

# Artificial Intelligence-Empowered Resource Management System for Fog Computing Networks

by

Jungyeon Baek

MANUSCRIPT-BASED THESIS PRESENTED TO ÉCOLE DE  
TECHNOLOGIE SUPÉRIEURE IN PARTIAL FULFILLMENT FOR THE  
DEGREE OF DOCTOR OF PHILOSOPHY  
Ph.D.

MONTREAL, MAY 18, 2022

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE  
UNIVERSITÉ DU QUÉBEC



Jungyeon Baek, 2022



This Creative Commons license allows readers to download this work and share it with others as long as the author is credited. The content of this work cannot be modified in any way or used commercially.

**BOARD OF EXAMINERS**

**THIS THESIS HAS BEEN EVALUATED**

**BY THE FOLLOWING BOARD OF EXAMINERS**

M. Georges Kaddoum, Thesis Supervisor  
Department of Electrical Engineering, École de Technologie Supérieure

M. Mohamed Faten Zhani, President of the Board of Examiners  
Department of Software and Information Technology Engineering, École de Technologie Supérieure

M. Julien Gascon-Samson, Member of the jury  
Department of Software and Information Technology Engineering, École de Technologie Supérieure

Mme Anjali Agarwal, External Independent Examiner  
Faculty of Electrical and Computer Engineering, Concordia University

**THIS THESIS WAS PRESENTED AND DEFENDED**

**IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC**

**ON "MAY 11, 2022"**

**AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE**



## **FOREWORD**

This dissertation is written based on the author's Ph.D. research outcomes under the supervision of Professor Georges Kaddoum from September 2017 to December 2021. This work is financially supported by the Natural Sciences and Engineering Research Council of Canada under Project RDCPJ 501617-16 and the Fonds de recherche du Québec under Doctoral Research Scholarships 2020-2021. The main theme of this dissertation focuses on the emerging topic for resource management in fog computing networks. This dissertation is written as a monograph based on two published IEEE journal papers and two submitted IEEE journal papers as the first author.

In this dissertation, the first two chapters present the introduction and the literature review of fog computing and resource management algorithms. Next, the following four chapters are written based on my research journal papers. Finally, the conclusion and the recommendation for future work are given in the last chapter.



## ACKNOWLEDGEMENTS

I am deeply grateful and blessed for having all those people, who support me, uplift me, comfort me, and bring joy in my life.

First and foremost, I would like to express my sincere gratitude to my supervisor, Professor Georges Kaddoum, for having been supportive of my endeavors throughout my PhD. It was a great privilege to work and research under his guidance. Especially, I thank him for always having my best interests at heart and believing in my abilities endlessly. This thesis would not have been possible without his guidance.

I would like to truly thank all my PhD committee members, Professor Mohamed Faten Zhani, and Professor Julien Gascon-Samson, for reviewing and giving invaluable comments for my thesis. I would also like to thank Professor Anjali Agarwal for her time and serving as the external examiner to my PhD defense.

I would like to take this opportunity to deliver my deep gratitude to Professor Een-Kee Hong for his inspiring lessons and for guiding me into the academic path. My thanks also go to my labmates in LACIME. Especially, many thanks to Dr. Ha Vu Tran and Dr. Thanh-Dat Le for giving me help and sincere support whenever I was in need.

I am grateful to my close friends for always supporting me from far away by giving me warm words and encouragement. I would also like to thank my amazing friends who helped me get adjusted and made my life in Canada much more fun.

Special thanks to my fiancé, Vladislav Golovine, for making me realize the real value of life and encouraging me to complete this thesis successfully; I could not have made it without you.

Finally, I would like to acknowledge the support and love of my family: my parents, Yong-ho Baek and Young-eun An, and my brother, Seung-tak Baek. I am forever indebted to my family for believing in me, providing me with all I ever needed, and helping me live my dreams out

## VIII

in the world, even if it hurts them to have me so far away. This journey would not have been possible if not for them, and I dedicate this milestone to my family.



# **Système de gestion des ressources basé sur l'intelligence artificielle pour les réseaux informatiques de brouillard**

Jungyeon Baek

## **RÉSUMÉ**

De nos jours, la technologie est devenue une partie essentielle de la vie humaine et le monde numérique se développe rapidement et progresse en termes de technologies de mise en réseau. L'augmentation rapide des services et des applications sans fil exige des réseaux plus rapides et de plus grande capacité. De plus, l'Internet des objets (IoT) alimente le besoin d'une connectivité massive des appareils et d'interactions ultra-fiables et en temps réel. Dans ce contexte, l'informatique de brouillard (fog computing) est apparu comme une solution attrayante pour répondre à ces exigences tout en gérant de près les demandes croissantes de données en rapprochant les services de l'informatique dans les nuages (cloud computing) des terminaux et en intégrant des serveurs virtualisés. D'une part, en raison des demandes de trafic volatiles et des ressources limitées en capacité (par exemple, calcul, stockage et batteries), les réseaux de brouillard nécessitent une plate-forme intelligente distribuée qui peut s'adapter aux changements du réseau et gérer efficacement l'exécution de tâches complexes basées sur les exigences de candidature. D'autre part, étant donné que les nœuds de calcul sont relativement proches les uns des autres dans les réseaux de brouillard, le déchargement des tâches est particulièrement utile et permet d'équilibrer la charge en répartissant la charge de travail entre les différents nœuds du réseau.

Dans cette thèse, trois objectifs principaux pour la conception de réseaux informatiques intelligents distribués sont considérés. Les trois objectifs sont l'évolutivité, l'hétérogénéité et la gestion de la qualité de service (QoS). Plus précisément, afin d'améliorer l'utilisation des ressources et les performances du réseau, l'hétérogénéité des ressources, la qualité de service et les caractéristiques des tâches sont étudiées. Pendant ce temps, des architectures hiérarchiques de l'informatique de brouillard sont appliquées pour répartir les charges de travail et les communications dans le temps et dans l'espace, permettant une gestion flexible. De plus, des algorithmes de gestion des ressources distribués au niveau des nœuds de brouillard, des points d'accès (AP) et des utilisateurs mobiles (MU), sont mis en œuvre pour gérer et allouer de manière optimale les ressources. En particulier, cette thèse propose des schémas prometteurs pour permettre une gestion des ressources efficace et évolutive pour les réseaux de l'informatique dans les nuages en utilisant l'apprentissage par renforcement (RL) et l'apprentissage profond comme outils principaux.

Dans cette veine, le chapitre 2 étudie le problème mixte de déchargement de tâches et d'allocation de ressources tout en considérant des tâches de service hétérogènes en termes de caractéristiques de ressources et d'exigences de QoS. Nous proposons un schéma dans lequel chaque nœud de brouillard trouve indépendamment les politiques optimales de déchargement des tâches et d'allocation des ressources dans des environnements partiellement observables dans le but de maximiser les tâches de traitement achevées avec succès dans leurs limites de temps.

Par conséquent, une approche basée sur le l'apprentissage par renforcement est proposée pour relever les défis associés aux informations de réseau incomplètes et à l'observabilité partielle.

Le chapitre 3 propose un nouvel algorithme de déchargement partiel et de planification des ressources pour les réseaux multi-brouillards où la quantité de tâches de déchargement, la vitesse de calcul et le niveau d'utilisation du processeur sont conjointement optimisés. Par conséquent, une nouvelle méthode basée sur le réseau Q récurrent profond est fournie pour minimiser la consommation d'énergie totale tout en maximisant le nombre de tâches exécutées avec succès avec une bande passante et des ressources unité centrale de traitement (CPU) limitées.

Contrairement aux chapitres 2 et 3, où les décideurs (nœuds de brouillard) sont entièrement décentralisés, c'est-à-dire des apprenants indépendants qui ne communiquent pas directement entre eux, au chapitre 4, les points d'accès, en tant que décideurs, apprennent à communiquer avec les points d'accès voisins via des canaux de communication limités pour coordonner leur comportement. Dans ce contexte, un nouveau cadre l'apprentissage par renforcement acteur-critique est proposé pour minimiser les liens et serveurs surchargés et le coût global de la bande passante. Nous étendons le modèle acteur-critique selon lequel le réseau critique est conçu pour un apprentissage centralisé en partageant des paramètres entre les points d'accès. En revanche, les réseaux d'acteurs individuels dans chaque points d'accès s'efforcent d'apprendre la politique optimale uniquement en utilisant des messages d'information et de communication locaux. Le schéma proposé peut faire progresser le développement de la communication pour un apprentissage efficace de la périphérie (edge learning) et l'application d'algorithmes d'apprentissage distribué.

Le chapitre 5 présente l'apprentissage par renforcement multi-agents (MARL) et les principales applications potentielles de MARL pour les réseaux de sixième génération (6G). À mesure que les services et applications sans fil deviennent plus sophistiqués et intelligents, il est prévisible que les futurs réseaux sans fil deviendront omniprésents par l'intelligence artificielle (IA). Compte tenu des applications d'IA omniprésentes et des réseaux de communication sans fil dynamiques, il est crucial de créer des agents d'IA capables de s'adapter aux changements du réseau et de coopérer les uns avec les autres. Ce chapitre comprend une étude de cas sur la gestion coordonnée des ressources multi-agents dans les réseaux informatiques de périphérie 6G. L'étude démontre l'importance des méthodes de coordination pour obtenir une intelligence distribuée dans les réseaux 6G.

**Mots-clés:** informatiques de brouillard, déchargement des calculs, répartition de charge, apprentissage par renforcement, réseaux de neurones.

# **Artificial Intelligence-Empowered Resource Management System for Fog Computing Networks**

Jungyeon Baek

## **ABSTRACT**

Nowadays, technology has become an essential part of human life and the digital world is expanding rapidly and advancing in terms of networking technologies. The rapid increase wireless services and applications is demanding faster and higher-capacity networks. Additionally, the Internet-of-Things (IoT) is fueling the need for massive device connectivity and ultra-reliable and real-time interactions. In this context, fog computing has emerged as an appealing solution to meet these requirements while closely handling growing data demands by bringing cloud computing services closer to end devices and integrating virtualized servers. On the one hand, due to volatile traffic demands and capacity-limited resources (e.g., computation, storage, and batteries), fog networks require a distributed intelligent platform that can adapt to network changes and efficiently manage the execution of complex tasks based on the application requirements. On the other hand, given the fact that computing nodes are relatively close to each other in fog networks, task offloading is particularly useful and enables load balancing by distributing the workload among different nodes throughout the network.

In this thesis, three main objectives for designing distributed intelligent computing networks are considered. The three objectives are scalability, heterogeneity, and quality of service (QoS) management. Specifically, in order to improve resource utilization and network performance, heterogeneity in resources, QoS, and task characteristics are investigated. Meanwhile, hierarchical fog computing architectures are applied to distribute workloads and communications across time and space, enabling flexible management. Moreover, distributed resource management algorithms at the fog nodes, access points (AP), and mobile users (MU), are implemented to optimally manage and allocate resources. In particular, this thesis proposes promising schemes to enable efficient and scalable resource management for fog computing networks using reinforcement learning (RL) and deep learning as primary tools.

In this vein, Chapter 2 studies a joint task offloading and resource allocation problem that considers heterogeneous service tasks in terms of resource characteristics and QoS requirements. We propose a scheme in which each fog node independently finds the optimal task offloading and resource allocation policies in partially-observable environments with the aim of maximizing the processing tasks successfully completed within their time limits. Hence, a deep recurrent RL-based approach is proposed to tackle the challenges associated with incomplete network information and partial observability.

Chapter 3 proposes a novel partial offloading and resource scheduling algorithm for multi-fog networks where the amount of offloading tasks, computational speed, and CPU utilization level are jointly optimized. Hence, a novel method based on the deep recurrent Q-network is provided to minimize the total energy consumption while maximizing the number of tasks successfully executed with limited bandwidth and CPU resources.

In contrast to Chapters 2 and 3, where decision-makers (fog nodes) are fully decentralized, i.e., independent learners that do not directly communicate with each other, in Chapter 4, APs as the decision-makers learn to communicate with neighboring APs over limited communication channels to coordinate their behavior. In this context, a new actor-critic RL framework is proposed to minimize the overloaded links and servers and overall bandwidth cost. We extend the actor-critic model whereby the critic network is designed for centralized learning by sharing parameters among the APs. In contrast, the individual actor networks in each AP strive to learn the optimal policy only using local information and communication messages. The proposed scheme can advance the development of communication for efficient edge learning and the application of distributed learning algorithms.

Chapter 5 introduces multi-agent reinforcement learning (MARL) and major potential applications of MARL for sixth-generation (6G) networks. As wireless services and applications become more sophisticated and intelligent, it is foreseeable that future wireless networks will become AI-pervasive. Given the ubiquitous AI applications and dynamic wireless communication networks, it is crucial to build AI agents that are capable of adapting to network changes as well as cooperate with each other. This chapter includes a case study of coordinated multi-agent resource management in 6G edge computing networks. The study demonstrates the importance of coordination methods to achieve distributed intelligence in 6G networks.

**Keywords:** fog computing, computation offloading, load balancing, reinforcement learning, and neural networks.

## TABLE OF CONTENTS

	Page
INTRODUCTION .....	1
CHAPTER 1 LITERATURE REVIEW: EDGE COMPUTING TECHNOLOGY, FOG COMPUTING ARCHITECTURE, AND RESOURCE MANAGEMENT .....	7
1.1 Edge computing technologies .....	7
1.1.1 Definition of Fog Computing, MEC, and Cloudlet .....	7
1.1.1.1 Fog computing .....	7
1.1.1.2 MEC .....	8
1.1.1.3 Cloudlet .....	9
1.1.2 Comparison of Fog Computing, MEC, and Cloudlet .....	10
1.2 Fog Computing Architecture .....	11
1.2.1 Software-Defined Networking (SDN)-based Fog Computing Architecture .....	11
1.2.1.1 Definition of SDN .....	11
1.2.1.2 Recent SDN-based Fog Computing Architectures .....	13
1.2.2 Hierarchical Fog Computing Architecture .....	15
1.2.2.1 Hierarchical Network Architecture .....	15
1.2.2.2 Hierarchical Fog Computing Architecture for Resource management .....	15
1.2.2.3 Recent Approaches to Hierarchical Fog Computing .....	16
1.3 Intelligent Resource Management in Fog Computing Networks .....	19
1.3.1 Distributed Computing Applications .....	19
1.3.1.1 Resource Sharing .....	19
1.3.1.2 Task Scheduling and Resource Allocation .....	20
1.3.1.3 Offloading and Load Balancing .....	22
1.3.2 Learning Approaches for Intelligent Systems .....	24
1.3.2.1 Reinforcement Learning .....	24
1.3.2.2 Neural Network and Deep Reinforcement Learning .....	26
1.3.2.3 Recurrent Neural Network .....	29
1.3.2.4 Multi-Agent Learning .....	30
1.3.3 Recent Approaches to Resource Management in Fog Computing .....	31
1.3.3.1 Related Work on Cooperative Computing between Fog and Cloud .....	31
1.3.3.2 Related Work on Task Offloading and Scheduling .....	33
1.3.3.3 Related Work on Load Balancing .....	36
CHAPTER 2 HETEROGENEOUS TASK OFFLOADING AND RESOURCE ALLOCATIONS VIA DEEP RECURRENT REINFORCEMENT	

	LEARNING IN PARTIAL OBSERVABLE MULTI-FOG NETWORKS	41
2.1	Introduction	41
2.2	System description	48
2.2.1	Three-layer fog network system	49
2.2.2	SDN-based fog nodes and inter-SDN communications	51
2.2.3	Fog slices based on heterogeneous task models	51
2.2.4	Calculation of task latency	53
2.3	Problem formulation	56
2.3.1	Partially observable MDP based problem formulation	56
2.3.2	Cooperative games by independent learners	59
2.4	Learning the optimal offloading and resource allocation policies	60
2.4.1	Optimal policy solution using Q-learning	60
2.4.2	Convergence to equilibrium	62
2.4.3	Deep Q-learning with nonlinear transformation	63
2.4.4	Deep-recurrent Q-learning for partial observability	64
2.5	Performance evaluation	67
2.5.1	Simulation settings	68
2.5.2	Performance analysis	71
2.5.2.1	Complexity analysis	71
2.5.2.2	Convergence performance	72
2.5.2.3	Performance under different characteristics of fog slices	73
2.5.2.4	Performance under different task arrival rates	75
2.6	Conclusion	77
CHAPTER 3	ONLINE PARTIAL OFFLOADING AND TASK SCHEDULING IN SDN-FOG NETWORKS WITH DEEP RECURRENT REINFORCEMENT LEARNING	79
3.1	Introduction	79
3.2	System description	84
3.2.1	Hierarchical Fog System Architecture	85
3.2.2	User Workload Model	86
3.2.3	Energy Consumption Model	87
3.3	Problem formulation	89
3.3.1	POMDP-based Task Offloading Problem	89
3.3.2	Optimal Control Policy	92
3.4	Learning the optimal computation offloading and task scheduling policies	93
3.4.1	Optimal policy solution using Q-learning	93
3.4.2	Deep recurrent Q-learning with non-linear transformation	94
3.5	Performance evaluation	97
3.5.1	Simulation settings	97
3.5.2	Performance analysis	99

3.5.2.1	Complexity analysis .....	99
3.5.2.2	Convergence performance .....	100
3.5.2.3	Performance under various average task arrival rates .....	100
3.6	Conclusion .....	104
CHAPTER 4 FLOADNET: LOAD BALANCING IN FOG NETWORKS WITH COOPERATIVE MULTI-AGENT USING ACTOR- CRITIC METHOD .....		
4.1	Introduction .....	105
4.2	System description and assumptions .....	110
4.2.1	Three-Layer Fog Networks .....	112
4.2.2	Task offloading and load balancing model .....	113
4.2.3	Model objective and assumptions .....	114
4.3	Game-theoretical problem formulation .....	118
4.3.1	Stochastic game-based problem formulation .....	118
4.3.2	Finding an equilibrium solution in stochastic game .....	119
4.4	Learning the optimal policy using actor-critic methods in the multi-agent problem .....	121
4.4.1	Properties of the proposed learning algorithm .....	121
4.4.2	Learning the optimal policy and communication protocol via Actor-critic methods .....	122
4.5	Performance evaluation .....	126
4.5.1	Simulation settings .....	126
4.5.2	Performance analysis .....	128
4.5.2.1	Convergence performance .....	128
4.5.2.2	Variation in weights of link cost $\varpi_1$ .....	130
4.5.2.3	Variation in weights of link utilization $\varpi_2$ .....	131
4.5.2.4	Variation in weights of server buffer length $\varpi_3$ .....	131
4.5.2.5	Variation in bandwidth and computation resource demands .....	132
4.6	Conclusion .....	135
CHAPTER 5 TOWARD 6G: A NEW ERA OF DISTRIBUTED INTELLIGENCE WITH MULTI-AGENT REINFORCEMENT LEARNING .....		
5.1	Introduction .....	137
5.2	Overview and Motivations .....	139
5.3	Case Study: Multiple Agents with Coordination for Resource Management .....	142
5.4	Applications of MARL for Distributed Intelligence in 6G Networks .....	144
5.4.1	Autonomous Driving .....	145
5.4.2	Cognitive Radio Network .....	146
5.4.3	Edge Computing .....	147
5.5	Open challenges and future works .....	148
5.5.1	Compact and heterogeneous MARL model .....	148
5.5.2	The combination of RL and other AI techniques .....	148

5.5.3	Reward specification and credit assignment .....	149
5.6	Conclusion .....	149
CONCLUSION AND RECOMMENDATIONS .....		151
6.1	Conclusion and Lessons that we learned .....	151
6.2	Recommendations: Fog computing and resource management for 6G networks .....	154
6.2.1	Fog computing-based WSN systems .....	154
6.2.2	Fog computing for massive MIMO beamforming .....	155
6.2.3	Generative adversarial network (GAN) for resource managements in fog computing networks .....	156
6.2.4	Digital twin for reliable and efficient fog computing networks .....	156
LIST OF REFERENCES .....		158



## LIST OF TABLES

	Page
Table 2.1	List of Notations ..... 49
Table 2.2	Two-level of heterogeneity values to service tasks in simulation ..... 69
Table 2.3	Simulation cases according to the three slices' characteristics ..... 71
Table 3.1	List of Notations ..... 84
Table 4.1	List of Notations ..... 111
Table 4.2	The values of topology related parameters ..... 127



## LIST OF FIGURES

		Page
Figure 0.1	ITU estimations of global mobile traffic with and without machine-to-machine (M2M) traffic in 2020-2030 .....	2
Figure 1.1	Example of hierarchical fog architecture .....	17
Figure 1.2	Deep Q-Network based fog computing system.....	28
Figure 1.3	Recurrent neural network.....	30
Figure 2.1	Three-layer fog network system.....	50
Figure 2.2	Application of a deep Q-network (DQN) to approximate the optimal joint offloading and resource allocation control policy of the SDN-based fog nodes .....	65
Figure 2.3	The proposed DRQN structure with GRU .....	66
Figure 2.4	The convergence property of the proposed algorithm using different neural networks.....	73
Figure 2.5	Performance of (a) average success rate and (b) average overflow rate of normal traffic under different cases .....	75
Figure 2.6	Performance of (a) average success rate and (b) average overflow rate of heavy traffic under different cases .....	76
Figure 2.7	Task delay performance of (a) normal traffic and (b) heavy traffic under different cases.....	77
Figure 2.8	Performance of (a) average success rate, (b) average overflow rate, and (c) average task delay of Case-2 under different task arrival rates .....	77
Figure 3.1	A hierarchical fog network system. ....	85
Figure 3.2	The convergence property of the proposed algorithm.....	101
Figure 3.3	Average success rate per fog node under different average task arrival rates .....	101
Figure 3.4	Average overflow rate per fog node under different average task arrival rates .....	102

Figure 3.5	Average transmission energy and CPU energy consumption per fog node under different average task arrival rates .....	103
Figure 4.1	A hierarchic edge-fog-cloud computing network .....	112
Figure 4.2	The proposed multi-agent learning using the Actor-critic policy gradient method with communication messages .....	123
Figure 4.3	The convergence property across the learning procedure and versus depths of neural network .....	129
Figure 4.4	The average payoff performance across the learning procedures.....	129
Figure 4.5	Average link cost, maximum link utilization rate, and average buffer length across the learning procedure versus link cost weight $\varpi_1$ .....	130
Figure 4.6	Average link cost, maximum link utilization rate, and average buffer length across the learning procedure versus link utilization weight $\varpi_2$ .....	131
Figure 4.7	Average link cost, maximum link utilization rate, and average buffer length across the learning procedure versus server buffer length weight $\varpi_3$ .....	132
Figure 4.8	Average link cost and maximum link utilization rate across the learning procedure versus average resource demand of tasks .....	133
Figure 4.9	Average buffer length and average overflow ratio across the learning procedure versus average resource demand of tasks .....	134
Figure 5.1	(a) A flowchart of MARL model settings (b) MARL architecture for decentralized policies with a centralized control unit (c) MARL architecture for decentralized policies with networked agents .....	140
Figure 5.2	Convergence of MARL algorithms .....	144
Figure 5.3	Elements of three key applications for physical, network, and application layers in wireless communication systems .....	145

## LIST OF ABBREVIATIONS

5G	the fifth generation
6G	the sixth generation
AI	artificial intelligence
AP	access point
BS	base station
CRN	cognitive radio network
DNN	deep neural network
DQN	deep Q-network
DRL	deep reinforcement learning
ETSI	European Telecommunications Standards Institute
ICT	information and communication technology
IoT	internet of things
IIoT	industrial internet of things
IoV	internet of vehicles
ITU	international telecommunication union
MARL	multi-agent reinforcement learning
MDP	markov decision process
MEC	multiple access edge computing
ML	machine learning

MU	mobile user
MIMO	multiple input multiple output
NFV	network function virtualization
NN	neural network
POMDP	partial observable markov decision process
QoS	quality of service
RAN	radio access network
RL	reinforcement learning
RNN	recurrent neural network
RRH	remote radio heads
SDN	software defined networking
SP	service provider
VM	virtual machine
WSN	wireless sensor network

## INTRODUCTION

### **Motivation, challenges, and research objective**

Managing the data traffic generated by mobile devices, the Internet of things (IoT), and sensors is one of the biggest challenges expected in the development of future wireless networks. According to Cisco (2018), by 2023, 29.3 billion networked devices will be connected to the Internet, and global mobile devices will grow from 8.8 billion in 2018 to 13.1 billion. The International telecommunication union (ITU) in [ITU (2015)] estimates that each mobile subscriber will consume 39.4 GB of average data traffic per month in 2025, and this will be around 257 GB in 2030.

To handle such explosive traffic demands, a shared pool of computing resources, such as cores, memory, and storage in geographically remote cloud data centers has been an important trend over the past decade. However, traditional cloud computing is encountering growing challenges. The fundamental challenge is the connectivity between the cloud and end devices. Such connectivity is set over the Internet, which is not suitable for a large set of applications, such as delay-sensitive services, due to high communication latency. Well-known examples of delay-sensitive applications include autonomous vehicles, augmented reality (AR)/virtual reality (VR), live-streaming, and a broad range of mobile applications. Furthermore, such applications are mainly distributed and consumed at the network edge, where traditional cloud computing induces network bottlenecks and privacy gaps due to the transfer of traffic from end devices to the remote cloud center. In addition, the irreplaceable dependency on cloud computing demands the data centers to be continuously up and running, which consumes a huge amount of power and yields a large carbon footprint [Sarkar, S., Chatterjee, S. & Misra, S. (2018)].

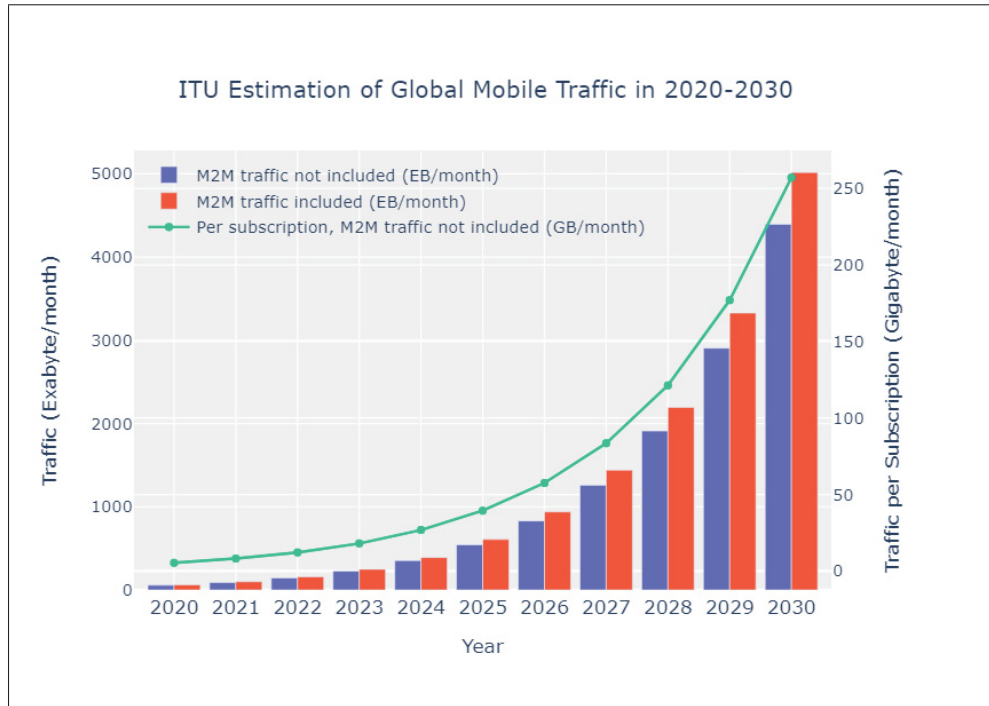


Figure 0.1 ITU estimations of global mobile traffic with and without machine-to-machine (M2M) traffic in 2020-2030  
Taken from [ITU (2015)]

To address these challenges, fog computing has been introduced as a solution that brings cloud services physically and logically closer to ‘Things’ (e.g., sensors, mobile phones, switches, routers, vehicles) [Peng, M., Yan, S., Zhang, K. & Wang, C. (2016); Chiang, M. & Zhang, T. (2016)]. For example, commercial edge routers with high processing speed, a large number of cores, and a connection to the outer layers of the network, can become a server for the fog computing network [Tordera, E. M., Masip-Bruin, X., Garcia-Alminana, J., Jukan, A., Ren, G.-J., Zhu, J. & Farré, J. (2016)]. Fog computing is a highly virtualized platform that builds and offers computing capabilities by an end-to-end architecture [Yi, S., Li, C. & Li, Q. (2015); Buyya, R. & Dastjerdi, A. (2016)]. Fog computing is versatile due to its diverse range of features, such as computation, networking, storage, and control. Fog computing supporting real-time processing, analysis, and decision-making can accelerate the deployment of mission-



critical applications. Some of the expected benefits from the deployment of fog computing are new ultra-low latency services, traffic optimization, and energy efficiency improvements.

From an architectural perspective, hierarchical computing architectures have been suggested for fog computing [OpenFog Consortium Architecture Working Group (2017); Hu, P., Dhelim, S., Ning, H. & Qiu, T. (2017)]. This structure is composed of fog nodes located between the cloud and end devices, which helps distribute the processing of delay-aware services and applications [Sarkar, S. & Misra, S. (2016)]. Fog nodes can be either physical elements (e.g., base stations (BS), switches, gateways) or virtual elements (e.g., virtual machines (VMs)), which typically increase in capacity as they get closer to the cloud.

Although moving services to the fog nodes introduces many advantages, such as lower response time and bandwidth, distributing the logic to different network nodes introduces new issues and challenges. First, the fog computing network must consider the heterogeneity of the fog nodes, with various degrees of computational and storage capabilities and energy constraints. Further, because these nodes are connected to the local devices that are distributed over a large geographic area, the traffic demands are likely to be extremely dynamic. Imbalanced fog nodes that handle highly disparate traffic volumes would lead to inefficient use of resources and unequal quality of service (QoS). Moreover, due to the multi-layer and distributed environments, traditional resource management schemes need to be redesigned to accommodate the fog computing paradigm.

In this context, the need for smart and autonomous network designs has become a central research topic in fog computing implementations. Taking the aforementioned challenges into account, this thesis aims to design a distributed intelligent platform in which the fog nodes can interact and cooperate to satisfy stringent QoS requirements as well as self-adapt to time-varying environments with uncertainties. The growing intelligence of wireless networks motivates the development of learning-based approaches. In particular, we focus on reinforcement

learning (RL) approaches that can find the optimal control policy for unknown environments by learning from experience. Learning-based approaches are expected to play a critical role in the deployment of distributed, intelligent, and flexible networks. More specifically, future fog computing networks require high levels of intelligence to ensure efficient, robust, and resilient resource management.

### **Contributions and Outline**

The organization of this dissertation, which includes five chapters, is structured and detailed as follows. In Chapter 1, a comprehensive literature review of edge computing technology and resource management in fog computing networks is provided. In this vein, the concepts of distributed computing and related technologies are presented. Additionally, recent works on resource management and system designs in fog networks are discussed. In particular, we focus on learning-based approaches for modeling and solving distributed resource management problems.

Chapter 2 presents the first article studying a computational task offloading technique in a fog network and proposing a novel task offloading and resource allocation scheme for a two-level heterogeneous task model. More specifically, service tasks are classified into types according to two characteristics, i.e. resource configurations (required resource sizes) and QoS requirements. In this context, we develop a novel architecture in which each fog node independently finds the optimal task offloading and resource allocation policies in partially-observable environments. The aim is to maximize the processing tasks completed within their delay time limits. To tackle the challenges associated with incomplete network information and partial-observability, we develop a deep reinforcement learning (DRL)-based scheme and extend it by combining recurrent neural networks (RNN).

Chapter 3 presents the second article proposing a novel partial offloading and resource scheduling algorithm in a hierarchical fog network. In this model, two kinds of service types are considered: service types that can be offloaded to other nodes for execution and service types that can only be processed by the node where the data originally arrived. With the stochastic nature of task arrivals and limited resources, independent fog nodes are modeled as a stochastic game. A deep recurrent Q-network (DRQN) is used to find an optimal joint policy that minimizes the total energy consumption while maximizing the number of tasks completed within their deadline. In particular, the number of offloading tasks, computational speed, and CPU utilization level are jointly optimized under service requirements and limited bandwidth and CPU resources constraints.

Chapter 4 presents the third article developing a novel load balancing scheme in a combined edge-fog-cloud environment. The joint optimization problem is formulated with multiple cooperative access points (APs) to minimize the number of overloaded network links and servers as well as the overall link bandwidth cost. In this context, we propose a novel multi-agent actor-critic policy gradient method to design the load balancing scheme in fog networks. To enhance our proposed distributed learning method, we extend the actor-critic model with communication protocols to coordinate the behavior of the individuals and improve the overall learning performance.

Chapter 5 presents the fourth article introducing the importance of multi-agent reinforcement learning (MARL) in future wireless networks and discussing the potential applications of MARL in 6G networks. While single-agent RL methods have rapidly emerged in the wireless communication domain, future wireless networks will face a wide range of multi-agent problems in which many distributed nodes need to make independent decisions based on their local observations. Furthermore, future intelligent objects (IoT devices, network nodes, BSs, etc.) will have to cooperate with other intelligent objects and humans as artificial intelligence

(AI) functions become pervasive in wireless applications. Thus, investigating the MARL developments in the context of wireless communication is of critical importance. Moreover, we provide open challenges and core research questions to be considered in future wireless applications.

## **CHAPTER 1**

### **LITERATURE REVIEW: EDGE COMPUTING TECHNOLOGY, FOG COMPUTING ARCHITECTURE, AND RESOURCE MANAGEMENT**

#### **1.1 Edge computing technologies**

The critical need for IoT devices and near-user edge devices to carry out substantial data processing with ultra-low latency triggered adaptive and decentralized computational paradigms that complement the centralized cloud computing model. Thus, a new computational paradigm, called edge computing, has been introduced to bridge these technological gaps. Edge computing is a distributed computing paradigm that selectively moves some functionalities of the cloud (e.g., computation, control, and decision-making) to the vicinity of end-users [Kaur, K., Dhand, T., Kumar, N. & Zeadally, S. (2017); Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J. & Polakos, P. A. (2017)].

This section describes three different technologies (Fog computing, Multi-access edge computing (MEC), and Cloudlet) within the realm of edge computing and highlights how fog computing differs from other technologies. In addition, related works on the fog computing architecture and its motivations are discussed.

##### **1.1.1 Definition of Fog Computing, MEC, and Cloudlet**

###### **1.1.1.1 Fog computing**

Fog computing represents a platform that brings cloud computing to the proximity of end-users. The term “Fog” was initially introduced by Cisco and proposed in the area of IoT networks to help execute applications and services [Bonomi, F., Milito, R., Zhu, J. & Addepalli, S. (2012)]. As the clouds are far above the sky, the fog is closer to the ground. The same concept is used by fog computing, in which fog nodes as a distributed computing unit are deployed at any point between the end devices and the cloud. The fog nodes are heterogeneous and can be

different kinds of elements with processing capabilities, including but not limited to routers, APs, IoT gateways, and BSs. The heterogeneity of the nodes enables them to support devices with different protocol layers and access technologies [Tordera *et al.* (2016)].

Although both cloud and fog paradigms share an almost similar set of functionalities, such as computation, storage, and networking, there are some differences. Fog deployments target a specific geographic region [Yi *et al.* (2015)]. Moreover, the fog is specifically designed for applications requiring real-time response with minimum latency, e.g., mission-critical applications. Alternatively, the cloud is centralized and is mostly far from the user. Hence, it suffers from some performance limitations regarding latency and response time for real-time applications. This makes cloud-only solutions impractical for many use-cases. In other words, because of its capability to support services that require fast analysis and decision-making, the fog computing platform is suited for use-cases with demanding requirements for scalability, low latency, availability, and bandwidth.

#### **1.1.1.2 MEC**

Mobile edge computing is designed to bring cloud computing capabilities and IT services at the edge of cellular networks [Ahmed, E. & Rehmani, M. H. (2017)]. MEC offers low latency, proximity, context and location awareness, and high network bandwidth. MEC servers are deployed at cellular BSs enabling flexible and rapid deployment of new applications and services. It can also be deployed at the LTE macro base station (eNodeB) sites, 3G radio network controller (RNC), and multi-radio access technology (RAT) sites. MEC can be envisioned as cloud servers running at the edge of mobile networks and performing specific tasks that cannot be achieved with the traditional cloud network infrastructure. Instead of forwarding all traffic to the remote cloud, the MEC shifts traffic targeted for the centralized cloud to the MEC servers. In this way, the MEC servers run applications and perform related processing tasks closer to the cellular customers, reducing network congestion and application response time [Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S. & Sabella, D. (2017)].

MEC was announced in 2014 by the European telecommunications standards institute (ETSI) as an industry specification [Patel, M., Naughton, B., Chan, C., Sprecher, N., Abeta, S., Neal, A. et al. (2014)]. An Industry Specification Group (ISG) within ETSI has been developing system architectures and standardizing some APIs essential for MEC. In 2013, Nokia introduced MEC as a step toward autonomous driving. BSs with distributed MEC have shown an end-to-end latency of less than 20 ms, while the latency between vehicles and a central cloud is usually more than 100 ms [Chamola, V., Tham, C.-K. & Chalapathi, G. S. (2017)]. However, the initial MEC scope was expanded in March 2017 to encompass non-mobile network requirements. With the replacement of ‘Mobile’ by ‘Multi-Access’ in the name, multi-access edge computing (MEC) provides a new ecosystem in which operators can open their radio access network (RAN) edge to authorized third parties, allowing them to flexibly and rapidly deploy innovative applications as a result of the scope expansion [ETSI (2018)].

#### **1.1.1.3 Cloudlet**

Despite the significant technological advances, mobile devices, such as smartphones and tablets, still lack resources compared to other stationary devices, like laptops and wireless access points. Meanwhile, there is a significant increase in the development of various mobile applications. Most of the emerging applications require more resources to handle their service demands with minimum latency [Kaur *et al.* (2017)]. To meet these requirements, cloudlets were designed explicitly as a virtualization feature that provides computing resources to mobile users. The concept of cloudlet was proposed by [Satyanarayanan, M., Bahl, P., Caceres, R. & Davies, N. (2009)]. A cloudlet is a “data center in a box” that can provide cloud services closer to mobile users. Cloudlets reuse modern cloud computing techniques such as VM-based virtualization [Satyanarayanan, M., Lewis, G., Morris, E., Simanta, S., Boleng, J. & Ha, K. (2013)]. Thanks to the VM technology, cloudlets can dynamically scale up and down to support the broadest possible range of mobile users with minimal software constraints.

Using cloudlets, mobile devices run one or more VMs that can offload resource-intensive computational tasks to ensure a real-time interactive response. The current definition of cloudlets

has no regard for the interaction with the cloud. Instead, cloudlets can act as a complete cloud on the edge because they can exist as a standalone environment without cloud intervention [Satyanarayanan *et al.* (2009)]. In addition, a cloudlet is based on standard cloud technologies, making it similar to traditional cloud infrastructures, such as Amazon and OpenStack [Ha, K. & Satyanarayanan, M. (2015)].

### 1.1.2 Comparison of Fog Computing, MEC, and Cloudlet

This discussion focuses on fog computing, MEC, and cloudlet, where most of the currently available edge technology works are based on these technologies. Fog computing, MEC, and cloudlet essentially propose the use of proximity computing resources rather than remote resources in data centers and rely on virtualization. However, there are a few subtle differences between these technologies that need to be distinguished.

First, the three technologies were proposed and are being developed by different organizations. MEC is driven by ETSI, an industry consortium that has been developing technical standards for MEC. Cloudlet was introduced by the cloud computing research community, and an implementation prototype was developed as a research project [Mouradian *et al.* (2017)]. Fog computing, on the other hand, originated from the area of networking, where OpenFog was founded by high-tech companies and academic institutions with the goal of producing a standard specification for fog computing [OpenFog Consortium (2017)].

Another difference is that cloudlet solely relies on VM technology for virtualization, whereas MEC and fog consider virtualization technologies other than VM [Bilal, K., Khalid, O., Erbad, A. & Khan, S. U. (2018)]. Furthermore, each technology targets different central applications. Cloudlets focus primarily on mobile offloading applications, while MEC targets any application that is better provisioned at mobile or non-mobile edges. Fog computing offers more flexibility in choosing devices that can span the cloud and edge. Because fog nodes leverage legacy devices by adding processing and storage, applications can be fully provisioned anywhere between the end-devices and the cloud, but typically have less computation and storage



capacities than the cloud. Moreover, an abstraction layer is required in fog networks due to the heterogeneity and diversity of fog nodes, which is unnecessary in MEC and Cloudlets as they use dedicated devices as nodes. Unlike cloudlet and MEC, fog is closely linked to the presence of the cloud. This has driven particular attention to the interaction between fog and cloud.

## **1.2 Fog Computing Architecture**

The goal of the fog network is to connect every component along a continuum from cloud to things. However, managing such a network, maintaining connectivity, and providing services is challenging, especially in massive connectivity scenarios. This section discusses available works related to the architectural aspects of fog computing.

### **1.2.1 Software-Defined Networking (SDN)-based Fog Computing Architecture**

#### **1.2.1.1 Definition of SDN**

SDN has emerged to provide a flexible and scalable architecture that can handle network congestion and easily maintain a network environment. In SDN, the data and control planes are separated from each other to reduce network congestion and complexity. In general, the SDN's communication infrastructure operates according to standards designed by the Open Networking Foundation (ONF). The OpenFlow (OF) protocol is used to handle traffic flows in SDN [Mahmood, K., Chilwan, A., Østerbø, O. & Jarschel, M. (2015)]. The SDN-based architecture consists of three separate planes: data, control, and application planes [Benzekki, K., El Fergougui, A. & Elbelrhiti Elalaoui, A. (2016)]. The operation of these planes is discussed below.

#### **- SDN Data plane**

The data plane in SDN consists of all forwarding devices, such as OF physical switches, OF virtual switches, OF routers, and OF gateways [Jain, R. & Paul, S. (2013)]. All these forwarding devices act on the forwarding decisions made by the controller in the SDN control plane. These decisions are organized into a flow table in the delivery device using

a data control plane interface. The flow tables operate according to the instructions added to the instruction set available on the controller [Aujla, G. S., Chaudhary, R., Kumar, N., Rodrigues, J. J. & Vinel, A. (2017)].

#### - **SDN Control plane**

The control plane is the core of the SDN architecture and serves as the decision-making plane. This plane works according to the centralized control logic provided to the controller. The main functions of this plane are to install control commands on the forwarding devices, manage and keep global information of all SDN applications running at the application plane, and collect feedback from the forwarding devices. Therefore, the SDN controller provides an abstract model of the underlying network for the SDN application layer. Moreover, the controller can use the network operating system to create a virtual controller using a hypervisor. One of the essential characteristics of SDN is that the control logic can be programmed and reconfigured according to the environment [Li, H., Dong, M. & Ota, K. (2016)].

#### - **SDN Application plane**

With the help of network virtualization, the controller creates multiple virtual networks on the physical network. Virtualization allows multiple virtual machines to run multiple SDN applications simultaneously. Hence, network virtualization is an efficient solution for handling extensive resources. It shares resources, provides isolation between users, and aggregates small resources across physical devices [Mahmood *et al.* (2015)]. SDN applications are software programs that run to manage the resources and networks efficiently. With the interface between the application and the control plane, the control logic generated by the SDN controller handles internal decisions directly and maintains an abstract view of the network.

### 1.2.1.2 Recent SDN-based Fog Computing Architectures

The deployment of SDN can enable the implementation and management of many aspects of fog computing, such as resource allocation, VM migration, topology monitoring, application-aware control, and programmable interfaces. The potential of this concept lies in the fact that traffic engineering and resource management can be performed more efficiently in a centralized system with insights into the applications' requirements and available resources. Fog nodes in a fog computing network can be managed by an SDN controller to address the applications that require mobility support and minimum delay [Poularakis, K., Qin, Q., Nahum, E., Rio, M. & Tassiulas, L. (2017)]. The role of the controller is to manage and keep the global information of all SDN applications running at the application plane, which enables the control logic to be programmed and reconfigured very easily according to different environments. Thus, SDN is the most viable network technology in fog environments [Kaur, K., Garg, S., Aujla, G. S., Kumar, N., Rodrigues, J. J. & Guizani, M. (2018)].

In [Tomovic, S., Yoshigoe, K., Maljevic, I. & Radusinovic, I. (2017)], the authors described an IoT architecture model that utilizes the SDN and fog computing paradigms. They analyzed generic IoT scenarios where features of both technologies are combined in one integrated system. The proposed system structure consists of end devices equipped with multiple wireless interfaces, SDN controllers, a heterogeneous fog infrastructure (virtualized servers, routers, access points, etc.), and cloud in the network core. Since IoT applications are geographically distributed, they assumed a hierarchical deployment of the fog network.

The authors in [Vilalta, R., Lopez, V., Giorgetti, A., Peng, S., Orsini, V., Velasco, L., Serral-Gracia, R., Morris, D., De Fina, S., Cugini, F., Castoldi, P., Mayoral, A., Casellas, R., Martinez, R., Verikoukis, C. & Munoz, R. (2017)] proposed a novel fog computing architecture called TelcoFog. which can be deployed at the edge of wired and wireless networks for telecom operators to provide new 5G services in a unified and cost-effective manner. TelcoFog combines SDN, network function virtualization (NFV), and MEC into its architecture to enable distributed and programmable fog technologies [Garg, S., Kaur, K., Kaddoum, G. & Guo, S.

(2021)]. The key benefit of TelcoFog is its ability to dynamically and intelligently allocate network resources for low-latency services that are spread across the entire network.

In [Cao, B., Sun, Z., Zhang, J. & Gu, Y. (2021a)], the authors developed a novel 5G Internet-of-vehicles (IoV) architecture based on fog computing and SDN. Considering the high mobility of vehicles, the concept of a fog cluster was introduced to provide services to each connected vehicle, thereby preventing frequent handovers of service data. The roadside units (RSUs) in the fog cluster collect road and vehicle information and deliver it to the RSU controller. The RSU controller is the decision center of the fog layer that can make real-time decisions about the allocation of heterogeneous resources for different service types. On the other hand, the SDN controller resides in the cloud to maintain a global network state and forward different service rules from the application layer to the RSU controller. As a result, the dual-layer control model of the RSU and SDN controllers simplifies management and configuration, and improves the real-time performance and scalability of the system.

A novel SDN-based multi-layer routing solution targeting fog-based deployments was proposed in [Bellavista, P., Giannelli, C. & Montenero, D. D. P. (2020)]. This solution addresses the challenges arising from the increased node heterogeneity in terms of hardware/software, time-varying applications that can be served by multiple service providers simultaneously, and frequent node joins/leaves. Their SDN controller configures the appropriate multi-layer routing forwarding mechanism and determines the most suitable path based on its centralized point of view.

The work in [Bellavista *et al.* (2020)] investigated next-generation SDN networks based on P4 and P4Runtime, which are a data plane programming language and runtime protocol that control P4-defined switches, respectively. The authors argued that fog networks could especially benefit from P4/P4Runtime as local and remote SDN controllers can work together for the same data planes. For example, a local controller can handle requests that demand real-time responses (e.g., task execution), while a remote controller is responsible for making decisions that require a global view of the network (e.g., routing selection). As a result, they proposed

a novel SDN control plane design that manages both the SDN data plane belonging to the fog network and the SDN data plane deployed close to the core network. In other words, the SDN controllers can manage cross-layer SDN data planes to offload delay-critical functionalities to the edge and transfer delay-tolerant applications to the cloud.

## **1.2.2 Hierarchical Fog Computing Architecture**

### **1.2.2.1 Hierarchical Network Architecture**

A hierarchical architecture comprises multiple layers of components connected and reflects the control and communication relations between them. The hierarchy illustrates the relative significance of the components, where the higher ones in the structure typically have more central roles [Graziani, R. & Vachon, B. (2014)].

A flat network model is characterized by a single layer and a large number of components connected at the same level to a single parent node. This structure can affect scalability as the parent node must manage and communicate with each child node. A hierarchical network model consists of multiple layers with a relatively small number of nodes in each layer. Hierarchical models are widely adopted for designing reliable, scalable, and cost-effective networks because they can break down the complex network design problem into small and manageable areas [Vázquez, A., Pastor-Satorras, R. & Vespignani, A. (2002)]. In addition, various functions are separated into layers to facilitate network management in hierarchical models. Compared to flat networks, it is easier to modify portions of the network, add new services, or increase capacity without large-scale upgrades.

### **1.2.2.2 Hierarchical Fog Computing Architecture for Resource management**

With emerging wireless technologies, various end-devices generate different types of traffic with different requirements in terms of latency, compute, bandwidth, security, etc. Therefore, fog nodes are expected to support multiple data types. Because fog systems are spatially dis-

tributed in nature, their performance is highly dependent on the communication and workload distribution in time and space. It may not be possible to satisfy all kinds of requirements in a fog node, and the same is true for the cloud.

To achieve efficient resource management in fog computing networks, a hierarchical fog architecture is more suitable than a flat one. The basic idea is to serve workloads at the optimal fog layer with respect to service properties and requirements, and the resource status of each fog layer. Because fog nodes are closer to the users generating the data, fog computing can be superior to cloud computing in terms of response time. As a result, we can serve more workloads with the same amount of computing capacity provisioned at the edge and in the cloud while satisfying the required service performance. In Fig. 1.1, the depicted system consists of four main layers: a sensing layer, single fog layer, multiple fog layer, and cloud layer, for which various objectives, such as energy, accuracy, or bandwidth should be met. Depending on the network conditions, the number of layers or the size of each layer can change dynamically.

### **1.2.2.3 Recent Approaches to Hierarchical Fog Computing**

A four-layer hierarchical fog computing platform for smart cities was presented in [Tang, B., Chen, Z., Hefferman, G., Wei, T., He, H. & Yang, Q. (2015)]. At the edge of the network, layer 4 is a sensing network containing numerous sensor nodes. The large streams of sensing data generated by these sensors are geospatially distributed to monitor state changes over time. The next layer, layer 3, comprises many low-power and high-performance computing nodes or edge devices. Its function is to send simple and quick feedback control to local components, while reporting the data processing results to the next layer. Layer 2 consists of multiple intermediate computing nodes, each connected to a group of edge devices at layer 3. Especially when hazardous events are detected, smart cities must quickly respond to take control of the infrastructure. Data analysis results are also reported to higher layers to perform large-scale and long-term behavioral analysis and monitoring. The top layer is the cloud data center. Complex and city-wide behavioral analyses can be done at this layer. They further enhance the smartness of the infrastructure by employing advanced machine learning algorithms across

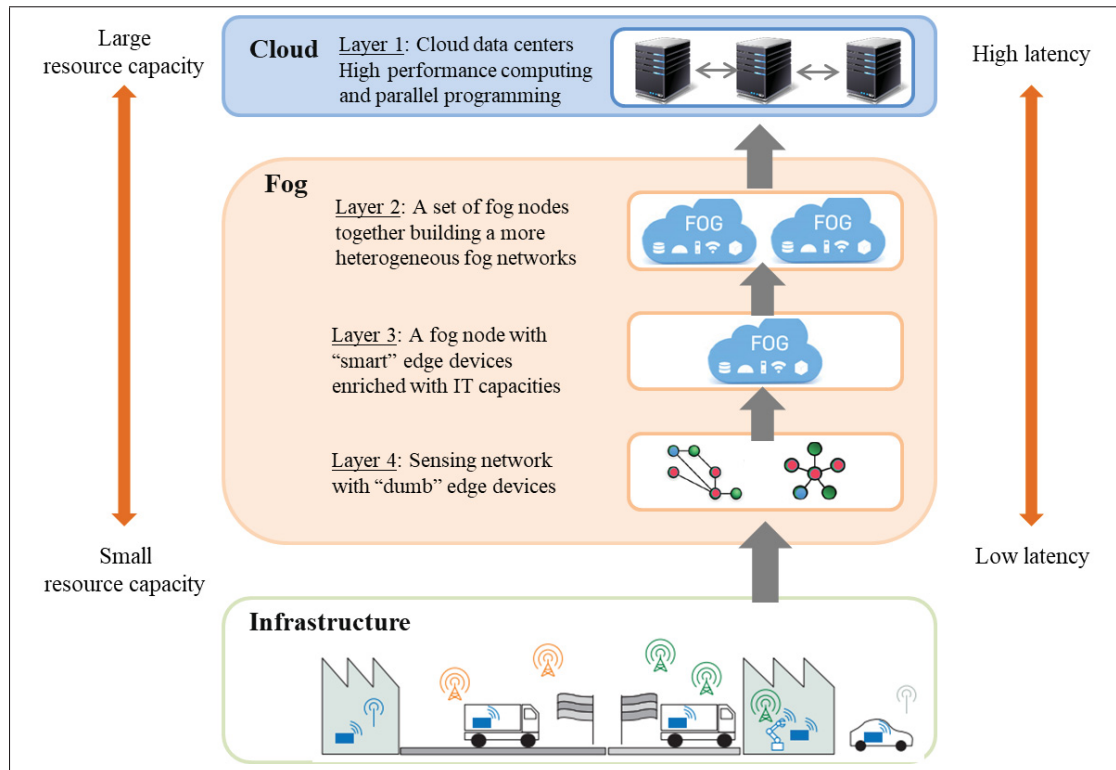


Figure 1.1 Example of hierarchical fog architecture

all system layers. The observed performance indicates that the multi-layer fog computing architecture has significant potential for future smart city monitoring and control applications.

In [Roca, D., Milito, R., Nemirovsky, M. & Valero, M. (2018)], a generic fog-based infrastructure in which fog nodes are interconnected in a hierarchy to provide services to multiple IoT applications is described. In contrast to the structure in [Tang *et al.* (2015)] that partitions the system layers based on proximity to end devices, they also consider the size of the resource pool. Due to the wide geographic deployment and location of fog nodes, it is possible to provide resources in real-time by processing data close to where it is generated. In most cases, higher layers have larger resource pools at the expense of increased latency.

In [Zhang, W., Zhang, Z. & Chao, H.-C. (2017b)], a flexible hierarchical resource management methodology is proposed. The authors present a regional cooperative fog computing architecture to support large-scale IoV applications. The coordinator can conduct resource manage-

ment to balance the load by allocating the traffic from a congested local fog server (LFS) to a nearby LFS. However, each LFS has its own priorities that determine the working state of its own VMs that are not dominated by the coordinator server. Therefore, they model two layers of resource management, i.e., intra-fog resource management which controls the tasks assigned to each LFS and adjusts the throughput of each LFS and inter-fog resource management in the coordinator. The LFSs report their status to the coordinator, who can cooperate with the intra-fog resource management layer to optimize the latency and dropping rate of the LFSs.

The authors in [Chekired, D. A., Khoukhi, L. & Mouftah, H. T. (2018)] designed a decentralized multi-tier fog architecture for industrial IoT (IIoT) request scheduling and data analysis. The basic idea is to optimally aggregate the IoT peak load and offload the load that exceeds the capacity of fog servers in the lower tier to nodes in the higher tier. They used a probabilistic model to compare the efficiency of the fog resource utilization between flat and hierarchical architectures. The results showed that multi-tier fog architectures could handle large amounts of IIoT device requests and data from various factory components. In addition, the hierarchical fog architecture is more efficient in terms of minimizing communication and computation latency.

In [Jia, S., Ai, Y., Zhao, Z., Peng, M. & Hu, C. (2016)], a hierarchical content caching method for fog-RANs (F-RANs) was studied. In the proposed model, radio APs in F-RANs are divided into remote radio heads (RRHs) relying on a centralized cloud, without edge cache, and fog-APs (F-APs) with edge caches. Both RRHs and F-APs can provide content delivery for users. The aim is to alleviate capacity constraints on the fronthaul and decrease the transmit delay by optimally designing the content access protocol. In order to fully explore the potential of hierarchical content caching, three content transfer cases using different radio access links were used to optimize the transmission latency of users requesting different contents. As a result, the capacity of both the fronthaul and radio access links were improved, effectively reducing transmission latency.



### **1.3 Intelligent Resource Management in Fog Computing Networks**

As wireless services and applications become ubiquitous and sophisticated, there is a growing need to efficiently manage the execution of increasingly complex tasks based on the requirements of the application. Specifically, how to provide pervasive network resources close to end-users and how to design resource management algorithms that can derive globally emerging system characteristics, such as agility, reliability, efficiency, and quality guarantee, are central questions that are being studied in the literature. This section presents some critical considerations of distributed computing algorithms and recent works on resource management in edge computing, particularly fog computing networks.

#### **1.3.1 Distributed Computing Applications**

##### **1.3.1.1 Resource Sharing**

The first aspect investigated concerning distributed computing in fog systems is the sharing of networking and computing resources and cooperation among the nodes. In fog networks, the number of servers as well as the number of types of computing servers, including IoT devices with computing capabilities, can be enormous. In particular, the differences between fog and cloud nodes should be accounted for in terms of storage and computational capabilities and communication bandwidth limitations. Discovering physical resources, such as CPU, memory, and network interfaces, are one of the biggest challenges, and thus accurate resource provisioning is required for reliable operation [Tomovic *et al.* (2017)].

On the other hand, virtualization involves abstracting and sharing resources among different service providers. In particular, wireless network virtualization is a technology that abstracts the physical wireless network infrastructure and physical radio resources, partitions them into virtual wireless networks with specific corresponding functions, and isolates each function so that multiple service providers can share them [Liang, C. & Yu, F. R. (2014); Cao, H., Aujla, G. S., Garg, S., Kaddoum, G. & Yang, L. (2021b)]. In other words, the resources of physical

machines are partitioned into virtual machines that host application computations and data while isolating the applications from applications in other virtual machines [Lu, P., Barbalace, A., Palmieri, R. & Ravindran, B. (2013)].

Compared to traditional servers, fog nodes are primarily designed based on the characteristics and capabilities of the end devices. A fog node can be formed by one or more physical devices with high processing capabilities [Tordera *et al.* (2016)]. For a better understanding, fog nodes are logical concepts encompassing heterogeneous types of devices as physical infrastructure. In this way, fog nodes contain the end devices together, while the processing capacity of end devices should be presented in relation to the virtual computing units. Hence, all physical devices in a fog node are aggregated as one single logical entity that can seamlessly run distributed services as if these were on a single device.

The size of a fog node composed of physical devices should be carefully defined to efficiently use limited physical resources to provide the maximum amount of end-device demand. Specifically, a large fog can create long queuing and communication delays, while a fog node that is too small may lack sufficient physical resources for computing and storage.

#### **1.3.1.2 Task Scheduling and Resource Allocation**

Because fog systems add computational power to the network's edge, the primary challenge is figuring out how to manage the actual task execution [Mouradian *et al.* (2017)]. Specifically, how do we decide which tasks to run on the end-device layer, fog layer, and cloud layer, or which nodes should be assigned to a particular task? For example, depending on the available resources, tasks requested on a node may be executed all at once, some may be executed and others postponed, or they may be sent to the remote cloud layer to be executed.

In this context, there are two different resource management techniques, which are resource allocation and task scheduling. In fog computing networks, the resources of the fog nodes can be divided into multiple applications, slices, or end devices. Resource allocation is about allocating and utilizing available resources across various applications. There are two resource

allocation methods. The first is a static allocation, in which allocation rules are programmed into the system based on preliminary resource demand information. Since it considers obsolete information, it often leads to under-utilize or over-utilize of resources depending on the time the applications are run. To tackle this issue, dynamic allocations adaptively adjust the resource allocation decisions based on real-time workloads. In this context, resource allocation strategies should be devised to account for the application or device type differences in fog networks to avoid resource wastage.

Task scheduling, on the other hand, is the process of deciding which of the tasks should be planned over a period of time and in what order. While resource allocation is concerned with allocating all available resources to various applications, slices, or end-devices, task scheduling is engaged with completing tasks in a timely manner with restricted resources. In priority-based scheduling, tasks are sorted within the buffer by the deadline, so that high-priority tasks are scheduled before low-priority ones [Verma, M. & Yadav, N. B. A. K. (2015)]. In fog computing networks, end devices can request the nearest fog node to run their tasks. As more tasks are sent to fog nodes than to the cloud server, one expected effect is that the system's power consumption increases while the system delay reduces. This is happening because the servers in the cloud layer are more powerful and energy-efficient than fog nodes while imposing additional communication delays. Therefore, the resource limitations and delay constraints on the user side should be addressed in task scheduling.

Moreover, intelligent caching resource allocation strategies and cooperative caching policies among edge devices as well as the cloud are essential to achieve significant resource and cost/energy expenditures savings [Peng *et al.* (2016); Lin, H., Garg, S., Hu, J., Kaddoum, G., Peng, M. & Hossain, M. S. (2021)]. To this end, network nodes need to learn and build users' demand profiles to predict future requests. One of the possible trends is to identify the social relationship between edge devices. By leveraging users' social relationships, future networks can learn correlation patterns from connected social and geographic data networks to better predict and infer users' behavior [Bastug, E., Bennis, M. & Debbah, M. (2014)]. This

allows cloud and edge devices to allocate storage resources properly and consequently reduce unnecessary traffic by caching popular data.

### 1.3.1.3 Offloading and Load Balancing

While distributing computational tasks over computing nodes across multiple layers of the system is allowed by the resource sharing and task scheduling discussed in the previous section, the possible workload imbalance between nodes must also be considered to improve the overall system performance. Specifically, load balancing is one of the criteria used when deciding whether to offload a particular task [Verma & Yadav (2015); Mouradian *et al.* (2017)]. When one server reaches its full capacity to run tasks, additional tasks will need to be distributed among other servers within the range of the service provider. For example, a cloud data center manages tasks by appropriately distributing them among different servers in the data center. Similarly, fog nodes with multiple virtual machines can distribute tasks to balance the load of the incoming requests. As mobile devices, usually resource-constrained, serve as fog nodes providing fog applications, the performance and QoS of computation-intensive applications will be significantly affected by the device's limited computational capabilities [Wang, X., Hu, J., Lin, H., Garg, S., Kaddoum, G., Piran, M. J. & Hossain, M. S. (2022)]. The tension between computation-intensive applications and resource-constrained mobile devices will create a bottleneck that prevents satisfactory QoS. Thus, by offloading computation tasks to the resource-rich servers, computational QoS and the efficiency of mobile devices to run diverse resource-demanding applications can be greatly improved.

On the other hand, there are several other criteria that determine whether offloading is necessary. Data management is one example. Data could be moved from one device to another based on the popularity of the data. Data that is rarely used may no longer be needed by the end device, but at the same time, it should be stored somewhere as it may be needed in the future [Aazam, M., Zeadally, S. & Harras, K. A. (2018)]. In these cases, data can be offloaded from the mobile devices to another storage-rich location, such as the cloud. This mechanism is used in various data management problems, such as caching and service placement [Zhang,

Z., Ma, L., Leung, K. K., Tassiulas, L. & Tucker, J. (2018c); Zhu, J., Chan, D. S., Prabhu, M. S., Natarajan, P., Hu, H. & Bonomi, F. (2013); Kaur, K., Garg, S., Kaddoum, G., Gagnon, F. & Jayakody, D. N. K. (2019)].

Furthermore, privacy and security are other examples that can determine the validity of offloading. Depending on the sensitivity and confidentiality of the data or tasks, offloading may occur for privacy and security reasons. For example, the fog can act as the first node for access control and traffic encryption, provide context integrity and isolation, and serve as an aggregation and control point for privacy-sensitive data before leaving the edge [Chiang & Zhang (2016)]. Hence, offloading can carry out selected security functions for resource-constrained devices that may not have as many resources as the cloud on their own to meet certain security requirements.

Load balancing is required to distribute large amounts of data across servers. Equally distributing workloads across the network can improve resource utilization, user satisfaction, and overall system performance [Neghabi, A. A., Navimipour, N. J., Hosseinzadeh, M. & Rezaee, A. (2018)]. Load balancing in fog computing networks applies to the physical nodes as well as the VMs. In a fog architecture, fog nodes continuously receive task processing requests from IoT devices and distribute the tasks among all processing nodes. Loads can be divided into different types, such as CPU, storage, memory, and network. Load balancing is the process of detecting over-loaded and under-loaded nodes and balancing the workload across all nodes. The objective of load balancing is to enhance the execution speed of applications on resources, where their execution times are run-time dependent and thus unpredictable.

In general, load balancing algorithms are divided into two types, i.e., static and dynamic [Aslam, S. & Shah, M. A. (2015)]. In static load balancing, distribution rules are programmed into the load balancer using preliminary information about the system. Given the dynamic nature of server resources and unpredictable user behavior, static load balancing methods are not very efficient in fog computing networks. On the other hand, dynamic load balancing algorithms take real-time load information into account. The main idea behind dynamic load

balancing is to find servers with a light load and distribute the load in real-time from the over-loaded servers to under-loaded ones. This allows the workload to be shared between the servers at run-time. Dynamic load balancing methods yield better overall performance but are more challenging.

### **1.3.2 Learning Approaches for Intelligent Systems**

Intelligent resource management methods are critical for the development and evolution of future networks. To cope with the unprecedented complexity of future networks, the integration of machine learning (ML) into wireless communication networks continues to increase [Zhang, C., Patras, P. & Haddadi, H. (2019a); Wang, J., Jiang, C., Zhang, H., Ren, Y., Chen, K.-C. & Hanzo, L. (2020)]. This section provides the background necessary to understand the learning algorithms used in the thesis.

#### **1.3.2.1 Reinforcement Learning**

With advances in computer technology, we currently have the ability to store and process large amounts of data. The stored data becomes useful only when it is analyzed and turned into proper information, for example, to make control decisions. ML is a technique that automates the statistical analysis of the data we observe [Goodfellow, I., Bengio, Y. & Courville, A. (2016)].

RL is an area of ML that can perceive and interpret the environment through trial-and-error. One of the key features of RL is that it explicitly considers the whole problem of a goal-directed agent interacting with an uncertain environment. RL refers to a general set of training methods in which an agent can learn how to make good decisions and avoid undesired ones in its environment through a sequence of interactions. In this context, the problem of RL can be formulated using ideas from dynamical systems theory, specifically as the optimal control of the incompletely known Markov decision process (MDP). While classical dynamic programming (DP) algorithms require information of a complete MDP model [Bellman, R. & Karush,

R. (1964)], RL can solve MDPs with unknown reward and transition functions by making observations from experience [Sutton, R. & Barto, A. (2018)]. This is a very attractive solution because it is impossible to accurately predict in advance the transition probability distributions and rewards in wireless communication systems that dynamically vary in time [Luong, N. C., Hoang, D. T., Gong, S., Niyato, D., Wang, P., Liang, Y.-C. & Kim, D. I. (2019)].

In RL, a learning agent must be able to sense the state of the environment and take actions that affect the next state. The system controller can be thought of as an agent that provides appropriate decisions about an application's underlying network. The agent interactively learns whether this action is good or not through the reward, and the reward function can be set according to the requested system performance or QoS management function.

Most RL algorithms involve estimating a value function, a function of states that estimates how good it is to perform a given action in a given state. Here, the notion of "how good" is defined in terms of expected future rewards. Value functions are determined with respect to the agent's behavior at a given time, called a policy. A policy is a mapping from states to probabilities of selecting each possible action. If the agent follows policy  $\pi$ , then  $\pi(a|s)$  is the probability of taking action  $A_t = a$  in state  $S_t = s$  at time  $t$ . RL methods specify how the agent's policy changes as a result of experience.

The value of a state  $s$  under a policy  $\pi$ , denoted  $V_\pi(s)$ , is the expected return when starting in  $s$  and following  $\pi$  thereafter. For MDPs, we can formally define  $V_\pi$  as [Sutton & Barto (2018)]

$$\begin{aligned} V_\pi(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right]. \end{aligned} \tag{1.1}$$

Among the various RL techniques, Q-learning is a classic model-free algorithm. Q-learning is typically used to find the optimal state-action policy for any MDP without an underlying policy. Given a controlled system, the learning controller interactively observes the current state  $s$ , takes action  $a$ , and then a transition occurs, and it observes the new state  $s'$  and the

reward  $r$ . From these observations, the Q-function is updated for state  $s$  and action  $a$  as follows [Sutton & Barto (2018)]:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[ R(s, a) + \gamma \max_{a' \in A_{s'}} Q(s', a') - Q(s, a) \right], \quad (1.2)$$

where,  $\alpha$  is the *learning rate* ( $0 < \alpha < 1$ ), balancing the weight of what has already been learned with the weight of the new observation.

### 1.3.2.2 Neural Network and Deep Reinforcement Learning

Although RL algorithms have been largely employed for control systems' optimization, they are not adequate for solving large-scale network problems due to scalability issues. In fact, because the size of the state space grows exponentially as the number of state elements increases, the RL method suffers from computational complexity. Moreover, theoretically, convergence is guaranteed when the mathematical conditions are met [Bertsekas, D. & Tsitsiklis, J. (1996)]. However, the distance between the initial state value and the one at the time of convergence greatly affects the learning speed. In addition, model-free RL algorithms like Q-learning learn the state-action value function very progressively, updating only the state visited at each time interval. Thus, it may take a long time to converge in all states [Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Hiedmiller, M. A., Fiedjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. (2015)]. After all, RL applications are very limited in practice.

To address this challenge, NNs come to the rescue. NNs are a powerful technique for processing high-dimensional data and extracting discriminative information from the data [Deng, L., Hinton, G. & Kingsbury, B. (2013); LeCun, Y. & Bengio, Y. (1998)]. NNs, which are usually composed of multiple layers with neurons, can automatically extract features from the data. A NN consisting of three or more hidden layers is known as a deep neural network (DNN). Representations learned from the data as input are fed into the network and since the output



of one layer is passed on to the next layer, any change can have a cascading effect on other neurons in the network [Goodfellow *et al.* (2016)].

Starting around 2013, researchers showed an increasing interest in using DNNs to learn the value, policy, and Q-functions of existing RL algorithms, and DRL was introduced [Li, Y. (2017); Mnih *et al.* (2015)]. DRL uses a non-linear function approximator like NNs to estimate the state-action value function, which is usually a linear function approximator in RL. DRL takes advantage of NNs or DNNs to improve the learning process and enable the implementation of RL on large-scale MDP problems.

#### - Deep Q-network

The Q-network can be considered as a neural network approximator with an approximate action-value function  $Q(s, a; \theta)$  with weights  $\theta$  [Mnih *et al.* (2015)]. This is called deep Q-network (DQN). DQN can be trained by iteratively adjusting the weights  $\theta$  to minimize a sequence of the loss functions,  $L_i(\theta_i)$ , where the loss function at time-step  $t$  is defined as

$$L_t(\theta_t) = \mathbb{E}[(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_{t-1}) - Q(s_t, a_t; \theta_t))^2]. \quad (1.3)$$

In other words, given a transition  $\langle s_t, a_t, r_t, s_{t+1} \rangle$ , the weights  $\theta$  of the Q-network are updated in a way that minimizes the squared error loss between the current predicted Q-value of  $Q(s_t, a_t)$  and the target Q-value of  $(r_t + \gamma \max_{a'} Q(s_{t+1}, a'))$ .

Moreover, in the DQN algorithm, the experience replay technique is adopted as the training method to address the instability of the Q-network due to the use of a non-linear approximation function [Mnih *et al.* (2015)]. This helps avoid forgetting the previous experiences, which prevents the network from only learning about what is immediately done, but also reduces correlations between experiences, which helps avoid being fixated on one region of the state space by preventing the agent from taking the same action over and over.

Here, the transition experiences,  $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$  are stored into a replay buffer  $\omega = \{e_{t-B}, \dots, e_t\}$ , where  $B$  is the replay buffer capacity. At each time step, a random mini-batch of transitions from the replay memory is chosen to train the Q-network, instead of

the most recent transition  $e_t$ . Fig. 1.2 summarizes the DQN-based general-purpose fog computing system.

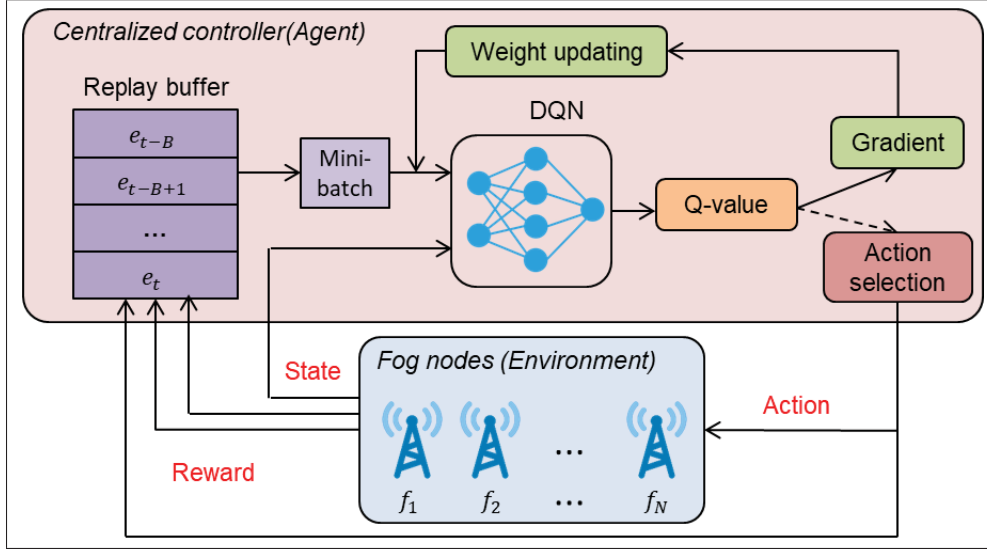


Figure 1.2 Deep Q-Network based fog computing system

### - Policy gradient

The policy gradient is another popular RL method. In policy-based methods, instead of learning a value function used in Q-learning, which gives us the expected return given a state and an action, we learn directly a policy function that maps states to actions [Sutton, R. S., McAllester, D., Singh, S. & Mansour, Y. (2000)]. The main idea is to directly adjust the parameters  $\theta$  of the policy in order to maximize the total expected return  $J(\theta) = \mathbb{E}_{s \sim p^\pi, a \sim \pi_\theta} [R]$  by taking steps in the direction of policy gradient ascent,  $\nabla_\theta J(\theta)$ . Using the Q-function defined previously, the gradient of the policy can be written as [Sutton *et al.* (2000)]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim p^\pi, a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)], \quad (1.4)$$

where  $p^\pi$  is the state distribution.

There are several advantages to using policy gradient methods. First, policy gradient methods have better convergence properties. The problem with value-based methods is that they

can have large oscillations in selected actions while training. This is because value-based methods directly estimate the Q-values and any small change in the estimated action value can dramatically change the choice of action. On the other hand, with the policy gradient, the agent simply follows the gradient to find the best policy instead of estimating the Q-values. Because the gradient tells the agent in which direction it should update its policy parameters without drastic changes in policy, the policy gradient method guarantees convergence to either a local (worst case) or global (best case) maximum. Furthermore, policy gradients are more effective in the context of high-dimensional action spaces or continuous actions [Sutton & Barto (2018)]. Last but not least, policy gradient methods can learn a stochastic policy. A stochastic policy allows an agent to explore the state space without always taking the same action because it outputs a probability distribution over the actions. As a consequence, it handles the exploration and exploitation trade-off without hard coding.

### 1.3.2.3 Recurrent Neural Network

In feedforward neural networks, the data is simply passed forward from input to output, and thus the networks cannot understand the sequential relationship of the current input state with the previous input states. Recurrent neural networks (RNNs) solve this issue with a hidden state that plays a role of memory to remember information about what has been learned. More specifically, the hidden states have the same parameters and hence can be rolled in together in a single recurrent layer, as illustrated in Fig. 1.3. The right-hand side of Fig. 1.3 is the unrolled RNN that represents the individual layers of the neural network at each time step. Therefore, RNNs take two inputs, the input from the sample sequence and previously hidden state values. This loop allows information to be passed from one step of the network to the next.

RNNs share the same weight parameter within each layer of the network. That said, these weights are updated through the processes of back-propagation over sequential data. However, vanilla RNNs run into a major problem, known as vanishing gradients. This issue is defined by the size of the gradient, where it continues to become smaller and eventually stops learning, and thus neural networks may forget important information from the beginning. To tackle

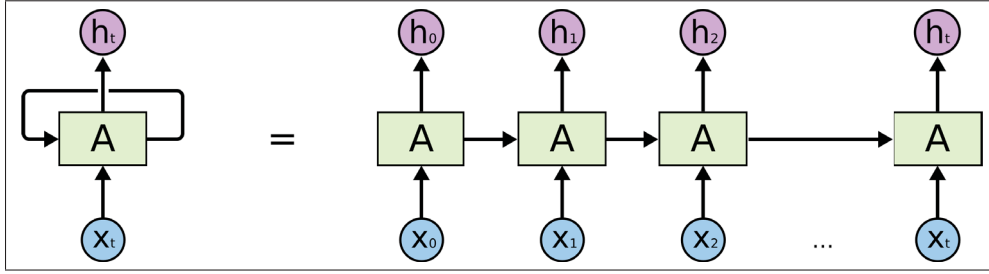


Figure 1.3 Recurrent neural network

this issue, there are variant RNN architectures, such as Long short-term memory (LSTM) and Gated recurrent units (GRU). These architectures address the vanishing gradient problem of RNN by having "gates" that control how much and which information to retain in the network.

#### 1.3.2.4 Multi-Agent Learning

When extending the decision-making process of RL to multi-agent systems (MASs), the problem can be formulated as a Markov game [Busoniu, L., Babuska, R. & Schutter, B. D. (2010); Hernandez-Leal, P., Kartal, B. & Taylor, M. E. (2019)]. The Markov game was introduced by [Littman, M. L. (1994b)] to generalize the MDP to scenarios where multiple agents interact simultaneously with a shared environment and possibly with each other.

The Markov game is formalized by the tuple  $\langle N, \mathcal{S}, \mathbf{A}, P, R, \mathbf{O}, \gamma \rangle$ , where  $N$  is the number of agents,  $\mathcal{S}$  is the state space, and  $\mathbf{A} = \{A_1, \dots, A_N\}$  denotes the actions of all agents. The transition probability  $P : \mathcal{S} \times \mathbf{A} \rightarrow P(\mathcal{S})$  describes the probability of a state transition.  $R_i$  is the reward function of agent  $i$ , and  $\mathbf{O} = \{O_1, \dots, O_N\}$  is the observations of all agents. Finally,  $\gamma \in [0, 1)$  represents the discount factor. In a cooperative problem with  $N$  agents in a fully observable environment, each agent  $i$  at time  $t$  observes the global state  $s_i^t$ , takes action  $a_i^t$  using the local stochastic policy  $\pi_i^t$ , and receives a joint reward value  $R_i^t$ , i.e.,  $R_i^t = R_1^t = \dots = R_N^t$ . If the agents cannot fully observe the state of the system, each agent only has access to its own local observations  $O_i^t$ . Furthermore, the agents' reward function may not be the same, i.e.,  $R_i^t \neq R_1^t \neq \dots \neq R_N^t$ .

In contrast to the single-agent case in Eq.(1.1), the value function  $V_i$  does not only depend on the individual policy of agent  $i$  but also on the policies of the other agents. The value function for agent  $i$  can be defined as:

$$V_{\pi_i, \pi_{-i}}(s) = \mathbb{E}_{\mathbf{s}^t \sim P, \mathbf{a}^t \sim \pi} \left[ \sum_{k=0}^{\infty} \gamma^k R_i^{t+k} | S_t = s \right], \quad (1.5)$$

where the agents behave according to the joint policy  $\pi = \{\pi_1, \dots, \pi_N\}$ .

### 1.3.3 Recent Approaches to Resource Management in Fog Computing

#### 1.3.3.1 Related Work on Cooperative Computing between Fog and Cloud

As discussed above, both the fog (or edge) and cloud should complement each other to fulfill various performance requirements and heterogeneous resource configurations. In what follows, recent studies showing the cooperation between fog and cloud are introduced.

The research in [Masip-Bruin, X., Marín-Tordera, E., Tashakor, G., Jukan, A. & Ren, G.-J. (2016)] proposed a fog-to-cloud computing architecture that handles various cloud and fog resources in a joint framework. The need for coordinated management in the fog-to-cloud architecture was discussed. Specifically, it was mentioned that a management entity must discover the set of available fogs and their resource status to choose the ones that best meet the service requirements. However, as the fog's visibility might not be sufficient to discover the global system status, the cloud plays an important role in the resource discovery and allocation. Moreover, the overall performance observed by the end users is strongly affected by the resources available to deploy the service. In dynamic scenarios where fog nodes can be frequently joined and torn down, the management entity must ensure seamless performance for services. A solution can be to use the cloud as a backup to ensure the system is tolerant to the fog nodes' failures.

In [Ren, J., Yu, G., He, Y. & Li, G. Y. (2019)], the authors studied the collaboration between cloud and edge computing, where the tasks on mobile devices can be partially processed by the edge nodes and a cloud server. Specifically, they considered a hierarchical computing system within a multi-cell mobile cellular network where each BS has limited edge computing capacity. The main idea of this study was to address two fundamental issues: how to collaborate between the edge nodes and cloud server to achieve optimal computing performance and how to jointly allocate limited network resources to minimize the end-to-end latency of mobile devices. As a result, they found that the optimal task allocation strategy for each mobile device largely depends on the back-haul communication capacity between the edge and the cloud, and the cloud computation capacity.

The task scheduling and resource allocation in a heterogeneous cloud, including the edge and remote cloud, was studied in [Zhao, T., Zhou, S., Guo, X. & Niu, Z. (2017)]. The optimization problem was formulated to maximize the success probability, which is the probability that the delay limit of a task is satisfied. The problem takes the stochastic wireless channels and time-varying traffic loads into consideration. Experimental results showed that the edge cloud should allocate more computational resources to users with stringent delay limits in the heterogeneous cloud scenario. However, in the scenario with only the edge cloud, more computational resources are allocated to users with looser delay limits when the traffic load is high. This shows that the edge cloud can better handle delay-sensitive tasks by using heterogeneous clouds. Thus, heterogeneous clouds must work together so that users with different delay requirements can be served simultaneously.

A novel framework for online fog network formation and task distribution in a hybrid fog-cloud network was studied in [Lee, G., Saad, W. & Bennis, M. (2019)]. The proposed online optimization problem aims to minimize the maximum computational latency of all fog nodes by adequately selecting a set of fog nodes from which computations are offloaded and properly distributing the tasks among those fog nodes and the cloud. To solve this problem without any prior knowledge of the future arrival of fog nodes, they defined a target competitive ratio, which is the target ratio between the latency achieved by the proposed algorithm and the op-

timal latency that can be achieved by the offline algorithm. Simulation results indicated that the proposed framework can find a suitable competitive ratio that can reduce the latency of fog computing while properly selecting the neighboring fog nodes with high performance and properly distributing the tasks among fog nodes and the cloud server.

### **1.3.3.2 Related Work on Task Offloading and Scheduling**

Offloading and distributing tasks while ensuring the QoS requirements of users is especially critical in fog nodes with limited resource capacity. Therefore, many approaches have been proposed in the literature to improve task offloading and scheduling problems for fog networks.

In [Yousefpour, A., Ishigaki, G., Gour, R. & Jue, J. P. (2018)], the authors introduced a delay-minimizing offloading policy for fog nodes in IoT-fog-cloud application scenarios. Here, the policy considers not only the length of the queue but also different types of requests with varying processing times. In this paper, the authors considered two types of tasks, light and heavy, and applied a fairness parameter based on how the fog node selects a certain type of task from its queue. Following this, the fog node determines whether to offload the selected tasks; if the estimated waiting time of the fog node is greater than an acceptable threshold, it will offload the request to its best neighboring fog node.

In [Chen, X., Jiao, L., Li, W. & Fu, X. (2015)], the authors designed a computation offloading problem between mobile device users equipped with MEC in a multi-channel wireless environment. They took a game-theoretic approach to address this problem. Their distributed computation offloading algorithm consists of two steps: radio interference measurement and offloading decision update. This algorithm allows mobile device users to take turns in improving their offloading decisions to reach mutually satisfactory decisions. Numerical results demonstrated that the proposed algorithm achieves superior computation offloading performance compared to centralized computation offloading, local computing by all users, and cloud computing by all users. They showed that MEC enhances the system performance because it allows some

executions to be done on the mobile itself instead of in the cloud to reduce overhead, including transmission time and energy.

In [Zhang, H., Xiao, Y., Bu, S., Niyato, D., Yu, F. R. & Han, Z. (2017a)], the resource allocation problem between fog nodes, data service operators, and data service subscribers was formulated. First, service subscribers purchase the optimal number of computing resource blocks from service operators using a Stackelberg game. Each subscriber competes for the required computing resource blocks owned by the nearby fog nodes. In a many-to-many matching game between service operators and fog nodes, each operator decides which fog nodes have computing resources to sell. Another many-to-many matching framework allows the resource blocks of fog nodes to be allocated to the service subscribers.

The research in [Wang, C., Liang, C., Yu, F. R., Chen, Q. & Tang, L. (2017b)] demonstrated a heterogeneous wireless cellular network with MEC that allows multiple mobile users (MUs) to simultaneously offload their computational tasks to the MEC server over small cell networks. To fulfill different user demands, different amounts of spectrum and computation resources should be allocated to different UEs. Moreover, because the MEC server has limited caching space, different caching policies should be applied for different types of content to maximize the benefit of caching. Taking these observations into account, the authors proposed to jointly optimize the computation offloading, spectrum resource allocation, and content caching in wireless cellular networks with MEC.

In [Pan, S., Zhang, Z., Zhang, Z. & Zeng, D. (2019)], the authors proposed a MEC-based dependency-aware task offloading algorithm with the goal of minimizing the execution time for mobile services with limited battery power consumption. To account for battery power constraints and inherent task dependencies within the application, they adopted a Q-learning approach that adaptively learns to jointly optimize the offloading decision and energy consumption by interacting with the network environment. Simulation results showed that the Q-learning-based solution outperforms baseline methods, including random-offloading and brute-force approach, while requiring similar time to reach the optimal solution.



In [Yadav, R., Zhang, W., Elgendy, I. A., Dong, G., Shafiq, M., Laghari, A. A. & Prakash, S. (2021)], the authors examined a computational offloading scheme for edge-enabled sensor networks. An intelligent task offloading scheme, targeted for smart healthcare applications that require long battery life and real-time monitoring, was needed to reduce overall latency and improve the battery life of devices. Therefore, a computation offloading using reinforcement learning (CORL) scheme was introduced to minimize the latency and the energy consumption in situations with limited battery capacity and service deadline constraints.

The authors in [Cheng, M., Li, J. & Nazarian, S. (2018)] proposed a novel resource provisioning and task scheduling algorithm for large-scale cloud service providers with large numbers of servers. To address the scalability issues encountered in large-scale cloud systems, they designed a two-stage deep Q-learning-based system that learns from user request patterns and a dynamic price model to generate the best long-term decisions automatically. This two-stage structure provides the proposed scheme with high efficiency and scalability.

The study in [Min, M., Xiao, L., Chen, Y., Cheng, P., Wu, D. & Zhuang, W. (2019)] investigated the computation offloading of IoT devices equipped with energy harvesting (EH) components and connected to multiple edge devices with different communication overheads. Using the energy extracted by the EH components, the IoT devices can either execute the computational tasks locally or offload a part of the totality of the tasks to the edge devices. In this context, they proposed an RL-based computation offloading method without complete knowledge of the energy consumption and latency models. Moreover, they extended this model to a DRL-based method to handle the huge state dimension using transfer learning to improve performance.

In [Chen, X., Zhang, H., Wu, C., Mao, S., Ji, Y. & Bennis, M. (2018)], the authors considered designing computation offloading policies for a MEC system in an ultra-dense network, where one of multiple BSs can be selected for computation offloading. They formulated the optimal offloading problem as a MDP that aims to minimize the long-term cost. The offloading decision is based on the channel quality between the MU and the BSs, energy queue state, and task queue state. However, the existing reinforcement learning algorithm is infeasible due to the

explosive increase in the state space since the BSs have different data transmission qualities. To solve this problem, they resorted to a DNN-based function approximator. Their numerical experiments found that a deeper DQN worsened the average cost performance because adding more hidden layers to the DQN leads to higher training errors. In the considered MEC scenario, one hidden layer with a large number of neurons can better approximate the Q-function than the approximator with a large number of hidden layers.

The study in [Wan, X., Sheng, G., Li, Y., Xiao, L. & Du, X. (2017)] investigated a dynamic malware detection strategy in which each mobile device selects the offloading rate of the application traces to a secure server in the cloud. Mobile devices can offload their malware detection tasks to secure servers with powerful computational resources and much larger security databases. Therefore, they proposed a novel offloading strategy based on a hotbooting Q-learning approach that exploits the malware detection experience to initialize the expected long-term utility values and accelerates the performance of standard Q-learning algorithm where all Q-values start at zero. In addition, they leveraged a deep convolutional neural network (CNN) to increase the accuracy of the malware detection and reduce the detection delay compared to the Q-learning-based detection method.

### **1.3.3.3 Related Work on Load Balancing**

The rapid growth of real-time applications and the diversification of computing capacity between servers in fog environments have increased the need for an efficient load balancing strategy that distributes the load between available fog nodes or clouds. Load balancing helps improve the resource utilization, user satisfaction, and the overall performance of the system.

In recent years, many researchers have contributed to the literature on load balancing of fog computing environments with various optimization objectives. A comprehensive review of load balancing approaches in fog computing was conducted in [Kaur, M. & Aron, R. (2021)], where the approaches reviewed were compared based on various aspects. Kashani, M. H., Ahmadzadeh, A. & Mahdipour, E. (2020) provided a systematic identification and taxonomic

classification of load balancing mechanisms in fog computing and a thorough comparison to analyze the potential and limitations of the existing approaches.

Xu, X., Fu, S., Cai, Q., Tian, W., Liu, W., Dou, W., Sun, X. & Liu, A. X. (2018) proposed a new resource allocation method for load balancing in fog environments. The proposed method is based on static resource allocation with dynamic service migration to achieve load balance with diversified computing nodes in fog and cloud computing layers. Naqvi, S. A. A., Javaid, N., Butt, H., Kamal, M. B., Hamza, A. & Kashif, M. (2018) studied load balancing based on a meta-heuristic approach in the fog environment. Specifically, fog nodes are allocated to handle requests from the smart grid, where each fog node contains a different number of VMs. They used an Ant Colony Optimization (ACO) technique for VM allocation to optimize the response time, processing time, and cost.

Tseng, C. H. (2016) proposed multi-path load balancing (MLB) routing to provide reliable services, especially for data-intensive IoT applications. IoT devices with heavy data transmission often cause bottleneck problems in wireless sensor networks (WSNs), therefore a proper load balancing mechanism is required to handle busty traffic. To avoid the bottleneck problem, the authors considered a distributed architecture that provides the possibility of selecting the neighboring nodes with the minimum amount of load. In the proposed model, nodes can choose the neighboring nodes with the least load as the next-hop, efficiently avoiding the bottlenecks.

The paper [Fan, Q. & Ansari, N. (2018)] introduced a load balancing scheme in which IoT devices are assigned to suitable fog nodes and BSs to minimize both communication and computing delays. In this scheme, each BS estimates its traffic load and the computing loads of fog nodes and broadcasts this information. IoT devices can then select the suitable fog node based on real-time estimated traffic and computing loads. They showed that the average latency could be reduced by adjusting the IoT device allocation to simultaneously balance the network traffic and computation loads.

In [He, X., Ren, Z., Shi, C. & Fang, J. (2016)], the authors integrated an SDN centralized controller with a fog network to improve the latency performance of delay-sensitive services,

such as the Internet of vehicles. Moreover, a novel load balancing algorithm using modified constrained optimization particle swarm optimization was proposed. This algorithm uses a centralized load balancer to balance workload between the fog and cloud servers, efficiently reducing the task processing latency.

The authors in [Wan, J., Chen, B., Wang, S., Xia, M., Li, D. & Liu, C. (2018)] proposed an energy-aware load balancing and scheduling model using fog computing in smart factories. In this context, the remote cloud has a significant delay in recognizing the operating equipment status, making it challenging to implement real-time multi-task and multi-object scheduling. Therefore, the proposed model is deployed on fog nodes by utilizing the fog nodes capable of automatic distributed processing. According to the relationship between the energy consumption and workload of the smart factory equipment, an improved particle swarm optimization algorithm is used to solve the load balancing model mathematically.

The work in [Elsharkawey, M. A. & Refaat, H. E. (2018)] demonstrated a fog computing environment where each fog server runs its own load balancing algorithm. The fog servers in the proposed model are designed to perform both real-time and non-real-time tasks. The goal of the load balancing algorithm is to meet task requirements that depend on the task deadline, execution time, server capacity, and resource utilization. The architecture of the proposed model is composed of four modules: the classifier, the task-load monitor, the fog-cloud-balancer, and the VM-manager. As a result, the proposed model provides elastic reallocation of the available processing resources of VMs according to load fluctuations. Moreover, it achieves an adaptive allocation of real-time tasks according to their deadlines and non-real-time tasks according to their priority levels.

In [Sharma, S. & Saini, H. (2019)], the authors proposed a delay-aware scheduling and load balancing algorithm in a fog environment. To handle the huge amount of sensor data from different IoT devices, a four-tier architecture was considered. Tier-1 is the lowest tier consisting of IoT devices, and Tier-2 is the workload tier. Routers classify the workloads into high and low priority based on the proposed fuzzy logic that considers the size of the task, arrival

time, and minimum and maximum completion time. In Tier-3, the fog nodes are clustered using the K-means clustering algorithm according to the current resource usage. Moreover, an artificial neural network (ANN) based dynamic load balancing threshold policy algorithm was introduced. The algorithm takes the current load of fog nodes as input to the ANN, and the output predicts the balanced workload among fog nodes. Hence, the workload is shared among multiple fog nodes based on real-time prediction. Finally, Tier-4 consists of the cloud that executes tasks that are not processed by fog nodes in Tier-3.



## CHAPTER 2

### HETEROGENEOUS TASK OFFLOADING AND RESOURCE ALLOCATIONS VIA DEEP RECURRENT REINFORCEMENT LEARNING IN PARTIAL OBSERVABLE MULTI-FOG NETWORKS

Jungyeon Baek<sup>a</sup> and Georges Kaddoum<sup>a</sup>

<sup>a</sup> Department of Electrical Engineering, École de Technologie Supérieure,  
1100 Notre-Dame west, Montreal, Canada H3C 1K3.

Paper published in *IEEE Internet of Things Journal*, vol. 8, no. 2, pp. 1041-1056,  
January 2021.

#### 2.1 Introduction

Over the past decade, moving computing, control, and data storage into the cloud has been an important trend in order to utilize much needed abundant computing resources to handle explosive traffic demands. However, this relocation introduces network delays that bring significant challenges related to meeting the latency requirements of critical applications. To overcome the disadvantages of the cloud, fog computing, which selectively moves computation, communication, control, and decision making close to the network edge where data is being generated, became inevitable in this era [A. Zaidi, Y. Hussain, M. Hogan, and C. Kuhllins, (2019)]. One of the key benefits of fog computing stems from its highly virtualized platform that offers computing capacities allowing various applications to run anywhere. Hence, fog computing resolves problems of cloud-only solutions for applications that require a real-time response with low latency, e.g., mission-critical applications [Chiang & Zhang (2016); Peng *et al.* (2016)]. Given the substantial benefits that can be drawn from this technology, fog computing is expected to play a crucial role in IoT, 5G, and other advanced distributed and connected systems [Ku, Y.-J., Lin, D.-Y., Lee, C.-F., Hsieh, P.-J., Wei, H.-Y., Chou, C.-T. & Pang, A.-C. (2017); Mouradian *et al.* (2017); Buyya & Dastjerdi (2016); Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. & Ayyash, M. (2015)].

In fog networks, where fog nodes and cloud data centers present heterogeneous resources (e.g., computational, bandwidth, and memory), service tasks are classified according to various performance requirements and heterogeneous resource configurations. In contrast to the cloud server, the computing capacity of fog nodes is usually limited and in-homogeneous. Thus, computation-intensive tasks often exhibit poor performance when they are processed by fog nodes with extremely limited resource capacities [Lee *et al.* (2019); Xu *et al.* (2018)]. In this context, offloading and distributing tasks over the network while guaranteeing the Quality-of-Service (QoS) requirements of the users, can be particularly useful. Considering the fact that fog nodes are located relatively close to each other, offloading from an originally requested fog node to an affordable neighbor node with available resources can be an attainable solution even for delay-critical services. Moreover, it is critical for the fog nodes and cloud to be able to cope with heterogeneous tasks when deciding which service tasks should be deployed and where. Specifically, the selection of suitable nodes and proper resource assignments are critical in fog networks, where various types of applications are simultaneously running over the same network [Aazam *et al.* (2018); Mach, P. & Becvar, Z. (2017)]. Hence, both the fog and cloud should complement each other in a distributive way to fulfill service needs. To this end, the hierarchical fog architecture was introduced to support a better distribution of the computing resources with an elastic and flexible placement of resources [Zhang *et al.* (2017b)].

Recently, many approaches have been proposed in the literature to enhance task offloading and resource allocation problems for fog networks. Yousefpour *et al.* (2018) introduce a delay-minimizing offloading policy for fog nodes in IoT-fog-cloud application scenarios, where the policy considers not only the length of the queue but also different request types that vary in processing times. Following this, it determines whether or not to offload the selected tasks as the estimated waiting time of fog node is greater than an acceptance threshold; it will offload the request to its best neighbor fog node. Zhang *et al.* (2017a) formulate the resource allocation problem between fog nodes, data service operators, and data service subscribers. First, service subscribers purchase the optimal number of computing resource blocks from service operators with a Stackelberg game. Each subscriber competes for the required computing resource



blocks owned by the nearby fog nodes. With a many-to-many matching game between service operators and fog nodes, each operator determines its fog nodes that have computing resources to sell. With another many-to-many matching framework, resource blocks of fog nodes are allocated to the service subscribers.

Although some promising works have been dedicated to studying task offloading and resource allocation in fog computing and edge computing networks, it is necessary to jointly address the two issues to improve the overall performance. Wang, C., Liang, C., Yu, F. R., Chen, Q. & Tang, L. (2017a) propose to jointly address computation offloading, resource allocation, and content caching in wireless cellular networks with mobile edge computing. First, they transform the original non-convex problem into a convex problem and prove the convexity of the transformed problem. Then, they decompose the problem and apply an alternating direction method of multipliers to solve it in an distributed and practical way. Alameddine, H. A., Sharafeddine, S., Sebbah, S., Ayoubi, S. & Assi, C. (2019) address task offloading with joint resource allocation and scheduling specifically focused on delay-sensitive IoT services. They mathematically formulate the problem as a mixed-integer problem and present a decomposition approach to achieve faster run times while providing the optimal solution. For heterogeneous real-time tasks, Li, L., Guan, Q., Jin, L. & Guo, M. (2019) propose task offloading and resource allocation problems in a three-tier fog system with a parallel virtual queueing model. They apply an adaptive queueing weight resource allocation policy based on the Lyapunov function. Moreover, they propose multi-objective sorting policies in terms of both the laxity and execution times of the task to achieve a trade-off between the throughput and task completion ratio optimization.

However, the computation offloading and resource allocation designs [Yousefpour *et al.* (2018); Wang *et al.* (2017a); Alameddine *et al.* (2019); Li *et al.* (2019)] are mostly based on one-shot optimization and may not be able to achieve a long-term stable performance in dynamic situations. And since most optimization problems that arise in network resource management are non-convex and NP-hard, all these algorithms generally impose restrictions on the network to simplify non-trivial mathematical equations [Luo, Z.-Q. & Yu, W. (2006)]. Nevertheless, such

assumptions would require a revision of the objective functions, or even the system models, that lead to these problem formulations in the first place.

Furthermore, there are related works using different meta-heuristic methods [Mishra, S. K., Puthal, D., Rodrigues, J. J. P. C., Sahoo, B. & Dutkiewicz, E. (2018); Tsai, C.-W. & Rodrigues, J. J. P. C. (2014); Neil Bergmann, B., Chung, Y., Yang, X., Chen, Z., Yeh, W., He, X. & Jurdak, R. (2013); Zhang, D., Haider, F., St-Hilaire, M. & Makaya, C. (2019b)]. Tsai & Rodrigues (2014) introduce the scheduling of service requests to virtual machines (VMs) with the minimum energy consumption at the fog servers. They formulate the service allocation algorithm for the heterogeneous fog server system using three meta-heuristic methods, namely particle swarm optimization (PSO), binary PSO, and bat algorithm. Moreover, the authors in [Zhang *et al.* (2019b)] introduce a new evolutionary algorithm (EA) that is combined with a PSO and genetic algorithm (GA) for the joint design of the computation offloading and fog nodes provisioning.

However, in meta-heuristic algorithms, the memory required to maintain a population of candidate solutions becomes vast as the size of problems increases. Specifically, due to the larger search space in large-scale problems, almost every state encountered will never have been seen before, which makes it impossible to converge in limited time steps. In that respect, as the system model becomes more complex, meta-heuristic methods can no longer be applied. In this context, to make sensible decisions in such large search spaces, it is necessary to generalize from previous encounters with different states that are in some sense similar to the current one.

In order to cope with an unprecedented level of complexity as we consider many parameters to accurately model the system, embedding versatile machine intelligence into future wireless networks is drawing unparalleled research interest [Wang *et al.* (2020); Zhang *et al.* (2019a)]. A lot of recent works try to address the resource allocation problem in IoT networks by using supervised machine learning, i.e., Support Vector Machines (SVMs), Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), etc [Sun, Y., Peng, M., Zhou, Y., Huang, Y. & Mao, S. (2019d)]. Nevertheless, supervised learning is learning from a fixed

data set. Thus the algorithm does not directly interact with the environment where it operates, which is not adequate to dynamically provision the on-demand resources, especially for highly volatile IoT application demands. Moreover, in the context of resource management for IoT networks, the lack of sufficient labeled data is another factor that hinders the practicality of supervised learning-based algorithms.

On the other hand, a different machine learning technique that does not fall in the category of supervised and unsupervised learning is reinforcement learning (RL) [Sutton & Barto (2018)]. One of the key features of reinforcement learning is that it explicitly considers the problem of a goal-directed algorithm interacting with an uncertain environment. Therefore, RL-based algorithms can continually adapt as the environment changes without needing explicit system models. To tackle the curse of dimensionality of RL, deep reinforcement learning (DRL) was recently introduced [Mnih *et al.* (2015)]. DRL embraces deep neural networks to train the learning process, thereby improving the learning speed and the performance of RL-based algorithms. Therefore, a DRL can provide efficient solutions for future IoT networks [Luong *et al.* (2019)].

Chen *et al.* (2018) introduce an optimal computation offloading policy for mobile edge computing (MEC) in an ultra dense system based on a deep Q-network (DQN) without prior knowledge of the dynamic statistics. Pan *et al.* (2019) study the dependency-aware task offloading decision in MEC based on Q-learning aiming to minimize the execution time for mobile services with limited battery power consumption. Van Huynh, N., Thai Hoang, D., Nguyen, D. N. & Dutkiewicz, E. (2019) develop an optimal and fast resource slicing framework based on a semi-Markov decision process (MDP) that jointly allocates the computing, storage, and radio resources of the network provider to different slice requests. To further enhance the performance, they propose a deep Q-learning approach with a deep dueling neural network, which can improve the training process and outperform all other reinforcement learning techniques in managing network slicing. Chen, X., Zhao, Z., Wu, C., Bennis, M., Liu, H., Ji, Y. & Zhang, H. (2019) consider a software-defined radio access network where multiple service providers (SPs) compete to acquire channel access for their subscribed mobile users, thereby each mo-

mobile user proceeds to offload tasks and schedule queued packets over the assigned channel. They transform the stochastic game between non-cooperative SPs into an abstract stochastic game and propose a linear decomposition approach to simplify decision making. Also, a DRL algorithm is leveraged to tackle the huge state space. Sun, Y., Peng, M. & Mao, S. (2019a) propose a DRL based joint communication mode selection and resource management approach with the objective of minimizing the network power consumption. This approach can help the network controller learn the environment from collected data and make fast and intelligent decisions to reduce power consumption. Moreover, the tremendous growth in data traffic over next-generation networks can be substantially reduced via caching, which proactively stores reusable contents in geographically distributed memory storage [Sadeghi, A., Wang, G. & Giannakis, G. B. (2019); Sun, Y., Peng, M. & Mao, S. (2019b); He, Y., Zhang, Z., Yu, F. R., Zhao, N., Yin, H., Leung, V. C. M. & Zhang, Y. (2017); Doan, K. N., Vaezi, M., Shin, W., Poor, H. V., Shin, H. & Quek, T. Q. S. (2020)]. Sun *et al.* (2019b) study the joint cache and radio resource optimization on different timescales in fog access networks. The optimization problem is modeled as a Stackelberg game. To solve the problem, single-agent RL and multi-agent RL are utilized and rigorously analyzed. Meanwhile, Doan *et al.* (2020) exploit the power allocation problem in non-orthogonal multiple access for a system with cache-enabled users. They propose a DRL based scheme, which responds quickly upon requests from users as well as allows all users to share the full bandwidth. Also, they show that applying iterative optimization algorithms is not suitable for satisfying a short-response requirement from the base station to users.

However, these methods require full knowledge of the environment and can be not suitable in many applications due to a non-trivial amount of signaling overhead and communication latency. Therefore, how to distribute the computing resources optimally throughout the network and design algorithms based on local knowledge that can derive globally emergent system characteristics such as agility, efficiency, and reliability are the central questions that lead this paper.

This paper focuses on resource management in a fog system with the aim of guaranteeing the specific quality of service of each task as well as maximizing the resource utilization by cooperating between fog computing nodes. To address this problem, we design a joint heterogeneous task offloading and resource allocation algorithm whose goal is to maximize the processing tasks completed within their delay time limits. More precisely, we consider an independent multi-agent decision-making problem that is cooperative and partially observable. To solve this problem, we propose a deep recurrent Q-network (DRQN) based learning algorithm, namely deep Q-learning combined with a recurrent layer. The DRQN-based algorithm aims to resolve partially observable environments by maintaining internal states. In particular, to guarantee the convergence and accuracy of the neural network, the proposed DRQN-based algorithm adopts an adjusted exploration-exploitation scheduling method, which efficiently avoids the exploitation of incorrect actions as the learning progresses. To our best knowledge, this is the first work that introduces DRQN to solve the joint task offloading and resource allocation problems in fog computing networks. The key contributions of this paper are summarized as follows.

- The proposed algorithm considers two-levels of heterogeneity of service tasks in terms of QoS requirements and resource demand characteristics. In real IoT scenarios, various service tasks demanding different resource sizes can require different service performances. In order to consider these heterogeneities, we propose a fog network slicing model that manages different types of tasks separately and partitions physical resources to each slice.
- Regarding the feedback and memory overhead, we consider cooperative scenarios where the independent multi-fog nodes perceive a common reward that is associated with each joint action while estimating the value of their individual actions solely based on the rewards that they receive for their actions. Therefore, this reduces the feedback and memory overheads considerably compared to joint-action learning schemes where the fog nodes require the reward, observation, and action sets of others.
- To deal with partial observability, we apply a DRQN approach to approximate the optimal value functions. The DRQN-based algorithm can tackle partial observability issues by enabling the agent to perform the temporal integration of observations. This solution is more

robust than DQN and deep convolutional Q-network (DCQN)-based methods in ways that the neural network with a recurrent layer can learn its output depending on the temporal pattern of observations by maintaining a hidden state, and thus it can keep internal states and aggregate observations. Moreover, to guarantee the convergence and accuracy of the neural network, an adjusted exploration-exploitation method is adopted.

- Numerical experiments using Tensorflow are presented to support the model and the proposed algorithm. The proposed DRQN-based algorithm requires much less memory and computation than the conventional Q-learning and meta-heuristic algorithms which are impractical for solving the problem considered in this paper. Particularly, the proposed DRQN-based algorithm is compared to the DQN and DCQN approaches where it is demonstrated that the performance in terms of average success rate, average overflow rate, and task delay can be significantly enhanced by using the proposed DRQN-based algorithm.

The remainder of this article is organized as follows: in section 2.2, the system description is presented. The formulation of the offloading and resource allocation problem as a partially observable MDP (POMDP) is detailed in Section 2.3. In section 2.4, we propose the cooperative decision-making problem between independent multi-nodes and derive a deep reinforcement learning scheme to solve the problem formulated in Section 2.3. Simulation results are presented in Section 2.5. Finally, Section 2.6 concludes this paper and provides insight on possible future work.

## 2.2 System description

In this section, we introduce a three-layer fog network system model that supports the integration of different services while serving the best of each dissimilar service characteristics, such as CPU processing density and delay requirements, through a hierarchical model. The time horizon is divided into decision epochs of equal durations (in millisecond) and indexed by an integer  $t \in \mathbb{N}_+$ . The symbols used in this paper are listed in Table 2.1.

Table 2.1 List of Notations

Symbol	Definition
$I$	The set of fog nodes
$Z$	The set of cloud servers
$K_i$	The set of fog slices at fog node $i$
$T_k$	The packet size of the task of slice $k$
$D_k^{max}$	The maximum delay budget of the task of slice $k$
$\lambda_{i,k}$	The task of slice $k$ arrival rate for the fog node $i$
$a_{i,k}$	The boolean variable whether the task of slice $k$ arrives at fog node $i$ or not
$b_{i,k}$	The number of tasks in the buffer of slice $k$ at fog node $i$
$b_{i,k}^e$	The number of tasks are allocated resources for processing among all the tasks in the buffer of slice $k$ at fog node $i$
$L_k^c$	CPU processing density demanded for the task of slice $k$
$L_k^m$	Memory size demanded for the task of slice $k$
$U_i^c$	Total CPU resource capacity of fog node $i$
$U_i^m$	Total memory resource capacity of fog node $i$
$U_z^c$	Total CPU resource capacity of cloud server $z$
$U_z^m$	Total memory resource capacity of cloud server $z$
$\eta_i^c$	The allocation unit of CPU resource at fog node $i$
$\eta_i^m$	The allocation unit of memory resource at fog node $i$
$\eta_z^c$	The allocation unit of CPU resource at cloud server $z$
$\eta_z^m$	The allocation unit of memory resource at cloud server $z$
$BW_i$	The transmission bandwidth of fog node $i$
$P_i$	The transmission power of fog node $i$
$\beta_1, \beta_2$	The path loss constant and exponent
$r_i^c$	The available CPU resource units at fog node $i$
$r_i^m$	The available memory resource units at fog node $i$
$f_{i,k}$	The offloading decision by fog node $i$ that where the task of slice $k$ will be processed
$w_{i,k}$	The resource allocation decision by fog node $i$ that how many tasks of slice $k$ will be allocated resources for processing
$\psi_i$	The local reward observed by fog node $i$

### 2.2.1 Three-layer fog network system

To improve scalability and resource utility in fog networks, a three-layer hierarchy is the most considered architecture [Zhang *et al.* (2017b); Mouradian *et al.* (2017)]. A three-layer fog network consists of an end-device layer, a fog layer, and a cloud layer. The end-device layer

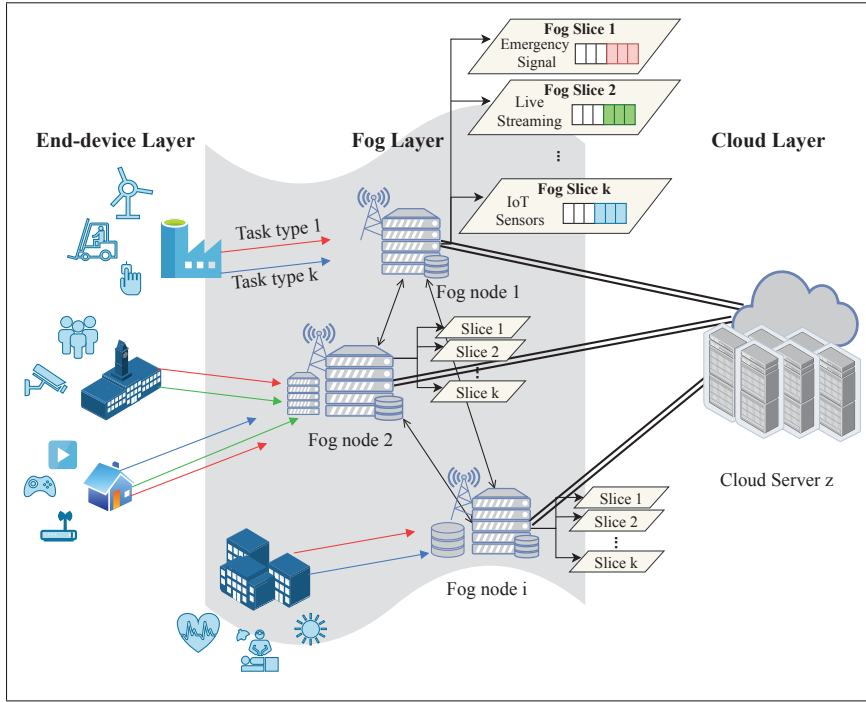


Figure 2.1 Three-layer fog network system

includes various types of devices, known as unintelligent devices, that are only producing data and are not equipped with processing capacity. Therefore, devices can request the nearest fog node to run their tasks on behalf of them. These tasks can be classified into types according to two characteristics, the task performance requirement (also called QoS) and the heterogeneous resource configuration (e.g., different CPU and memory configurations).

The fog layer is composed of multiple fog nodes  $i \in \mathcal{I}$ . As illustrated in Fig. 2.1, we consider the fog layer where the physical network infrastructure is split into multiple virtual networks to offer heterogeneous service requests for different types of end-device segments, known as network slicing [Jain & Paul (2013)]. With network slicing technology [Naeem, F., Kaddoum, G. & Tariq, M. (2021)], fog nodes can set up customized slices to guarantee specific latency and resource requirements by the supported services. Fog nodes are formed by at least one or more physical devices with high processing capabilities, which are aggregated as one single logical entity able to seamlessly execute distributed services as if it was on a single device. The shared physical resources (e.g., bandwidth, CPU, and memory) on fog nodes are partitioned into fog



slices to enable running the network functions that meet certain required slice characteristics (e.g., ultra-low-latency, ultra-reliability, etc).

Finally, the cloud layer includes various servers that are capable of sufficient storage and computational resources but are physically remote from end-devices. In our architecture, a fog network comprises of a single cloud cluster  $z \in \mathcal{Z}$  that interacts with all fog nodes.

### 2.2.2 SDN-based fog nodes and inter-SDN communications

To provide more distributed and scalable management, fog nodes make use of software-defined networking techniques where the control plane is capable of decision-making while the data plane simply serves forwarding and computing tasks [Mahmood *et al.* (2015); Tomovic *et al.* (2017)]. Furthermore, many different applications are operated concurrently in the SDN application plane. Besides, as individual SDN controllers are located in separate fog nodes, we apply the concept of inter-SDN communications, which interconnects controllers to share information and coordinate their decisions [D. Gupta and R. Jahan (2014)]. It is noted that the need for inter-SDN communications is increased as the explosive increase in task demands of end devices is requiring networks formed by more than one SDN controller [Hou, X., Muqing, W., Bo, L. & Yifeng, L. (2019)]. In our system model, each fog node defines, deploys, and adapts independent decision-making in its separate SDN controller, and communication between the SDN controllers of the fog nodes aims to exchange feedback information required by the independent decision-making processes. Details about the information exchange is provided in section 2.3.

### 2.2.3 Fog slices based on heterogeneous task models

We consider a fog network with fog nodes deploying the same set of logical fog slices for different task types over separate infrastructures (i.e.  $\mathcal{K}_i = \mathcal{K}, \forall i \in \mathcal{I}$ ). Let  $k \in \mathcal{K}_i$  be the set of slices available in the  $i^{th}$  node where each slice processes a specific type of tasks in a separate buffer. The task demanded from the end devices by slice  $k$  has a size of  $T_k$  bits. In this

context, since the time slot duration is relatively small, at most one task arrives at each slice of the fog node within a time slot. At the beginning of each time slot  $t$ , let  $a_{i,k}(t)$  be the arrived task, where  $a_{i,k}(t) = 1$  if a task of slice  $k$  arrives at fog node  $i$ , otherwise  $a_{i,k}(t) = 0$ . Hence, the probability that a new task of slice  $k$  arrives at fog node  $i$  within time slot  $t$  follows a Bernoulli distribution with parameter  $\lambda_{i,k}$ ,  $\mathbb{P}(a_{i,k} = 1) = \lambda_{i,k}$ . The number of tasks in a buffer for slice  $k$  at fog node  $i$  and time slot  $t$  is  $b_{i,k}(t)$ , of which  $b_{i,k}^e(t)$  tasks are in progress in time slot  $t$ . Meanwhile, the maximum buffer size is  $\bar{b}_{i,k}$ .

Classified tasks in each slice have specific QoS requirements as well as different resource configurations. We assume that tasks delivered from end-devices are classified to the corresponding slices regarding their characteristics without manual intervention since classifying tasks to predict the type of application [Wang, P., Lin, S.-C. & Luo, M. (2016a)] are outside the scope of this paper. In terms of QoS, we categorize tasks into three classes: 1) delay-critical class (e.g., self-driving cars, live-streaming), 2) delay-sensitive class (e.g., augmented reality/virtual reality (AR/VR), smartphone applications) and 3) delay-tolerant class (e.g., IoT sensors). The priority of tasks is determined in a way that provides maximum reliability within an acceptable delay, proper to each slice.

On the other hand, with regard to resource configurations, tasks of each slice demand two types of resources (i.e. CPU and memory). To process a task of slice  $k$ , we denote the CPU processing density (in cycles/bit) and the memory (in Mbyte) demands as  $L_k^c$  and  $L_k^m$ , respectively. Therefore, although tasks require the same QoS demands, the resource demands can be dissimilar [Kwak, J., Choi, O., Chong, S. & Mohapatra, P. (2016)]. One use case example of the delay-critical class is a live sport-streaming application requiring high-throughput, while another from the same class would be an emergency signal for self-driving cars which doesn't necessarily require high-throughput. Thus, they are processed through different slices.

Furthermore, the total resource capacities (i.e. CPU and memory) of fog node  $i$  and of the cloud server  $z$  are  $U_i = (U_i^c, U_i^m)$ ,  $\forall i \in \mathcal{I}$  and  $U_z = (U_z^c, U_z^m)$ , respectively, where the superscript  $c$  and  $m$  indicate CPU speed (in cycles/ $\Delta t$ ) and memory size (in Mbyte), respectively. We assume

that the cloud server is much more computationally powerful than its associated fog nodes, i.e.,  $U_z^c \gg U_i^c, \forall i \in \mathcal{I}$ , and provides limitless storage, i.e.,  $U_z^m \sim \infty$ . The fog nodes and the cloud server can allocate their resource on a resource unit basis. Hence, the total amount of resource units, which can be allocated by fog node  $i$  to all slices, can be computed as  $N_i = (N_i^c, N_i^m) = \left( \lfloor \frac{U_i^c}{\eta_i^c} \rfloor, \lfloor \frac{U_i^m}{\eta_i^m} \rfloor \right)$  where  $\eta_i^c$  and  $\eta_i^m$  stand for the number of allocated units of computing and memory resources at fog node  $i$ , respectively, and  $\lfloor \cdot \rfloor$  denotes the floor function. Likewise,  $\eta_z^c$  and  $\eta_z^m$  indicate the number of allocated units of computing and memory resources at cloud server  $z$  and the total number of resource units of cloud server is unlimited.

Thus, for a given node  $i$  at time  $t$ , the occupied resource units of all slices can be calculated as

$$G_i(t) = (g_i^c, g_i^m) = \left( \sum_k b_{i,k}^e(t), \sum_k b_{i,k}^e(t) \cdot \lceil \frac{L_k^m}{\eta_i^m} \rceil \right), \quad (2.1)$$

where  $\lceil \cdot \rceil$  is the ceiling function since a minimum of memory units greater than or equal to  $\frac{L_k^m}{\eta_i^m}$  must be allocated to execute the task of slice  $k$ . At every time slot  $t$ , fog nodes only monitor their own available resources which correspond to the total resources minus the sum of resources being allocated to tasks of all slices. Hence, the available resource units at fog node  $i$  and time  $t$  can be measured as

$$R_i(t) = (r_i^c, r_i^m) = (N_i^c - g_i^c, N_i^m - g_i^m), \quad (2.2)$$

where  $r_i^c + g_i^c \leq N_i^c$  and  $r_i^m + g_i^m \leq N_i^m$ . Once the task processing is completed during the time slot, the finished task will be eliminated from the buffer and the resource allocated to this task will return to the available resource pool in the next time slot.

#### 2.2.4 Calculation of task latency

In our architecture, at the beginning of each time slot  $t$ , fog nodes use an independent offloading policy to decide whether they will process arrived tasks locally or offload them to another node between neighboring fog nodes and the cloud server. Furthermore, decisions on resource

allocation are simultaneously made by fog nodes with regard to their own resources through separate allocation policies.

We define a task latency to enable different delay constraints for tasks, thereby minimizing timeout failures that result from high transmission latency from offloading to a remote node or long waiting delays due to insufficient resource capacities at the processing node. Formally, the task latency can be denoted as the sum of the transmission delay, waiting delay, and processing delay. We assume that the fog nodes have information regarding the distance to neighboring fog nodes in the fog network as well as to the cloud server. To model the transmission delay of offloading, the tasks are transmitted to the selected node over a wireless channel. Then the transmission delay for fog node  $i$  to forward the task of slice  $k$  can be defined as

$$D_{i,j,k,n}^s(t) = \begin{cases} \frac{T_k}{v_{i,j,n}(t)}, & \text{if } i \neq j \\ 0, & \text{otherwise,} \end{cases} \quad (2.3)$$

where  $j \in \{\mathcal{I}, \mathcal{L}\}$  if the selected node is a fog or cloud node, respectively, and  $n \in \{1, 2, \dots, K\}$  indicates the total number of tasks offloaded by fog node  $i$  at time slot  $t$ . Moreover,  $v_{i,j,n}(t)$  represents the transmission rate from fog node  $i$  to the selected node  $j$ , which is given by [Lee *et al.* (2019)]:

$$v_{i,j,n}(t) = BW_{i,n}(t) \cdot \log \left( 1 + \frac{\beta_1 d_{i,j}^{-\beta_2} \cdot P_i}{BW_{i,n}(t) \cdot \sigma^2} \right), \quad (2.4)$$

where  $d_{i,j}$ ,  $\beta_1$ , and  $\beta_2$  are the distance between two nodes, the path loss constant, and the path loss exponent, respectively. The variable  $P_i$  denotes the transmission power of fog node  $i$  and  $\sigma^2$  is the noise power spectral density. Additionally, the bandwidth is given by  $BW_{i,n}(t) = \frac{BW_i}{n}$ , which means that the total bandwidth of the fog node  $BW_i$  is equally shared by  $n$  tasks. For example, when a fog node  $i$  offloads a total of two different tasks during a time slot, each task is transmitted with  $\frac{BW_i}{2}$  in separate ways. When  $i = j$ , a fog node  $i$  processes this task locally, thus there is no transmission delay. Moreover, in most cases, the size of task after processing is small, thus the transmission delay of the result after processing can be ignored.

Next, when the slice  $k$  task arrives in the corresponding buffer at node  $j$ , the waiting delay  $D_{i,j,k}^q(t)$  can be calculated as

$$D_{i,j,k}^q(t) = \begin{cases} \frac{b_{j,k}(t)}{\mu_{j,k}(t)}, & \text{if } j \in I \\ 0, & \text{otherwise,} \end{cases} \quad (2.5)$$

where  $b_{j,k}(t)$  is the number of tasks previously existing in a buffer and  $\mu_{j,k}(t)$  is the service rate (i.e., the rate of tasks leaving a buffer). However, since a fog node does not have prior information about the buffer status of the other nodes when offloaded tasks arrive in its buffer and given that the service rate varies depending on the resource scheduling process, the waiting delay cannot be calculated in advance. On the other hand, we assume that the waiting time at the buffer of the cloud can be disregarded because the cloud is equipped with a larger number of cores than the fog node. This indicates that the cloud initiates the computation for the received tasks without queuing delay.

When a task is computed by fog nodes, the processing delay  $D_{i,j,k}^p(t)$  can be denoted as

$$D_{i,j,k}^p(t) = \begin{cases} \frac{T_k \cdot L_k^c}{\eta_j^c}, & \text{if } j \in I \\ \frac{T_k \cdot L_k^c}{\eta_z^c}, & \text{otherwise,} \end{cases} \quad (2.6)$$

where  $T_k \cdot L_k^c$  refers to the number of CPU cycles required to complete the execution of a task of slice  $k$ . When the task is offloaded to a fog node  $j \in \mathcal{J}$ , the task of slice  $k$  is executed by a fog node  $j$  with the CPU speed  $\eta_j^c$ . Likewise, when the task is offloaded to the cloud server  $z$ , the processing delay is formulated in the bottom equation of (2.6) where  $\eta_z^c$  is the CPU speed of cloud server  $z$ . Thus, the processing delay is dependent on both the resource configuration of the task and the amount of allocated resources.

In essence, if a slice  $k$  task is offloaded from a fog node  $i$  to a neighboring fog node  $j \neq i$ , the latency is obtained as

$$\begin{aligned} D_{i,j,k,n}(t) &= D_{i,j,k,n}^s(t) + D_{i,j,k}^q(t) + D_{i,j,k}^p(t) \\ &= \frac{T_k}{v_{i,j,n}(t)} + \frac{b_{j,k}(t)}{\mu_{j,k}(t)} + \frac{T_k \cdot L_k^c}{\eta_j^c}. \end{aligned} \quad (2.7)$$

If a slice  $k$  task is computed locally by a fog node  $i$ , the latency becomes

$$\begin{aligned} D_{i,i,k,n}(t) &= D_{i,i,k}^q(t) + D_{i,i,k}^p(t) \\ &= \frac{b_{i,k}(t)}{\mu_{i,k}(t)} + \frac{T_k \cdot L_k^c}{\eta_i^c}. \end{aligned} \quad (2.8)$$

Finally, if a slice  $k$  task is offloaded from a fog node  $i$  to the cloud server  $z$ , the latency is

$$\begin{aligned} D_{i,z,k,n}(t) &= D_{i,z,k,n}^s(t) + D_{i,z,k}^p(t) \\ &= \frac{T_k}{v_{i,z,n}(t)} + \frac{T_k \cdot L_k^c}{\eta_z^c}. \end{aligned} \quad (2.9)$$

## 2.3 Problem formulation

In this section, we define the problem of heterogeneous task offloading and resource allocation in a system with multiple fog nodes as a POMDP across the time horizon.

### 2.3.1 Partially observable MDP based problem formulation

The main goal of the system is to make an optimal offloading and resource allocation decision at each node with the objective of maximizing the successfully processed tasks while guaranteeing the corresponding delay constraint of each task. Therefore, the joint offloading and resource allocation decision is achieved by finding proper processing nodes for the tasks and an optimal allocation of the node's resources to all individual slices. We assume that the joint offloading and resource allocation decisions from all fog nodes are made simultaneously at every

time slot  $t$ . To this end, each node repeatedly observes its own system states at the beginning of the time slot. The local observation by fog node  $i$  is defined as

$$O_i(t) = \left( A_i(t), B_i(t), B_i^e(t), R_i(t) \right), \quad (2.10)$$

where  $A_i(t) = (a_{i,k}(t) : k \in \mathcal{K}_i)$ ,  $B_i(t) = (b_{i,k}(t) : k \in \mathcal{K}_i)$ , and  $B_i^e(t) = (b_{i,k}^e(t) : k \in \mathcal{K}_i)$  are the set of arrived tasks, the number of tasks in the buffer, and the number of tasks in progress among  $B_i(t)$  from all slices at the fog node  $i \in \mathcal{J}$  and time  $t$ , respectively. Moreover,  $R_i(t) = \left( r_i^c(t), r_i^m(t) \right)$  is the available resource units at the fog node  $i \in \mathcal{J}$  at time  $t$ .

Note that the underlying states of the system including the states of other fog nodes are not accessible by the fog node. Instead, only the aforementioned state set in (2.10) can be observed and thus the system becomes a POMDP. We suppose that the observations are limits to the measurement accuracy of the state but are enough to make usable state data for a POMDP system.

In the presence of uncertainties stemming from the task demands and resource availability at the fog nodes, we formulate the POMDP based problem across the time horizon as a stochastic game in which each node selects actions as a function of their local observation. In our model, a fog node's offloading and resource allocation policy operates independently from the other nodes' policies. Thus, each fog node does not have any prior information on the task demands, buffer status, and resource availability of the other fog nodes. Accordingly, the actions are defined as

$$X_i(t) = \left( X_i^f(t), X_i^w(t) \right), \quad (2.11)$$

where  $X_i^f(t) = (f_{i,k}(t) : k \in \mathcal{K}_i)$  and  $X_i^w(t) = (w_{i,k}(t) : k \in \mathcal{K}_i)$  denote the offloading decision and resource allocation decision, respectively.  $f_{i,k}(t) \in \{0, 1, \dots, I+1\}$  represents by whom the task will be processed, where  $f_{i,k}(t) = 0$  if the slice  $k$  task doesn't arrive at the fog node  $i$  at time  $t$ ,  $f_{i,k}(t) = i$  if the slice  $k$  task arrives and will be computed locally, and  $f_{i,k}(t) = j, j \in \mathcal{J}$  and  $j \neq i$ , if the slice  $k$  task arrives and will be offloaded to another node ( $f_{i,k}(t) = I+1$  implies that the

fog node will offload this task to the cloud server). The resource allocation decision  $w_{i,k}(t) \in \{0, 1, \dots, \lfloor \frac{U_i^m}{L_k^m} \rfloor\}$  represents how many tasks will be initiated by being allocated resources where  $\lfloor \frac{U_i^m}{L_k^m} \rfloor$  is the maximum number of tasks that can be simultaneously processed by the fog node  $i$ . For example,  $w_{i,k}(t) = 2$  indicates that, at time  $t$ , fog node  $i$  allocates its resources to slice  $k$  to execute two tasks that are not in progress in the buffer. Each node takes an action  $X_i(t)$  only among the ones allowed in that observation, i.e.,  $X_i(t) \in \mathcal{X}_i(O(t))$ . We apply the following constraints for the offloading and resource allocation at time  $t$ ,

$$f_{i,k}(t) = 0, \text{ if } a_{i,k}(t) = 0, \forall k \in \mathcal{K}_i, \forall i \in \mathcal{I}, \quad (2.12)$$

$$w_{i,k}(t) \leq (B_i(t) - B_i^e(t)), \forall k \in \mathcal{K}_i, \forall i \in \mathcal{I}, \quad (2.13)$$

$$\sum_{k \in \mathcal{K}_i} w_{i,k}(k) \leq r_i^c, \forall i \in \mathcal{I}, \quad (2.14)$$

$$\sum_{k \in \mathcal{K}_i} w_{i,k}(k) \cdot \lceil \frac{L_k^m}{\eta_i^m} \rceil \leq r_i^m, \forall i \in \mathcal{I} \quad (2.15)$$

to ensure that the fog node cannot offload the task when it doesn't arrive by (2.12), cannot allocate more than the number of tasks waiting for allocation by (2.13), and the sum of newly allocated resources cannot exceed the available resources by (2.14) and (2.15).

Given that each node is in state  $O_i(t)$  and action  $X_i(t)$  is chosen, a transition probability is given by (2.16), where  $\mathbf{X}(t) = (X_i(t) : i \in \mathcal{I})$  are the set of actions occurring at time  $t$ . From (2.16),  $B_i^e(t+1)$  and  $R_i(t+1)$  only depend on the action  $X_i(t)$  of fog node  $i$ , while  $A_i(t+1)$  is determined regardless of the action. Since one node's offloading decisions result in increasing others' buffers, the sequence of each node's buffer status  $B_i(t)$  depends on the actions of all agents  $\mathbf{X}(t)$ .

$$\begin{aligned} \mathbb{P}(O_i(t+1)|O_i(t), \mathbf{X}(t)) &= \mathbb{P}(A_i(t+1)) \times \mathbb{P}(B_i(t+1)|B_i(t), \mathbf{X}(t)) \\ &\quad \times \mathbb{P}(B_i^e(t+1)|B_i^e(t), X_i^w(t)) \times \mathbb{P}(R_i(t+1)|R_i(t), X_i^w(t)) \end{aligned} \quad (2.16)$$



$$\psi_i(O_i(t), \mathbf{X}(t)) = \frac{1}{K} \cdot \sum_{k \in \mathcal{K}} a_{i,k}(t) \cdot \left( (-1)^{\mathbb{1}(D_k^{max} \leq D_{i,k}(t))} - \xi_k \cdot \mathbb{1}(b_{f_{i,k},k}(t + D_{i,k}^t(t)) \geq \bar{b}_{f_{i,k},k}) \right) \quad (2.17)$$

Based on the set of actions  $\mathbf{X}(t)$  in local observation  $O_i(t)$ , we define the local reward in (2.17), where  $D_k^{max}$  is the maximum delay budget of the task in slice  $k$  where the task is discarded if its processing is not completed within this budget. The first term of the summation in (2.17) represents the success reward, a positive reward if the task is successfully completed and negative reward if timeout failure is encountered, which depends on both offloading decisions of arrived fog node  $i$  and resource allocation decision of the processing fog node. The second term describes the overflow cost which defines whether the task is dropped because the slice buffer is already full, thus it is related to the buffer status of processing fog node  $b_{f_{i,k},k}$ . Moreover,  $\xi_k$  is a constant weighting factor that balances the importance of the overflow failure for tasks of slice  $k$ .

### 2.3.2 Cooperative games by independent learners

Although each fog node's main goal is to optimize its own service performance and its resource interests, the fog nodes must still coordinate on the resource flows between neighboring nodes in order to achieve a meaningful solution from an overall system perspective [Busoniu, L., Babuska, R. & De Schutter, B. (2008)]. In addition, the service performance experienced by service tasks during the processing is determined by the offloading and the resource allocation decisions of all fog nodes. Therefore, our stochastic game, sometimes called Markov game, follows a cooperative network to maximize the common goal rather than a competitive game where each fog node has opposing goals [Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P. & Mordatch, I. (2020); Hausknecht, M. J. & Stone, P. (2015)].

More precisely, we apply cooperative scenarios between independent multi-fog nodes where the fog nodes share their local rewards with others as feedback information. This decision-making problem implies that independent fog nodes perceive the common reward that is as-

sociated with each joint-action while estimating the value of their individual actions solely based on the rewards that they receive for their actions. Therefore, this reduces the feedback and memory overheads considerably compared to joint-action learning schemes where the fog nodes share their reward, observation, and action sets with others to maintain a model of the strategy of other agents. As such, at each time step, each node executes an individual action, with the joint goal of maximizing the average rewards of all nodes which can be formally formulated as

$$\psi(t) = \sum_{i \in \mathcal{I}} \psi_i(O_i(t), \mathbf{X}(t)). \quad (2.18)$$

Thus, each node's reward is drawn from the same distribution, reflecting the value assessment of all nodes [Claus, C. & Boutilier, C. (1998)]. Moreover, the convergence performance of joint-action learning schemes is not enhanced dramatically despite the availability of more information due to the exploration strategy [Kapetanakis, S. & Kudenko, D. (2002)]. As detailed in section 2.2, the reward feedback is transmitted through inter-SDN communications to the SDN controllers of all the fog nodes for the decision-making process. In summary, the decision-making process at each fog node is fully distributed for real-time task offloading and resource management while communications between SDN controllers aims to exchange less time-sensitive reward information.

## 2.4 Learning the optimal offloading and resource allocation policies

In this section, we propose a Q-learning-based optimal policy solution to address the limitations of the traditional approaches and discuss deep recurrent Q-networks (DRQN) which can better approximate actual Q-values from sequences of observations, leading to better policies in a partially observable environment.

### 2.4.1 Optimal policy solution using Q-learning

In the case where the system has access to transition probability functions and rewards for any state-action pair, the MDP can be solved through dynamic programming (DP) approaches

to find the optimal control policy [Bellman & Karush (1964); Puterman, M. (2005)]. However, in our cases, the system cannot precisely predict the transition probability distributions and rewards. To address this limitation, reinforcement learning is proposed in which the lack of information is solved by making observations from experience [Sutton & Barto (2018)]. Among the different reinforcement learning techniques, Q-learning is used to find the optimal state-action value for any MDP without an underlying policy. Given the controlled system, the learning node  $i$  repeatedly observes the current state  $O_i^t$ , takes action  $X_i^t$  that incurs a transition, then it observes the new state  $O_i^{t+1}$  and the reward  $\psi_i^t$ . From these observations, it can update its estimation of the Q-function for state  $O_i^t$  and action  $X_i^t$  as follows:

$$Q_i(O_i^t, X_i^t) \leftarrow (1 - \alpha) \cdot Q_i(O_i^t, X_i^t) + \alpha \cdot [\psi_i^t + \gamma \max_{X' \in \mathbf{X}(O_i^{t+1})} Q_i(O_i^{t+1}, X')], \quad (2.19)$$

where  $\alpha$  is the *learning rate* ( $0 < \alpha < 1$ ), balancing the weight of what has already been learned with the weight of the new observation, and  $\gamma$  is the *discount factor* ( $0 < \gamma < 1$ ). The most common action selection rule is the  $\varepsilon$ -greedy algorithm that behaves greedily most of the time i.e., Greedy selection ( $X^t \doteq \arg \max_{X'} Q(O^t, X')$ ) and explores other options by selecting a random action with a small probability  $\varepsilon$ . This greedy selection and the  $\varepsilon$  probability of random selection are called exploitation and exploration, respectively. Non-optimal action selection can be uniform during exploration ( $\varepsilon$ -greedy algorithm) or biased by the magnitudes of Q-values (such as Boltzmann exploration) [Sutton, R. S. (1990); Perkins, T. J. & Precup, D. (2002)].

Moreover, we discuss the computational complexity of the Q-learning algorithm. The Q-algorithm requires storing a  $|\mathcal{O}| \times |\mathcal{X}|$  size table of Q-values, i.e.,  $Q(O, X)$  for all  $O \in \mathcal{O}$  and  $X \in \mathcal{X}$ . In our problem, the size of local observation spaces  $|\mathcal{O}_i|$  and local action spaces  $|\mathcal{X}_i|$  is calculated as  $\prod_{k \in \mathcal{K}} \left( 2 \times (1 + \bar{b}_{i,k})^2 \right) \times (1 + N_i^c) \times (1 + N_i^m)$  and  $(I + 2)^K \times \prod_{k \in \mathcal{K}} \left( 1 + \lfloor \frac{U_k^m}{L_k^m} \rfloor \right)$ . When  $I = 5$ ,  $K = 3$ ,  $\bar{b}_{i,k} = 5$ ,  $N_i^c = 5$ ,  $N_i^m = 5$ , and  $\lfloor \frac{U_k^m}{L_k^m} \rfloor = 5$ , one node  $i$  has to update a total of  $9.955 \times 10^8$  Q-function values, which makes it impossible for the conventional Q-learning process to converge within a limited number of time steps. This problem is even more pronounced in multi-agent scenarios, where the number of joint actions grows exponentially with the number of agents in the system.

### 2.4.2 Convergence to equilibrium

$\pi_i^*$  of a node  $i$  is the optimal policy to other nodes. Recall from fictitious play [Fudenberg, D. & Kreps, D. (1990)], the exploration strategy is required to be asymptotically myopic to ensure that Nash equilibrium will be reached in multi-agent RL strategies. An action selection rule  $\pi_i$  is said to be asymptotically myopic if the loss from agent  $i$ 's choice of actions at every given history  $\pi_i$  goes to zero as  $t$  proceeds [Fudenberg & Kreps (1990)]:

$$\psi(\pi_i^t) \nearrow \max\{\psi(X_i) | X_i \in \mathcal{X}_i(O_i)\}, \quad (2.20)$$

as  $t \rightarrow \infty$ , where  $\psi$  denotes the reward function. Therefore, the independent multi-agent Q-learning in cooperative systems will converge to equilibrium almost surely when the following conditions are satisfied [Claus & Boutilier (1998)]:

- The learning rate  $\alpha$  decreases over time such that  $\sum^t \alpha = \infty$  and  $\sum^t \alpha^2 < \infty$ .
- Each node visits every action infinitely often.
- The probability  $\mathbb{P}_i^t(x)$  that node  $i$  selects action  $x$  is nonzero,  $x \in \mathcal{X}(o)$ .
- The exploration strategy of each node is exploitative such that

$$\lim_{t \rightarrow \infty} \mathbb{P}_i^t(\pi_i^t) = 0,$$

where  $\pi_i^t$  is a random variable denoting a non-optimal action was taken based on estimated Q-values of node  $i$  at time  $t$ .

The first two conditions are required for convergence in Q-learning, while the third ensures that nodes explore with a positive probability at all times, which will ensure the second condition. Last but not least, the fourth condition guarantees that agents exploit their knowledge as the number of time steps increases. In fact, convergence of Q-learning does not depend on the exploration strategy used, which implies that there is no rule to choose actions as long as every action is visited infinitely often. However, effective exploration strategies will encourage

long run optimal equilibrium [Claus & Boutilier (1998)]. To this end, we propose an adjusted exploration-exploitation method in the next subsection.

### 2.4.3 Deep Q-learning with nonlinear transformation

To solve the scalability issues of Q-learning, we adopt Q-learning with a neural network, called deep Q-network (DQN). The DQN embraces the advantage of deep neural networks (DNNs) to train the learning process at each node  $i \in \mathcal{I}$ , thereby improving the learning speed and the performance of Q-learning algorithms.

The Q-network can be trained by iteratively adjusting the weights  $\theta$  to minimize a sequence of the loss function,  $\mathcal{L}_i(\theta^t)$ , where the loss function at time slot  $t$  is defined in (2.21).

$$\mathcal{L}_i(\theta_i^t) = \mathbb{E} \left[ \left( \psi^t + \gamma \max_{X' \in \mathcal{X}_i} Q_i(O_i^{t+1}, X'; \theta_i^{t-1}) - Q_i(O_i^t, X_i^t; \theta_i^t) \right)^2 \right] \quad (2.21)$$

Precisely, given a transition  $\langle O_i^t, X_i^t, \psi^t, O_i^{t+1} \rangle$ , the weights  $\theta_i^t$  of the Q-network of node  $i \in \mathcal{I}$  are updated in a way that minimizes the squared error loss between the current predicted Q-value of  $Q_i(O_i^t, X_i^t)$  and the target Q-value of  $[\psi^t + \gamma \max_{X' \in \mathcal{X}_i} Q_i(O_i^{t+1}, X')]$ . The gradient of the loss function with respect to the weights  $\theta_i^t$  is given by (2.22).

$$\nabla_{\theta_i^t} \mathcal{L}_i(\theta_i^t) = \mathbb{E} \left[ \left( \psi^t + \gamma \max_{X' \in \mathcal{X}_i} Q_i(O_i^{t+1}, X'; \theta_i^{t-1}) - Q_i(O_i^t, X_i^t; \theta_i^t) \right) \cdot \nabla_{\theta_i^t} Q_i(O_i^t, X_i^t; \theta_i^t) \right] \quad (2.22)$$

Moreover, in the DQN algorithm, the experience replay technique is adopted as the training method to address the instability of the Q-network due to the use of non-linear approximation functions [Mnih *et al.* (2015)]. Hence, the transition experiences,  $e_i^t = \langle O_i^t, X_i^t, \psi^t, O_i^{t+1} \rangle$  are stored into a replay buffer  $\mathcal{M}_i = \{e_i^{t-\mathcal{D}_i}, \dots, e_i^t\}$ , where  $\mathcal{D}_i$  is the replay buffer capacity. Due to possible delays in the reward feedback between fog nodes, the past transition experiences may need to wait in the temporal replay buffer to combine the rewards (as in (2.18)), where they are transferred to the replay buffer as soon as it is ready. At each time step, instead of the most recent transition  $e_i^t$ , a random mini-batch  $\mathcal{N}_i$  of transitions from the replay memory is chosen to

train the Q-network by node  $i$ . Furthermore, every  $\mathcal{C}$  time steps, the network  $Q_i$  is duplicated to obtain a target network  $\hat{Q}_i$  which is used for generating the target Q-value for the following  $\mathcal{C}$  updates to  $Q_i$ .

In addition, to guarantee the convergence and accuracy of the neural network, we adopt an adjusted exploration-exploitation scheduling method. At the beginning of the process, the agent with a normal  $\varepsilon$ -greedy algorithm selects more random actions with a probability  $\varepsilon = \varepsilon_{start}$  to encourage initial exploration. Then, the exploration rate is asymptotically decayed with  $\varepsilon_{decay}$  until it reaches a certain minimum value  $\varepsilon_{min}$  and is preserved until the last iteration. Since  $\varepsilon_{min}$  is a very small number, after this initial exploration phase, most decisions take the highest estimated value at the present iteration. However, this often leads to a sub-optimal policy due to exploiting bad estimates of the Q-value which were learned during the early iterations and insufficient exploration in large state-action space cases. To deal with this problem, the adjusted exploration-exploitation method allows the agent to shift back into exploratory mode every  $\mathcal{R}^\varepsilon$  time slots, where the starting exploration probability  $\varepsilon_{start}$  is decreased  $\delta^\varepsilon$  ( $0 < \delta^\varepsilon < 1$ ) times every update. Therefore, this method efficiently avoids the exploitation of incorrect actions by selecting better estimates of the Q-value as the learning progresses. The optimal control policy learning implementation using the DQN algorithm is illustrated in Fig. 2.2.

#### 2.4.4 Deep-recurrent Q-learning for partial observability

Another problem is that estimating a Q-value from an immediate observation in DQN can be arbitrarily wrong since the network states are partially observed and hinge upon multiple users [Omidshafiei, S., Pazis, J., Amato, C., How, J. P. & Vian, J. (2017)]. Any system that requires a memory of more than an immediate observation will appear to be non-Markovian because the future system states depend on more than just the current input. This issue can be solved by allowing the agent to perform temporal integration of observations. The solution adopted in [Mnih *et al.* (2015)] stacks the last four observations in memory and feeds them to the convolutional neural network (called DCQN) instead of a single observation at a time. However, their DCQN takes in a fixed size vector as input, a stack of 4 observations, which

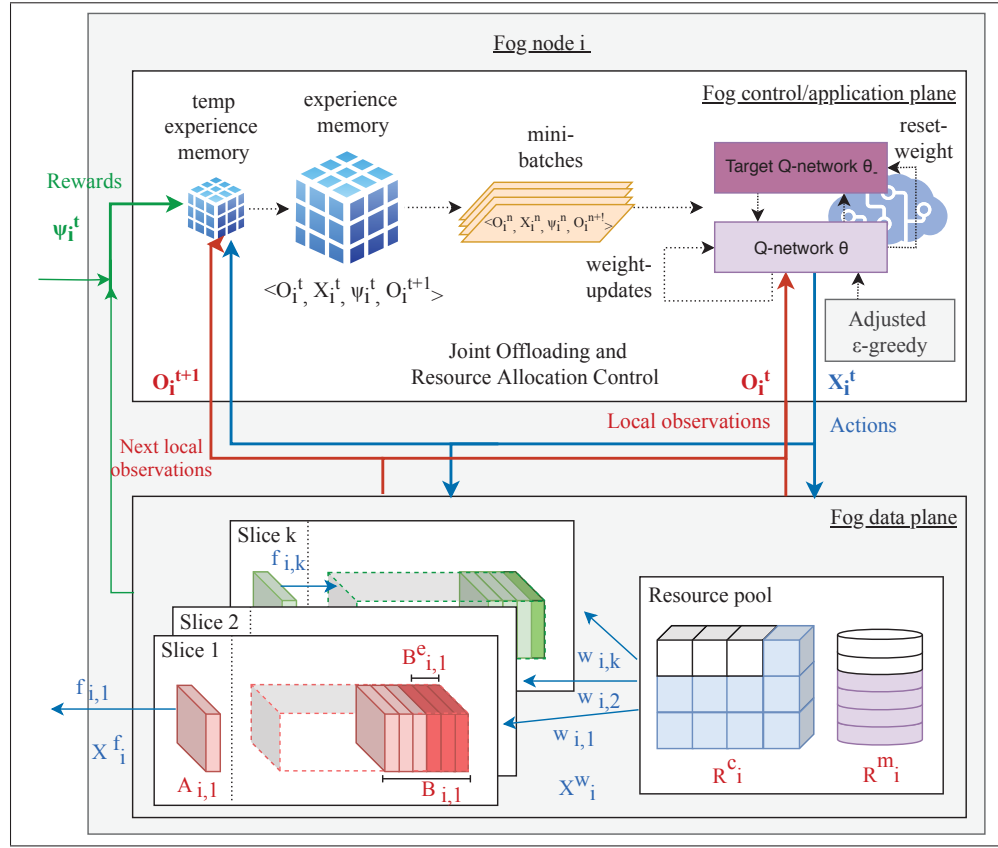


Figure 2.2 Application of a deep Q-network (DQN) to approximate the optimal joint offloading and resource allocation control policy of the SDN-based fog nodes

limits its usage in situations that involve a sequence type input with no predetermined size. In order to address this issue, we implement a DRQN which replaces the DCQN's first fully connected layer by a recurrent layer. By utilizing a recurrent layer, the neural network will be able to learn its output depending on the temporal pattern of observations by maintaining a hidden state that it computes at every iteration. The recurrent layer can feed the hidden state back into itself, and thus it can maintain internal states and aggregate observations.

However, during backpropagation, vanilla recurrent neural networks suffer from the vanishing gradient problem, which makes layers that get a small gradient value stop learning, and thus neural networks may forget important information from the beginning. To tackle this problem, we use Gated Recurrent Unit (GRU) for the recurrent layer. Similar to Long short-term memory

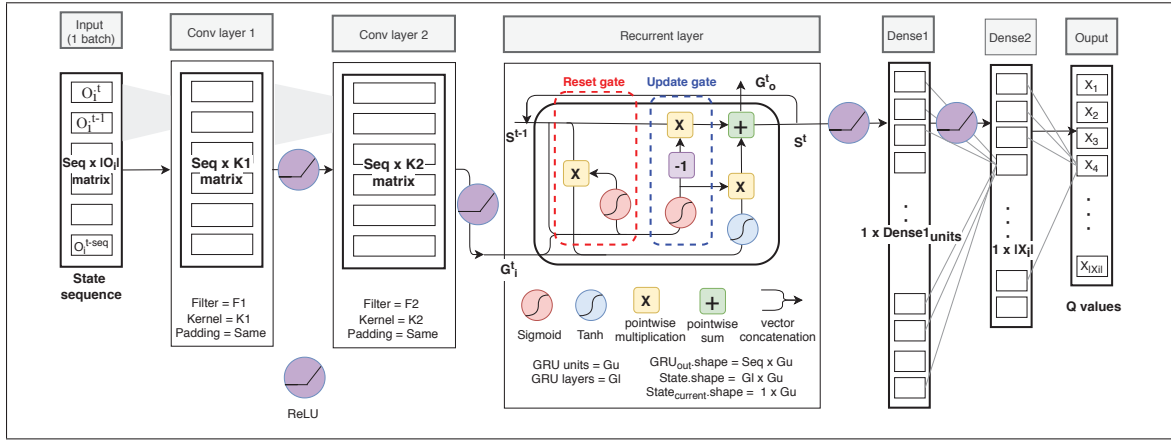


Figure 2.3 The proposed DRQN structure with GRU

(LSTM), GRU was introduced as a solution to the short-term memory of vanilla recurrent neural networks [Chung, J., Gülçehre, Ç., Cho, K. & Bengio, Y. (2014)]. The main concept of GRUs is a gating mechanism, which can learn which information is relevant to keep or forget during training in the recurrent network. GRU has two gates (reset and update) and is known to be computationally more efficient and faster than LSTM which consists of three gates (forget, input, and output) and cell state, while its performance is comparable to LSTM [Jozefowicz, R., Zaremba, W. & Sutskever, I. (2015); Le, T.-D. & Kaddoum, G. (2021)]. The proposed DRQN structure is illustrated in Fig. 2.3.

$$\begin{aligned}
 G_r^t &= \sigma(W_r^s S^{t-1} + W_r^g G_i^t + \text{bias}_r), \\
 G_z^t &= \sigma(W_z^s S^{t-1} + W_z^g G_i^t + \text{bias}_z), \\
 \tilde{S}^t &= \tanh(W^s (G_r^t \odot S^{t-1}) + W^g G_i^t + \text{bias}), \\
 S^t &= G_z^t \odot S^{t-1} + (1 - G_z^t) \odot \tilde{S}^t,
 \end{aligned} \tag{2.23}$$

where  $G_r^t$  and  $G_z^t$  are reset and update gates, respectively. With that, the recurrent network can learn how to use some of its units to selectively cancel out the irrelevant information and protect the state. Sigmoid and Tanh activation functions can make these decisions by filtering values between 0 and 1 for each state element.



Algorithm 2.1 details the procedure of the proposed learning algorithm at fog node  $i$ . The neural network takes the state sequence as an input to the first convolutional layer. Since the valid action space is dependent upon the current state value, we involve a step in the action selection that sets the probability of invalid actions to zero and re-normalizes the sum of the probabilities of the other actions to 1.

Algorithm 2.1 Deep recurrent Q-learning algorithm for approximating the optimal state-action value functions of a fog node  $i \in I$  with experience memory

```

1 Set Initialize replay buffer  $\mathcal{M}_i$  to capacity  $\mathcal{D}_i$ , state-action value function  $Q_i$  with
   random weights  $\theta_i$ , target state-action function  $\hat{Q}_i$  with weights  $\theta_{i-} = \theta_i$ .
2 while ( $t \leq \text{maximum iteration}$ ) do
3   Observe the arrival task  $A_i^t$ , buffer state  $B_i^t, B_i^{e,t}$ , resource status  $R_i^t$  and combine
   them as  $O_i^t$ 
4   Take  $\hat{O}_i^t$  as an input to the DRQN network with parameter  $\theta_i$ , where  $\hat{O}_i^t$  is a state
   sequence
5   Calculate  $\varepsilon = \max[\exp(-\varepsilon_{decay} \cdot (t \bmod \mathcal{R}^\varepsilon) + \log \varepsilon_{start}), \varepsilon_{min}]$ , and choose
   random action  $X_i^t$  from valid action spaces  $X(O_i^t)$  with probability  $\varepsilon$  otherwise
   select  $X_i^t \doteq \arg \max_{X'} Q(O_i^t, X'; \theta_i)$ 
6   Execute action  $X_i^t$ ; offload tasks according to  $X_i^{f,t}$  and allocate the resource  $X_i^{w,t}$ 
7   Observe local reward  $\psi_i^t$ , next state  $O_i^{t+1}$  and receive rewards from other nodes
    $\psi_{j \neq i}^t$ 
8   Save transition  $\langle O_i^t, X_i^t, \psi^t, O_i^{t+1} \rangle$  in  $\mathcal{M}_i$ 
9   Sample a random mini-batch from  $\mathcal{M}_i$   $\mathcal{N}_i = \langle \hat{O}_i^n, X_i^n, \psi^n, \hat{O}_i^{n+1} \rangle$ 
10  Set  $y_i^n = \psi^n + \gamma \max_{X' \in \mathcal{X}_i} Q_i(O_i^{n+1}, X'; \theta_{i-}^t)$ 
11  Perform a gradient step in (19) with respect to the parameter  $\theta_i^t$ 
12  Every  $\mathcal{C}$  time step, reset the target network parameters  $\theta_{i-}^{t+1} = \theta_i^t$ 
13  Every  $\mathcal{R}^\varepsilon$  time step, update  $\varepsilon_{start} = \delta^\varepsilon \cdot \varepsilon_{start}$  and  $\varepsilon_{decay} = -\log \frac{\varepsilon_{min}}{\mathcal{R}^\varepsilon} - \frac{\varepsilon_{start}}{\mathcal{R}^\varepsilon}$ 
14   $t \leftarrow t+1$ 
15 end

```

## 2.5 Performance evaluation

In this section, we quantify the performance gain from the proposed DRQN-based learning algorithm for heterogeneous task offloading and resource allocation problems in multi-fog networks using numerical experiments based on Python-TensorFlow simulator. We used three different environments which are equipped with Inter(R) Core i7-7500 U CPU @ 2.7GHz 64-bit

OS, Intel(R) Xeon(R) CPU E3-1225 v6 @ 3.3GHz 64-bit OS, and AMD Ryzen Threadripper 1920X 12-Core Processor.

### 2.5.1 Simulation settings

For our simulations, we consider a fog layer consisting of five fog nodes that are randomly distributed within a network area of  $100 \times 100 \text{ m}^2$ . In addition, a total of three different slices are created on top of each fog node. Slice characteristics are customized by the two-level of heterogeneity, namely the resource demands types and delay constraints, which are summarized in Table 2.2. As an example of slice characteristics in Table 2.2, slice  $k$  can be dedicated to Standard resource type services to meet the delay-critical constraint. To obtain realistic values for the processing capacities of fog nodes, we use the CPU processing densities and memory sizes from [Kwak *et al.* (2016)] which used real applications data including a YouTube video data set in [Cheng, X., Dale, C. & Liu, J. (2008)]. For slice  $k$  at fog node  $i$ , the task arrivals follow a Bernoulli distribution with parameter  $\lambda_{i,k}$  (in task/slot) and the packet size is  $5 \cdot 10^6$  bits. Additionally, the buffer size in each slice is 10, which means that a maximum of 10 slice tasks can stay in the buffer concurrently until processing terminates. The path loss constant and exponent are set to  $10^{-3}$  and 4, respectively. The bandwidth for each fog node is 1MHz. The transmission power of the fog node is 20dBm, while the noise power density is -174dBm/Hz [Lee *et al.* (2019)]. In regard to resource capacity distribution at fog nodes, the CPU speed of a fog node is randomly sampled from [5GHz, 6GHz, 7GHz, 8GHz, 9GHz, 10GHz], where the memory size of a fog node is randomly sampled from [2.4GB, 4GB, 8GB]. The allocation unit of CPU and memory resources are 1GHz and 400MB, respectively.

To evaluate the performance of different neural network settings, three neural networks are considered to estimate the Q-value in our simulation. For all of them, the output layer is a fully connected layer of  $|X_i(t)|$  units, where  $|X_i(t)|$  represents the dimension of the action set. Additionally, the activation function of the output layer is a linear activation function, which corresponds to the predicted Q-value of all possible actions. These neural networks differ from each other on the input layer and the hidden layers as detailed below.

Table 2.2 Two-level of heterogeneity values to service tasks in simulation

Resource Types	Delay Constraints		
	Critical	Sensitive	Tolerant
Standard	$D_k^{max} = 10\text{ms}$ $L_k^c = 400 \text{ cycles/bit}$ $L_k^m = 400 \text{ Mbytes}$	$D_k^{max} = 50\text{ms}$ $L_k^c = 400 \text{ cycles/bit}$ $L_k^m = 400 \text{ Mbytes}$	$D_k^{max} = 100\text{ms}$ $L_k^c = 400 \text{ cycles/bit}$ $L_k^m = 400 \text{ Mbytes}$
CPU intensive	$D_k^{max} = 10\text{ms}$ $L_k^c = 600 \text{ cycles/bit}$ $L_k^m = 400 \text{ Mbytes}$	$D_k^{max} = 50\text{ms}$ $L_k^c = 600 \text{ cycles/bit}$ $L_k^m = 400 \text{ Mbytes}$	$D_k^{max} = 100\text{ms}$ $L_k^c = 600 \text{ cycles/bit}$ $L_k^m = 400 \text{ Mbytes}$
Memory intensive	$D_k^{max} = 10\text{ms}$ $L_k^c = 200 \text{ cycles/bit}$ $L_k^m = 1200 \text{ Mbytes}$	$D_k^{max} = 50\text{ms}$ $L_k^c = 200 \text{ cycles/bit}$ $L_k^m = 1200 \text{ Mbytes}$	$D_k^{max} = 100\text{ms}$ $L_k^c = 200 \text{ cycles/bit}$ $L_k^m = 1200 \text{ Mbytes}$

- **DRQN**: for the design of the deep recurrent Q-network, the input to the network consists of  $Seq \times |O_i(t)|$ , where  $|O_i(t)|$  is the dimension of the state set and  $Seq$  is the sequence size for the 1D-convolutional network. The first hidden layer convolves 32 filters with a kernel size of 3 and applies a Rectified Linear Unit (ReLU). The second hidden layer convolves 64 filters with a kernel size of 3, again followed by a ReLU. This is followed by a recurrent layer in which we use GRU. The number of units in the GRU cell is 128 and the sequence length is 10. The final GRU state is followed by a fully connected layer with ReLU, which has 64 units.
- **DCQN**: the deep convolutional Q-network is almost identical to the deep recurrent Q-network except for a recurrent hidden layer. The resulting activations from the second convolutional hidden layer are followed by two fully-connected layers with ReLU, the first one has 128 units and the second has 64 units.
- **DQN**: we use four fully-connected hidden layers consisting of 64, 128, 128, and 64 units with ReLU.

In all the experiments, we use the Adam optimizer with a learning rate of 0.001 and learning starts after  $10^4$  iterations. A discount factor  $\gamma$  of 0.98 is used in the Q-learning update. The replay memory size of  $\mathcal{D}_i$  is  $10^4$ . The target network parameters  $\mathcal{C}$  is updated every  $10^3$  time slots. We use a mini-batch size of 32 transition samples per update. The  $\varepsilon$ -renewal factor

$\delta^\varepsilon$ ,  $\varepsilon$ -renewal rate  $\mathcal{R}^\varepsilon$ ,  $\varepsilon$ -start  $\varepsilon_{start}$  and minimum- $\varepsilon$   $\varepsilon_{min}$  are set to 0.9, 5000, 1 and 0.01, respectively.

For performance comparisons, the existing methods are simulated as baseline schemes. Given the large state and action spaces in the problem considered, we compare methods that are practicable using limited computational resources. Specifically, one baseline offloading method is used as follows:

- **Threshold Offloading with Nearest Node selection:** the node offloads its tasks only if the buffer is above a certain threshold and we set the threshold to 0.8 which implies that the task is to be offloaded if the buffer is more than 80% full. Also, the node selects the most adjacent neighboring node aiming to minimize communication delays and energy consumption which is an offloading algorithm widely used in IoT and device-to-device communications.

On the other hand, two conventional resource allocation algorithms are simulated as baseline schemes, namely:

- **Round Robin (RR):** this algorithm allows every slice that has tasks in the queue to take turns in processing on a shared resource in a periodically repeated order.
- **Priority Queuing (PQ):** this algorithm handles the scheduling of the tasks following a priority-based model. Tasks are scheduled to be processed from the head of a given queue only if all queues of higher priority are empty, which is determined by a delay constraint.

For different evaluation scenarios, we specifically assign three different cases in terms of slices' characteristics to analyze how each slice's different resource demands and delay priorities are interrelated to each other. Thus, the three simulation cases according to the three slices' characteristics are summarized in Table 2.3. It is noted that this evaluation can simply be expanded by configuring Table 2.2 and Table 2.3 to suit the needs of the fog network.

Table 2.3 Simulation cases according to the three slices' characteristics

Case	fog slice-1	fog slice-2	fog slice-3
<b>case-1</b>	Standard Delay-Critical	CPU-intensive Delay-Critical	Memory-intensive Delay-Critical
	All slices have the same delay constraint, but different resource type tasks		
<b>case-2</b>	Standard Delay-Critical	Standard Delay-Sensitive	Standard Delay-Tolerant
	All slices have the same resource type tasks, but different delay priorities		
<b>case-3</b>	Standard Delay-Critical	CPU-intensive Delay-Critical	Standard Delay-Sensitive
	Some slices have the same resource type tasks, but different delay priorities and some vice versa		

## 2.5.2 Performance analysis

In this subsection, we evaluate the performance of the proposed algorithm by comparing the simulation results under different system parameters.

### 2.5.2.1 Complexity analysis

In this section, the memory complexity and processing time of the proposed algorithm are investigated. The proposed DRQN-based learning algorithm described in Section 2.4.C requires storing a replay buffer which consists of the state  $O_i^t$ , action  $X_i^t$ , reward  $\psi^t$ , next state  $O_i^{t+1}$ , and valid action spaces of the next state  $\Xi(O_i^{t+1})$  for the target Q-network. In single transition experience, the state, action, and reward require storing a  $\text{len } A_i + \text{len } B_i + \text{len } B_i^e + \text{len } R_i = 3 \times K \times 2$  size array of decimal values, single integer numbers, and single float numbers, respectively. Moreover, the valid action space of the next state is an  $|\mathcal{X}_i|$  size array of binary values, where  $\Xi[X'] = 0$  if  $X' \in \mathcal{X}_i$  is invalid in state  $O_i^{t+1}$ , otherwise  $\Xi[X'] = 1$ . Finally, using the parameters in Section 2.4.A, the proposed algorithm requires much less memory than the conventional Q-learning algorithm, i.e., approximately 2.8GB compared to 7.24TB. It is worth mentioning that we leverage a Python-based simulator where the array as a whole is an object that stores the float data in consecutive regions of the memory, and thus the memory size of an

individual float is not explicit. Therefore, the memory usage compared corresponds to the data contained in the object.

Furthermore, multi-node learning alleviates the overhead of the network infrastructure as well as improves the system response time, compared to the centralized architecture. In our simulation, assuming each node is equipped with a single CPU, the processing time per iteration is around 0.04s. Moreover, the proposed neural network model can be trained in parallel on multiple CPUs or GPUs to improve the training time and memory usage.

### 2.5.2.2 Convergence performance

In this experiment, we evaluate the convergence property of different neural networks with the above parameter settings to confirm whether the proposed deep reinforcement learning-based algorithm can achieve stable performance. To quantify the impact of task traffic status on the convergence performance, we implement two different average task arrival rates, which are categorized into normal( $\bar{\lambda}=0.6$ ) and heavy( $\bar{\lambda}=0.8$ ), where  $\bar{\lambda}$  is the average task arrival rate per slice at the fog node. Since the uniform random policy runs for  $10^4$  iterations at all nodes before learning starts, the total average reward value is not enhanced during this time and thus we show the average total reward of fog nodes after they start learning their networks. Once each node starts learning its own state-action value functions with a preassigned neural network, the total average rewards are increasing and asymptotically converge after around  $1.5 \times 10^4$  iterations as shown in Fig. 2.4. In regard to the average task arrival rate, when the nodes receive a smaller number of tasks per time slot, the average total reward value is larger over all simulation cases. The main reason behind this is that the number of successfully processed tasks with limited resources of fog nodes is higher when a fewer number of tasks are waiting in the buffer and also that the buffers are less likely to be overflowed. Given the findings from this experiment, the proposed algorithm using DRQN can achieve greater total reward compared to DQN and DCQN. This result implies that DRQN controllers can handle partial observability by retaining some memory of previous observations to help the nodes achieve better decision making.

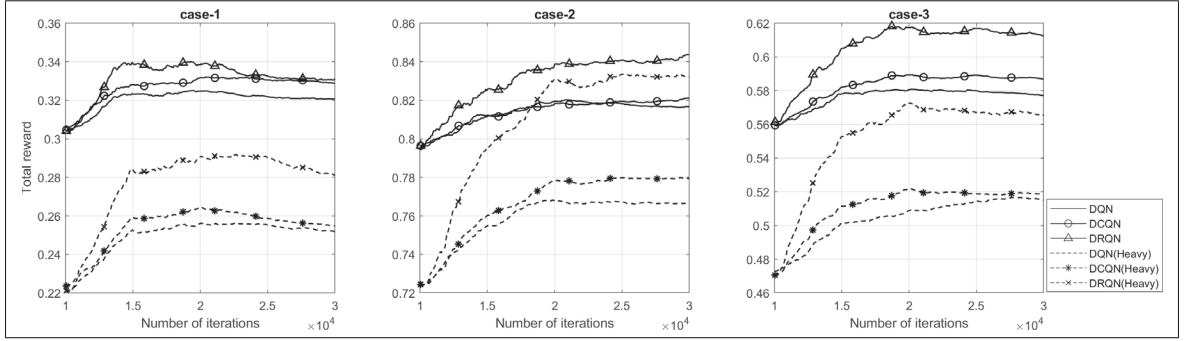


Figure 2.4 The convergence property of the proposed algorithm using different neural networks

### 2.5.2.3 Performance under different characteristics of fog slices

This experiment mainly aims to demonstrate the performance in terms of the average processing success rate, the average overflow rate, and the average task latency under different characteristics of fog slices as shown in Table 2.3. Like the previous experiment, two kinds of traffic demands, normal and heavy, are used for evaluation. Fig. 2.5 and Fig. 2.6 illustrates the average success rate of tasks and the average overflow rate per fog node. From Fig. 2.5 (a) and Fig. 2.6 (a), it can be observed that the proposed scheme using DRQN achieves the highest average success rate. Moreover, as the traffic requested increases, a larger number of tasks fail to attain their delay performance requirements due to the lack of resources. Comparing the three cases, when the nodes are requested the same traffic rate but demanding high computation and memory resources (Case-1), fog nodes are more likely to experience a lower success rate. This is because the processing time takes longer with limited resources, which also leads to failure of the delay requirements.

In Fig. 2.5 (b) and Fig. 2.6 (b), it is shown that, when implementing baseline methods, extremely large amounts of tasks are dropped due to overloading buffers. This is because fog nodes always select the most proximate fog node to offload their tasks, where the same fog node can be selected by several neighbors and thus its buffers will fill up quickly with tasks from multiple neighboring nodes. Furthermore, the resource allocation methods also affect the overflow performance. As we mentioned, the slices with large resource demands take more

processing time than the slices with small resource demands. Thus, when implementing RR and PQ methods, there is unfairness in the allocation of slices with small resource demands, which induces the high number of task drops from overloaded buffers. On the other hand, even though slices of Case-2 constitute the tasks with the same resource demands, their overflow rate is higher than that of Case-1 and Case-3. The difference in Case-2 is that the fog slice-3 of nodes are dedicated to delay-tolerant tasks where tasks of this slice can stay in the buffer until they exceed their large delay limit. Hence, the average processing success rate from Case-2 is higher due to relatively adequate delay limits, while the average overflow performance is worse. However, the proposed algorithm can offload their tasks to different neighboring nodes depending on their buffer and resource status and avoid unfairness among slices with different priorities in the resource allocation process, leading to an increased average success rate.

Moreover, Fig. 2.5 and Fig. 2.6 show that the variances of the success and overflow rates indicated by the error bars vary from one algorithm to the other. The error bars in these figures represent the largest value as the upper limit and the smallest value as the lower limit among all the nodes. For example in Fig. 2.5 (a), the success rate of Case-2 using DRQN has a mean of 95.6% and varies between 95.3% to 96.1%, while the success rate of Case-2 using nearest node selection with PQ resource allocation has a mean of 62.1% and varies between 36.8% to 90.8%. We can clearly see that the variability of the task success rate between fog nodes is greater for baseline methods than for the proposed algorithms, where the same trend is shown in the overflow rate. This result indicates that the proposed algorithm discourages selfish behavior in nodes and achieves a win-win cooperation between fog nodes by making rational offloading and resource allocation decisions.

Fig. 2.7 illustrates the average task delay under different cases. In contrast to the baseline methods, the proposed algorithm decreases the average task delay by selecting a neighboring fog node that minimizes transmission delay as well as waiting time in the buffers and allowing distinct resource allocation with respect to characteristics of each slice.



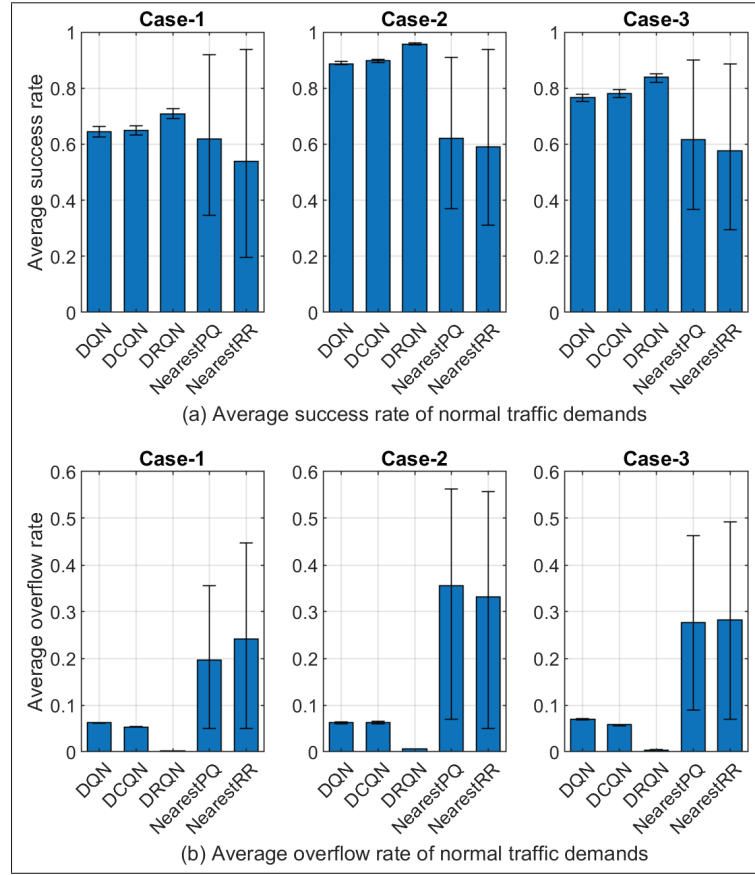


Figure 2.5 Performance of (a) average success rate and (b) average overflow rate of normal traffic under different cases

#### 2.5.2.4 Performance under different task arrival rates

In this experiment, we consider how the task arrival rates impact the average performance in terms of the average processing success rate, the average overflow rate, and the average task latency per slice. In this simulation, we set all fog nodes' slice characteristics to Case-2. In Fig. 2.8 (a), as the task arrival rate increases, more tasks fail to be completed within their delay limits. Similar observations can be found from Fig. 2.8 (b) where more tasks are dropped when the task arrival rate increases from 0.5 to 0.9. The reason behind this is that, as the task arrival rate increases, the waiting time becomes longer due to the larger number of tasks waiting in the buffer, which means that the tasks are more likely to fail their delay requirements or be dropped if they arrive when the buffer is full. Meanwhile, over the variation of the task arrival rate, the

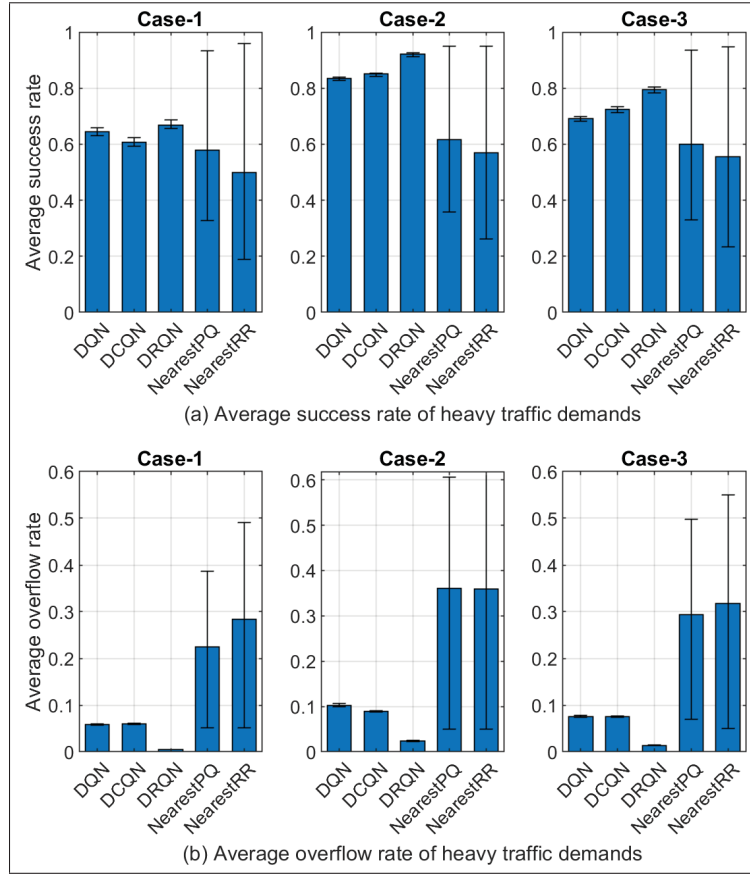


Figure 2.6 Performance of (a) average success rate and (b) average overflow rate of heavy traffic under different cases

maximum success rate and minimum overflow rate is achieved from the proposed algorithm. Moreover, as shown in Fig. 2.8 (c), the DRQN-based algorithm has the lowest average delay in Case-2.

These empirical results show that temporal integration of observations from a recurrent network allows the nodes to coordinate in their choices without knowing the explicit state and action sets of the others which makes the proposed DRQN-based algorithm relatively robust to the dynamics of the partially observable environment. These results also demonstrate that intelligently distributing resources to slices requiring different delay constraints makes a huge impact on the overall system performance.

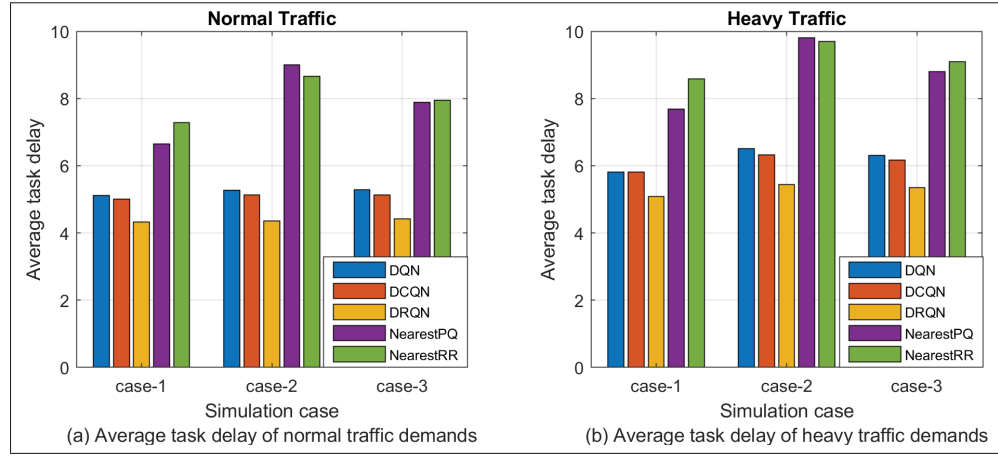


Figure 2.7 Task delay performance of (a) normal traffic and (b) heavy traffic under different cases

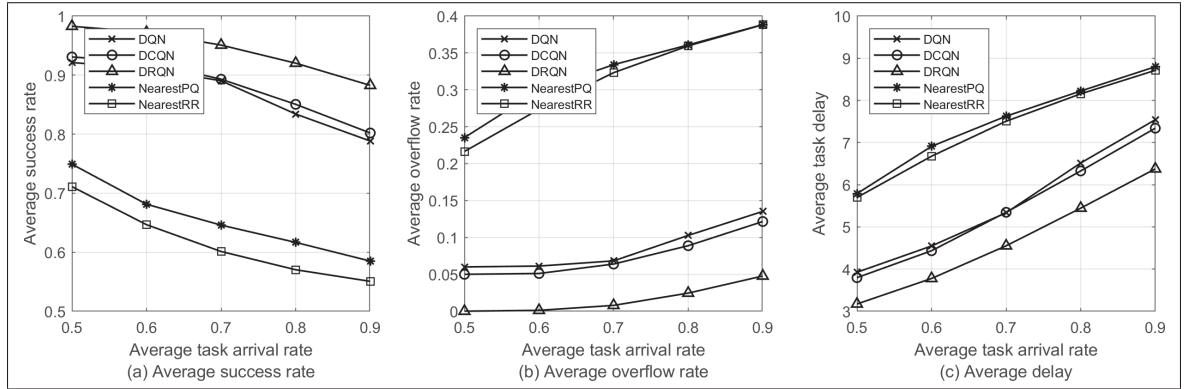


Figure 2.8 Performance of (a) average success rate, (b) average overflow rate, and (c) average task delay of Case-2 under different task arrival rates

## 2.6 Conclusion

In this paper, we devised a joint heterogeneous task offloading and resource allocation algorithm whose goal is to maximize the processing tasks completed within their delay constraints while minimizing the task drops from buffer overflows. The SDN-based fog network we consider has multiple fog nodes that are coordinating to achieve the best overall network performance without knowing the explicit status of other fog nodes. In the presence of uncertainties stemming from task demands and resource status, we formulate the problem as a partially ob-

servable stochastic game and apply cooperative multi-agent deep reinforcement learning with a global reward that aims to maximize the common goal of nodes and stabilize the convergence property. Further, we implement a recurrent neural network to tackle the partial-observability by maintaining internal states and aggregating temporal observations. Moreover, to guarantee the convergence and accuracy of the neural network, an adjusted exploration-exploitation method was adopted. Provided numerical results show that the proposed DRQN-based algorithm can achieve a higher average success rate and lower average overflow than DQN and DCQN as well as non-deep learning based baseline methods.

## CHAPTER 3

### ONLINE PARTIAL OFFLOADING AND TASK SCHEDULING IN SDN-FOG NETWORKS WITH DEEP RECURRENT REINFORCEMENT LEARNING

Jungyeon Baek<sup>a</sup> and Georges Kaddoum<sup>a</sup>

<sup>a</sup> Department of Electrical Engineering, École de Technologie Supérieure,  
1100 Notre-Dame west, Montreal, Canada H3C 1K3.

Paper published in *IEEE Internet of Things Journal*, Early Access, November 2021.

#### 3.1 Introduction

The goals of smart industrial developments are to achieve a higher level of operational efficiency and productivity, as well as a higher level of automatization [Lu, Y. (2017); Schumacher, A., Erol, S. & Sihm, W. (2016)]. However, they are vulnerable to many challenges created by volatile user demands and miscellaneous applications [Liu, X. F., Shahriar, M. R., Al Sunny, S. N., Leu, M. C. & Hu, L. (2017)]. The rapid proliferation of industrial Internet-of-Things (IIoT) devices and extensive data analysis are gradually increasing the load on remote cloud centers, which suffers high execution latency. Consequently, traditional cloud computing will be unable to guarantee real-time multitask and multi-object processing in future markets [Liu *et al.* (2017)]. To overcome these challenges, fog computing has emerged as an extension of traditional cloud computing, where computing, control, and networking can reside in multiple layers of a network's topology. Furthermore, fog computing is a distributed intelligent platform that exploits several technologies, including Software-defined networking (SDN) and virtualization [Schumacher *et al.* (2016)]. In this context, the deployment of SDN can enable the implementation and management of many aspects of resource allocation, virtual machine migration, topology monitoring, application-aware control, and programmable interfaces in fog computing [Tomovic *et al.* (2017)].

Considering its high scalability, close proximity, efficient information sharing, fog nodes can easily form a group and cooperate. Indeed, during a computation process, any individual fog

node failure could cause a catastrophe; however, with grouped fog nodes, one fog node failure can be compensated by other healthy nodes in the group [Aazam *et al.* (2018)]. Furthermore, computation tasks can be offloaded to more powerful nodes, enabling the fog node to share the load with others [Baek, J., Kaddoum, G., Garg, S., Kaur, K. & Gravel, V. (2019)]. The computation offloading model can be divided into full offloading and partial offloading models. In the full offloading model, also known as binary offloading, all the tasks can only be processed locally or offloaded to the other server as a whole. As comparing to the full offloading model, partial offloading refers to a scenario where a user's workload can be partially offloaded to other nodes and can partially executed locally. The partial offloading model is more suitable for delay-critical services since it takes advantage of parallelism [Tang, Q., Lyu, H., Han, G., Wang, J. & Wang, K. (2020)]. Moreover, unnecessary transmissions can be eliminated, which is essential as the bandwidth between fog nodes is limited.

In this regard, many works are devoted to partial offloading models. Ning, Z., Dong, P., Kong, X. & Xia, F. (2019) design an iterative heuristic mobile edge computing (MEC) resource allocation algorithm for the partial computation offloading problem by taking both MEC resource restriction and interference among IoT-based users into consideration. Kuang, Z., Li, L., Gao, J., Zhao, L. & Liu, A. (2019) study a partial offloading and power allocation problem in single-user MEC systems, whose is to minimize the execution delay and energy consumption while guaranteeing the transmission power constraint of the tasks. The authors in [Wang, Y., Sheng, M., Wang, X., Wang, L. & Li, J. (2016b)] focus on jointly optimizing communication and computation resources for partial computation offloading using dynamic voltage scaling, which aims to minimize the energy consumption and the application execution time. Muñoz, O., Pascual-Iserte, A. & Vidal, J. (2015) present an energy-optimal partial offloading strategy by parallelising the processing in a multiple-input multiple-output (MIMO) setup. Mao, Y., Zhang, J., Song, S. & Letaief, K. B. (2016) formulate a stochastic optimization problem to optimize the energy efficiency subject in aspect to the network stability, the maximum CPU frequency, peak transmission power, and energy causality constraints in mobile edge computing systems. Yadav, R., Zhang, W., Kaiwartya, O., Song, H. & Yu, S. (2020) propose an energy-

efficient dynamic computation offloading and resources allocation scheme, called ECOS, to minimize the energy consumption and service delay in a three-tier vehicular cloud network architecture. Tang *et al.* (2020) formulate the scheduling of service requests to virtual machines (VMs) as a bi-objective minimization problem, where a trade-off is maintained between the energy consumption and makespan.

On the other hand, computation offloading may not be beneficial due to possible long transmission delays and high transmission energy from remote task executions. As an operator, enabling networks to use as little energy as possible, while managing expected growth in data traffic is critical in terms of both cost and environmental impact. In [Kim, J. (2012)], the author introduces the concept of offloadable task, which is a task initially intended to process on a primary fog node but later could process on other nodes. An offloadable task is typically a task that requires high processing power and a small amount of data transmission. In contrast, a non-offloadable task is a task that requires extensive data communication time. For example, a task manipulating a large amount of memory footage located on the primary fog node is not offloadable since the benefit of offloading may not be justified because of high transmission cost. Neto, J. L. D., Yu, S.-Y., Macedo, D. F., Nogueira, J. M. S., Langar, R. & Secci, S. (2018) design and develop a comprehensive mobile computing framework that provides accurate execution time and energy consumption estimations to support the offloading decision in order to take care of the remote execution of the offloadable computations. Thus, to increase the benefits of cooperation between multiple computing nodes, offloading problems in fog networks should not only consider the dynamics of the environment but also take into account the characteristics of the offloading tasks.

However, existing methods based on one-shot optimization are no longer adequate to deal with the inconsistent network environment, which needs to re-calculate the optimal policies when the environment changes. To this end, deep learning (DL) techniques have become popular in fog computing networks to assure the requirements of complex applications [Zhang *et al.* (2019a)]. In particular, deep reinforcement learning (DRL), which has the significant advantage of not requiring apriori knowledge of the uncertain environment [Sutton & Barto

(2018)], has the potential to provide efficient solutions for future wireless networks. Here, the learning agent is not informed about which decisions to take but instead must discover which decisions yield the most reward by trying different approaches. Cheng *et al.* (2018) propose a novel resource provisioning and task scheduling algorithm using DRL aiming to minimize energy cost for large-scale cloud service providers with a large number of servers. Min *et al.* (2019) propose an RL-based computation offloading method with energy harvesting in a mobile edge network without complete knowledge of the energy consumption and latency model. Moreover, they expand this model to a DRL-based method to tackle the huge state dimension, which is processed with transfer learning to improve the performance. The authors in [Ke, H., Wang, J., Wang, H. & Ge, Y. (2019)] focus on jointly optimizing the transmission delay, renewable energy consumption, and bandwidth allocation based on a deep deterministic policy gradient method.

However, DRL is limited to fully-observable environments, which would constitute an invalid assumption for the application at hand. Nonetheless, deep recurrent reinforcement learning has been shown to handle partial-observations by taking leverage over a recurrent neural network architecture [Hochreiter, S. & Schmidhuber, J. (1997)]. The deep recurrent Q-network (DRQN), an extension of the DQN that combines recurrent neural networks (RNN), was proposed in [Hausknecht & Stone (2015)]. DRQN gives the network the ability to deal better with partially-observable models by integrating information over an extended period of time. The authors in [Sorokin, I., Seleznev, A., Pavlov, M., Fedorov, A. & Ignateva, A. (2015)] introduced deep attention recurrent Q-network (DARQN), where additional connections are added from the recurrent units to lower layers. This method allows the network to select which part of its next input needs attention, making the DARQN better than DQN on systems requiring long-term estimation.

Inspired by the above discussions, in this paper, we propose an online partial offloading and resource scheduling scheme in SDN-fog network where multi-fog nodes optimize the offloading decisions and resource allocation in a partially-observable environment. The key contributions of this paper are summarized as follows:



- We proposed an online partial computational offloading algorithm by jointly optimizing the number of offloading tasks, computational speed, and CPU utilization level to minimize energy consumption while maximizing the number of tasks successfully executed with the limited transmission and CPU resources. The CPU utilization level of computing nodes has a direct impact on the energy consumption and their resilience to failure, and thus fog nodes benefit from maintaining an optimal utilization level.
- We consider two different types of task demands, namely offloadable and non-offloadable types, which demand different data sizes, CPU processing densities, and delay constraints. They are segregated for processing in separate buffers. In contrast to the non-offloadable tasks, offloadable tasks could process on other nodes. However, if they are queued in the same buffer, offloadable tasks can be located behind non-offloadable tasks in the buffer. In this situation, offloadable tasks cannot be offloaded even if the fog nodes have communication resources to offload them to neighboring nodes.
- In the presence of uncertainties stemming from the user workload and limited resources, the independent cooperative fog nodes are modeled as a stochastic game, aiming to maximize the common goal for the overall system performance.
- To deal with the vast state space and partial-observability instead of the DQN, we apply a DRQN approach to approximate the optimal value functions and better deal with the insufficient state observations. The proposed learning scheme is fully online, where the weight updates of the neural networks are performed using information available at that time.
- Numerical experiments using Tensorflow are presented to validate the proposed model and algorithm.

The remainder of this article is organized as follows: Section 3.2 includes the system description. The partial offloading and task scheduling problem among the multiple cooperative fog nodes is presented in Section 3.3. In Section 3.4, we derive a DRQN method to solve the

problem. Simulation results are presented in Section 3.5. Finally, Section 3.6 concludes this paper.

### 3.2 System description

In this section, we introduce the system model which includes a hierarchical fog system architecture, user workload model, and energy consumption model. The time horizon is divided into decision epochs of equal durations  $\tau$  (in millisecond) and indexed by an integer  $t \in \mathbb{N}_+$ . The symbols used in this paper are listed in Table 3.1.

Table 3.1 List of Notations

Symbol	Definitions
$F$	The set of fog nodes
$I$	The number of fog nodes in the network
$K_i$	The total CPU resources of fog node $f_i$
$\eta_i^c$	The allocation unit of CPU resource at fog node $f_i$
$N_i$	The total CPU resource units at fog node $f_i$
$g_{i,j}$	The transmission bandwidth between node $f_i$ and node $f_j$
$d_{i,j}$	The distance between node $f_i$ and node $f_j$
$\Lambda_{i,n}$	The arrival rate for $n$ tasks at node $f_i$
$L_n$	The input data size of $n$ type task
$\gamma_n$	The required CPU cycles of $n$ type task
$D_n^{(max)}$	The deadline of $n$ type task
$B_{i,n}^{(t)}$	The number of $n$ type tasks queued in node $f_i$ at time $t$
$M_{i,n}^{(t)}$	The number of $n$ type task processing by node $f_i$ at time $t$
$R_{i,n}^{(t)}$	The number of $n$ type task removed from node $f_i$ at time $t$
$P_i^{(max)}$	The maximum power of fog node $f_i$
$\beta_1, \beta_2$	The path loss constant and exponent
$\kappa_i, \rho_i$	The coefficients representing the energy efficiency of node $f_i$
$\Pi_{i,(o)}$	The computation offloading policy of fog node $f_i$
$\Pi_{i,(r)}$	The task scheduling policy of fog node $f_i$
$\Psi_i$	The local reward observed by fog node $f_i$



same node does not induce a transmission delay, i.e.,  $g(i, i) = \infty$ . In this aspect, the system can be managed by mobile operators, cloud platform providers or other types of service providers.

Furthermore, the total computing resource of fog node  $f_i$  is  $K_i$ ,  $\forall f_i \in \mathcal{F}$ , which corresponds to its CPU speed (in cycles/ $\tau$ ). We are assuming that the maximum CPU capacity differs between fog nodes. It is determined at the initial stage and does not change over time. The fog nodes allocate their CPU resources on a resource unit basis, represented as  $\eta_i$ . Hence, the total number of CPU units at node  $f_i$  can be calculated as  $N_i = \lfloor \frac{K_i}{\eta_i} \rfloor$ . Finally, the cloud layer includes various servers that are capable of sufficient storage and computational resources but are physically remote from end-devices. We assume that the cloud server is much more computationally powerful than its associated fog nodes, i.e.,  $K_{cloud} \gg K_i$ ,  $\forall f_i \in \mathcal{F}$ .

To provide more distributed and scalable management, we leverage a SDN-based fog network [Chiang & Zhang (2016)]. To interconnect individual SDN controllers located in separate fog nodes, we apply the concept of inter-SDN communications, which connect controllers to share information and coordinate their decisions [D. Gupta and R. Jahan (2014)]. With the help of inter-SDN communications, each node deploys independent decision-making in its separate SDN controller, while communication between the SDN controllers of the nodes aims to exchange feedback required by the independent decision-making processes.

### 3.2.2 User Workload Model

We consider two types of tasks: offloadable tasks (*OFF*), which can be offloaded and processed on other nodes, and non-offloadable tasks (*NON*), that are only processed on a primary node. Monitoring tasks that manipulate data stored on a primary fog node and a confidential task which requires a special verification are examples of non-offloadable tasks. For each time slot  $t$ , the task arrival rates are  $\Lambda_{i,OFF}^{(t)} \in \{0, 1, \dots, \Lambda_{i,OFF}^{(max)}\}$  and  $\Lambda_{i,NON}^{(t)} \in \{0, 1, \dots, \Lambda_{i,NON}^{(max)}\}$  for *OFF* and *NON* tasks, respectively. Here, tasks of each type demand different data sizes, CPU processing densities, and delay constraints. Hence, each task is associated with a three-tuple parameter set  $(L_{OFF}, \gamma_{OFF}, D_{OFF}^{(max)})$  for *OFF* and  $\{L_{NON}, \gamma_{NON}, D_{NON}^{(max)}\}$  for *NON*, where  $L$ ,

$\gamma$ , and  $D^{(max)}$  denote the input task data size (bit), number of CPU cycles required per bit (cycles/bit), and deadline, respectively. Moreover, the arrived tasks get queued in two separate buffers until they are 1) processed locally at a primary node for  $\{OFF, NON\}$  tasks or 2) offloaded to other nodes for  $\{OFF\}$  tasks. Let  $B_{i,OFF}^{(t)}$  and  $B_{i,NON}^{(t)}$  be, respectively, the buffer length of offloadable and non-offloadable tasks of node  $f_i$  at the beginning of slot  $t$ . During operation, the processing state of node  $f_i$  is represented as  $\{M_{i,OFF}^{(t)}, M_{i,NON}^{(t)}\}$ , where  $M_{i,OFF}^{(t)}$  and  $M_{i,NON}^{(t)}$  are the number of offloadable and non-offloadable tasks that are processed on node  $f_i$  at time slot  $t$ , respectively.

Since we consider a partial offloading model, the computation offloading decision for node  $f_i$  at slot  $t$  specifies the number  $W_i^{(t)}$  of  $OFF$  tasks to be transmitted to node  $U_i^{(t)} \in \{1, 2, \dots, I+1\}$ , where  $U_i^{(t)} = i$  and  $W_i^{(t)} = 0$  if the node  $f_i$  processes all arrived tasks  $\Lambda_{i,OFF}^{(t)}$  locally, while  $U_i^{(t)} = I+1$  and  $W_i^{(t)} \neq 0$  implies that the node  $i$  offloads the number  $W_i^{(t)}$  of  $OFF$  tasks to the cloud server. Hence, the final number of  $OFF$  tasks to be queued in the buffer is  $\Lambda_{i,OFF}^{(t)} - W_i^{(t)}$ . Meanwhile, the task scheduling decision for node  $f_i$  at slot  $t$  specifies  $R_{i,OFF}^{(t)}$  and  $R_{i,NON}^{(t)}$ , which represent the number of tasks to be removed from the  $OFF$  and  $NON$  task buffers of node  $f_i$ , respectively. Therefore, the buffer lengths of node  $f_i$  are updated as follows:

$$\begin{aligned} B_{i,OFF}^{(t+1)} &= \min\{B_{i,OFF}^{(t)} + \Lambda_{i,OFF}^{(t)} - W_i^{(t)} + V_i^{(t)} - R_{i,OFF}^{(t)}, B_{i,OFF}^{(max)}\}, \\ B_{i,NON}^{(t+1)} &= \min\{B_{i,NON}^{(t)} + \Lambda_{i,NON}^{(t)} - R_{i,NON}^{(t)}, B_{i,NON}^{(max)}\}, \end{aligned} \quad (3.1)$$

here  $V_i^{(t)}$  is the number of  $OFF$  tasks offloaded from other nodes  $f_j \in F \setminus \{f_i\}$  to node  $f_i$  at slot  $t$ , and also  $B_{i,OFF}^{(max)}$  and  $B_{i,NON}^{(max)}$  are the maximum buffer size of offloadable and non-offloadable tasks, respectively.

### 3.2.3 Energy Consumption Model

The total energy consumption of node  $f_i$  in time slot  $t$  is composed of both the offloading transmission energy consumption and the CPU energy consumption. We first introduce the offloading communication model in the system. The transmission power for a node  $f_i$  at time

slot  $t$  is denoted as  $P_{i,(tr)}^{(t)}$ . The transmission rate from a node  $f_i$  to the selected node  $U_i^{(t)}$  for computation offloading is given by

$$v_i^{(t)} = g\left(i, U_i^{(t)}\right) \cdot \log \left( 1 + \frac{\beta_1 d_{i,g(i, U_i^{(t)})}^{-\beta_2} \cdot P_{i,(tr)}^{(t)}}{g\left(i, U_i^{(t)}\right) \cdot \sigma^2} \right), \quad (3.2)$$

where  $d_{i,U_i^{(t)}}$ ,  $\beta_1$ , and  $\beta_2$  are the distance between the node  $f_i$  and node  $U_i^{(t)}$ , the path loss constant, and the path loss exponent, respectively, whereas the variable  $\sigma^2$  is the noise power spectral density. Therefore, when offloading data of size  $L_{OFF} \cdot W_i^{(t)}$ , the transmission time of a node  $f_i$  can be defined as:

$$D_{i,(tr)}^{(t)} = \frac{L_{OFF} \cdot W_i^{(t)}}{v_i^{(t)}}, \quad (3.3)$$

where  $D_{i,(tr)}^{(t)} = 0$  if  $U_i^{(t)} = i$  as  $g(i, i)$  is infinite. For simplicity, we assume that the maximum available transmission power at the fog node is equal to the power consumption of the maximum task offloading and hence the node can transmit  $W_i^{(t)}$  tasks within one time slot, namely  $P_{i,(tr)}^{(t)} \leq P_{i,(tr)}^{(max)}$ . In other words, the transmission time for offloading tasks is fixed to the length of one time slot, i.e.,  $D_{i,(tr)}^{(t)} = \tau$ , and thus the transmission rate can be denoted as

$$v_i^{(t)} = \frac{L_{OFF} \cdot W_i^{(t)}}{\tau}. \quad (3.4)$$

Therefore, the power consumed by node  $f_i$  for reliable transmission of input data of offloadable tasks during a time slot  $t$  can be computed by combining (3.2) and (3.4) as

$$P_{i,(tr)}^{(t)} = \frac{g\left(i, U_i^{(t)}\right) \cdot \sigma^2}{\beta_1 d_{i,U_i^{(t)}}^{-\beta_2}} \left( 2^{\frac{L_{OFF} \cdot W_i^{(t)}}{\tau \cdot g(i, U_i^{(t)})}} - 1 \right). \quad (3.5)$$

Next, the CPU power consumption consists of the static  $P_{i,(st)}^{(t)}$  and the dynamic  $P_{i,(dy)}^{(t)}$  power consumption, where both are dependent on the CPU utilization rate of node  $f_i$  at time slot  $t$ ,

which is calculated as

$$Util_i^{(t)} = \frac{\sum_{n \in \{OFF, NON\}} \eta_i \cdot M_{i,n}^{(t)}}{K_i} \times 100\%. \quad (3.6)$$

We assume that fog nodes benefit from an optimal utilization level in terms of performance-per-watt, which is defined as  $Util_i^{(Opt)}$  for node  $f_i$  and is known as  $Util_i^{(Opt)} \approx 70\%$  for modern servers [Gao, Y., Wang, Y., Gupta, S. K. & Pedram, M. (2013)]. Thus,  $P_{i,(st)}^{(t)}$  is constant when  $Util_i^{(t)} > 0$  and zero otherwise. On the other hand,  $P_{i,(dy)}^{(t)}$  linearly increases when  $Util_i^{(t)}$  is under  $Util_i^{(Opt)}$ , and grows quadratically with the increase in power consumption beyond this level. Therefore,  $P_{i,(dy)}^{(t)}$  is calculated as:

$$P_{i,(dy)}^{(t)} = \begin{cases} \kappa_i \cdot Util_i^{(t)}, & \text{if } Util_i^{(t)} < Util_i^{(Opt)} \\ \kappa_i \cdot Util_i^{(Opt)} + \rho_i \cdot (Util_i^{(t)} - Util_i^{(Opt)})^2, & \text{if } Util_i^{(t)} \geq Util_i^{(Opt)} \end{cases} \quad (3.7)$$

where  $\kappa_i$  and  $\rho_i$  are coefficients characterizing the energy efficiency of node  $f_i$ . The total CPU power consumption of node  $f_i$  at time slot  $t$  is  $P_{i,(CPU)}^{(t)} = P_{i,(st)}^{(t)} + P_{i,(dy)}^{(t)}$ . The expected time to compute a task scheduled on node  $f_i$  can be calculated as:

$$D_{i,n,(CPU)}^{(t)} = \frac{\gamma_n \cdot L_n}{\eta_i}, \quad \forall n \in \{OFF, NON\}. \quad (3.8)$$

### 3.3 Problem formulation

In this section, we present the partially-observable MDP (POMDP) based offloading problem formulation and discuss the optimal control policy from a stochastic game perspective.

#### 3.3.1 POMDP-based Task Offloading Problem

At the beginning of each time slot  $t$ , fog nodes use an independent offloading policy to decide whether they will process the arrived offloadable tasks locally or offload them. If so, the policy also helps in determining tasks will be transmitted and to which node among neighboring fog

nodes in the same cluster and the cloud server. Furthermore, decisions on task scheduling are simultaneously made by fog nodes with regard to their own CPU resources through scheduling policies.

- **Observation:** We denote the local network state as  $\mathbf{X}_i^{(t)} = (\Lambda_i^{(t)}, V_i^{(t)}, \mathbf{B}_i^{(t)}, Ready_i^{(t)}) \in \mathbf{X}_i$  at node  $f_i \in \mathcal{F}$ , where  $\Lambda_i^{(t)} = (\Lambda_{i,OFF}^{(t)}, \Lambda_{i,NON}^{(t)})$  and  $\mathbf{B}_i^{(t)} = (B_{i,OFF}^{(t)}, B_{i,NON}^{(t)})$ .  $Ready_i^{(t)} = N_i - (M_{i,OFF}^{(t)} + M_{i,NON}^{(t)})$  denotes the available CPU resources of node  $f_i$  at time slot  $t$ . It is noted that the system is a POMDP because the underlying states of the system including the states of other fog nodes are not accessible by the fog node.
- **Action:** Each fog node  $f_i \in \mathcal{F}$  designs a control policy  $\Pi_i = (\Pi_{i,(o)}, \Pi_{i,(r)})$ , where  $\Pi_{i,(o)}$  and  $\Pi_{i,(r)}$  are the computation offloading and the task scheduling policies, respectively. With the observation of  $\mathbf{X}_i^{(t)}$ , node  $f_i$  determines the number of *OFF* computation tasks  $W_i^{(t)}$  that are to be offloaded to node  $U_i^{(t)}$  and the number of offloadable tasks  $R_{i,OFF}^{(t)}$  and non-offloadable tasks  $R_{i,NON}^{(t)}$  to be executed following  $\Pi_i$ , that is  $\Pi_i(\mathbf{X}_i^{(t)}) = (U_i^{(t)}, W_i^{(t)}, R_{i,OFF}^{(t)}, R_{i,NON}^{(t)})$ .
- **Reward:** The reward function determines what the decision maker needs to accomplish, i.e., the objective. The objective of each node is to find an optimal offloading and resource allocation solution that minimizes the total energy consumption and the task drops while maximizing the number of tasks being successfully executed. Therein, the objective problem is a multi-objective optimization problem that consists of multiple sub-objective functions. Solving this problem is not straightforward because sub-objectives are conflicting with limited bandwidth, CPU resources, and the delay constraints of tasks. To solve this problem, we convert the problem with multiple sub-objectives into a single-objective optimization problem, which is later used for the reward function. Considering the local observation and policies, we define the instantaneous reward function for node  $f_i$  at slot  $t$  as:

$$\Psi_i(\mathbf{X}_i^{(t)}, \Pi_i(\mathbf{X}_i^{(t)})) = \Psi_i^{(1)}(Profit_i^{(t)}) - \Psi_i^{(2)}(Drops_i^{(t)}) - \xi_i \cdot \Psi_i^{(3)}(P_{i,(tr)}^{(t)} + P_{i,(CPU)}^{(t)}), \quad (3.9)$$



where  $Profit_i^{(t)}$  and  $Drops_i^{(t)}$ , given in (3.10) and (3.11), denote the number of tasks that are completed within the deadline by task scheduling  $(R_{i,OFF}^{(t)}, R_{i,NON}^{(t)})$  and the number of task drops that occur when the buffer vacancy is less than the number of arriving tasks, respectively.

$$Profit_i^{(t)} = \sum_{R_{i,OFF}^{(t)}} \mathbb{1} \left( t + D_{i,OFF,(CPU)}^{(t)} \leq D_{OFF}^{(max)} \right) + \sum_{R_{i,NON}^{(t)}} \mathbb{1} \left( t + D_{i,NON,(CPU)}^{(t)} \leq D_{NON}^{(max)} \right) \quad (3.10)$$

$$Drops_i^{(t)} = \max \left\{ B_{i,OFF}^{(t)} + \Lambda_{i,OFF}^{(t)} - W_i^{(t)} + V_i^{(t)} - R_{i,OFF}^{(t)} - B_{OFF}^{(max)}, 0 \right\} \\ + \max \left\{ B_{i,NON}^{(t)} + \Lambda_{i,NON}^{(t)} - R_{i,NON}^{(t)} - B_{NON}^{(max)}, 0 \right\} \quad (3.11)$$

The third term on the right hand side of (3.9) denotes the energy cost corresponding to the overall energy consumption during a time step, consisting the energy consumed to transmit offloading tasks and the CPU energy consumed for processing scheduled tasks. Moreover,  $\xi_i \in \mathbb{R}_+$  is a constant weighting factor that balances the importance of the energy consumption within the time slot. In other words, the higher  $\xi_i$  is used, the more the nodes focuses on minimizing the energy consumption at the cost of deadline failures and task drops. This reward function allows fog nodes to execute as many tasks as possible with minimal energy consumption.

We apply the following constraints in the offloading and task scheduling of node  $f_i$  at time  $t$ ,

$$W_i^{(t)} = 0, \text{ if } U_i^{(t)} = i \text{ or } \Lambda_{i,OFF}^{(t)} = 0, \forall f_i \in \mathcal{F}, \quad (3.12)$$

$$W_i^{(t)} \leq \Lambda_{i,OFF}^{(t)}, \forall f_i \in \mathcal{F}, \quad (3.13)$$

$$R_{i,n}^{(t)} \leq \min \left( B_{i,n}^{(t)}, Ready_i^{(t)} \right), \forall n \in \{OFF, NON\}, \quad (3.14)$$

$$\left( R_{i,OFF}^{(t)} + R_{i,NON}^{(t)} \right) \leq Ready_i^{(t)}, \quad (3.15)$$

where constraints (3.12) and (3.13) ensure that the node cannot offload a task that didn't arrive and cannot offload more than the number of arrived tasks at time slot  $t$ . In addition, (3.14) and

(3.15) guarantee that the node cannot schedule more tasks than the number of tasks queued in the buffers and the available CPU resources.

### 3.3.2 Optimal Control Policy

In the presence of uncertainty stemming from the stochastic nature and limited resources, the POMDP-based problem is formulated as a stochastic game in which each node selects actions given local observation. In this paper, multiple cooperative fog nodes aim to minimize the energy consumption under a delay constraint. The state transitions and the sequences of instantaneous rewards per slot  $\{\Psi_i(\mathbf{X}_i^{(t)}, \Pi_i(\mathbf{X}_i^{(t)})), t \in \mathbb{N}_+\}, \forall f_i \in \mathcal{F}$  are determined according to the joint control policy. The cooperative fog nodes share their instantaneous rewards with each other through inter-SDN communication at each time slot, aiming to maximize the common goal on the overall system performance. The service quality experienced by users' tasks is determined by the offloading and task scheduling decisions of all fog nodes, and thus fog nodes must cooperate to achieve a desirable solution from an overall system perspective.

By taking the expectation with respect to the sequence of instantaneous rewards per slot, the expected long-term reward of the node  $f_i \in \mathcal{F}$  for a given initial local state  $\mathbf{X}_i^{(1)}$  can be defined as in (3.16), where  $\gamma$  is the discount factor ( $0 < \gamma < 1$ ).

$$V_i(\mathbf{X}_i, \Pi_i) = \mathbf{E} \left[ \sum_{t=1}^{+\infty} \gamma^{t-1} \cdot \sum_{f_i \in \mathcal{F}} \Psi_i(\mathbf{X}_i^{(t)}, (U_i^{(t)}, W_i^{(t)}, R_{i,OFF}^{(t)}, R_{i,NON}^{(t)})) \mid \mathbf{X}_i^{(1)} = \mathbf{X}_i \right] \quad (3.16)$$

Here, the aim of each fog node  $f_i$  is to attain an optimal control policy  $\Pi_i^*$  that maximizes the average rewards of all nodes, which can be formulated as

$$\begin{aligned} \Pi_i^* &= \underset{\Pi_i}{\operatorname{argmax}} V_i(\mathbf{X}_i, \Pi_i), \quad \forall \mathbf{X}_i \in \mathbf{X} \\ s.t. & \text{ (3.12), (3.13), (3.14), and (3.15).} \end{aligned} \quad (3.17)$$

With the objective optimization problem (3.17), each fog node can attain an optimal control policy that maximizes the average rewards of all nodes, which allows all fog nodes to execute a maximum number of tasks within their delay constraints with minimal energy consumption.

**Definition 3.1.** *An equilibrium describes the rational behavior of the nodes in a stochastic game. In our stochastic game, an equilibrium is a tuple of control policies  $\Pi_i^*, \forall f_i \in \mathcal{F}$ , where each  $\Pi_i^*$  of a node  $f_i$  is the optimal policy to other nodes.*

### 3.4 Learning the optimal computation offloading and task scheduling policies

This section proposes a Q-learning-based online optimal policy solution and discusses DRQN, which can better approximate actual Q-values, leading to better policies in a partially observable environment.

#### 3.4.1 Optimal policy solution using Q-learning

In order to solve the optimization problem (3.17), the well-known value iteration [Bellman, R. (1957)] can be implemented. However, this method requires complete knowledge of the state transitions and rewards, which is impossible without prior information of the user's workloads and available resources. To address this limitation, Q-learning can be used to find the optimal state-action value for any MDP without prior knowledge of the network transitions [Sutton & Barto (2018)]. Given the controlled system, node  $f_i$  repeatedly observes the current state  $\mathbf{X}_i^{(t)}$ , takes action  $\Pi_i(\mathbf{X}_i^{(t)})$  that incurs a transition, then it observes the new state  $\mathbf{X}_i^{(t+1)}$  and the reward  $\Psi^{(t)} = \sum_{f_i \in \mathcal{F}} \Psi_i^{(t)}$ . From these observations, the Q-function is updated as:

$$\begin{aligned} Q_i^{(t+1)}(\mathbf{X}_i^{(t)}, \Pi_i(\mathbf{X}_i^{(t)})) &= (1 - \alpha) \cdot Q_i^{(t)}(\mathbf{X}_i^{(t)}, \Pi_i(\mathbf{X}_i^{(t)})) \\ &+ \alpha \cdot \left[ \sum_{f_i \in \mathcal{F}} \Psi_i^{(t)} + \gamma \max_{\Pi'_i} Q_i^{(t)}(\mathbf{X}_i^{(t+1)}, \Pi'_i) \right], \end{aligned} \quad (3.18)$$

here  $\alpha$  is the *learning rate* ( $0 < \alpha < 1$ ). The most common action selection rule is the  $\epsilon$ -greedy algorithm that exploits the best action from recently learned Q-function (exploitation) most of

the time and explores other options by selecting a random action with a small probability  $\epsilon$  (exploration).

**Definition 3.2.** *Nash equilibrium will surely be reached in cooperative multi-agent Q-learning systems when the following conditions are satisfied [Claus & Boutilier (1998)]:*

- *The learning rate  $\alpha$  decreases over time such that  $\sum^t \alpha = \infty$  and  $\sum^t \alpha^2 < \infty$ .*
- *Each node visits every action infinitely often.*
- *The exploration strategy of each node is exploitative such that a probability of non-optimal action taken, based on estimated Q-values becomes zero as time increases.*

**Remark 1.** *The tabular based Q-algorithm requires the storage  $|\mathcal{X}| \times |\Pi|$  table of Q-values. For our study, the sizes of the local observation spaces  $|\mathcal{X}_i|$  and local action spaces  $|\Pi_i|$  are calculated as:*

$$|\mathcal{X}_i| = (1 + \Lambda_{i,OFF}^{(max)}) \cdot (1 + \Lambda_{i,NON}^{(max)}) \cdot (1 + B_{i,OFF}^{(max)}) \cdot (1 + B_{i,NON}^{(max)}) \cdot (1 + N_i)$$

$$|\Pi_i| = (I + 1) \cdot (1 + \Lambda_{i,OFF}^{(max)}) \cdot (1 + B_{i,OFF}^{(max)}) \cdot (1 + B_{i,NON}^{(max)})$$

When  $I = 5$ ,  $\Lambda_{i,OFF}^{(max)} = \Lambda_{i,NON}^{(max)} = 5$ ,  $B_{i,OFF}^{(max)} = B_{i,NON}^{(max)} = 10$ , and  $N_i = 10$ , one node  $f_i$  has to update a total of  $2.0872 \times 10^8$  Q-values, which makes it impossible for the Q-learning process to converge within a limited time steps.

### 3.4.2 Deep recurrent Q-learning with non-linear transformation

To solve the scalability issues of Q-learning, the DQN embraces the advantage of deep neural networks (DNNs) to improve the learning speed and the performance of Q-learning algorithms. However, DQNs assume that the agent has full-observability of the complete state information. To this end, we attempt to find an efficient algorithm that redeems incomplete state observations resulting from partial-observability. The authors in [Hausknecht & Stone (2015)] ob-

served that the performance of DQNs declines when given incomplete state information. Thus, they proposed the deep recurrent Q-network, which is a combination of an RNN and a DQN, to deal efficiently with the partial-observability of local network states by maintaining hidden states in the network. In feed-forward neural networks, all the inputs and outputs are independent, and hence the hidden layers cannot learn and understand the relationship between previous and current inputs. A recurrent neural network solves this issue with the help of the hidden state, which preserves the information and passes it from one step of the network to the next. More specifically, a Long Short Term Memory (LSTM) network resolves the vanishing gradient problem and the short-term memory of the vanilla recurrent neural network [Hausknecht & Stone (2015)]. LSTM consists of a cell that remembers information values over an arbitrary time and three gates (forget, input, and output) that regulate which values are relevant to keep and forget.

In this work, we consider a concurrent learning scheme for a multi-fog node system requiring cooperative behavior where each node learns its own value function model from local observations and hidden states while rewards are shared jointly by all nodes in order to maximize the overall system performance. Algorithm 3.1 discusses the procedure of the proposed learning algorithm at fog node  $f_i$ , which involves the following steps:

1. Initialize the replay buffer  $\mathcal{O}_i$  and parameters of neural networks  $Q_i$  and  $\hat{Q}_i$ .
2. At the beginning of time slot  $t$ , fog node  $f_i$  observes its local network state,  $X_i^{(t)} = (\Lambda_i^{(t)}, V_i^{(t)}, \mathbf{B}_i^{(t)}, \text{Ready}_i^{(t)})$ .
3. For the actor selection, with a small probability of  $\varepsilon$ , a random exploratory action is selected. Otherwise, fog node  $f_i$  selects the action associated with the largest Q-value output, i.e.  $\Pi_i^{(t)} \doteq \arg \max_{\Pi'} Q(X_i^{(t)}, \Pi'; \theta_i^{(t)})$ . Here, the fog node is only able to select actions from the valid action space, which is determined by the current observation value and is denoted as  $\text{valid}(X_i^{(t)})$ . Therefore, the probabilities of selecting invalid actions are set to zero, and the sum of the probabilities of the valid actions is normalized to 1. To this end, the fog

node is required to store the action probability array with the corresponding transition experience in the replay buffer.

4. Based on the actions taken by fog nodes, fog node  $f_i$  receives the reward and observes the next local state. Then, the transition experience,  $o_i^t = \langle X_i^{(t)}, \Pi_i^{(t)}, \Psi^{(t)}, X_i^{(t+1)} \rangle$  is stored into a replay buffer  $\mathcal{O}_i = \{o_i^{t-O+1}, \dots, o_i^t\}$ , where  $O$  is the replay buffer size.
5. At every learning step, given a random mini-batch of experiences  $\mathcal{J}_i$ , the DRQN is trained by adjusting the weights  $\theta_i^{(t)}$  to minimize the loss function,  $\mathcal{L}_i(\theta_i^{(t)})$ , where the loss function at time slot  $t$  is defined in (3.19).

$$\mathcal{L}_i(\theta_i^{(t)}) = \mathbb{E} \left[ \left( \sum_{f_i \in \mathcal{F}} \Psi_i^{(t)} + \gamma \max_{\Pi'_i} Q_i(X_i^{(t+1)}, \Pi'_i; \theta_{i,-}^{(t)}) - Q_i(X_i^{(t)}, \Pi_i(X_i^{(t)}); \theta_i^{(t)}) \right)^2 \right] \quad (3.19)$$

The Q-function in (3.18) can be approximated with LSTM network instead of the feed-forward network [Baek, J. & Kaddoum, G. (2021)]. Precisely, the DRQN estimates the Q-value of  $Q_i(X_i^{(t)}, H_i^{(t-1)}, \Pi_i(X_i^{(t)}); \theta_i^{(t)})$  by adding an extra input  $H_i^{(t-1)}$ , where  $H_i^{(t-1)}$  and  $\theta_i^{(t)}$  denote the hidden state of the network returned at the previous time slot and the parameters of the neural network, respectively. Thus,  $Q_i^{(t)}$  and  $H_i^{(t)}$  are the outputs of the DRQN at each time slot  $t$ , where the current hidden state  $H_i^{(t)}$  is calculated using information in  $X_i^{(t)}$  and  $H_i^{(t-1)}$ . Consequently, the weights  $\theta_i^{(t)}$  of the Q-network of node  $f_i \in \mathcal{F}$  are updated in a way that minimizes the squared error loss between the current predicted Q-value of  $Q_i(X_i^{(t)}, H_i^{(t-1)}, \Pi_i(X_i^{(t)}); \theta_i^{(t)})$  and the target Q-value of  $[\Psi^{(t)} + \gamma \max_{X' \in \mathcal{X}_i} \hat{Q}_i(X_i^{(t+1)}, H_i^{(t)}, \Pi'_i; \theta_{i,-}^{(t)})]$ . The gradient of  $\mathcal{L}_i(\theta^{(t)})$  w.r.t. the weights  $\theta_i^{(t)}$  is given in (3.20).

$$\begin{aligned} \nabla_{\theta_i^{(t)}} \mathcal{L}_i(\theta_i^{(t)}) = \mathbb{E} \left[ \left( \sum_{f_i \in \mathcal{F}} \Psi_i^{(t)} + \gamma \max_{\Pi'_i} Q_i(X_i^{(t+1)}, \Pi'_i; \theta_{i,-}^{(t)}) - Q_i(X_i^{(t)}, \Pi_i(X_i^{(t)}); \theta_i^{(t)}) \right) \cdot \right. \\ \left. \nabla_{\theta_i^{(t)}} Q_i(X_i^{(t)}, \Pi_i(X_i^{(t)}); \theta_i^{(t)}) \right] \quad (3.20) \end{aligned}$$

6. Furthermore, every time step ( $\mathcal{C}$ ), the parameters of the DRQN are copied to the target DRQN, which is used to generate the target Q-value for the following  $\mathcal{C}$  updates.

To update the LSTM layer, we can consider two update methods, i.e., Bootstrapped sequential updates and Bootstrapped random updates [Hausknecht & Stone (2015)]. In this analysis, we use the random update strategy in which the experience sequence  $\{o_i^k, \dots, o_i^{k+J}\}$  is selected randomly from the replay buffer, where  $J$  is the number of time steps, and updates proceed for only a single rolled LSTM cell that is unrolled  $J$  times. As our problem involves a continuing process, i.e., a non-episodic and never-ending process, the random update strategy is more applicable than the sequential update where updates begin at the beginning of the episode and progress through time until the end of the episode. Furthermore, the random update strategy is better suited for the experience replay in DQN that guarantees low correlations and better convergence behavior in the observation sequence [Mnih *et al.* (2015)]. As a consequence of using random updates, the hidden states in LSTM must be zeroed at the beginning of each update. We investigate the memory usage of the proposed DRQN networks in Section 3.5.2.1.

### 3.5 Performance evaluation

In this section, we quantify the performance gain from the proposed DRQN-based partial offloading method using numerical experiments with Python-Tensorflow simulator.

#### 3.5.1 Simulation settings

For our simulations, we considered that the fog network has a coverage of 100m, where  $I = 5$  fog nodes are randomly distributed in the region. Node  $f_i$ 's CPU capacity  $k_i$  is randomly set to  $\{6, 7, \dots, 10\}$  GHz while the bandwidth between two nodes is randomly set to  $\{0.5, 1, 1.5, 2, 2.5\}$  MHz. We refer to the values of CPU capacity and bandwidth from [Kwak *et al.* (2016)], which used real measurements of contemporary computing servers. Moreover, the path loss variables  $\beta_1$  and  $\beta_2$  are set to  $10^{-3}$  and 4 [Baek *et al.* (2019)], respectively. The task arrivals follow Poisson distribution with an average rate  $\lambda$  (tasks/ $\tau$ ) varying between 3 to 8 for each

Algorithm 3.1 DRQN algorithm for partial computation offloading and task scheduling of a fog node  $f_i \in \mathcal{F}$

```

1 Set Initialize replay buffer  $\mathcal{O}_i$ , state-action value function  $Q_i$  with random weights  $\theta_i$ ,
   target state-action function  $\hat{Q}_i$  with weights  $\theta_{i-} = \theta_i$ 
2 while ( $t \leq \text{maximum iteration}$ ) do
3   Observe the local network state  $X_i^{(t)} = (\Lambda_i^{(t)}, V_i^{(t)}, \mathbf{B}_i^{(t)}, \text{Ready}_i^{(t)})$  as an input to the
     network with parameter  $\theta_i^{(t)}$  Choose random action
      $\Pi_i^{(t)} = (U_i^{(t)}, W_i^{(t)}, R_{i,OFF}^{(t)}, R_{i,NON}^{(t)})$  from valid action spaces with probability  $\varepsilon$ 
     otherwise select  $\Pi_i^{(t)} \doteq \arg \max_{\Pi'} Q(X_i^{(t)}, H_i^{(t-1)}, \Pi'; \theta_i^{(t)})$  Execute action  $\Pi_i^{(t)}$ ;
     transmit the offloadable tasks and schedule the tasks
4   Observe local reward  $\Psi_i^{(t)}$ , next state  $X_i^{(t+1)}$  and get feedback from other nodes
      $\Psi_{j \neq i}^{(t)}$ 
5   Save transition  $\langle X_i^{(t)}, \Pi_i^{(t)}, \Psi^{(t)}, X_i^{(t+1)} \rangle$  in  $\mathcal{O}_i$ 
6   if learning step then
7     Sample a random mini-batch from  $\mathcal{O}_i$   $\mathcal{J}_i = \langle X_i^{(n)}, \Pi_i^{(n)}, \Psi^{(n)}, X_i^{(n+1)} \rangle$ 
8     Set  $\Psi^{(n)} + \gamma \max_{\Pi_i} Q_i(X_i^{(n+1)}, H_i^{(n)}, \Pi'; \theta_{i-}^{(n)})$ , and perform a gradient step in (19)
       w.r.t. the parameter  $\theta_i^{(n)}$ 
9     Reset the LSTM's hidden state  $H_i^{(n)}$ 
10  end
11  Every time step  $\mathcal{C}$ , update the target network parameters  $\theta_{i-}^{(t+1)} = \theta_i^{(t)}$ 
12  The time slot is updated by  $t \leftarrow t+1$ 
13 end

```

type of task. The input data size  $L_{OFF}$  and  $L_{NON}$  are both of  $5 \cdot 10^3$  bits [Chen *et al.* (2019)]. Additionally, we set  $B_{OFF}^{(max)} = B_{NON}^{(max)} = 10$ , which implies that a maximum of 10 tasks can wait in a buffer. The processing densities  $\gamma_{OFF}$  and  $\gamma_{NON}$  are of 600 and 400 cycles/bit, respectively. For the CPU energy consumption, the coefficients  $\kappa$  and  $\rho$  are set to 0.2 and 0.05 [Gao *et al.* (2013)], respectively, where the static power consumption is 3 W [Chen *et al.* (2019)].

To evaluate the utility of partial offloading over full offloading, we compare the proposed DRQN-based partial offloading algorithm with the DRQN-based full offloading algorithm. Also, the conventional DQN model for partial offloading is simulated to verify the need for recurrent networks in our problem. For the design of the DRQN, the input state is processed by



two 1D-convolutional layers, with a kernel size of 3, that apply a Rectified Linear Unit (ReLU), with 32 and 64 filters, respectively. This is followed by LSTM layer with 256 units. The final LSTM state is followed by a fully connected layer that has 128 units with a ReLU. The output layer is a fully connected layer of  $|\Pi_i|$  units with a linear activation function. For the design of the DQN, the input state is followed by four fully connected layers consisting of 128, 256, 256, and 128 units, respectively, consisting ReLU activation function, where the output layer is identical to that of the DRQN. All networks are trained using the Adam optimizer with a learning rate of  $10^{-3}$ . The discount factor  $\gamma$  is set to 0.98. The replay buffer contains one million most recent transitions. The number of time steps and the mini-batch size are set to 10 and 32, respectively. The network learns every five times slot, and the target network parameters are updated every  $10^3$  time slot.

In addition to that, considering performance comparisons, many existing solutions of optimization and meta-heuristic methods cannot be implemented to solve our problem due to the large search space as shown in Section 3.4.1. Therefore, we compare the proposed algorithm with two baseline schemes, namely, local processing and random offloading schemes, which demands feasible computational resources. For local processing and random offloading schemes, round-robin scheduling is simulated as a baseline task scheduling method.

### 3.5.2 Performance analysis

#### 3.5.2.1 Complexity analysis

This experiment investigates the complexity of memory in the proposed algorithm. The proposed DRQN-based learning algorithm described in Section 3.4.2 requires each node to store the replay buffer which contains the local state observation  $X_i^{(t)}$ , local action  $\Pi_i^{(t)}$ , reward  $\Psi^{(t)}$ , next state observation  $X_i^{(t+1)}$ , and valid action probability  $valid(X_i^{(t)})$ . In our research, the local observation and action arrays consist of six decimal values and a single integer value, respectively. In addition, the reward requires storing a single float number and the action probability array consisting  $|\Pi_i|$  binary values, where  $valid[\Pi'] = 1$ , if the action  $\Pi'$  is valid in state

observation  $X_i^{(t)}$ , otherwise  $valid[\Pi'] = 0$ . Furthermore, to calculate the memory required to load all neural network models in the system, we compute the number of trainable parameters calculated for each layer in the network. As a result, the DRQN and DQN networks have a total of 3,372,242 and 1,162,802 parameters, respectively. The reason behind the obtained result is that the DRQN model includes a single LSTM layer that requires weights for the input, forget, and output gates, and also the cell states. Although the DRQN model needs more memory than the DQN model, we found that the memory usage is tolerable and that the proposed DRQN-based algorithm can achieve a higher average success rate, lower overflow, and lower energy consumption, as shown in Section 3.5.2.3. The memory consumption of the proposed algorithm is about 1.6GB, which is much lower than the conventional Q-learning algorithm that demands approximately 25GB for the parameter settings in Section 3.5.1.

### 3.5.2.2 Convergence performance

This experiment evaluates the convergence property of the proposed DRQN-based offloading and task scheduling algorithm. To quantify the impact of the task arrival rate on the convergence performance, we implement different average task arrival rates. Fig. 3.2 validates the convergence behaviour of our method, which shows convergence after  $1.5 \times 10^4$  iterations. Considering the average task arrival rate, the average total reward is decreased while the task arrival rate increases. This is because the number of successfully processed tasks with limited fog node resources is lower when more tasks are waiting in the buffer. Besides, the proposed partial offloading method outperforms in comparisons with full offloading method in terms of the average reward, which implies that partial offloading is better when it comes to optimizing the expected long-term performance.

### 3.5.2.3 Performance under various average task arrival rates

This experiment demonstrates the performance in terms of average success rate, overflow rate, and energy consumption under different average task arrival rates. Fig. 3.3 and Fig. 3.4 illustrate the average success and overflow rates per fog node, respectively. From Fig. 3.3, it is

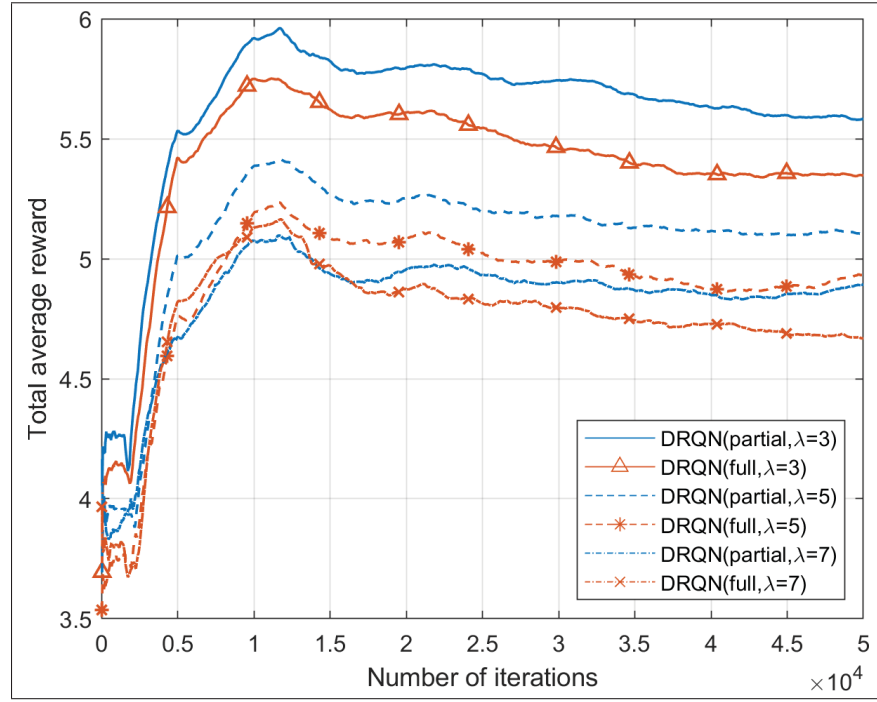


Figure 3.2 The convergence property of the proposed algorithm

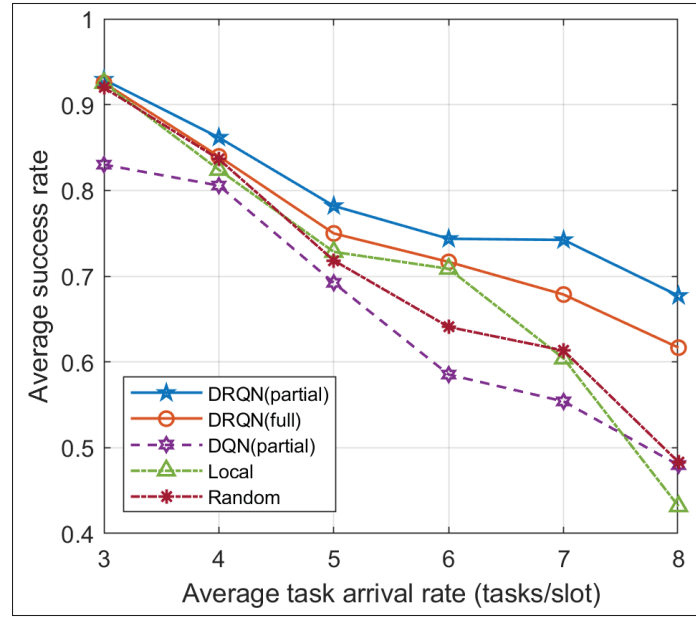


Figure 3.3 Average success rate per fog node under different average task arrival rates

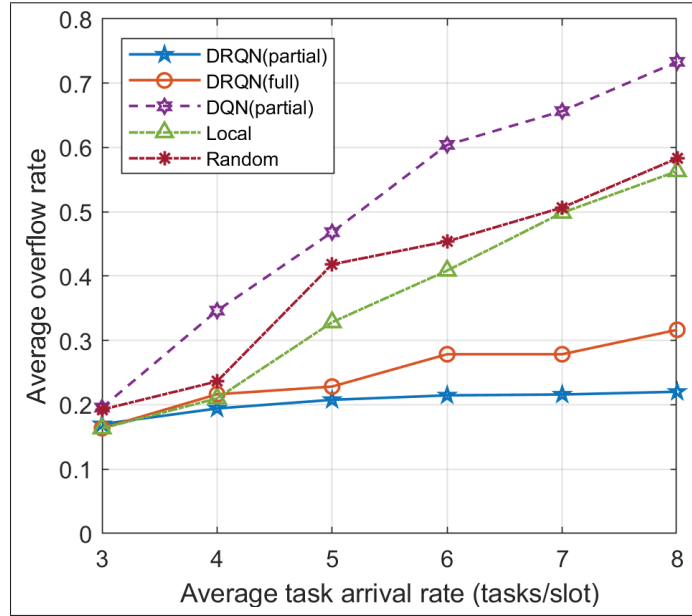


Figure 3.4 Average overflow rate per fog node under different average task arrival rates

observed that the proposed DRQN partial offloading algorithm is superior to the DRQN full offloading algorithm, which proves that partial offloading can achieve better performance for delay-critical services with limited communication and computation resources. In addition, the proposed algorithm outperforms the DQN-based method, random offloading, and local processing schemes. Compared to the local computing method, where the nodes execute all their tasks locally, the DRQN-based offloading algorithm achieves better results, especially for high task arrival rates. In this case, the proposed partial offloading algorithm and the full offloading algorithm tend to offload most of the computation tasks to the neighboring nodes or the cloud server to process them with limited computing resources. Thus the average success rate is higher than for the local computing method. Similar observations can be drawn from Fig. 3.4, which demonstrates that the average overflow rate from the proposed partial offloading algorithm is far lower than it was in baseline methods. Specifically, a much larger amount of tasks from the DQN-based method are dropped due to overflowed buffers, as the task arrival rate increases. These empirical results show that adding a recurrent layer to the DQN allows the network to estimate the partially-observable environment efficiently. Otherwise, making decisions from an inefficient observation in DQN can be arbitrarily wrong and

yield even worse performance than non-neural network-based methods. Here, Fig. 3.5 de-

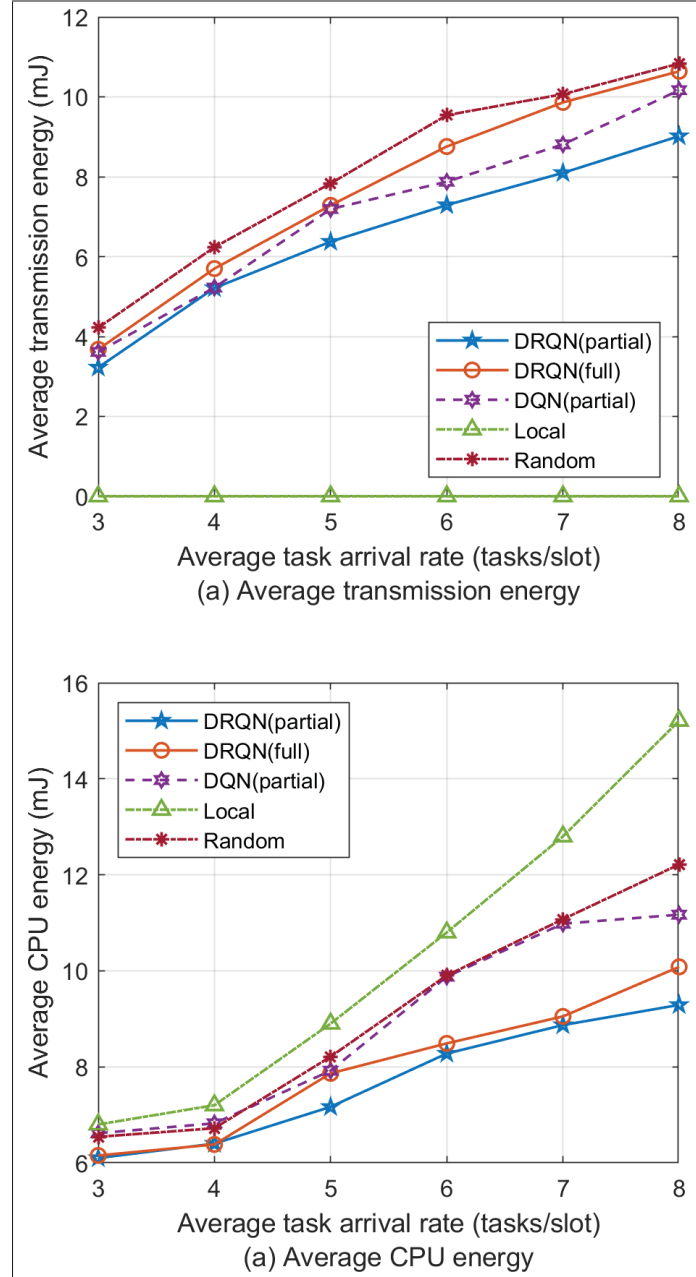


Figure 3.5 Average transmission energy and CPU energy consumption per fog node under different average task arrival rates

picts the average transmit energy and CPU energy consumption achieved from the proposed scheme and the three other baselines under different task arrival rates. It is observed that the

average energy consumption in the transmission of the proposed partial offloading algorithm is lower than that of the full offloading scheme. This is because the partial offloading algorithm can reduce unnecessary transmissions, which is essential given the limited bandwidth between fog nodes. Also, the average CPU consumption of partial offloading is much lower than that of the local processing and random offloading schemes. This result indicates that the proposed partial offloading algorithm not only offloads the offloadable computation tasks to other nodes (based on the trade-off between the merits of local computing and full offloading) but also exploits the affordable transmission energy while considering the queue length and assuring the corresponding delay constraint of each task.

### 3.6 Conclusion

In this paper, an online partial offloading and task scheduling problem is analyzed with the objective of minimizing the energy consumption under the corresponding delay constraint of each task. Over the scheduling slots, each fog node determines the offloading and task scheduling policies given its local observation, which is overall modeled as a stochastic game with cooperative behaviors to maximize the overall system performance. To solve the problem, we propose a deep Q-network-based algorithm to find optimal control policies. Furthermore, a recurrent neural network is applied in the DQN to tackle the partial-observability of local network states. The proposed DRQN-based method required comparatively less computational complexity than the conventional Q-learning algorithm. The experimental results show that the proposed DRQN-based partial offloading algorithm outperforms the DQN-based algorithm as well as non-neural network-based baseline methods. More specifically, the proposed method can effectively deal with both transmission and CPU energy consumption while guaranteeing convergence in a limited time. This work allows the offloading and resource allocation in fog networks to be more robust and adaptable to the difficulties arising from the high levels of volatility and partial observability.

## CHAPTER 4

### **FLOADNET: LOAD BALANCING IN FOG NETWORKS WITH COOPERATIVE MULTI-AGENT USING ACTOR-CRITIC METHOD**

Jungyeon Baek<sup>a</sup> and Georges Kaddoum<sup>a</sup>

<sup>a</sup>Department of Electrical Engineering, École de Technologie Supérieure,  
1100 Notre-Dame west, Montreal, Canada H3C 1K3.

Paper Submitted in *IEEE Transactions on Network and Service Management*,  
July 2021, Under Review.

#### **4.1 Introduction**

Tremendous developments in wireless networks, telecommunications, and informatics have paved the way for prevalent intelligence [Shafique, K., Khawaja, B. A., Sabir, F., Qazi, S. & Mustafaqim, M. (2020)], which will enable the future Internet of things (IoT). According to Cisco (2018), 29.3 billion networked devices will be connected to the internet by 2023, and global mobile devices will grow from 8.8 billion in 2018 to 13.1 billion by 2023. Among many challenges emerging as a direct result of the growing demands of the IoT, the inefficiency of the conventional cloud-based central computing paradigm in supporting massive connectivity of devices and providing them necessary nearly instantaneous response time is widely discussed. To address this limitation, fog computing has become increasingly popular as it offers virtual real-time computing solutions. Establishing a fog computing architecture involves locating servers closer to the data-generating IoT devices. In this context, reducing the resource cost and the end-to-end delay supporting real-time applications in the fog computing environment is both a research and an operational challenge for the current research community and industry. The main advantage of the fog computing architecture comes from its ability to transfer some of the cloud network's functions closer to the network edge. Given the physical proximity of the fog servers, computational workloads from end-user devices can be distributed among the servers, guaranteeing the latency constraints of applications, while respecting the bandwidth and power constraints of the network.

Despite the benefits of using fog computing, the uncertainties innate in fog networks, such as the incomplete information on future task arrivals, network link status, and diversified computing capabilities, bringing new challenges. Considering the limited computing power of edge servers, offloading tasks between servers across the different layers of the system for load balancing is a prominent feature in fog computing networks. While load balancing helps achieve high resource utilization, it also improves the overall performance of the system, in terms of throughput, latency, and energy consumption [Kaur & Aron (2021)].

In this context, finding an optimal computation offloading method for load balancing, which allows the avoidance of resource wastage while serving the best of each dissimilar service requirement, has been an active research area. Furthermore, the utilization of network devices directly affects the performance of the network and its resilience to failure [Tseng (2016)], and thus the desired network load and utilization are other critical aspects of the decision making process [Ghobaei-Arani, M., Souri, A. & Rahmanian, A. (2020)]. Hence, research efforts should not only be aimed at improving the computational resources but also at ensuring the network resource management is intelligent enough to flexibly and automatically adapt to sudden fog computing environment changes, as well as rapid traffic evolutions.

At the moment, the most existing learning solutions consider a system consisting of a centralized controller and multiple servers, where each server makes a decision based on the policy that is operated by the centralized controller [Zhang, C., Liu, Z., Gu, B., Yamori, K. & Tanaka, Y. (2018a); Sun, Y., Peng, M. & Mao, S. (2019c)]. However, such centralized solutions may lead to a violation of the latency constraints of delay-critical applications, or may be infeasible due to bandwidth and power constraints. Furthermore, it may impose security and privacy concerns arising from interactions and information exchanges with the centralized decision-makers.

Despite the advances made in decentralized resource management for fog computing networks in recent years [Liu, X., Yu, J., Feng, Z. & Gao, Y. (2020); Li, Y., Qi, F., Wang, Z., Yu, X. & Shao, S. (2020)], a number of challenges remain to be addressed, including the poor



scalability, heterogeneous workload demands, and the impact of integration on network performance. To address these challenges, distributed machine learning at the network edge provides a promising solution, where edge servers collaboratively train their models in an online manner subject to uncertain environments. In fully distributed systems, edge servers need to learn how to behave cooperatively as well as how to communicate with each other to effectively coordinate their decisions towards an universal goal. Consequently, distributed learning structures are naturally suitable for future network architectures and can be used to support distributed IoT demands.

Even though effective communications between edge servers is key to successfully distributed coordination, how benefits are achieved from communication is not necessarily determined, especially in complex settings with multiple learning agents where the optimal strategy is unknown. The recent rapid progress of machine learning and deep learning, in particular, opens the door to a new perspective on this aspect [Hernandez-Leal *et al.* (2019)]. In particular, reinforcement learning (RL) has become one of the most promising methods to solve such problems. In this context, to solve the scaling issue where a common environment is influenced by the joint actions of multiple decision-making entities, multi-agent reinforcement learning models arise as a natural solution [Oroojlooyjadid, A. & Hajinezhad, D. (2019)].

The multi-agent learning literature has been studied from a broad range of communities, such as RL, dynamic programming, game theory, heuristic search, etc [Hernandez-Leal *et al.* (2019)]. Among them, machine learning has been adopted as a popular approach to solve multi-agent system (MAS) problems because the complexity associated to such problems can make conventional heuristic solutions prohibitively infeasible. The simplest approach is to use the independent learners directly by applying single-agent algorithms in multi-agent settings [Tan, M. (1993)]. But this approach ignores the non-stationarity in multi-agent systems and can fail to learn from the past history of interactions when opponent learners adapt their behaviors. Hu, J. & Wellman, M. P. (1998) discussed theoretical perspectives on the multi-agent Q-learning algorithm for general-sum games, and showed that it converges to a Nash equilibrium. Claus & Boutilier (1998) studied the behavior of multiple agents employing Q-learning

independently, with a focus on how the exploration strategies affect the convergence to Nash equilibria. In [Peshkin, L., Kim, K., Meuleau, N. & Kaelbling, L. P. (2014)], the authors studied policy search method-based distributed learning applied to cooperative multi-agent domains with partially observable environments where agents have incomplete perceptions of the global state. Also, Schaerf, A., Shoham, Y. & Tennenholtz, M. (1995) particularly discussed multi-agent learning for adaptive load balancing. In this context, the goal of each agent is to adapt its resource selection behavior to the other agents' behaviors as well as to its environment's dynamics.

On the other hand, there are studies [Foerster, J. N., Assael, Y. M., de Freitas, N. & Whiteson, S. (2016); Sukhbaatar, S., Szlam, A. & Fergus, R. (2016); Kim, D., Moon, S., Hostallero, D., Kang, W. J., Lee, T., Son, K. & Yi, Y. (2019)] on the emergence of communication between agents using MARL. Such research usually considers a set of cooperative agents in a partially observable environment, where agents seek to maximize their shared payoff by means of communications. The authors in Foerster *et al.* (2016) proposed two methods, Reinforced Inter-Agent Learning (RIAL) and Differentiable Inter-Agent Learning (DIAL), to learn to communicate in DRL. Both methods were designed with a neural network that outputs the Q-values as well as a communication message to other agents in the subsequent time step. RIAL uses the concept of parameter sharing to share parameters of the deep recurrent Q-network (DRQN) among all agents while DIAL directly pushes gradients of continuous communication messages through the communication channel during training, and messages are discretized during execution. Also, Kim *et al.* (2019) proposed a MARL framework, called SchedNet where the agents communicate over a shared limited medium, and thus only a restricted number of agents can transmit their messages via communication scheduling. Their experiments showed that the learning of communication protocols is essential for the agents to coordinate their behavior, especially in partially observable environments with limited communication channels.

As discussed above, DRL is a promising and powerful tool that can provide autonomous and effective solutions to enhance the resource efficiency in fog computing networks. However, most research contributions focused on centralized approaches [Pan *et al.* (2019); Min *et al.* (2019);

Ke *et al.* (2019)], which makes modeling and computational complexity become challenging as the search space continues to increase. Though distributed schemes with multiple agents were considered in [Liu *et al.* (2020); Li *et al.* (2020); Wei, Y., Yu, F. R., Song, M. & Han, Z. (2019); Cui, J., Liu, Y. & Nallanathan, A. (2020); Nasir, Y. & Guo, D. (2019)], the prior studies has considered only computation offloading for server-level load balancing, and no work is done to optimize both the network and server load balancing together in fog computing environments.

This paper considers a joint network link and server load balancing with multiple cooperative access points (APs) in a fog network. The joint optimization problem is formulated as a stochastic game with the aim of minimizing the overloaded links and servers and the overall bandwidth cost. To this end, we propose the actor-critic RL framework, called FLoadNet, to optimize a joint load balancing policy in the fog networks. Motivated by the advantages of communication protocols in MARL [Foerster *et al.* (2016); Sukhbaatar *et al.* (2016); Kim *et al.* (2019)], we consider settings where cooperative APs as concurrent learning agents learn to communicate to coordinate the behavior of each individual and have one shared learning network as a way to improve learning. Thereby, the proposed approach will advance on the development of communication for efficient edge learning and the application of distributed learning algorithms to fog network optimization. To the best of our knowledge, no work has considered these problems jointly in a distributed manner with cooperative APs in the area of load balancing in fog networks. The key contributions of this paper are summarized as follows.

- The proposed approach addresses the minimization of the overall bandwidth cost and overloaded servers while balancing the computational load among all the available servers and network links. In this context, we grasp the latency for selecting edge server against server in fog and cloud layers through different weights for the bandwidth cost per link in different layers. An efficient load balancing method can allow not only a reduction in transmission delay at the link level but also faster processing at the server level.
- The proposed multi-agent load balancing model is formalized as a stochastic game. In the cooperative setting, each AP aims to maximize the global average payoff.

- We propose FLoadNet which is a multi-agent actor-critic RL framework for load balancing in fog networks. We extend the actor-critic model whereby the critic network facilitates the centralized learning by sharing parameters among the agents, while the individual actor networks strive to learn the optimal policy only using local information and communication messages. Due to the imperfect observations and limited communication channel capacity, the learning agents must discover a proper communication protocol that allows them to coordinate their decisions and achieve the optimal performance for all the agents.
- In particular, centralized learning enables the agents to not only share parameters of the neural network but also back-propagate errors with respect to the value functions and communication messages. As a result, each agent can give other agents more precise feedback on both the value function and communication messages, which reduces the required amount of learning iterations and accelerates the discovery of proper communication protocols.
- Numerical experiments using Tensorflow 2.0 are presented to support the proposed algorithm. Performance analysis is conducted using different values of the weight factors associated with each goal of the combined objective function. In other words, the weighting factors are used to adjust the priority to minimize link cost, link and server utilization, and task drop, respectively.

The remainder of this article is organized as follows: in Section 4.2, the system description and assumptions are presented. The stochastic game-based problem formulation and the best-response solution are discussed in Section 4.3. In Section 4.4, we describe the concrete framework of FLoadNet. Simulation results are presented in Section 4.5. Finally, Section 4.6 concludes this paper.

## 4.2 System description and assumptions

In this section, we define the concrete framework in which we study distributed offloading and load balancing. The model we propose captures adaptive offloading and load balancing in a three-layer fog computing network. The whole system operates across discrete scheduling

slots of equal duration (in milliseconds) that are indexed by an integer  $t \in \mathbb{N}_+$ . The symbols used in this paper are listed in Table 4.1.

Table 4.1 List of Notations

Symbol	Definition
<b>Constants</b>	
$I, i$	The set of access points, index
$K, k$	The set of task types, index
$d_k, h_k, c_k$	Data size, bandwidth, and CPU demands for task $k$
$P^i, p$	The set of available paths from AP $i$ , index
$L, l$	The set of all available links in the system, index
$s_p^i, L_p^i$	The destination server pool and set of links of path $p$
$\kappa_{l,p}^i$	Link-path indicator, where $\kappa_{l,p}^i = 1$ if path $p$ passes link $l$ , otherwise $\kappa_{l,p}^i = 0$
$g_l, \phi_l$	Bandwidth capacity and unit cost per bandwidth of link $l$
$L_e$	The subset of links included in the edge layer
$L_f$	The subset of links included in the fog layer
$L_c$	The subset of links included in the cloud layer
$\omega_s$	Computation capacity of single server
$N_a$	The number of servers in each AP's service pool
$N_f$	The number of servers in fog service pool
$N_c$	The number of servers in cloud service pool
$B_{max}^i$	The maximum task buffer size of AP $i$
<b>Variables</b>	
$W_t^i$	Arrived tasks for AP $i$ at time $t$
$\delta_{m,k,t}^i$	Task-type arrival variable, where $\delta_{m,k,t}^i = 1$ if the $m^{th}$ arrival task from AP $i$ at time $t$ is type $k$ , otherwise $\delta_{m,k,t}^i = 0$
$\eta_{m,p,t}^i$	Task-path allocation variable, where $\eta_{m,p,t}^i = 1$ if AP $i$ transfers the $m^{th}$ task along the path $p$ at time $t$ , otherwise $\eta_{m,p,t}^i = 0$
$y_{p,t}^i$	The total amount of bandwidth utilized on path $p$ by AP $i$ at time $t$
$\tau_t^i$	The total link cost for AP $i$ at time $t$
$B_t^i$	The length of task buffer of AP $i$ at time $t$

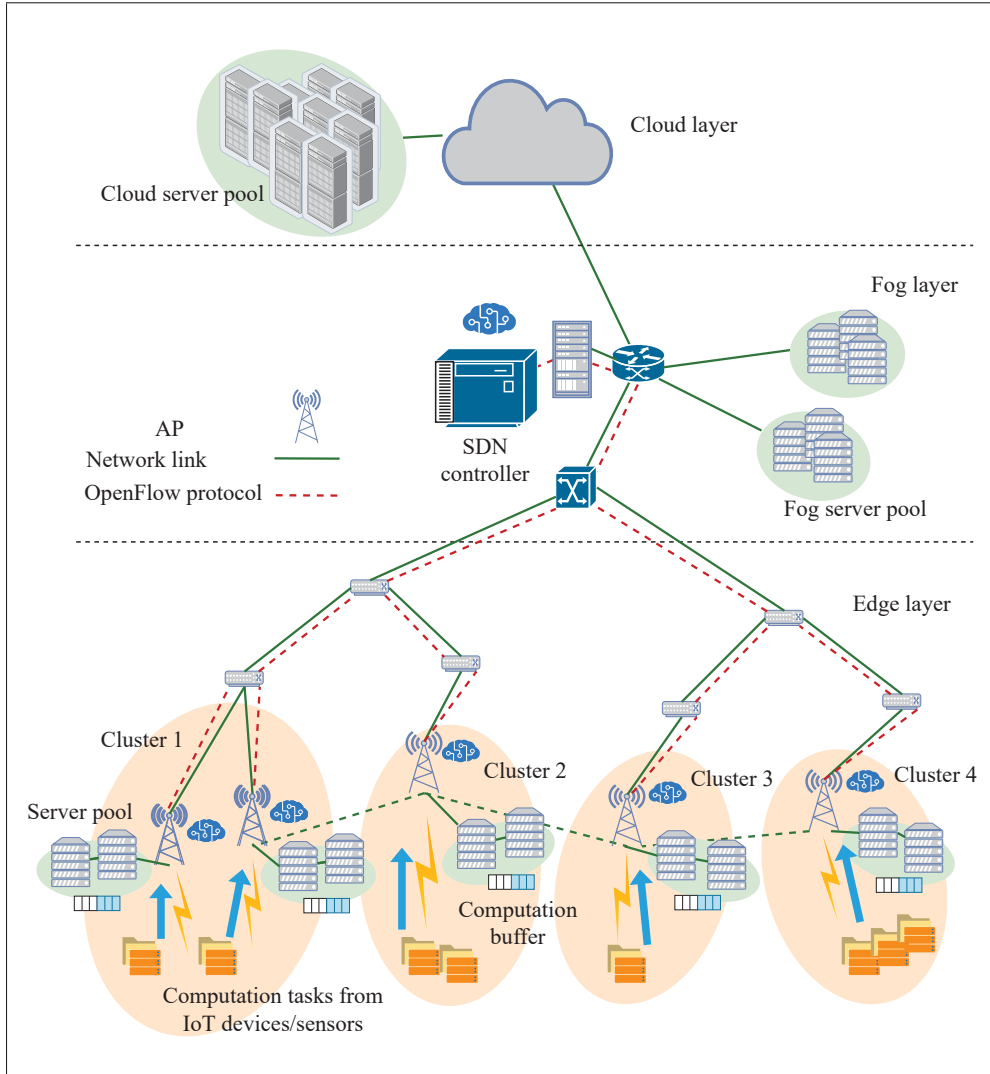


Figure 4.1 A hierarchic edge-fog-cloud computing network

### 4.2.1 Three-Layer Fog Networks

In this paper, we consider a hierarchical fog computing network consisting of three layers, namely the edge, fog, and cloud layers, as shown in Fig. 4.1. The edge layer consists of multiple APs  $i \in \mathcal{J} = \{1, \dots, I\}$ , which are grouped into clusters. Each AP receives computation demands from different types of edge-user devices and has its own pool of computational servers that is less powerful than server pools in the fog and cloud layers. Hence, the APs apply a cooperative computing scheme where the additional resources can be borrowed from

nearby clusters of the same layer as well as higher layers to support the demands from edge-user devices. In the proposed system, each AP has an autonomous decision-making capability aiming to share the system resources and coordinate its decisions with others. In what follows, we treat APs as decision-making agents who can sense the environment, act on it, and pursue their own objective, and hence we use the terms AP and agent interchangeably.

As illustrated in Fig. 4.1, the APs can send information required for their resource management to an SDN controller placed in the fog layer over OpenFlow communications. Using this communication, each AP  $i$  can also be informed of a set of available server pools and paths to offload its demands, where a server pool contains more than one server. For simplicity, we assume that the set of server pools and paths active in the coverage of the SDN controller is static over the time horizon. As a result, the APs and OpenFlow nodes (switches and routers) act as an IoT gateway, which provides a communication link between servers or layers and real-time control of edge devices. While there are related works [Schaerf *et al.* (1995)] associated with strictly distributed load balancing where the agents only use their own local information for making decisions, we don't impose a complete limitation on the communication between agents and instead fully encourage them to coordinate their decisions with each other and take full advantage of the SDN architectures. Given a hierarchical and distributed control design, APs operate on their local view and may exchange messages to enhance their knowledge, while processes that require network-wide knowledge are performed by a centralized controller in the fog layer. However, we acknowledge that some problems necessitate restrictions on the communications, such as real-time applications which require considerable restrictions on communication due to both latency and overhead. Hence, we allow communication with limited bandwidth, further described in Section 4.3.

#### 4.2.2 Task offloading and load balancing model

In this work, we consider heterogeneous tasks received from the APs, where the associated bandwidth and computation resource demands can differ. At the beginning of each scheduling slot  $t$ , each AP  $i$  receives a number  $W_i^t \in \mathcal{W} = \{0, 1, \dots, W_{i,max}\}$  of computational tasks. The

APs support  $k \in \mathcal{K}$  type of tasks, where these tasks are classified according to the resource demand configuration. The task  $k$  consists of a 3-tuple  $\langle d_k, h_k, c_k \rangle$ , where  $d_k, h_k$ , and  $c_k$  are the input data size (in bits), bandwidth resource (in MHz), and computation resource (in CPU cycles/bits), respectively. Hence, we define  $\delta_{i,w,k}^t$  as a task-type variable, where  $\delta_{i,w,k}^t = 1$  if the  $w^{th}$  ( $\leq W_i^t$ ) arrival task from AP  $i$  at time slot  $t$  is of type  $k$ , otherwise  $\delta_{i,w,k}^t = 0$ . In this work, we assume that the sequence of computational tasks  $(W_i^t, t \in \mathbb{N}_+)$  follows a Markov process, and the task arrival process is independent among the APs and across the scheduling slots. Furthermore, the type of each arrival task is determined by a probabilistic process.

During the scheduling slot, each AP  $i$  determines the scheduling decisions for each computation task. The computation tasks arrived in AP  $i$  can be processed locally by the servers in its own server pool or be offloaded to the server pools in other APs, the fog, or the cloud. In particular, AP  $i$  chooses a path  $p \in \mathcal{P}_i$ , which routes each of its demand to a specific service pool.

**Definition 4.1.** *The proposed multi-AP stochastic model is a 5-tuple  $\langle \mathcal{I}, \mathcal{W}, \mathcal{P}, \delta, \Pi \rangle$ , which involves the set of APs, the probabilistic arrival of new tasks to the APs, the path map, the probabilistic task types, and the joint load balancing policy.*

For each new task, the AP selects one of the paths, which is denoted as  $\Pi$  above, called a task-path allocation policy. This is further discussed in Section. 4.3.

### 4.2.3 Model objective and assumptions

In this section, we formulate an optimization model to mathematically represent the system framework of this work. In the proposed system architecture, there are multiple paths for transferring the tasks between each AP and server pool. We assume that the APs choose the paths that pass the smallest number of links, i.e. the shortest paths. In particular,  $\mathcal{P}_i$  represents the set of paths to the server pools which are accessible by AP  $i$ . Path  $p \in \mathcal{P}_i$  consists of a 2-tuple  $\langle s_{i,p}, \mathcal{L}_{i,p} \rangle$ , where  $s_{i,p}$  is the destination server pool and  $\mathcal{L}_{i,p}$  is the set of links on this path. Moreover,  $\kappa_{i,l,p}$  represents a link-path indicator, where  $\kappa_{i,l,p} = 1$  if path  $p$  passes link  $l$ , otherwise  $\kappa_{i,l,p} = 0$ .



Next, we define  $\eta_{i,w,p}^t$  as a task-path allocation variable, where  $\eta_{i,w,p}^t = 1$  if AP  $i$  transfers the  $w^{th}$  task along the path  $p$  at time slot  $t$ . Note that the number of tasks scheduled must be equivalent to the number of arrived tasks at time  $t$ :

$$\sum_{p \in \mathcal{P}_i} \eta_{i,w,p}^t = W_i^t. \quad (4.1)$$

The total amount of bandwidth utilized on path  $p$  by AP  $i$  can be calculated as

$$y_{i,p}^t = \sum_{w \in \mathcal{W}_i^t} \sum_{k \in \mathcal{K}} \delta_{i,w,k}^t \cdot \eta_{i,w,p}^t \cdot h_k. \quad (4.2)$$

If path  $p$  is selected to transfer the tasks from AP  $i$ , all the links  $\mathcal{L}_{i,p}$  used by this path carry the tasks to the destination server pool  $s_{i,p}$ . As a result, the total amount of bandwidth utilized on link  $l$  for all paths can be measured as

$$y_{i,l}^t = \sum_{p \in \mathcal{P}_i} \kappa_{i,l,p} \cdot y_{i,p}^t. \quad (4.3)$$

On the other hand, the bandwidth capacities of the links are different according to the layer where the links are included. For example, the capacities of the links included in the edge layer ( $g_l : \forall l \in \mathcal{L}_e$ ) are smaller than their counterparts ( $g_l : \forall l \in \mathcal{L}_f, \mathcal{L}_c$ ) in the fog and cloud layers, where  $\mathcal{L}_e, \mathcal{L}_f$ , and  $\mathcal{L}_c$  are the subsets of links included in the edge, fog, and cloud layers, respectively. Finally, the link  $l$ 's bandwidth utilization at time slot  $t$  can be calculated as

$$u_l^t = \frac{\sum_{i \in \mathcal{I}} y_{i,l}^t}{g_l}, \quad (4.4)$$

which shows the percentage of bandwidth utilized off the total bandwidth available. In order to avoid overloading the links, the maximum utilization of any link cannot be more than 1 at any point

$$0 \leq u_l^t \leq 1, \quad \forall l \in \mathcal{L}. \quad (4.5)$$

To verify the proposed load balancing scheme among all the available links, we assume that the total amount of bandwidth demands during each scheduling slot cannot exceed the available bandwidth capacity in the system. In other words, the available bandwidth capacity of the edge, fog, and cloud layers can fulfill the sum of the allocated bandwidth to establish paths from the APs to the server pools of all these layers as described in Eq. (4.6).

$$\sum_{i \in \mathcal{I}} \sum_{w \in \mathcal{W}_i^t} \sum_{k \in \mathcal{K}} \delta_{i,w,k}^t \cdot h_k \leq \sum_{l \in \mathcal{L}_e} g_l + \sum_{l \in \mathcal{L}_f} g_l + \sum_{l \in \mathcal{L}_c} g_l \quad (4.6)$$

Upon determining the load balancing decisions of tasks on dedicated paths, the total link cost for AP  $i$  at time slot  $t$  can be measured as

$$\tau_i^t = \sum_{l \in \mathcal{L}_{i,p}} \phi_l \cdot y_{i,l}^t, \quad (4.7)$$

where  $\phi_l$  is a unit cost per bandwidth for link  $l$ . We assume that all APs are synchronized, and the bandwidth demand required for each task is set to arrive to the destination server pool during the scheduling slot.

Meanwhile, a computation task buffer is maintained at each AP  $i$  to buffer the tasks that are decided to be processed locally and are offloaded from other APs. We denote the number of servers in the service pool of each AP, fog, and cloud as  $\mathcal{N}_a$ ,  $\mathcal{N}_f$ , and  $\mathcal{N}_c$ , respectively, where we assume that the capacity of a single server is equal to  $\omega_s$ . If all servers are busy in the service pool, the task waits in the buffer until a server becomes available. Then, the server removes the task from the buffer and starts processing it. Let  $B_i^t$  be the length of the task buffer of AP  $i$  at the beginning of slot  $t$ , which can be measured as

$$B_i^t = \min\{B_i^{t-1} + new_i^t - idle_{i,s}^t, B_{i,max}\}, \quad (4.8)$$

where  $new_i^t$  and  $idle_{i,s}^t$  represent the total number of arrived tasks and the number of servers becoming idle in the buffer of AP  $i$  at time slot  $t$ . Herein,  $new_i^t$  can be calculated as

$$new_i^t = \sum_{i \in \mathcal{I}} \sum_{w \in \mathcal{W}_i^t} \sum_{p \in \mathcal{P}_i} \eta_{i,w,p}^t \cdot \mathbb{1}(s_{i,p} = i), \quad (4.9)$$

where the second term of the multiplication is 1 if the destination server pool of the path  $p$  is that of the AP  $i$ , and 0 otherwise. Also,  $B_{i,max}$  describes the maximum buffer size for AP  $i$ . In addition, the number of task drops can be described as

$$Z_i^t = \max\{B_i^{t-1} + new_i^t - idle_{i,s}^t - B_{i,max}, 0\}, \quad (4.10)$$

which occurs due to the overload of AP  $i$  buffer at time slot  $t$ .

At the beginning of each time slot  $t$ , each AP repeatedly observes its own local state, which is denoted as  $X_i^t = \{\delta_i^t, B_i^t, \mathbf{U}_{i,l}^t\} \in \mathcal{X}_i$ , where  $\delta_i^t = (\delta_{i,m,k}^t : \forall w \in \mathcal{W}_i^t, \forall k \in \mathcal{K})$  and  $\mathbf{U}_{i,l}^t = (u_l^t : \forall l \in \mathcal{L}_{i,a})$ . Herein,  $\mathcal{L}_{i,a}$  is a set of adjacent links from AP  $i$ , which implies that each AP only has partial information on the link status in the network. Hence,  $\mathbf{X}^t = (X_i^t : \forall i \in \mathcal{I})$  describes the global state. Then, the APs simultaneously choose their load balancing decisions  $\eta_i^t = (\eta_{i,w,p}^t : \forall w \in \mathcal{W}_i^t, \forall p \in \mathcal{P}_i)$ . Likewise, the joint load balancing decision of all the APs is given by  $\eta^t = (\eta_i^t : \forall i \in \mathcal{I})$ .

The objective of the proposed model is to minimize the total link cost, maximum link utilization level, average length of the task buffer, and the number of task drops. To this end, we define the composite objective function as

$$F(\mathbf{X}^t, \eta^t) = \left( \sum_{i \in \mathcal{I}} \varpi_1 \cdot \tau_i^t + \varpi_2 \cdot B_i^t + \varpi_3 \cdot Z_i^t \right) + \varpi_4 \cdot \max\{u_l^t : \forall l \in \mathcal{L}\}, \quad (4.11)$$

where the first term denotes the total link cost corresponding to the total bandwidth that is allocated to all available links in the network. The second term denotes the length of the task buffer, and the third term is the number of task drops because of buffer overflow. Furthermore, the last term describes the maximum utilization level among the links. This objective function

allows the APs to balance task loads between available links with minimal bandwidth cost but also to monitor the risk for servers overload. Moreover,  $\varpi_1, \varpi_2, \varpi_3$ , and  $\varpi_4$  are constant weighting factors that balance the priority associated with each part of the objective function. For example, the more the APs focus on minimizing the link bandwidth cost at the expense of ideal link and server utilization and task drops, the higher  $\varpi_1$  is.

In summary, the goal of the optimization problem is to minimize (4.11) subject to the constraints (4.1), (4.5), and (4.6), namely

$$\begin{aligned} & \min F(\mathbf{X}^t, \boldsymbol{\eta}^t) \\ & \text{subject to (4.1), (4.5) and (4.6).} \end{aligned} \tag{4.12}$$

### 4.3 Game-theoretical problem formulation

In this section, we first adopt the game theory model of a stochastic game to formalize the proposed problem of multi-agent load balancing. Then, finding the best-response solution in game theoretic perspective is discussed.

#### 4.3.1 Stochastic game-based problem formulation

In the proposed scenario, each AP is responsible of solving the optimization model. First, we adopt the stochastic game settings to model the proposed load balancing problem and formalize the interaction between multi-agent and environment. Stochastic games are an extension of Markov decision processes (MDPs), which include multiple agents [Lisa, J. Yan and Nick, Cercone (2010)]. The environment is modeled by a finite set of states, and the dynamics of the environment employed by the other agents are unknown to the given agent. Based on the joint action/policy, the environment transitions into the next state and each agent receives a payoff.

**Definition 4.2.** *A load balancing stochastic game can be represented as  $\langle \mathbf{I}, \mathbf{X}, \boldsymbol{\eta}, T, F \rangle$ , where*

- $I = \{1, \dots, I\} \in \mathcal{I}$  is the set of agents;
- $X = (X_i : \forall i \in \mathcal{I}) \in \mathcal{X}$  is the joint state from all the agents;
- $\eta = (\eta_i : \forall i \in \mathcal{I})$  is the joint load balancing decision from all the agents;
- $T : \mathcal{X} \times \eta \times \mathcal{X} \rightarrow [0, 1]$  is a transition function, where  $T(x, \eta, x') = \Pr(x'|x, \eta)$  is the probability that the environment transitions to state  $x'$  when joint load balancing decision  $\eta$  is taken at state  $x$ , and  $\sum_{x'} T(x, \eta, x') = 1$ .
- $F : \mathcal{X} \times \eta \times \mathcal{X} \rightarrow \mathbb{R}$  is a payoff function.  $F_i(x_i, \eta, x'_i)$  is agent  $i$ 's payoff upon transitioning from state  $x_i$  to state  $x'_i$  given joint task-path allocation  $\eta$ .

The behavior of agent  $i$  in a stochastic game is described by a policy. The policy is a mapping  $\pi_i : X_i \rightarrow \Pr_{\pi_i}(\eta_i)$ , where  $\Pr_{\pi_i}(\eta_i)$  is a probability of taking action  $\eta_i$  in state  $X_i$  under stochastic policy  $\pi_i$ . A tuple of policies  $\Pi = (\pi_1, \dots, \pi_I)$  for all agents is called a joint load balancing policy. As the objective of this study is to obtain Eq. (4.12), (4.11) can be converted into a payoff function by negation ( $F = -G$ ). The objective of an agent is to maximize its accumulated payoffs presented as the agent's return. Then, the corresponding value function of agent  $i$  can be presented as in (4.13).

$$V_i^{\pi_i}(X_i) = \sum_{\eta_i} \pi_i(\eta_i) \sum_{F, X'_i} \Pr(X'_i, F(\mathbf{X}, \eta) | X_i, \eta_i) \cdot [F(\mathbf{X}, \eta) - f(\pi_i) + V_i^{\pi_i}(X'_i)] \quad (4.13)$$

#### 4.3.2 Finding an equilibrium solution in stochastic game

A policy  $\pi$  is considered as being better than a policy  $\pi'$  if its expected return is greater than that of  $\pi'$  for all states. Given the stochastic game model, the goal of each agent is to find a best-response policy that maximizes the value function (4.13). Thus, the best-response of agent  $i$  can be obtained as in (4.14). Note that a policy for a given agent can only be evaluated in the context of all the agents' policies. A Nash equilibrium (NE) is a collection of strategies for each of the agents where each agent's strategy is a best-response to the other agents' strategies

[Nash, J. F. (1950)]. Game-theoretical problems analyze the performance by the notion of a NE, which is the most common interpretation to capture rational behaviors in multi-agent games.

**Definition 4.3.** *The best-response policy for agent  $i$  is the set of all policies that are optimal given the other agents' policies  $\pi_{-i}$ . Formally,  $\pi_i^*$  is in the best-response function, if and only if,  $V_i^{<\pi_i^*, \pi_{-i}>}(x) \geq V_i^{<\pi_i, \pi_{-i}>}(x), \forall \pi_i \in \Pi_i, \forall x \in \mathbf{X}_i$ .*

Thus, the best-response of agent  $i$  can be obtained as in (4.14).

$$\begin{aligned} \pi_i^* = \operatorname{argmax}_{\pi_i(\eta_i)} \sum_{F, X_i'} Pr(X_i', F(\mathbf{X}, \eta_i, \eta_{-i}) | X_i, \pi_i(\eta_i | X_i), \pi_{-i}^*(\eta_{-i} | \mathbf{X}_{i-1})) \\ \cdot [F(\mathbf{X}, \eta_i, \eta_{-i}) - f(\pi_i) + V_i^{\pi_i}(X_i')] \end{aligned} \quad (4.14)$$

However, the approaches from game theory usually rely on the strong assumption that the game is perfectly known and observable to the agents. To address the limitations of game-theoretical approaches, the goal of RL techniques is to learn through interaction rather than solving an equilibrium [Sutton & Barto (2018)]. Instead of building an explicit model of the other agents' strategy, the agent learns through observation over time and selects actions in the environment based on observations of state transition and payoff. Littman, M. L. (1994a) proposed a Q-learning algorithm, called Minimax-Q, which converges to optimal decisions for two-agent zero-sum games. This is relatively straightforward to compute, as it can be calculated in polynomial time using linear programming. However, our proposed problem is defined as a general-sum stochastic game as agents' payoffs are not always negatively related (zero-sum) but are arbitrarily related. In general games, the learning problem has been proven much more challenging as the problem of computing any NE is now known to be PPAD-complete [Sodomka, E., Hilliard, E., Littman, M. & Greenwald, A. (2013)]. Due to the complexity of finding NE, a lot of recent works, such as cheap-talk [Foerster *et al.* (2016)], cognitive hierarchy, and side payment [Sodomka *et al.* (2013)], have been studied in solving correlated equilibrium. A joint policy is called a correlated equilibrium if no agent can improve its expected value by changing its policy. The main advantage of correlated equilibrium is that they

are computationally less expensive than NE and allow agents to correlate their actions in the game. In this study, we leverage communication protocols among the agents, allowing the possibility of playing correlated strategies.

#### 4.4 Learning the optimal policy using actor-critic methods in the multi-agent problem

In this section, we first present the RL model proposed to find the optimal solution in the stochastic game. Thereafter, we discuss an actor-critic method proposed to learn approximations to policy, value functions, and communication protocols in multi-agent task offloading for load balancing problems.

##### 4.4.1 Properties of the proposed learning algorithm

In multi-agent scenarios, where the joint state and joint action spaces grow exponentially with the number of agents in the system, RL approaches face significant scalability issues. Therefore, we use deep neural networks (NN) to represent both policies and value functions of the RL method, i.e. deep RL.

The main objective of this work is to find a deep RL algorithm that allows each agent to independently estimate and converge to a policy that enables all agents to achieve the maximum average payoff. In particular, the learning algorithm considered in the proposed problem is characterized by:

- **Concurrent learning:** concurrent learning uses multiple concurrent learners where each learner aims to achieve the common objective by modifying its own strategy from a separate learning process.
- **Identical payoff stochastic game (IPSG)** in [Matignon, L., Laurent, G. J. & Le Fort-Piat, N. (2009)]: all agents in a Markov environment receive the same payoff based on the joint policy. In this study, we consider an average-payoff setting where returns are defined by differences between instantaneous payoffs and the average payoff to all agents. This is

called the differential return  $F - f(\pi)$ , where  $f(\pi)$  is the average payoff while following the policy  $\pi$ . The average payoff is usually used in continuing problems where the interaction between agents and environment goes on forever without end or start states.

- **Communication protocol:** agents share information related to their current policies by exchanging communication messages over a limited bandwidth channel. This helps enhance the agents' knowledge and increase the size of the external information available to the agents. We are interested in such settings because since multiple learning processes and partial observability coexist in the proposed problem, the communication between agents is vital to coordinate the behavior of each individual. As the communication is implicit and is not given beforehand, the agents must collaborate and develop an appropriate protocol to accomplish the objective. Consequently, the agents learn a problem-specific communication protocol that aids converging to a correlated equilibrium.

#### 4.4.2 Learning the optimal policy and communication protocol via Actor-critic methods

To this end, we propose a new load balancing scheme in multi-agent fog networks using *Actor-Critic* framework with communications, called FLoadNet, whose overall architecture is depicted in Fig. 4.2. For simplicity, only two agents are presented in Fig. 4.2. FLoadNet consists of two components:

- **Distributed individual actor networks:** The individual actor network per AP is a function approximator with a neural network that produces the load balancing action distribution  $\pi_{\theta_i}$  for a given local state  $X_i^t$ .
- **Centralized critic network:** We extend the critic network, which outputs the value  $V_\phi^t$  and the communication message  $M_\phi^t$  given by the state  $X_i^t$  and combined messages  $\mathbf{M}_{\phi, -i}^{t-1}$  received from other agents at the previous learning step. Consequently, the critic network is trained to decode the coming messages to coordinate amongst the APs while evaluating the actions from the actor networks by computing the value function. Note that since the



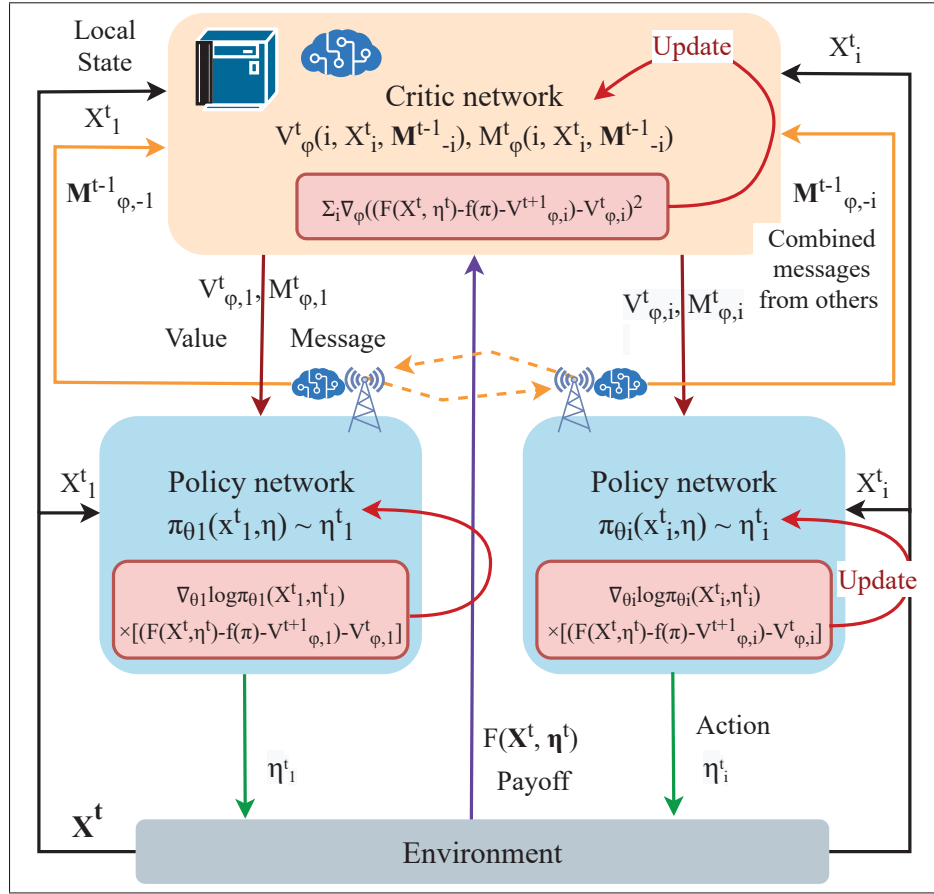


Figure 4.2 The proposed multi-agent learning using the Actor-critic policy gradient method with communication messages

communication messages are continuous, i.e.  $M_{\phi} \in \mathcal{M} \rightarrow \mathbb{R}$ , they can be trained via back-propagation, and thus be combined with the critic network.

According to the cooperative setting, we use *parameter sharing* for the critic network across all agents, in which the value functions and communication protocols of all agents are parameterized by the same weight,  $\phi$ . The advantage of parameter sharing is that the computational cost can be significantly reduced by sharing weight parameters, which also improves the learning speed by aggregating experiences across different agents. Despite using the same parameters, the agents still learn differently because they input their own agent index  $i$ , local states, and combined communication messages to the network, which makes each agent's learning process distinct.

Another modification to the conventional DRL method made in the proposed scheme is disabling experience replay on which deep Q-learning relies [Mnih *et al.* (2015)]. As we mentioned, multi-agent learning settings introduce non-stationarity as multiple agents learn concurrently, which makes experience obsolete and deceptive. To update the networks, we consider *n-step bootstrapping* methods. While the update of one-step temporal-difference (TD) methods is based on just the one next return, *n-step bootstrapping* methods perform an update based on the next *n*-step returns. *n-step bootstrapping* works better when a recognizable state change has occurred over a one-time step, but also the update can be more stable because of more real return information. Also, one important observation [Mnih *et al.* (2015)] is that the bias introduced through bootstrapping is often advantageous to reduce the variance and enhance the sample efficiency.

Algorithm 4.1 details the procedure of the FLoadNet learning algorithm. At the beginning of time step  $k$ , each agent receives communication messages from other agents at the previous time step,  $\mathbf{M}_{-1}^{k-1}$ . Simultaneously, each agent selects an action  $\eta_i^k$  with respect to the actor network  $\pi_i(X_i^k)$ . Then, the agents feed in the agent index  $i$  and the combined incoming messages  $\mathbf{M}_{-1}^{k-1}$  along with the local state  $X_i^k$  to the critic network. Once the outgoing messages are generated, the agents broadcast their messages to other agents. After all agents have implemented their actions, the state of the environment and total payoff are updated.

During the update steps, the agents look backward to recently visited states to accumulate the payoffs and gradients. Here, the agent  $i$  calculates a target value  $\Upsilon_i^k$  using the observed payoff and average payoff. As the steps go on, the average payoff will need to be updated as well, where  $\beta$  is the learning rate dedicated for the payoff update. Next, the policy of agent  $i$  can be improved through gradient ascent using the policy gradient. As each agent passes its batches through the centralized critic network individually, all the gradients from the agents are accumulated and applied in one optimizer step. To give feedback on communication messages, agent  $i$  accumulates the gradients with regard to its outgoing message  $M_i^k$ . The message gradient represents how much impact the outgoing message made on the TD error of other agents. After this, the weights of the actor and critic networks are updated with the

Algorithm 4.1 FLoadNet: Actor-critic method with communications for load balancing in multi-agent fog networks

```

1 Set Initialize the initial state  $X^0$ , the actor networks with random weights
    $\langle \theta_1, \dots, \theta_i \rangle$ , the critic network with random weights  $\varphi$ 
2 for  $t \in [0, \text{end}]$  do
3   for  $k \in [0, N]$  do
4     for Each  $i \in \mathcal{I}$  do
5       Receive messages  $M_{-i}^{k-1}$  from other agents
6       Select an action  $\eta_i^k$  using the actor  $\pi_{\theta_i}$ 
7       Get  $V_i^k, M_i^k = \text{Critic}(i, X_i^k, \mathbf{M}_{-i}^{k-1}; \varphi)$ 
8     end
9     Get next state  $X_i^{k+1}$ , payoff  $F^{k+1}$  given actions  $\eta^k$ 
10    Transfer communication messages to other agents
11  end
12  for  $k = N$  to  $1$ ,  $-1$  do
13    for Each  $i \in \mathcal{I}$  do
14       $\Upsilon_i^k = F^k - f(\pi_{\theta_i}) + V_i(X_i^{k+1}, M_{-i}^k; \varphi)$ 
15       $\Delta V_i^k = \Upsilon_i^k - V_i(X_i^k; \varphi)$ 
16       $f(\pi_{\theta_i}) \leftarrow f(\pi_{\theta_i}) + \beta \cdot \Delta V_i^k$ 
17      Accumulate the actor gradients using the critic:
18       $\nabla \theta_i \leftarrow \nabla \theta_i + \nabla_{\theta_i} \log \pi_{\theta_i}(X_i^k) \cdot [\Upsilon_i^k - V_i(X_i^k; \varphi)]$ 
19      Accumulate gradients for the value function:
20       $\nabla \varphi \leftarrow \nabla \varphi + \nabla_{\varphi} (\Delta V_i^k)^2$ 
21      Accumulate gradients for the communications:
22       $\mu_i^k = 1\{k < N - 1\} \sum_{j \neq i} \frac{\partial}{\partial M_i^k} (\Delta V_j^{k+1})^2$ 
23       $\nabla \varphi \leftarrow \nabla \varphi + \mu_i^k \cdot \frac{\partial M_i^k}{\partial \varphi}$ 
24    end
25    Update the actors using gradient descent:
26    for Each  $i \in \mathcal{I}$  do
27       $\theta_i \leftarrow \theta_i + \alpha_a \cdot \nabla \theta_i$ 
28    end
29    Update the critic using gradient descent:
30     $\varphi \leftarrow \varphi + \alpha_c \cdot \nabla \varphi$ 
31  end
32 end

```

accumulated gradients, where  $\alpha_a$  and  $\alpha_c$  are the learning rates of the actor and critic optimizer, respectively.

## 4.5 Performance evaluation

In this section, we evaluate FLoadNet and compare it with baseline methods. The numerical experiments are performed using the Python-Tensorflow 2.0 simulator to quantify the performance gain of FLoadNet.

### 4.5.1 Simulation settings

For simulation, we leverage a traditional 3-tier network topology matched with the cloud, fog, and edge layers in the proposed network. The edge switches in the edge layer receive tasks from the APs and forward them to the fog layer, while the fog switches connect the fog layer to the cloud. We divide the links into four subsets based on their bandwidth capacities, where  $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ , and  $\mathcal{L}_4$  consist of links from the APs to the edge switches, from the edge switches to the fog controller, from the fog controller to the fog servers, and from the fog controller to the cloud, respectively.

The distance to reach the servers of a given layer is set by the number of required links. Also, the bandwidth cost  $\phi_l$  to use links in edge, fog, and cloud layers is in increasing order, which implicitly reflects the delay cost to reach the servers of each layer. The number of servers in the server pool varies based on the layer that the server pool is located in, namely  $\mathcal{N}_a, \mathcal{N}_f, \mathcal{N}_c$ , where the cloud layer has the largest number since the cloud is the service of on-demand computing. To simplify, the computation capacity  $\omega_s$  of all the available servers of each layer is considered equal, and therefore the total server capacity of each server pool varies based on the number of servers. The parameters used in the experiments are listed in Table 4.2. It is noted that the topology used in this evaluation can be simply reconfigured to suit the needs of fog networks.

For the design of FLoadNet, the input state is followed by four dense layers consisting of 64, 128, 256, and 128 neurons, respectively, with the ReLu activation function for both the actor and critic networks. The output layer of the actor-networks is a dense layer consisting of  $|\eta_i|$  neurons, where  $|\eta_i|$  is the number of all possible actions from AP  $i$ . On the other hand, the critic

Table 4.2 The values of topology related parameters

Parameter	Value
The number of access points	8
The number of clusters	4
The number of access points per cluster	2
The number of edge and fog switches	4, 2
The capacity and cost of links in $\mathcal{L}_1$	100Mbps, 1
The capacity and cost of links in $\mathcal{L}_2$	300Mbps, 2
The capacity and cost of links in $\mathcal{L}_3$	500Mbps, 5
The capacity and cost of links in $\mathcal{L}_4$	1000Mbps, 10
The number of servers in $\mathcal{N}_a, \mathcal{N}_f, \mathcal{N}_c$	3, 15, 40
The capacity of server	2.5GHz

network has two separate output layers for the value and communication message functions, respectively. The activation function for the output of the communication message is the tahn function. The parameters are optimized using the Adam optimizer with a learning rate of 0.005 for the actors and 0.001 for the critic [Liu *et al.* (2020)].

For performance comparisons, five baseline methods are simulated:

- **FLoadNet w/o communications:** to assess the need for communication protocols, we compare FLoadNet with a version of FLoadNet which has the same features except for the communication protocols.
- **IL-based MA-Q** [Liu *et al.* (2020)]: independent learning agents are used to solve the proposed optimization problem in multi-agent settings. Each agent learns its own policy and treats dynamics from other agents as part of the environment. For DQN, the hidden layers and output layer are identical to that of the actor network in FLoadNet.  $\epsilon$ -greedy policy with  $\epsilon = 0.1$  is used, and the target network is updated every 500-time steps.
- **Local computing:** all APs execute their computation tasks using their own local servers and do not offload.
- **Round-Robin (RR):** each AP assigns each task among servers and paths in a circular order.

- **Random policy:** each AP generates a load balancing policy randomly in which the execution location of each task is determined at random.

## 4.5.2 Performance analysis

### 4.5.2.1 Convergence performance

In this experiment, we evaluate the convergence property of FLoadNet and compare it with other learning settings. To demonstrate the need for deep neural networks in the proposed multi-agent problem, we implement a shallow neural network-based FLoadNet, which consists of two hidden layers with 128 neurons. In the left four subplots in Fig. 4.3, we plot the loss functions of the actor and critic networks over  $5 \times 10^3$  time steps with different depth of the neural networks. Here, DNN-based FLoadNet results in a more stable gradient estimate across the learning procedures. On the other hand, the right two subplots in Fig. 4.3 present the loss functions of FLoadNet without communication protocols, which fails to converge after given the learning iterations. This result proofs that the communication between agents is crucial to coordinate the behavior of each individual in the non-stationary environment.

Fig. 4.4 illustrates the average payoff performance across different learning models. As we can see, the total average payoff of FLoadNet is increasing and asymptotically converges after around 1500 iterations. Meanwhile, the other two learning models do not converge properly and give unstable learning results. From our investigation, the policy of each agent can easily get stuck in a local optimum in the beginning of the learning procedures, when the learned policy may be only optimal to the current policies of the other agents at that time, which means the approximations of the value and policy are not accurate. Once more, this emphasizes the impact of learning a problem-specific communication that boosts performance, especially in a partially observed environment with concurrent learners.

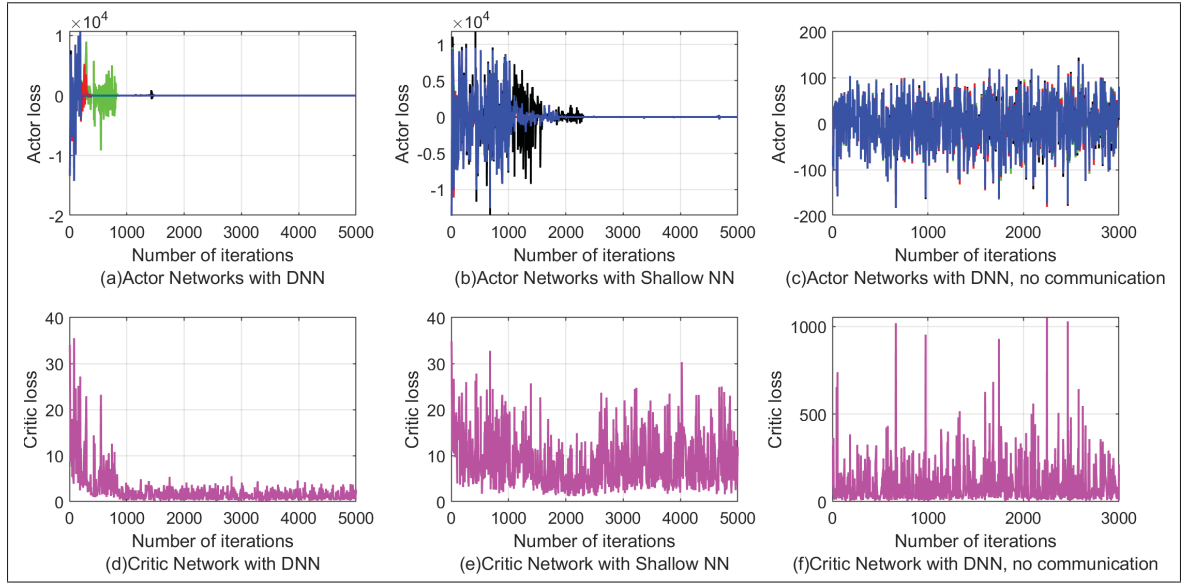


Figure 4.3 The convergence property across the learning procedure and versus depths of neural network

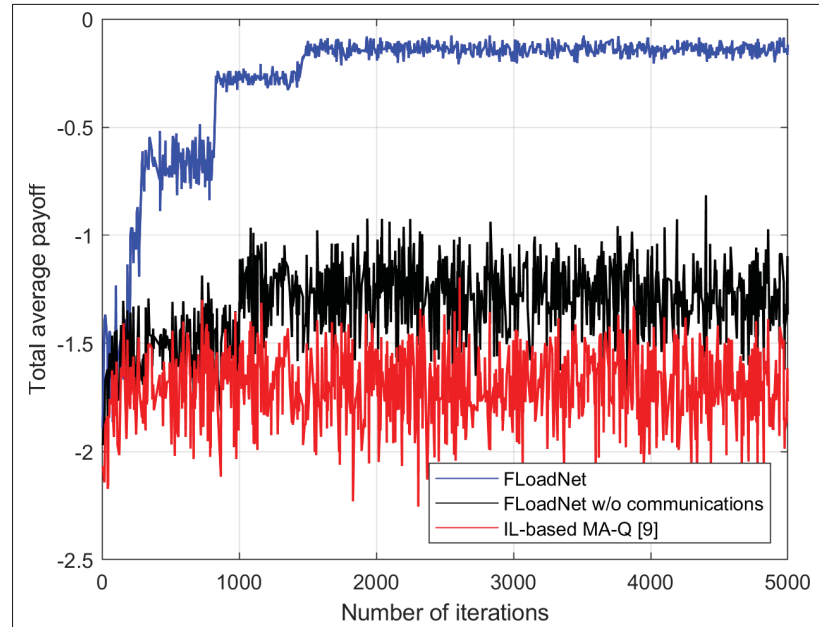


Figure 4.4 The average payoff performance across the learning procedures

#### 4.5.2.2 Variation in weights of link cost $\varpi_1$

This experiment demonstrates the performance, in terms of average link cost, maximum link utilization, and average server buffer length under different link cost weights  $\varpi_1$  in Eq. (4.11). From Fig. 4.5, it is observed that FLoadNet is superior to the baseline methods in minimizing the link bandwidth usage cost while minimizing the maximum link utilization and average buffer length in servers. Compared to the Round-Robin and Random policies, learning models can dynamically change their behavior based on the cost information. As  $\varpi_1$  increases, the APs are less likely to choose to send their tasks to the cloud, which is most costly due to the distance from the APs and then servers in the fog layer.

One of the interesting observations from the results is that two learning models without communication protocols show the decreasing average link cost as the weights of link cost are increased with the expense of overloaded links and server buffers. This is because these models minimize the link cost too much by allocating most tasks to local servers or servers in neighbor clusters. Unlike this, agents using FLoadNet can jointly minimize the link cost while distributing tasks among all the possible servers. This implies that FLoadNet can implicitly reduce the link-level delay as well as computation delay at servers.

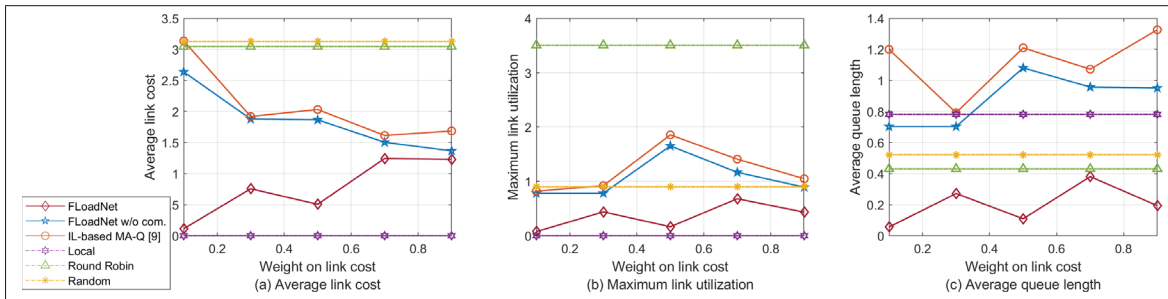


Figure 4.5 Average link cost, maximum link utilization rate, and average buffer length across the learning procedure versus link cost weight  $\varpi_1$



#### 4.5.2.3 Variation in weights of link utilization $\varpi_2$

Fig. 4.6 depicts how the average link cost, the maximum link utilization, and the server buffer length varies under  $\varpi_2$  in Eq. (4.11). When the value of  $\varpi_2$  is increased, a higher priority is pursued on the link-level utilization. All three learning models, including FLoadNet are able to regulate all the under-loaded links. However, the learning models with no communication are unsuccessful in reducing the link cost and server utilization, which shows a worse performance than static load balancing schemes.

Furthermore, FLoadNet shows a higher average link cost than the one of 4.5.2.3. The reason behind this is that FLoadNet distributes computation tasks among different layers, and this leads to the number of offloading to the fog and cloud layers increases. Therefore, the maximum link utilization can be balanced by applying an appropriate  $\varpi_2$  value, while adjusting the bandwidth cost within an affordable amount.

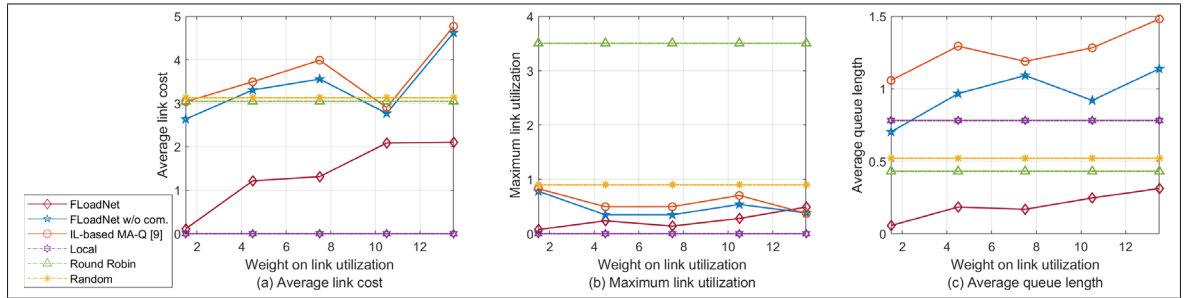


Figure 4.6 Average link cost, maximum link utilization rate, and average buffer length across the learning procedure versus link utilization weight  $\varpi_2$

#### 4.5.2.4 Variation in weights of server buffer length $\varpi_3$

Fig. 4.7 aims to demonstrate the performance under different weights  $\varpi_3$  on server buffer length in Eq. (4.11). The length of server buffers is a direct implication of the performance in terms of the processing latency, which is faster when a fewer number of demands are waiting in the buffer. Therefore, this is very useful in identifying the state of servers and preventing overloaded servers. Since the total server capacity available in the higher layers is larger than

the local capacity in APs, more computation tasks can be processed by the servers of the higher layer within one scheduling step. As a result, the server buffers in the APs tend to be more occupied.

In this context, as weight  $\varpi_3$  increases, APs using FLoadNet are more likely to send their tasks to the higher layer to adjust the number of tasks waiting in the buffers. However, if all the APs offload their tasks to the higher layers, the overall link utilization is increased, which can aggravate the delay performance by increasing the queuing delay at the link level. This trend can be seen in the results of two learning models with no communication protocols. Therefore, one of the key findings in this experiment is that cooperative APs, who only have local information, effectively optimize the trade-off between opposing performances utilizing communications.

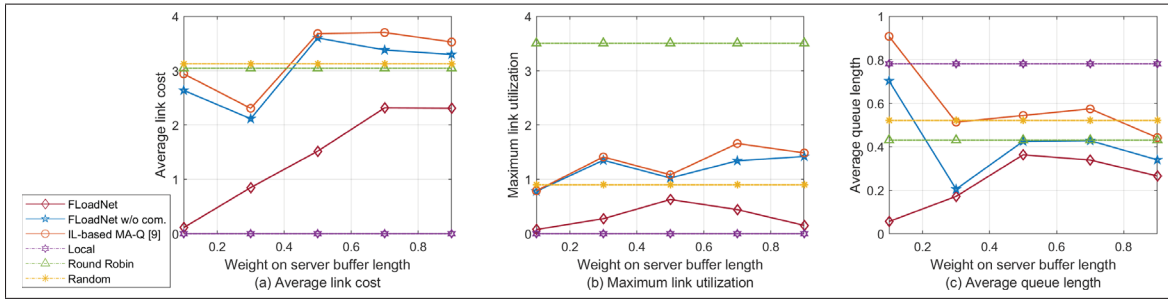


Figure 4.7 Average link cost, maximum link utilization rate, and average buffer length across the learning procedure versus server buffer length weight  $\varpi_3$

#### 4.5.2.5 Variation in bandwidth and computation resource demands

This experiment aims to show how the variation of bandwidth and computation resource demands impacts the performance. Figs. 4.8 (a) and (b) demonstrate the average link cost and the maximum link utilization, respectively, under different resource demands arranged in ascending of the average data and bandwidth size from left to right. As the resource demands increase, the average link cost also increases because the link cost of offloading a task is proportional to the size of the bandwidth demand. This is also because a larger number of tasks are sent to the higher layers, due to the lack of computation resources in the APs.

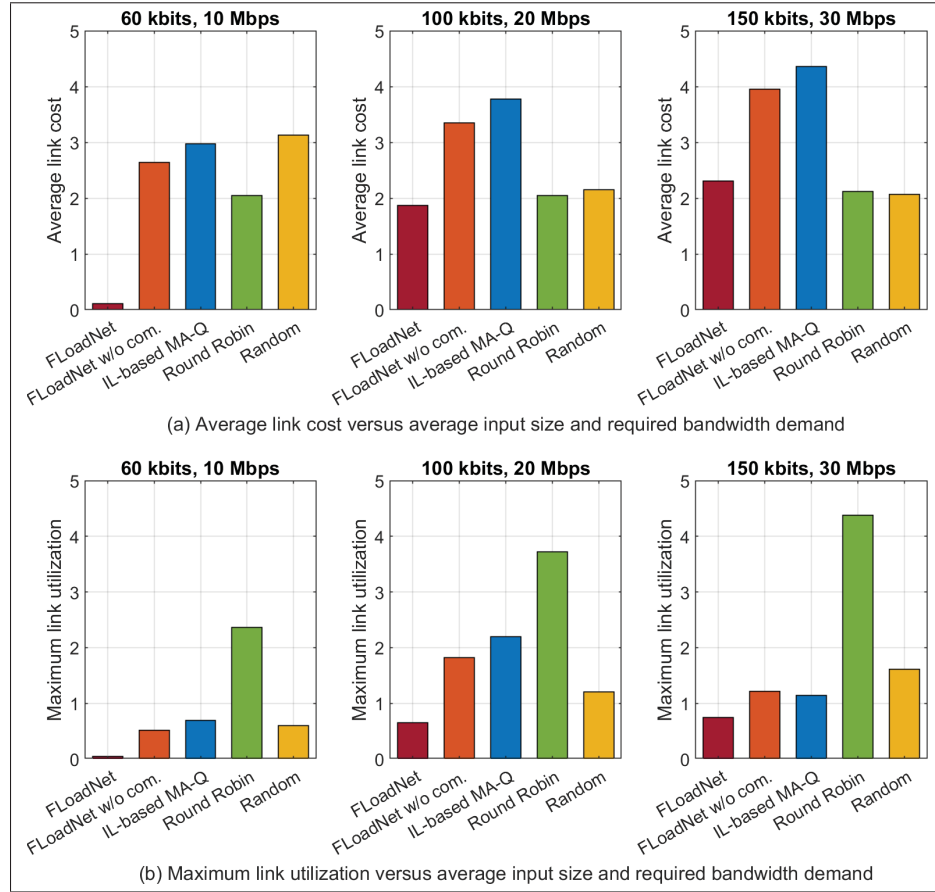


Figure 4.8 Average link cost and maximum link utilization rate across the learning procedure versus average resource demand of tasks

From Fig. 4.8 (b), only FLoadNet achieves the maximum link utilization under 1 over all the variations of resource demands. This indicates that FLoadNet can use real-time information to flexibly and automatically adapt to the high traffic demands at run time.

On the other hand, Figs. 4.9 (a) and (b) depict the average server buffer length and overflow ratio, respectively, with increasing bandwidth and computation demands. Here, it is shown that even if local computing does not require any bandwidth resources since the APs process all their tasks locally, the average buffer status is very large, which incurs a high number of task drops from overflowing.

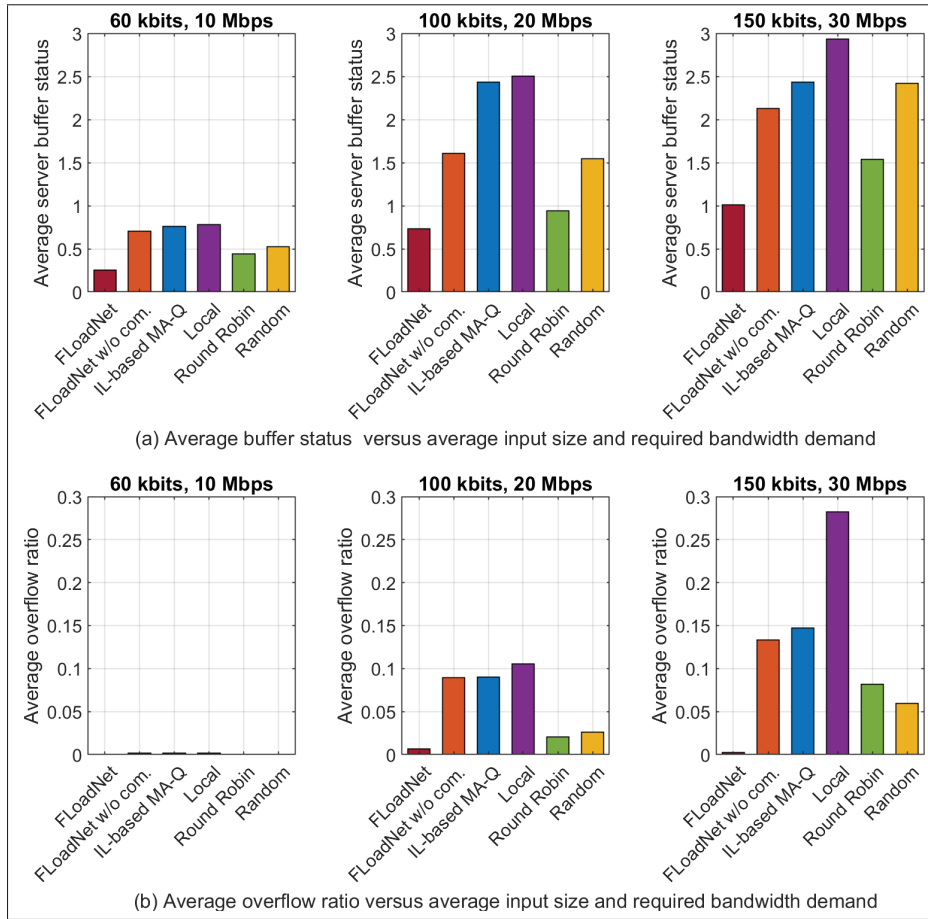


Figure 4.9 Average buffer length and average overflow ratio across the learning procedure versus average resource demand of tasks

For the FLoadNet with no communication and IL-based MA-Q models, the average performance on this experiment is observed to be worse than Round-Robin and Random policy as the resource demands increase. These empirical results show that the learning properties of FLoadNet allow multiple agents to estimate the partially observable environment better by communicating and aggregating experiences across different agents. Otherwise, making decisions from a misleading observation in no communication learning models can be arbitrarily wrong and yield even worse performance than non-learning-based static load balancing methods.

## 4.6 Conclusion

In this paper, a joint link and server load balancing problem, with multiple cooperative access points in a three-layer fog network was studied. First, the optimization problem was formulated as a stochastic game with the aim of minimizing the link cost, the maximum link utilization, the server buffer length, and the number of overflow tasks. To this end, an actor-critic reinforcement learning framework, called FLoadNet, was proposed to optimize the joint policy for load balancing. To advance the performance, centralized learning with parameter sharing and distributed execution was considered to approximate individual policies, value functions, and communication protocols between the access points. The experimental results showed that FLoadNet outperforms no communication learning models and other baseline load balancing methods. This work demonstrated that communication protocols aid the co-adaptation of the learning processes, which helps the access points distribute tasks while satisfying the objective.



## CHAPTER 5

### TOWARD 6G: A NEW ERA OF DISTRIBUTED INTELLIGENCE WITH MULTI-AGENT REINFORCEMENT LEARNING

Jungyeon Baek<sup>a</sup> and Georges Kaddoum<sup>a</sup>

<sup>a</sup> Department of Electrical Engineering, École de Technologie Supérieure,  
1100 Notre-Dame west, Montreal, Canada H3C 1K3.

Paper Submitted in *IEEE Communications Magazine*,  
December 2021, Under Review.

#### 5.1 Introduction

Over the past several decades, we have witnessed the wireless communications market continue to grow and wireless communications be used in a broad spectrum of applications. Following 3GPP Release 15, telecommunication operators around the world are still developing the fifth generation (5G), yet researchers are already looking ahead to the sixth generation (6G). The central theme of the 6G network will be the merging of digital and real worlds in all dimensions [Jiang, W., Han, B., Habibi, M. A. & Schotten, H. D. (2021)]. Sixth-generation networks are expected to deliver a high-quality experience through the seamless integration of distributed communication, computing, control, and artificial intelligence (AI). The growing use of AI and machine learning (ML) techniques in recent years has been attributed to a broad spectrum of wireless communication applications. Furthermore, a new computing paradigm, called edge computing, has been introduced to extend the computing, networking, and storage capabilities of the cloud to the edge of networks that are physically and logically close to end users. The emergence of AI and edge computing is expected to contribute significantly to future 6G networks and support the coordination of vertically and horizontally distributed intelligence [Peltonen, E., Bennis, M., Capobianco, M., Debbah, M., Ding, A., Gil-Castiñeira, F., Jurmu, M., Karvonen, T., Kelanti, M., Kliks, A. et al. (2020)]. A prime example is an industrial internet of things, which brings intelligence to connected sensors and edge devices to help improve product quality, productivity, and operational efficiency in real time. With the

emergence of new technologies and the continuous evolution of existing technologies, many unprecedented applications will become possible in 6G networks [Wu, J., Li, R., An, X., Peng, C., Liu, Z., Crowcroft, J. & Zhang, H. (2021)].

Traditionally, ML has been used by transferring all data collected from connected devices to a centralized server to train a generic model that is in turn deployed on the local devices. In other words, decision rules are defined a priori. Some types of ML algorithms that are based on a dataset of examples are classification, support vector machine, and many other supervised and unsupervised learning methods. However, all these algorithms become no longer viable as future training data is continuously altered according to the algorithms' decisions or the dynamic nature of the environments. One area in which this commonly occurs is wireless communication networks where the decision-makers are located in the nodes of a time-varying environment and actively change the kind of data they need in later parts of the training process. For these reasons, reinforcement learning (RL) has rapidly emerged in many wireless communication and networking domains. Human and animal behavior inspired the development of RL algorithms that interact with the environment to solve assigned tasks by sequentially taking actions based on observations. The learner or decision-maker is called "agent" in RL. At each time step, the agent observes some representation of the environment's state and selects an action on the basis of its observations. The agent receives a numerical reward in response to its action and observes a new state. The agent aims to find a policy that maximizes the expected sum of rewards received. The action taken determines the immediate reward and the transition probability of the following state, which in turn changes future rewards. Therefore, the policy is not a priori but instead has to be learned from experience.

Most of the progress that has been made in RL has been focused on settings in which a single agent learns an optimal policy in a particular environment and is operated by a centralized entity. This single-agent setting is not quite suitable for environments that contain many distributed nodes that need to make independent decisions based upon their local observations [Busoniu *et al.* (2008)]. Examples of this types of environment include autonomous vehicles, traffic control, smart factories, smart cities, and a broad range of mobile applications. Hence,



multi-agent RL (MARL) naturally become an appealing solution to deal with problem domains where multiple agents make individual decisions by distributing a comprehensive observation space across multiple local spaces [Panait, L. & Luke, S. (2005)]. As AI-based applications become increasingly pervasive in wireless networks, future AI agents will have to interact with other intelligent agents and humans. Thus, human-like cognitive skills and reasoning abilities should be the goal of *distributed intelligent* 6G systems. To this end, MARL will play a critical role in beyond-5G networks as a foothold for the development of human-level intelligence [Dafoe, A., Hughes, E., Bachrach, Y., Collins, T., McKee, K. R., Leibo, J. Z., Larson, K. & Graepel, T. (2020)].

This article introduces the general concept of multi-agent learning from a RL perspective and depicts several challenges that arise due to the presence of multiple learners in a shared environment, e.g., non-stationarity, coordination, credit assignment problems, that need to be addressed in the future. Furthermore, we provide a variety of MARL models as potential solutions for some of challenges outlined in the article. In addition, we suggest MARL models for 6G system architectures and analyze key drivers of distributed intelligence in 6G systems. More specifically, we conduct simulations to demonstrate how MARL can be utilized to optimize offloading decisions with multiple agents. We then describe key applications of MARL in wireless communications. Finally, the article concludes with core research questions for MARL development in future wireless networks.

## 5.2 Overview and Motivations

As connected objects become more intelligent to run our lives, many real-world problems arise concerning environments that include multiple decision-makers and are thus inherently multi-agent systems (MASs). In this view, future wireless communication networks will face a wide range of multi-agent learning problems. Multi-agent learning has been studied by various communities, including game theory, dynamic programming, RL, and heuristic methods. More specifically, RL solutions have become increasingly popular to find the optimal policy for incompletely known systems. Moreover, because the complexity of multi-agent learning

problems can make manual solutions extremely difficult and infeasible, deep reinforcement learning (DRL), which embraces neural networks to improve the learning speed and performance of RL algorithms, is widely used.

MARL is the extension of RL to MASs, where a common environment is influenced by the actions of multiple agents. Each agent implicitly or explicitly interacts with the other agents in its environment to take actions concurrently, and their actions jointly determine the next state of the environment. Unlike many ML algorithms that learn from data collected from past events, MARL addresses learning in an interactive environment where each agent learns from observations made in the moment and adjusts its policy in response to changes in the behavior of others. MARL has been shown to act as an accelerator for intelligent behavior among multiple decision-makers [Dafoe *et al.* (2020)] and will therefore play a fundamental role in enabling distributed intelligence in 6G systems.

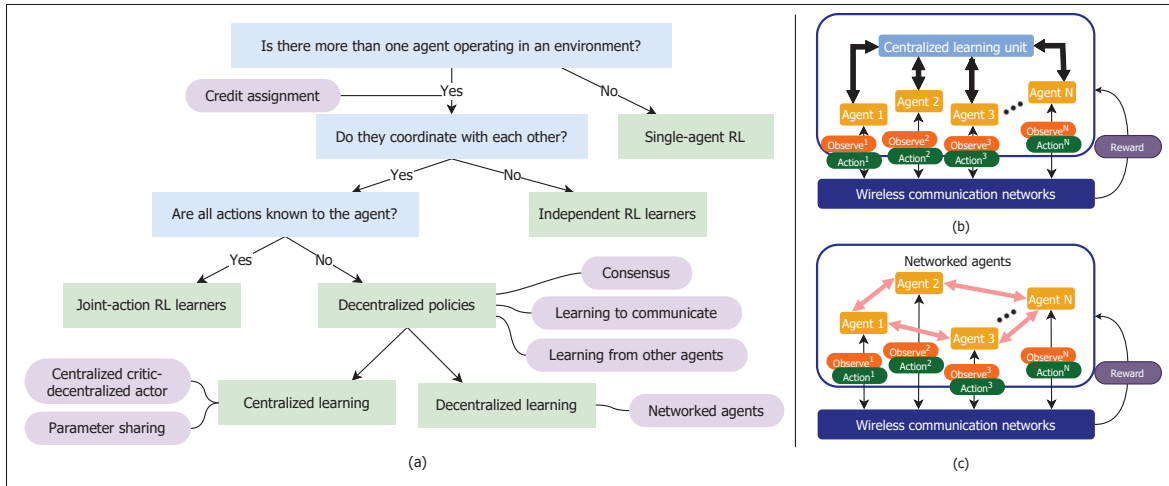


Figure 5.1 (a) A flowchart of MARL model settings  
 (b) MARL architecture for decentralized policies with a centralized control unit  
 (c) MARL architecture for decentralized policies with networked agents

The most straightforward MARL setting, called independent learners, applies single-agent RL techniques and ignores the existence of other agents. In contrast, joint-action learners attempt to learn the values of joint actions, the policies employed by other agents, or both. This joint-

action learning implies that each agent has all the action (and state) information of other agents. However, the joint state and action spaces of all agents grow exponentially with the number of agents, which makes joint-action learning settings infeasible in a large-scale MAS [Lisa, J. Yan and Nick, Cercone (2010)]. To address this challenge, many MARL problems have focused on decentralized policies, integrating independent learners with coordination methods. Decentralized policies seek an optimal solution under the constraint that each agent selects its own action based only upon its local observation. Coordination methods are critical to address the challenges associated with the resulting lack of information and allow the agents in a group to take account of the existence of other agents and understand others' points of view [Varshavskaya, P., Kaelbling, L. P. & Rus, D. (2009)]. A flowchart of MARL model settings and two different MARL architectures are shown in Fig. 5.1.

On the other hand, MARL deals with various problem domains, each of which caters to different settings to achieve a particular task or take place in a specific environment [Oroojlooyjadid & Hajinezhad (2019)]. Depending on the characteristics of the tasks, the agents can be cooperative or competitive with each other. Cooperative agents work towards a common goal, whereas competitive agents compete to achieve distinct goals. Many works involving multiple RL agents have achieved prominent outcomes in competitive settings, such as chess, poker, and Go. However, such settings of purely conflicting interests are exceptionally rare in wireless communication networks; the agents more commonly have mixed interests, both cooperative and competitive. For instance, in radio resource allocation problems, users have a common interest in reaching a maximum throughput but are in conflict due to scarce frequency resources and inter-user interference. Given the prevalence of mixed-interests settings and the challenges associated with MARL in wireless communications, building AI agents that are capable of cooperating in that context will be a critical goal. Whereas most RL research has focused on improving the individual intelligence of algorithms and agents, now is the time to develop cooperative MARL where all agents effectively cooperate to solve the problems they face.

### 5.3 Case Study: Multiple Agents with Coordination for Resource Management

This case study presents MARL models for 6G system architectures in which base stations (BSs) provide both traditional cellular service and multi-access edge computing (MEC) service to mobile users (MUs). The intelligent management of computational task offloading between resource-limited mobile devices and BSs in addition to the dynamic allocation of cellular packets will be even more critical in 6G networks. Here, MARL models are based on distributed policy learning at the MUs to address the challenges involved in providing low latency, real-time reactivity, and scalability. Consequently, numerical experiments are conducted using different MARL-based offloading algorithms, and the importance of coordination methods to train decentralized policies in MARL is discussed.

**Experiment Settings:** We consider a cellular network composed of 4 BSs and 10 MUs in a  $2 \times 2 \text{ km}^2$  area. The BSs are placed equidistance from one another, and each one has a limited number of orthogonal channels with the same bandwidth, 500 MHz. The MUs move within the service area following a random waypoint model. This mobility model is widely used in the literature [Mao, S. (2010)]. The average channel gain experienced by each MU is determined by the distance between the MU and the selected BS. Each MU maintains a queue to buffer packets for traditional cellular service until transmission. We assume that packet arrivals follow a Poisson process with an average rate of five packets/slot and the packet size is  $5 \times 10^3$  bits. Meanwhile, each MU independently and randomly generates computation tasks at every time slot. The computation tasks can be processed locally on the MU's end or be offloaded to one of the BSs. The input size and the number of CPU cycles required to complete one bit of the computation task are  $3 \times 10^3$  bits and 700 cycles/bit, respectively. At the beginning of each slot, each MU makes joint decision; 1) which BS to select, 2) how many packets will be transmitted to the BS selected, and 3) how many computation tasks will be offloaded to the BS. Each MU aims to solve an optimal decision-making problem that minimizes the power required to transmit the packets and computation tasks and perform the computation tasks, the number of packets queued, and the number of packets dropped due to queue overflows [Chen *et al.* (2019)]. The challenge when it comes to finding the optimal decision arises from the

randomness associated with the movement of MUs and the stochastic nature of communication channels. Furthermore, each MU learns its own control policy from its local observations. Since the MUs that select the same BS share the BS's limited number of channels, MUs need to cooperate effectively. We leverage MARL algorithms to learn the optimal policies and consider each MU as an agent. For the DRL algorithm, we take an actor-critic approach. The actor learns the policy to control the agent's behavior, and the critic estimates a value function to measure the actions taken by the actor. We design both the actor and the critic with one hidden layer of 128 neurons. We select a rectified linear unit as the activation function and Adam as the optimizer.

**Analysis:** Figure 5.2 presents the convergence of the globally averaged returns of different decentralized MARL models for our problem. We perform two different reward functions for the MARL models. In Fig. 5.2(a), each MU receives the same global reward obtained at a team level in response to joint actions, while in Fig. 5.2(b), each MU receives a local reward determined solely based on its individual action. The results show that if there is no coordination among users ("Independent"), the average return obtained from the local reward does not differ significantly from the average one obtained from the global reward but shows unstable results. However, once the "Central" and "Networked2" models start learning their networks, the average return obtained from the local reward is higher than the one obtained from the global reward. These results suggest that coordination methods are crucial when using the local reward function to train decentralized policies. While the local reward function better evaluates the performance of each MU's policy, the coordination methods help to discourage selfish behaviors among MUs. On the other hand, the "Networked1" model underperforms compared to the "Independent" model, which is expected as communicating with only one neighbor does not give MUs enough cooperation advantages in our mixed-interests setting.

Here, the coordination overhead indicates the size of the memory of the object sent to other learning systems at each learning step. In the "Central" model, this is a sum of the observation history transferred from each MU to the centralized critic and the state value sent from the central unit to the MUs. In the "Networked" model, the parameters of the decentralized critic



Figure 5.2 Convergence of MARL algorithms

The “Independent”, “Central”, and “Networked” MARL methods are performed for the learning architecture. The “Central” method uses a centralized critic at a central unit that is accessed by all MUs, while each MU has a distributed actor for its local policy [Foerster, J., Farquhar, G., Afouras, T., Nardelli, N. & Whiteson, S. (2018)]. In the “Networked” method, each MU performs fully decentralized learning based on its local observation and the information it receives from its neighbors [Zhang, K., Yang, Z., Liu, H., Zhang, T. & Basar, T. (2018b)]. Here, MUs choose to communicate with one (“Networked1”) or two (“Networked2”) neighboring MUs in the communication range (500 m) and exchange their parameters for critic networks.

at each MU are shared with the neighbor(s). Although the coordination methods require more overhead to improve performance, finding the right balance between the overhead and the reliability that can be achieved by sharing intelligence with other learning systems is still a problem to be discussed.

#### 5.4 Applications of MARL for Distributed Intelligence in 6G Networks

As the location of AI models plays a huge role in meeting the stringent 6G requirements, such as real-time decision-making and adaptability to dynamic environments, distributed intelligence system development will evolve over the next decade. In this context, we elaborate three applications of MARL in wireless communication systems, as shown in Fig. 5.3.




	Agents	Observation	Action	Reward
 Edge computing	Edge, fog, cloud servers, data centers	States of the computation buffer, the number of active virtual machines (VMs), CPU usage/capacity, the number of requests for content	Computational resource allocation, task offloading, load balancing, VM allocation, VM migration, caching	Processing delay, buffer overflow ratio, utilization rate, service cost, CPU overload probability, popularity of content stored in cache
 Cognitive radio	Wireless nodes, radio units, base stations, cellular devices	Channel state information, idle spectrum band, SNR, path loss, the set of neighboring nodes, maximum terminal output power, link failures	Communication resource allocation, power allocation, multi-user scheduling policies, overlay/underlay	Network throughput, transmission delay, spectral efficiency, transmission power consumption, error probability
 Autonomous driving	Vehicles, traffic lights, pedestrians, parking lots	Location of vehicles, image of surroundings, direction and speed of movement, the set of neighboring vehicles, traffic signal states	Brake or maintain or accelerate, adjustment of the direction, steering of the wheels, change of lanes	Collision avoidance, driving efficiency (driving speed, arrival time), keeping in the lane, gas/battery consumption

Figure 5.3 Elements of three key applications for physical, network, and application layers in wireless communication systems

#### 5.4.1 Autonomous Driving

Autonomous driving is a broad research area covering applications from driver assistance systems to self-driving vehicles that operate without human intervention. Despite the high-speed data transfer rates and connectivity benefits of 5G, self-driving vehicles are not on our roads yet. A key challenge in designing a fully autonomous driving system is building vehicles that are capable of performing as autonomous agents with cognitive capabilities like humans [Dafoe *et al.* (2020)]. ML models rely on static information downloaded from data sources that are effective in well-defined scenarios, such as driving along a well-paved road on a clear day, but may yield undesirable outcomes in poorly defined scenarios, such as driving along unpaved or snowy roads. Challenges associated with the development of autonomous agents in dynamic and open environments have been a continuing concern in the field of MARL. The primary objective is to develop autonomous vehicles that are capable of coexisting and cooperating with other vehicles and humans in the real world while continuously acquiring knowledge to adapt quickly to network changes. Because the goal of MARL is to learn a policy for each agent through interaction with other agents so that all agents together attain the system's goal, MARL is a promising tool for developing the human-level cognitive systems necessary for self-driving vehicles. MARL would benefit from improved communication and network technology enablers, such as software-defined networking, MEC, fog computing, and IoT gateways, to address flexible implementations and maximize collaboration between agents.



### 5.4.2 Cognitive Radio Network

IoT devices undoubtedly play a central role in 5G networks. It is foreseeable that 6G networks will be hyper-connected networks consisting of pervasive smart devices, and communication among these devices will be the primary traffic type in future wireless cellular networks. However, the communication spectrum has insufficient availability to accommodate the continuous growth of IoT devices. Thus, beyond-5G networks need to increase spectrum use by leveraging the under-utilized frequency band for future devices. The cognitive radio network (CRN) is emerging to increase spectrum use by detecting under-utilized spectrum bands, re-configuring system parameters, and allowing dynamic spectrum access based on the surrounding environment. Related works on CRNs focus mainly on a centralized architecture that groups together multiple cognitive radio units (CRUs) and allows one control unit to reduce the overhead and computational cost. Although methods that attempt to share the spectrum in a distributed manner have been studied, their CRUs simply follow a greedy approach to maximize individual capacity or have no regard for coordination between them. This opens up a number of challenges that must be addressed to deploy intelligent systems that automatically adapt to dynamic networks and allocate radio resources to where they are most needed. We believe that this challenging design target will be significantly addressed by MARL. The main motive is to mesh the objective of MARL with the definition of a CRN framework. From its definition in [Haykin, S. (2005)], cognitive radio is aware of its surrounding environment, learns from its environment, and adapts its internal states by making corresponding changes to specific operational parameters, e.g., frequency band, transmission power, modulation schemes, and routing path. Similarly, MARL algorithms enable agents to learn by interacting with their environment through observation-action loops. MARL agents train continuously on new samples (observations, actions, and rewards) in an online manner and can thus respond to their evolving environment by changing their behavior. Decentralized MARL models in particular have been explored to limit interactions between agents by enforcing conditional independencies such as observation independence among agents. Therefore, distributed MARL can be an efficient and scalable solution for CRNs to perform computation and limit the amount of communication.



### 5.4.3 Edge Computing

The unprecedented amount of traffic data drives academia and industry to invest in developing an intelligent edge computing network. Pushing learning abilities toward the edge is motivated by making it possible for edge devices to quickly train AI models, which empowers intelligent devices to solve real-time policies, such as scheduling, computational task offloading, load balancing, and routing [Peltonen *et al.* (2020)]. Edge intelligence will be an essential functionality to guarantee the efficiency of 6G networks. Although edge computing has versatile potential applications due to its diverse range of features, MARL can allow edge computing systems to speed up their support of mission-critical applications in various ways. First, MARL is needed to make edge computing faster, more efficient, and more scalable. Edge computing aims to integrate a growing number of edge devices, nodes, and servers to process the data generated nearby, and multi-agent learning strives to resolve complex problems in decentralized ways by using autonomous agents. While a few applications may work with centralized AI models and deploy pre-trained models at the edge nodes, most mission-critical applications demand distributed learning algorithms that are subject to delay constraints, limited computational capacities, and communication bottlenecks between the edges. With the increasing capability of intelligence at the edge, it is possible to bring AI features to each edge node, which becomes a learning agent, to learn to understand the goals of target services. Furthermore, when dealing with distributed learning algorithms, it is essential to minimize dependencies between learning agents to guarantee efficient, robust, secure, and resilient network functionalities. From this perspective, a centralized component that has sole authority over all systems is not very suitable. MARL can address these problems in which distributed agents need to learn how to behave independently and cooperate with other agents to effectively coordinate their decisions towards a common goal. In addition, MARL can address the challenges associated with edge devices having access to limited data by enabling dynamic decision-making based on only the knowledge available from each device.

## **5.5 Open challenges and future works**

Despite the advances made in MARL methods [Oroojlooyjadid & Hajinezhad (2019)], several challenges remain to be addressed and are core research questions to consider in future 6G applications.

### **5.5.1 Compact and heterogeneous MARL model**

From a wireless communication perspective, on-device hardware limitations and insufficient communication channels among devices become more critical when developing MARL algorithms. The agents in a large-scale MAS range from devices with limited storage capacity and batteries (e.g., mobile and IoT devices) to higher-performance objects (e.g., BSs, vehicles, and cloud servers). While transforming high-dimensional neural networks into RL models and frequent coordination among agents can help make cognitive decisions in complex MAS environments, doing so could be infeasible due to energy limitations, hardware failures, and geographical limitations. These constraints demand low-complexity, low-capacity, and energy-efficient MARL model designs. Moreover, since agents have heterogeneous capacities and abilities, how different agents can utilize the other agents' capacities to learn a more efficient policy is an open question.

### **5.5.2 The combination of RL and other AI techniques**

Potential applications of MARL require real-time execution while ensuring reasonable performance to guarantee reliability constraints are met and avoid catastrophic situations during training. Despite the numerous successes of model-free RL methods, a typical limitation is sample efficiency because they rely on actual samples, and a large number of samples is needed to obtain adequate performance. To address the challenge of developing ultra-reliable low-latency communication systems, generative adversarial networks can be used to pre-train the DRL framework using a mix of real and synthetic data to gain experience faster, especially in the case of sporadic events. Furthermore, transfer learning, which is a technique that utilizes

external knowledge to accelerate the learning process of the target task, can apply to MARL architectures in which homogeneous agents run the same learning model or share common goals. Federated learning can also be combined with MARL centralized learning models in which agents share one network's parameters and perform the federated updating that enables high-quality updates with fewer iterations.

### **5.5.3 Reward specification and credit assignment**

One of the most critical components of RL is the reward function. While in some problems (e.g., games), the reward is evident, in many wireless applications, where the problems are often continuous and involve multi-objective optimization, formulating the reward function is burdensome and may lead to undesirable results. In addition, one inherent problem when dealing with multiple agents is how to assign credit to each agent after the reward is received based upon their joint actions, which is termed the credit assignment problem in MARL research. The problem deals with assessing the contribution of individual agents to the overall reward without encouraging selfish behavior. As highlighted above, mixed interests between agents in wireless communications should also be considered when studying the credit assignment problem.

## **5.6 Conclusion**

MARL is a framework that addresses learning in an interactive environment with multiple distributed agents. This article aims to introduce MARL's importance for AI-pervasive future networks and key applications of MARL in the area of wireless communications. Given that network dynamics incorporate uncertainties and non-stationarity is at play when there are multiple learning agents, building AI agents that are capable of adapting to network changes and cooperating with other AI agents will be a critical goal for future networks. With the integration of MARL frameworks and coordination methods, the agents can effectively cooperate to solve shared problems under the system constraints. Finally, we see an opportunity to increase MARL's applicability to problems in 6G networks.



## CONCLUSION AND RECOMMENDATIONS

### 6.1 Conclusion and Lessons that we learned

Over the past decade, wireless technology has had an incredible impact on society. According to the Ericsson Mobility Report [Cerwall, P., Lundvall, A., Jonsson, P., Carson, S., Möller, R., Saksena, R., Lu, D. & Celik, I. (2021)], there has been an almost 300-fold increase in mobile data traffic since 2011 due to the explosive progress of smart devices. Moreover, massive IoT deployments are forecast to account for 51% of all cellular connections by 2027. Meanwhile, performance requirements for wireless network applications are becoming increasingly stringent, with potential speeds of up to 10 gigabits per second, increased network capacity, and ultra-low latency in the millisecond range. In this context, fog computing has emerged to overcome the challenges of traditional cloud computing architectures where the connections between remote data centers and end devices cause unpredictably high latencies.

Nowadays, fog computing is one of the key technologies that is enabling 5G networks to support low-latency services and IoT applications, such as smart homes, industrial automation, and autonomous vehicles. Indeed, numerous new computing-intensive applications and AI-powered devices are driving the development of fog computing in the next-generation of wireless systems, i.e., beyond 5G and 6G. In particular, this thesis targeted the design of an intelligent fog computing platform that implements efficient resource management strategies in a complex environment with heterogeneous resources and volatile traffic demands. In this vein, this thesis proposed promising distributed computing approaches for future fog computing networks. Specifically, the contributions of the thesis are summarized as follows:

- In Chapter 2, we proposed a joint heterogeneous task offloading and resource allocation scheme for SDN-based fog networks and designed a DQL-based algorithm where multiple fog nodes aim to maximize the processing tasks successfully completed within their delay

constraints. Furthermore, we leveraged RNN to tackle the partial-observability from the limited information and evaluated the performance of the proposed scheme against different neural networks. Numerical results show that the proposed DRQN-based algorithm can achieve a higher average success rate and lower average overflow than baseline methods.

- In Chapter 3, an online partial offloading and task scheduling problem was analyzed to minimize the energy consumption. We considered two types of tasks, namely, offloadable and non-offloadable tasks to investigate the characteristics for offloading and optimizing their allocation. In addition, we implemented DRQN-based algorithms to deal with the vast state space and partial-observability. It was demonstrated that the proposed DRQN-based method requires comparatively less computational complexity than the conventional Q-learning algorithm and can effectively deal with both transmission and CPU energy consumptions while guaranteeing convergence in a limited time.
- Chapter 4 provided a novel offloading problem for network link and server levels load balancing in a combined edge-fog-cloud environment. We formulated the optimization problem as a stochastic game with multiple cooperative APs and proposed an actor-critic framework that consists of a centralized critic and distributed actor networks in all the APs. We further extended the critic network to learn to communicate among APs while evaluating the value function. The experimental results showed that the proposed algorithm outperforms baseline load balancing methods.
- Chapter 5 presented an introduction to multi-agent learning from an RL perspective and suggested MARL models for 6G system architectures. In addition, we discussed key enablers for distributed intelligence in 6G and potential applications of MARL in 6G networks. Particularly, we conducted a case study that presents MARL-based offloading models with different coordination methods in future 6G architectures.

As a summary, according to the conducted research, valuable lessons we learned are listed as follows:

- Computational and system hierarchies are essential for fog architecture to support effective QoS management. Additionally, fog architecture modules, such as network slicing and priority-based buffers are required to ensure the QoS requirements of each application.
- RL approaches are especially promising in fog computing architectures where edge and fog nodes have access to limited data and need to make dynamic decisions solely based on the local information.
- The performance of RL algorithms can be enhanced by leveraging advanced neural networks, e.g., RNN, to deal with complex dynamical systems, such as fog computing architectures.
- The communication between intelligent nodes is crucial to coordinate the behavior of each individual in non-stationary environments due to the presence of multiple learners.

### **Author's Publications**

The outcomes of the author's PhD research are the articles listed below, published or submitted to IEEE journals.

J. Y. Baek and G. Kaddoum (2021), Heterogeneous Task Offloading and Resource Allocations via Deep Recurrent Reinforcement Learning in Partial Observable Multi-Fog Networks, *IEEE Internet of Things Journal*, 8(2), 1041-1056.

J. Y. Baek and G. Kaddoum (2021), Online Partial Offloading and Task Scheduling in SDN-Fog Networks with Deep Recurrent Q-Network, *IEEE Internet of Things Journal*, early access.

J. Y. Baek and G. Kaddoum (2021), FloadNet: Load Balancing in Fog Networks with Cooper-

ative Multi-Agent using Actor-Critic Method, *IEEE Transactions on Network and Service Management*, under review.

J. Y. Baek and G. Kaddoum (2021), Multi-agent Reinforcement Learning with Coordination for Wireless Communications: Motivations, Applications, and Challenges, *IEEE Communications Magazine*, under review.

Beside the above articles that contribute to the main contents of this dissertation, other publications that the author was involved in and which are not included in this dissertation are

C. Miranda, G. Kaddoum, and J. Y. Baek, and B. Selim (2021), Task Allocation Framework for Software-Defined Fog v-RAN, *IEEE Internet of Things Journal*, 8(18), 14187-14201.

J. Y. Baek, G. Kaddoum, S. Garg, K. Kaur, and V. Gravel (2019), Managing Fog Networks using Reinforcement Learning Based Load Balancing Algorithm, *IEEE Wireless Communications and Networking Conference (WCNC)*, Marrakesh, Morocco

## **6.2 Recommendations: Fog computing and resource management for 6G networks**

According to [Xiao, Y., Shi, G., Li, Y., Saad, W. & Poor, H. V. (2020)], 6G networks will be a new vision of ubiquitous AI architecture that brings human-level intelligence into every aspect of communication and networking systems. Given the requirements of this challenging design target, it is expected that future 6G networks will be highly distributed intelligent architectures, combining seamless integration of AI, advanced communication technologies, and edge computing infrastructures. Hence, some potential directions for future research are discussed below.

### **6.2.1 Fog computing-based WSN systems**

With the rapid acceleration of smart sensors, wireless sensor networks have been receiving increasing attention in the field of wireless communication research. One of the major challenges



for WSNs stems from the limited capacity of sensor nodes for computing and data storage and access. Moreover, due to the limitations of the WSN communication range, the sensing data from devices cannot be transmitted over long distances, where various communication protocols further complicate the connection between sensors and the internet. To overcome these limitations, fog computing can play a key role in providing specialized functionalities to smart gateways [Sun, Z., Wei, L., Xu, C., Wang, T., Nie, Y., Xing, X. & Lu, J. (2019e)]. A fog-based sensor gateway can provide sensor data aggregation, flexible query management between sensors and the web, and efficient resource access among crowded devices.

### **6.2.2 Fog computing for massive MIMO beamforming**

To ensure that every user gets a high-quality experience in future 6G networks, the RAN processing power will need to be able to execute advanced radio frequency functions for more users and devices, more antenna branches, and more frequency bands [Sim, M. S., Lim, Y.-G., Park, S. H., Dai, L. & Chae, C.-B. (2020)]. In particular, an exponential increase in processing demand will be required in Layer 1 and 2 for 5G with massive multi-input multi-output (MIMO) beamforming [Long, Y., Chen, Z., Fang, J. & Tellambura, C. (2018)]. In this context, a fog computing-based radio access network (F-RAN) is a promising solution to address this big processing challenge. F-RAN can provide lower layer RAN functions over a generic computing platform instead of a purpose-built hardware platform, and manage the RAN application virtualization. By performing radio frequency functions at distributed fog-based radio units (i.e., RRHs), the amount of front-haul overhead and the overall complexity can be greatly reduced while satisfying the latency constraints.

### **6.2.3 Generative adversarial network (GAN) for resource managements in fog computing networks**

A GAN is a class of unsupervised learning that formulates a game between two opponents: a generator and a discriminator [Goodfellow, I. J. (2017)]. The core idea of a GAN is based on the discriminator's prediction on synthetic data produced by the generator from the distribution of the real data. The goal of the generator is not to minimize the distance to a specific labeled training data, but to increase the error rate of the discriminator. GANs have been proven useful for predicting possible futures from time-series data and thereby could be used for planning RL algorithms. For instance, the combination of GAN and RL enables learning a conditional distribution over future states of the environment, given the current state and hypothetical actions that an agent might take. In summary, GANs are a promising solution to cope with the statistical properties of wireless channels and the uncertainties associated with communication environments in real-time resource management applications.

### **6.2.4 Digital twin for reliable and efficient fog computing networks**

The various types of smart devices with heterogeneous capacities and diverse applications with different resource demands pose significant challenges on the implementation of intelligent fog computing applications. In this context, a digital twin is a promising technology, which brings state abstractions of physical devices into mirrored virtual spaces [Zhang, K., Cao, J. & Zhang, Y. (2021)]. This state information is used to track the evolving behavior of individual devices and to analyze incoming events. The digital twin simplifies the re-design and re-engineering process by enabling thorough self-examination of the evolving status of devices and providing automatic correlation of incoming events for each device. Thus, merging digital twin and AI can provide comprehensive and accurate system information, which is needed to design reliable learning processes. More specifically, the digital twin can demonstrate potential coop-

eration for multi-agent learning between smart devices and reduce the complexity of service management required to organize hierarchical fog computing networks.



## LIST OF REFERENCES

- A. Zaidi, Y. Hussain, M. Hogan, and C. Kuhlins,. (2019). *Cellular IoT evolution for industry digitalization*. Consulted at <https://www.ericsson.com/en/reports-andpapers/white-papers/cellular-iot-evolution-for-industry-digitalization>.
- Aazam, M., Zeadally, S. & Harras, K. A. (2018). Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities. *Future Generation Computer Systems*, 87, 278–289.
- Ahmed, E. & Rehmani, M. H. (2017). Mobile edge computing: opportunities, solutions, and challenges. Elsevier.
- Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys Tutorials*, 17(4), 2347-2376. doi: 10.1109/COMST.2015.2444095.
- Alameddine, H. A., Sharafeddine, S., Sebbah, S., Ayoubi, S. & Assi, C. (2019). Dynamic Task Offloading and Scheduling for Low-Latency IoT Services in Multi-Access Edge Computing. *IEEE Journal on Selected Areas in Communications*, 37(3), 668-682. doi: 10.1109/JSAC.2019.2894306.
- Aslam, S. & Shah, M. A. (2015). Load balancing algorithms in cloud computing: A survey of modern techniques. *2015 National software engineering conference (NSEC)*, pp. 30–35.
- Aujla, G. S., Chaudhary, R., Kumar, N., Rodrigues, J. J. & Vinel, A. (2017). Data offloading in 5g-enabled software-defined vehicular networks: A stackelberg-game-based approach. *IEEE Communications Magazine*, 55(8), 100–108.
- Baek, J. & Kaddoum, G. (2021). Heterogeneous Task Offloading and Resource Allocations via Deep Recurrent Reinforcement Learning in Partial Observable Multifog Networks. *IEEE Internet of Things Journal*, 8(2), 1041-1056. doi: 10.1109/JIOT.2020.3009540.
- Baek, J., Kaddoum, G., Garg, S., Kaur, K. & Gravel, V. (2019). Managing Fog Networks using Reinforcement Learning Based Load Balancing Algorithm. *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1-7. doi: 10.1109/WCNC.2019.8885745.
- Bastug, E., Bennis, M. & Debbah, M. (2014). Living on the edge: The role of proactive caching in 5G wireless networks. *IEEE Communications Magazine*, 52(8), 82–89.
- Bellavista, P., Giannelli, C. & Montenero, D. D. P. (2020). A Reference Model and Prototype Implementation for SDN-Based Multi Layer Routing in Fog Environments. *IEEE Transactions on Network and Service Management*, 17(3), 1460-1473. doi: 10.1109/TNSM.2020.2995903.

- Bellman, R. & Karush, R. (1964). *Dynamic Programming: A Bibliography of Theory and Application*. Rand Corporation. Consulted at <https://books.google.ca/books?id=zHG1AQAACAAJ>.
- Bellman, R. (1957). *Dynamic Programming* (ed. 1). Princeton, NJ, USA: Princeton University Press. Consulted at <http://books.google.com/books?id=fyVtp3EMxasC&pg=PR5&dq=dynamic+programming+richard+e+bellman&client=firefox-a#v=onepage&q=dynamic%20programming%20richard%20e%20bellman&f=false>.
- Benzekki, K., El Fergougui, A. & Elbelrhiti Elalaoui, A. (2016). Software-defined networking (SDN): a survey. *Security and communication networks*, 9(18), 5803–5833.
- Bertsekas, D. & Tsitsiklis, J. (1996). *Neuro-dynamic Programming*. Athena Scientific. Consulted at <https://books.google.ca/books?id=WxCCQgAACAAJ>.
- Bilal, K., Khalid, O., Erbad, A. & Khan, S. U. (2018). Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Computer Networks*, 130, 94–120.
- Bonomi, F., Milito, R., Zhu, J. & Addepalli, S. (2012). Fog computing and its role in the internet of things. *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16.
- Busoniu, L., Babuska, R. & Schutter, B. D. (2010). Multi-agent reinforcement learning: An overview. In Srinivasan, D. & Jain, L. (Eds.), *Innovations in Multi-Agent Systems and Applications – I* (vol. 310, ch. 7, pp. 183–221). Berlin, Germany: Springer. doi: 10.1007/978-3-642-14435-6\_7.
- Busoniu, L., Babuska, R. & De Schutter, B. (2008). A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2), 156–172. doi: 10.1109/TSMCC.2007.913919.
- Buyya, R. & Dastjerdi, A. (2016). *Internet of Things: Principles and Paradigms*. Elsevier Science. Consulted at <https://books.google.ca/books?id=dbvajwEACAAJ>.
- Cao, B., Sun, Z., Zhang, J. & Gu, Y. (2021a). Resource Allocation in 5G IoV Architecture Based on SDN and Fog-Cloud Computing. *IEEE Transactions on Intelligent Transportation Systems*, 22(6), 3832–3840. doi: 10.1109/TITS.2020.3048844.
- Cao, H., Aujla, G. S., Garg, S., Kaddoum, G. & Yang, L. (2021b). Embedding Security Awareness for Virtual Resource Allocation in 5G Hetnets Using Reinforcement Learning. *IEEE Communications Standards Magazine*, 5(2), 20–27. doi: 10.1109/MCOMSTD.001.2000026.
- Cerwall, P., Lundvall, A., Jonsson, P., Carson, S., Möller, R., Saksena, R., Lu, D. & Celik, I. (2021). Ericsson mobility report. *Ericsson: Stockholm, Sweden*.

- Chamola, V., Tham, C.-K. & Chalapathi, G. S. (2017). Latency aware mobile task assignment and load balancing for edge cloudlets. *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 587–592.
- Chekired, D. A., Khoukhi, L. & Mouftah, H. T. (2018). Industrial IoT Data Scheduling Based on Hierarchical Fog Computing: A Key for Enabling Smart Factory. *IEEE Transactions on Industrial Informatics*, 14(10), 4590–4602. doi: 10.1109/TII.2018.2843802.
- Chen, X., Zhang, H., Wu, C., Mao, S., Ji, Y. & Bennis, M. (2018). Performance optimization in mobile-edge computing via deep reinforcement learning. *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pp. 1–6.
- Chen, X., Zhao, Z., Wu, C., Bennis, M., Liu, H., Ji, Y. & Zhang, H. (2019). Multi-Tenant Cross-Slice Resource Orchestration: A Deep Reinforcement Learning Approach. *IEEE Journal on Selected Areas in Communications*, 37(10), 2377–2392. doi: 10.1109/JSAC.2019.2933893.
- Chen, X., Jiao, L., Li, W. & Fu, X. (2015). Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5), 2795–2808.
- Cheng, M., Li, J. & Nazarian, S. (2018). DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. *2018 23rd Asia and South pacific design automation conference (ASP-DAC)*, pp. 129–134.
- Cheng, X., Dale, C. & Liu, J. (2008). Statistics and Social Network of YouTube Videos. *2008 16th International Workshop on Quality of Service*, 229–238.
- Chiang, M. & Zhang, T. (2016). Fog and IoT: An Overview of Research Opportunities. *IEEE Internet of Things Journal*, 3(6), 854–864. doi: 10.1109/JIOT.2016.2584538.
- Chung, J., Gülçehre, Ç., Cho, K. & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR*, abs/1412.3555. Consulted at <http://arxiv.org/abs/1412.3555>.
- Cisco. (2018). *Cisco Annual Internet Report (2018–2023) White Paper*. Consulted at <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>.
- Claus, C. & Boutilier, C. (1998). The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, (AAAI '98/IAAI '98), 746–752.

- Cui, J., Liu, Y. & Nallanathan, A. (2020). Multi-Agent Reinforcement Learning-Based Resource Allocation for UAV Networks. *IEEE Transactions on Wireless Communications*, 19, 729-743.
- D. Gupta and R. Jahan. (2014). *Inter-SDN controller communication: Using border gateway protocol*. Consulted at [https://www.researchgate.net/publication/324452059\\_InterSDN\\_controller\\_communication\\_using\\_border\\_gateway\\_protocol](https://www.researchgate.net/publication/324452059_InterSDN_controller_communication_using_border_gateway_protocol).
- Dafoe, A., Hughes, E., Bachrach, Y., Collins, T., McKee, K. R., Leibo, J. Z., Larson, K. & Graepel, T. (2020). Open problems in cooperative AI. *arXiv preprint arXiv:2012.08630*.
- Deng, L., Hinton, G. & Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: an overview. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8599-8603. doi: 10.1109/ICASSP.2013.6639344.
- Doan, K. N., Vaezi, M., Shin, W., Poor, H. V., Shin, H. & Quek, T. Q. S. (2020). Power Allocation in Cache-Aided NOMA Systems: Optimization and Deep Reinforcement Learning Approaches. *IEEE Transactions on Communications*, 68(1), 630-644. doi: 10.1109/TCOMM.2019.2947418.
- Elsharkawey, M. A. & Refaat, H. E. (2018). Mlrts: multi-level real-time scheduling algorithm for load balancing in fog computing environment. *International Journal of Modern Education and Computer Science*, 11(2), 1.
- ETSI. (2018). *Multi-access Edge Computing (MEC); Study on MEC Support for V2X Use Cases* (Report n°V2.1.1).
- Fan, Q. & Ansari, N. (2018). Towards workload balancing in fog computing empowered IoT. *IEEE Transactions on Network Science and Engineering*, 7(1), 253-262.
- Foerster, J., Farquhar, G., Afouras, T., Nardelli, N. & Whiteson, S. (2018). Counterfactual multi-agent policy gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).
- Foerster, J. N., Assael, Y. M., de Freitas, N. & Whiteson, S. (2016). Learning to Communicate with Deep Multi-Agent Reinforcement Learning. *CoRR*, abs/1605.06676. Consulted at <http://arxiv.org/abs/1605.06676>.
- Fudenberg, D. & Kreps, D. (1990). *Lectures on Learning and Equilibrium in Strategic-Form Games*.



- Gao, Y., Wang, Y., Gupta, S. K. & Pedram, M. (2013). An energy and deadline aware resource provisioning, scheduling and optimization framework for cloud systems. *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 1-10. doi: 10.1109/CODES-ISSS.2013.6659018.
- Garg, S., Kaur, K., Kaddoum, G. & Guo, S. (2021). SDN-NFV-Aided Edge-Cloud Interplay for 5G-Envisioned Energy Internet Ecosystem. *IEEE Network*, 35(1), 356-364. doi: 10.1109/M-NET.011.1900602.
- Ghobaei-Arani, M., Souri, A. & Rahmanian, A. (2020). Resource Management Approaches in Fog Computing: a Comprehensive Review. *Journal of Grid Computing*, 18. doi: 10.1007/s10723-019-09491-1.
- Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep Learning*. MIT Press. Consulted at <https://books.google.ca/books?id=Np9SDQAAQBAJ>.
- Goodfellow, I. J. (2017). NIPS 2016 Tutorial: Generative Adversarial Networks. *CoRR*, abs/1701.00160. Consulted at <http://arxiv.org/abs/1701.00160>.
- Graziani, R. & Vachon, B. (2014). *Cisco Networking Academy: Connecting Networks Companion Guide*. Cisco Press.
- Ha, K. & Satyanarayanan, M. (2015). Openstack++ for cloudlet deployment. *School of Computer Science Carnegie Mellon University Pittsburgh*, (2014).
- Hausknecht, M. J. & Stone, P. (2015). Deep Recurrent Q-Learning for Partially Observable MDPs. *CoRR*, abs/1507.06527. Consulted at <http://arxiv.org/abs/1507.06527>.
- Haykin, S. (2005). Cognitive radio: brain-empowered wireless communications. *IEEE journal on selected areas in communications*, 23(2), 201–220.
- He, X., Ren, Z., Shi, C. & Fang, J. (2016). A novel load balancing strategy of software-defined cloud/fog networking in the Internet of Vehicles. *China Communications*, 13(Supplement2), 140–149.
- He, Y., Zhang, Z., Yu, F. R., Zhao, N., Yin, H., Leung, V. C. M. & Zhang, Y. (2017). Deep-Reinforcement-Learning-Based Optimization for Cache-Enabled Opportunistic Interference Alignment Wireless Networks. *IEEE Transactions on Vehicular Technology*, 66(11), 10433-10445. doi: 10.1109/TVT.2017.2751641.
- Hernandez-Leal, P., Kartal, B. & Taylor, M. E. (2019). A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6), 750–797. doi: 10.1007/s10458-019-09421-1.

- Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735-1780. doi: 10.1162/neco.1997.9.8.1735.
- Hou, X., Muqing, W., Bo, L. & Yifeng, L. (2019). Multi-Controller Deployment Algorithm in Hierarchical Architecture for SDWAN. *IEEE Access*, 7, 65839-65851. doi: 10.1109/ACCESS.2019.2917027.
- Hu, J. & Wellman, M. P. (1998). Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm. *Proceedings of the Fifteenth International Conference on Machine Learning*, (ICML '98), 242-250.
- Hu, P., Dhelim, S., Ning, H. & Qiu, T. (2017). Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of network and computer applications*, 98, 27-42.
- ITU. (2015). *IMT Traffic estimates for the years 2020 to 2030* (Report n°M.2370-0). Consulted at <http://www.itu.int/pub/R-REP-M.2370>.
- Jain, R. & Paul, S. (2013). Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine*, 51(11), 24-31.
- Jia, S., Ai, Y., Zhao, Z., Peng, M. & Hu, C. (2016). Hierarchical content caching in fog radio access networks: ergodic rate and transmit latency. *China Communications*, 13(12), 1-14. doi: 10.1109/CC.2016.7897534.
- Jiang, W., Han, B., Habibi, M. A. & Schotten, H. D. (2021). The Road Towards 6G: A Comprehensive Survey. *IEEE Open Journal of the Communications Society*, 2, 334-366. doi: 10.1109/OJCOMS.2021.3057679.
- Jozefowicz, R., Zaremba, W. & Sutskever, I. (2015). An Empirical Exploration of Recurrent Network Architectures. *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, (ICML'15), 2342-2350.
- Kapetanakis, S. & Kudenko, D. (2002). Reinforcement Learning of Coordination in Cooperative Multi-Agent Systems. *Eighteenth National Conference on Artificial Intelligence*, pp. 326-331.
- Kashani, M. H., Ahmadzadeh, A. & Mahdipour, E. (2020). Load balancing mechanisms in fog computing: A systematic review. *arXiv preprint arXiv:2011.14706*.
- Kaur, K., Dhand, T., Kumar, N. & Zeadally, S. (2017). Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers. *IEEE wireless communications*, 24(3), 48-56.

- Kaur, K., Garg, S., Aujla, G. S., Kumar, N., Rodrigues, J. J. & Guizani, M. (2018). Edge computing in the industrial internet of things environment: Software-defined-networks-based edge-cloud interplay. *IEEE communications magazine*, 56(2), 44–51.
- Kaur, K., Garg, S., Kaddoum, G., Gagnon, F. & Jayakody, D. N. K. (2019). EnLoB: Energy and Load Balancing-Driven Container Placement Strategy for Data Centers. *2019 IEEE Globecom Workshops (GC Wkshps)*, pp. 1-6. doi: 10.1109/GCWkshps45667.2019.9024592.
- Kaur, M. & Aron, R. (2021). A systematic study of load balancing approaches in the fog computing environment. *The Journal of Supercomputing*, 1–46.
- Ke, H., Wang, J., Wang, H. & Ge, Y. (2019). Joint Optimization of Data Offloading and Resource Allocation With Renewable Energy Aware for IoT Devices: A Deep Reinforcement Learning Approach. *IEEE Access*, 7, 179349-179363. doi: 10.1109/ACCESS.2019.2959348.
- Kim, D., Moon, S., Hostallero, D., Kang, W. J., Lee, T., Son, K. & Yi, Y. (2019). Learning to Schedule Communication in Multi-agent Reinforcement Learning. *CoRR*, abs/1902.01554. Consulted at <http://arxiv.org/abs/1902.01554>.
- Kim, J. (2012, 05). Design and Evaluation of Mobile Applications with Full and Partial Offloadings. 7296, 172-182. doi: 10.1007/978-3-642-30767-6\_15.
- Ku, Y.-J., Lin, D.-Y., Lee, C.-F., Hsieh, P.-J., Wei, H.-Y., Chou, C.-T. & Pang, A.-C. (2017). 5G Radio Access Network Design with the Fog Paradigm: Confluence of Communications and Computing. *IEEE Communications Magazine*, 55(4), 46-52. doi: 10.1109/MCOM.2017.1600893.
- Kuang, Z., Li, L., Gao, J., Zhao, L. & Liu, A. (2019). Partial Offloading Scheduling and Power Allocation for Mobile Edge Computing Systems. *IEEE Internet of Things Journal*, 6(4), 6774-6785. doi: 10.1109/IIOT.2019.2911455.
- Kwak, J., Choi, O., Chong, S. & Mohapatra, P. (2016). Processor-Network Speed Scaling for Energy–Delay Tradeoff in Smartphone Applications. *IEEE/ACM Transactions on Networking*, 24(3), 1647-1660. doi: 10.1109/TNET.2015.2419219.
- Le, T.-D. & Kaddoum, G. (2021). LSTM-Based Channel Access Scheme for Vehicles in Cognitive Vehicular Networks With Multi-Agent Settings. *IEEE Transactions on Vehicular Technology*, 70(9), 9132-9143. doi: 10.1109/TVT.2021.3100591.
- LeCun, Y. & Bengio, Y. (1998). Convolutional Networks for Images, Speech, and Time Series. In *The Handbook of Brain Theory and Neural Networks* (pp. 255–258). Cambridge, MA, USA: MIT Press.

- Lee, G., Saad, W. & Bennis, M. (2019). An Online Optimization Framework for Distributed Fog Network Formation With Minimal Latency. *IEEE Transactions on Wireless Communications*, 18(4), 2244-2258. doi: 10.1109/TWC.2019.2901850.
- Li, H., Dong, M. & Ota, K. (2016). Control plane optimization in software-defined vehicular ad hoc networks. *IEEE Transactions on Vehicular Technology*, 65(10), 7895–7904.
- Li, L., Guan, Q., Jin, L. & Guo, M. (2019). Resource Allocation and Task Offloading for Heterogeneous Real-Time Tasks With Uncertain Duration Time in a Fog Queueing System. *IEEE Access*, 7, 9912-9925. doi: 10.1109/ACCESS.2019.2891130.
- Li, Y., Qi, F., Wang, Z., Yu, X. & Shao, S. (2020). Distributed Edge Computing Offloading Algorithm Based on Deep Reinforcement Learning. *IEEE Access*, 8, 85204-85215. doi: 10.1109/ACCESS.2020.2991773.
- Li, Y. (2017). Deep Reinforcement Learning: An Overview. *CoRR*, abs/1701.07274. Consulted at <http://arxiv.org/abs/1701.07274>.
- Liang, C. & Yu, F. R. (2014). Wireless network virtualization: A survey, some research issues and challenges. *IEEE Communications Surveys & Tutorials*, 17(1), 358–380.
- Lin, H., Garg, S., Hu, J., Kaddoum, G., Peng, M. & Hossain, M. S. (2021). Blockchain and Deep Reinforcement Learning Empowered Spatial Crowdsourcing in Software-Defined Internet of Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 22(6), 3755-3764. doi: 10.1109/TITS.2020.3025247.
- Lisa, J. Yan and Nick, Cercone. (2010). *Thoughts on Multiagent Learning: From A Reinforcement Learning Perspective* (Report n°CSE-2010-07). Consulted at <http://www.cs.toronto.edu/~lyan/CSE-2010-07.pdf>.
- Littman, M. L. (1994a). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994* (pp. 157–163). Elsevier.
- Littman, M. L. (1994b). Markov Games as a Framework for Multi-Agent Reinforcement Learning. *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, (ICML'94), 157–163.
- Liu, X., Yu, J., Feng, Z. & Gao, Y. (2020). Multi-agent reinforcement learning for resource allocation in IoT networks with edge computing. *China Communications*, 17(9), 220-236. doi: 10.23919/JCC.2020.09.017.
- Liu, X. F., Shahriar, M. R., Al Sunny, S. N., Leu, M. C. & Hu, L. (2017). Cyber-physical manufacturing cloud: Architecture, virtualization, communication, and testbed. *Journal of Manufacturing Systems*, 43, 352-364. doi: <https://doi.org/10.1016/j.jmsy.2017.04.004>. High Performance Computing and Data Analytics for Cyber Manufacturing.

- Long, Y., Chen, Z., Fang, J. & Tellambura, C. (2018). Data-driven-based analog beam selection for hybrid beamforming under mm-wave channels. *IEEE Journal of Selected Topics in Signal Processing*, 12(2), 340–352.
- Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P. & Mordatch, I. (2020). Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments.
- Lu, P., Barbalace, A., Palmieri, R. & Ravindran, B. (2013). Adaptive live migration to improve load balancing in virtual machine environment. *European Conference on Parallel Processing*, pp. 116–125.
- Lu, Y. (2017). Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration*, 6, 1-10. doi: <https://doi.org/10.1016/j.jii.2017.04.005>.
- Luo, Z.-Q. & Yu, W. (2006). An introduction to convex optimization for communications and signal processing. *IEEE Journal on Selected Areas in Communications*, 24(8), 1426-1438. doi: 10.1109/JSAC.2006.879347.
- Luong, N. C., Hoang, D. T., Gong, S., Niyato, D., Wang, P., Liang, Y.-C. & Kim, D. I. (2019). Applications of Deep Reinforcement Learning in Communications and Networking: A Survey. *IEEE Communications Surveys Tutorials*, 21(4), 3133-3174. doi: 10.1109/COMST.2019.2916583.
- Mach, P. & Becvar, Z. (2017). Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Communications Surveys Tutorials*, 19(3), 1628-1656. doi: 10.1109/COMST.2017.2682318.
- Mahmood, K., Chilwan, A., Østerbø, O. & Jarschel, M. (2015). Modelling of OpenFlow-based software-defined networks: the multiple node case. *IET Networks*, 4(5), 278–284.
- Mao, S. (2010). Chapter 8 - Fundamentals of communication networks. In Wyglinski, A. M., Nekovee, M. & Hou, Y. T. (Eds.), *Cognitive Radio Communications and Networks* (pp. 201-234). Oxford: Academic Press. doi: <https://doi.org/10.1016/B978-0-12-374715-0.00008-3>.
- Mao, Y., Zhang, J., Song, S. & Letaief, K. B. (2016). Power-delay tradeoff in multi-user mobile-edge computing systems. *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6.
- Masip-Bruin, X., Marín-Tordera, E., Tashakor, G., Jukan, A. & Ren, G.-J. (2016). Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems. *IEEE Wireless Communications*, 23(5), 120–128.

- Matignon, L., Laurent, G. J. & Le Fort-Piat, N. (2009). *Coordination of independent learners in cooperative Markov games*. working paper or preprint. Consulted at <https://hal.archives-ouvertes.fr/hal-00370889>.
- Min, M., Xiao, L., Chen, Y., Cheng, P., Wu, D. & Zhuang, W. (2019). Learning-based computation offloading for IoT devices with energy harvesting. *IEEE Transactions on Vehicular Technology*, 68(2), 1930–1941.
- Mishra, S. K., Puthal, D., Rodrigues, J. J. P. C., Sahoo, B. & Dutkiewicz, E. (2018). Sustainable Service Allocation Using a Metaheuristic Technique in a Fog Server for Industrial Applications. *IEEE Transactions on Industrial Informatics*, 14(10), 4497–4506. doi: 10.1109/TII.2018.2791619.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518, 529–533.
- Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J. & Polakos, P. A. (2017). A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE communications surveys & tutorials*, 20(1), 416–464.
- Muñoz, O., Pascual-Iserte, A. & Vidal, J. (2015). Optimization of Radio and Computational Resources for Energy Efficiency in Latency-Constrained Application Offloading. *IEEE Transactions on Vehicular Technology*, 64(10), 4738–4755. doi: 10.1109/TVT.2014.2372852.
- Naeem, F., Kaddoum, G. & Tariq, M. (2021). Digital Twin-empowered Network Slicing in B5G Networks: Experience-driven approach. *2021 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–5. doi: 10.1109/GCWkshps52748.2021.9682073.
- Naqvi, S. A. A., Javaid, N., Butt, H., Kamal, M. B., Hamza, A. & Kashif, M. (2018). Metaheuristic optimization technique for load balancing in cloud-fog environment integrated with smart grid. *International Conference on Network-Based Information Systems*, pp. 700–711.
- Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1), 48–49. doi: 10.1073/pnas.36.1.48.
- Nasir, Y. & Guo, D. (2019). Multi-Agent Deep Reinforcement Learning for Dynamic Power Allocation in Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 37(10), 2239–2250. doi: 10.1109/JSAC.2019.2933973.
- Neghabi, A. A., Navimipour, N. J., Hosseinzadeh, M. & Rezaee, A. (2018). Load balancing mechanisms in the software defined networks: a systematic and comprehensive review of the literature. *IEEE Access*, 6, 14159–14178.



- Neil Bergmann, B., Chung, Y., Yang, X., Chen, Z., Yeh, W., He, X. & Jurdak, R. (2013). Using swarm intelligence to optimize the energy consumption for distributed systems. *Modern Applied Science*, 7(6), 59–66. doi: 10.5539/mas.v7n6p59.
- Neto, J. L. D., Yu, S.-Y., Macedo, D. F., Nogueira, J. M. S., Langar, R. & Secci, S. (2018). ULOOF: A User Level Online Offloading Framework for Mobile Edge Computing. *IEEE Transactions on Mobile Computing*, 17(11), 2660–2674. doi: 10.1109/TMC.2018.2815015.
- Ning, Z., Dong, P., Kong, X. & Xia, F. (2019). A Cooperative Partial Computation Offloading Scheme for Mobile Edge Computing Enabled Internet of Things. *IEEE Internet of Things Journal*, 6(3), 4804–4814. doi: 10.1109/JIOT.2018.2868616.
- Omidshafiei, S., Pazis, J., Amato, C., How, J. P. & Vian, J. (2017). Deep Decentralized Multi-task Multi-Agent Reinforcement Learning under Partial Observability. *CoRR*, abs/1703.06182. Consulted at <http://arxiv.org/abs/1703.06182>.
- OpenFog Consortium. (2017). *OpenFog reference architecture*.
- OpenFog Consortium Architecture Working Group. (2017). *OpenFog reference architecture for fog computing*. Consulted at [https://www.iiconsortium.org/pdf/OpenFog\\_Reference\\_Architecture\\_2\\_09\\_17.pdf](https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf).
- Oroojlooyjadid, A. & Hajinezhad, D. (2019). A Review of Cooperative Multi-Agent Deep Reinforcement Learning. *CoRR*, abs/1908.03963. Consulted at <http://arxiv.org/abs/1908.03963>.
- Pan, S., Zhang, Z., Zhang, Z. & Zeng, D. (2019). Dependency-aware computation offloading in mobile edge computing: A reinforcement learning approach. *IEEE Access*, 7, 134742–134753.
- Panait, L. & Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3), 387–434.
- Patel, M., Naughton, B., Chan, C., Sprecher, N., Abeta, S., Neal, A. et al. (2014). Mobile-edge computing introductory technical white paper. *White paper, mobile-edge computing (MEC) industry initiative*, 29, 854–864.
- Peltonen, E., Bennis, M., Capobianco, M., Debbah, M., Ding, A., Gil-Castiñeira, F., Jurmu, M., Karvonen, T., Kelanti, M., Kliks, A. et al. (2020). 6G white paper on edge intelligence. *arXiv preprint arXiv:2004.14850*.
- Peng, M., Yan, S., Zhang, K. & Wang, C. (2016). Fog-computing-based radio access networks: Issues and challenges. *Ieee Network*, 30(4), 46–53.

- Perkins, T. J. & Precup, D. (2002). A Convergent Form of Approximate Policy Iteration. *Proceedings of the 15th International Conference on Neural Information Processing Systems*, (NIPS'02), 1627–1634.
- Peshkin, L., Kim, K., Meuleau, N. & Kaelbling, L. P. (2014). Learning to Cooperate via Policy Search. *CoRR*, abs/1408.1484. Consulted at <http://arxiv.org/abs/1408.1484>.
- Poularakis, K., Qin, Q., Nahum, E., Rio, M. & Tassiulas, L. (2017). Bringing SDN to the mobile edge. *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBD-Com/IOP/SCI)*, pp. 1–6.
- Puterman, M. (2005). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley. Consulted at <https://books.google.ca/books?id=Y-gmAQAIAAJ>.
- Ren, J., Yu, G., He, Y. & Li, G. Y. (2019). Collaborative cloud and edge computing for latency minimization. *IEEE Transactions on Vehicular Technology*, 68(5), 5031–5044.
- Roca, D., Milito, R., Nemirovsky, M. & Valero, M. (2018). Tackling IoT Ultra Large Scale Systems: fog computing in support of hierarchical emergent behaviors. In *Fog computing in the internet of things* (pp. 33–48). Springer.
- Sadeghi, A., Wang, G. & Giannakis, G. B. (2019). Deep Reinforcement Learning for Adaptive Caching in Hierarchical Content Delivery Networks. *IEEE Transactions on Cognitive Communications and Networking*, 5(4), 1024–1033. doi: 10.1109/TCCN.2019.2936193.
- Sarkar, S. & Misra, S. (2016). Theoretical modelling of fog computing: a green computing paradigm to support IoT applications. *IET Networks*, 5(2), 23–29. doi: <https://doi.org/10.1049/iet-net.2015.0034>.
- Sarkar, S., Chatterjee, S. & Misra, S. (2018). Assessment of the Suitability of Fog Computing in the Context of Internet of Things. *IEEE Transactions on Cloud Computing*, 6(1), 46–59. doi: 10.1109/TCC.2015.2485206.
- Satyanarayanan, M., Bahl, P., Caceres, R. & Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4), 14–23.
- Satyanarayanan, M., Lewis, G., Morris, E., Simanta, S., Boleng, J. & Ha, K. (2013). The role of cloudlets in hostile environments. *IEEE Pervasive Computing*, 12(4), 40–49.
- Schaerf, A., Shoham, Y. & Tennenholtz, M. (1995). Adaptive Load Balancing: A Study in Multi-Agent Learning. *CoRR*, cs.AI/9505102. Consulted at <https://arxiv.org/abs/cs/9505102>.



- Schumacher, A., Erol, S. & Sihn, W. (2016). A Maturity Model for Assessing Industry 4.0 Readiness and Maturity of Manufacturing Enterprises. *Procedia CIRP*, 52, 161-166.
- Shafique, K., Khawaja, B. A., Sabir, F., Qazi, S. & Mustaqim, M. (2020). Internet of Things (IoT) for Next-Generation Smart Systems: A Review of Current Challenges, Future Trends and Prospects for Emerging 5G-IoT Scenarios. *IEEE Access*, 8, 23022-23040. doi: 10.1109/ACCESS.2020.2970118.
- Sharma, S. & Saini, H. (2019). A novel four-tier architecture for delay aware scheduling and load balancing in fog environment. *Sustainable Computing: Informatics and Systems*, 24, 100355.
- Sim, M. S., Lim, Y.-G., Park, S. H., Dai, L. & Chae, C.-B. (2020). Deep learning-based mmWave beam selection for 5G NR/6G with sub-6 GHz channel information: Algorithms and prototype validation. *IEEE Access*, 8, 51634–51646.
- Sodomka, E., Hilliard, E., Littman, M. & Greenwald, A. (2013). Coco-q: Learning in stochastic games with side payments. *International Conference on Machine Learning*, pp. 1471–1479.
- Sorokin, I., Seleznev, A., Pavlov, M., Fedorov, A. & Ignateva, A. (2015). Deep Attention Recurrent Q-Network.
- Sukhbaatar, S., Szlam, A. & Fergus, R. (2016). Learning Multiagent Communication with Backpropagation. *CoRR*, abs/1605.07736. Consulted at <http://arxiv.org/abs/1605.07736>.
- Sun, Y., Peng, M. & Mao, S. (2019a). Deep Reinforcement Learning-Based Mode Selection and Resource Management for Green Fog Radio Access Networks. *IEEE Internet of Things Journal*, 6(2), 1960-1971. doi: 10.1109/JIOT.2018.2871020.
- Sun, Y., Peng, M. & Mao, S. (2019b). A Game-Theoretic Approach to Cache and Radio Resource Management in Fog Radio Access Networks. *IEEE Transactions on Vehicular Technology*, 68(10), 10145-10159. doi: 10.1109/TVT.2019.2935098.
- Sun, Y., Peng, M. & Mao, S. (2019c). Deep Reinforcement Learning-Based Mode Selection and Resource Management for Green Fog Radio Access Networks. *IEEE Internet of Things Journal*, 6(2), 1960-1971. doi: 10.1109/JIOT.2018.2871020.
- Sun, Y., Peng, M., Zhou, Y., Huang, Y. & Mao, S. (2019d). Application of Machine Learning in Wireless Networks: Key Techniques and Open Issues. *IEEE Communications Surveys Tutorials*, 21(4), 3072-3108. doi: 10.1109/COMST.2019.2924243.
- Sun, Z., Wei, L., Xu, C., Wang, T., Nie, Y., Xing, X. & Lu, J. (2019e). An energy-efficient cross-layer-sensing clustering method based on intelligent fog computing in WSNs. *IEEE Access*, 7, 144165–144177.

- Sutton, R. S. (1990). Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In Porter, B. & Mooney, R. (Eds.), *Machine Learning Proceedings 1990* (pp. 216-224). San Francisco (CA): Morgan Kaufmann. doi: <https://doi.org/10.1016/B978-1-55860-141-3.50030-4>.
- Sutton, R. S., McAllester, D., Singh, S. & Mansour, Y. (2000). Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Advances in Neural Information Processing Systems*, 12. Consulted at <https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf>.
- Sutton, R. & Barto, A. (2018). *Reinforcement Learning, second edition: An Introduction*. MIT Press. Consulted at <https://books.google.ca/books?id=uWV0DwAAQBAJ>.
- Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S. & Sabella, D. (2017). On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3), 1657–1681.
- Tan, M. (1993). Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 330–337.
- Tang, B., Chen, Z., Heffernan, G., Wei, T., He, H. & Yang, Q. (2015). A hierarchical distributed fog computing architecture for big data analysis in smart cities. In *Proceedings of the ASE BigData & SocialInformatics 2015* (pp. 1–6).
- Tang, Q., Lyu, H., Han, G., Wang, J. & Wang, K. (2020). Partial offloading strategy for mobile edge computing considering mixed overhead of time and energy. *Neural Computing and Applications*, 32. doi: 10.1007/s00521-019-04401-8.
- Tomovic, S., Yoshigoe, K., Maljevic, I. & Radusinovic, I. (2017). Software-defined fog network architecture for IoT. *Wireless Personal Communications*, 92(1), 181–196.
- Tordera, E. M., Masip-Bruin, X., Garcia-Alminana, J., Jukan, A., Ren, G.-J., Zhu, J. & Farré, J. (2016). What is a fog node a tutorial on current concepts towards a common definition. *arXiv preprint arXiv:1611.09193*.
- Tsai, C.-W. & Rodrigues, J. J. P. C. (2014). Metaheuristic Scheduling for Cloud: A Survey. *IEEE Systems Journal*, 8(1), 279-291. doi: 10.1109/JSYST.2013.2256731.
- Tseng, C. H. (2016). Multipath load balancing routing for Internet of things. *Journal of Sensors*, 2016.
- Van Huynh, N., Thai Hoang, D., Nguyen, D. N. & Dutkiewicz, E. (2019). Optimal and Fast Real-Time Resource Slicing With Deep Dueling Neural Networks. *IEEE Journal on Selected Areas in Communications*, 37(6), 1455-1470. doi: 10.1109/JSAC.2019.2904371.

- Varshavskaya, P., Kaelbling, L. P. & Rus, D. (2009). Efficient distributed reinforcement learning through agreement. In *Distributed Autonomous Robotic Systems 8* (pp. 367–378). Springer.
- Vázquez, A., Pastor-Satorras, R. & Vespignani, A. (2002). Large-scale topological and dynamical properties of the Internet. *Physical Review E*, 65(6), 066130.
- Verma, M. & Yadav, N. B. A. K. (2015). An architecture for load balancing techniques for Fog computing environment. *International Journal of Computer Science and Communication*, 8(2), 43–49.
- Vilalta, R., Lopez, V., Giorgetti, A., Peng, S., Orsini, V., Velasco, L., Serral-Gracia, R., Morris, D., De Fina, S., Cugini, F., Castoldi, P., Mayoral, A., Casellas, R., Martinez, R., Verikoukis, C. & Munoz, R. (2017). TelcoFog: A Unified Flexible Fog and Cloud Computing Architecture for 5G Networks. *IEEE Communications Magazine*, 55(8), 36–43. doi: 10.1109/M-COM.2017.1600838.
- Wan, J., Chen, B., Wang, S., Xia, M., Li, D. & Liu, C. (2018). Fog computing for energy-aware load balancing and scheduling in smart factory. *IEEE Transactions on Industrial Informatics*, 14(10), 4548–4556.
- Wan, X., Sheng, G., Li, Y., Xiao, L. & Du, X. (2017). Reinforcement Learning Based Mobile Offloading for Cloud-Based Malware Detection. *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pp. 1–6. doi: 10.1109/GLOCOM.2017.8254503.
- Wang, C., Liang, C., Yu, F. R., Chen, Q. & Tang, L. (2017a). Computation Offloading and Resource Allocation in Wireless Cellular Networks With Mobile Edge Computing. *IEEE Transactions on Wireless Communications*, 16(8), 4924–4938. doi: 10.1109/TWC.2017.2703901.
- Wang, C., Liang, C., Yu, F. R., Chen, Q. & Tang, L. (2017b). Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. *IEEE Transactions on Wireless Communications*, 16(8), 4924–4938.
- Wang, J., Jiang, C., Zhang, H., Ren, Y., Chen, K.-C. & Hanzo, L. (2020). Thirty Years of Machine Learning: The Road to Pareto-Optimal Wireless Networks. *IEEE Communications Surveys Tutorials*, 22(3), 1472–1514. doi: 10.1109/COMST.2020.2965856.
- Wang, P., Lin, S.-C. & Luo, M. (2016a). A Framework for QoS-aware Traffic Classification Using Semi-supervised Machine Learning in SDNs. *2016 IEEE International Conference on Services Computing (SCC)*, pp. 760–765. doi: 10.1109/SCC.2016.133.

- Wang, X., Hu, J., Lin, H., Garg, S., Kaddoum, G., Piran, M. J. & Hossain, M. S. (2022). QoS and Privacy-Aware Routing for 5G-Enabled Industrial Internet of Things: A Federated Reinforcement Learning Approach. *IEEE Transactions on Industrial Informatics*, 18(6), 4189–4197. doi: 10.1109/TII.2021.3124848.
- Wang, Y., Sheng, M., Wang, X., Wang, L. & Li, J. (2016b). Mobile-edge computing: Partial computation offloading using dynamic voltage scaling. *IEEE Transactions on Communications*, 64(10), 4268–4282.
- Wei, Y., Yu, F. R., Song, M. & Han, Z. (2019). Joint Optimization of Caching, Computing, and Radio Resources for Fog-Enabled IoT Using Natural Actor–Critic Deep Reinforcement Learning. *IEEE Internet of Things Journal*, 6(2), 2061–2073. doi: 10.1109/JIOT.2018.2878435.
- Wu, J., Li, R., An, X., Peng, C., Liu, Z., Crowcroft, J. & Zhang, H. (2021). Toward Native Artificial Intelligence in 6G Networks: System Design, Architectures, and Paradigms. *arXiv preprint arXiv:2103.02823*.
- Xiao, Y., Shi, G., Li, Y., Saad, W. & Poor, H. V. (2020). Toward self-learning edge intelligence in 6G. *IEEE Communications Magazine*, 58(12), 34–40.
- Xu, X., Fu, S., Cai, Q., Tian, W., Liu, W., Dou, W., Sun, X. & Liu, A. X. (2018). Dynamic resource allocation for load balancing in fog environment. *Wireless Communications and Mobile Computing*, 2018.
- Yadav, R., Zhang, W., Kaiwartya, O., Song, H. & Yu, S. (2020). Energy-Latency Tradeoff for Dynamic Computation Offloading in Vehicular Fog Computing. *IEEE Transactions on Vehicular Technology*, 69(12), 14198–14211. doi: 10.1109/TVT.2020.3040596.
- Yadav, R., Zhang, W., Elgendy, I. A., Dong, G., Shafiq, M., Laghari, A. A. & Prakash, S. (2021). Smart healthcare: RL-based task offloading scheme for edge-enable sensor networks. *IEEE Sensors Journal*, 21(22), 24910–24918.
- Yi, S., Li, C. & Li, Q. (2015). A Survey of Fog Computing: Concepts, Applications and Issues. (Mobidata '15), 37–42. doi: 10.1145/2757384.2757397.
- Yousefpour, A., Ishigaki, G., Gour, R. & Jue, J. P. (2018). On reducing IoT service delay via fog offloading. *IEEE Internet of things Journal*, 5(2), 998–1010.
- Zhang, C., Patras, P. & Haddadi, H. (2019a). Deep Learning in Mobile and Wireless Networking: A Survey. *IEEE Communications Surveys Tutorials*, 21(3), 2224–2287. doi: 10.1109/COMST.2019.2904897.

- Zhang, C., Liu, Z., Gu, B., Yamori, K. & Tanaka, Y. (2018a). A Deep Reinforcement Learning Based Approach for Cost- and Energy-Aware Multi-Flow Mobile Data Offloading. *IEICE Transactions on Communications*, E101.B(7), 1625–1634. doi: 10.1587/transcom.2017cqp0014.
- Zhang, D., Haider, F., St-Hilaire, M. & Makaya, C. (2019b). Model and Algorithms for the Planning of Fog Computing Networks. *IEEE Internet of Things Journal*, 6(2), 3873–3884. doi: 10.1109/JIOT.2019.2892940.
- Zhang, H., Xiao, Y., Bu, S., Niyato, D., Yu, F. R. & Han, Z. (2017a). Computing resource allocation in three-tier IoT fog networks: A joint optimization approach combining Stackelberg game and matching. *IEEE Internet of Things Journal*, 4(5), 1204–1215.
- Zhang, K., Yang, Z., Liu, H., Zhang, T. & Basar, T. (2018b). Fully decentralized multi-agent reinforcement learning with networked agents. *International Conference on Machine Learning*, pp. 5872–5881.
- Zhang, K., Cao, J. & Zhang, Y. (2021). Adaptive Digital Twin and Multi-agent Deep Reinforcement Learning for Vehicular Edge Computing and Networks. *IEEE Transactions on Industrial Informatics*.
- Zhang, W., Zhang, Z. & Chao, H.-C. (2017b). Cooperative fog computing for dealing with big data in the internet of vehicles: Architecture and hierarchical resource management. *IEEE Communications Magazine*, 55(12), 60–67.
- Zhang, Z., Ma, L., Leung, K. K., Tassiulas, L. & Tucker, J. (2018c). Q-Placement: Reinforcement-Learning-Based Service Placement in Software-Defined Networks. *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1527–1532. doi: 10.1109/ICDCS.2018.00159.
- Zhao, T., Zhou, S., Guo, X. & Niu, Z. (2017). Tasks scheduling and resource allocation in heterogeneous cloud for delay-bounded mobile edge computing. *2017 IEEE International Conference on Communications (ICC)*, pp. 1–7. doi: 10.1109/ICC.2017.7996858.
- Zhu, J., Chan, D. S., Prabhu, M. S., Natarajan, P., Hu, H. & Bonomi, F. (2013). Improving Web Sites Performance Using Edge Servers in Fog Computing Architecture. *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, pp. 320–323. doi: 10.1109/SOSE.2013.73.