs best among all cooperative approaches.

In BEST-Q algorithm, the best Q-value is calculated using the following equation.

$$Q_{old\ value}\ (s, a) \leftarrow Q_j^{best}(s', a') \text{ if s = s' \& a = a'} \qquad [78]$$

2.5 Conclusion

This chapter provides related works that address the problem of offline/online computational offloading and resource management. Afterwards, works investigating computational offloading and resource management in stochastic fog systems are presented. Finally, a brief background on RL and Q-learning algorithms is provided. In addition to that an introduction and literature review of cooperative Q-learning is discussed.

3 Minimising Delay and Energy in Online Dynamic Fog Systems

3.1 Introduction

In online fog systems, optimising delay and energy consumption is one of the critical factors to ensure efficiency, reliability, and performance. For real-world IoT applications, a delay can impair the system's ability to process data and make decisions in real-time. Furthermore, not managing the fog system's energy consumption can result in higher operational costs and shorter lifespans. In this chapter we address the problem of computational offloading and resource management with the aim to minimise the delay and energy consumption in online dynamic fog systems using static and dynamic offloading threshold that determines whether the task is processed locally or offloaded to a neighbour. This chapter is structured as follows. System modelling and constraints is described in section 3.2, which includes network diagram, application module description, communications and constraints between vehicles and fog nodes, between fog nodes, and between fog nodes and the cloud serves. The optimisation problem of minimising the delay and the energy consumption is formulated in section 3.3. In section 3.4, the proposed solution regarding the two methods is discussed. The performance evaluation including the environment settings, set of experiments, and the results is provided in section 3.5. In section 3.6, types of applied offloading thresholds and related results are discussed. Finally, the conclusion of this chapter is in section 3.7.

3.2 System Modelling and Constraints

System model is presented in section 3.2.1, and Types of Connections and Constraints is described in section 3.2.2.

3.2.1 System Model

This section will discuss network diagram and application module.

Network Diagram

An overview of the fog computing architecture is shown in Figure 3.1 and consists of three layers:

- The IoT devices layer: this layer contains of mobile vehicles. The vehicle node has a set of sensors. Each sensor transmits different types of tasks and an actuator and once they are within the coverage radius of a fog node, they will send their tasks. Two types of tasks are emitted by the mobile vehicle. The first type is non-urgent and contains information such as current location, speed, and direction of the vehicle. The second task is an urgent request that requires a quick response. For example, this task may contain a video stream of a moving vehicle’s surroundings, which requires quick processing by fog nodes to help avoid collisions. This might be important, especially for autonomous driverless vehicles.
- Fog computing layer: this layer consists of a set of fog nodes and a fog controller. Fog nodes reside in roadside units (RSU) that are deployed in different areas of a city. Fog nodes can communicate with each other if they are located within each other’s vicinity **[82]**. Fog nodes can form an ad hoc network between themselves to share and exchange data. All fog nodes are logically connected to the fog controller which monitors the performance of all fog nods and manages the resources. The fog nodes are static and receive two different types of tasks from all vehicles within their radius. These tasks are called priority and non-priority tasks. Regarding priority tasks, fog nodes process requests generated by a user’s sensor and send the response back to the user. For non-priority tasks, fog nodes do some processing of the information provided by the vehicles within their range and send the results to the cloud for further analysis and storage for retrieval by traffic management organisations.

- Cloud computing layer: this layer contains cloud servers. It manages and controls the traffic at the city-level based on historical data received by fog nodes.

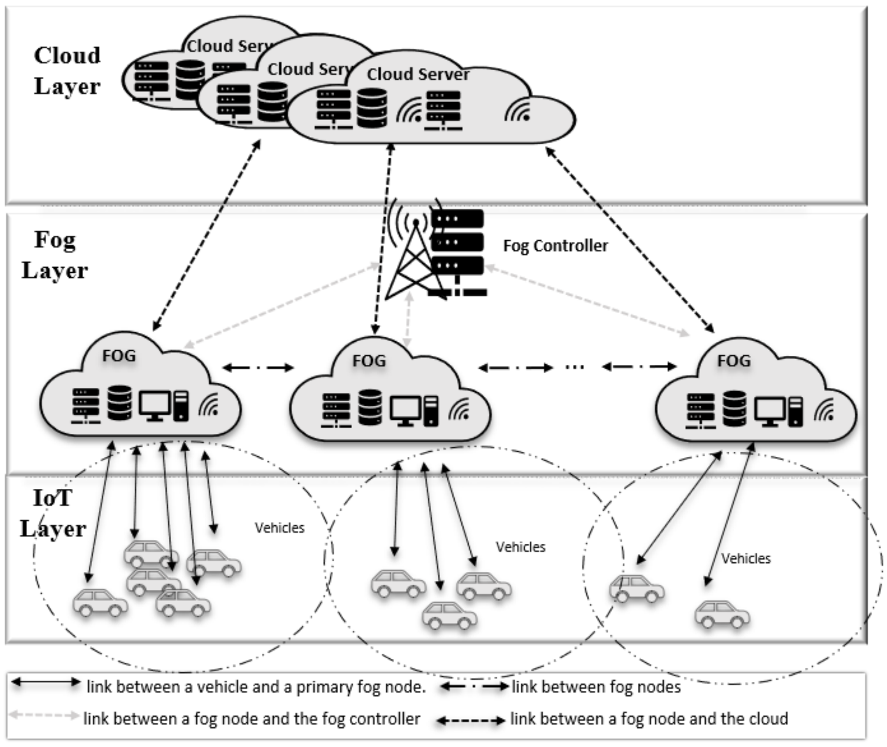


Figure 3.1: Fog Computing Model.

Application module description

The application model of this study consists of three modules named Road Monitor, Global Road Monitor and Process Priority Tasks. The first two modules are responsible for traffic light control systems and the last module is only for processing end-user priority tasks. The function of each of these modules is as follows:

• **Road Monito**r: this module is placed in fog nodes. If a vehicle enters an area within the coverage of a fog node, the sensor automatically sends the current car location, its speed, weather conditions and road conditions to the connected fog node for analysis. Then the module processes these data, and the results are sent to the cloud for further analysis.

• **Global Monitor**: this module is placed in the cloud and receives the collected data from fog nodes (after being processed by the Road Monitor module), analyses these data, and stores the results.

• **Process Priority task**: this module is placed in fog nodes and is responsible for processing the priority requests from the user. The results are then sent back to the user. The application in iFogSim is represented as a directed acyclic graph $(DAG) = (M, E)$ where M is the set of application modules deployed $= \{m_1, m_2, m_3, \dots, m_n\}$, e.g., Process Priority Task, Road Monitor and Global Road Monitor modules. Between application modules, there is a set of edges belonging to E, which represents the data dependencies between application modules. This is shown in Figure 3.2.

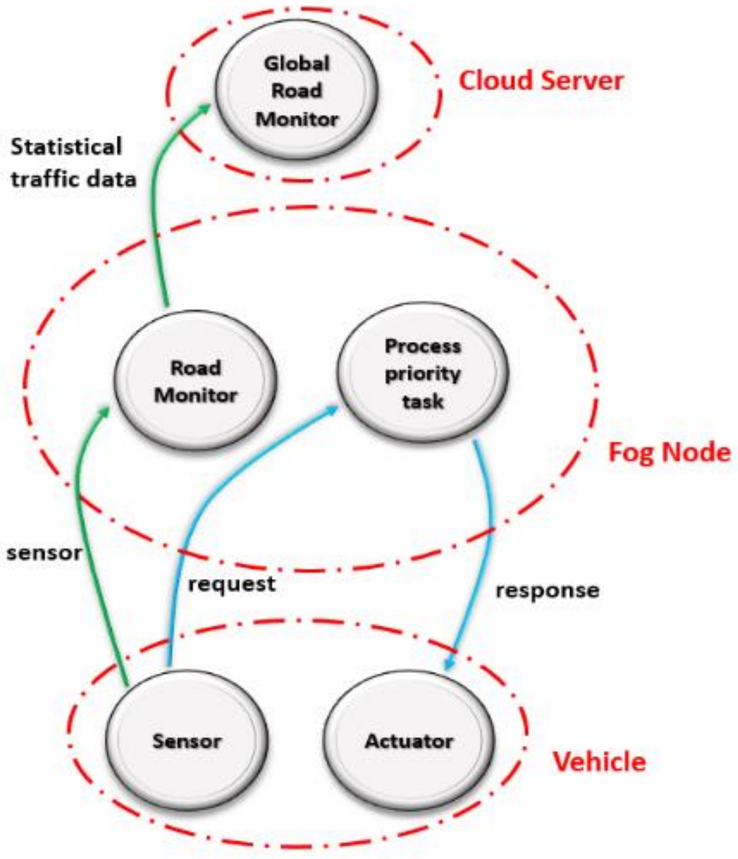


Figure 3.2: Directed Acyclic Graph (DAG) of the application model.

3.2.2 Types of Connections and Constraints

This section describes the connections between a vehicle and a fog node, between fog nodes, and between fog nodes and cloud. Also, the set of constraints involved within these connections.

Connection between Vehicles and Fog nodes

The connection between a vehicle and a fog node is made with communication and processing constraints.

- **Communication Constraints**

Vehicles connect to the fog node if and only if it is located within its communication range, as constraint (3.1)

$$D_{v,f} \leq \ max\ Coverage_f;\ \forall\ v \in V, \forall\ f\ \in\ FN \tag{3.1}$$

Where V is all vehicles, v one vehicle, FN is all fog nodes and f is one fog node. $D_{v,f}$ is the distance between a vehicle v and a fog node f, is calculated as

$$D_{v,f} = \ \sqrt{\left(X_v - X_f\right) + (Y_v -\ Y_f\ )};\quad \forall\ v \in V, \forall\ f\ \in\ FN \tag{3.2}$$

where (X_V, Y_V) and (X_f, Y_f) are the coordinates of a vehicle and a fog node, respectively. If a vehicle is located within the coverage radius of more than one fog node it will connect to the nearest fog node. This to reduce delay because the expected arrival time of the task at the connected fog node depends on the transmission and the propagation delay, but the propagation delay depends solely on the distance between the two connected objects. Propagation delay (PD) is calculated as

$$PD = \frac{D_{v,f}}{PS} \tag{3.3}$$

Following [83], we assume that the speed of signal propagation (PS) is equal to the speed of light, $c = 3 \times 10^8$.

- **Processing Constraints**

For fog nodes to process user tasks, application modules in which these tasks are processed should be placed at fog nodes. To ensure the placement of these application modules, application modules require CPU, Ram, and Bandwidth capacity so that fog nodes will have enough CPU, Ram and Bandwidth capacity to place these application modules, thus processing end-user tasks at the fog paradigm.

$$\sum_{i=0}^{M} Required_{Capacity}\, m_i \leq \sum Available_{Capacity}\, f;\ \forall m_i \in \mathrm{M}, \forall f \in \mathrm{FN} \quad (3.4)$$

Where $Required_{Capacity}$ for each application module = {CPU, Ram, Bandwidth} and the fog node capacity = {CPU, Ram, Bandwidth}. Constraint (3.4) ensures that the total required capacity of all application modules should not exceed the available capacity of the fog node in which they should be placed. In iFogSim, if the capacity required to place application modules exceeds the available capacity of fog nodes, the system will iterate through upper tiers fog computing system until it reaches the cloud and places these application modules. The CPU required for an application module is calculated as following:

$$CPU = NV * (Rate * TaskCPU) \quad (3.5)$$

Where NV is the total number of connected vehicles to a fog node, and TaskCPU is the task CPU length which is the number of instructions contained in each task in Million Instructions Per Second (MIPS). Rate is calculated as:

$$Rate = \frac{1}{Transmission\ Time\ in\ ms} \quad (3.6)$$

The placement of application modules in iFogSim is done before running the system and starting the emission of tasks. If the number of vehicles increases, this will impact the required CPU capacity for an application module. In this case the number of connected vehicles for each fog node is limited as constraint (3.7).

$$\sum_{i=0}^{V} v_i\, f_j \leq\ MAX_{vehicle\ number};\ \forall\ v_i \in V, \forall\ f_j \in\ FN \quad (3.7)$$

Connection between Fog nodes

This section describes the waiting queue for fog nodes in which the offloading decision is determined, how fog nodes communicate and the selection criteria for the best neighbouring fog node.

- **Fog nodes' waiting queue.**

Each fog node maintains a waiting queue into which tasks are placed upon their arrival at the fog node. Fog nodes process one task at a time. Once the execution of that task is completed the fog node will check its waiting queue and process the next task according to its scheduling policy, i.e., first come, first served. This process continues until no tasks are in the waiting queue. Following the work [45], the waiting queue time triggers the decision to start computational offloading to neighbouring fog nodes. To start sharing workloads, the queue waiting time ($\boldsymbol{T^{Queue}}$) should exceed the offloading threshold, e.g., 50ms, 100ms or 200ms.

$$T^{Queue} > Max_{threshold} \tag{3.8}$$

T^{Queue} is calculated as

$$T^{Queue} = \sum T_i * T_i^{process} + \sum T_z * T_z^{process};\ \forall\, i, z \in T \tag{3.9}$$

Where T_i and T_z are the total number of tasks of the type *i* and *z*, e.g., priority or non-priority. T is all tasks and $T_i^{process}$ is the expected execution time of a specific task and calculated as

$$T^{process} = \frac{TaskCPU}{F\_MIPS * N\ of\ PS} \tag{3.10}$$

Where F_MIPS is the total mips available in a fog node and N of PS is the total number of processing units allocated in that fog node.

- **Coverage Method**

To achieve area coverage, several fog nodes are required. Fogs can also overlap to achieve maximum coverage as in [84] see Figure 3.3.

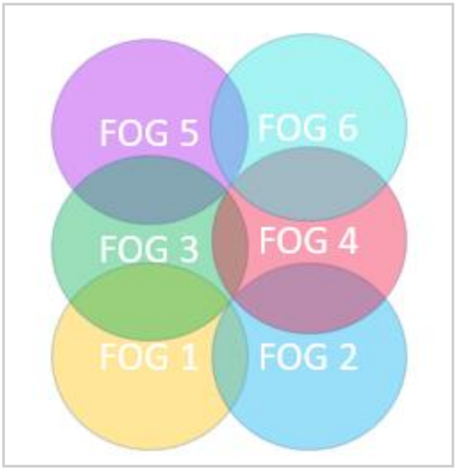


Figure 3.3: Overlapping Fog Nodes.

- **Selecting the Best Neighbouring Fog Node**

The process of selecting the best neighbouring fog node follows the work in [45]. It happens when a fog node reaches its offloading threshold, e.g., 50ms, 100ms or 200ms waiting queue time, for each upcoming task that is generated from vehicles in the coverage range of this fog node. The neighbouring fog nodes of a fog node are the fog nodes that are located within the coverage radius of the fog node itself. This is shown in constraint (3.11)

$$d_{ij} \leq \ Coverage_{radius}; \ \forall \ i,j \ \in \ FN \tag{3.11}$$

Where d_{ij} is the distance between fog nodes i and j. In Figure 3.3, FOG 2 and FOG 3 are the neighbouring fog nodes for FOG 1. Also, FOG 1, FOG 4, FOG 5 are the neighbouring fog nodes for FOG 3. The criteria for selecting the best neighbouring fog node depends on two factors. First, the neighbouring fog node should be within the communication range of the primary fog node. Second, and most importantly, a neighbouring fog node should have the minimum sum of waiting queue time plus propagation delay amongst all available neighbours.

$$Min \ \sum T^{Queue} + PD \tag{3.12}$$

PD is calculated as

$$PD \ = \frac{D_{f,f'}}{PS} \tag{3.13}$$

($D_{f,f'}$) is the distance between fog nodes *f* and *f'*, and it is calculated similar the distance between a vehicle and a fog node ($D_{v,f}$) and propagation speed PS is equal to the speed of light, its value 3×10^8, this done similar to the work in [83].

Between Fog Nodes and the Cloud

When fog nodes finish the processing of non-urgent tasks the results are sent to the cloud for further analysis and processing by the application module named Global Road Monitor. In the current work, the cloud is the least to be considered in sharing the workload of fog nodes when they reach the offloading threshold. This is due to the availability of neighbouring fog nodes and in order to get maximum usage of the available resources in the fog system. However, if all neighbours reach their offloading threshold, the primary fog node will determine to send the task to the cloud if its queue waiting time is higher than transmission delay caused by sending the task for processing to the cloud and getting the results back. Due to the powerful computational capabilities at the cloud server compared to fog nodes, queueing delay is neglected so tasks are processed upon their arrival [85-87] .

3.3 Problem Formulation

The optimisation problem of minimising the delay and the energy consumption has been decomposed into two sub-problems: the delay minimisation problem and the energy saving problem.

3.3.1 Delay Minimization Problem

The response time includes the round-trip time for transmitting the workload between a user and the associated fog node. It includes the transmission delay, propagation delay, queuing delay and processing delay. If the workload is processed by the vehicle's primary fog node, then the service latency is calculated as

$$T = T^{sTv} + 2\ X\ (T_{vTf}^{Transmision} + PD_{vTf}) + T^{Queue} + T^{procss} + T^{vTa} \qquad (3.14)$$

Where T^{sTv} and T^{vTa} is the latency time between a vehicle and its sensor, and between the vehicle and its actuator, respectively. $T_{vTf}^{Transmision}$ is transmission delay between the vehicle and its primary fog node. It is based on the network length of the task and the bandwidth, and it is calculated as

$$T^{Transmision} = \frac{Network\ Length\ of\ Task}{Bandwidth} \tag{3.15}$$

If the primary fog node decides to offload the workload to one of its neighbours, then the latency is calculated as

$$T = T^{sTv} + 2\,x\left(T_{vTf}^{Transmsiion} + PD_{vTf}\right) + 2\,x\left(T_{fTf}^{Transmission} + PD_{fTf}\right) + T^{Queue} + T^{Process} + T^{vTa} \tag{3.16}$$

If the primary fog node decides to send the task to the cloud, then the latency is calculated as

$$T = T^{sTv} + 2\,x\,(T_{vTf}^{Transmsiion} + PD_{vTf}) + 2\,x\,(T_{fTc}^{Transmission}) + T^{Process} + T^{vTa} \tag{3.17}$$

In our system, fog nodes are overlapped in order to achieve full converge in the area. Regarding vehicles, we don’t consider the mobility aspect in our work, and we assume vehicles send requests and receive responds while they still in the coverage range of the primary fog node. Vehicles are randomly distributed and connected to the nearest fog node as long as the maximum number of connected vehicles for that fog node is not reached.

In regard to the influence of the proposed type of connections and constraints that affect the delay formulation, we have the following: in terms of connections between vehicles and fog nodes, the delay associated with this constraint is between 1 ms and 5 ms depending on the distance between the vehicle and the primary fog node. This is shown in Table 3.5. For the processing constraint, to ensure the meeting of this constraint, we limited the number of connected vehicles to each fog node. This is done to ensure the placement of the required application modules offline before running the application to process different types of requests. As increasing the number of connected vehicles, will increase the amount of required resources to place each application module. In this case, the delay is not influenced by this constraint.

3.3.2 Energy Saving Problem

By minimising the power consumption of fog nodes, the overall cost of electricity consumption and environmental impact is reduced. In the system, the energy spent by fog nodes is correlated to the state of the fog node as in [88-90]. There are five states of energy consumption; when processing tasks, transmitting tasks, receiving tasks, being idle, and powering on a previously switched-off fog node. So, the total energy spent by fog node i during the processing of tasks stage is calculated as

$$E_e^i = \sum_{x=1}^{X} T_x^{process} * e_e \tag{3.18}$$

Where X is the total tasks processed in fog node i during the system, and $T_x^{process}$ is the time required to process a single task x, e_e is the unit of energy spent during the time of processing. For the transmission of tasks to neighbours and receiving of tasks from neighbours, the energy consumption associated is calculated as

$$E_{tr}^i = \sum_{z=1}^{Z} T_z^{Transmision} * e_{tr} \tag{3.19}$$

$$E_{re}^i = \sum_{w=1}^{W} T_w^{rec} * e_{re} \tag{3.20}$$

Where Z is the total number of tasks offloaded and W is the total number of received tasks after processing from neighbours, e_{tr} and e_{re} are the units of energy spent during the offloading and receiving of tasks. The energy spent during the idle state is calculated as

$$E_{idle}^i = T_i^{idle} * e_{idle} \tag{3.21}$$

The energy overhead associated with powering on fog node i if it was switched off previously is calculated as

$$E_{on}^i = T^{on} * e_{on} \tag{3.22}$$

So, the total energy spent by fog node is calculated as

$$E_i = E_e^i + E_{tr}^i + E_{re}^i + E_{idle}^i + E_{on}^i$$

Then, the total energy consumption in the system is defined as

$$E = \sum_{i \in FN} E_i$$

In regard to the influence of the proposed type of connections and constraints that affect the energy saving, we have the following: in terms of connections between vehicles and fog nodes, if the connected vehicle is far, more energy is consumed due to the long transmission delay. The energy required to transmit a single task is shown in equation (3.19).

The problem of minimizing delay and energy is formulated as follows:

$$\text{Minimise} \quad \sum_{x \in \boldsymbol{TA}} \boldsymbol{T} + \sum_{j \in \boldsymbol{FN}} \boldsymbol{E} \tag{3.23}$$

Subject to Equations (3.1), (3.4), (3.7), (3.11)

C1: $T_i^{Queue} \leq Max_{threshold} \; \forall \, i \in \boldsymbol{FN}$

C2: $P_F + P_N = 1, P_F \,\&\, P_N = \{0, 1\}$

Equation (3.1) ensures the connection between a fog node and a vehicle that is located within its communication range. Constraint (3.4) ensures the placement of application modules at fog nodes. Equation (3.7) ensures the number of vehicles connected to one fog node does not exceed the threshold number. Constraint C1 ensures the stability of fog nodes' queues so that, to process its upcoming tasks, the waiting queue time should not exceed its threshold. In constraint C2, PF and PN mean that if the task is processed in its primary fog node, then PF = 1 and PN = 0 and vice versa. Therefore, the task is either processed in the primary fog node or one of its neighbours.

3.4 Proposed Algorithms

An approach that combines two algorithms has been proposed to solve the above stated problem. The first algorithm is called dynamic task allocation and the second is called dynamic resource saving. In this paper, both stated algorithms need to work together to achieve the intended outcome.

3.4.1 Dynamic Task Scheduling (DTS):

The aim of this algorithm is to minimise delay by allowing cooperation between fog nodes in terms of workload sharing, to maximise the resource utilization and maximise throughput. The fog controller is not involved in the selection of the best neighbouring fog node, it is mainly involved in the DEC algorithm. Also, in regard to DTS algorithm, if the best neighbour is switched OFF, the fog controller will send a signal to switch ON the selected best neighbour, this is further explained in section 5.2.

The process of offloading a task based on the queue waiting time of the fog nodes was originally proposed by [45]. In [45], the task can be offloaded multiple times, which means that if the primary fog node decides to offload the upcoming task to its neighbour *i*, by the time this task arrives at fog node *i*, fog node *i* might have reached its offloading threshold. Then fog node *i* will select fog node *j* to offload this task to, resulting in offloading this task multiple times and adding additional transmission and propagation delay. As stated by [45], multiple task offloading will increase the delay compared to only allowing the task to be offloaded one time, and this is applied to the current work. The technique is shown in Figure 3.4.

When a fog node receives a task, if this task is the first task in its queue it will immediately process it, if not, it will check its queue waiting time. If its queue did not reach its offloading threshold, e.g., 50ms, 100ms or 200ms, the task will be added to its queue, but if the queue reaches its threshold the fog node will check if the task has been offloaded by another fog node. If it has, then it will add this task to its queue. If it has not been offloaded by another fog node it will select the best neighbour to offload this task to, according to the criteria described in section 3.1.2. If the best neighbour reaches its offloading threshold during the selection process and the task of type priority, then the primary fog node will process the task locally.

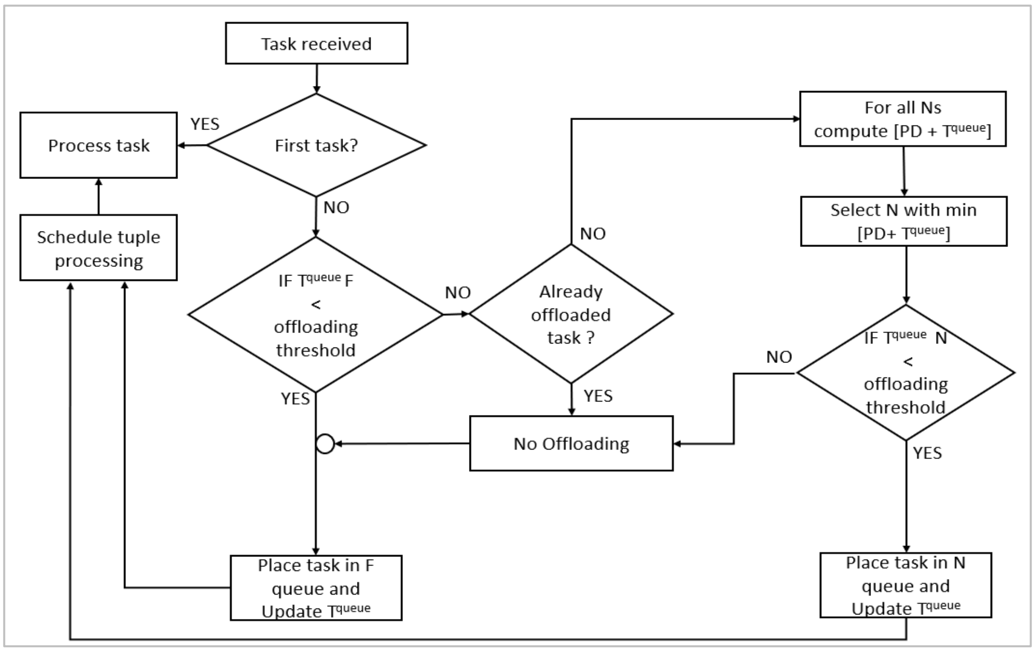

Task received
First task?
YES
Process task
NO
Schedule tuple processing
IF T^queue F < offloading threshold
NO
Already offloaded task ?
NO
For all Ns compute [PD + T^queue]
Select N with min [PD+ T^queue]
IF T^queue N < offloading threshold
NO
YES
YES
No Offloading
YES
Place task in F queue and Update T^queue
Place task in N queue and Update T^queue

Figure 3.4: Flowchart of Dynamic Task Scheduling Algorithm.

3.4.2 Dynamic Energy Control (DEC)

The need for 24/7 availability of fog nodes poses a challenge on energy efficiency and cost since the fog provider needs to maintain available resources that may be used but are not continuously needed. If a fog node is not needed it should be turned off to save energy. Dynamic energy control (DEC) has been proposed in order to optimise resource utilisation by dynamically deciding when to switch off an active fog node(s) and conserve overall system energy. The pseudo code of our proposed algorithm is given in Algorithm 3.1.

In this system, the ON and OFF switching of fog nodes is carried out by the fog controller which runs algorithm 3.1 each time it receives information about the system. Fog nodes update the fog controller with their information so that fog controller can make the appropriate decision to save energy. At the beginning of the simulation, all fog nodes are switched OFF. This happens each 10 ms, fog nodes send their information to the local controller. This information will help the local controller determine whether each fog node should be turned off or on based on its status. The priority in our proposed scheme is to minimise the average delay in the system, so fog nodes only switched off is there is no tasks waiting in its queue, in the case where the system is congested, fog nodes will stay powered on to process as much tasks as required, unless it notifies local controller about its status which specifies that its queue is empty and it is in its idle state.

Algorithm 3.1: Dynamic Energy Control.

```
Dynamic Energy Control Algorithm
Input: System Data: 1- current waiting time; 2- current processing states; 3-
       if awaiting task/s
Output: Sending signals to switch ON/OFF determined FNs
 1:  Fog Controller receives System data
 2:  for all FNs do:
 3:    if (FN. status ==OFF)
 4:      if (FNQueueSize! = 0)
 5:        Send Signal ON
 6:      else
 7:    else
 8:      if (processingStatus =1) //fog node is not processing task/s
 9:        Send Signal OFF
10:      else
11:    end if
12:  end for
```

Algorithms DTS and DEC are not independent. DTS runs by each primary fog node to decide whether to process its arrival task/s locally or send them to the best neighbour. If the best neighbour is switched off, the primary fog node will notify the local controller, and the task will be sent to the switched-off fog node and placed in its queue. The local controller will check if the time since switching off this neighbour has exceeded the minimum allowed time of 5 ms. If it has, the fog node will be switched on; otherwise, it will remain off, and the tasks in its queue will wait. In DEC, local controller receives information from fog nodes each 10 ms. Then, based on this information, the local controller will determine the states of each fog node in regard to being off or on. Thus, the fog node is switched off by local controller through the use of DEC algorithm, and then switched on by the notification of primary fog node to the local controller through the use of DTS algorithm.

3.5 Performance Evaluation

In this section, we first provide the details of the simulations, then we investigate the performance of our two combined algorithms.

3.5.1 Simulation Environment Settings

iFogSim has been used to simulate the environment. It is a toolkit developed by Gupta et. al [91], which is an extension of the CloudSim simulator. It is a toolkit allowing the modelling and simulation of IoT and fog environments and is capable of monitoring various performance parameters, such as energy consumption, latency, response time, cost, etc. For this research, the three-tier fog system was established first as shown by the simulation in Figure. 1. The simulation was run with one cloud server, seven fog nodes, the fog controller, and a total of 50 vehicles. Two fog nodes connected to 25 vehicles each, but the other five fog nodes are not connected to any vehicles and are willing to help with the processing of the offloaded tasks. Each vehicle transmits two different tasks every 3ms. The parameter values used in the simulation is in Tables 3.1 – 3.5.

Table 3.1: Energy consumption parameters

Energy Consumption	Parameter value
Idle power e_{idle}	0.01 W [90]
Processing power e_e	0.9 W [90]
Transmitting and receiving power e_{tr}, e_{re}	1.3 W and 1.1 W [90]
Power overhead for activating a switched off FN e_{on}	0.002 W

Table 3.2: Application Modules Requirements.

Module	CPU (mips/vehicle)	BW (Mbps)	Ram (GB)
Process priority task	333.33	1000	10
Road Monitor	300	1000	10
Global Road Monitor	99.99	1000	10

Table 3.3: Tasks details.

Task Type	Processed module	CPU length (MIPS)	Network Length (Mbps)
Request (urgent)	Process priority task	1000	1000
Sensor (nonurgent)	Road Monitor	900	500
Statistical traffic data	Global Road Monitor	300	500

Table 3.4: Entity Configurations in iFogSim.

Characteristics	Vehicle	Fog nodes	Cloud servers
CPU (MIPS)	0.0	15100	448000
RAM (MB)	0	40000	40000
Uplink BW (Mbps)	1000	1000000	1000000
Downlink BW (Mbps)	1000	1000000	1000000
Rate Per MIPS	0.0	0.001	0.01
Level	2	1	0

Table 3.5: Latency values between entities.

Between		Link latency (ms)
Cloud	Fog node	100 ms
Fog node	Neighbouring FN	2 ms
Vehicle	Fog node	[1-5] depends on location
Sensor/Actuator	Vehicle	1 ms

3.5.2 Performance Metrics

The metrices used to measure the performance are:

• ***Service latency*** is the average round trip time for all tasks processed in the fog environment. Two control loops are used in the simulation:

Control loop A: *Sensor -> Process Priority Tasks -> Actuator*. This control loop represents the path of priority requests.

Control loop B: *Sensor -> Road Monitor -> Global Road Monitor*. This control loop represents the path of non-priority requests.

• ***Throughput***, which is measured as the percentage of processed tasks within a time window and calculates as follows.

$$Throughput\ \% = \frac{total\ number\ of\ processed\ \ tasks}{total\ number\ of\ generated\ \ tasks} * 100$$

- ***Total Energy Consumption*** in fog environment caused by powering on fog nodes, processing, and offloading tasks.

3.6 Offloading thresholds

In this section, we discuss the use of static offloading thresholds and dynamic offloading thresholds and their impact on the system. In the static offloading thresholds, we examine the effects of varying the threshold value in the system and increasing the number of neighbours. Within the context of dynamic offloading thresholds, we explore the system model that deploys dynamic thresholds and highlight the distinctions between static and dynamic offloading thresholds. Furthermore, we present a set of benchmarks within the system, and analyse the consequences of increasing the number of neighbours and vehicles.

3.6.1 Static offloading thresholds

Static offloading means that offloading thresholds for fog nodes remain the same during the experiment.

3.6.1.1 Experiments

The conducted experiments are shown in Table 3.6.

Table 3.6: Set of Conducted Experiments Details.

EXPERIMENT		DYNAMIC TASK SCHEDULING		DYNAMIC ENERGY CONTROL
NO	Name	Yes/No	When	
1	No offloading	no	-	No
2		no	-	Yes
3	Offloading-10	yes	10 ms	No
4		yes	10 ms	Yes
5	Offloading-30	yes	30 ms	No
6		yes	30 ms	Yes
7	Offloading-50	yes	50 ms	No
8		yes	50 ms	Yes
9	Offloading-100	yes	100 ms	No
10		yes	100 ms	Yes
11	Offloading-200	yes	200 ms	No
12		yes	200 ms	Yes

3.6.1.2 Experiment's structure

The structure of the conducted experiments is shown in Figure 3.5.

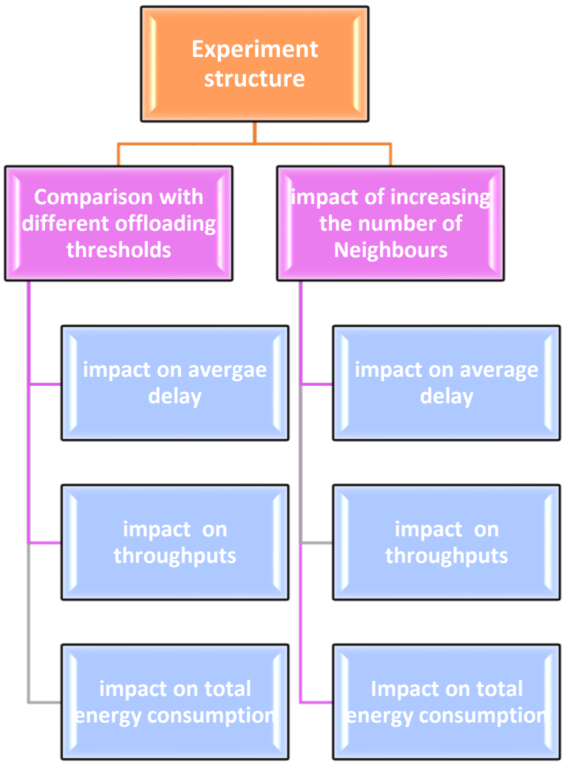

Experiment structure
Comparison with different offloading thresholds
impact of increasing the number of Neighbours
impact on avergae delay
impact on throughputs
impact on total energy consumption
impact on average delay
impact on throughputs
Impact on total energy consumption

Figure 3.5: experiments structure.

3.6.1.3 Average round trip time

There are two control loops in the simulation:

• Sensor → Process Priority Tasks → Actuator. This control loop represents the path of the priority requests, and it is called Control loop A.

• Sensor → Road Monitor → Global Road Monitor. This control loop represents the path of the non-priority requests, and it is called Control loop B.

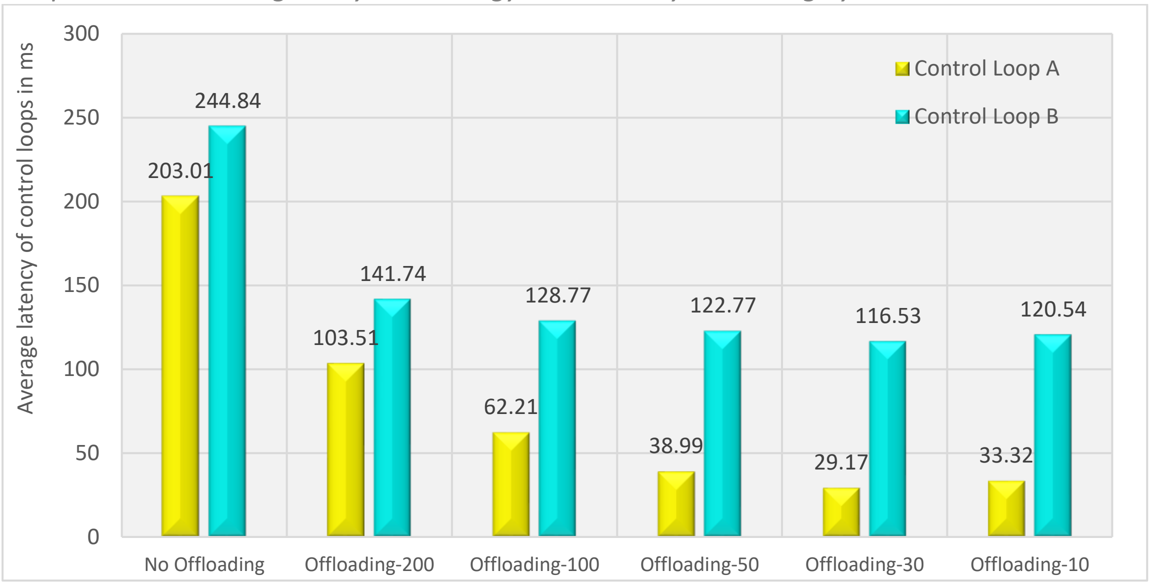


Figure 3.6: Average Round Trip Time with no-offloading and different Offloading Thresholds

The aim here is to minimise the average round-trip time for control loop A, in which the result is sent back to the users, compared to control loop B, in which the user tasks should be processed at fog nodes and the results sent to the cloud for further analysis and storage. The results in Figure 3.6 show that when a fog node is not offloading its tasks to neighbouring fog nodes, the average round trip time for all the processed tasks for control loop A is 203.01 ms. This is due to the long queueing delay.

Also, it can be seen that as we decrease the offloading threshold, the average delay is decreased accordingly. However, if the offloading threshold is decreased by more than 30 ms, the results seem to have a negative impact. Based on the proposed algorithm, when the offloading threshold is set to 10 ms, the primary fog node will share its workload earlier than the 30-ms offloading threshold. As a result, there will be congestion, however, as neighbours will reach the 10-ms waiting time faster. As a result, the primary fog node will not share more workload, but instead will schedule the task to be processed locally.

Accordingly, it can be said that the optimal offloading threshold in the system is influenced by the amount of received workloads. In this experiment, the 30-ms offloading threshold seems to be the optimal value that helps to distribute workload among the computing devices in the system. This is because the lowest average latency is achieved when the offloading threshold is set to 30 ms. With a 30-ms threshold, the average latency of control loop A was reduced by 85.5% compared to the no-offloading method.

3.6.1.4 Throughput Evaluation

According to the results in Figure 3.7, the poorest percentage of executed tasks is when no offloading is involved. Additionally, it can be seen that similar to the impact on the average delay, as we decrease the offloading threshold, the percentage of executed tasks is improved until reaching the optimal offloading threshold, which is 30 ms, upon which the percentage of executed tasks decreases. Because of the 10-ms offloading threshold, neighbours will get congestion faster, resulting in the primary fog node scheduling tasks locally. This will result in higher queueing delay in the primary fog nodes and thus processing less tasks compared to the 30-ms offloading threshold.

The percentage of executed tasks is increased by around 40% when the 30-ms offloading threshold is considered compared to the no-offloading method. As with the no-offloading method, many tasks are waiting to be executed in the queue compared to when the offloading threshold is set to 30 ms.

Furthermore, the highest percentage of processed tasks occurs when the offloading threshold is set to 30 ms. This is followed by the offloading thresholds of 50 ms, 10 ms, 100 ms and 200 ms, respectively, at 94.56%, 94.1%, 91.52%, and 88.50%. Comparing the offloading threshold values, the 200-ms threshold has the lowest processing percentage. This is because fog nodes will wait until the waiting time of their queues has exceeded 200 ms in order to start sharing their tasks.

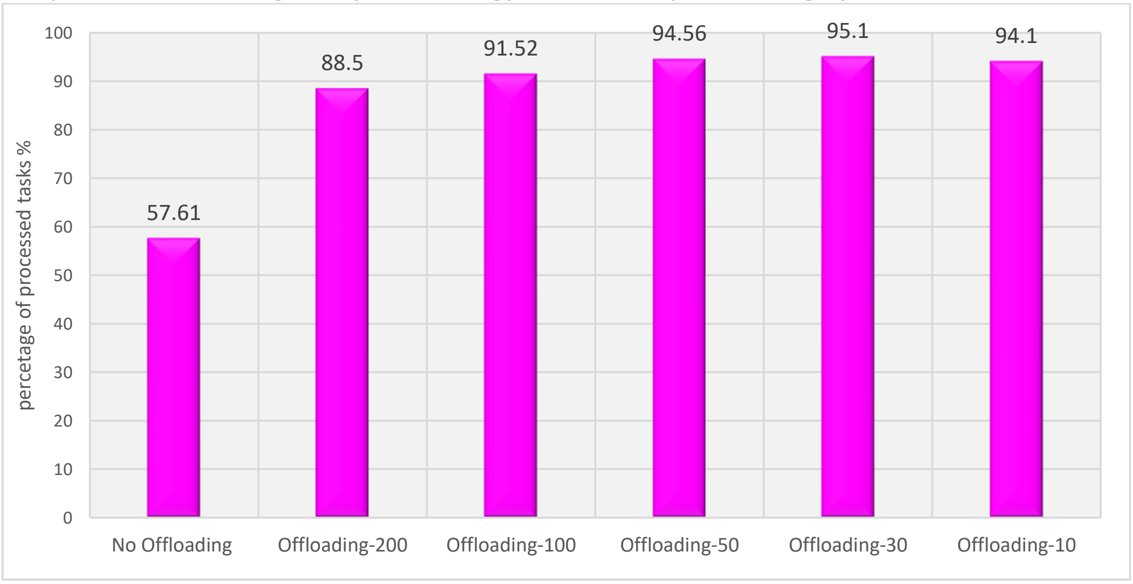


Figure 3.7: Percentage of executed Tasks in Fog Nodes with no offloading and different Offloading Thresholds

3.6.1.5 Total Energy Consumption

In cases where a dynamic energy control algorithm is not applied the highest energy consumption in the fog environment occurs when the offloading threshold is set to 30 ms. This is because more fog nodes are involved in the execution process and are in their busy power mode. This compares to the no-offloading method where only two fog nodes are busy processing tasks while the rest of the fog nodes are not doing any processing and are in their idle power mode (see Figure 3.8). In the no-offloading method, DEC saves around 75% of power. This power was spent powering on unused fog nodes, which cause a wastage in resources.

Applying the DEC algorithm helps to minimise the total energy consumed in the fog environment by around 27.2%, 28%, 28.3%, 30.2%, and 31.1% with the various offloading thresholds of 30 ms, 50 ms, 10 ms, 100 ms and 200 ms, respectively. The reason for a low energy saving with various offloading thresholds compared to a high energy saving with the no-offloading approach is that the workload of the primary fog nodes is high, thus sharing some of their workloads with their neighbours. As a result, neighbours staying ON most of the time helps to process these tasks.

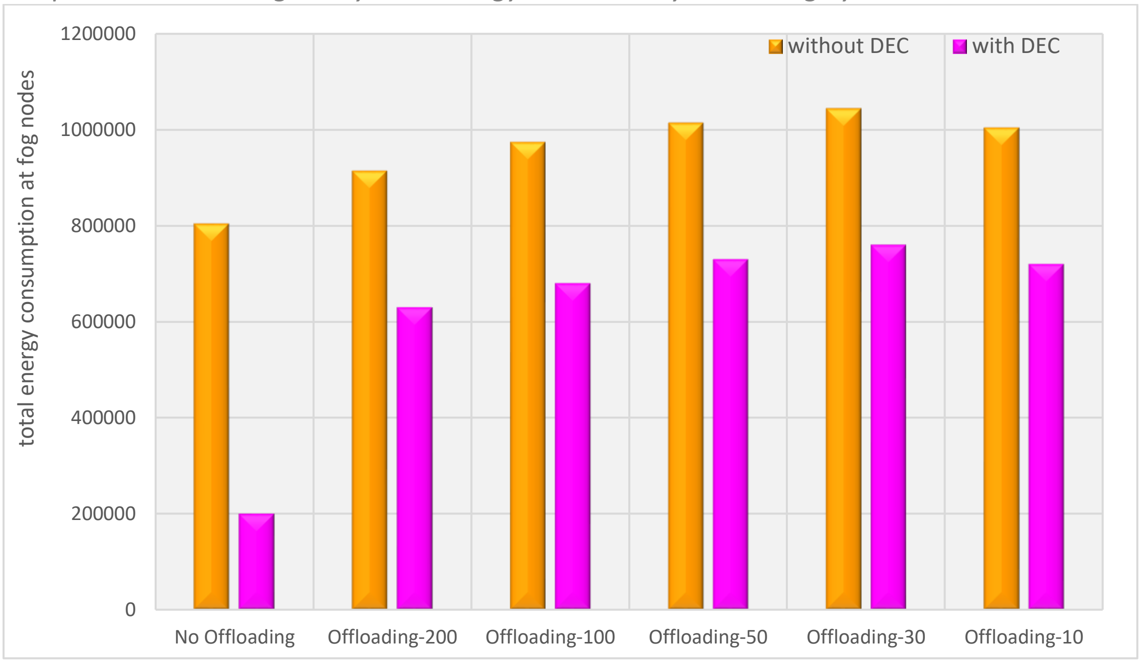


Figure 3.8: Total Energy Consumed in the fog environment with no offloading and different offloading thresholds with and without Dynamic Energy Control (DEC) algorithm.

Overall, lowering the offloading threshold to a certain level improves average delay and throughput, but increases energy consumption. If the offloading threshold is decreased below this level, it will negatively affect these parameters, as in the case of 10-ms offloading thresholds. Regarding total energy consumption, applying the DEC algorithm does help to minimise it. Varying the offloading threshold and testing its impact on the system has not been addressed before in other publications. It does have a significant influence on overall system performance.

3.6.1.6 Impact of increasing the number of neighbours on service latency

Based on the results shown in Figure 3.9, it seems that the impact of increasing the number of neighbours on the offloading thresholds is equal. Increasing the number of neighbours helps to drastically reduce the average latency of control loops. This continues until reaching an optimal number of neighbours at which point the average delay will no longer improve, but instead there is a slight increase. The reason for the slight increase at the end can be attributed to a linear increase in communication overhead as the number of neighbours rose. This is due to sending the status to all neighbours each time the status is changed. The following table demonstrate the impact of increasing the number of neighbours when the offloading threshold is 30 ms.

	Number of neighbours in 30-ma offloading threshold							
	3	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Control Loop A	140	90	38	33	28	22	23.5	25
Control Loop B	210	170	118	113	108	104	103.5	105

At the offloading thresholds of 50 ms, 100 ms and 200 ms, the average delay decreases until having five neighbours in the system. After that, the average latency of control loops remains stable with a slight increase. The reason for the pattern of stability is because the primary fog node processes most of its tasks and only offloads tasks to its neighbours when its offloading threshold reaches its limit (e.g., 50 ms). This is not the case when the offloading threshold is set to 30 ms. In this case, the average delay decreases as we increase the number of neighbours. This increases until there are eight neighbours in the system. As a result, the average delay remains the same with a slight increase. The optimal number of neighbours required for each offloading approach depends on the threshold. Additionally, increasing the number of neighbours in the system more than the required optimal number may impact the system negatively. This is due to increased energy consumption and delays caused by communication overhead.

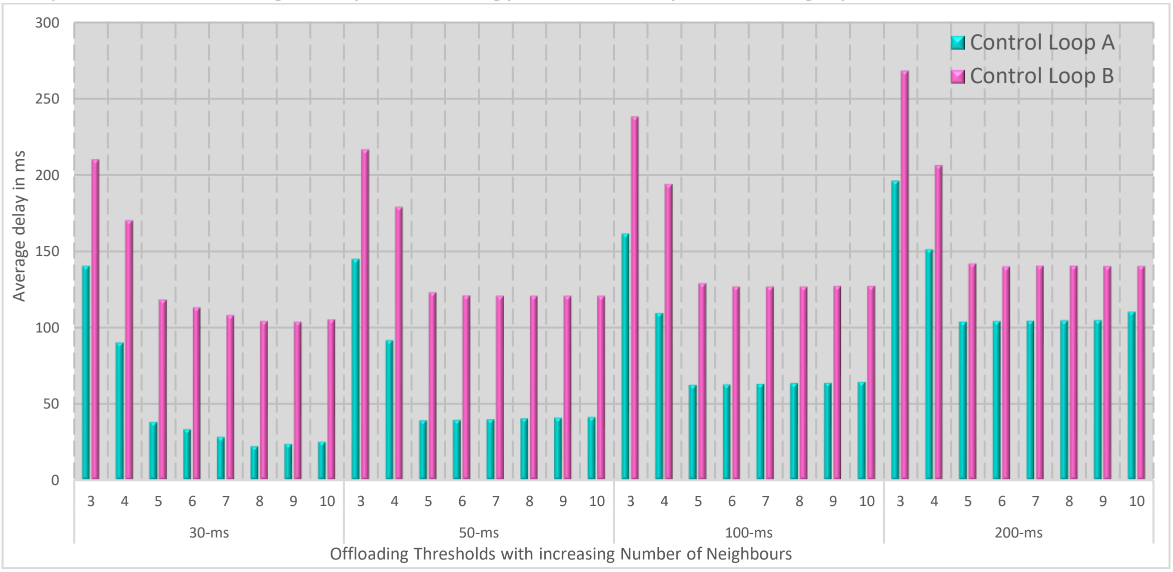


Figure 3.9: Impact of Increasing Number of Neighbours with Various Offloading Threshold on Service Latency

3.6.1.7 Impact of increasing the number of neighbours on throughput

The impact of increasing the number of neighbours with various offloading thresholds on throughputs is shown in Figure 3.10. For all offloading threshold approaches, it can be noticed that as we increase the number of neighbours, the percentage of the executed tasks is increased, until having an optimal number of neighbours, upon which the percentage of executed tasks remains almost the same. The reason for this stability is attributed to the offloading threshold. In general, increasing the number of neighbours gives congested fog nodes more options to begin sharing their tasks. Congested fog nodes, however, may not begin offloading tasks until they reach their threshold for offloading. When the offloading threshold is set too high, e.g., 200 milliseconds, the queue waiting time must reach this level in order for them to begin offloading. Thus, some tasks may still be handled locally, despite having neighbours with low queue waiting times, and the benefits of having more neighbours may not be realised.

Moreover, it can be seen that the highest percentage of processed tasks is achieved when the offloading threshold is set to 30 ms, and for the other approaches, 50 ms, 100 ms and 200 ms, increasing the number of neighbours will not help to achieve similar results to when having a 30-ms offloading threshold. The same reasons mentioned earlier are also responsible for this.

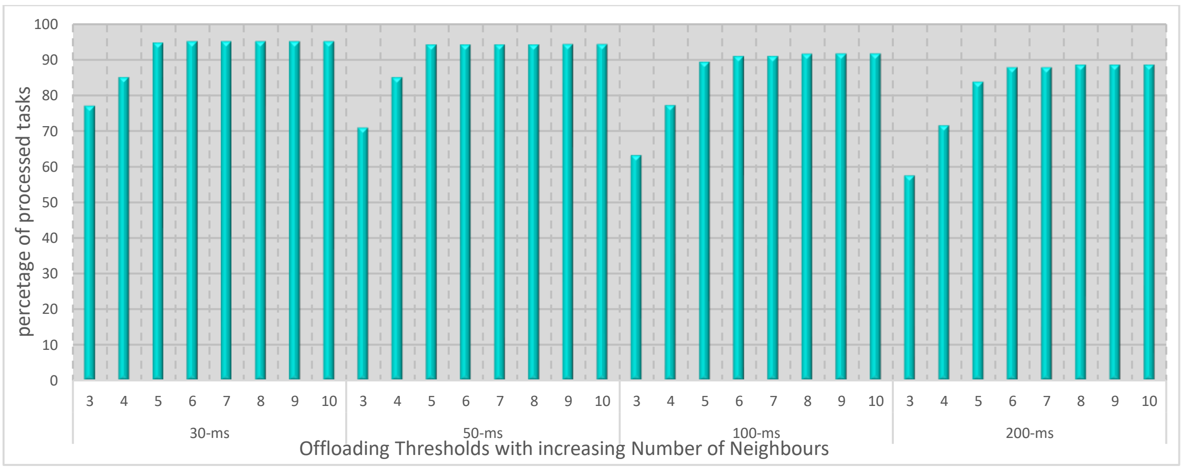


Figure 3.10: Impact of increasing the number of neighbours on throughput with various offloading thresholds.

3.6.1.8 Impact of increasing the number of neighbours on total energy consumption

The results regarding this are shown in Figure 3.11. Overall, it can be seen that without applying the DEC method, as we increase the number of neighbours in the system, overall energy consumption in the system is rising linearly. This will be overcome when applying the DEC method as it saved up to 29.35%, 38.58%, 40.79%, and 42.11% for the 30 ms, 50 ms, 100 ms, and 200 ms offloading thresholds when the number of neighbours is ten. When applying the DEC method, it can be seen that as we expand the number of neighbours, the total energy consumption increases. However, when the optimal number of neighbours is reached, which is five neighbours for threshold 50 ms, 100 ms, and 200 ms and eight neighbours for threshold 30 ms, the total energy consumption will remain unchanged with a slight increase. The reason for the stable pattern in total energy consumption was described earlier.

Furthermore, it can be seen that the highest energy consumption is reached when the offloading threshold is set to 30 ms, followed by the 50-ms offloading threshold, then 100-ms offloading threshold and finally 200 ms. The reason for that is because at the 30-ms offloading threshold, the highest percentage of processed tasks is accomplished and fog nodes and their neighbours are busy processing as many tasks as needed, thus consuming more energy.

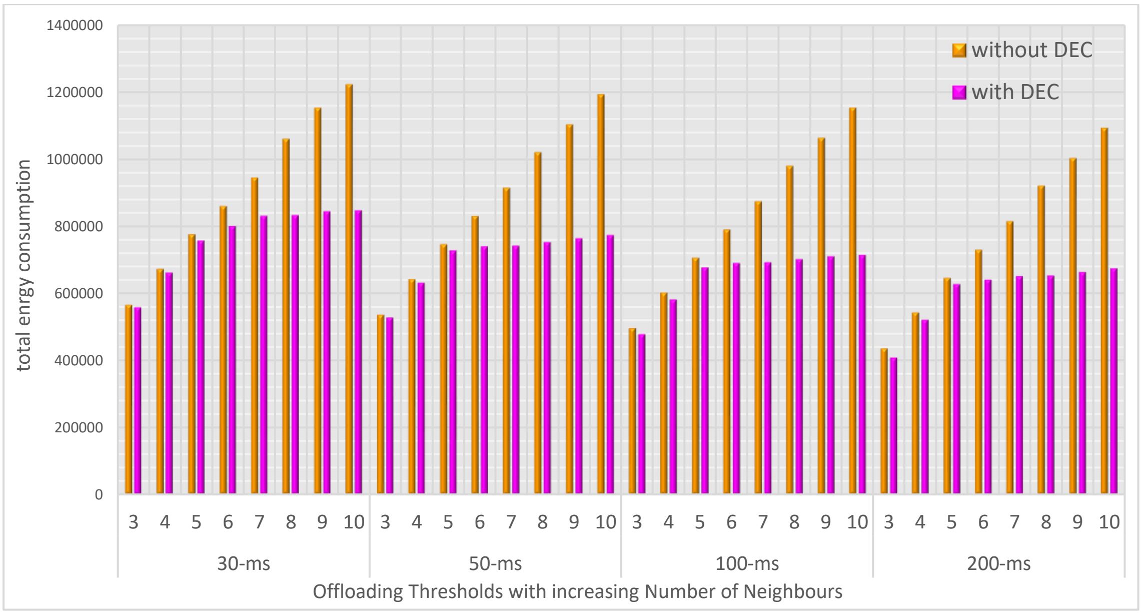


Figure 3.11: Impact of increasing the number of neighbours on throughputs with various offloading thresholds.

Overall, it can be seen that as we increase the number of neighbours, the system will benefit from additional computational resources. This is seen as improving average delay and throughput. However, increasing the number of neighbours more than the optimal number will impact the system negatively in terms of average delay and total energy consumption due to communication overhead. Increasing the number of neighbours will add complexity in the system. In the presence of more neighbours, more messages will be exchanged between them regarding their status updates, consuming more bandwidth and processing power on the network. As a result, there may be bottlenecks and delays in communication between fog nodes, which can negatively impact the performance of the system.

In this case, it is ideal to determine the optimal number of neighbours. However, this is a challenge in dynamic online systems. Additionally, the optimal number of neighbours varies depending on the applied threshold, and the optimum value of the threshold is influenced by the amount of received workload. Energy consumption increases as we decrease offloading thresholds as a result of processing more tasks. As a result of using the DEC method, some energy can be saved.

3.6.2 Dynamic offloading thresholds

3.6.2.1 System Model

In this chapter we apply the same system model and constraints stated above. Moreover, additional changes have been made to the proposed fog node architecture which are described below.

Fog Node Architecture

The proposed fog node architecture consists of a task scheduler, best neighbour selector, and threshold monitor (see Figure 3.12). Task scheduler receives tasks generated from IoT devices within the proximity of the primary fog node and from other neighbouring fog nodes. If a fog node receives a task that is already offloaded from another neighbour, task scheduler immediately inserts this task in the processing queue. If the task is generated from other IoT devices, then task scheduler will check the offloading threshold and compare this to the queuing delay at the current node. If the queueing delay reaches the offloading threshold, then task scheduler sends this task to the best neighbour selection, which in turn decides the best neighbour node to offload this task to.

The selection of the best neighbour is described earlier in section 3.2.2. Threshold monitor is responsible for dynamically increasing and decreasing the offloading threshold for both the primary fog node and all its neighbours, based on the workload and the availability of other neighbours: this is done from the perspective of the primary fog node. On the one hand, it is assumed that fog nodes are cooperative and accept tasks coming from their neighbour nodes, even if this exceeds their threshold.

On the other hand, each neighbour has its own threshold monitor, and the primary fog node and all its neighbours may not have the same threshold value. In Figure 3.13 (a), we can see that primary fog node A set its threshold to 9 ms for itself and all its neighbours. At the same time, primary fog node B in Figure 3.13 (b) set its threshold to 6 ms, even for its neighbour fog node A; therefore, it can be seen that fog node A is congested and will not be selected as the best neighbour for fog node B. Determining when to increase and decrease the offloading threshold is described in Algorithm 3.2.

The term “dynamic” in the context of the offloading scenario refers to the concept that the offloading decision is influenced by the continuous change of the system’s conditions. This encompasses variables such as current workloads on fog devices, available computational resources in the neighbouring devices, and shifting user requirements or priorities. This is opposite to static offloading scenario, where system’s conditions are fixed and predictable. In the offloading scenario, the terms “dynamic“ and “static“ are also referred to as “online“ and “offline” offloading, respectively.

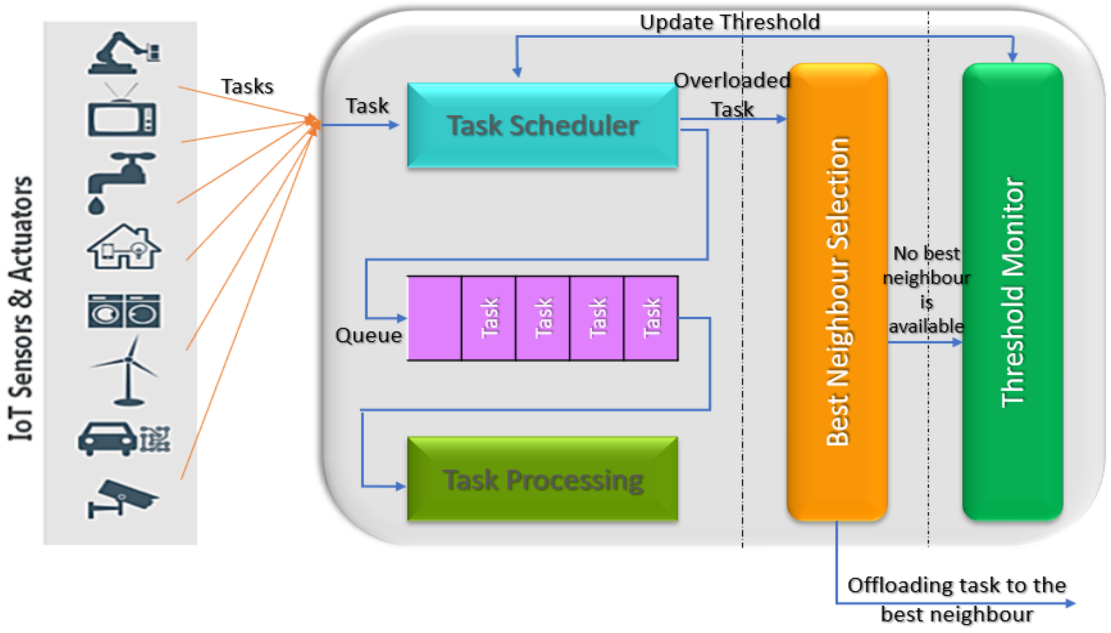
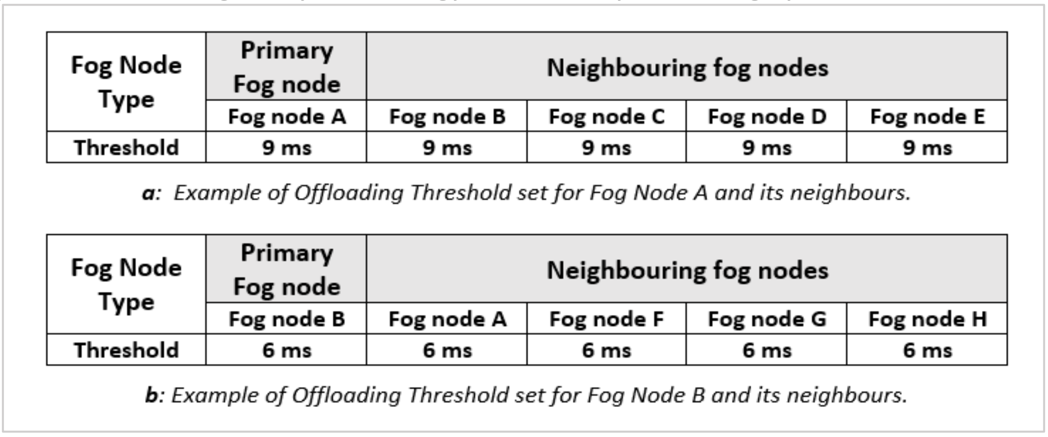


Figure 3.12: Fog Node Architecture Model.



Fog Node Type	Primary Fog node	Neighbouring fog nodes			
	Fog node A	Fog node B	Fog node C	Fog node D	Fog node E
Threshold	9 ms	9 ms	9 ms	9 ms	9 ms

***a**: Example of Offloading Threshold set for Fog Node A and its neighbours.*

Fog Node Type	Primary Fog node	Neighbouring fog nodes			
	Fog node B	Fog node A	Fog node F	Fog node G	Fog node H
Threshold	6 ms	6 ms	6 ms	6 ms	6 ms

***b**: Example of Offloading Threshold set for Fog Node B and its neighbours.*

Figure 3.13: Example of Offloading Threshold values for the Primary Fog Node and all its Neighbours at the same Time.

3.6.2.2 Proposed Dynamic Offloading Threshold

The dynamic threshold is managed by the Threshold Monitor, which adjust its value periodically according to the received workload and the availability of other neighbours, as described in Algorithm 3.2. The first part of the algorithm (Procedure 1) determines whether to increase the offloading threshold of the primary node and its neighbours. This runs each time a new task arrives at the primary fog node. It starts by checking if the current threshold exceeds the maximum offloading threshold calculated in equation (3.24), if this occurs then the best decision for the arrival task is to be migrated to the cloud for processing.

$$\text{Maximum Threshold} = 2 * (Transmission\,{}_{cloud}^{delay}) \tag{3.24}$$

Otherwise, it checks whether the queuing delay of the primary fog node has reached its offloading threshold, i.e., to decide whether to process the task locally and add it to its queue or select the best neighbour with the least queueing delay as per lines 4-16. The current threshold is then updated using equation (4.2) and procedure 2 is called.

$$\delta^{n+1} = \begin{cases} \delta^n - p, & Q \geq \alpha, VQ < x \\ \delta^n, & Q \geq \alpha, VQ \geq x \\ \delta^n, & Q < \alpha \\ \delta^n + p, & Q \geq \delta^n, VQ \geq \delta^n \end{cases} \quad \forall\, x, \alpha, p > 0 \tag{3.25}$$

The parameters involved in equation (3.25) and all parameters used for Dynamic Threshold Algorithm is defined in table 3.7.

The second part of the algorithm (Procedure 2) determines whether the threshold should be decreased. This runs each time a new task is received, and when the fog node finishes the execution of a task. It starts by checking if the current threshold of the primary fog node is larger than a threshold, as per line 25. If this occurs, then the average queueing delay for all the neighbours is calculated as in (3.26) and the current threshold is updated, as per lines 26-27. The computational complexity of the proposed algorithm is O(n).

$$VQ = \frac{\sum_{s=0}^{Ns} Qs}{Ns} \tag{3.26}$$

Table 3.7: Description of Parameters used for Dynamic Threshold Algorithm.

Symbol	Description
δ^n	Refers to the initial offloading threshold and the current threshold.
VQ	Average queueing delay of all the neighbours
δ^{n+1}	New offloading threshold.
x	x = δ^n /2.
Ns	All neighbouring fog nodes
Qs	Set of all queueing delay of all its neighbours
$Q_{Neighbours}$	Set of all neighbours and their queueing delay
α	When the queuing delay reaches this threshold, the fog node might consider decreasing its offloading threshold.
N	The Best neighbour fog node
T	The arrival task
Q	Queuing delay in the primary fog node
p	A number bigger than zero that determines how much to modify the offloading threshold based on the current offloading threshold
Q_N	Queueing delay of one neighbour

Algorithm 3.2: Dynamic Offloading Threshold.

```
Algorithm 1 Dynamic Offloading Threshold
Input:          $\delta^n$, $Q_{Neighbours}=[(N_1,Q_1), (N_2,Q_2), \ldots,(N_M,Q_M)]$, T, Q;
Initialisation  $\delta^{n+1} = \emptyset$, max_threshold = 2*Latency;
Result/s:       $\delta^{n+1}$;
Procedure_1.
    IF ($\delta^n$<max_threshold)
        IF (Q >= $\delta^n$)
            Best ← $N_1$
            min ← $Q_1$
            For each $n \in Q_{Neighbours}$ do
                IF ($Q_i < Q_1$) then
                    Best ← $N_i$
                    min ← $Q_i$
                end if
            end for
            Offload (T,Best)
            $Update(\delta^n)$ using equation 3.25
        Else
            Queue.add(T)
            $Update(\delta^n)$ using equation 3.25
        end if
    else
        migrate to cloud
    end if
    return $\delta^{n+1}$
    Call procesdure_2
end Procedure_1.
Procedure_2.
    IF ($\delta^n$ >= α) then
        Calculate VQ using equation 3.26
        $Update(\delta^n)$ using equation 3.25
    end if
    return $\delta^{n+1}$
end Procedure_2.
```

3.6.2.3 Baseline Approaches

To evaluate the effectiveness of our proposed algorithm, the comparisons with various computation offloading schemes are provided, where the number of vehicles is set to 50 and total number of fog nodes is set to 7. Although we implement uncertainty within the system to mimic real world scenarios, we maintain the number of total generated tasks, the capacity of fog nodes, and the size of the generated tasks to be identical for fair comparison. In particular, the following four schemes are selected as benchmarks:

Benchmark 1**:** *No Offloading Scheme (NO):* in this scheme, each primary fog node processes all the tasks without cooperation with other neighbouring fog nodes.

Benchmark 2**:** *Joint Task Offloading and Resource Allocation Scheme (JTORA)* [40]: in this scheme, if the primary fog node does not have enough computational resources that meet the delay requirement of a task, then the task will be offloaded to a neighbouring fog node within the proximity of the primary fog node that has enough computational resources. Any underutilised neighbour is a candidate of processing the overload, ignoring the selection of the least utilised fog node. In this scheme, a static threshold is applied.

Benchmark 3**:** *Workload Offloading Scheme (WO)* [92]. In their work, end users offload their computational tasks to a broker node that manages the system, the broker node will send tasks to a fog node closest to end users (primary fog node). If the primary fog node is congested (e.g., its queueing delay reaches 50ms), then the broker node will offload the task to any underutilised neighbouring fog node. In this scheme, a static threshold is determined.

Benchmark 4*: Static Threshold 50ms Scheme (ST50):* where offloading threshold is set to 50ms, upon which the primary fog node makes the decision on whether to process the task locally or offload it to the best neighbouring fog node. The four benchmarks are compared to the proposed offloading policy called Dynamic Threshold (DT).

3.6.2.4 Experimental results structure

The structure of the conducted experiments is shown in Figure 3.14.

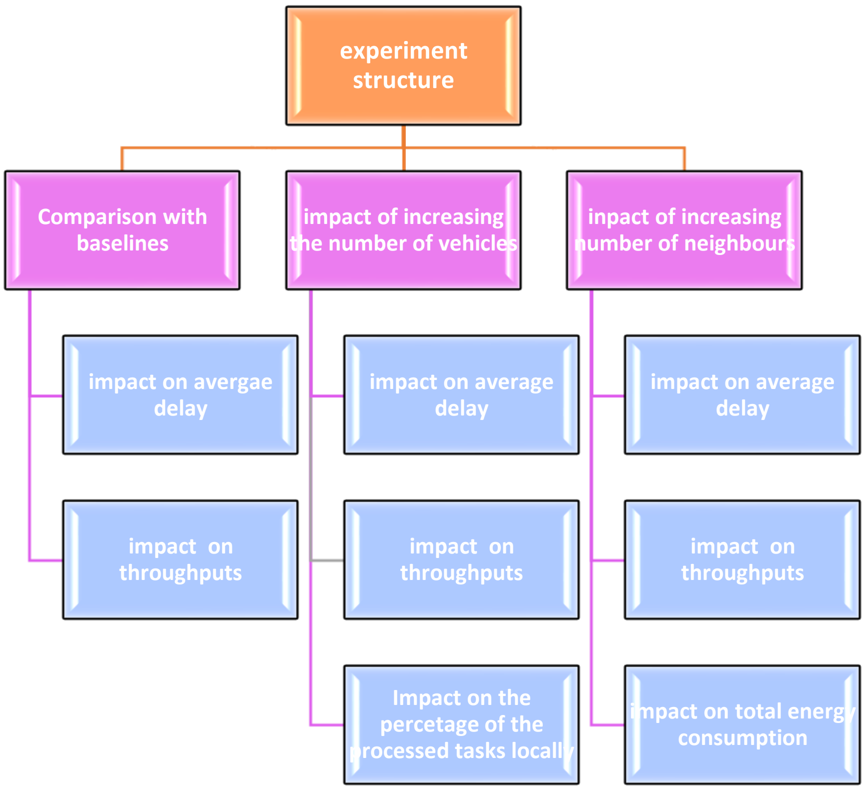

experiment structure
Comparison with baselines
impact of increasing the number of vehicles
inpact of increasing number of neighbours
impact on avergae delay
impact on average delay
impact on average delay
impact on throughputs
impact on throughputs
impact on throughputs
Impact on the percetage of the processed tasks locally
impact on total energy consumption

Figure 3.14: Experimental results structure.

We do not compare the total energy consumption with different baselines and when increasing the number of vehicles. This is because the set of baselines do not consider minimising energy in their work. In regard to addressing the impact of increasing the number of vehicles, in this experiment we compare this impact with the same stated baselines that do not consider energy consumption.

3.6.2.5 Impact of the proposed scheme and different offloading schemes on delay

In Figure 3.15, the impact of various offloading schemes on average latency is addressed. It can be seen that delay is very high in the no offloading scheme; this is due to a long queueing delay as tasks are not shared by the primary node with other neighbouring fog nodes, so they are waiting to be executed by the primary fog node. This is followed by WO and JTORA schemes. This is because in the WO and JTORA schemes, when the primary fog node is congested (e.g., reaching its offloading threshold), it selects any underutilised neighbour to share the workload, rather than selecting the least utilised neighbour, as in ST50 and DT.

Furthermore, the delay is higher in the WO scheme compared to JTORA; this is due to a communication overhead caused by sending tasks to a broker node first, which in turn decides whether to process these tasks; either at the primary fog node or any underutilised neighbour. The least delay is achieved for both control loops when applying our proposed algorithm, DT, compared other benchmarks. Regarding Control Loop A, DT approach helps to minimise the average delay by 55.9%, 70.6%, 72.5%, and 95.4% compared to ST50, JTORA, WO and No Offloading approaches. Moreover, in Control Loop B, DT approach helps to save the average latency by 4.2%, 28.6%, 29.5%, and 74.1% for the same set of benchmarks. Comparing ST50 with DT, DT reduces delay by 4% in Control loop B, and around 56% in Control loop A.

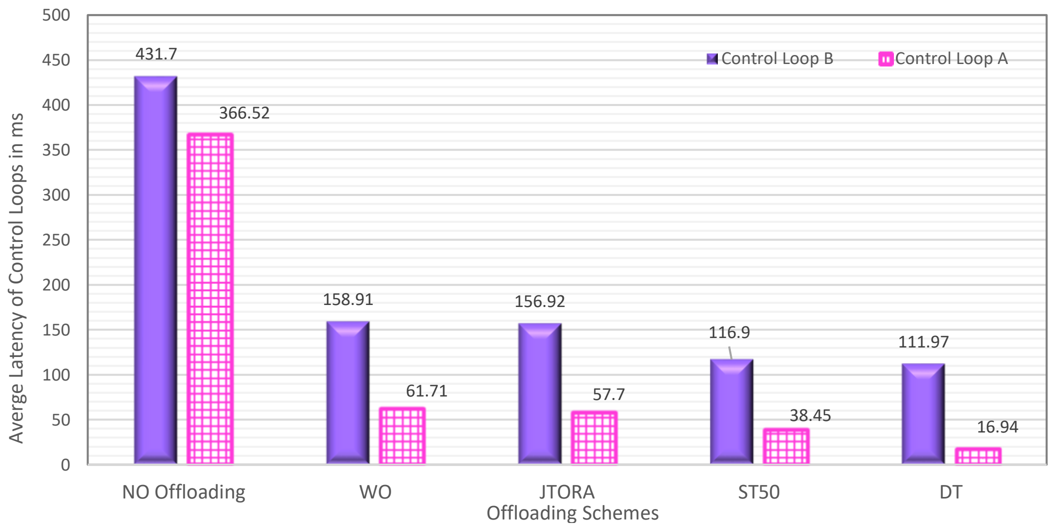


Figure 3.15: The comparison of the average latency with various offloading schemes

3.6.2.6 Impact of the proposed scheme and different offloading schemes on throughputs

The impact of various offloading schemes on throughput is shown in Figure 3.16. It can be seen that the lowest percentage of executed tasks is when no offloading is applied; this is obvious as most of the tasks are waiting in the queue to be executed by the primary fog nodes. Sharing workload with any underutilised neighbours has a strong impact on WO and JTORA.

It results in almost 90.88% and 91.06% of tasks being processed, as opposed to 93.56% and 97.67% with ST50 and DT schemes. The highest percentage of processed tasks is in DT, due to the characteristics of the proposed scheme. When comparing ST50 and DT, there is around 4% improvement in the throughputs. The reason DT does not outperform ST50 with a higher rate is attributed to the fact that the simulation ends while there are still tasks in queues waiting to be executed.

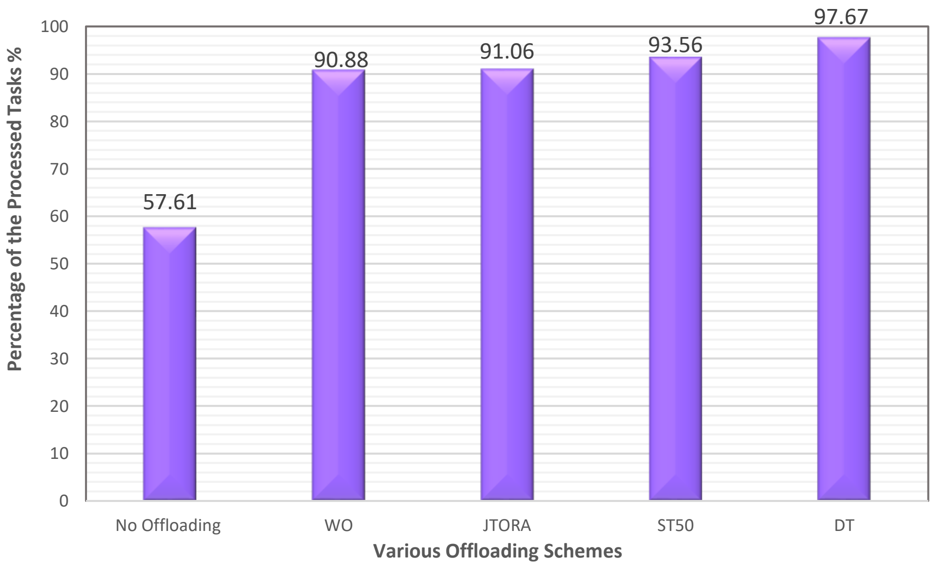


Figure 3.16: The comparison of the throughputs with various offloading schemes.

3.6.2.7 Impact of increasing number of vehicles on delay with different offloading schemes

The impact of increasing the number of vehicles investigated to see how the delay is maintained as we increase the workload in the online system. In this experiment, the total number of fog nodes is set to seven and the number of vehicles ranges from 4 to 48. In Figure 3.17 we can observe that when the number of vehicles is small, between 4 to 12 vehicles, the DT, ST50 and JTORA schemes exhibit an identical pattern. This is because the generated workloads are small, resulting in the primary fog nodes processing most of these workloads themselves. When the number of vehicles increases, all three approaches ST50, JTORA and WO show a dramatic increase in delay compared to DT, which displays a stable pattern with a slight increase in delay that increases as the number of vehicles increased.

The reason for the huge increase in delay for ST50, JTORA and WO is that increasing the workload makes the primary fog nodes almost reach their offloading threshold (e.g., 50ms), but not always exceeding it, resulting in the primary nodes processing most of the workload with little help from neighbouring nodes. The impact of selecting the best neighbour to share the workload becomes clear when the number of vehicles is high (i.e., 28 vehicles). The overall results show the effectiveness of the DT scheme even when increasing the number of vehicles.

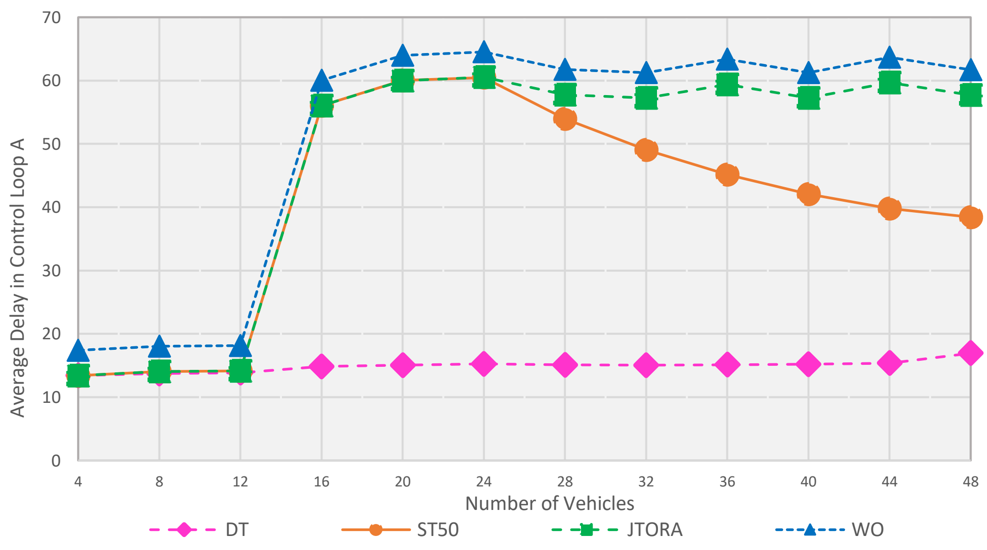


Figure 3.17: Impact of Increasing Number of Vehicles on average delay with Different Offloading Schemes

3.6.2.8 Impact of increasing number of vehicles on throughputs with different offloading schemes

The impact on throughput has also been investigated while increasing the number of vehicles, see Figure 3.18. When there is a small number of vehicles, ranging from 4 to 12, all the offloading schemes operate in a similar way; this is because the workload is minimal and can be processed at the primary fog nodes without using capacity of neighbours. When the number of vehicles is increased, DT achieves the highest throughput, with almost 97.5% compared to other schemes, which accomplish 93.5%, 91% and 90% for ST50, JTORA and WO, respectively. Additionally, it is noticed that when the number of vehicles reaches 24 vehicles, the pattern of all approaches will remain the same even when increasing the number of vehicles. Stability can be attributed to the behaviour of these approaches because this is the highest percentage of tasks they can accomplish. Overall, our approach ST50 helps to improve throughputs compared to a set of benchmarks, and this is overcome by the DT by around 4%.

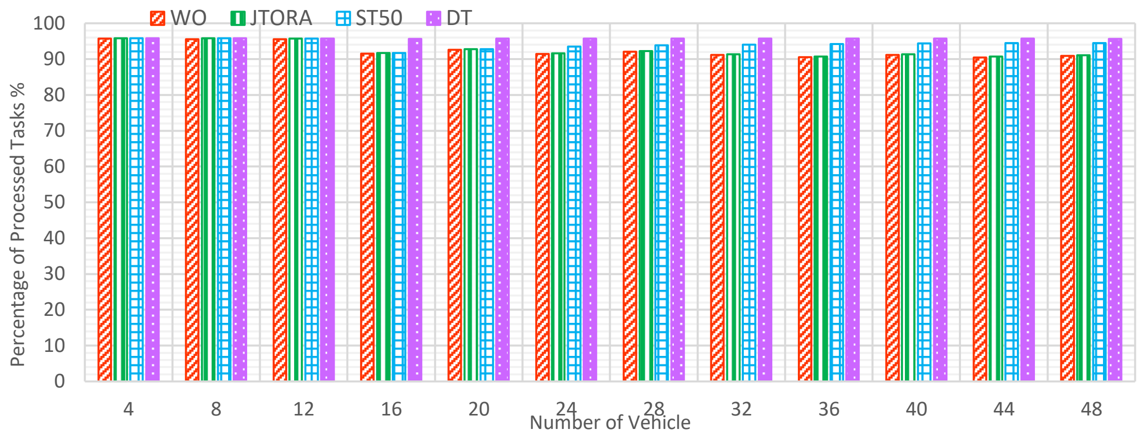


Figure 3.18: Impact of Increasing Number of Vehicles on throughputs with Different Offloading Schemes

3.6.2.9 Impact of increasing number of vehicles on the percentage of the processed tasks locally.

In this experiment, we compare the performance of fixed and dynamic thresholds in regard to the percentage of locally processed tasks. This is to analyse when primary fog nodes will start cooperating with their neighbours as their workload increases. From Figure 3.19, it can be seen that when the number of vehicles is small, e.g., 12, most of the generated workload is processed locally in the fixed threshold approach. This is because the workload is small and the primary fog nodes did not reach their fixed offloading threshold to begin offloading their tasks to their neighbours. However, the dynamic threshold approach starts sharing some of its workload around 3% when the number of vehicles is between 8 and 12.

We notice that both approaches offload their workload to neighbours as the number of vehicles increases, and the percentage of tasks handled locally decreases. However, the dynamic threshold seems to process less of its workload locally compared to the fixed threshold. This is because, at the fixed offloading threshold, primary fog nodes process most of their workloads until the offloading threshold is reached. While at the dynamic threshold, the primary fog node will start sharing its workload once there are available neighbours. This is because the offloading threshold is changing dynamically by not only considering the status of the primary fog node itself, but also the status of all the computing devices in the system. This will help to exploit the available resources in the system more than when the fixed threshold is involved.

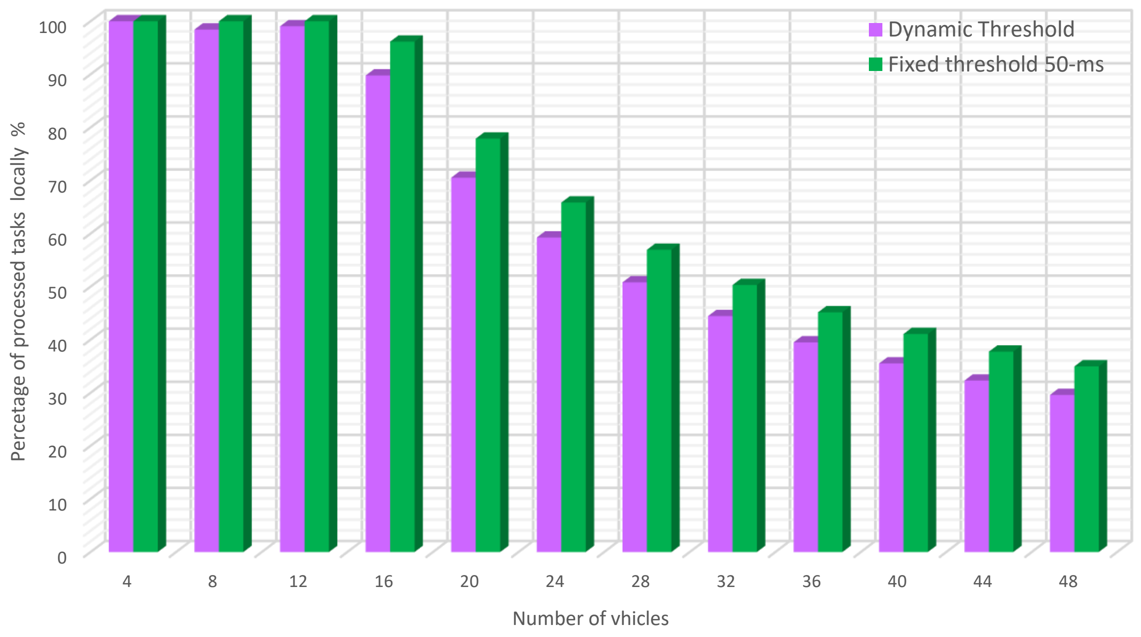


Figure 3.19: impact of increasing number of vehicles on the percentage of processed tasks locally in dynamic and fixed offloading threshold

3.6.2.10 Impact of increasing number of neighbours on delay with various offloading schemes

The impact of increasing the number of neighbours is carried out to investigate its impact on overall system performance and to find the optimal number of neighbours that are required. From Figure 3.20, we observe that as the number of neighbours is increased, the delay decreases for both control loops. However, when a certain number of neighbours is reached (e.g., five neighbours), the delay remains almost stable with slight increase despite adding further neighbours. This means that the optimal number that is required to achieve minimum delay has been reached, and no additional neighbours are needed to save energy consumption of the fog paradigm. The reason for the stable pattern is attributed to the workload, as most of the generated tasks have been processed. We note that DT accomplished the least delay for both control loops as the number of neighbours is increased, compared to the other schemes: ST50, JTORA and WO. With three neighbours, DT decreases delay by 10.80%, 13.38% and 15.29% compared to ST50, JTORA and WO, respectively. When the number of neighbours is five, the DT scheme reduced delay by 55.94%, 70.64% and 72.55% in comparison to ST50, JTORA and WO, respectively. When comparing DT to ST50 in control loop B, it can be observed that there is a slight difference in the average delay, and this remains unchanged even when increasing the number of neighbours. This is because tasks in control loop B should be processed in the cloud servers. Regarding control loop A, DT helps reduce the average delay by around 30% when the number of neighbours is 3 and 4. A further increase in the number of neighbours will lead to a reduction of the average delay by approximately 56%.

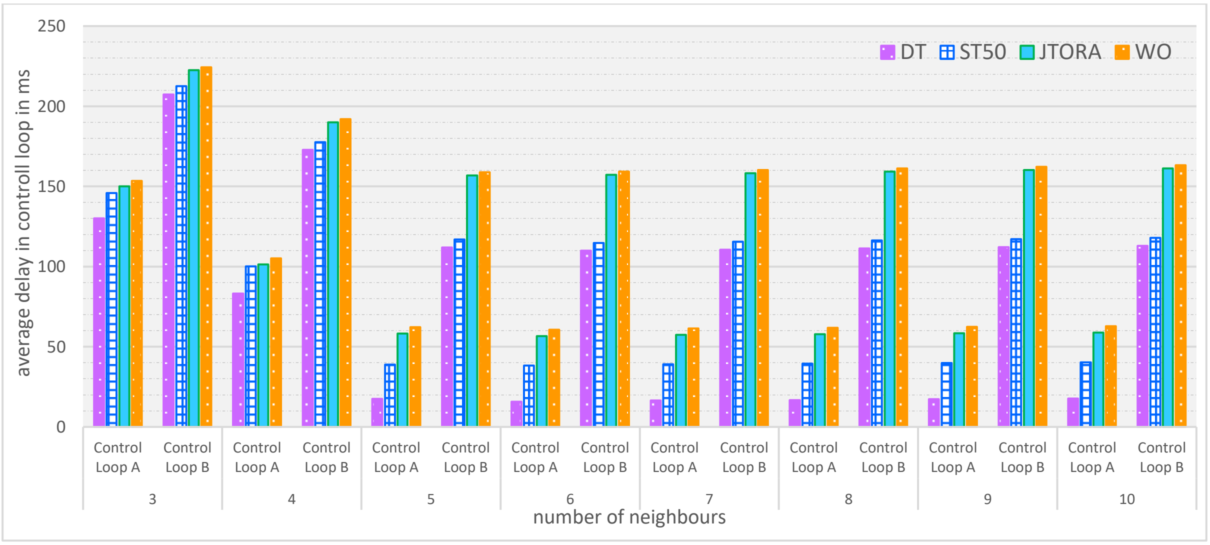


Figure 3.20: impact of increasing number of neighbours on average delay in control loops with various offloading schemes

3.6.2.11 Impact of increasing number of neighbours on throughputs with various offloading schemes

The impact of increasing the number of neighbours on throughput shows a similar pattern as increasing the number of neighbours to decrease delay. Results is shown in Figure 3.21. As the number of neighbours increases, the percentage of processed tasks increases, until a certain number of neighbours is achieved (e.g., five neighbours), after which the pattern remains almost stable.

In terms of the comparison with other schemes, DT improves throughputs by 2.7% when the number of neighbours is three, 3.5% when the number of neighbours is four, and 4.2% when the number of neighbours is five, six, seven, eight, nine and ten, compared to ST50 scheme. When the optimal number of five neighbouring fog nodes is reached, the DT processed 97.66% of the total generated tasks, while ST50, JTORA and WO processed 93.55%, 91.06% and 90.5%, respectively. The DT scheme improves throughput compared to other stated schemes as the number of neighbouring fog nodes was increased.

It can be observed that when the number of neighbours is small, e.g., 3, both approaches, DT and ST50, achieve lower percentages of throughputs at approximately 72.1% and 69.07%, respectively. This is primarily due to the highly congested environment and the system's limited computational resources. However, with four neighbours, both approaches perform better compared to having three neighbours. In this scenario, DT and ST50 achieve throughput percentages of 85.22% and 81.03%, respectively. Increasing the number of neighbours to five or more will further improve throughput compared to having four neighbours. In this case, DT will accomplish a throughput percentage of 97.67%, while ST50 achieves 93.56%.

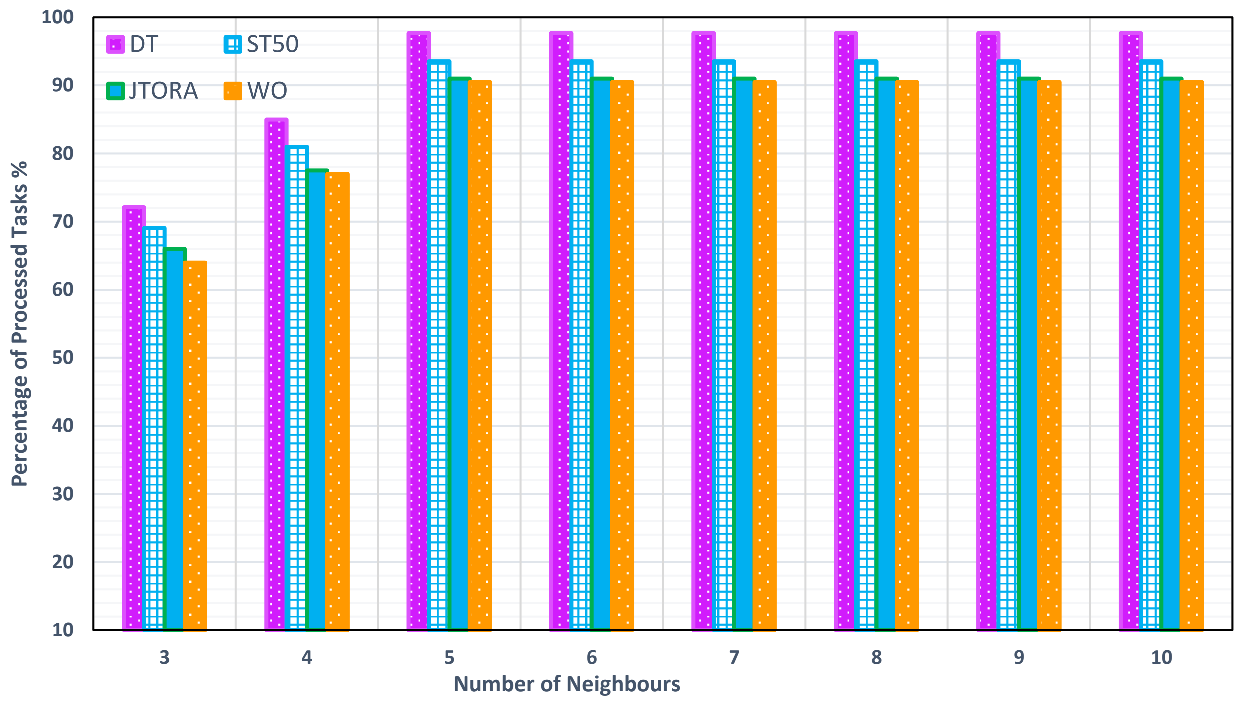


Figure 3.21: impact of increasing number of neighbours on throughputs with various offloading schemes

3.6.2.12 Impact of increasing number of neighbours on energy consumption with various offloading schemes

The impact of increasing the number of neighbours on energy consumption is investigated with various offloading schemes, as shown in Figure 3.22. When increasing the number of neighbours, the energy consumption in the system is increased because of operating additional fog nodes. Addressing the impact of increasing the number of neighbours helps to find the optimal number of neighbouring fog nodes that is necessary to achieve optimum results. When having five neighbours the difference between the energy consumed with and without DEC is very low; then as we increase the number of neighbours, the difference starts to increase. In the no offloading scheme, the impact of utilising DEC can be observed, i.e., reducing the wastage of energy by 55.72% when the number of neighbours is three, and up to 80.74% when the number of neighbours is ten. This method can also be applied to ST50 and DT, as DEC saves up to 38.58% and 32.16% of energy for each scheme respectively, when the number of neighbours is ten.

Without applying DEC, DT consumes slightly more energy than ST50. This is because, in both methods, all fog devices are switched on continuously. In addition to that ST50 does not process more tasks in the system compared to DT, and most of the time, fog devices remain in an idle state. When comparing ST50 to DT after applying DEC, DT consumes more energy. This is due to the nature of this scheme, where more tasks are processed in DT than in ST50. The energy consumed during the processing of these tasks leads to an overall increase in energy consumption within the system.

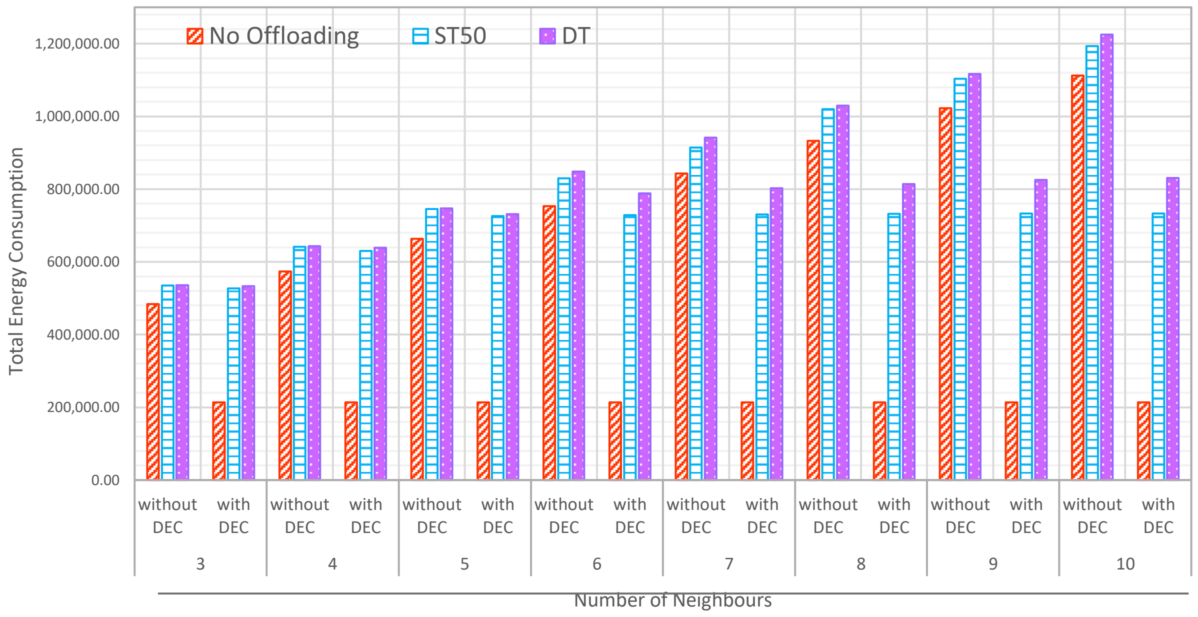


Figure 3.22: impact of increasing number of neighbours on Energy consumption with various offloading schemes

3.7 Conclusion

In this chapter, we studied the problem of minimising service latency and power consumption in online fog computing systems and proposed a combination of two efficient and effective algorithms named: dynamic task scheduling (DTS) and dynamic energy control (DEC). We have demonstrated their performance on various static offloading thresholds and when increasing the number of neighbours in the system. Compared to the no-offloading scheme along with other offloading approaches with various offloading thresholds, the experimental results validate that our proposed solution can reduce up to 89.16% of the task round-trip time and save up to 70% of the total energy consumption. In addition to that it enhances throughput by almost 40% compared to the no-offloading approach. Then, we proposed a dynamic offloading threshold that allows a fog node to adjust its threshold dynamically, and we investigated its influence of system performance with a set of benchmarks. Various numerical results are included, and the performance evaluations were presented to illustrate the effectiveness of the proposed scheme and demonstrate the superior performance over existing schemes.

It is also noticed that determining the offloading threshold upon which fog nodes start sharing their workloads with neighbours, and the optimal number of available neighbours play a significant role in the performance of the system. The values of these parameters cannot be determined in advance in such a dynamic online system. In this regard, it is necessary for us to propose a dynamic offloading threshold that takes into considering the status of the fog node and all its neighbours which is addressed in the next chapter.

4 Optimising the Energy Consumption in Stochastic Cooperative Fog Computing Systems

4.1 Introduction

By integrating fog computing into IoT systems, new levels of flexibility and scalability have been introduced to data processing. However, as the number of IoT devices and the volume of data generated continue to increase, this will pose a burden on the system. Consequently, the deployment of a significant number of fog devices is expected to increase in order to meet QoS requirements. In this context, it becomes increasingly important to manage energy consumption in these systems. This is to ensure that resources are being used efficiently at minimum cost, and that the system is operating at optimal levels by avoiding any failure caused by overloading or overheating of servers. In this chapter, the optimisation problem that considers minimising the energy consumption in the fog computing system in stochastic environment under certain constraints is investigated. The rest of this chapter is organized as follows; Section 5.2 defines the system model, and the communication model between system entities is described in section 5.3. Section 5.4 presents the Queueing model, and workload model is described in section 5.5. Processing model, Delay model, and energy consumption model are defined in sections 5.6, 5.7, and 5.8, respectively. In section 5.9, the optimisation problem is formulated along with system constraints. A plan to solve to the optimisation problem is provided in section 5.10, and finally, section 5.11 summaries this chapter.

4.2 System model

In our proposed system architecture, we have a set of IoT regions, fog nodes (FN), local controllers, global controller, and cloud servers. Each set of fog nodes with one local controller are grouped as a cluster, and there is K clusters in the system. The global controller monitors the performance of all local controllers. In each IoT region, we have a set of IoT devices that produce heterogenous tasks stochastically. These tasks will arrive at the primary fog node, in which these IoT devices located under its coverage. For each primary fog node (PFN), there is a set of neighbours (N), which are willing to cooperate and help processing its workload to achieve overall system goal. Each primary fog node connects to its neighbours and to the local controller via wireless links, while it connects to the cloud data centres over wired links. The interactions among fog nodes belonging to the same cluster is allowed but there are no interactions between fog nodes which belong to different clusters.

In this system, we have a time-slotted structure, indexed by $t \in$ {0, 1, 2, …, T-1}, and the length of each time slot is constant and its Δt. Each fog node is characterized by {f_i, δ_i(t-1), δ_i(t), memory}. f_i determines its computational capacity and is measured in mips, and δ_i(t-1) determines the previous status of fog node in time slot (t-1) in terms of being active or asleep, which is specified by its local controller, and δ_i(t) specifies the current status of the fog node. If δ_i(t)=1, it means that the status of fog node i in time slot (t) is active, and asleep otherwise. In time slot (t), the number of active fog nodes in a cluster is referred as (AN) and (SN) for sleeping fog nodes. We consider a stochastic task arrival model, where the arrival of tasks is not known in advance. Each task i $\in T^{task}$ generated from IoT devices is characterized by a tuple represented as {Type$_i$, M_i, S_i, D_i, ME$_i$}, where Type$_i$ is the type of task i, which is either private, semi-private and public, M_i is the processing requirements of task i and is defined as million instructions (MI). S_i is the size of task i in bits, D_i is the maximum deadline for task i. ME$_i$ is the required memory to store ith task in the arrival queue before processing it.

The parameters used to represent the system is shown in Table 4.1.

Table 4.1: System model parameters definitions.

Parameter	**Definition**
FN	Set of fog nodes in the cluster
PFN	Primary fog node
N	Set of neighbours
AN	Active fog nodes
SN	Sleeping fog nodes
f_i	Determines the computational capacity of fog node i and measured in mips
t	Time slot
$\Delta\ t$	Length of time slot in ms
T^{task}	Set of all tasks in the system
M_i	The processing requirements of task i in MI
S_i	The size of task i in bits
δ_i	The status of the fog node i in regard to being active or in a sleep mode
D_i	Maximum deadline for task i
MEi	The required memory to store ith task in the arrival queue
Q_i	Arrival queue of fog node i
LQ_i	Local queue of fog node i
HQ_i	Help queue of fog node i
$b_i^e(t)$	The amount of workload that can be assigned to fog node i in time slot (t),
$b_{i,max}^e$	Is the maximum amount of workload that can be processed by fog node i in time slot (t)
$\gamma_i\ (t)$	Decision vector to *i*th fog node in time slot (t)
$T_e, T_{tr}, T_{idle}, T_{sleep,} T_{so}$	execution delay, transmission delay, time for being idle, sleeping time, time overhead for activating a sleeping fog node, respectively
$E_e, E_{tr}, E_{re}, E_{idle}, E_{sleep,} E_{so}$	Execution energy, transmission energy, receiving energy, being idle energy, sleeping energy and energy overhead for activating a sleeping fog node, respectively.
$\overline{E^c(t)}$	Time average energy consumption for all fog nodes

4.3 Communication model

The proposed system architecture is shown in Figure 4.1. In time slot (t) which refers to the time duration [t-1, t), each IoT region i generates a number of tasks referred to as $A_i(t)$. These tasks enter the arrival queue of the PFN i until the decision is made on where to process them. Each FN_i maintains three types of queues called, arrival queue Q_i, local queue $LQ_i$and help queue HQ_i. Arrival queue contains tasks that are arriving from IoT regions, and local queue contains tasks that are transferred from the arrival queue and the decision is made to process them locally. Help queue contains tasks that are offloaded from other neighbours to process them in FN_i. More details about the queues are provided in the Queue model.

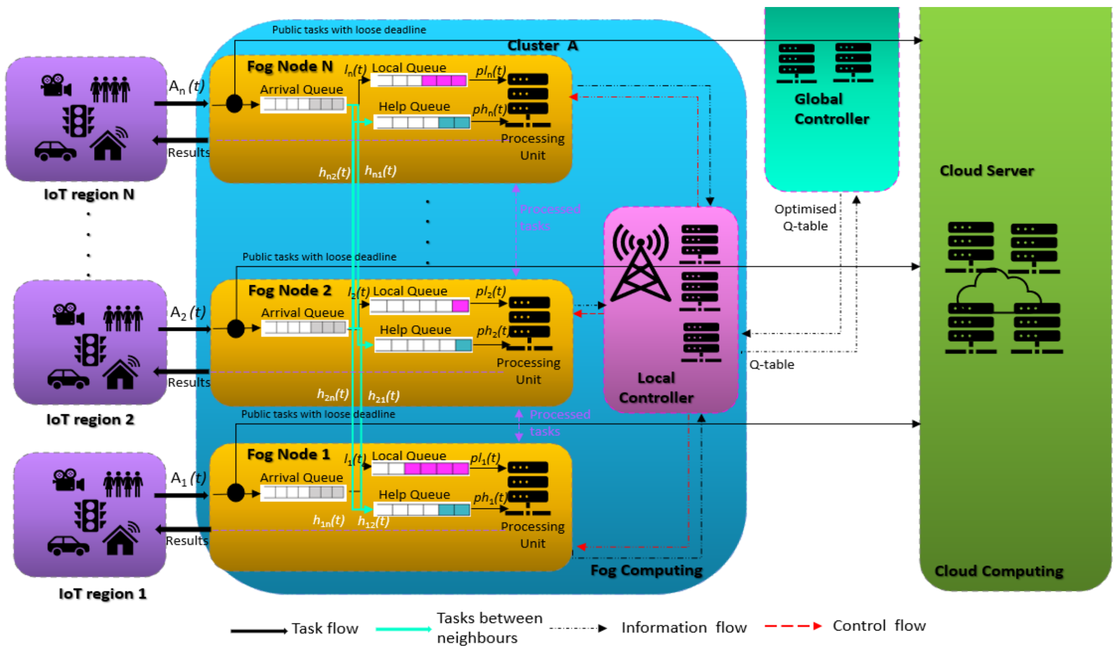


Figure 4.1: The proposed architecture in IoT-Fog-Cloud system in one cluster.

A number of N fog nodes are grouped in one cluster and controlled by one local controller. This is done to deal with a large-scale system with stochastic arrival of tasks and dynamic changes in system status. In this manner, the local controller is aware of the status of all fog nodes under its controller at the beginning of each time slot, and thus make the optimal offloading decision. The responsibility of a local controller in each cluster are to:

- Monitor the system.
- Analyse the conditions of the fog nodes under its controller and determine the status of each fog node in terms of being active or sleep.

- Make the optimal offloading decision between fog nodes considering the status of the computing devices.

The communication between each PFN, the local controller, and its neighbours in one cluster is shown in Figure 4.2. At the beginning of each time slot (t), the following procedures are accomplished in the following order.

1) Each PFN sort tasks upon their arrival to the Q_i based on their types and deadlines, this is described more in the workload model.
2) Then, each PFN determines the amount of workload that will leave its Q(t) and enters its LQ(t) based on the policy.
3) Each FN forms an information vector X_i(t) contains the status of its queues and sends it to the local controller.
4) Then the local controller receives and analyses the global information from all FN under its controller as

$$X(\mathrm{t}) \triangleq [\, X_1(\mathrm{t}), X_2(\mathrm{t}), \dots \dots, X_k(\mathrm{t})], t \in [1, 2, \dots, T]$$

5) Based on the received information, the local controller applies Q-learning to determine the status δ_i(t) of each FN as

$$\delta(\mathrm{t}) \triangleq [\delta_1(t), \delta_2(t), \dots \dots, \delta_k(t)\,], t \in [1, 2, \dots, T]$$

6) The proposed strategy is then applied based on Lyapunov optimisation and creates the optimal offloading decisions for all its FNs regarding which neighbours process which workloads, as

$$\gamma(t) \triangleq [\, \gamma_1(\mathrm{t}), \gamma_2(\mathrm{t}), \dots \dots, \gamma_k(\mathrm{t})], t \in [1, 2, \dots, T]$$

Where $\gamma_k(\mathrm{t})$ is the decision vector of the *k*th FN.

7) The decision vector $\gamma_i(\mathrm{t})$ is defined as

$$\gamma_i(\mathrm{t}) = [h_{i0}, h_{i1}, \dots., h_{iAN}]$$

Where h_{i0} determines the amount of workload that will be transmitted from PFN i to neighbour 0.

8) Then, the local controller notifies each PFN with its status and offloading decisions along with the expected amount of workload that will be received from all neighbours.
9) Then, two cases of PFN are considered and classified as sleeping FN, and active FN.
10) Each sleeping FN will offload its tasks based on the offloading decision and then enter a sleeping mode. At the end of time slot (t), the PFN will be activated to receive the processed workload from its neighbours and the cloud.
11) Each AN will offload some of its workload to other neighbours based on the decision vector created by the local controller. The Lyapunov optimisation is then applied to determine the amount of workload that should be processed in time slot (t) from its LQ(t) and HQ(t), which referred to as $pl(t)$ and $ph(t)$.

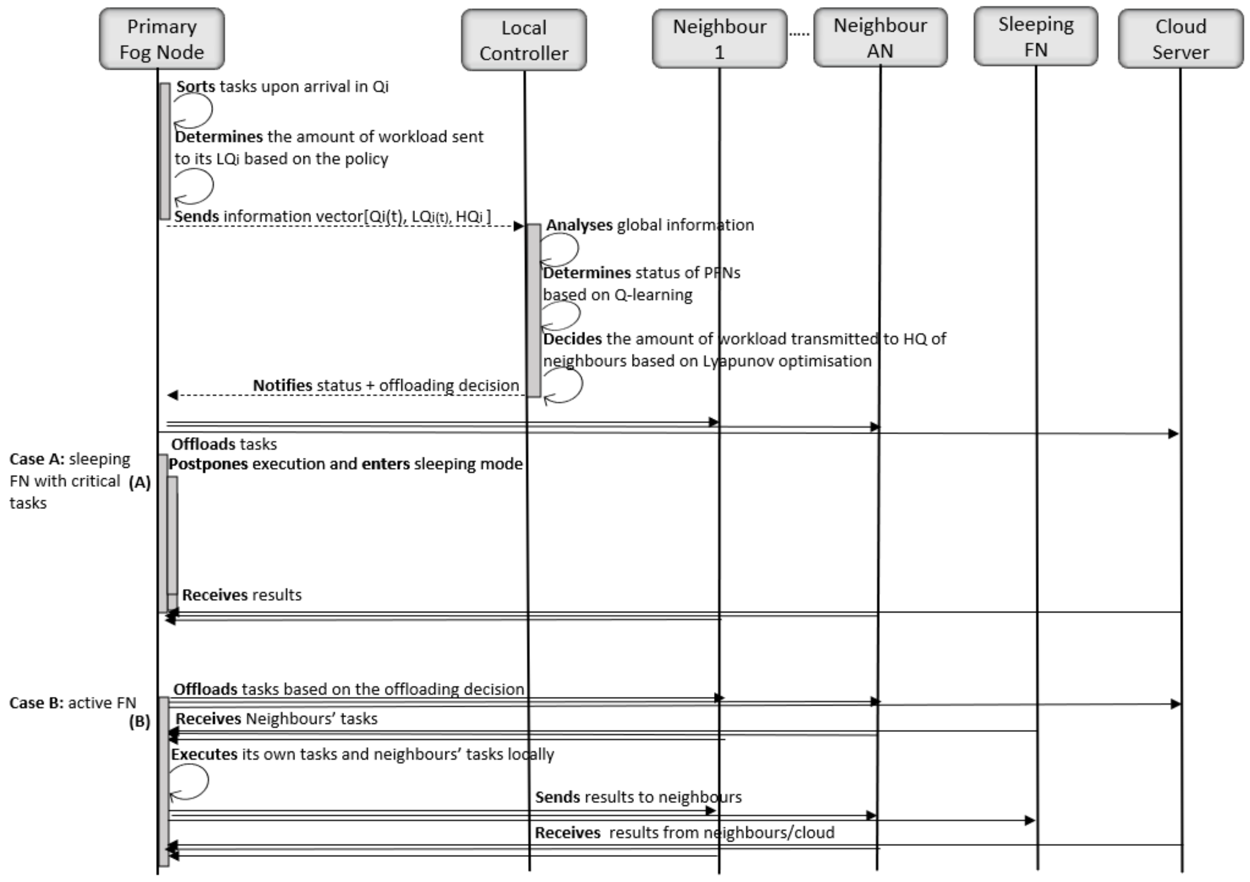

Primary Fog Node
Local Controller
Neighbour 1
.....
Neighbour AN
Sleeping FN
Cloud Server
Sorts tasks upon arrival in Qi
Determines the amount of workload sent to its LQi based on the policy
Sends information vector[Qi(t), LQi(t), HQi]
Analyses global information
Determines status of PFNs based on Q-learning
Decides the amount of workload transmitted to HQ of neighbours based on Lyapunov optimisation
Notifies status + offloading decision
Case A: sleeping FN with critical tasks
(A)
Offloads tasks
Postpones execution and enters sleeping mode
Receives results
Case B: active FN
(B)
Offloads tasks based on the offloading decision
Receives Neighbours' tasks
Executes its own tasks and neighbours' tasks locally
Sends results to neighbours
Receives results from neighbours/cloud

Figure 4.2: Sequence diagram of communications between system entities within a cluster.

4.4 Workload model

Upon the arrival of tasks in $Q(t)$, each PFN ranks the arrival tasks based on their types and deadlines. Private tasks will be located at the head of $Q(t)$, followd by semi-private tasks and public. The reason for organising tasks based on their types is to make it easier when making the offloading decision to process each task, which is described in the Lyapunov Optimisation section in Chapter 5. In each section, tasks are ranked based on their deadlines, where tight-deadline tasks will be at the top and tasks with loose deadlines are located at the end. Regarding the ranking of tasks based on their deadlines, this is done to meet various deadline requirements and speed up the process of executing tight-deadline tasks to avoid dropping them as the result of being in the queue for a long time.

Upon the arrival of public type of tasks, if its deadline is not violated by processing it in the cloud, the task is sent upon its arrival to the cloud. In this context, loose-deadline public tasks are sent to the cloud for processing, and tight-deadline public tasks are processed within the fog infrastructure. This is to assure having enough fog resources to process private and semi-private tasks which can only be processed within the fog infratsture along with public tasks with tight deadlines.

In the proposed framework, we consider IoT tasks differing in processing requirements, deadlines, priorities, and security requirements. Each IoT device generates six types of tasks with heterogeneous characteristics as shown in the Table 4.2.

Table 4.2: Characteristics of various types of tasks.

Type	***Priority***	***Where to process***	***Processing requirement in MI***	***Deadline***
Private 1	-	PFN	Low	Loose
Private 2	✓	PFN	High	Tight
Semi-private 1	-	PFN & AN	Low	Loose
Semi-private 2	-	PFN & AN	High	Tight
Public 1	-	PFN & AN & C	Low	Loose
Public 2	-	PFN & AN & C	High	Tight

We investigated three main types of tasks regarding their security level, Private, Semi-private, and public tasks. Private tasks can only be processed locally within the primary fog nodes. Semi-private tasks can be processed within the fog infrastructure, for example, the primary fog node and any of its neighbours. Public tasks can be processed in any computing device located in the fog system and the cloud system. In each of these types, we have two different categories. The first category is computational-light tasks with loss deadline and the second are computational-intense tasks with tight deadlines. Moreover, we assign a priority for processing private2 tasks. This is to examine how the system would perform with various tasks' characteristics when assigning a priority to a specific task. The security level of tasks is ranked as

$$Private\ tasks > Semi - Private\ tasks > Public\ tasks \tag{4.1}$$

For computing devices, the primary fog node is considered as high-secured computing device, neighbours are semi-secured, and the cloud server is not secured, and they are ranked as

$$High - Secured\ device\ > Semi - Secured\ device\ > Not - Secured\ device \tag{4.2}$$

4.5 Queueing Model

As seen in Figure 4.1 each primary fog node maintains three queues. The first queue is called arrival queue Q_i and is utilized to store all tasks arriving from the IoT devices under their coverage area. The arrival tasks will not departure from the arrival queue unless the decision is made to where to process them. At the beginning of each time slot t_h the vector of the arrival queues of all FNs are

$$Q(t_h) = \big(Q_1(t_h), Q_2(t_h), Q_3(t_h), \dots, Q_k(t_h)\big) \tag{4.3}$$

The arrival queue of a single fog node evolves in the next time slot (t+1) as

$$Q_i(t+1) = \max\left[Q_i(t) - l_i(t) - \sum_{j \in AN, i \neq j} h_{ij}(t), 0\right] + \mathrm{A}_i(\mathrm{t}) \tag{4.4}$$

Where $l_i(t)$ is the amounts of workload that leaves the arrival queue Q_i based on the proposed policy and enters the local queue LQ_i. h_{ij} is the amount of workload that transmitted from the arrival queue of fog node i to the help queue HQ_j of neighbour j.

The second queue is called local queue LQ_i, this queue contains tasks that are transmitted from the arrival queue to it based on the proposed algorithm, so it only contains tasks that belong to IoT devices under its coverage. At the beginning of each time slot t_h the vector of the local queues of all FNs are

$$LQ(t_h) = \big(LQ_1(t_h), LQ_2(t_h), LQ_3(t_h), \dots, LQ_k(t_h)\big) \tag{4.5}$$

The evolution of the local queue in time slot (t+1) is as follows:

$$LQ_i(t+1) = \max[LQ_i(t) - pl_i(t), 0] + l_i(t) \tag{4.6}$$

Where $pl_i(t)$ is the amounts of tasks that leave local queue for processing in time slot (t).

The third queue is called the help queue (HQ_i), HQ_i belongs to fog node i and contains tasks that are offloaded from the neighbouring fog nodes to be processed in fog node i. At the beginning of each time slot t_h the vector of the help queues of all FNs are

$$HQ(t_h) = \left(HQ_1(t_h), HQ_2(t_h), HQ_3(t_h), \dots, HQ_k(t_h)\right) \tag{4.7}$$

HQ_i evolves as

$$HQ_i(t+1) = \max[HQ_i(t) - ph_i(t), 0] + \sum_{j \in FN, i \neq j} h_{ji}(t) \tag{4.8}$$

Where $ph_i(t)$ is the amount of workload that leaves the help queue based on the policy and determined to be processed in time slot (t). $h_{ji}(t)$ is the amounts of workloads that received from other neighbours and enters the help queue of fog node i.

4.6 Processing model

Following that each fog node determines the total amount of workload that is selected from the local queue and the help queue to be processed in the current time step (t) based on its proposed policy. These amounts of workload should not exceed the available computational resources in the computing device. The maximum amount of workload that can be processed in fog node i referred to as $b_{i,max}^{e}\ (t)$ and it is defined as

$$b_{i,max}^{e} = (\Delta\, t - x) *\ f_i \tag{4.9}$$

$\Delta\, t$ is the length of the time slot, for example, 5 ms, and x is the latency overhead associated if fog node i was in a sleep state in time slot (t-1). f_i is the computational capacity of fog node i and measured in mips. According to that the following constraint should be met.

$$pl_i(t) + ph_i(t)\ \leq\ b_{i,max}^{e} \tag{4.10}$$

4.7 Delay model

In our system, once the tasks are generated from the IoT regions, they are transmitted to the fog infrastructure, and queued in an arrival queue of each primary fog node, until the decision is made where to optimally process them. The waiting time of the tasks in the arrival queue, local queue and help queue is unpredictable due to having a stochastic system [93]. In more details, tasks will be waiting in the arrival queue until the beginning of the time slot (t) where the PFN decides to move some tasks from its arrival queue to its local queue or the local control decides which neighbour/s to handle them in their help queue/s. However, if there are not enough computational and energy resources within a cluster in the fog computing system in time slot (t), tasks will be waiting in the next time slot (t+1) until the availability of the fog resource. Moreover, the availability of the fog resources cannot be determined in advance due to the randomness of the arrival of tasks in different IoT regions. To avoid having tasks being in the queues for several time slots and to control the waiting time of tasks in queues, we introduce the following constraint.

$$D_{max}^{i} < D_{max} \tag{4.11}$$

Where D_{max}^{i} represents the worst-case delay. This constraint will be further clarified in Lyapunov chapter. T_a^i represents the waiting time in the arrival queue, T_l^i and T_h^i are the waiting time in the local queue and the help queue respectively.

The worst-case delay defined in the optimisation problem, which controls the maximum waiting time of a task in each queue, is replaced with a virtual queue following the work in [94, 95].

A virtual queue is created for each actual queue in the system, and the virtual queue correlated to the arrival queue is denoted as X_i^Q, initially $X_i^Q(0) = 0$, and it evolves as

$$X_i^Q(t+1) = \max\left[X_i^Q(t) - l_i(t) - \sum_{j \in AN, i \neq j} h_{ij}(t), 0\right] + \epsilon_{i1}$$

Where ϵ_{i1} is a constant value added at each time slot if the actual arrival queue Q_i is not empty. The concept of introducing virtual queues is useful when making the offloading decision. For example, if there is one task is in the arrival queue and it has been waiting for several time slots to be transmitted to the local queue or to the help queue of other neighbours, however, the system is focusing on the congested queues to process as much of its workload to ensure stability in the system over the queue that contains one task whose deadline is nearly violated. As each time slot passes, the value of the virtual queue increases by ϵ_{i1} which drive the attention of the system to process this task. With regards to the local queue, its virtual queue X_i^{LQ}, and initially in time slot (0), $X_i^{LQ}(0) = 0$, and it evolves as

$$X_i^{LQ}(t+1) = \max\left[X_i^{LQ}(t) - pl_i(t), 0\right] + \epsilon_{i2}$$

The concept of constant ϵ_{i2} is similar to ϵ_{i1}.

The virtual queue related to the help queue denoted as X_i^{HQ} and it has the value zero in time slot (0), and it evolves as

$$X_i^{HQ}(t+1) = \max\left[X_i^{HQ}(t) - ph_i(t), 0\right] + \epsilon_{i3}$$

4.7.1 Execution delay

The execution delay (T_e) is calculated as

$$T_e = \frac{M_i}{f_j} \tag{4.12}$$

4.7.2 Transmission delay

Transmission latency (T_{tr}) is the time taken for a task to transmit from the primary fog node to the neighbouring node/cloud server, and after processing the task, the processed result is sent back to the primary fog node.

The transmission latency is related to the size of the offloaded task (S_i) and the bandwidth rate (R_i^{tr}). The bandwidth rate is defined as

$$R_{ij}^{tr} = B_{ij}^{tr} log_2 \left(1 + \frac{P_i^{tr}\, hp_j^i}{\xi^2} \right) \tag{4.13}$$

Where B_{ij}^{tr}refers to the channel bandwidth between the primary fog node i and its neighbour j. P_i^{tr} is the required transmission power to offload task i to fog device j. hp_j^i is the channel gain between fog node i and j, , and ξ^2 is the white Gaussian noise.

Then, the transmission delay (T_{tr}) is calculated as

$$T_{tr} = \frac{S_i}{R_{ij}^{tr}} \tag{4.14}$$

After processing task i, the results are transmitted back to the primary fog node i, and the time taken to send the processed task T_{re} is.

$$T_{re} = \frac{SR_i}{R_{ji}^{tr}} \tag{4.15}$$

Where SR_i is the size of the processed task i.

4.7.3 End-to-end delay

If the optimal decision is to process the task x locally on the primary fog node i, end-to-end delay (T_{total}) is calculated as

$$T_{total}^{x} = (T_a^i + T_l^i + T_e^i) + (1 - \delta_i(t-1))(T_{so}^i) \tag{4.16}$$

Where $\delta_i(t-1)$ represents the status of ith fog node in last time step (t-1). If ith fog node previous status is active, then $\delta_i(t-1) = 1$ and 0 otherwise. T_{so}^i is the latency overhead for activating the sleeping ith fog node. If the best decision is to process the task x in the neighbouring fog node j, T_{total} is defined as

$$\begin{aligned} T_{total}^{x} = {} & (T_a^i + T_{tr}^{ij} + T_h^j + T_e^j + T_{re}^{ji}) + (1 - \delta_i(t-1))(T_{so}^i) \\ & + (1 - \delta_i(t-1))(T_{so}^j) \end{aligned} \tag{4.17}$$

T_{re}^{ji} is the transmission delay for sending the results back from the neighbouring fog node j to the primary fog node i. If the optimal decision is to process the task in the cloud server, T_{total} is calculated as

$$T_{total}^{x} = (T_{tr}^{ic} + T_a^c + T_e^c + T_{re}^{ci}) + (1 - \delta_i(t-1))(T_{so}^i) \tag{4.18}$$

The average end-to-end latency for all tasks of the same type that are processed in the system is calculated as

$$T_{average} = \frac{\sum_{x=1}^{X} T_{total}^{x}}{X} \tag{4.19}$$

Where X is the total number of tasks of the same type in the system.

4.8 Energy Consumption model

In the time-slotted system, the energy is renewed at the beginning of each time slot, and each fog node has a maximum amount of energy that should not exceed during the time slot (t). If a fog node exceeds the maximum threshold of its energy, tasks will not be executed, and the device will shut down. The energy model considered in the offloading system is the energy spent during executing, transmitting, and receiving tasks, being idle and in a sleep mode. We only consider the energy consumption in the fog computing system.

The energy consumption in each fog node in time slot (t) is calculated based on its state/s, $s \in S$, which is defined as

$S = \{computing, transmitting\ and\ reciving\ , being\ idle, being\ sleep, sleep\ overhaed\}$

The energy consumed at each state is correlated to the time spent during that state (T_s) [88-90] and it is defined as

$$E = E_s * T_s\ s \in S$$

$$E_s = \{E_e, E_{tr}, E_{re}, E_{idle}, E_{sleep,} E_{so}\}$$

$$T_s = \{T_e, T_{tr}, T_{re}, T_{idle}, T_{sleep,} T_{so}\}$$

$$E_e > E_{tr} + E_{re} > E_{idle} > E_{sleep} + E_{so} \quad (4.20)$$

T_{idle} is the time spent for the computing device while not doing any processing. The total energy consumption for fog node i in time slot (t) is calculated as follows.

$$E_i(t) = \underbrace{\big(1-\delta_i(t-1)\big)(E_{so})}_{\substack{energy\ overhead\ if\ fog\ node\ i\ was \\ a\ sleep\ in\ time\ slot(t-1)}} + \underbrace{\left(E_i^e * \sum_{x=1}^{pl_i} T_x^e\right)}_{processing\ its\ own\ tasks} +$$
$$\underbrace{\left(E_i^{tr} * \sum_{j=1}^{h_{ij}} T_j^{tr}\right) + \left(E_i^{re} * \sum_{j=1}^{h_{ij}-b+a} T_j^{re}\right)}_{offloading\ its\ tasks\ to\ neighbours} + \quad (4.21)$$
$$\underbrace{\left(E_i^{tr} * \sum_{j=1}^{X_{cloud}} T_j^{tr}\right) + \left(E_i^{re} * \sum_{j=1}^{X_{cloud}-b+a} T_j^{re}\right)}_{offloading\ its\ tasks\ to\ cloud} +$$

$$\underbrace{\left(E_i^{re} * \sum_{j=1}^{h_{ji}} T_j^{re}\right) + \left(E_i^{e} * \sum_{j=1}^{ph_{ji}} T_j^{e}\right) + \left(E_i^{tr} * \sum_{j=1}^{ph_{ji}} T_j^{tr}\right) +}_{fog\ node\ i\ neighbours\ tasks}$$

$$\underbrace{\left(E_{idle}^{i} * T_{idle}^{i}\right)}_{time\ for\ being\ idle}$$

The first part of the energy consumption calculation is related to the energy overhead for activating a sleeping fog node in time slot (t-1). The second part is related to the energy consumption of processing its own tasks, where pl_i is the number of tasks that processed locally in time slot (t). The third part regards the energy consumed caused by transmitting a set of h_{ij} tasks to its neighbours and receiving a set of $h_{ij} - b + a$ processed tasks from the neighbours. $h_{ij} \neq h_{ij} - b + a$ as the number of tasks transmitted to all neighbours will not be processed in the same time slot (t), instead there will be b amount of these tasks that will be waiting in the help queue of the neighbours to be processed in the next time slot, and a is the number of tasks that had been in the help queues of the neighbours in the previous time slot. The same is applied for transmitting and receiving the processed tasks from the cloud. The fifth part is related to the energy consumed by fog node i to handle its neighbours' workload, where h_{ji} is the number of tasks that offloaded from its neighbours to be processed in fog node i, and ph_{ji} is the number of tasks that is processed in time slot (t) by fog node i that belongs to its neighbours, and $h_{ji} \neq ph_{ji}$. Finally, the energy that is consumed when the fog node is active and not doing any processing.

The total energy consumption of all fog nodes in one cluster is calculated as

$$E^c(t) = \sum_{i \in FN} E_i(t) \tag{4.22}$$

4.9 Problem formulation

In IoT environment, a large number of intelligent devices and industrial manufacturing will produce a huge energy consumption. Furthermore, there will be an increase in energy consumption at the fog infrastructure if an inefficiency fog resource management is considered, which will entail an increase in operational costs. Moreover, it may result in a reduction in the lifespan of the computing device and environment pollution. In addition to that, as the IoT environment and fog infrastructure consume more energy, possible resource depletion may result.

Energy conservation and emission reduction has become an important task of modern society. Thus, we focused on the minimisation of the energy consumption in a collaborative fog computing system. To be more specific, we aimed at minimising the expected time average energy consumption for all fog nodes in the system, which is expressed as

$$\overline{E^c(t)} = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{i=1}^{FN} \mathbb{E}\{ E_i(t)\} \tag{4.23}$$

Where $E_i(t)$ is the total energy consumed at fog node i during time slot (t)

The considered problem can be considered as a constrained optimisation problem as follows.

$$\boldsymbol{P1}:\ \boldsymbol{Minimse\ \overline{E^{c}(t)}} \tag{4.24}$$

Subject to

C1: $\bar{Q}_i = \lim_{T\to\infty}\sup \frac{1}{T}\sum_{t=0}^{T-1} Q_i(t) < \infty$

C2: $\overline{LQ}_i = \lim_{T\to\infty}\sup \frac{1}{T}\sum_{t=0}^{T-1} LQ_i(t) < \infty$

C3: $\overline{HQ}_i = \lim_{T\to\infty}\sup \frac{1}{T}\sum_{t=0}^{T-1} HQ_i(t) < \infty$

C4: $Q_i \le Q_i^{max}, LQ_i \le LQ_i^{max}, HQ_i \le HQ_i^{max}$

C5: $0 < E_i\,(t) < E_{max}^{i}\,(t), \forall\ \mathrm{i} \in \mathrm{FN}$

C6: $T_{total}^{i} \le D_i\ \ \forall\ \mathrm{i} \in \mathrm{T}^{task}$

C7: $SE_i \le SE_j\ \ \forall\ \mathrm{i} \in \mathrm{T}^{task},\ \forall\ j\ \in (\mathrm{AN}\ \cup \mathrm{C})$

C8: $\sum_{k=1}^{K} M_k(t) \le\ b_{j,max}^{e}\ (t), \forall\ \mathrm{j} \in \mathrm{FN}, \forall\ \mathrm{K} \in\ \mathrm{T}^{task}$

C9: $\sum_{k=1}^{K} S_k(t) \le\ B_j^{i}\ (t), \forall\ \mathrm{i} \in \mathrm{FN}, \forall\ \mathrm{j} \in (\mathrm{AN}\ \cup \mathrm{C}), \forall\ \mathrm{K} \in\ \mathrm{T}^{task}$

C10: $D_{max}^{i} <\ D_{max}, \quad \forall\ \mathrm{i}\ \in \{Q_i, LQ_i, HQ_i\}$

C11: $d_i^{\%}(t) \le\ d_{max}^{\%}(t)$

The average long-term energy consumption $\overline{E^{c}(t)}$ is defined in (4.23). Constraints C1 to C3 means that arrival queues, local queues, and help queues of all the primary fog nodes should be stable in the average time sense, which in turn ensures the stability in the fog computing systems. Constraint C4 is to ensure that the size of the arrival queues, local queues, and help queues have an upper bound. Constraint C5 ensures that the estimated energy consumed in a single fog node in time slot (t) is not exceeding the maximum allowed energy in a time slot (t). constraint C6 ensures that the end-to-end latency for task i is not violating the deadline requirement of task i. Constraint C7 indicates that, if the computing device j is selected to process task i, the computing device should have equal or higher security level than that of task i. the security levels of tasks and computing devices are defined in equations (4.1) and (4.2). constraint C8 states that the total processing requirement to process a number of tasks in the current time slot (t) should not exceed the maximum available computing resources in the fog node. Constraint C9 is to ensure that the total size of tasks transmitted from fog node *i* to the *j*th computing device should not exceed the available bandwidth between them. In constraint C10 we ensure the maximum waiting time for tasks in queues is not exceeding the worst-case delay. In constraint C11, $d_i^{\%}(t)$ refers to the percentage of dropped tasks in ith fog node at time slot (t) which it should not exceed the maximum allowed percentage of drooping tasks $d_{max}^{\%}(t)$.

4.10 Plan of solutions:

The long-term energy consumption in the online fog infrastructure can be minimised by efficiently addressing the problems of resource management and computational offloading and processing decisions. In this manner, an efficient energy management scheme for fog nodes based on sleeping cycle is proposed. This is accomplished through exploiting the advantage of Q-learning approach while still meeting the QoS requirements and the system constraints. Additionally, energy savings can be further maximised if an efficient computational offloading and processing decisions scheme is considered based on Lyapunov optimisation theory.

In this context, the problem of minimising energy in the fog computing systems can be expressed as two subproblems. The first subproblem is concerned with developing an efficient cooperative dynamic energy management scheme, which is explained in chapter six. Moreover, the second subproblem is concerned with finding the optimal offloading decision for each fog node, which is described in section seven. The proposed solution based on cooperative Q-learning and Q-learning runs on the global controller and local controllers of each cluster, respectively. Regarding the solution based on Lyapunov optimisation, it is applied by local controllers of each cluster and each fog node. Our proposed solution which is based on Q-learning and Lyapunov Optimisation is called Joint Q-Learning and Lyapunov Optimisation Algorithm (JQLLO).

4.10.1 Brief description of the proposed solution

In this section, the rules accomplished by the entities in the system represented by the global controller, local controllers, and each fog node in order to achieve the proposed solution is summarised.

4.10.1.1 The global controller

The global controller is responsible for applying cooperative Q-learning. This is accomplished once the global controller receives all Q-tables from all local controllers, then the optimised Q-table is created. Following that the global controller sends the optimised Q-table to all local controllers. The process is summarised in Figure 4.3. Full details are provided in chapter 5.

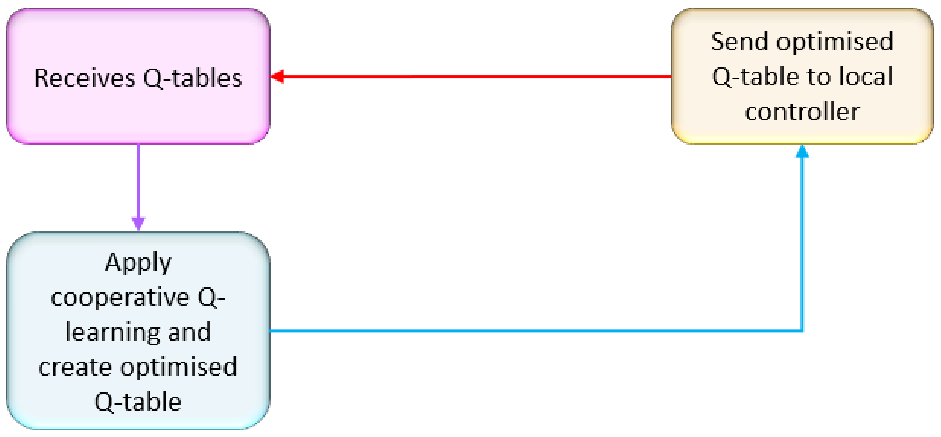

Receives Q-tables
Send optimised Q-table to local controller
Apply cooperative Q-learning and create optimised Q-table

Figure 4.3: An overview of the main operations performed by the global controller at the beginning of each time step.

4.10.1.2 Local controllers

The sequence order of the main functions for each local controller at the beginning of each time step is summarised in Figure 4.4. First, local controller receives the system status of all FNs in its cluster which includes the total number of tasks in their arrival queues, plus local queues, help queues, and the corresponding virtual queues. This information represents the congestion level in each fog node. Accordingly, each local controller ranks fog nodes in its cluster based on their congested level in ascending order, starting from the least to the most congested. Following that and based on the received reward and the Q-learning policy, a set of fog nodes is determined to be in a sleep mode, the least congested fog nodes.

Then, based on the local queues and the corresponding virtual queues, local controller ranks sleeping FNs starting from the most congested to the least. Then local controller ranks active FNs based on the arrival queue plus its virtual queue starting from the most congested to the least. Following that the local controller ranks neighbours based on their help queues and its corresponding virtual queues. Then, the optimal offloading decision is made for the sleeping fog nodes by selecting one of the neighbours and then updating its help queue.

The process of finding the optimal offloading decision for active fog nodes is based on Lyapunov optimisation and follows the same process of finding the optimal offloading decision. The local controller then sends offloading decisions to all FNs. Following this, each local controller receives a reward and observes the new system status and accordingly update its Q-learning policy. Then each local controller sends its Q-table to the global controller to perform cooperative Q-learning and accordingly, each local controller receives the optimised Q-table and updates their Q-learning policy based on it.

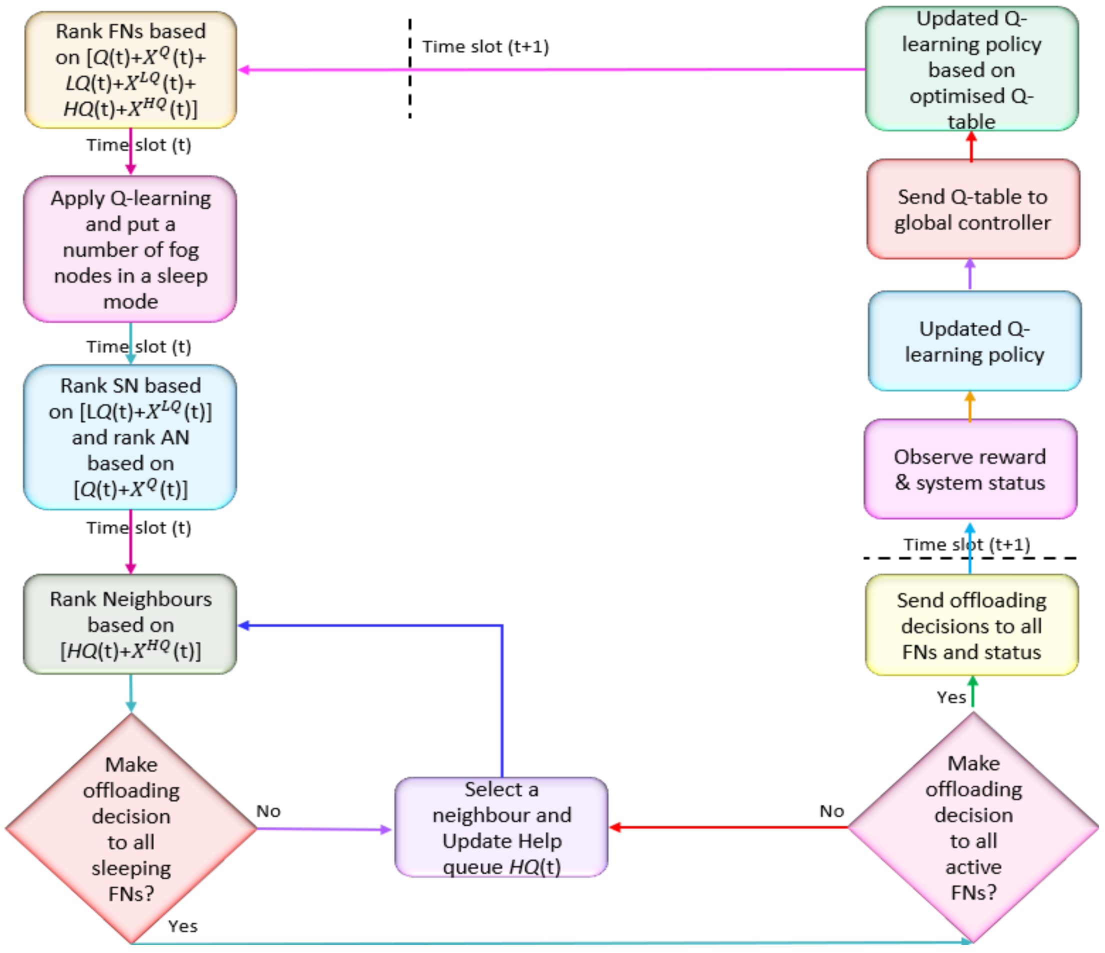

Rank FNs based on [Q(t)+X^{Q}(t)+ LQ(t)+X^{LQ}(t)+ HQ(t)+X^{HQ}(t)]
Time slot (t+1)
Updated Q-learning policy based on optimised Q-table
Time slot (t)
Apply Q-learning and put a number of fog nodes in a sleep mode
Send Q-table to global controller
Time slot (t)
Updated Q-learning policy
Rank SN based on [LQ(t)+X^{LQ}(t)] and rank AN based on [Q(t)+X^{Q}(t)]
Observe reward & system status
Time slot (t)
Time slot (t+1)
Rank Neighbours based on [HQ(t)+X^{HQ}(t)]
Send offloading decisions to all FNs and status
Yes
Make offloading decision to all sleeping FNs?
No
Select a neighbour and Update Help queue HQ(t)
No
Make offloading decision to all active FNs?
Yes

Figure 4.4: An overview of the main operations performed by local controllers at the beginning of each time step.

4.10.1.3 Fog nodes

The main rules of fog nodes are summarized in Figure 4.5. In arrival queues, tasks are sorted by type with private tasks at the top, then semi-private tasks and finally public tasks. Moreover, tasks are sorted according to their deadline, with tight-deadline tasks at the top followed by loose-deadline tasks. Then at the beginning of each time step, each fog node applies Lyapunov to determine the number of tasks transformed from its arrival queue to its local queue. Following that an information vector is created by each fog node containing information regarding the actual and virtual queues and sent to the local controller. Fog nodes, then, receive information about the optimal offloading strategy, the expected amount received from neighbours, and the status after the local controller makes its decisions. Following that each fog node will perform accordingly. Each active fog node applies Lyapunov optimization to determine the amount of workload that belongs to local and help queues that will be processed in the current time slot (t). This process is repeated at each time step.

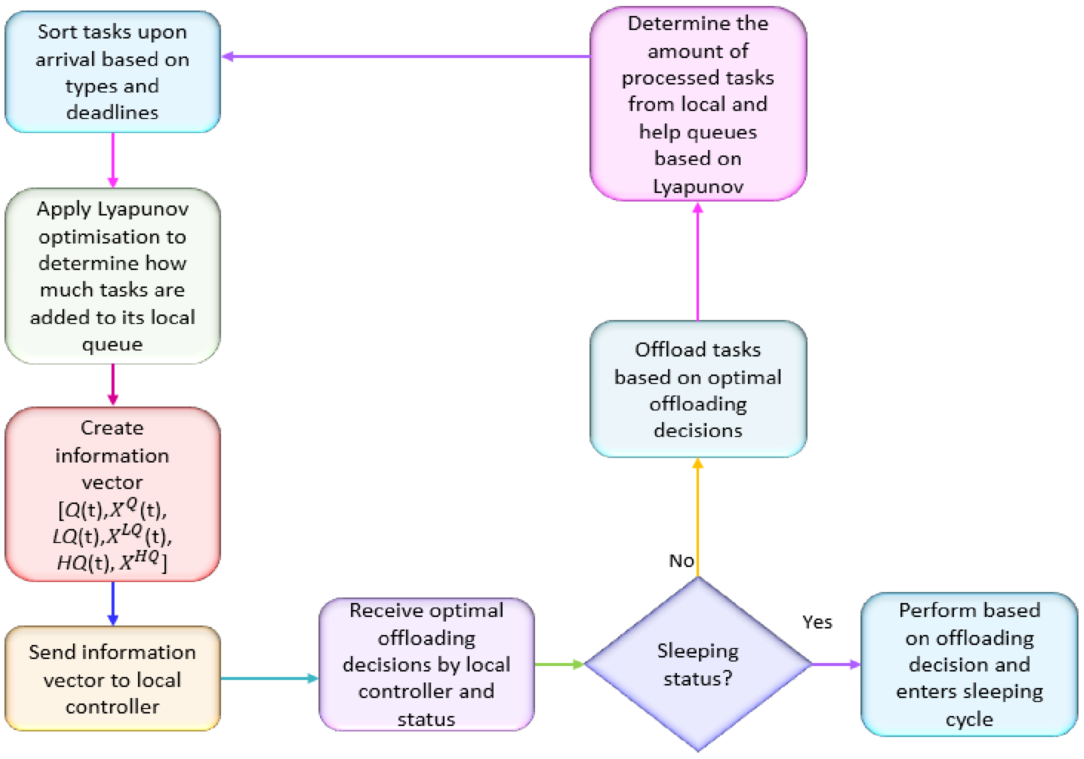

Sort tasks upon arrival based on types and deadlines
Apply Lyapunov optimisation to determine how much tasks are added to its local queue
Create information vector [Q(t),X^{Q}(t), LQ(t),X^{LQ}(t), HQ(t), X^{HQ}]
Send information vector to local controller
Receive optimal offloading decisions by local controller and status
Sleeping status?
Yes
Perform based on offloading decision and enters sleeping cycle
No
Offload tasks based on optimal offloading decisions
Determine the amount of processed tasks from local and help queues based on Lyapunov

Figure 4.5: An overview of the main operations performed by local controllers at the beginning of each time step.

4.11 Conclusion

This chapter represents the proposed system architecture along with system models and the main components considered in the system. The optimisation problem has been formulated with the objective of minimising long-term energy consumption in stochastic dynamic systems under certain constraints. Then, the main optimisation problem has been decomposed into two sub-problems and solved using a combination of Q-learning and Lyapunov Optimisation. A brief overview of the main operations performed by the entities in the system has been provided. In the next chapter, we propose a cooperative dynamic energy management scheme based on sleeping cycles as a solution to the first subproblem. As part of chapter 5, we describe how Q-learning, cooperative Q-learning, and a proposed approach are being incorporated into a learning agent's learning process to help it learn its environment more efficiently and quickly. Furthermore, in chapter 5, Lyapunov optimisation has been presented as an optimisation scheme that helps to solve the second subproblem regarding finding the optimal offloading and processing strategy. This is done to minimise energy consumption and ensure system stability.

5 Dynamic Resource Management, Computational Offloading and Processing Decisions Problems

5.1 Introduction

The process of allocating and managing resources in a system based on the changing demands and conditions of the system is referred to as dynamic resource management. For improving system performance, resource management aims to utilise limited network resources, such as energy and computation, as efficiently as possible. This is done while meeting QoS requirements and system constraints. In such a stochastic fog system designing an efficient dynamic resource management scheme is challenging and is not addressed in the literature adequately. Additionally, computational offloading and processing decisions is a crucial aspect of fog computing systems. Addressing computational offloading and processing decisions in an efficient manner will lead to an improvement in system performance and enhancement of the user experience.

In this chapter, the two subproblems are addressed, namely: Dynamic Resource Management problem as in section 5.2 and Computational Offloading and Processing Decisions Problem as in section 5.3. In section 5.2.1, Q-learning and its structure are described. The proposed solution (CQL-EUAs) that is based on the Cooperative Q-learning (CQL) algorithm along with the Eliminating Unacceptable Actions (EUAs) algorithm is described in section 5.2.2. In section 5.2.3, the results of experiments are described. The framework of Lyapunov Optimisation and drift-plus-penalty theory is addressed in section 5.3.1. In section 5.3.2 the problem transformation is provided along with the proposed solution. The set of conducted experiments and the results are in sections 5.3.3 and 5.3.4. Finally, the conclusion of this chapter is provided in section 5.4.

5.2 Dynamic Resource Management problem

This chapter addresses the first sub-problem stated in the plan of solutions in chapter five, regarding dynamic resource management. To optimise resource management in stochastic fog systems regarding energy efficiency, we propose the concept of putting some fog nodes in sleep mode. This will reduce energy consumption within the fog infrastructure without violating the QoS requirements. The problem is considered as a stochastic optimisation problem with the objective of maximising the number of sleeping fog nodes in order to save energy consumption within the fog infrastructure while meeting the QoS requirements in the system, and is represented as follows:

$$\max \sum_{i=0}^{K} \alpha_i \, FN_i \, , \forall \, i \, \in K \tag{5.1}$$

s.t C1: $\alpha_i \in \{0, 1\}, \forall \, i \in K$

C2: $\sum_{k=1}^{K} M_k(t) \leq b_{j,max}^{e} \, (t), \forall \; \mathrm{j} \in \mathrm{FN}, \forall \; \mathrm{K} \in \; \mathrm{T}^{task}$

C3: $cost_i < cost_{max} \, , \forall \, i \in K$

C4: $0 < E_i \, (t) < E_{max}^{i} \, (t), \forall \; \mathrm{i} \in \mathrm{K}$

The aim is to maximise the number of fog nodes that are in sleep mode. In constraint C1, α_i is a binary variable that represents the status of fog node FN_i, if $\alpha_i = 1$, this means that fog node FN_i is in sleep mode, and FN_i is active if $\alpha_i = 0$. Constraint C2 ensures that the total CPU resources required to process a set of assigned tasks K in a fog node i should not exceed the available CPU resources in fog node I. C3 is to ensure that the cost for offloading tasks that belong to a sleeping fog node is not exceeding the maximum budget for the sleeping fog node. Constraint C4 is to ensure that each fog node is not exceeding its maximum energy consumption in each time slot (t).

Traditional optimisation methods become nontrivial and challenging in dynamic and stochastic environments where tasks arrive unexpectedly, and computational and energy resources are not known in advance. The optimisation problem, in this context, is transformed into a Markov decision process (MDP) problem and then solved using a model-free Reinforcement Learning (RL) approach called the Q-learning Algorithm.

5.2.1 Q-learning

The following terms are defined in relation to the problem we are solving using Q-learning.

5.2.1.1 State Space:

In this work, at the beginning of each time slot (t), the local controller receives an information vector from all fog nodes under its controller

$$X(\mathrm{t}) \triangleq [\, X_1(\mathrm{t}), X_2(\mathrm{t}), \dots \dots, X_i(\mathrm{t})], t \;\in [1, 2, \dots, T]$$

Where $X_i(\mathrm{t})$ is defined as.

$$X_i(\mathrm{t}) \triangleq [\, Q_i(t), X_i^{Q}(t), LQ_i(t), X_i^{LQ}(t), HQ_i(t), X_i^{HQ}(t)]$$

Where $Q_i(t)$, $X_i^{Q}(t)$, $LQ_i(t)$, $X_i^{LQ}(t)$, $HQ_i(t)$, and X_i^{HQ} represent arrival queue, virtual arrival queue, local queue, virtual local queue, help queue, and virtual help queue. More details of Actual queues and virtual queues are described in chapter 4, sections 4.5 and 4.7. Each component plays a significant role in characterising the state of the fog node at a given time slot (t). For example, the length of the arrival queue represents the number of incoming tasks from IoT devices awaiting processing. A longer arrival queue signifies a higher flow of tasks, which in turn is a sign of congestion. For the virtual queue associated with the arrival queue, it mainly motivated by its function in overseeing and controlling latency within the fog computing ecosystem. This is an important factor to take into account, especially if there is just one task in the real arrival queue. This single task could have been waiting for several time slots. Thus, failing to include the virtual arrival queue in the state vector will lead to the task being dropped due to extended waiting and exceeding its deadline. This concept is applied to the remaining queues.

Then, to determine the state of fog node i, the local controller calculates the sum of all queue lengths as

$$S_i = Q_i(t) + X_i^{Q}(t) + LQ_i(t) + X_i^{LQ}(t) + HQ_i(t) + X_i^{HQ}$$

The outcome of S_i represents a numerical value by which the local controller classifies the fog node. The fog node's state can be categorized as empty, Stage-One low congestion, Stage-Two low congestion, Stage-One Medium congestion, Stage-Two Medium congestion, Stage-One High congestion, and Stage-Two High congestion. Table 5.1 shows the classification of fog nodes based on the total size of tasks in their queues, where i is the total size of tasks.

Table 5.1: classification of fog nodes based on their state.

Classification	S_i value
Empty	*i* = 0
Stage-One Low congestion	0 < *i* < 50
Stage-Two Low congestion	50 < *i* < 100
Stage-One Medium congestion	100 < *i* < 150
Stage-Two Medium congestion	150 < *i* < 200
Stage-One High congestion	200 < *i* < 250
Stage-Two High congestion	*i* >= 250

Each local controller will create a state vector of the fog nodes under its control. The observed state is represented as a vector **<A, B, C, D, E, F, G>**, where A is the number of empty fog nodes in the cluster, B is the number of Stage-One Low congestion fog nodes, and so on. For example, the state **<5,0,0,0,0,0,0>** means that all five fog nodes, which is the maximum number of fog nodes in one cluster, are empty, and state **<1,0,0,1,3,0,0>** means that one fog node is empty, one fog node is Stage-One Medium congestion, and three fog nodes are in Stage-Two Medium congestion. The state **<0,0,0,0,0,0,5>** means that all fog nodes are Stage-Two High congestion. The size of the queues is limited in each fog node which makes the number of states finite, thus making the use of the Q-learning algorithm feasible.

5.2.1.2 Action Space:

The action here is to determine the number of fog nodes that will be put in sleep mode after observing the current state. The action is determined by the total number of fog nodes in the system.

$$\boldsymbol{a(t)} \in \boldsymbol{A} = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \ldots\ldots, \boldsymbol{K} - \mathbf{1}\}$$

Where K is the total number of fog nodes in a cluster, and zero means that all fog nodes are active. We prevent the learning agent from putting all fog nodes in sleep mode in any time slot (t). This is to ensure the processing of tasks in each time slot and prevent unwanted congestion that threatens the stability of the system which is a concern in the second-sub problem, which is described in more detail in chapter seven.

5.2.1.3 Reward Function:

After taking an action, the local controller will receive a reward based on its action. In defining the reward function, it should be connected to the objective function [69, 70]. Accordingly, the objective function of the problem here is to maximise the number of sleeping fog nodes. This is done to save energy consumed of the fog system. In RL, the learner aims to maximise the received reward. However, if we don't put any constraints, the learner will put the maximum allowed number of fog nodes in its cluster in each time step in a sleep mode. Putting all fog nodes in sleep mode will impact the QoS requirements of the user. In this case, we should add some punishment [96].

In the reward design, the punishment component is essential. This is because it helps to balance the decision-making process in the fog computing systems. It serves as a discouragement to unwanted behaviours that can to poor performance which results in higher energy consumption and costs. Incorporating punishment will promote appropriate decision-making, which is in accordance with dynamic fog computing's goals of resource optimisation, energy efficiency, and cost savings.

As the reward function consists of two parts, it is idle to normalise the value of each part. This is to ensure fairness and consistency in the decision-making process of the agent. As a result of normalizing the rewards, we are able to prevent the agent from becoming unduly influenced by one attribute of the reward function over another, resulting in suboptimal behaviour.

So, the reward function is calculated as following:

$$\boldsymbol{R(s(t), a(t)) = \omega} \left(\frac{\sum_{i=0}^{K} \alpha_i \, FN_i}{K} \right) - \boldsymbol{(1 - \omega)P} \tag{5.2}$$

α_i is a binary variable that represents the status of fog node FN_i, if $\alpha_i = 1$, this means that fog node FN_i is in sleep mode, and FN_i is active if $\alpha_i = 0$**.** $\boldsymbol{\omega}$ is a weight parameter its value ranges between [0, 1], it determines the relative importance of the two factors in the calculation of the reward function: the number of sleeping fog nodes and the punishment $\boldsymbol{P}$. The total number of fog nodes in one cluster is $\boldsymbol{K}$. The value of $\boldsymbol{\omega}$ is determined based on the observed state by the agent, and it is described in more details in the following section.

When putting a fog node in sleep mode, the delay of tasks increases as they wait in the queue. In some cases, this will result in drooping of tasks by violating their deadlines. However, as tasks in our system can wait several time slots to be executed, we cannot measure the immediate delay or drooping of tasks that is caused by the previous action taken in time slot (t-1). This is because the delay and the drooping of tasks might be affected by a previous action taken in time slot (t-x). In this case, to measure the immediate effect of putting a set of fog nodes in a sleep mode in the system, we enforce the sleeping fog node to pay cost when offloading its tasks to active fog node/s to be processed. Additionally, as the sleeping fog node offloads its tasks to active neighbours, this will increase the energy on the active fog nodes side. This is because of processing tasks that belongs to sleeping fog node/s. In this case, the punishment is calculated as

$$P = \left[\sigma\left(\frac{\left(\frac{\sum_{i\in K} E_i(t)}{K}\right)}{E_{max}(t)}\right) + (1-\sigma)\left(\frac{\left(\frac{\sum_{i\in SN} cost_i(t)}{SN}\right)}{cost_{max}(t)}\right)\right] \tag{5.3}$$

The trade-off parameter σ indicates the importance of normalised energy consumption over normalised costs. $E_{max}(t)$ and $cost_{max}(t)$ are the maximum total energy spent by a single fog node and the maximum cost allowed for each fog node respectively.

The cost spent by each sleeping fog node is correlated with the number of tasks offloaded to active neighbours, and it is calculated as

$$cost_i(t) = U * T^{SN}$$

Where U is the unit/task and T^{SN} is the total tasks offloaded from a sleeping fog node. Sleeping fog nodes will not offload all of its tasks as it is limited to the maximum budget. More details about the process of offloading tasks are in chapter seven.

5.2.1.4 Shaped Reward Function

In this work, reward shaping is used to reflect the status of the current state in the calculation of the reward function by introducing a trade-off parameter $\boldsymbol{\omega}$. It is employed to dynamically balance the trade-offs between service quality and resource conservation, enabling the agent to make actions that are in line with the particular goals and environmental circumstances at any given time.

The value of $\boldsymbol{\omega}$ changes dynamically and is influenced based on the environment the agent observes. There are several reasons why the use of dynamically changing $\boldsymbol{\omega}$ is crucial in this environment. This is because it makes it possible to adjust in real time to the constantly changing environmental circumstances. Fog computing environments are by nature dynamic, experiencing variations in resource availability, traffic, and congestion. Introducing a dynamic weighting factor ensures that the system can adapt its resource allocation and decision-making processes in response to these changing circumstances. This adaptability results in improved QoS, cost savings, energy efficiency, and resource utilisation that is optimised.

On the other hand, a static weighting factor could negatively impact the system due to its inflexibility in adapting to the dynamic environmental conditions, a key characteristic of fog systems. One of these consequences is the failure of adopting to the changing traffic patterns, resource availability, or congestion levels. The lack of flexibility in the system may result in poor performance, as it can fail to adjust resource utilisation with actual requirements. This might lead to higher energy consumption and costs during periods of low demand or missed energy-saving opportunities when available.

For example, if the environment is extremely congested, the value of $\boldsymbol{\omega}$ will be set close to 0, which means that the importance of the total energy and cost in the system is higher compared to the number of sleeping fog nodes. In this case, the agent will receive a lower reward for keeping more fog nodes sleeping and a higher reward for keeping more fog nodes active, which will encourage the agent to reduce energy consumption and total cost.

In another case, where the state of the environment is slightly empty or fully empty, the value of $\boldsymbol{\omega}$ is set close to 1, which means that the importance of the number of sleeping fog nodes is higher compared to the total energy in the system and the cost. In this case, the agent will receive a higher reward for keeping more fog nodes sleeping and a lower reward for keeping more fog nodes active, which will encourage the agent to keep more fog nodes sleeping.

In this work, the learning agent after observing the environment regarding fog nodes status and creating the state vector, the agent uses a lookup table to find the best value of $\boldsymbol{\omega}$**.** The lookup table is a decision table that has been predetermined and maps particular state vectors to matching values of $\boldsymbol{\omega}$**.** Based on the present state of the system and environment, this approach enables the system to optimise its decisions and resource allocations in real-time to meet specified objectives, such as improving QoS or minimising energy usage.

An example of the lookup table of values of $\boldsymbol{\omega}$ is shown in the following.

State	**Value of $\boldsymbol{\omega}$**
<5,0,0,0,0,0,0>	1
<4,1,0,0,0,0,0>	0.96
…………	
<0,0,0,0,0,1,4>	0.05
<0,0,0,0,0,0,5>	0

The creation of the lookup table takes into account all possible states that can be encountered in the environment, and it is finite, given that the number of fog nodes in each cluster is limited. Using a ranking method, the system ranks the states according to the degree of congestion in order to systematically prioritise these states. This places the state with empty fog nodes at the top of the lookup table, followed by the slightly congested states, progressing towards the most congested state. Following that, the highest value of $\boldsymbol{\omega}$ will be associated with the empty state and gradually decreased as the level of congestion intensifies.

To solve the problem stated in equation (5.1), the problem is transformed into a constrained optimisation problem as follows:

$$\max \frac{1}{T}\sum_{t=0}^{T-1} R(s(t), a(t)) \qquad (5.4)$$

s.t C1: $Reward_j\ (t) \geq Reward_{min}\ , \forall\ j\ \in X$

C2: $cost_i < cost_{max}\ , \forall\ i \in K$

Where X is local controllers and K is fog nodes. The aim here is to maximise the mean amount of rewards that are received. Constraint C1 is to ensure that the received reward in a time slot (t) for a learner/local controller j is bigger than or equal to the allowed minimum reward in the system. Constraint C2 is to ensure that the cost of each sleeping fog node offloading its tasks is below a threshold value.

5.2.2 Proposed Approaches

We proposed CQL-EUAs algorithm to solve the problem stated in equation (6.4) which combines Cooperative Q-learning (CQL) algorithm along with Eliminating Unacceptable Actions (EUAs) algorithm. Further details are provided below.

5.2.2.1 Cooperative Q-learning (CQL) algorithm

In our work, learning agents (local controllers) don’t share their experiences represented by their Q-tables in a direct manner with each other, as they are only responsible for managing the resources of the fog nodes under their control. Additionally, this process is repeated after each episode, which adds an additional burden to local controllers. Here we assign this mission to the main controller who receives all the Q-tables from all local controllers and then analyses these data, and finally generates one optimised Q-table. After generating the optimised Q-table, it is distributed to all local controllers. In BEST-Q algorithm, the best Q-value is calculated using the following equation.

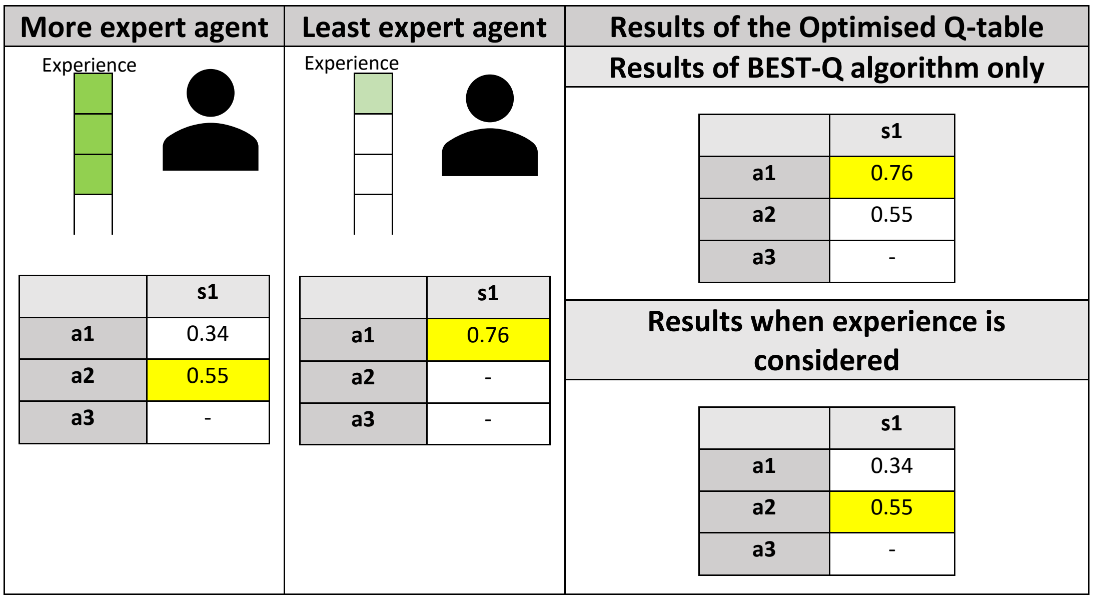
$$Q_{optimised\ table}\ (s,a)\ \leftarrow\ Q_j^{best}(s',a')\ \text{ if s = s' \& a = a'} \qquad [78]$$

Where j is the local controller that has the best Q-value for a state $\boldsymbol{s'}$ for an action $\boldsymbol{a'}$. In addition to applying BEST-Q algorithm, we added the following two features when creating the optimised Q-table.

- BEST-Q algorithm does not consider the fact that if an individual learner has encountered a new state that has not been encountered by other learners. In such a situation, the Q-values corresponding to that state is added to the optimised Q-table. Sharing information regarding this state with other learners is extremely beneficial, regardless of the fact that they have never experienced it before, since the information can be applied to future situations when they encounter it.
- Additionally, before applying BEST-Q algorithm, we first compare the level of experience between learners. In other words, a learning agent is said to be more expert than other learners, if this learning agent has explored more actions for a

given state. In this case, BEST-Q algorithm is only applied if the learning agents have the same level of expertise.

The reason for prioritising the level of experience an agent has over the BEST-Q algorithm is because in some circumstances this might mislead agents and drift them from performing the best action in a given state. The example of misleading agents that is caused by only applying the BEST-Q algorithm is demonstrated in Figure 5.1. In Figure 5.1, it can be seen that for state $s1$, the highest Q-value is gained when performing action $a1$. This is located in the Q-table of the least expert agent. However, this agent did not perform action $a2$ when encountering state $s1$. In this case, applying BEST-Q algorithm will result in adding the state-action pair $(s1, a1)$ that belongs to the least expert agent to the optimised Q-table. However, when considering the level of experience the agents have, it is ideal to perform $a2$ for the state $s1$, and adding the state-action pair $(s1, a2)$ to the Optimised Q-table.


More expert agent
Experience
Least expert agent
Experience
Results of the Optimised Q-table
Results of BEST-Q algorithm only
Results when experience is considered

More expert agent:

	s1
a1	0.34
a2	0.55
a3	-

Least expert agent:

	s1
a1	0.76
a2	-
a3	-

Results of BEST-Q algorithm only:

	s1
a1	0.76
a2	0.55
a3	-

Results when experience is considered:

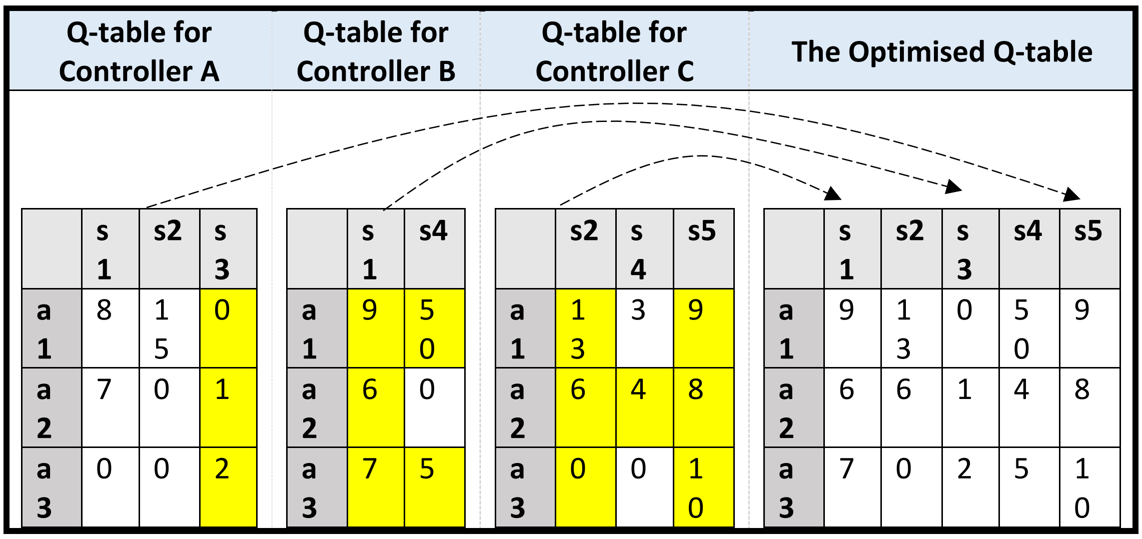
	s1
a1	0.34
a2	0.55
a3	-

Figure 5.1: Comparison example between BEST-Q only and when experience is considered in the creation of the Optimised Q-table.

Regarding the creation of the optimised Q-table, Figure 5.2 shows an example of this. When creating an optimised Q-table, the main controller in the system compared the states in Q-tables and the level of experience in local controllers (Figure 5.3). Furthermore, the main controller analyses the first state in controller A, which is named "s1", this state also exists in controller B. Controller B is more expert than controller A as controller A has explored two actions called "a1" and "a2" while controller B has applied the three actions, "a1", a2", and "a3". In this case, the main controller takes the state "s1" and all its corresponding Q-values from the Q-table that belongs to controller B and adds it to the optimised Q-table. This is also applied to state "s2", where controller C is more expert than controller A.

In regard to state "s3", this state only exists in controller A, then it is moved directly to the optimised Q-table. For state "s4", both controllers B and C have the same level of expertise, in this manner, we apply BEST-Q algorithm, for action "a1" the highest Q value is noted in controller B, which is 50, and for action "a2", the highest Q -value is 4 and it belongs to controller C, whilst for action "a3", controller B has the highest Q-value. Finally, state "s5" only exists in Controller C, thus it is moved directly to the optimised Q-table.

At the end of creating the optimised Q-table, all learners will have the same Q-table. Following on from this, each individual learner will exploit its updated Q-table and add to it the experience they gain from exploring their environment. The process is repeated until reaching the termination episode. This algorithm is referred to as Optimised Q-table (OQT) algorithm and is shown in algorithm 5.1.



Q-table for Controller A

	s1	s2	s3
a1	8	15	0
a2	7	0	1
a3	0	0	2

Q-table for Controller B

	s1	s4
a1	9	50
a2	6	0
a3	7	5

Q-table for Controller C

	s2	s4	s5
a1	13	3	9
a2	6	4	8
a3	0	0	10

The Optimised Q-table

	s1	s2	s3	s4	s5
a1	9	13	0	50	9
a2	6	6	1	4	8
a3	7	0	2	5	10

Figure 5.2: Example of optimisation Q-table

Algorithm 5.1: Optimised Q-table (OQT) algorithm.

```
                 Optimised Q-table (OQT) algorithm
 INPUT    $Q\_table_0, Q\_table_1, Q\_table_2, ……., Q\_table_j \; j \in X$
 OUTPUT   Optimised Q_table
 While (No_episode < Termination_episode)
     Calculate experience for all $j \in X$
     Analyse Q-tables
     For each $state_i$ do
     IF ($state_i = state_j$) -> the state is experienced by $agent_i$ and $agent_j$
         IF ($agent_i\_experience > agent_j\_experience$)
             addOptimisedTable ($state_i$, $Q\text{-}values_i$)
         Else IF ($agent_j\_experience > agent_i\_experience$)
             addOptimisedTable ($state_j$, $Q\text{-}values_j$)
         Else IF ($agent_j\_experience = agent_i\_experience$)
             Apply BEST-Q [78]
             updateOptimisedTable
     IF ($state_i \neq state_j$) -> the state is only experienced by $agent_i$
         addOptimisedTable ($state_i$, $Q\text{-}values_i$)
     Send Optimised Q-table to local controllers
```

5.2.2.2 Eliminating Unacceptable Actions (EUAs) Algorithm

To enhance the search speed in the Q-learning method and fasten the learning process for the agents, **Eliminating Unacceptable Actions (EUAs)** method is proposed. This method will help the learning agent to remove an unacceptable action or a sequence of actions that lead/s to unwanted rewards from the action space for a given state. By doing that we can minimise the search space for a given state and speed the processes of finding the optimal action. This enhancement improves the performance of pure Q-learning.

In the Q-learning algorithm, each agent has a Q-table to store the Q-values for the state-action pairs. Regarding EUAs algorithm, each learning agent has an additional table called "updated action space table" to keep track of the updated action space for each state. For instance, if a learning agent observed the state **<0,0,5,0,0,0,0>,** and the action applied is based on the Epsilon-greedy strategy is to put four fog nodes in a sleep mode. If the received reward caused by this action has exceeded the minimum allowed reward, the agent will remove action four from the action space related to this state, this is shown in Figure 5.3. Given another example, if the encountered state is **<0,1,0,0,1,3,0>,** and the agent has applied action three, that means putting three fog nodes in a sleep mode, and again the received reward is not acceptable, the agent will update the action space by removing action three and all higher actions, which is action four in this case.

The algorithm of EUAs is shown in algorithm 5.2.

State	**<5,0,0,0,0,0,0>**	**<4,1,0,0,0,0,0>**	**<0,0,5,0,0,0,0>**	**…….**	**<0,1,0,0,1,3,0>**
Action space	{0, ..., k-1}	{0, ..., k-1}	{0, ..., 3}	{0, ..., k-1}	{0, ..., 2}

Figure 5.3: Example of updating the action space table.

Algorithm 5.2: Eliminating Unacceptable Actions (EUAs) algorithm.

```
Eliminating Unacceptable Actions (EUAs) algorithm
INPUT     ($state_i$, $action_i$) $state_i \in$ S, $action_i \in$ A
OUTPUT    Updated Action Space Table
    While ($action_i \leq$ K-1)
        Update action space table ($state_i$, $action_i$)
        Increase $action_i$
```

5.2.2.3 The proposed CQL-EUAs algorithm

The proposed algorithm combining cooperative Q-learning algorithm (CQL) and Eliminating Unacceptable Actions algorithm (EUAs) is shown in algorithm 5.3. in this work, each learning episode lasts for up to the 500th time step. The length of each time step is 5 ms and in each time step each learner will encounter a state from its environment that might be different from other learners. The episode terminates for the following reasons:

1- An agent receives a reward that is below the minimum allowed reward.
2- The number of time slots in one episode has reached its maximum allowed time per episode, which is 500 time slots.
3- The number of episodes has reached to the termination episode number, which is 500 episodes.

At the beginning of each episode, each learning agent sends its Q-table to the main controller and the **OQT** method is called, this is shown in line 3**.** If the number of time slots in the episode has not reached its maximum, the agent observes its state S(t) and receives its reward R(t) in lines 5 and 6. From line 7 to 20, if the received reward is not acceptable, the EUAs method is invoked. Then the agent calculates its average rewards, and the value of ε decreased if it is higher than the minimum value, and the number of episodes increases, and the episode terminates.

From lines 21 to 31, if the received reward is acceptable, the agent will update its Q-table based on equation (5.4), and then make its action. The selection of an action considers the available actions from the action space and can be a random action or an optimal action with the highest Q-value based on the **Epsilon-greedy strategy**. If the applied action should be selected optimally, and however, the state has not been encountered before, the agent will select any action randomly. Then, the number of time slot increases. From lines 32 to 36, if the number of time slots has reached its maximum, the agent will calculate its average rewards and the epsilon value will decrease if it is higher than the minimum threshold. Then the number of episodes increases, and the episode terminates. This is repeated until the maximum number of episodes has been reached (e.g., 500 episodes).

Algorithm 5.3: The proposed CQL-EUAs algorithm

```
CQL-EUAs algorithm
INPUT   $\alpha$: learning rate, $\gamma$: discount factor, $\varepsilon$: epsilon = 1
OUTPUT  Updated Q (s, a)
1  While (no_episode < termination_episode)
2    $t = 0$
3    OQT (Q_table)
4    While ($t < T$)
5      Observes the state S (t)
6      Receives reward R (t) of state S (t-1) based on equation (5.2)
7      IF (reward <= X)
8        EUAs (S (t-1), a (t-1))
9        Calculate $Average\ Reward = (total\ rewards / t)$
10       IF ( $\varepsilon$ > Minimum value)
11         $\varepsilon = \varepsilon$ – minimise
12       Increase no_episode
20       End Episode, Go to line 1
21     Update Q-table (S (t), R (t), S (t-1)) based on Equation (2.1)
22     Generate Z in range [0,1] randomly
23     Choose a ∈ A(a) using $\epsilon$-greedy algorithm
24     IF (Z <= $\epsilon$)
25       Select a randomly from A
26     Else
27       IF (S(t) ∈ Q (s, a))
28         a = arg $\max_a$ Q (s, a)
29       Else
30         Select a randomly from A
31     $t = t + 1$, Go to line 4
32    Calculate $Average\ Reward = (total\ rewards / t)$
33    IF ( $\varepsilon$ > Minimum value)
34      $\varepsilon = \varepsilon$ – minimise
35    Increase no_episode
36    End Episode, Go to line 1
37 End Training Episodes
```

5.2.3 Experiments

Based on the proposed scheme, there are several parameters and aspects that influence the results of learning and show the effectiveness of the proposed scheme. In this case, we examine the impact of various epsilon approaches during the learning process. Moreover, the effect of the minimum allowed reward that causes episode termination is investigated with various values. Also, the impact of varying the average arrival rate for each type of task generated from the IoT devices is addressed. Furthermore, the impact of the proposed EUAs method on the learning process is analysed. Finally, we investigated how cooperative learning would influence the learning process for agents compared to single learning, where cooperation is not considered.

In each set of experiment, there are three main parameters that impact the results. These parameters are the maximum allowed received reward, the average arrival rate of tasks, and the epsilon approach. When running each experiment, we vary one parament while making the other two constants. The set of conducted experiments is shown in Table 5.3.

5.2.3.1 Environment

In all sets of experiments, the same environment is applied. Additionally, local controllers are learning simultaneously, and in each time step, each local controller might encounter a different state from others. Moreover, the action taken by a single local controller does not impact the environment of other local controllers. The value of the parameters considered in the experiments is shown in Table 5.2.

Table 5.2: simulation settings.

Parameter	Value
Length of time slot ($\Delta\ t$)	5 ms
Number of clusters	3
Number of FNs in each cluster	5 fog nodes
Maximum number of SN in a cluster	4 fog nodes
Number of IoT devices in each IoT region	10 devices
Average arrival rates of tasks	[1-20], [1-30], [1-40], [1-50] ms
Maximum time slots in an episode	500 time slots
Maximum number of episodes	500 episodes
Minimum reward thresholds	0, -0.05, -0.10, -0.15, -0.20
α: learning rate	0.01
γ: discount factor	0.9
ε: epsilon initial value	1
σ: weight in punishment components	0.3

Table 5.3: Experiment details.

Experiment		Values	Performance metric/s	Other parameters	
1	The impact of selecting epsilon approaches	Epsilon A	Average rewards per episode	[1-40]	Reward >= 0
		Epsilon B			
		Epsilon C			
		Epsilon D			
2	The impact of varying reward thresholds	Reward >= - 0.20	Average rewards per episode	[1-40]	Epsilon A
		Reward >= - 0.15			
		Reward >= - 0.10	Average number of sleeping fog nodes		
		Reward >= - 0.05	Convergence level per episode		
		Reward >= 0			
2	The impact of varying the average arrival rates	[1-20]	Average rewards per episode	Epsilon A	Reward >= 0
		[1-30]	Average number of sleeping fog nodes		
		[1-40]			
		[1-50]	Convergence level per episode		
4	The impact of EUAs approach	EUAs & Epsilon A	Average rewards per episode	[1-40]	Reward >= 0
		EUAs & No Epsilon A			
		No EUAs & Epsilon A			
		No EUAs & No Epsilon A			
5	Difference between Cooperative and Single learning	Epsilon A	Average rewards per episode	[1-40]	Reward >= 0
		Epsilon B	Various epsilon approaches		
		Epsilon C	Convergence level per episode		
		Epsilon D			

5.2.3.2 Impact of Epsilon approaches

We investigate the impact of various Epsilon approaches on how agents are functioning and overall system performance. The Epsilon methods incorporated in this experiment are shown in Table 5.4. In Epsilon A, the experiment starts with epsilon value of 1, which means the agent selects its actions 100% randomly, and then at the end of each episode, the epsilon value will be reduced by 0.001. This process happens continuously until the epsilon value reaches 0.1, upon which the value remains the same for the rest of the training episodes. This is the same for Epsilon B and C, except that epsilon B decreases by 0.01 at the end of each episode, and Epsilon C decreases by 0.1.

In epsilon D, the epsilon value is always 0.1 and remains the same for all the training episodes. Table 5.4 and Figure 5.4 shows the starting value of each epsilon approach, how it decreases, and the end value. In all these approaches, in any given state, if the agent should select a greedy action based on the epsilon value, and this state has not been encountered before, the agent will select its action randomly since the agent has no knowledge about this state.

Table 5.4: Epsilon approaches incorporated in the experiment.

Name	Initial value	Decreases by	Remains at	At Episode
Epsilon A	1	0.001 per Episode	0.1	250
Epsilon B	1	0.01 per Episode	0.1	100
Epsilon C	1	0.1 per Episode	0.1	50
Epsilon D	0.1	-	0.1	-

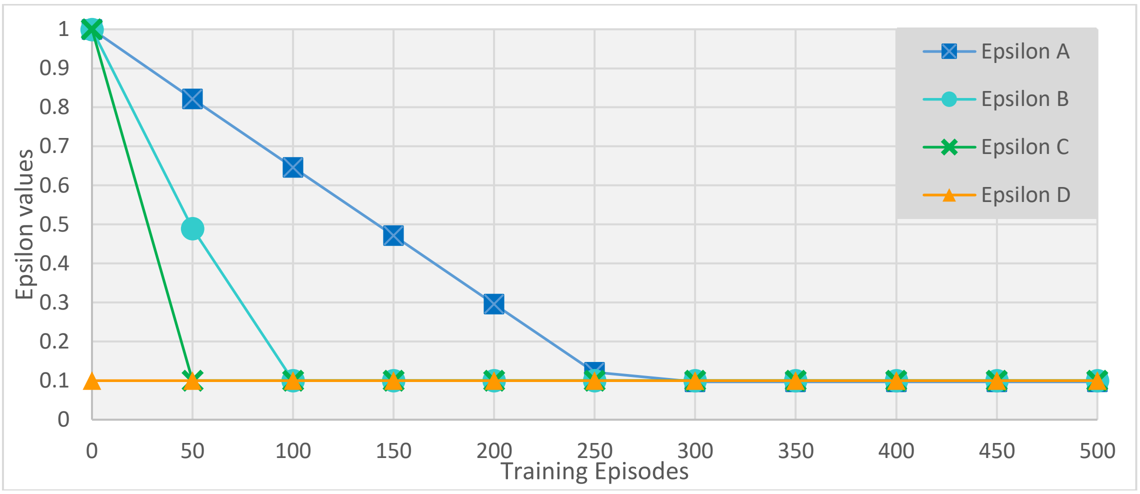


Figure 5.4: Various Epsilon approaches.

Impact on the average rewards per episode

We investigate the impact of various epsilon approaches on the average rewards per episode in the system. Clearly form Figure 5.5, it is clear that various epsilon approaches influence the average rewards per episode on the same manner. Each pattern of these epsilon approaches is divided into two main parts, the first part is when the epsilon value starts with a value of one and then decreases, and the second part is when the epsilon value remains at a value of 0.1.

In the first part, the average rewards per episode starts low as the agents are acting randomly. As the number of episodes increases, the average rewards per episode starts to increase until reaching the second part, where epsilon has a value of 0.1, upon which the pattern starts to converge with slight variations. This is because the agents begin to exploit the knowledge they have gained so far about their environment, acting 90% optimally and 10% randomly. The highest average rewards per episode are shown in Epsilon A, and the lowest is in Epsilon D. In Epsilon A, as the epsilon value decreases by 0.001 which is very low compared to Epsilon B and C, we give the agent more time to explore its environment, thus giving more chances to explore various actions for a given state. In this case, the agent knows exactly which optimal action has a higher Q value that helps to achieve higher rewards. As opposed to Epsilon D, the agent only has 10% chance of exploring its environment and 90% of taking the optimal action found so far, which will lead the agent to stick with its local optimal without exploring better options.

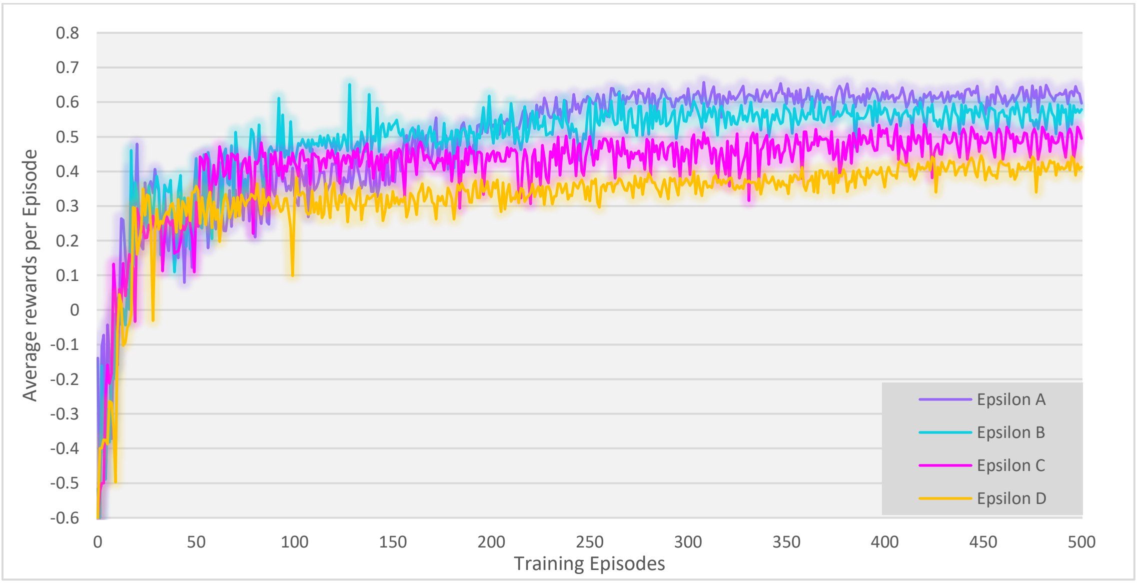


Figure 5.5: impact of epsilon approaches on the average rewards per episode.

5.2.3.3 The impact of varying the minimum allowed received reward on the learning process.

In the learning process, and based on the proposed scheme, one of the reasons that make the episode terminate is when an agent receives an unwanted reward based on its previous action. Based on the reward function in equation (5.2), the highest reward value the agent can receive is 0.8, and this can be achieved if the state of the environment is empty and the learning agent puts the maximum allowed number of fog nodes into sleep mode, where the value of $\omega = 1$, and there is no processing of tasks or cost caused by offloading tasks from sleeping fog nodes. Additionally, the lowest achieved reward is around -0.9, this can occur for example if the state of the environment is very congested and the agent puts three or four fog nodes in a cluster in a sleep mode and thus increasing the total energy and the cost in the system where the value of $\omega = 0$, and the weight $\sigma = 0.1$.

In this context, we examined the impact when setting the minimum allowed reward to -0.20, -0.15, -0.10, -0.05 and 0, and analysed its effect on the average rewards per episode for all learners, average number of sleeping fog node per episode, and the convergence level per episode. To test this set of experiments, the average arrival rates are be set from [1-40], and Epsilon A is applied, and both parameters remain constant for all values of the allowed minimum reward that can be obtained.

Impact on average rewards per episode

From Figure 5.6, it can be seen that the performance of the average rewards per episode for all values is affected by the selected epsilon approach which is epsilon A. With regards to varying the minimum allowed reward, it can be noted that the highest average reward is scored when the minimum reward threshold is set to 0, and this decreases as the allowed reward is decreased, where the lowest average rewards is seen when the minimum allowed reward is set to -0.20. This is because when the minimum allowed reward is set to 0, learners are forced to not have a reward that is below 0, and if this occurs, the episode will end and the action that caused this reward will be eliminated from the action space along with higher actions. This is caused by the EUAs method. However, when the minimum allowed reward is set to -0.20, when agents are exploring their environment, they can still select actions that lead to rewards of -0.20 or higher and only eliminate the actions that cause rewards that are lower than -0.20. Also, it is noted that there is a slight difference between these values in the last 250 episodes.

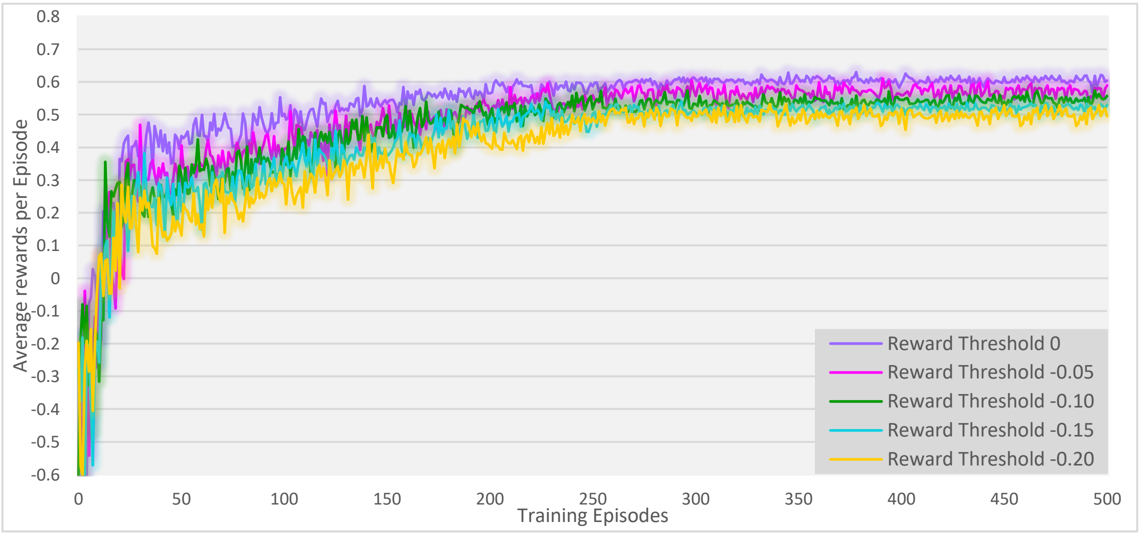


Figure 5.6: Impact of varying the minimum allowed received reward on the average rewards per episode.

Impact on average number of sleeping fog nodes per episode.

We addressed the impact of various values of the allowed minimum rewards on the average number of sleeping fog nodes. In this experiment, the total number of fog nodes in the system is 15 fog nodes, each five fog nodes are located in one cluster, and we have three clusters in total, and in each cluster, there is a learning agent. From Figure 5.7, it can be seen that all reward thresholds exhibit almost the same pattern. In more detail, in the first 250 episodes, all reward thresholds show a pattern of extreme variations, and then in the last 250 episodes, the pattern remains stable with slight oscillations. The reason for the significant fluctuation is that the agent is exploring its environment most of the time in the first 250 episodes and takes random actions. In a given state, when the applied action causes an unwanted reward, the episode will end and the action causing the unwanted reward along with higher actions are eliminated from the action space, giving the agent a narrow selection of actions when facing this state in the future and this is the cause of the convergence in the last 250 episodes. This implies the impact of epsilon approach A.

In addition to that, the least average number of sleeping fog nodes is noted when the maximum allowed reward is set to 0, and as we reduce the minimum allowed reward, the average number of sleeping fog nodes slightly increases and the highest average number of sleeping fog nodes is scored when the allowed reward is set to -0.20.

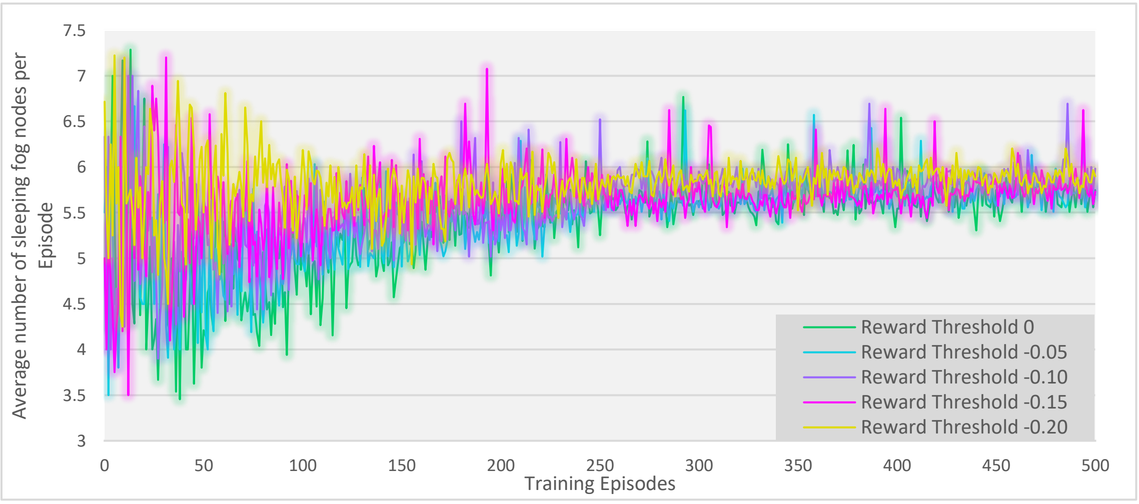


Figure 5.7: Impact of varying rewards threshold on the average number of sleeping fog nodes per episode.

Impact on the convergence level per episode

In Figure 5.8, we analyse the impact of varying the minimum allowed reward on the convergence level in each episode. In our approach if the episode has reached 500 time slots without being terminated, we can say that the learning process had been converged. Convergence means that the learning agent knows its environment by applying actions that meets the strict requirements and constraints. In this regard, we examined the convergence level for the training process when the minimum received reward were set to -0.20, -0.10 and 0.

We noted that the approach when having the minimum received reward set at -0.20, the training process in each episode had more time and did not terminate as fast as when having -0.10 as the allowed minimum reward, followed by the approach when it is set to 0. The reason for that is because when the agent is allowed to have -0.20 as the minimum received reward, the episode is less likely to terminate because of the loose constraint, for example, receiving a reward of -0.21 or less. However, when 0 is set as the minimum allowed reward, this constraint is very tight causing the episode to end sooner, and thus having less time compared to when setting the minimum received reward to -0.20. Regarding the convergence, reward threshold -0.20 converged faster than when the minimum received reward is set to -0.10 and 0. As the agent educates itself faster as a result of having loss constraint, e.g., only terminate when -0.21 or less reward is received.

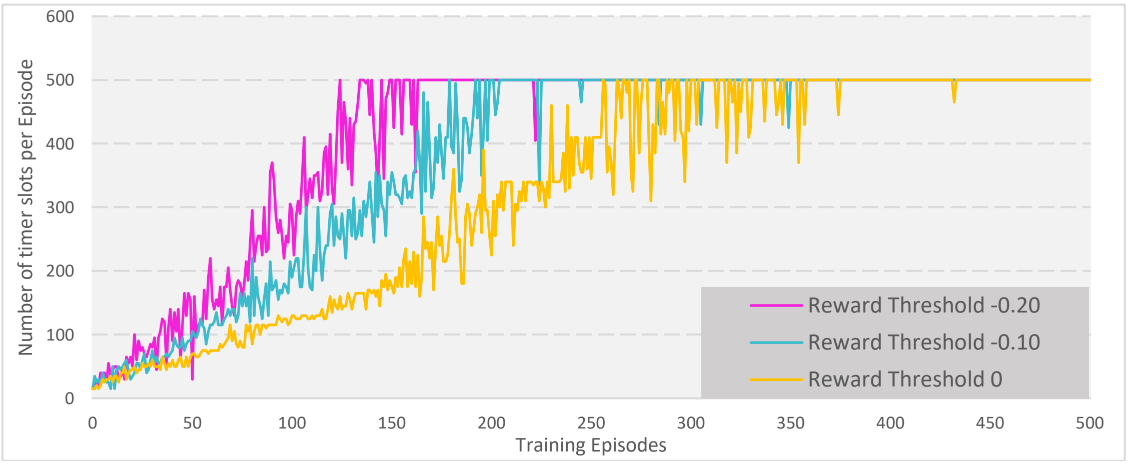


Figure 5.8: Impact of the maximum allowed reward on the number of time slots encountered per episode.

5.2.3.4 The impact of varying the average arrival rates of tasks on the learning process

During these sets of experiments, the minimum reward is set to 0, and Epsilon A is applied to all average arrival rates. In this experiment, we examined the impact when the average arrival rates are [1-20], [1-30], [1-40], and [1-40] ms on the average gained rewards per episode, the average number of sleeping fog nodes per episode, and the convergence level per episode.

Impact on the average rewards per episode

Figure 5.9 demonstrates the impact of increasing the average arrival rates on the average rewards per episode. It can be seen that when the environment is congested as in the average arrival rates of [1-20], the average reward per episode is low compared to when the environment is slightly congested as in [1-30]. The higher average rewards are gained when the average arrival rate is low, in [1-50].

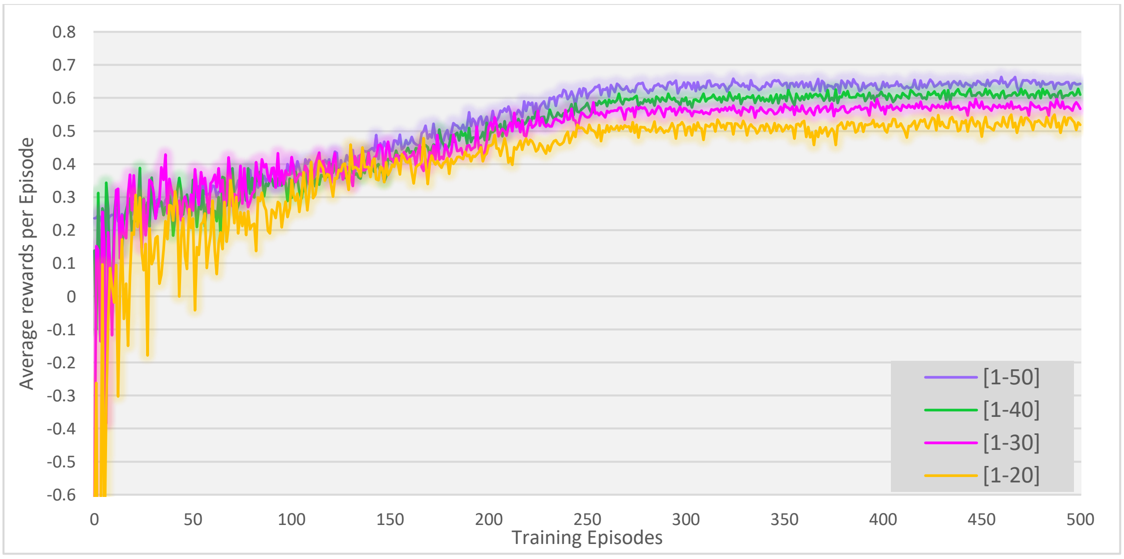


Figure 5.9: impact of varying the average arrival rates of tasks on average rewards per episode.

Impact on average number of sleeping fog nodes per episode.

In Figure 5.10, it can be seen that with the lower congested environment, in [1-50], the average number of sleeping fog nodes is higher compared to other systems when the average arrival rate of [1-40], [1-30] and [1-20] respectively. In extremely congested systems when the average arrival rate is [1-20], the agents manage to put few fog nodes into sleep mode to save the energy whilst meeting system requirements.

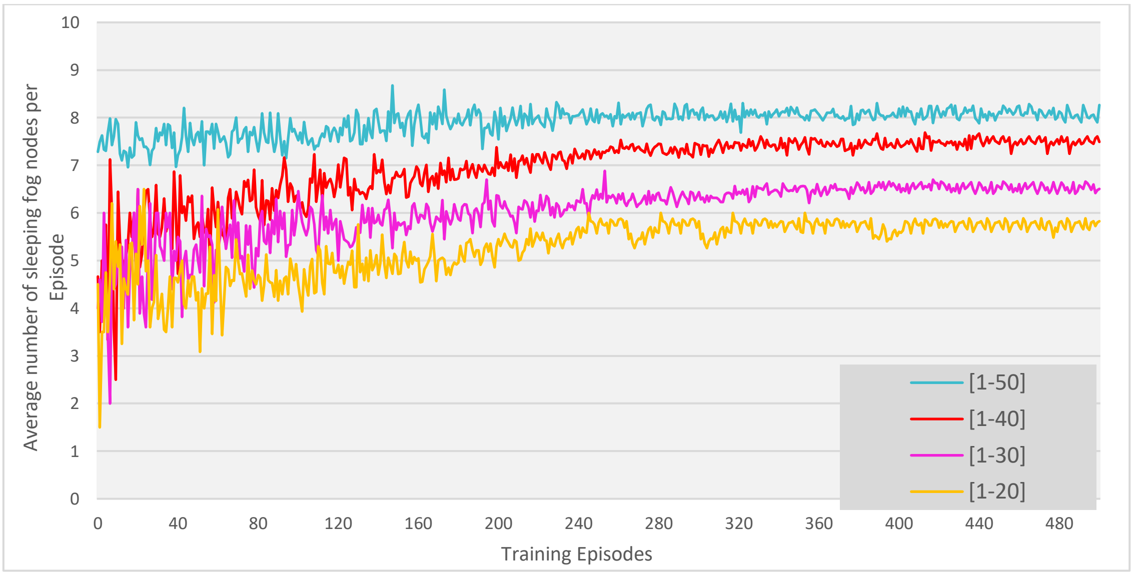


Figure 5.10: Impact of varying the average arrival rates of tasks on average number of sleeping fog nodes per episode.

Impact on the convergence level per episode

In Figure 5.11 it can be concluded that, when the average arrival rates are [1-50], [1-40], [1-30] and [1-20] the approaches exhibit the same pattern of starting low and gradually increasing until they become stable at around 100, 150, 240, and 300 episodes respectively. Additionally, it seems that when the system is not congested as in [1-50], the agents learn faster about their environment, and this is shown as the learning process converges faster. This convergence is caused by the agent learning to avoid actions that lead to unwanted rewards. This is compared to when the system is extremely congested, as in [1-20], as the convergence of the learning process is evident after having a significant number of training episodes.

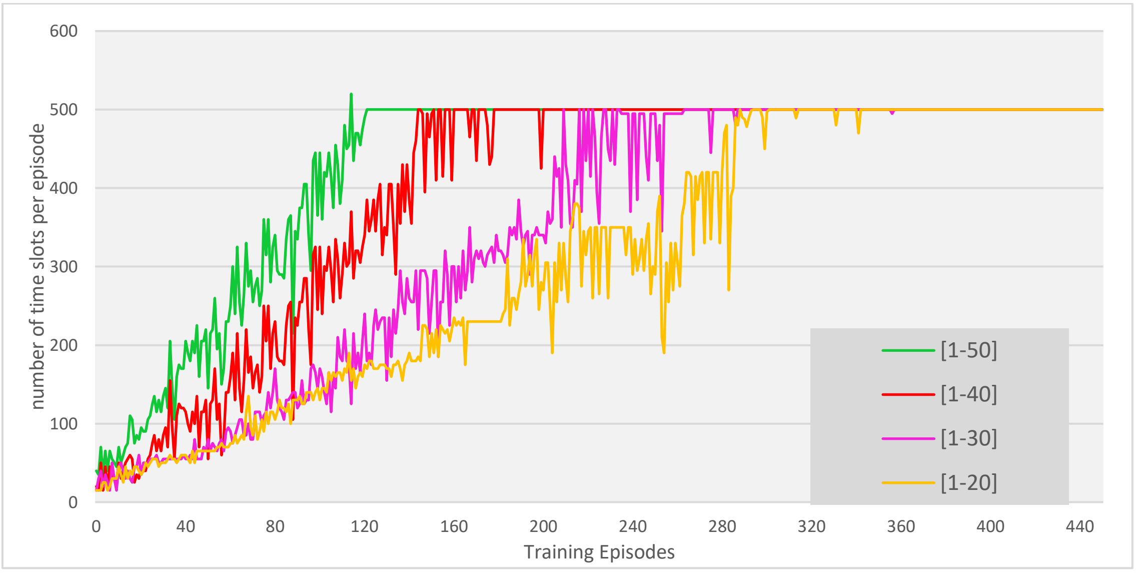


Figure 5.11:Impact of varying the average arrival rates of tasks on the number of time slots encountered per episode.

5.2.3.5 Impact of Eliminating Unwanted Actions (EUAs) method.

This method has been proposed to help the agent learn faster about its environment by eliminating actions that cause inefficient rewards. In this experiment we address the effect of EUAs method on system performance regarding the average rewards per episode in the following scenarios:

- The EUAs & Epsilon A approach: when both EUAs and Epsilon A are applied.
- The EUAs & No Epsilon A approach: we apply the EUAs method without including Epsilon A, so the agent plays random all the time, but the actions that violate the reward threshold will be removed from the action space.
- No EUAs & Epsilon A approach: EUAs are not involved, but Epsilon A is.
- No EUAs & No Epsilon A approach: in this case, EUAs and Epsilon A are not applied.

Both the EUAs method and Epsilon are learning tools that help the agent to learn and explore its environment. Figure 5.12 shows the average rewards per episode for the four stated scenarios.

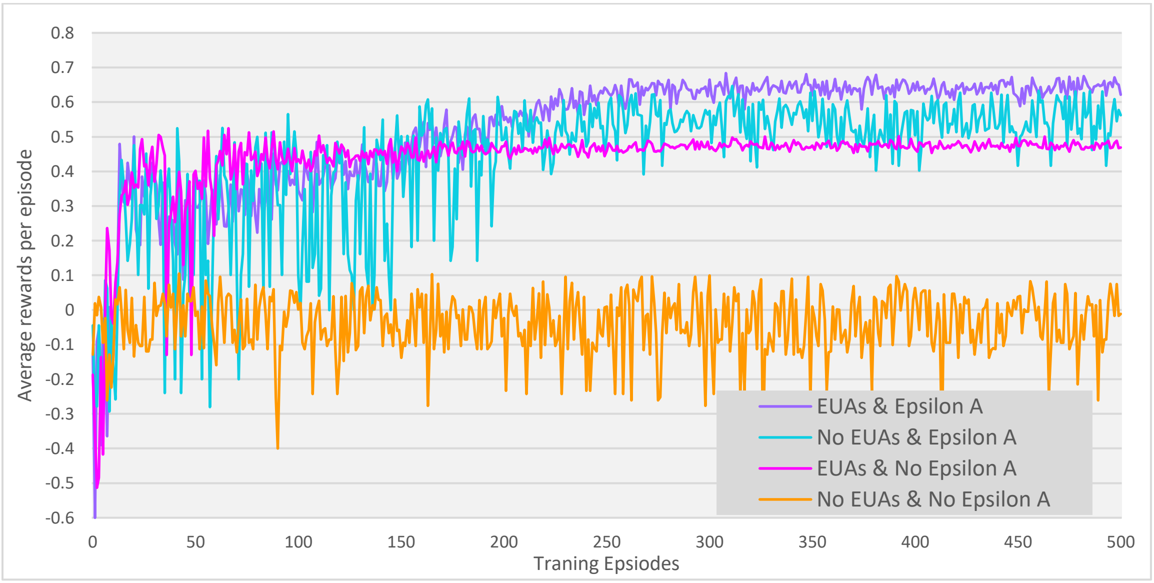


Figure 5.12: Impact of EUAs and Epsilon A on the average rewards per episode

According to Figure 5.12, the Epsilon approach alone does not help the agent learn its environment more effectively in a non-stationary MDP. It is evident when comparing the involvement of the EUAs method in EUAs & Epsilon A approaches to its disengagement in No EBAs & Epsilon A approaches. Additionally, the system performed better in EUAs & Epsilon A compared to in No EUAs & Epsilon A due to having higher average rewards per episode. When examining the importance of involving Epsilon, in EUAs and No Epsilon approach, the agent seems to learn its environment in a stable manner. However, the system performed better with higher average rewards per episode when Epsilon is engaged as in EUAs and Epsilon A approach. When both EUAs and Epsilon are not considered, it seems that the agent is not learning at all. This can be seen from the performance of the system in No EUAs and No Epsilon A approach. To conclude, both EUAs method and Epsilon are learning approaches that aid the agent in the learning process, especially if the environment is non-stationary.

5.2.3.6 Cooperative and single learning

In cooperative learning, learners share the knowledge they have gained so far about their environment after each episode. This is the opposite of single learning, where learners keep the information, they have gained to themselves without sharing it with other learners. In cooperative learning, a variety of actions are explored in a given state. This will result in an enhanced learning process and the ability to achieve optimal results. We examine the impact of cooperative learning and single learning on the average rewards per episode and the number of episodes needed to reach convergence, in other words, how fast the agents are learning. Considering cooperative and single learning, we compare them with Epsilon A, B, C, and D since each of the Epsilon approaches impacts the system differently.

Impact of Cooperative and Single learning on Average rewards per episode

From Figures 5.13, 5.14, 5.15, and 5.16 we can see that the average rewards per episode in cooperative learning is higher than the average rewards per episode in single learning for all epsilon methods.

Also, it is noted that before reaching the convergence level, the average rewards per episode in single learning, fluctuates extremely as compared to cooperative learning. In respect to how various epsilon approaches have an impact on the difference between cooperative and single learning, it can be seen that once the learning converges, the difference between cooperative and single learning is smaller when Epsilon A is involved. This difference starts to increase slightly when Epsilon B is considered and even more when Epsilon C and D is involved. The reason for a small difference between cooperative and single learning when considering Epsilon A, is because, as Epsilon A decreases very slightly in each episode, this gives the learners higher chances to act randomly and explore various actions, thus learning better about their environment compared to other epsilon approaches. The highest difference between cooperative and single learning is noted in Epsilon D. This is because in Epsilon D, the epsilon value decreases sharply and then remains constant at early stages of learning which makes the learners bound to their local optimal, compared to cooperative learning as leaners share their knowledge they can perform better and overcome the local optima of each learner.

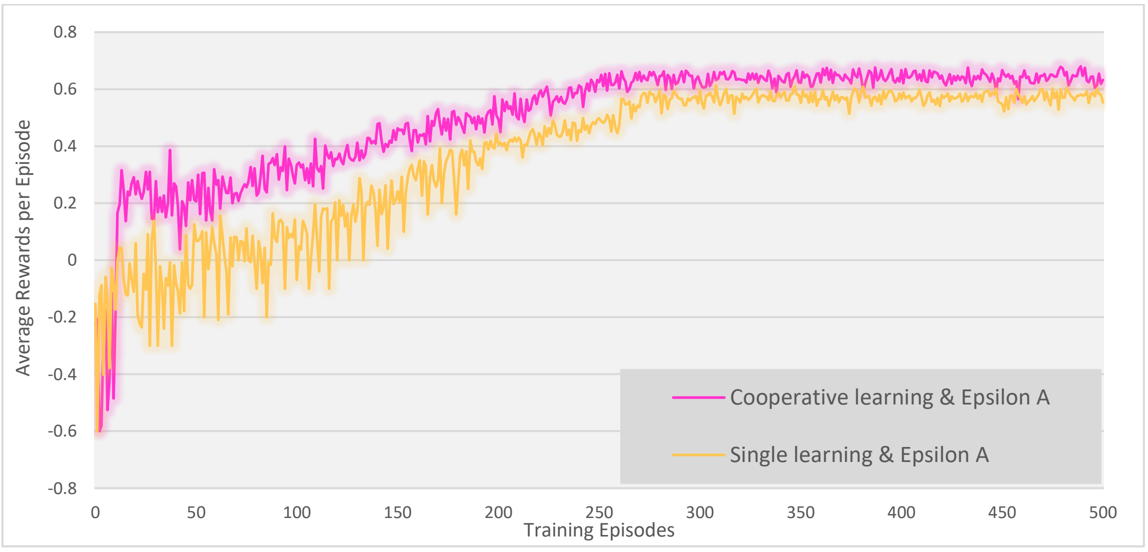


Figure 5.13: Impact of Cooperative and Single learning with Epsilon A on the average rewards per episode

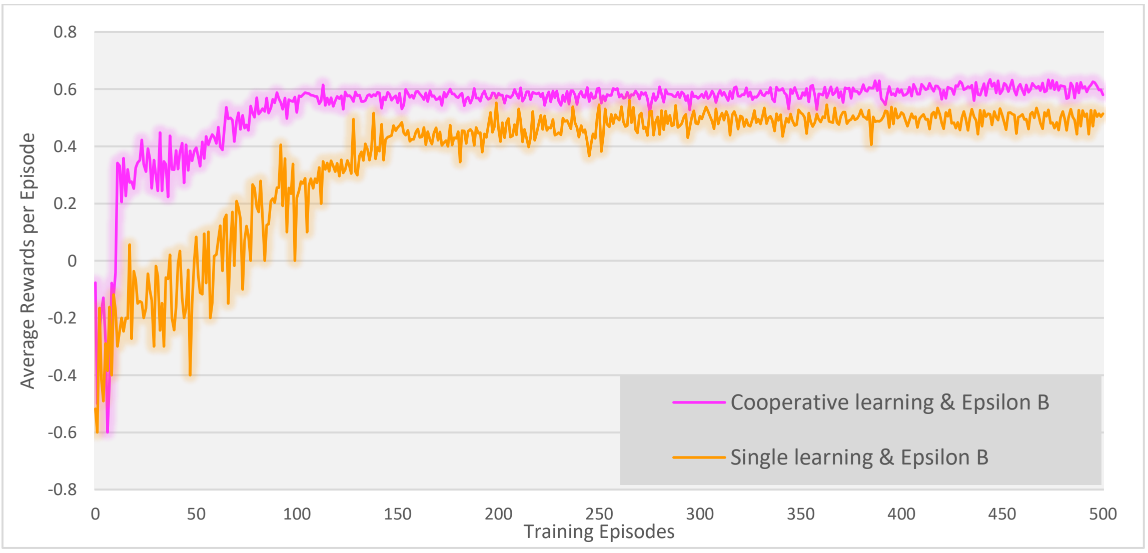


Figure 5.14: Impact of Cooperative and Single learning with Epsilon B on the average rewards per episode

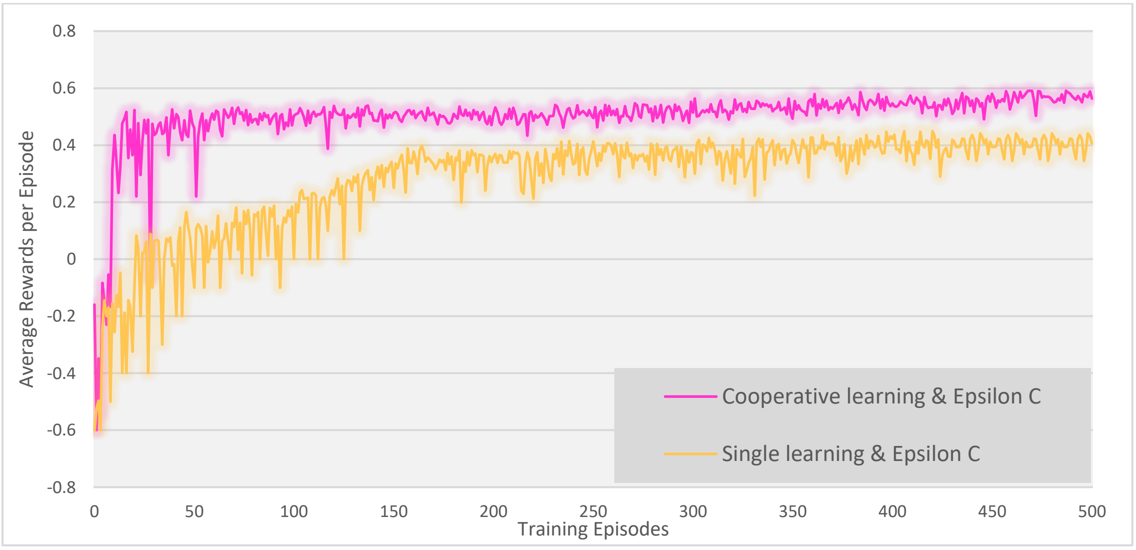


Figure 5.15: Impact of Cooperative and Single learning with Epsilon C on the average rewards per episode

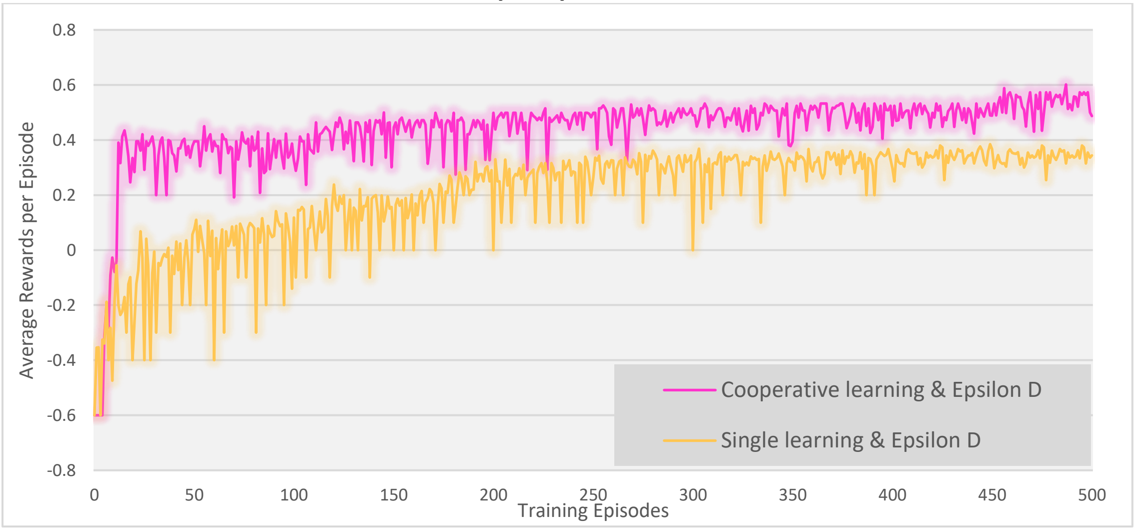


Figure 5.16: Impact of Cooperative and Single learning with Epsilon D on the average rewards per episode

Impact of Epsilon Approaches on Average rewards per episode for Cooperative Learning

The results in Figure 5.17, show that comparing various epsilon approaches, the highest average rewards per episode is shown in Epsilon A and the lowest in Epsilon D. Furthermore, the convergence level after the epsilon value remains constant, Epsilon A remains stable with slight variation, however, Epsilon B, C, and D shows slight increase, and this is noticed in episode 500. It can be said that given a greater number of training episodes, more than 500 episodes, can help the approaches with Epsilon B, C, and D to have higher average rewards similar to that in Epsilon A. This means that cooperation can help to overcome the limitations of the trade-off between exploring and exploiting with various epsilon approaches having higher number of episodes.

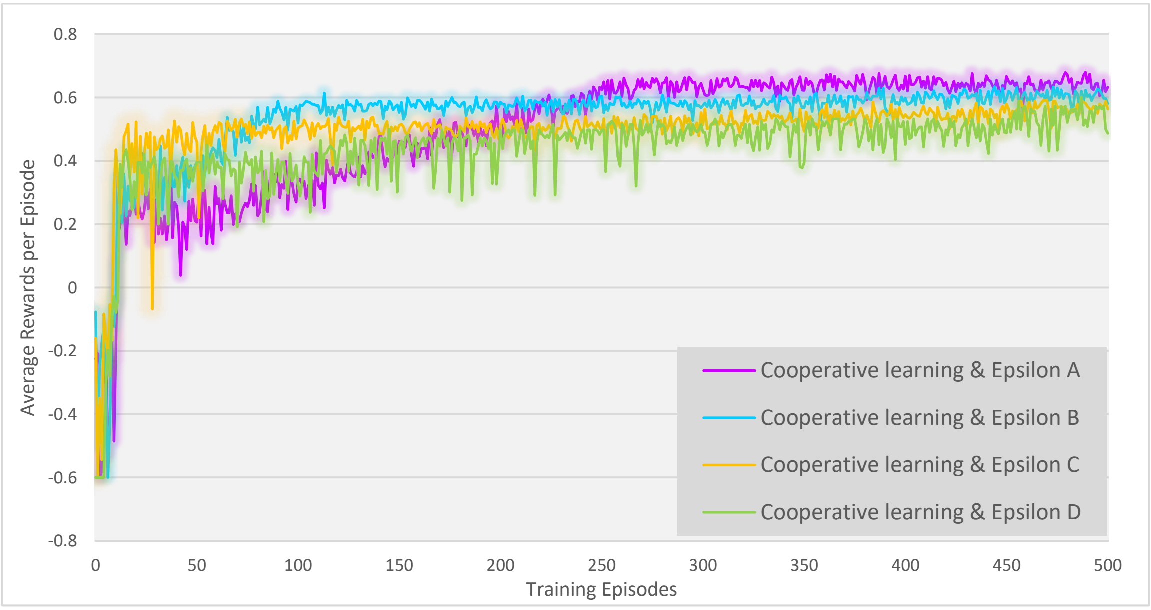


Figure 5.17: Impact of Cooperative learning with various Epsilon approaches on the average rewards per episode

Impact of Epsilon Approaches on Average rewards per episode for Single Learning

The result in this regard is shown in Figure 5.18. If we compare the impact of various Epsilon approaches in cooperative learning to single learning, we can observe that the variation is extremely high with single learning. Additionally, once the epsilon value remains constant all approaches become steady with fluctuation, but no increase is noticed. Moreover, the highest average reward is achieved when epsilon A is involved and this decreases with other approaches, and the lowest average reward is when Epsilon D is considered.

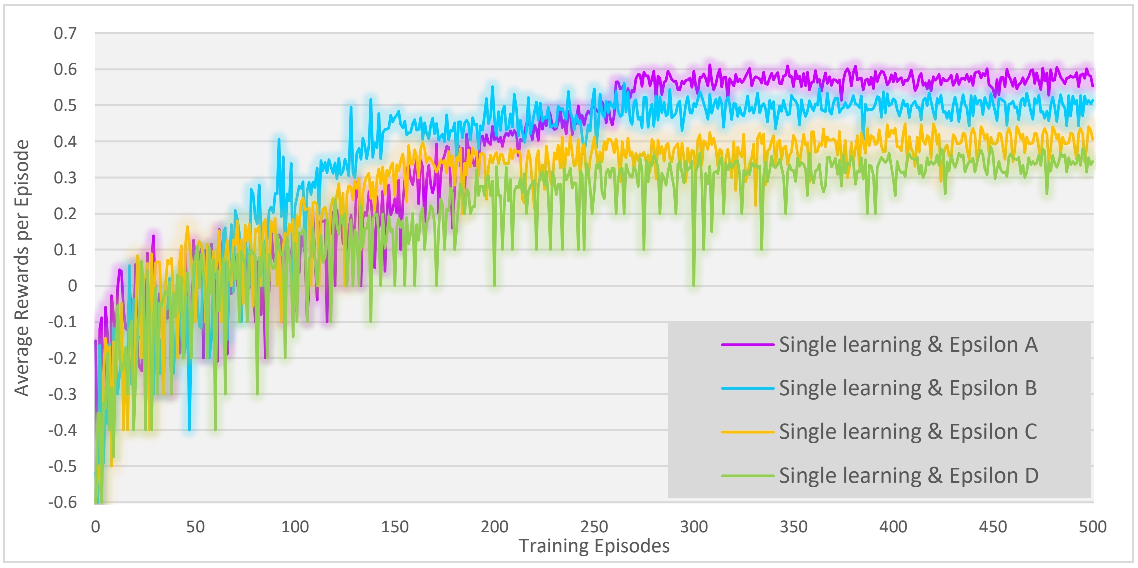
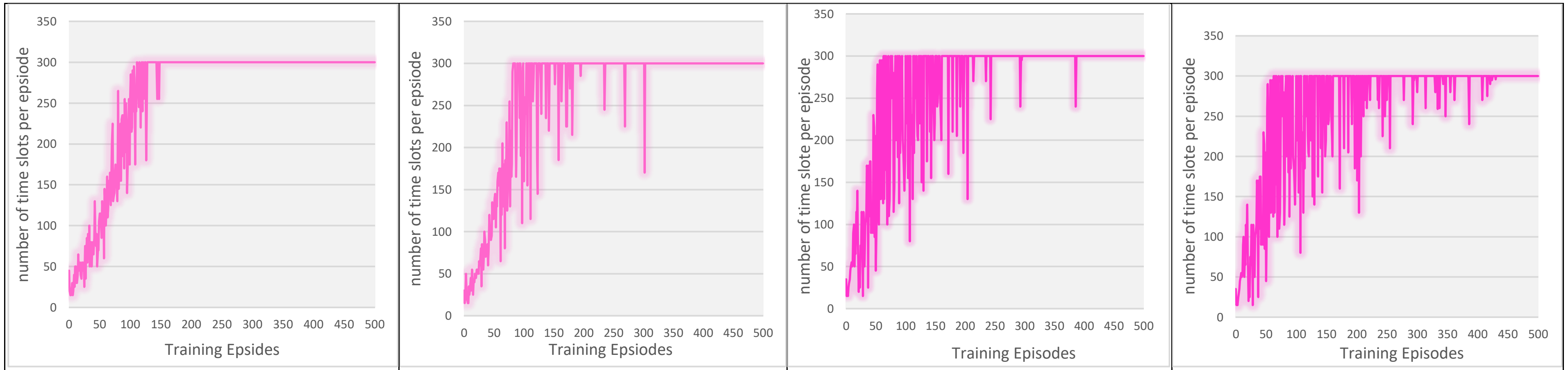


Figure 5.18: Impact of Single learning with various Epsilon approaches on the average rewards per episode

Impact on the convergence level per episode

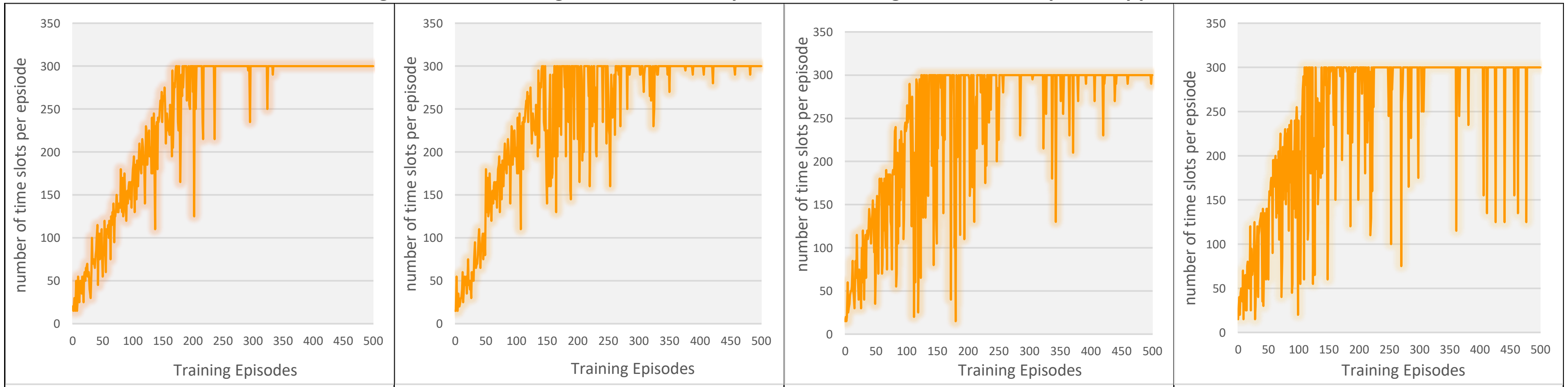
In this set of experiments, we put the maximum number of time slots in an episode to 300 time slots. Figures 5.19 and 5.20 show how various epsilon approaches impact the convergence of the learning process in cooperative and single learning. The convergence level represents how fast the learning agent knows about its environment and not behaving in a manner that causes the episode to terminate. It can be seen that comparing the convergence level in all epsilon approaches, the learning process in cooperative learning converges faster than in single learning. Additionally, the fluctuation pattern is extremely high in single learning as compared to cooperative learning. Furthermore, in all learning types, the approach with Epsilon A is the best in terms of converging faster, followed by the approach with Epsilon B, then the approach with Epsilon C, and lastly the approach with Epsilon D.

Moreover, it can be noticed that in single learning when Epsilon D is considered, it makes it really challenging for the learning agents to learn about their environment. this is due to the high variation even with higher number of training episodes. This is overcome when cooperation is considered.



a) Cooperative & Epsilon A ***b) Cooperative & Epsilon B*** ***c) Cooperative & Epsilon C*** ***d) Cooperative & Epsilon D***

Figure 5.19: Convergence level in Cooperative learning with various epsilon approaches.



a) Single & Epsilon A ***b) Single & Epsilon B*** ***c) Single & Epsilon C*** ***d) Single & Epsilon D***

Figure 5.20: Convergence level in Single learning with various epsilon approaches

.

In summary, it can be said that cooperative learning helps to improve the average rewards in the system along with leading to faster convergence. This is attributed to the fact that cooperative learning allows for the exploration of different parts of the state-action space by different agents. This can result in a more efficient exploration process compared to single learning, where a single agent might get stuck in a suboptimal policy.

5.3 Computational offloading and processing decisions Problem

In this chapter the second sub-problem is addressed. The aim of this sub-problem is to find the optimal computational offloading and processing decisions. The centralised system-wide long-term energy minimisation problem is defined as follow.

$$\overline{E^c(t)} = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{i=1}^{FN} \mathbb{E}\{ E_i(t)\}$$

The problem of finding the optimal computational offloading and processing decisions can be solved in a centralised and distributed manner. To be more specific, let’s assume that each fog node is aware of the status of all its neighbours, and as fog nodes make their decisions simultaneously at the beginning of each time slot, there is a high chance that a single fog node is selected by multiple fog nodes. This may lead to resource contention and overload at the selected neighbouring fog node. This could result in the selected neighbouring fog node not being able to process all the tasks assigned to it, which could lead to delays or task failure. To avoid this scenario and as local controller is aware of the status of all the fog nodes, the process of finding the optimal offloading decision will be handled by the local controller.

With regards to the processing decisions problem, this can be solved in a distributed manner in each fog node. Each fog node in the network makes processing decisions based on its own local information and resources, leading to a more efficient and effective use of network resources. Therefore, the long-term energy minimisation problem in each fog node under the same constraints stated in the problem in equation (4.24) C1-C10, is defined as

$$\overline{E_i(t)} = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{E_i\,(t)\}$$

The minimisation of the long-term time-average energy consumption requires prior knowledge about the information of the system, such as the status of the queues, available computation resources, and neighbours' details. Having this information in advance in a stochastic system is very difficult which makes the stochastic constrained optimisation problem harder to be solved. Dealing with such stochastic systems requires the utilisation of a powerful optimisation technique that can handle the characteristic of such stochastic systems. Lyapunov optimisation is a mathematical approach, that helps to solve time-average optimisation problems with constraints in stochastic systems without having prior knowledge about the systems' conditions [97]. Along with optimising the constrained optimisation problem it takes into consideration the stability of the systems' queues.

The reason for selecting Lyapunov optimisation is due to its unique structure which provides the following:

- Helping to decompose the time-average optimisation problem into a series of subproblems that can be solved efficiently in each time slot. By solving the subproblems, the optimal solution to the optimisation problem can be found. This can be accomplished without prior knowledge of all system information by only considering the current system status [98].
- Converting some constraints into virtual queues in order to meet their requirements [99].
- Addressing the trade-off between the objective function and the stability of the systems' queues [98].

In this context, Lyapunov Optimisation is used to solve the optimisation problem in each time slot. Based on Lyapunov Optimization theory, the drift-plus-penalty framework is used to optimise the time-average energy consumption and ensure the stability of the queues in the system [97].

5.3.1 Lyapunov Optimisation Framework

The optimisation problem of minimising the time-average energy consumption is transformed based on the Lyapunov Optimisation technique, so that it can be solved in each time slot. Lyapunov Function

The vector of system states in this work is defined as

$$\theta_i(t) \triangleq \left[Q_i(t), {X_i}^{Q}(t), LQ_i(t), {X_i}^{LQ}(t), HQ_i(t), {X_i}^{HQ}(t)\right] \tag{5.5}$$

According to Lyapunov optimisation theory, Lyapunov function is defined to be used for the scalar network congestion measurement [97], and it is the sum of the squares of queue backlogs in the system. Thus, the Lyapunov function for the system states is defined as

$$L(\theta_i(t)) \triangleq \frac{1}{2}[Q_i(t)^2 + {X_i}^{Q}(t)^2 + LQ_i(t)^2 + {X_i}^{LQ}(t)^2 + HQ_i(t)^2 + {X_i}^{HQ}(t)^2] \tag{5.6}$$

lower value of $L\big(\theta(t)\big)$ indicates lower congestion which implies that the queues in the system are more stable. Higher value means that at least one of the queues is large. In this case, it is an essential that the Lyapunov function be pushed towards a lower congestion state to maintain the stability in the system.

5.3.1.1 Lyapunov Drift

The change of the Lyapunov function from one time slot to the next is denoted as Lyapunov drift, which helps to maintain the system. Minimising the drift is aimed to push the queues towards a lower congested state, which means maintaining network stability. The one-step Lyapunov drift is expressed as:

$$\triangle\left(\theta_i(t)\right) = \mathbb{E}\ \left[L\left(\theta_i(t+1)\right) - L\left(\theta_i(t)\right)\middle|\theta_i(t)\right] \tag{5.7}$$

Minimising the drift $\triangle\left(\boldsymbol{\theta(t)}\right)$ for every time slot, will only ensure meeting the desired time average constraints, while the objective function (minimising the energy consumption) is not yet involved. In this manner, the drift-plus-penalty is considered.

5.3.1.2 Lyapunov Drift and Penalty Function

Achieving queue stability while minimizing the energy consumption drift-plus-penalty is considered and expressed as:

$$\triangle\left(\theta_i(t)\right) + V\,.\,\mathbb{E}\left\{E_i\left((t)\middle|\theta_i(t)\right)\right\} \tag{5.8}$$

V is a non-negative control parameter that emphasise the importance of minimising the energy and controls the trade-off between the energy consumption and the queues. In other words, it is the penalty weight of the objective function to the drift. When V=0 this means that we only consider minimising the drift. Higher value of V gives a priority in minimising the energy consumption over the queues. $\boldsymbol{E}(t)$ is the energy consumed by all fog nodes at time slot (t) when making the offloading decisions.

Based on Lyapunov optimisation theory, and to achieve the objective of minimising the drift plus penalty, an upper bound should be determined upon which the drift-plus-penalty should not be exceeded.

5.3.1.3 Upper-bound of the Drift-Plus-Penalty Function

First, considering the Lyapunov drift, and substituting equation (5.6) into (5.7) we have

$$\triangle\left(\theta_i(t)\right) = \frac{1}{2}\begin{bmatrix} [Q_i\,(t+1)^2 - \,Q_i\,(t)^2] + \left[{X_i}^{Q}\,(t+1)^2 - \,{X_i}^{Q}\,(t)^2\right] + \\ [LQ_i\,(t+1)^2 - \,LQ_i\,(t)^2] + \left[{X_i}^{LQ}\,(t+1)^2 - \,{X_i}^{LQ}\,(t)^2\right]\ + \\ [HQ_i\,(t+1)^2 - \,HQ_i\,(t)^2] + \left[{X_i}^{HQ}\,(t+1)^2 - \,{X_i}^{HQ}\,(t)^2\right] \end{bmatrix} \tag{5.9}$$

For simplicity, the process of finding the output of the first part $[Q_i\,(t+1)^2 - \,Q_i\,(t)^2]$ in (5.9) will be shown based on Lyapunov Optimisation, and then the final output of the equation (5.9) is demonstrated, as all the parts follow the same pattern. The value of $Q_i\,(t+1)$ in (5.9) is defined in (4.4) and when replacing its value in the section $[Q_i\,(t+1)^2 -\ Q_i\,(t)^2]$ in (5.9) we derive.

$$\left[\max\left[Q_i(t) - l_i(t) - \sum_{j\in AN, i\neq j} h_{ij}\,(t), 0\right] +\ \mathrm{A}_i(\mathrm{t})\right]^2 - Q_i^2\,(t)$$

Based on Lyapunov optimisation, the following lemma is considered [97].

Lemma 1:

Having a set of positive real numbers, e.g., A ≥ 0, B ≥ 0, C ≥ 0, and D ≥ 0 that satisfied

$$A = \max\ [B - C, 0] + D$$

Then we can obtain the following

$$A^2 \leq\ B^2 + D^2 +\ C^2 + 2B\,(D - C) \tag{5.10}$$

Applying *lemma 1* to $Q_i\,(t+1)^2$ we have

$$Q_i\,(t+1)^2\ \leq\ Q_i^2(t) + A_i^2(t) + l_i^2(t) + \sum_{j\in AN, i\neq j} h_{ij}^2(t) \tag{5.11}$$
$$+2Q_i(t)\left(\mathrm{A}_i(\mathrm{t}) - \left(l_i(t) + \sum_{j\in AN, i\neq j} h_{ij}\,(t)\right)\right)$$

Substituting the value of $Q_i\,(t+1)^2$ in part of equation (5.9) we have

$$\triangle\left(\theta_i(t)\right) \leq \tfrac{1}{2}\left(Q_i^2(t) + A_i^2(t) + l_i^2(t) + \textstyle\sum_{j\in AN, i\neq j} h_{ij}^2(t) + 2Q_i(t)\left(\mathrm{A}_i(\mathrm{t}) - \left(l_i(t) + \textstyle\sum_{j\in AN, i\neq j} h_{ij}\,(t)\right)\right) - Q_i^2\,(t)\right) \tag{5.12}$$

By applying mathematical operations, and in the case that $A_i(t)$, $l_i(t)$, $\sum_{j \in AN} h_{ij}(t)$ are upper bound by A_i^{max}, l_i^{max} and $h_{ij_i}^{max}$ we have.

$$\triangle\left(\theta_i(t)\right) \leq B + Q_i(t)\, \mathbb{E}\left\{A_i(t) - \left(l_i(t) + \sum_{j \in AN} h_{ij}(t)\right) \mid \theta_i(t)\right\} \tag{5.13}$$

Where $B = \frac{1}{2}\left[(A_i^{max})^2 + (l_i^{max})^2 + \left(h_{ij_i}^{max}\right)^2\right]$, and defined as a constant [100]. Applying the same steps to all parts in equation (7.6), and adding the penalty expression to both sides, we have the following.

$$\begin{aligned}
&\triangle\left(\theta_i(t)\right) + V \,.\, \mathbb{E}\left\{E_i(t) \mid \theta(t)\right\} \qquad (5.14)\\
&\leq B_1 + Q_i(t)\, \mathbb{E}\left\{A_i(t) - \left(l_i(t) + \sum_{j \in AN} h_{ij}(t)\right) \mid \theta_i(t)\right\}\\
&+ X_i^Q(t)\, \mathbb{E}\left\{\epsilon_{i1} - \left(l_i(t) + \sum_{j \in AN} h_{ij}(t)\right) \mid \theta_i(t)\right\}\\
&+ B_2 + LQ_i(t)\, \mathbb{E}\{l_i(t) - pl_i \mid \theta_i(t)\} + X_i^{LQ}(t)\, \mathbb{E}\{\epsilon_{i2} - pl_i \mid \theta_i(t)\}\\
&+ B_3 + HQ_i(t)\, \mathbb{E}\left\{\sum_{j \in FN} h_{ji}(t) - ph_i(t) \mid \theta_i(t)\right\}\\
&+ X_i^{HQ}(t)\, \mathbb{E}\{\epsilon_{i3} - ph_i(t) \mid \theta_i(t)\} +\\
&* V\left\{\left(1 - \delta_i(t-1)\right)(E_{so}) + E_{pl_i}^{proc} + \left(\sum_{j \in AN\; i \neq j} E_{h_{ij}}^{tr}\right)\right.\\
&+ \left(\sum_{j \in AN\; i \neq j} E_{h_{ij-b+a}}^{re}\right) + \left(\sum_{j \in FN\; i \neq j} E_{h_{ji}}^{re}\right) + E_{ph_{ji}}^{proc}\\
&\left. + \left(\sum_{j \in FN\; i \neq j} E_{ph_{ji}}^{tr}\right) + \left(E_{idle}^i * T_{idle}^i\right)\right\}
\end{aligned}$$

In the right-hand side of the drift plus penalty, the first and second lines of the drift plus penalty are related to the arrival queue and its virtual queue. The third line corresponds to the local queue and its virtual queue, and the fourth and fifth lines are related to the help queue and its virtual queue. The final line is the penalty represented as the total energy consumption during time slot (t).

B_1, B_2, and B_3 are constants and based on Lyapunov optimisation they are defined as

$$B_1 = \frac{1}{2}\left[A_{max}^2 + l_{max}^2 + h_{ij_{max}}^2 + \epsilon_1^2\right]$$

$$B_2 = \frac{1}{2}\left[l_{max}^2 + pl_{max}^2 + \epsilon_2^2\right]$$

$$B_3 = \frac{1}{2}\left[h_{ji_{max}}^2 + ph_{max}^2 + \epsilon_3^2\right]$$

Since receiving the processed tasks from the neighbours that belongs to fog node i is not affected by the action taken in time slot (t), and the size of processed tasks is small compared to unprocessed tasks, we remove the parameter $\sum_{j \in AN\ i \neq j} E_{h_{ij-b+a}}^{re}$ from the penalty definition.

By applying multiplications and combining the variables that are affected by the same type of workload (e.g., the amount of workload that leaves the arrival queue and its virtual queue and enters the local queue is considered as the same type of workload), and combining the penalty correlated to each specific type of workload, the right-hand side of the drift plus penalty in equation (5.14) will be reformulated as the following:

$$\underset{l_i(t), h_{ij}(t), pl_{ij}(t), ph_{ij}(t)}{\boldsymbol{Minimise}} \quad B_1 + B_2 + B_3 + Q_i(t)\, A_i(t) + X_i^Q(t)\epsilon_{i1} + X_i^{LQ}(t)\epsilon_{i2} + X_i^{HQ}(t)\epsilon_{i3} + \Lambda_1(t) + \Lambda_2(t) + \Lambda_3(t) \tag{5.15}$$

Where $\Lambda_1(t), \Lambda_2(t)$ and $\Lambda_3(t)$ are defined as:

$$\Lambda_1(t) = \left[-Q_i(t) - X_i^Q(t) + LQ_i(t)\right] l_i(t)$$

$$\Lambda_2(t) = \left[-Q_i(t) - X_i^Q(t) + HQ_j(t) + V\left(E_{h_{ij}}^{tr}(t)\right)\right] h_{ij}(t)$$

$$\Lambda_3(t) = \left[-LQ_i(t) - X_i^{LQ}(t) + V\left(E_{pl_i}^{proc}(t)\right)\right] pl_{ij}(t) + \left[-HQ_i(t) - X_i^{HQ}(t) + V\left(E_{ph_i}^{proc}(t) + E_{ph_{ji}}^{tr}(t)\right)\right] ph_{ij}(t)$$

Minimising the right-hand side of the drift plus penalty in the system is hard, as it is a complex non-linear problem with four decision variables that is difficult to solve directly. Decomposing a complex non-linear problem into smaller, simpler sub-problems and addressing each one individually can result in more efficient solutions. By focusing on each sub-problem separately, specialized methods that are best suited for that specific sub-problem can be applied, leading to faster computation times and more precise results. Since B_1, B_2, and B_3 are finite constants, and the parameters $A_i(t)$, $\epsilon_{i1}, \epsilon_{i2}$, and ϵ_{i3} are not impacted by the offloading/processing decisions, they are removed from the equation (5.15) [101]. Therefore, we focus on minimising $\Lambda_1(t), \Lambda_2(t)$, and $\Lambda_3(t)$ in each time step.

5.3.2 Problem transformation and Solution

The problem of minimising the upper-bound of the drift plus penalty is decomposed into several problems and solved accordingly following the steps in [102]. In our work we partitioned the problem into three subproblems based on the four decision variables.

1- Subproblem 1: called “Local queue offloading decisions”.
2- Subproblem 2: named “Offloading to neighbours’ decisions”.
3- Subproblem 3: referred to as “Workload processing decisions”.

By solving each one of these problems independently, we can manage to minimise the upper bound of the drift plus penalty.

5.3.2.1 Local Queue Offloading Decisions:

In this subproblem we aim to find the optimal amount of workload that moved from the arrival queue to the local queue in each fog node. The process is done locally in each fog node as it only requires knowing the information of the arrival queue and the local queue of the fog node itself. This can be solved by minimising the equation in $\Lambda_1(t)$ as follows.

$$\underset{l_i(t)}{\boldsymbol{Minimise}} \quad \left[-Q_i(t) - X_i^Q(t) + LQ_i(t)\right] l_i(t) \tag{5.16}$$

As the problem in this case is a linear problem with one variable, it can be solved according to a threshold policy. In this manner, as we are willing to find the optimal amount of workload that can be transmitted from the arrival queue to the local queue, the threshold is set considering both the arrival and local queue as follows:

$$l_i(t) \begin{cases} l_i^{max}, if\ Q_i(t) + X_i^Q(t) > LQ_i(t)\ and\ LQ_i(t) < LQ_i^{max} \\ 0, \qquad otherwise \end{cases} \tag{5.17}$$

The above equation means that each fog node will compare its arrival queue $Q_i(t)$ plus its virtual queue $X_i^Q(t)$ to it local queue $LQ_i(t)$, and if there is an enormous amount of workload in its arrival queue compared to its local queue, it will offload as much workload as possible l_i^{max} until the stopping condition is terminated which is $Q_i(t) + X_i^Q(t) \leq LQ_i(t)$ or $LQ_i(t)$ has reached its maximum length.

5.3.2.2 Offloading to Neighbours' Decisions:

This subproblem is solved by the local controller after receiving information vectors from all fog nodes under its controller. The information vectors contain the status of all actual and virtual queues. The aim of this sub-problem is to minimise the term $\Lambda_2(t)$ on the right-hand side of the drift plus penalty, by making the optimal decision regarding how much data $h_{ij}(t)$ is transmitted from one fog node to another based on their current information status. If the local controller makes the offloading decision for an active FN, the sub-problem here is related to the difference between the arrival queue of the active FN and the help queue of its neighbour. If the offloading decision is related to asleep FN, the sub-problem is transmitted to take into consideration the local queue of the sleeping FN and the help queue of the neighbour. The reason we include the local queue status when making the offloading decision for the sleeping FN rather than the arrival queue status, is because as the decision is already made by the FN itself to transfer tasks from its arrival queue to its local queue, there might be no available tasks in the arrival queue to be transmitted to neighbours. Additionally, since the sleeping FN will not do any processing, tasks in its local queue will suffer from delay that might lead to the violation of the tasks' deadline. The sub-problem regarding active and sleeping FN is in equations (5.18) and (5.19) respectively.

$$\underset{h_{ij}(t)}{\boldsymbol{Minimise}} \quad \left[-Q_i(t) - X_i^Q(t) + HQ_j(t) + V\left(E_{h_{ij}}^{tr}(t)\right)\right] h_{ij}(t) \tag{5.18}$$

$$\underset{h_{ij}(t)}{\boldsymbol{Minimise}} \quad \left[-LQ_i(t) - X_i^{LQ}(t) + HQ_j(t) + V\left(E_{h_{ij}}^{tr}(t)\right)\right] h_{ij}(t) \tag{5.19}$$

This can be solved using a threshold policy which determines a threshold value upon which the data is transmitted. For active FNs, the threshold policy is defined as

$$h_{ij}(t)\begin{cases} h_{ij}^{max}, if\ Q_i(t) + X_i^Q(t) - HQ_j(t) > V\left(E_{h_{ij}}^{tr}(t)\right) \& \ HQ_j(t) < HQ_j^{max} \\ 0, \qquad otherwise \end{cases} \tag{5.20}$$

For sleeping FNs, the threshold policy is defined as

$$h_{ij}(t)\begin{cases} h_{ij}^{max}, if\ LQ_i(t) + X_i^{LQ}(t) - HQ_j(t) > V\left(E_{h_{ij}}^{tr}(t)\right) \& \left[HQ_j(t) < HQ_j^{max} OR\ cost_i < cost_{max}\right] \\ 0, \qquad otherwise \end{cases} \tag{5.21}$$

The terms $Q_i(t) + X_i^Q(t) - HQ_j(t)$ represents the difference of the congestion level between the arrival queue of fog node i and the help queue of fog node j. Additionally, the threshold balances the trade-off between the total energy of transmitting workloads and the amount of workload transmitted, by taking into account the value of V. This is shown in the term $V\left(E_{h_{ij}}^{tr}(t)\right)$, where higher value of V will force the system to prioritise minimizing the total energy of transmitting workloads over transmitting a large number of workloads. On the other hand, if V is low, this means that the system will prioritise transmitting large amounts of workloads over minimising the energy of transmitting these workloads.

To solve this subproblem, the local controller first ranks sleeping fog nodes based on their congestion level represented by their actual local queues $LQ_i(t)$ plus virtual local queues $X_i^{LQ}(t)$ descending from the highest to the lowest. Then the local controller will rank the active neighbours ascending based on their help queues status and select the active fog node with the least amounts of workloads in its help queue $HQ_j(t)$ to handle the workload from the most congested sleeping FN. Accordingly, the expected help queue information $\mathbb{E}\left\{HQ_j(t)\right\}$ is updated in the local controller with the expected received workload. The conditions $HQ_j(t) < HQ_j^{max}$ and $cost_i < cost_{max}$ mean that the offloading of tasks will stop if the help queue of the neighbour has reached its maximum threshold or the total cost spent by the sleeping FN to offload its tasks has reached its maximum value. The cost is described in more details in chapter six. The process is repeated until making the offloading decision for all sleeping FNs. Following that the local controller will make the offloading decision for all active FN following the same previous steps.

5.3.2.3 Workload Processing Decisions

The aim of this subproblem is to minimise the term $\Lambda_3(t)$ in the drift plus penalty equation, and it is applied by active FN. This can be accomplished by finding the optimal amounts of workloads that will be processed in the current time slot (t) from the local queue and the help queue respectively. This is accomplished by taking into account the available computational capacity of the fog node in the current time slot (t) and the proposed policy. So, the problem is formulated as follows:

$$\begin{aligned}\underset{pl_{ij}(t),ph_{ij}(t)}{\boldsymbol{Minimise}}\ & \left[-LQ_i(t) - X_i^{LQ}(t) + V\left(E_{pl_i}^{proc}(t)\right)\right] pl_{ij}(t) \\ &+ \left[-HQ_i(t) - X_i^{HQ}(t) + V\left(E_{ph_i}^{proc}(t) + E_{ph_{ji}}^{tr}(t)\right)\right] ph_{ij}(t)\end{aligned} \tag{5.22}$$

$subject\ to$

$$pl_{ij}(t) + ph_{ij}(t) \le b^{e}_{i,max}(t)$$

$$0 \le pl_{ij}(t) \le LQ_i(t)$$

$$0 \le ph_{ij}(t) \le HQ_i(t)$$

$$0 \le \mathbb{E}\left\{E^{proc}_{pl_i}(t)\right\} + \mathbb{E}\left\{E^{proc}_{ph_i}(t)\right\} + \mathbb{E}\left\{E^{tr}_{ph_{ji}}(t)\right\} \le E^{max}_i(t)$$

Where $b^{e}_{i,max}(t)$ is the maximum amount of workload that can be processed by fog node i in time slot (t) based on the computational resources of fog node i. $E^{max}_i(t)$ is the maximum energy consumption for fog node i in time slot (t). The problem can be represented as a linear optimization problem with two decision variables $pl_{ij}(t)$ and $ph_{ij}(t)$ which can be solved using linear programming techniques. The Simplex method is one of the commonly used approaches for solving linear programming problems [103], in this context, we apply the Simplex algorithm to solve the above problem. The Simplex algorithm is an iterative method that starts with a feasible solution and then improves it by moving along the edges of the feasible region until the optimal solution is achieved. The reasons for selecting this method are attributed to its characteristics [103]. It is a widely used optimization algorithm that is relatively simple to understand and implement. Moreover, it is robust and reliable, and guarantees convergence to an optimal solution if one exists.

The Simplex algorithm helps to find the optimal value of $pl_{ij}(t)$ and $ph_{ij}(t)$ which helps to minimise the linear optimisation problem. Since the cost of energy for processing the amount of $pl_{ij}(t)$ tasks depends of the value of $pl_{ij}(t)$, and the cost for processing the amount of $ph_{ij}(t)$ tasks depends on the amount of $ph_{ij}(t)$, before we start the Simplex algorithm, we should assign initial values for $pl_{ij}(t)$ and $ph_{ij}(t)$ so we can calculate the cost for processing these tasks. Initially, based on the structure of the Simplex algorithm [103], the decision variables $pl_{ij}(t)$ and $ph_{ij}(t)$ will have a value of zero. Then we apply Simplex to find the optimal values of $pl_{ij}(t)$ and $ph_{ij}(t)$. After finding the optimal values of $pl_{ij}(t)$ and $ph_{ij}(t)$ using the Simplex algorithm, the new values of $pl_{ij}(t)$ and $ph_{ij}(t)$ are substituted into the optimisation problem again to calculate the cost for processing these tasks, and then the Simplex algorithm is re-applied to find the optimal values of $pl_{ij}(t)$ and $ph_{ij}(t)$. This process is repeated for a fixed number of iterations or until the values of $pl_{ij}(t)$ and $ph_{ij}(t)$ have converged. The reason for repeatedly changing the values of $pl_{ij}(t)$ and $ph_{ij}(t)$ and then applying the Simplex algorithm is to ensure that the solution is not trapped at a local optimum. Regarding the convergence of the solution, it can be said that the solution had converged if the new optimal values of $pl_{ij}(t)$ and $ph_{ij}(t)$ are similar to the previous optimal values of $pl_{ij}(t)$ and $ph_{ij}(t)$. The process of finding the optimal values of the decision variables of $pl_{ij}(t)$ and $ph_{ij}(t)$ using the Simplex algorithm is described in algorithm 5.4.

Algorithm 5.4: Workload Processing Decisions Algorithm

Workload Processing Decisions Algorithm based on Simplex Algorithm

```
Input   $LQ_i(t)$, $X_i^{LQ}(t)$, $HQ_i(t)$, $X_i^{HQ}(t)$, V, $b_{i,max}^{e}(t)$, $Px = null$, $Py = null$
Output  $pl_{ij}(t)$, $ph_{ij}(t)$
1  Determine slack variables
2  If (iteration == 1)
3     Create initial values $pl_{ij}(t) = ph_{ij}(t) = 0$
4  Else
5     If ( $Px = pl_{ij}(t)$ & $Py = ph_{ij}(t)$)
6        Convergence exists, Break
7     Else
8        If (iteration > max)
9           Break
10       Else
11          Go to step (16)
12       End If
13    End If
14 End If
15 $Px = pl_{ij}(t)$   $Py = ph_{ij}(t)$
16 Calculate $E_{pl_i}^{proc}(t), E_{ph_i}^{proc}(t), E_{ph_{ji}}^{tr}(t)$ based on equations
17 Apply Simplex algorithm
18    Create the tableau with objective function and constraints.
19    If (Optimal solution exists)
20       $pl_{ij}(t) = pl_{ij}(t)''$ , $ph_{ij}(t) = ph_{ij}(t)''$.
21       Go to point (5)
22    Else
23       Identify pivot element and perform the pivot operation.
24       Update tableau by pivoting
25       Go to point (19)
26    End If
```

In Algorithm 5.4, line 1 indicates that the inequality constraints in the linear problem is transformed into equality constraints by adding slack and surplus variables [104]. In line 2, if this is the first iteration, the initial values of $pl_{ij}(t)$ and $ph_{ij}(t)$ are created as initial feasible solutions to the linear problem. If this is not the first iteration, the convergence of the optimal solution is checked in line 5. This is done by comparing the previous optimal solution of $pl_{ij}(t)$ and $ph_{ij}(t)$ created by the Simplex algorithm with the new optimal solution. If they are the same, the optimal solution has been reached and the iteration will stop. Otherwise, if the number of iterations has reached its maximum threshold, the algorithm will stop, and the last optimal solution is performed. If this is not the case, the last optimal solution is saved for future comparison and the potential cost represented as the total energy consumption for processing and transmitting a total amount of $pl_{ij}(t)$ and $ph_{ij}(t)$ based on the initial solution is calculated, in line 16. From lines 17 to 26, the steps of simplex algorithm are performed.

First, the initial tableau with the objective function and constraints is created. Then the optimality of the solution is evaluated. The optimality of the solution is determined based on the values of the tableau that belongs to the objective row, if all are non-negative, then the optimality has been reached. Otherwise, pivot operations are processed as follows. The highest negative value in the bottom row that belongs to the objective function is selected in the pivot column. Following this process, the elements in the rightmost column (except for the objective row) in the tableau are divided by the corresponding positive elements in the pivotal column. Then, the row with the lowest value is selected as the pivot row. The pivot element is the number located at the intersection between the pivot column and row. According to the following procedure, the pivot operation is carried out:

- In the pivot column, apply mathematical operations to make the pivot element equal to one, and apply the same operation to all the elements in the pivot row.
- In the pivot column, make the elements that are located in different rows around the pivot element equal to zero by doing mathematical operations, and apply the same operation to the same row that each element belongs to.

Following this, in line 25, the algorithm checks again if the solution is optimal. If the solution is optimal, the algorithm will check if the convergence exists, and so on. In the case that the solution is not optimal, pivot operations will be applied again. Since this subproblem is a simple linear problem with limited constraints, it can be solved using an IP solver such as CPLEX.

5.3.3 Numerical Results

In this section we evaluate the performance of the proposed algorithms compared to other baselines.

5.3.3.1 Simulation setup

In this study we used iFogSim simulator to simulate the environment. System parameters are summarised in Table 5.5, and tasks characteristics are presented in Table 5.6.

Table 5.5: Simulation settings.

Parameter	Value
Length of time slot ($\Delta\ t$)	5 ms
Number of clusters	3
Cooperative or single learning	Cooperative learning
Epsilon approach	Epsilon A
Minimum threshold reward	0
Number of FNs in each cluster	5 fog nodes
Number of IoT devices in each IoT region	10 devices
Average arrival rates of tasks	[1-20], [1-30], [1-40], [1-50] ms
Latency from fog nodes to cloud layer	50 ms
Memory capacity of the fog nodes	4000 MB
CPU capacity of fog node (f_i)	8000 MIPS
Bandwidth	10000 Kbps
Maximum arrival queue length (Q_i)	100 tasks
Maximum local queue length (LQ_i)	100 tasks
Maximum help queue length (HQ_i)	100 tasks
Idle power	0.01 W [90]
Sleeping power	0.001 W
Processing power	0.9 W [90]
Transmitting and receiving power	1.3 W and 1.1 W [90]
Power overhead for activating a sleeping FN	0.002 W
Delay overhead for activating a sleeping FN	1 ms
Size if tasks in bytes	1000

Table 5.6: Tasks Characteristics.

Task Type	CPU length (MI)	Deadline (ms)	Priority
Public 1	800	200	-
Public 2	2000	100	-
Semi-private 1	800	70	-
Semi-private 2	2000	30	-
Private 1	800	70	-
Private 2	2000	30	✓

5.3.3.2 Baseline Approaches

In this work, we consider a set of baselines to compare our work with, a brief description of these baselines is provided in Table 5.7.

Table 5.7: Baselines Overview.

Baseline	Method
DEBTS [105]	Lyapunov Optimisation applied by local controller
CESC [55]	Lyapunov Optimisation and sleeping cycle.
EETO [24]	Lyapunov Optimisation applied at the gateway devices simulatiosly.
Local Execution (LE)	Each PFN process all its workload
Random Offloading (RO)	Each PFN decides randomly whether to process its tasks locally or offload it to a random neighbour.

A more detailed description about the baselines and drawbacks of each baseline is listed below. In EETO [24], the authors used Lyapunov optimisation technique to minimise the time-average energy consumption of end devices and fog nodes considering system constrains and the deadline of tasks. Their proposed approach has the following drawbacks.

1- Gateways make the offloading decisions simultaneously at the beginning of each time slot, which results in inaccurate current system information. This leads to selecting the same fog node as the most suitable candidate from several gateways. This leads to higher delays, consuming more energy and tasks drooping by violating their deadlines.

2- Having a limited number of gateways that receive a large number of IoT tasks in each time slot and then distribute them appropriately to fog devices may cause some tasks to wait in the gateway queue. This will result in having a long queueing delay on the gateways sides.

DEBTS approach [105]. The authors applied Lyapunov Optimisation with the aim of minimising time-average energy consumption within the fog system while ensuring system stability. The proposed approach is applied by the local controller at the beginning of each time slot.

CESC approach [55]. In this approach the authors aimed at minimising the average energy consumption of edge servers under a delay constraint. The proposed solution is based on using the Lyapunov optimisation technique along with applying a sleeping cycle approach. In their sleeping approach, energy is further saved by allowing any edge server to enter sleeping mode if its workload is below a certain threshold. The main drawback of their proposed scheme is when determining whether an edge server is in sleep mode or not, the status of other edge servers is not taken into consideration. This will impact delay, throughput, and energy consumption, especially if their proposed scheme encounters the following three scenarios.

1- Having a very congested environment where all edge servers are overloaded, and one edge server is below sleeping threshold. This will cause the edge server with lower workload to enter sleeping mode instead of helping other overloaded servers. This will have a negative impact on the system by causing long delay as well as drooping tasks by violating their deadline.
2- All edge servers are below sleeping threshold. In this case, all edge servers will be in a sleep mode. In this case, delay and throughputs are affected.
3- All edge servers are slightly higher than the sleeping threshold. In this manner, all edge servers are active. This will lead to a waste of energy, and it might be avoided if some of the edge servers had been put to sleep instead.

Local Execution (LE). In this approach each fog node will process all its tasks locally without cooperation. This will result in having long queueing delays and dropping of tasks due to violating QoS requirements (e.g., deadline), and exceeding computational resources. Moreover, a waste of energy will occur due to activating fog nodes in all time.

Random Offloading (RO). Each primary fog node decides whether to process the task locally or to offload it to one of its neighbours or to the cloud randomly. The decision of offloading tasks does not consider the availability of the computational and energy resources and system constraints.

5.3.3.3 Performance Metrics

To provide a comprehensive evaluation of our proposed approach compared to the stated baselines, the following metrics are considered.

1- **Average delay**, it is defined in chapter five in (5.18) equation.

2- **Average energy consumption**

The average energy consumption in the fog system in one cluster in one time slot is calculated as

$$\overline{\mathrm{E}_{cluster}}(\mathrm{t}) = \frac{\sum_{i \in FN} E_i}{FN\ number\ in\ one\ cluster}$$

And the average energy consumption in one cluster for all time slot is defined as

$$\text{Average energy consumption} = \frac{\sum_{t=1}^{T} \overline{\mathrm{E}_{cluster}}(\mathrm{t})}{T}$$

3- **Percentage of throughput.**

We cannot calculate the percentage of throughput in each time slot. This is because tasks that are generated in time slot (t) cannot always be processed within the same time slot, instead, sometimes they wait several time slots in the queues. In this manner, the percentage of throughputs is calculated as follows.

$$\text{Percentage of throughput} = \frac{\text{total drooped tasks}}{\text{total tasks generated in all time slots}} * 100$$

4- Average number of sleeping fog nodes

The average number of sleeping fog nodes is calculated as

$$\frac{\sum_{t=1}^{T} Number\ of\ SN}{T}$$

5.3.4 Experiment results structure

The structure of the results section is shown in Figure 5.21.

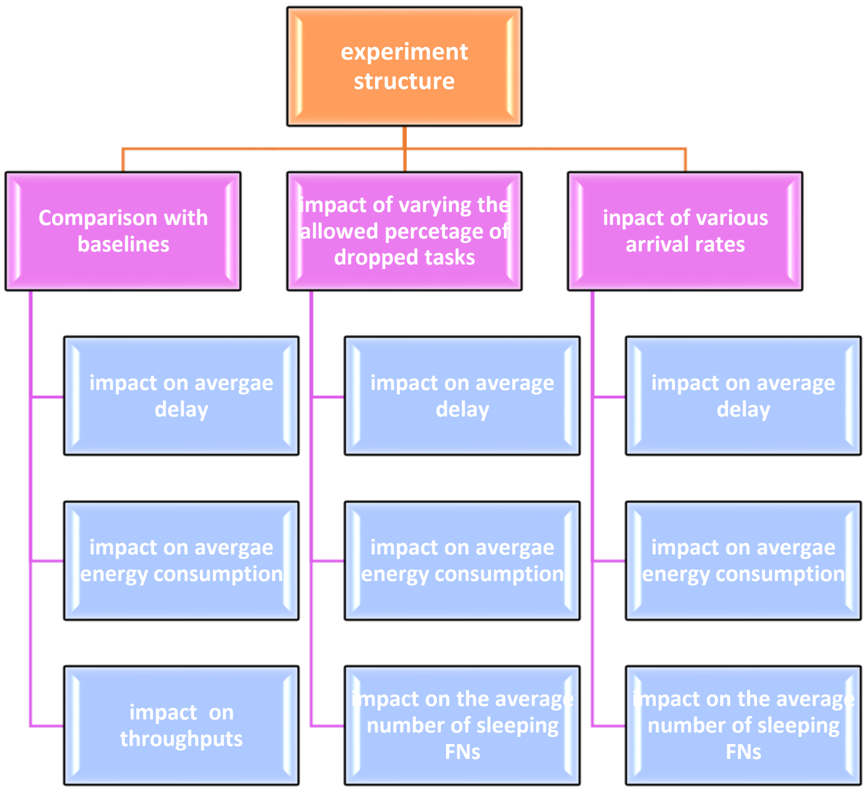

experiment structure
Comparison with baselines
impact of varying the allowed percetage of dropped tasks
inpact of various arrival rates
impact on avergae delay
impact on average delay
impact on average delay
impact on avergae energy consumption
impact on avergae energy consumption
impact on avergae energy consumption
impact on throughputs
impact on the average number of sleeping FNs
impact on the average number of sleeping FNs

Figure 5.21: Structure of the results section.

5.3.4.1 Comparison with baselines

In this section, we compared our proposed scheme JQLLO with all the stated baselines. In our approach, we allow each FN at the beginning of each time slot to drop 5% of its total tasks in its arrival queue and the approach in this case is called JQLLO-5%. The reason for this was to examine how the system would perform if 5% of its total workload was allowed to be dropped and how this will affect the efficiency of the proposed scheme that is based on combining Q-learning and Lyapunov Optimisation. The approach JQLLO-0% refers to when no dropping of tasks is allowed. As heterogeneous tasks and privacy requirements have not been investigated in these baselines, we apply our concept of sorting tasks based on their deadline and type and where each task should be processed for all these baselines.

The impact on the average delay

We examined the impact of various approaches on the average delay of each task based on its characteristics. The results are shown in Figure 5.22. it can be seen that public1 and public2 are processed in the cloud because of having loose deadlines. This is why the average delay is high compared to other type of tasks and this is due to the long transmission latency. In public1 and public2 type of tasks, all approaches exhibit the same patter. However, in LE public1 has higher average delay compared to public2 while both tasks are processed locally at the primary fog node. The reason for this is that public1 has loss deadline compared to public2, and due to the long queueing delay, the primary fog node will drop most of public2 tasks as the deadline cannot be met. For the remaining approaches, EETO has higher average delay compared to the rest approaches. The reason for that is attributed to the long queueing delay in the gateway side. As the type of task is waiting in the gateway queue in order to make the decision to offload it to the cloud for processing.

In terms of semi1 and semi2 tasks, each approach exhibits the same pattern in the two types of tasks except for the LE. In more details, in LE, the average delay of semi2 tasks is lower than semi1 tasks, and this is due to the loss of deadline in semi1 tasks similar to the scenarios of public1 and public2 tasks. For the rest of other approaches, Random has the highest average delay followed by EETO approach and then CESC. This is because Random approach offloads tasks without considering the best candidate to process them. As for EETO and CESC, the reason for higher average delays is attributed to the drawbacks mentioned in section 7.5.2 baseline approaches.

Additionally, for semi1 and semi2 tasks, the lowest average delay is seen in DEBTS approach followed by JQLLO-0%, and finally JQLLO-5% approaches. The reason that the lowest average delay is scored in BEBTS is because all the neighbouring fog devices in this approach are willing to help and are switched on in all time slots, in JQLLO-0% although there is no allowed dropping of tasks, we still managed to put a few fog nodes in the cluster in a sleep mode while still meeting the QoS requirements in the system. This resulted in having fewer available computational resources in the time slot. This caused some tasks to wait a bit longer in the queue of fog nodes compared to DEBTS.

Comparing CESC, EETO, Random and LE to our approach JQLLO-5% helps to reduce the delay by 7.18%, 26.77%, 58.64%, and 84.79% in semi1 tasks respectively. This is further reduced in JQLLO-0% approach compared to the same baselines by 14.96%, 32.91%, 62.11%, and 86.07% respectively. For semi2 tasks, JQLLO-5% manages to save around 6.46%, 27.64%, 60.50%, and 80.05% compared to CESC, EETO, Random and LE approaches. This is further reduced by JQLLO-0% approach, compared to the same set of baselines, JQLLO-0% saves up to 12.77%, 32.52%, 63.16%, and 81.39%. With regards to the difference between semi1 and semi2 type of tasks, overall, semi2 type of tasks have lower average delay compared to semi1 tasks with a difference of around 3.5%. This is attributed to the tight deadline of semi2 tasks, as the tasks are sorted based on their deadline in the arrival queue.

Regarding private tasks, for private1, JQLLO-5% reduces the average delay by 49.30%, 84.57%, 25.70%, and 11.96% compared to Random, LE, EETO and CESC approaches respectively. Further delay is accomplished with JQLLO-0% approach, as the average delay is decreased by 53.54%, 85.86%, 31.91%, and 19.31% for the same set of baselines. In terms of private2 tasks, JQLLO-5% saves around 56.87%, 81.72%, 36.82%, and 12.38% compared to Random, LE, EETO, and CESC approaches respectively. Additionally, further reduction is accomplished with JQLLO-0% approach by 60.21%, 83.14%, 41.72%, and 19.18% for the same stated baselines. Comparing private1 and private2 tasks, it seems that the lowest average delay is achieved in private2. This is due to having tight deadline compared to private1 tasks. However, the difference between the private1 and private2 tasks is around 11% which is higher than the difference between semi1 and semi2 tasks. This is attributed to the priority assigned to this type of tasks, as the system processes all private2 tasks before considering the process of any private1 tasks. Overall, our proposed schemes help to minimise the average delay compared to other benchmarks.

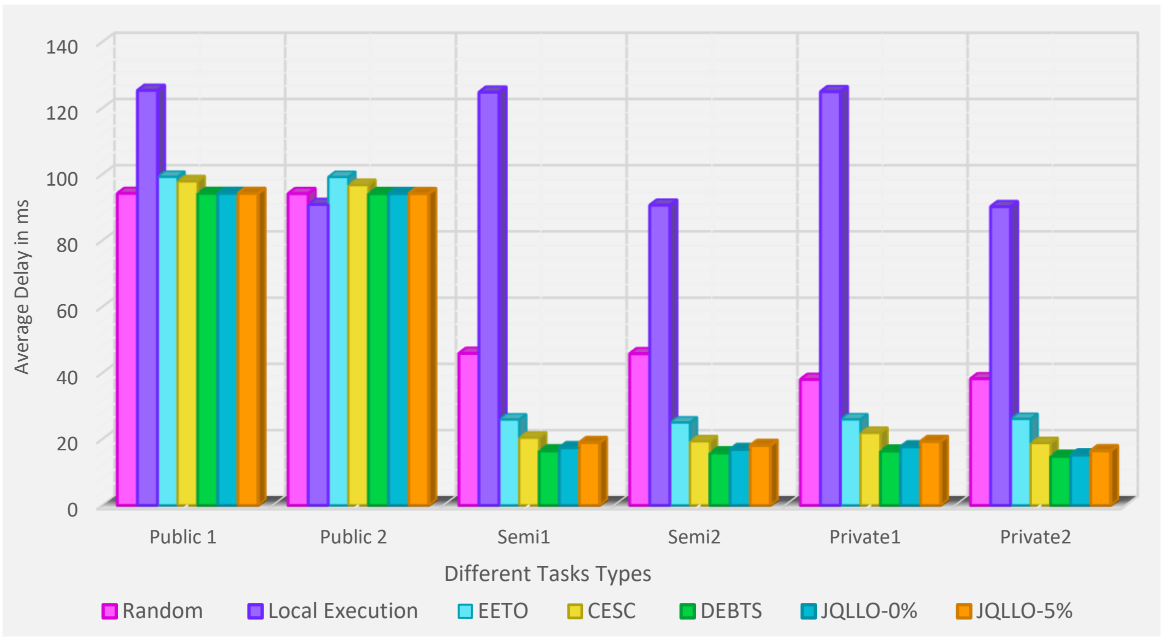


Figure 5.22: Impact of our approach and other baselines on the average delay

Impact on average energy consumption

Regarding saving energy consumption within the fog infrastructure, JQLLO approach saves more energy compared to all the stated baselines. This is shown in Figure 5.23. Since all the clusters exhibit the same features, we discuss the results with respect to cluster A. In more detail, when no tasks are allowed to be dropped, the JQLLO-0% approach helps to minimise the average energy consumption by 86.60%, 85.31%, 82.21%, 35.27%, and 26.17% in comparison to Random, LE, EETO, DEBTS, and CESC approaches respectively. This is further reduced when 5% of tasks are allowed to be dropped for the sake of conserving energy, where JQLLO-5% reduces the average energy by 87.91%, 86.98%, 84.81%, 38.20%, and 29.52% compared to Random, LE, EETO, DEBTS, and CESC approaches respectively.

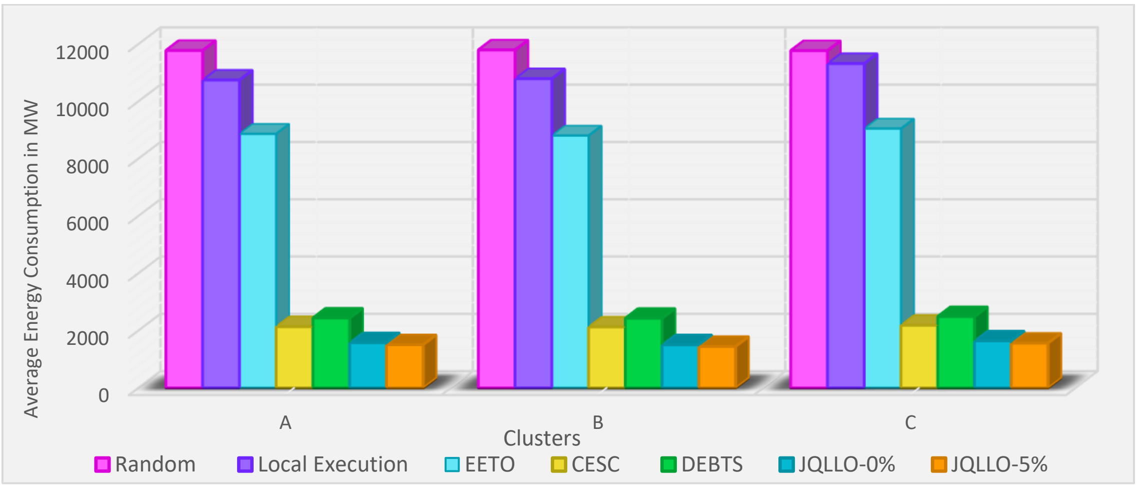


Figure 5.23: Impact of our approach and other baselines on the average energy consumption

Impact on the percentage of throughput

In terms of the percentage of throughput, results are shown in Figure 5.24. It can be seen that DEBTS and JQLLO-0% approaches achieved the highest throughput at 100%. This is followed by the JQLLO-5% and CESC approaches. Approach JQLLO-5% helps to accomplish around 95.2%, 95.71%, and 95.56% for cluster A, B, and C, respectively, and CESC approach manages to process around 94.3%, 95.01%, and 94.62% of the total tasks for clusters A, B and C. EETO approach managed to achieve around 93%, followed by LE and Random approaches with around 63% and 54.50% respectively.

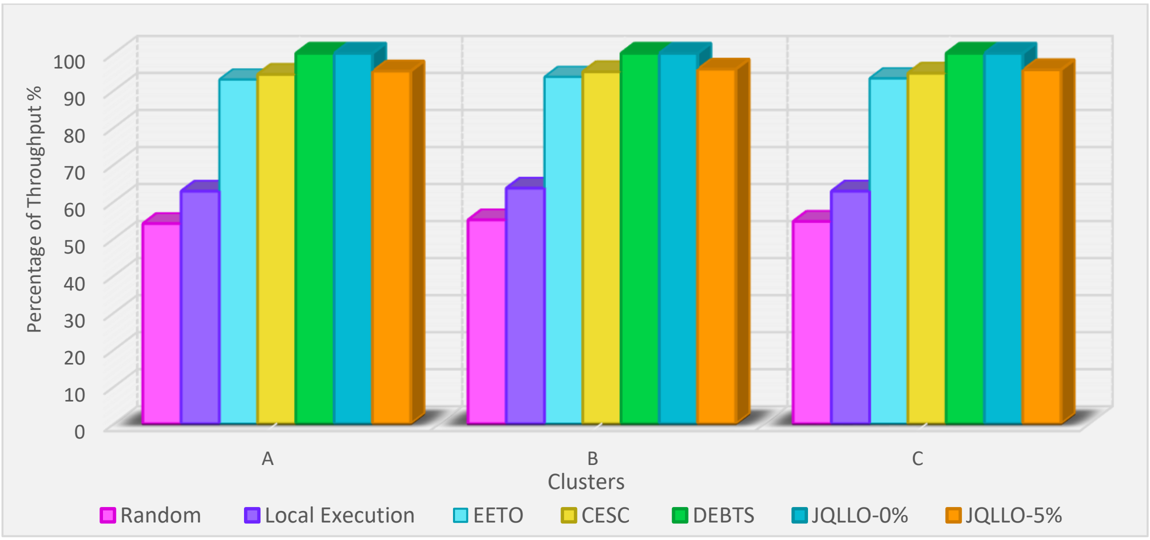


Figure 5.24: Impact of our approach and other baselines on the average percentage of throughputs

As a result of comparing our approach with baselines, we are able to significantly reduce the amount of energy consumed within the QoS requirements. The process can be achieved with only a small amount of time sacrificed.

5.3.4.2 The Impact of various allowed percentage of dropping tasks on the performance of JQLLO approach

In this set of experiments, we analysed the system performance when 5%, 10%, 15%, 20%, 25%, and 30% of total tasks are allowed to be dropped by each FN from its arrival queue at the beginning of each time slot, before creating and sending the information vector to the local controller. The dropping of tasks is equally distributed between different types of tasks. We investigate the effect of the various percentage of dropping of tasks on the average number of sleeping fog nodes, the average energy consumption, and finally on the average delay. These results are generated when the arrival rate of tasks is between 1 ms and 25 ms in each IoT device in which the system is extremely congested.

Average number of sleeping fog nodes and average percentage of energy consumption

It can be seen from Figures 5.25 and 5.26 that as we increase the allowed percentage of dropped tasks, the average number of sleeping fog nodes in each cluster grew. By increasing the number of sleeping fog nodes, more energy is saved.

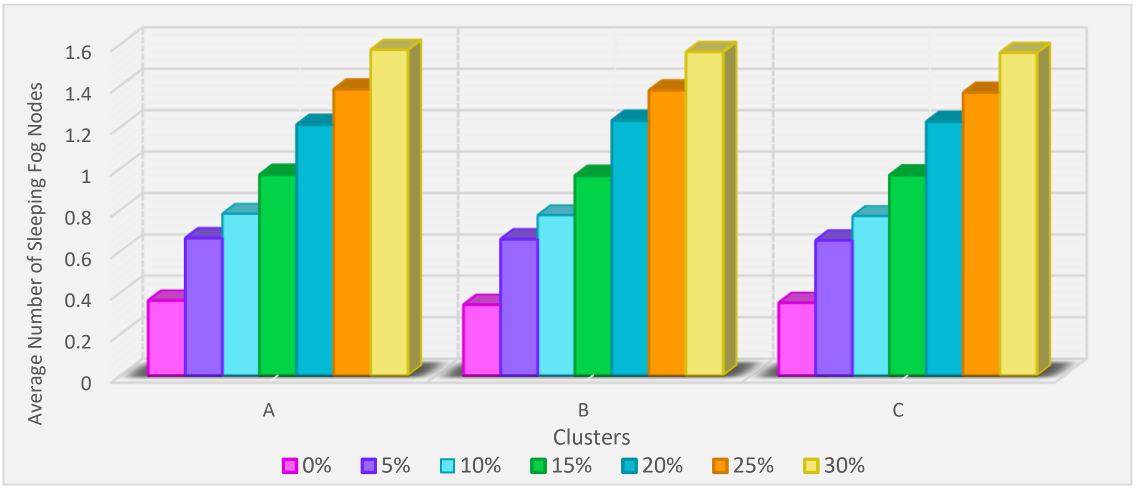


Figure 5.25: Impact of various allowed percentage of dropping tasks on the average number of sleeping fog nodes.

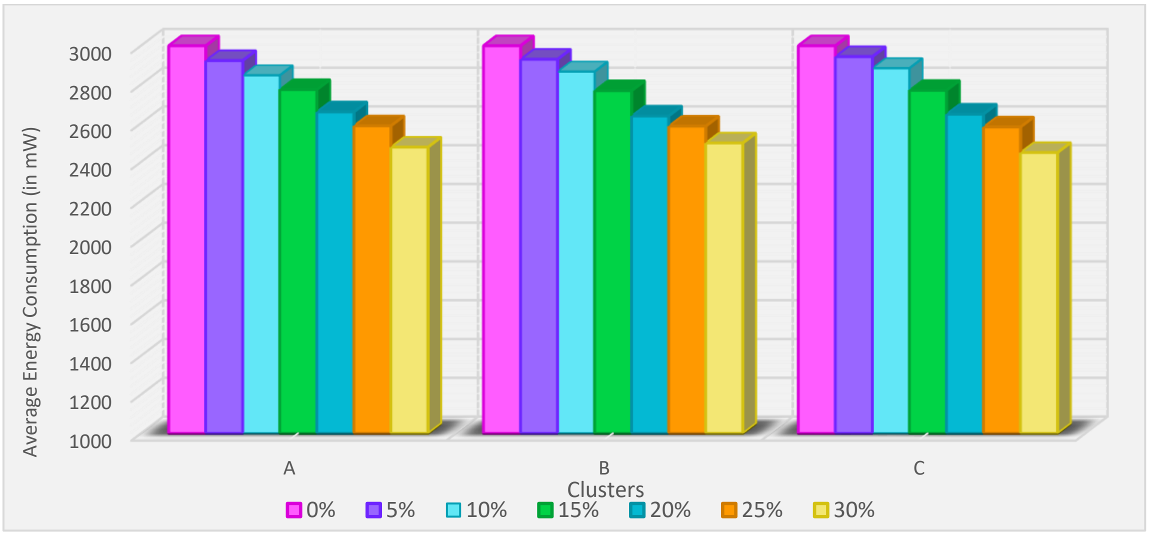


Figure 5.26: Impact of various allowed percentage of dropping tasks on the average energy consumption.

Impact on the average delay

We address if the allowed percentage of drooping tasks has an impact on the average delay of each type of task based on its properties, the results is shown in Figure 5.27. As shown previously in Figure 5.25, the increase in the percentage of drooping tasks leads to an increase in the average number of sleeping fog nodes, which in turn has an impact on the average delay as in Figure 5.27. it can be seen that the average delay of public type of tasks is not affected by the increase in the number of sleeping fog nodes as it is sent to the cloud for processing before putting fog nodes in sleep mode if their deadline is not violated. Putting PFNs in a sleep mode will impact the average delay on private type of tasks compared to other types. The reason for that is because once the PFN enters into the asleep cycle, the private tasks will wait in the queues until the PFN is activated, compared to semi-private tasks that can be processed by any fog node in the system. Private1 tasks are significantly affected by this, since private2 tasks are given priority, which barely has an effect as there are more sleeping fog nodes.

There was a slight increase in average delay for semi-private tasks as the number of sleeping fog nodes increased. The increase in the average delay in semi1 tasks is higher than the average delay in semi2 tasks. The reason for that is attributed to the tight deadline of semi2 tasks. This is because the system processes this type of tasks to meet its deadline while postponing the execution of semi1 tasks. The percentage of the increase in the average delay when 30% of drooping of tasks is allowed compared to when no drooping is allowed is shown in Table 5.8.

Table 5.8: The percentage of the impact of drooping 30% of tasks on the average delay

Semi1	**Semi2**	**Private1**	**Private2**
11.40%	7.29%	20.48%	7.29%

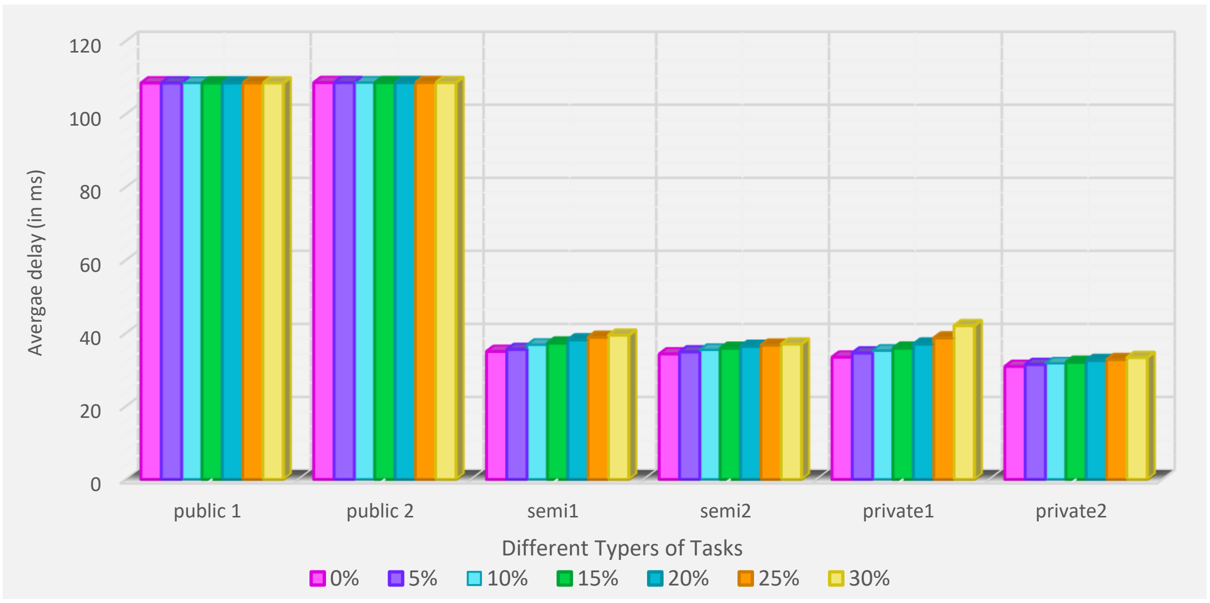


Figure 5.27: Impact of various allowed percentage of dropping tasks on the average delay.

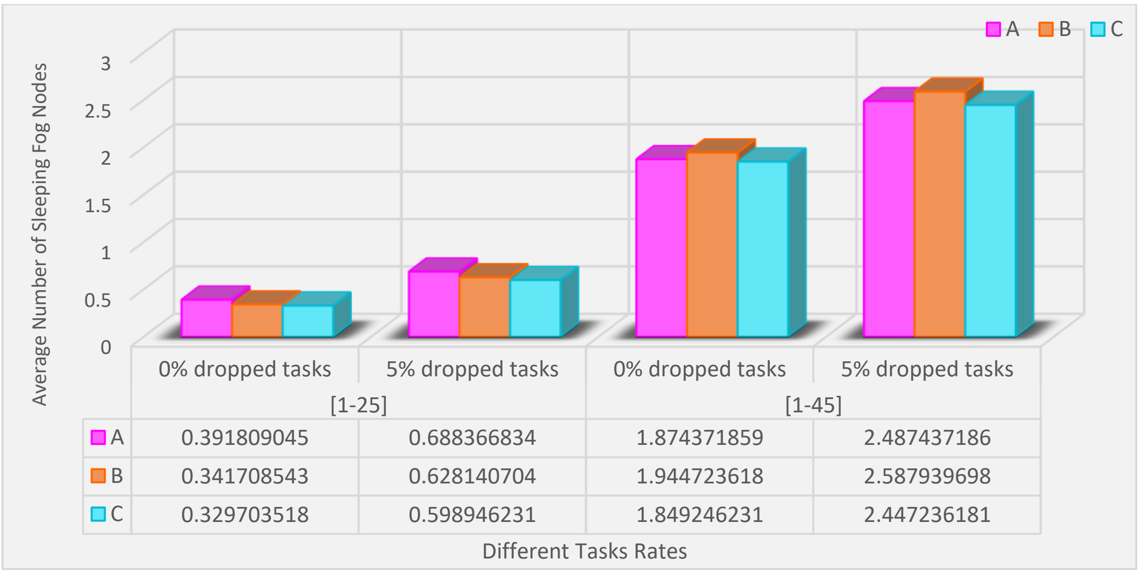
5.3.4.3 The impact of various arrival rates on the performance of JQLLO approach

In this section we examine the impact of two different task arrival rates on overall system performance. One of the analysed rates is [1-25], where the arrival rate of tasks ranges from 1 ms to a maximum of 25 ms randomly. In a rate [1-25], the system is extremely congested, while in a rate [1-45], the system is slightly stable and manageable. Additionally, within these two different rates, we examined the impact when zero and five percentages of tasks are allowed to be dropped.

We examined the impact of the above-mentioned scenario on the average delay in ms, average number of sleeping fog nodes, and average energy consumption.

The impact on the average number of sleeping fog nodes.

It can be seen from Figure 5.28 that in a congested environment, e.g., [1-25], our approach still manages to put fewer fog nodes in sleep mode to save more energy while still meeting the QoS requirements when 0% of dropping tasks is considered. The number of sleeping fog nodes is further increased when we allow the system to drop about 5% of the total generated tasks. The increment in the number of sleeping fog nodes is about 43.08%, 45.60%, and 44.95% for cluster A, B, and C, respectively. In a slightly congested environment, e.g., [1-45], more fog nodes enter sleep mode compared to high congested system. With 0% allowance of dropping tasks, for example, in cluster B, where the average number of fog nodes that are in sleep mode in [1-25] is about 0.34, and in [1-45] the average number is 1.94. This is further increased when five percent of the total generated tasks is allowed to be dropped, as in cluster B, in [1-25], the average number of asleep fog nodes is about 0.62, and with [1-45], the average number is about 2.58 for the same cluster. It is also observed that as the percentage of tasks dropped increases in [1-45], more fog nodes are going into sleep mode while meeting the QoS requirements. For example, in cluster B, the average number of fog nodes that are in sleep mode with 0% allowance of dropping tasks is about 1.94, and this number is increased to 2.58 with 5% allowance of dropping tasks.



	[1-25]		[1-45]	
	0% dropped tasks	5% dropped tasks	0% dropped tasks	5% dropped tasks
A	0.391809045	0.688366834	1.874371859	2.487437186
B	0.341708543	0.628140704	1.944723618	2.587939698
C	0.329703518	0.598946231	1.849246231	2.447236181

Figure 5.28: The impact of various arrival rates on the average number of sleeping fog nodes.

The impact on the average delay

The results are shown in Figure 5.29. In [1-25], it seems that with 0% allowance of dropping tasks, the average delay is minimised a bit more than when 5% is allowed. In more detail, the average delay is minimised by about 1.63%, 1.72%, 3.60%, and 1.84% for semi1, semi2, private1, and private2 tasks respectively. This is due to the fact that there are more available computing devices when no tasks are allowed to be dropped.

This is compared to when 5% of the generated tasks are allowed to be dropped and the number of active computing devices is reduced. The reason that the highest reduction in the average delay is noted in private1 tasks is because this type of task is the most affected when the number of sleeping fog nodes increases, especially as it should only be processed in the primary fog node and there is no alternative computing device. Additionally, private2 tasks are given priority when it comes to execution. So, the primary fog nodes process all private1 tasks, and once they are complete, private1 tasks are considered for processing.

The concept of having less average delay when 0% of tasks are allowed to be dropped more than when five percent is allowed, is also applied to [1-45]. The average delay with 0% of tasks allowed to be dropped outperforms the average delay when 5% of tasks are allowed to be dropped. This is by 8.62%, 8.20%, 12.50%, and 8.43% for semi1, semi2, private1, and private2 tasks respectively. Additionally, it appears that public tasks are not affected by the allowed percentage of dropping tasks since they are sent to the cloud. In a slightly congested environment, e.g., [1-45], average public task delays are reduced by around 13.16 percent compared to a congested environment, e.g., [1-25]. This is due to having a longer queueing delay on the cloud servers when a significant number of tasks are transmitted to the cloud compared to having fewer tasks to be transmitted. Among all the types of tasks, private2 tasks have the lowest average delay due to their priority.

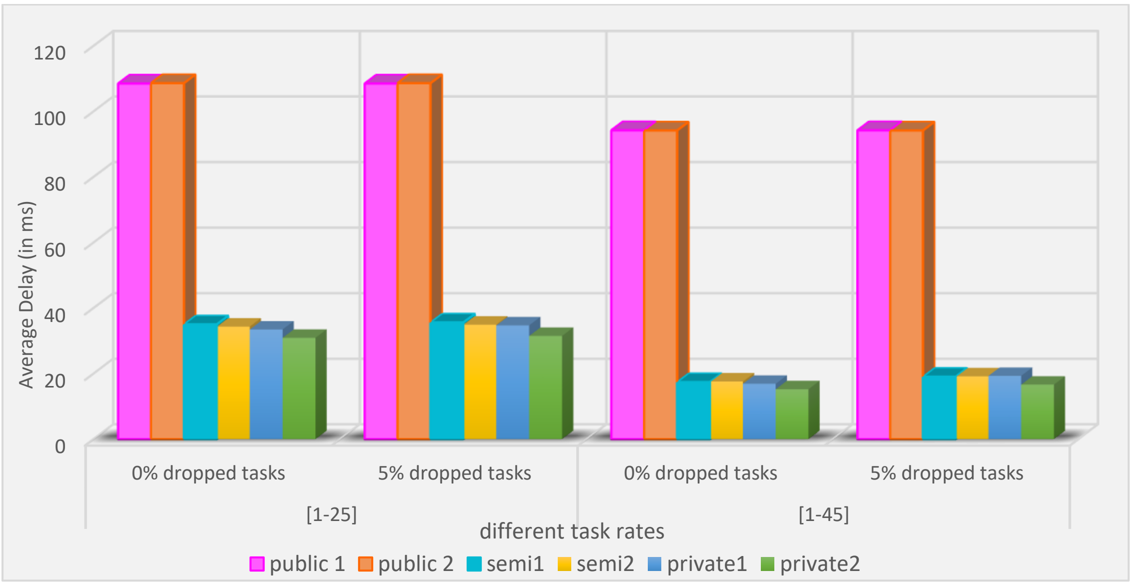


Figure 5.29: The impact of various arrival rates on the average delay.

The impact on the average energy consumption

In this experiment we examined three cases in the [1-25] and [1-45] environments. In the first case, we don't allow dropping tasks, and all fog nodes are in active mode. More precisely, we only apply Lyapunov optimisation with no Q-learning. Second, JQLLO-0% with zero percent of tasks drooping. Lastly, we have the JQLLO-5% approach with 5% drooping allowance. This is to show the effectiveness of our proposed approach.

The results are demonstrated in Figure 5.30. In extremely congested environments, e.g., [1-25], comparing Lyapunov-only to JQLLO-0%, our approach manages to save the average energy consumption by 10.45%, 11.25%, and 11.62% for cluster A, B, and C respectively. This is further reduced when we allow 5% of the total generated tasks to be dropped. In JQLLO-5%, the average energy consumption is further reduced by 4.67%, 3.99%, and 4.81% compared to JQLLO-0%.

When the environment is not congested, e.g., [1-45], further reduction is accomplished. In more detail, comparing JQLLO-0% with Lyapunov only, around 35.27%, 37.99%, and 34.06% of the energy is saved for clusters A, B, and C respectively. Additionally, when 5% of the total generated tasks are allowed to be dropped, the average energy consumption is reduced further by 4.53%, 3.91%, and 4.83% for the same set of clusters, compared to JQLLO-0%.

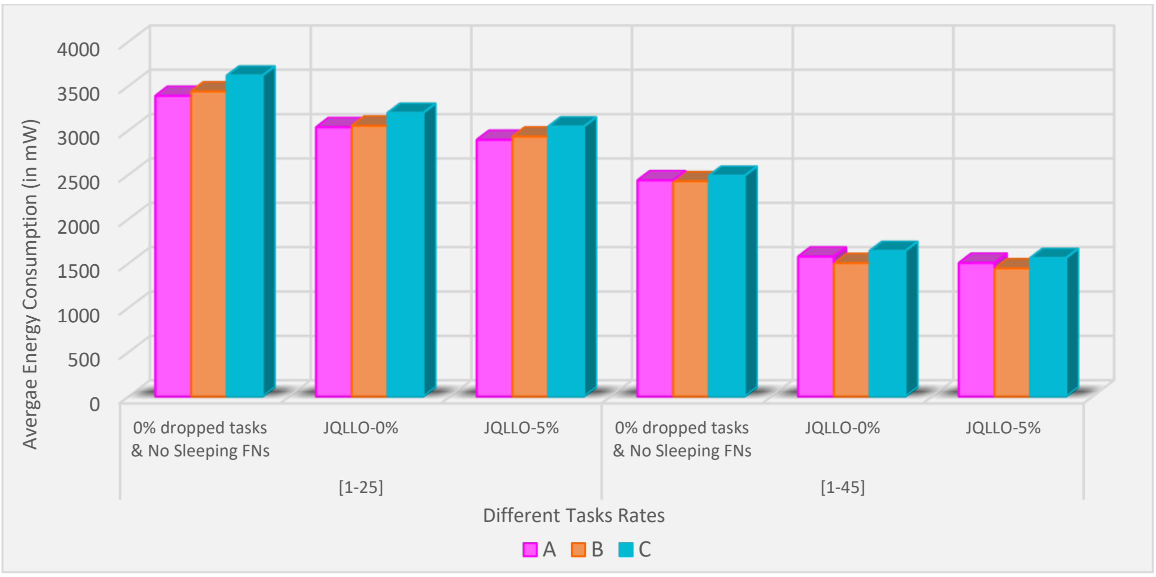


Figure 5.30: The impact of various arrival rates on the average energy consumption.

The results of this experiment show that JQLLO helps to minimise time-average energy consumption by putting some fog nodes in sleep mode, satisfying QoS requirements, and meeting system constraints. The system helps to accomplish that while sacrificing some delay. This is accomplished by combining the use of Q-learning and Lyapunov optimisation.

5.4 Conclusion

In this chapter, we have analysed the dynamic resource management problem and the problem of computational offloading and processing decisions in stochastic fog computing systems. The first subproblem is formulated first as a stochastic optimisation problem, and then transformed into a non-stationary MDP problem. the proposed solution CQL-EUAs effectively helps the learning agents to maximise their average rewards. Additionally, it helps the learning process to converge faster compared to other baselines such as single learning. The impact of CQL-EUAs approach was analysed and evaluated through extensive experiments.

The second subproblem has been solved using Lyapunov optimisation and drift-plus-penalty theory which helps to reformulate the optimisation problem of minimising time-average energy consumption into server sub-problems. Moreover, solving these sub-problems will lead to an optimal solution to the optimisation problem. A set of experiments is conducted to prove the efficiency of the proposed scheme which is based on Joint Q-learning Lyapunov optimisation (JQLLO). The results of the proposed scheme were compared to a set of baselines. We observed from the set of experiments that our proposed scheme minimises energy consumption while meeting system constraints and QoS requirements.

6 Conclusion and Future Works

6.1 Final Conclusion

This study is focused on addressing the problem of computational offloading and resource management in regard to server power management with the aim to optimise average delay and energy consumption at fog devices while meeting QoS requirements in the system, in online and stochastic fog computing system.

To show the contribution of this dissertation, the work has been designed into two main sections to purpose solutions regarding computation offloading and resource management from different angles.

In the first part, computational offloading and resource management problems have been analysed in online dynamic fog systems. This is with the aim of minimising the average delay and conserving energy. In this model, a joint algorithm is proposed to achieve the stated objectives. The problem is addressed with various fixed offloading thresholds. Simulation results demonstrate the efficiency of the proposed scheme in optimising delay, throughput, and energy consumption. In the simulation, the effect of varying offloading thresholds and increasing the number of available neighbours is analysed. The results indicated that selecting the value of the offloading threshold plays a significant role in the performance of the system. Having a very low or very high offloading threshold impacts the system negatively, which emphasizes the importance of having the optimal offloading threshold value. Further, increasing the number of available neighbours improves delay and throughput at the expense of energy consumption. It will however have a negative impact on delay and energy if the number of available neighbours is increased beyond the limit. This is demonstrated in chapter 3. This answers research question 1.

Further, since the optimal value of offloading threshold in an online dynamic system cannot be predicted in advance, we propose a dynamic offloading threshold approach. Experimental evaluation showed the efficiency of the proposed scheme compared to the fixed threshold and other benchmarks in terms of minimising average delay and enhancing throughput. However, this comes at the expense of energy consumption. A further analysis is performed to examine how the dynamic offloading threshold would perform as workload and available neighbour numbers increase. The results show stable behaviour in regard to the average delay as the delay rises slightly as the number of workloads increases. Compared to the fixed offloading threshold and other baselines where the average delay increases very rapidly as workload increases. In terms of the increase in neighbours, the results show a similar impact to that of the fixed offloading threshold. Chapter 3 addresses this and answers research question 1.

In the second part, computational offloading and resource management in a stochastic fog system is investigated with the aim to minimise the average energy consumption. This is while meeting QoS requirements and system constraints. The problem is formulated as a constraint stochastic optimisation problem and decomposed into two subproblems to be solved efficiently. This is addressed in chapter 4 and answers question 2.

Chapter 5 consists of two parts. The first part considers the first sub-problem as a dynamic resource management problem for minimizing average energy consumption. The problem is solved using a joint algorithm of a cooperative approach to a machine learning algorithm and an algorithm named Eliminating Unacceptable Actions. The proposed approach helps learning agents learn faster and more efficiently than a single learning approach. A further analysis is performed to examine how different epsilon approaches and the minimum allowed reward that causes the episode to terminate would impact the performance of the agents. Results showed that lower epsilon values allow the agent to explore a range of options and learn better. Moreover, defining the minimum allowed reward does impact the learning outcome. This answers research questions 2 and 3.

The second part of chapter 5 addresses the second sub-problem. The problem is addressed as computational offloading and decision processing problems and solved using Lyapunov optimisation theory. According to Lyapunov theory, the problem is divided into three subproblems and solved accordingly. Additionally, the results combining the two sub-problems demonstrate the efficiency of the proposed scheme when compared to a set of benchmarks.

6.2 Future works

On the basis of the study given in this thesis, the following elements have been identified as prospective future research directions:

Studying optimisation problems that increase energy efficiency and delay while ensuring important factors like security and enhancing reliability in online dynamic fog systems. Furthermore, taking into account some characteristics that relate to real-world scenarios such as heterogeneous fog/edge devices in terms of processing capacity, various tasks requirements, and mobility of end users.

Further research is required to verify the stochastic fog system methodologies provided in this thesis for optimising several objectives simultaneously. Fog computing environment is known to be dynamic, complex with heterogeneity resources that may need simultaneous optimization of several objectives. Assessing the effect of having a larger number of objectives is a challenging endeavour that requires more study.

For the stochastic optimisation problem, we integrate the use of Q-learning and Lyapunov optimisation to solve it. However, there are other optimisation techniques that are worth exploring. One example of these techniques can be Stochastic Gradient Descent which is utilised to train models to generate classification or predictions according to input data. It is known to be more rapid and effective for massive datasets, since it takes only a limited number of training instances to estimate the gradient. However, there are various obstacles to consider while implementing it. One of these obstacles is being caught in local optima and choosing the optimal learning rate to get excellent outcomes. Moreover, Evolutionary Optimisation Algorithms such as Genetic Algorithms and Particle Swarm Optimization can be used in stochastic optimisation problems. However, these algorithms should be designed to handle large amounts of complex data. Also, the fitness function should be implemented in a way that reflects the dynamic environment.

Although the experiments in this thesis aimed to model the behaviour of distributed systems in terms of having dynamic and stochastic attributes, it would be extremely beneficial to execute the work described in this thesis in a real distributed environment. This is to improve real-world systems for future computing paradigms such as Fog and Edge systems. Applying this is likely to be difficult and costly, but the effort is worthwhile.

References

1. Harbi, Y., Z. Aliouat, A. Refoufi, and S. Harous, *Recent security trends in internet of things: A comprehensive survey.* IEEE Access, 2021. **9**: p. 113292-113314.
2. Aazam, M., S. Zeadally, and K.A. Harras, *Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities.* Future Generation Computer Systems, 2018. **87**: p. 278-289.
3. Ghobaei-Arani, M., A. Souri, and A.A. Rahmanian, *Resource management approaches in fog computing: a comprehensive review.* Journal of Grid Computing, 2020. **18**(1): p. 1-42.
4. Al-Khafajiy, M., T. Baker, H. Al-Libawy, Z. Maamar, M. Aloqaily, and Y. Jararweh, *Improving fog computing performance via fog-2-fog collaboration.* Future Generation Computer Systems, 2019. **100**: p. 266-280.
5. Shakarami, A., M. Ghobaei-Arani, M. Masdari, and M. Hosseinzadeh, *A survey on the computation offloading approaches in mobile edge/cloud computing environment: a stochastic-based perspective.* Journal of Grid Computing, 2020. **18**: p. 639-671.
6. Atoui, W.S., W. Ajib, and M. Boukadoum, *Offline and online scheduling algorithms for energy harvesting RSUs in VANETs.* IEEE Transactions on Vehicular Technology, 2018. **67**(7): p. 6370-6382.
7. Hong, C.-H. and B. Varghese, *Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms.* ACM Computing Surveys (CSUR), 2019. **52**(5): p. 1-37.
8. Shakarami, A., H. Shakarami, M. Ghobaei-Arani, E. Nikougoftar, and M. Faraji-Mehmandar, *Resource provisioning in edge/fog computing: A comprehensive and systematic review.* Journal of Systems Architecture, 2022. **122**: p. 102362.
9. Sun, H., H. Yu, G. Fan, and L. Chen, *Energy and time efficient task offloading and resource allocation on the generic IoT-fog-cloud architecture.* Peer-to-Peer Networking and Applications, 2020. **13**(2): p. 548-563.
10. Wang, Q. and S. Chen, *Latency-minimum offloading decision and resource allocation for fog-enabled Internet of Things networks.* Transactions on Emerging Telecommunications Technologies, 2020: p. e3880.
11. Liu, L., Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, *Multiobjective optimization for computation offloading in fog computing.* IEEE Internet of Things Journal, 2017. **5**(1): p. 283-294.

12. Gao, X., X. Huang, S. Bian, Z. Shao, and Y. Yang, *Pora: Predictive offloading and resource allocation in dynamic fog computing systems.* IEEE Internet of Things Journal, 2019.

13. Xiao, Y. and M. Krunz. *QoE and power efficiency tradeoff for fog computing networks with fog node cooperation*. in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. 2017. IEEE.

14. Chen, M. and Y. Hao, *Task offloading for mobile edge computing in software defined ultra-dense network.* IEEE Journal on Selected Areas in Communications, 2018. **36**(3): p. 587-597.

15. Pu, L., X. Chen, J. Xu, and X. Fu, *D2D fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted D2D collaboration.* IEEE Journal on Selected Areas in Communications, 2016. **34**(12): p. 3887-3901.

16. Guo, M., L. Li, and Q. Guan, *Energy-efficient and delay-guaranteed workload allocation in IoT-edge-cloud computing systems.* IEEE Access, 2019. **7**: p. 78685-78697.

17. Chang, Z., L. Liu, X. Guo, and Q. Sheng, *Dynamic resource allocation and computation offloading for IoT fog computing system.* IEEE Transactions on Industrial Informatics, 2020. **17**(5): p. 3348-3357.

18. Chen, L., J. Xu, and S. Zhou. *Computation peer offloading in mobile edge computing with energy budgets*. in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. 2017. IEEE.

19. Jia, Q., R. Xie, Q. Tang, X. Li, T. Huang, J. Liu, and Y. Liu, *Energy-efficient computation offloading in 5G cellular networks with edge computing and D2D communications.* IET Communications, 2019. **13**(8): p. 1122-1130.

20. Zhang, N., S. Guo, Y. Dong, and D. Liu, *Joint task offloading and data caching in mobile edge computing networks.* Computer Networks, 2020. **182**: p. 107446.

21. Xu, J., L. Chen, and P. Zhou. *Joint service caching and task offloading for mobile edge computing in dense networks*. in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. 2018. IEEE.

22. Zhang, G., W. Zhang, Y. Cao, D. Li, and L. Wang, *Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices.* IEEE Transactions on Industrial Informatics, 2018. **14**(10): p. 4642-4655.

23. Mao, S., S. Leng, S. Maharjan, and Y. Zhang, *Energy efficiency and delay tradeoff for wireless powered mobile-edge computing systems with multi-access schemes.* IEEE Transactions on Wireless Communications, 2019. **19**(3): p. 1855-1867.

24. Hazra, A., M. Adhikari, T. Amgoth, and S.N. Srirama, *Joint computation offloading and scheduling optimization of iot applications in fog networks.* IEEE Transactions on Network Science and Engineering, 2020. **7**(4): p. 3266-3278.

25. Wang, Y., K. Wang, H. Huang, T. Miyazaki, and S. Guo, *Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications.* IEEE Transactions on Industrial Informatics, 2018. **15**(2): p. 976-986.

26. Dai, Y., K. Zhang, S. Maharjan, and Y. Zhang, *Deep reinforcement learning for stochastic computation offloading in digital twin networks.* IEEE Transactions on Industrial Informatics, 2020. **17**(7): p. 4968-4977.

27. Li, M., J. Gao, L. Zhao, and X. Shen, *Deep reinforcement learning for collaborative edge computing in vehicular networks.* IEEE Transactions on Cognitive Communications and Networking, 2020. **6**(4): p. 1122-1135.

28. Ke, H., J. Wang, L. Deng, Y. Ge, and H. Wang, *Deep reinforcement learning-based adaptive computation offloading for MEC in heterogeneous vehicular networks.* IEEE Transactions on Vehicular Technology, 2020. **69**(7): p. 7916-7929.

29. Zhan, W., C. Luo, J. Wang, C. Wang, G. Min, H. Duan, and Q. Zhu, *Deep-reinforcement-learning-based offloading scheduling for vehicular edge computing.* IEEE Internet of Things Journal, 2020. **7**(6): p. 5449-5465.

30. Hazra, A., M. Adhikari, T. Amgoth, and S.N. Srirama, *Collaborative AI-enabled intelligent partial service provisioning in green industrial fog networks.* IEEE Internet of Things Journal, 2021.

31. Chen, X., H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, *Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning.* IEEE Internet of Things Journal, 2018. **6**(3): p. 4005-4018.

32. Yan, J., S. Bi, and Y.J.A. Zhang, *Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach.* IEEE Transactions on Wireless Communications, 2020. **19**(8): p. 5404-5419.

33. Xu, S., Q. Liu, B. Gong, F. Qi, S. Guo, X. Qiu, and C. Yang, *RJCC: Reinforcement-learning-based joint communicational-and-computational resource allocation mechanism for smart city IoT.* IEEE Internet of Things Journal, 2020. **7**(9): p. 8059-8076.

34. Bae, S., S. Han, and Y. Sung, *A Reinforcement Learning Formulation of the Lyapunov Optimization: Application to Edge Computing Systems with Queue Stability.* arXiv preprint arXiv:2012.07279, 2020.

35. Zhang, L., B. Cao, Y. Li, M. Peng, and G. Feng, *A multi-stage stochastic programming-based offloading policy for fog enabled IoT-eHealth.* IEEE Journal on Selected Areas in Communications, 2020. **39**(2): p. 411-425.

36. Madni, S.H.H., M.S.A. Latiff, Y. Coulibaly, and S.i.M. Abdulhamid, *Recent advancements in resource allocation techniques for cloud computing environment: a systematic review.* Cluster Computing, 2017. **20**: p. 2489-2533.

37. Heidari, A., M.A. Jabraeil Jamali, N. Jafari Navimipour, and S. Akbarpour, *Internet of things offloading: ongoing issues, opportunities, and future challenges.* International Journal of Communication Systems, 2020. **33**(14): p. e4474.

38. Akherfi, K., M. Gerndt, and H. Harroud, *Mobile cloud computing for computation offloading: Issues and challenges.* Applied computing and informatics, 2018. **14**(1): p. 1-16.

39. Yin, L., J. Luo, and H. Luo, *Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing.* IEEE Transactions on Industrial Informatics, 2018. **14**(10): p. 4712-4721.

40. Mukherjee, M., S. Kumar, M. Shojafar, Q. Zhang, and C.X. Mavromoustakis. *Joint task offloading and resource allocation for delay-sensitive fog networks*. in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. 2019. IEEE.

41. Zhu, Q., B. Si, F. Yang, and Y. Ma, *Task offloading decision in fog computing system.* China Communications, 2017. **14**(11): p. 59-68.

42. Mukherjee, M., V. Kumar, S. Kumar, R. Matamy, C.X. Mavromoustakis, Q. Zhang, M. Shojafar, and G. Mastorakis. *Computation offloading strategy in heterogeneous fog computing with energy and delay constraints*. in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. 2020. IEEE.

43. Zhao, Z., S. Bu, T. Zhao, Z. Yin, M. Peng, Z. Ding, and T.Q. Quek, *On the design of computation offloading in fog radio access networks.* IEEE Transactions on Vehicular Technology, 2019. **68**(7): p. 7136-7149.

44. Meng, X., W. Wang, and Z. Zhang, *Delay-constrained hybrid computation offloading with cloud and fog computing.* IEEE Access, 2017. **5**: p. 21355-21367.

45. Yousefpour, A., G. Ishigaki, and J.P. Jue. *Fog computing: Towards minimizing delay in the internet of things*. in *2017 IEEE international conference on edge computing (EDGE)*. 2017. IEEE.

46. Mukherjee, M., M. Guo, J. Lloret, R. Iqbal, and Q. Zhang, *Deadline-aware Fair Scheduling for Offloaded Tasks in Fog Computing with Inter-fog Dependency.* IEEE Communications Letters, 2019.

47. Marsan, M.A. and M. Meo, *Queueing systems to study the energy consumption of a campus WLAN.* Computer networks, 2014. **66**: p. 82-93.

48. Li, F., X. Wang, J. Cao, R. Wang, and Y. Bi, *A State Transition-Aware Energy-Saving Mechanism for Dense WLANs in Buildings.* IEEE Access, 2017. **5**: p. 25671-25681.

49. Monil, M.A.H., R. Qasim, and R.M. Rahman, *Energy-aware VM Consolidation Approach Using Combination of Heuristics and Migration Control.*

50. Mosa, A. and N.W. Paton, *Optimizing virtual machine placement for energy and SLA in clouds using utility functions.* Journal of Cloud Computing, 2016. **5**(1): p. 17.

51. Mahadevamangalam, S., *Energy-aware adaptation in Cloud datacenters*. 2018.

52. Monil, M.A.H. and R.M. Rahman. *Implementation of modified overload detection technique with VM selection strategies based on heuristics and migration control*. in *2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS)*. 2015. IEEE.

53. Monil, M.A.H. and R.M. Rahman, *VM consolidation approach based on heuristics, fuzzy logic, and migration control.* Journal of Cloud Computing, 2016. **5**(1): p. 8.

54. Tang, Q., R. Xie, F.R. Yu, T. Huang, and Y. Liu, *Decentralized Computation Offloading in IoT Fog Computing System With Energy Harvesting: A Dec-POMDP Approach.* IEEE Internet of Things Journal, 2020.

55. Wang, S., X. Zhang, Z. Yan, and W. Wenbo, *Cooperative edge computing with sleep control under nonuniform traffic in mobile edge networks.* IEEE Internet of Things Journal, 2018. **6**(3): p. 4295-4306.

56. Arulkumaran, K., M.P. Deisenroth, M. Brundage, and A.A. Bharath, *Deep reinforcement learning: A brief survey.* IEEE Signal Processing Magazine, 2017. **34**(6): p. 26-38.

57. Alwarafy, A., M. Abdallah, B.S. Ciftler, A. Al-Fuqaha, and M. Hamdi, *Deep reinforcement learning for radio resource allocation and management in next generation heterogeneous wireless networks: A survey.* arXiv preprint arXiv:2106.00574, 2021.

58. Gandhi, N. and S. Mishra, *Modelling resource allocation in uncertain system environment through deep reinforcement learning.* arXiv preprint arXiv:2106.09461, 2021.

59. Sutton, R.S. and A.G. Barto, *Reinforcement learning: An introduction*. 2018: MIT press.

60. Szepesvári, C., *Algorithms for reinforcement learning.* Synthesis lectures on artificial intelligence and machine learning, 2010. **4**(1): p. 1-103.

61. Nian, R., J. Liu, and B. Huang, *A review on reinforcement learning: Introduction and applications in industrial process control.* Computers & Chemical Engineering, 2020. **139**: p. 106886.

62. Ullah, Z., F. Al-Turjman, L. Mostarda, and R. Gagliardi, *Applications of artificial intelligence and machine learning in smart cities.* Computer Communications, 2020. **154**: p. 313-323.

63. Liu, X., Z. Qin, and Y. Gao. *Resource allocation for edge computing in IoT networks via reinforcement learning*. in *ICC 2019-2019 IEEE international conference on communications (ICC)*. 2019. IEEE.

64. Yan, M., G. Feng, and S. Qin. *Multi-RAT access based on multi-agent reinforcement learning*. in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. 2017. IEEE.

65. Rahman, G.S., T. Dang, and M. Ahmed, *Deep reinforcement learning based computation offloading and resource allocation for low-latency fog radio access networks.* Intelligent and Converged Networks, 2020. **1**(3): p. 243-257.

66. Sewak, M., *Temporal difference learning, SARSA, and Q-learning*, in *Deep Reinforcement Learning*. 2019, Springer. p. 51-63.

67. Jang, B., M. Kim, G. Harerimana, and J.W. Kim, *Q-learning algorithms: A comprehensive classification and applications.* IEEE access, 2019. **7**: p. 133653-133667.

68. Padakandla, S., P. KJ, and S. Bhatnagar, *Reinforcement learning algorithm for non-stationary environments.* Applied Intelligence, 2020. **50**: p. 3590-3606.

69. Li, J., H. Gao, T. Lv, and Y. Lu. *Deep reinforcement learning based computation offloading and resource allocation for MEC*. in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*. 2018. IEEE.

70. Gaskett, C., D. Wettergreen, and A. Zelinsky. *Q-learning in continuous state and action spaces*. in *Australasian joint conference on artificial intelligence*. 1999. Springer.

71. Belousov, B., H. Abdulsamad, P. Klink, S. Parisi, and J. Peters, *Reinforcement learning algorithms: analysis and applications*. 2021: Springer.

72. Leng, L., J. Li, J. Zhu, K.-S. Hwang, and H. Shi, *Multi-agent reward-iteration fuzzy Q-learning.* International Journal of Fuzzy Systems, 2021. **23**: p. 1669-1679.

73. François-Lavet, V., P. Henderson, R. Islam, M.G. Bellemare, and J. Pineau, *An introduction to deep reinforcement learning.* Foundations and Trends® in Machine Learning, 2018. **11**(3-4): p. 219-354.

74. Ferreira, E. and F. Lefevre. *Expert-based reward shaping and exploration scheme for boosting policy learning of dialogue management*. in *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*. 2013. IEEE.

75. Icarte, R.T., T.Q. Klassen, R. Valenzano, and S.A. McIlraith, *Reward machines: Exploiting reward function structure in reinforcement learning.* Journal of Artificial Intelligence Research, 2022. **73**: p. 173-208.

76. Zhuang, W. and K. Qu, *Dynamic Resource Management in Service-Oriented Core Networks*. 2021: Springer.

77. Tan, M. *Multi-agent reinforcement learning: Independent vs. cooperative agents*. in *Proceedings of the tenth international conference on machine learning*. 1993.

78. Abed-Alguni, B.H., D.J. Paul, S.K. Chalup, and F.A. Henskens, *A comparison study of cooperative Q-learning algorithms for independent learners.* Int. J. Artif. Intell, 2016. **14**(1): p. 71-93.

79. Iima, H. and Y. Kuroe. *Swarm reinforcement learning algorithms based on Sarsa method*. in *2008 SICE Annual Conference*. 2008. IEEE.

80. Iima, H. and Y. Kuroe. *Reinforcement learning through interaction among multiple agents*. in *2006 SICE-ICASE International Joint Conference*. 2006. IEEE.

81. Abed-Alguni, B.H.K., *Cooperative reinforcement learning for independent learners.* Computer Science, 2014.

82. Abedin, S.F., M.G.R. Alam, N.H. Tran, and C.S. Hong. *A Fog based system model for cooperative IoT node pairing using matching theory*. in *2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. 2015. IEEE.

83. Soleymani, S.A., A.H. Abdullah, M. Zareei, M.H. Anisi, C. Vargas-Rosales, M.K. Khan, and S. Goudarzi, *A secure trust model based on fuzzy logic in vehicular ad hoc networks with fog computing.* IEEE Access, 2017. **5**: p. 15619-15629.

84. Zohora, F.T., M.R.R. Khan, M.F.R. Bhuiyan, and A.K. Das. *Enhancing the capabilities of IoT based fog and cloud infrastructures for time sensitive events*. in *2017 International Conference on Electrical Engineering and Computer Science (ICECOS)*. 2017. IEEE.

85. Lee, G., W. Saad, and M. Bennis. *An online secretary framework for fog network formation with minimal latency*. in *2017 IEEE International Conference on Communications (ICC)*. 2017. IEEE.

86. El Kafhali, S., K. Salah, and S.B. Alla. *Performance Evaluation of IoT-Fog-Cloud Deployment for Healthcare Services*. in *2018 4th International Conference on Cloud Computing Technologies and Applications (Cloudtech)*. 2018. IEEE.

87. Liu, L., Z. Chang, and X. Guo, *Socially aware dynamic computation offloading scheme for fog computing system with energy harvesting devices.* IEEE Internet of Things Journal, 2018. **5**(3): p. 1869-1879.

88. Bozorgchenani, A., D. Tarchi, and G.E. Corazza. *An energy and delay-efficient partial offloading technique for fog computing architectures*. in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. 2017. IEEE.

89. Bozorgchenani, A., D. Tarchi, and G.E. Corazza, *Centralized and distributed architectures for energy and delay efficient fog network-based edge computing services.* IEEE Transactions on Green Communications and Networking, 2018. **3**(1): p. 250-263.

90. Bozorgchenani, A., *Energy and Delay Efficient Computation Offloading Solutions for Edge Computing.* 2020.

91. Gupta, H., A. Vahid Dastjerdi, 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, 2017. **47**(9): p. 1275-1296.

92. Qayyum, T., A.W. Malik, M.A.K. Khattak, O. Khalid, and S.U. Khan, *FogNetSim++: A toolkit for modeling and simulation of distributed fog environment.* IEEE Access, 2018. **6**: p. 63570-63583.

93. Li, L., Q. Guan, L. Jin, and M. Guo, *Resource allocation and task offloading for heterogeneous real-time tasks with uncertain duration time in a fog queueing system.* IEEE Access, 2019. **7**: p. 9912-9925.

94. Neely, M.J., A.S. Tehrani, and A.G. Dimakis. *Efficient algorithms for renewable energy allocation to delay tolerant consumers*. in *2010 First IEEE International Conference on Smart Grid Communications*. 2010. IEEE.

95. Deng, Y., Z. Chen, D. Zhang, and M. Zhao, *Workload scheduling toward worst-case delay and optimal utility for single-hop Fog-IoT architecture.* IET Communications, 2018. **12**(17): p. 2164-2173.

96. Alam, M.G.R., M.M. Hassan, M.Z. Uddin, A. Almogren, and G. Fortino, *Autonomic computation offloading in mobile edge for IoT applications.* Future Generation Computer Systems, 2019. **90**: p. 149-157.

97. Neely, M.J., *Stochastic network optimization with application to communication and queueing systems.* Synthesis Lectures on Communication Networks, 2010. **3**(1): p. 1-211.

98. Shen, Q., *Resource Management in E-health Systems.* 2015.

99. Feng, L., Y. Zhou, T. Liu, X. Que, P. Yu, T. Hong, and X. Qiu, *Energy-efficient offloading for mission-critical IoT services using EVT-embedded intelligent learning.* IEEE Transactions on Green Communications and Networking, 2021. **5**(3): p. 1179-1190.

100. Li, Y., S. Xia, M. Zheng, B. Cao, and Q. Liu, *Lyapunov optimization-based trade-off policy for mobile cloud offloading in heterogeneous wireless networks.* IEEE Transactions on Cloud Computing, 2019. **10**(1): p. 491-505.

101. Chen, Y., N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. Shen, *Energy efficient dynamic offloading in mobile edge computing for internet of things.* IEEE Transactions on Cloud Computing, 2019. **9**(3): p. 1050-1060.

102. Zhou, C. and C.-K. Tham. *Deadline-aware peer-to-peer task offloading in stochastic mobile cloud computing systems*. in *2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. 2018. IEEE.

103. Wisniewski, M. and J.H. Klein, *Critical Path Analysis and Linear Programming*. 2017: Bloomsbury Publishing.

104. EDITION, T., *INTRODUCTION TO OPTIMUM DESIGN.* 2012.

105. Yang, Y., S. Zhao, W. Zhang, Y. Chen, X. Luo, and J. Wang, *DEBTS: Delay energy balanced task scheduling in homogeneous fog networks.* IEEE Internet of Things Journal, 2018. **5**(3): p. 2094-2106.