June 2021

Adaptive Network Slicing in Fog RAN for IoT with Heterogeneous Latency and Computing Requirements: A Deep Reinforcement Learning Approach

Almuthanna Nassar
University of South Florida

Adaptive Network Slicing in Fog RAN for IoT with Heterogeneous Latency and Computing Requirements: A Deep Reinforcement Learning Approach

by

Almuthanna Nassar

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Electrical Engineering
College of Engineering
University of South Florida

Date of Approval:
June 14, 2021

Keywords: Edge Computing, 5G Cellular Networks, Intelligent Vehicular Systems, Resource Allocation, Smart Cities, Ultra-Reliable Low-Latency Communications

Dedication

To my extremely supportive father, Dr. Turki Nassar, and my loving mother, Rasmieh Nassar, for being my great role models and first teachers, and building such a highly educated family. Their love, encouragement, inspiration, support, trust, drive, motivation, and prayers helped me to be what I am today. To my beloved and wonderful wife, Shatha, for her love, care, patience, and support. To my adorable and beloved children, Zain, Aljury, Haritha, and Kenan, for being the best thing in my life and making me always happy, and for their smiles, kisses, and big hugs. To my amazing siblings, Musab, Talha, Hajar, Almugheera, Omaima, Gutaiba, Ibrahim, and Horan, for their love and endless support. I love you all.

Acknowledgments

First and foremost, praise and thanks is to Allah, The Almighty, on whom ultimately we depend for sustenance and guidance. I would never have accomplished my PhD without the grace of The Almighty Allah.

Words cannot express my deepest gratitude and sincere appreciation to my mentor and major advisor, Dr. Yasin Yilmaz, for his guidance, encouragement, support, patience, and the great knowledge he shared. I especially appreciate his constant guidance and putting me on the right way from the first day I started my PhD, the unlimited support in exploring up to date research areas and improving my programming skills, the long hours and the meetings he dedicated for me, and his generous financial support in funding me over the course of my PhD. Dr. Yilmaz has always amazed me with his high degree of professionalism, expertise, humbleness, management skills, morality, and great kindness. His distinguished knowledge, skills, and personalityhave advanced my knowledge and refined my character. He will always continue to be a role model for me and a part of my prayers. His careful review for my papers, timely response, and constructive comments were invaluable. His efforts and great contribution helped me completing my PhD, contributing to the research areas of machine learning and wireless communications, and putting this dissertation into its final form.

I would also like to extend my deepest gratitude to my committee members, Dr. Nasir Ghani, Dr. Ismail Uysal, Dr. Ankit Shah, and Dr. Lu Lu, for the considerable time and effort dedicated towards serving on my committee. I am sincerely thankful for the DQE and candidacy presentation feedback, reviewing my dissertation, and being part of my dissertation defense. I also would like to thank Dr. Nasir Ghani for accepting to chair my DQE and candidacy. I am also grateful to Dr. John Murray-Bruce for chairing my dissertation defense.

I am also grateful to the University of South Florida administration, faculty and staff of the college of engineering, the Electrical Engineering Department for providing international students employment opportunities, and offering the facilities and the environment for research and studying. Special thanks to all professors who taught me during my PhD providing me with a solid academic base which has enabled me to take up this research and enrich my knowledge. My gratitude also extends to my friends and fellows in the Secure and Intelligent Systems (SIS) Lab, Abdullah, Mahsa, Ammar, Keval, and Salman, for the discussions, suggestions, and knowledge shared via course work and research. Special thanks to Diana, Kristen, and Gabriela at the Electrical Engineering Department for the academic and administration endless support throughout the past four years. I am grateful to IEEE ComSoc for awarding me the IEEE ComSoc student travel grant which helped me to travel and present my research work in the IEEE ICC 2019 in Shanghai, China. My gratitude is also extended to the Office of Graduate Studies for awarding me the Dissertation Completion Fellowship which has supported me in Summer 2021 to complete my dissertation.

Last not least, I would like to express my deepest gratitude to my parents, my wife, my children, and my siblings for their encouragement, support, permanent love, patience and prayers throughout my life. I have experienced many ups and downs in the past several years however, they were always there helping me to overcome those challenges. Special thanks to my father and my brother, Ibrahim, for their financial support throughout my PhD. My achievement today, stands as a testament to your constant love, support, trust and encouragement.

Table of Contents

List of Tables

List of Figures

Abstract

In view of the recent advances in Internet of Things (IoT) devices and the emerging new breed of smart city applications and intelligent vehicular systems driven by artificial intelligence, fog radio access network (F-RAN) has been recently introduced for the next generation wireless communications. The capability of F-RAN has emerged to overcome the latency limitations of cloud-RAN (C-RAN) and assure the quality-of-service (QoS) requirements of the ultra-reliable-low-latency-communication (URLLC) for IoT applications. To this end, fog nodes (FNs) are equipped with computing, signal processing and storage capabilities to extend the inherent operations and services of the cloud to the edge. However, due to their limited resources compared to the cloud, FNs should utilize their valuable resources intelligently to satisfy various QoS requirements in synergy and complementarity with the cloud. This dissertation considers the network slicing problem of sequentially allocating the limited resources at the network edge (i.e., FNs) to vehicular and smart city users with heterogeneous latency and computing demands in dynamic environments.

The network slicing problem for the edge resource allocation is formulated as a Markov decision process (MDP), for which dynamic programming and reinforcement learning (RL) solutions are proposed to adaptively learn the optimal slicing policy. The research conducted in this dissertation can be divided into two phases. Phase I considers a single FN with a tractable number of states, where tabular solution methods are used such as dynamic programming and model-free RL methods, which include Monte Carlo, SARSA, expected SARSA, and Q-Learning (QL). Phase II deals with a cluster of FNs coordinated with an EC with high-dimensional state space, where deep RL (DRL) methods such as deep Q-networks (DQN) are adopted to address the high-dimensionality using deep neural networks. A network slicing model based on a cluster of FNs coordinated with an edge controller (EC) is developed to efficiently utilize the limited resources at the network edge. Specifically, for each service request in a cluster, the EC decides which FN to execute the task (i.e. locally serve the request at the edge) or to reject the task and refer it to the cloud to conserve the edge resources for future users of potentially higher utility to the system (i.e., lower latency requirement).

The developed network slicing model and the proposed RL and DRL algorithms quickly learn the optimal policy through interaction with the IoT environment, which enables adaptive and automated network slicing for efficient edge resource allocation in dynamic traffic and load profiles of dense vehicular and smart city service requests with heterogeneous latency and computing needs. This means providing a cost-effective service customization through virtual partitioning of the RAN resources to enable low-latency IoT communications such as autonomous driving and public safety operations. The proposed approach also adapts to various design objectives including edge resource utilization, cloud offloading, and satisfying diverse QoS requirements.

Chapter 1: Introduction

There is an ever-growing demand for wireless communication technologies due to several reasons such as the increasing popularity of Internet of Things (IoT) devices, the widespread use of social networking platforms, the proliferation of mobile applications, and the current lifestyle that has become highly dependent on technology in all aspects.[1] It is expected that the number of connected devices worldwide will reach three times the global population in 2022 with 3.6 devices per capita. However, in some regions, such as North America, the number of connected devices is projected to reach about 13.4 devices per capita by 2022. IoT devices will constitute 50% of the 29.3 billion connected devices globally by 2023 which makes the massive IoT a very common concept, where Internet of Vehicles (IoV) and smart city applications are the fastest growing IoT implementations at annual growth rates of 30% and 26%, respectively [5]. The emerging new breed of IoT applications which involve video analytics, augmented reality (AR), virtual reality (VR), and artificial intelligence (AI) will produce an annual worldwide data volume of 4.8 zettabytes by 2022, which corresponds to 4 times the traffic in 2016 and 184 times the data traffic in 2005, in which wireless and mobile devices will account for 71% of this forcast [3, 6]. Equipped with variety of sensors, radars, lidars, ultra-high definition (UHD) video cameras, GPS, navigation system, and infotainment facilities, a connected and autonomous vehicle (CAV) will generate 4.0 terabyte of data in a single day, of which 1.0 gigabyte need to be processed every second [7].

[1]Portions of this chapter were published in IEEE GlobalSIP [1], IEEE ICC [2], IEEE Access [3] and arXiv [4]. Copyright permissions from the publishers are included in Appendix A.

This unprecedented demand for mobile data services makes it unbearable for service providers with the current third generation (3G) and fourth generation (4G) networks to keep pace with it [8]. The design criteria for fifth generation (5G) wireless communication systems will include providing ultra-low latency, wider coverage, reduced energy usage, increased spectral efficiency, more connected devices, improved availability, and very high data rates of multi giga-bit-per-second (Gbps) everywhere in the network including cell edges [9]. Several radio frequency (RF) coverage and capacity solutions are proposed to fulfill the goals of 5G including, beamforming, carrier aggregation, higher order modulation, and dense deployment of small cells [10]. Millimeter-wave (mm-wave) frequency range is utilized in 5G because of the spacious bandwidths available in these frequencies for cellular services [11]. Massive multi-input-multi-output (MIMO) is also involved for excellent spectral efficiency and superior energy efficiency [12].

1.1 Cloud and Fog RAN

Through centralization of network functionalities via virtualization, cloud radio access network (C-RAN) architecture is proposed to address the big data challenges of massive IoT. In C-RAN, densely-deployed disseminated remote radio units (RRUs) are connected through high capacity fronthaul trunks to a powerful cloud controller (CC) where they share a vast pooling of storage and baseband units (BBUs) [13]. The centralized computing, processing, and collaborative radio in C-RAN improves network security, flexibility, availability, and spectral efficiency. It also simplifies network operations and management, enhances capacity, and reduces energy usage [14]. However, considering the fast growing demands of IoT deployments, C-RAN lays overwhelming onus on cloud computing and fronthaul links, and dictates unacceptable delay caused mainly by the large return transmission times, finite-capacity fronthaul trunks, and flooded cloud processors [15]. The latency limitation in C-RAN makes it challenging to meet the desired quality-of-service (QoS) requirements, especially for the delay-sensitive IoV and smart city applications [16]. Hence, an evolved architecture, fog RAN (F-RAN) is introduced to extend the inherent operations and services of cloud to the edge [17]. In F-RAN, the fog nodes (FNs) are not only restricted to perform the regular radio frequency (RF) functionalities of RRUs, but they are also equipped with computing, storage, and processing resources to afford the low latency demand by delivering network functionalities directly at the edge and independently from the cloud [18]. However, due to their limited resources compared to the cloud, FNs are unable to serve all requests from IoV and smart city applications, and hence they should utilize their limited resources intelligently to satisfy the QoS requirements in synergy and complementarity with the cloud [19].

1.2 Network Slicing for Heterogeneous IoT Demands

IoT and smart city applications demand various computing, throughput, latency, availability, and reliability requirements to satisfy a desired level of QoS. Some applications are more delay-sensitive than others, while some can tolerate larger delays [20–22]. For instance, in-vehicle audio, news, and video infotainment services are satisfied by the traditional mobile broadband (MBB) services of high throughput and capacity with latency greater than 100 ms [20]. Cloud computing plays an essential role for such delay-tolerant applications. Other examples of delay-tolerant applications include smart parking [23], intelligent waste management [24], infrastructure (e.g., bridges, railways, etc.) monitoring [25], air quality management [26], noise monitoring [27], smart city lighting [28], smart management of city energy consumption [29], and automation of public buildings such as schools, museums, and administration offices to automatically and remotely control lighting and air condition [30].

On the other hand, latency and reliability are more critical for other IoV and smart city applications. For instance, deployment scenarios based on enhanced mobile broadband (eMBB) require latency of 4.0 ms. Enhanced vehicle-to-everything (eV2X) applications demand 3-10 ms latency with packet loss rate of 10^{-5}. Ultra-reliable and low-latency communications (URLLC) seek latency level of 0.5-1.0 ms and 99.999% reliability [31, 32], e.g., autonomous driving [33]. AI-driven and video analytics services are considered both latency-critical and compute-intensive applications [22]. For instance, real-time video streaming for traffic management in intelligent transportation system (ITS) [34] requires a frame rate of 100 Hz, which corresponds to a latency of 10 ms between frames [20]. Future electric vehicles (EVs) and CAVs are viewed as computers on wheels (COWs) rather than cars because they are equipped with super computers to execute extremely intensive computing tasks including video analytics and AI-driven functionalities. However, with the high power consumption associated with such intense computing, COWs capabilities are still bounded in terms of computing power, storage, and battery life. Hence, computing offloading to fog and cloud networks is inevitable [35]. Especially in a dynamic traffic and load profiles of dense IoV and smart city service requests with heterogeneous latency and computing needs, partitioning RAN resources virtually, i.e., network slicing [36], assures service customization.

Network slicing is introduced for the evolving 5G and beyond communication technologies as a cost-effective solution for mobile operators and service providers to satisfy various user QoS [1]. In network slicing, a heterogeneous network of various access technologies and QoS demands that share a common physical infrastructure is logically divided into virtual network slices to improve network flexibility. Each network slice acts as an independent end-to-end network and supports various service requirements and a variety of business cases and applications. In this dissertation, we consider the network slicing problem of adaptively allocating the limited edge computing and processing resources in F-RAN to dynamic IoT and smart city applications with heterogeneous latency demands and diverse computing loads. We present a novel framework for resource allocation in F-RAN for 5G by employing reinforcement learning methods to guarantee the efficient utilization of limited FN resources while satisfying the low-latency requirements of IoT applications and improve the overall system performance.

1.3 Related Work

There is an increasing number of works in the literature focusing on the edge resource allocation problem and network slicing as an emerging network architecture for 5G and future technologies. Recently, a good number of works in the literature focused on achieving low latency for IoT applications in 5G F-RAN. For instance, resource allocation based on cooperative edge computing has been studied in [37–41] for achieving ultra-low latency in F-RAN. The work in [37] proposed a mesh paradigm for edge computing, where the decision-making tasks are distributed among edge devices instead of utilizing the cloud server. The authors in [38, 41] considered heterogeneous F-RAN structures including, small cells and macro base stations, and provided an algorithm for selecting the F-RAN nodes to serve with proper heterogeneous resource allocation. The number of F-RAN nodes and their locations have been investigated by [42]. Content fetching is used in [19, 39] to maximize the delivery rate when the requested content is available in the cache of fog access points. In [43], cloud predicts users' mobility patterns and determines the required resources for the requested contents by users, which are stored at cloud and small cells. The work in [40] addressed the issue of load balancing in fog computing and used fog clustering to improve user's quality of experience. The congestion problem, when resource allocation is done based on the best signal quality received by the end user, is highlighted in [44, 45]. The work in [44] provided a solution to balance the resource allocation among remote radio heads by achieving an optimal downlink sum-rate, while [45] offered an optimal solution based on reinforcement learning to balance the load among evolved nodes for the arrival of machine-type communication devices. To reduce latency, soft resource reservation mechanism is proposed in [46] for uplink scheduling. The authors of [47] presented an algorithm that works with the smooth handover scheme and suggested scheduling policies to ease the user mobility challenge and reduce the application response time. Radio resource allocation strategies to optimize spectral efficiency and energy efficiency while maintaining a low latency in F-RAN are proposed in [48].

With regard to learning for IoT, [49] provided a comprehensive study about the advantages, limitations, applications, and key results relating to machine learning, sequential learning, and reinforcement learning. Multi-agent reinforcement learning was exploited in [50] to maximize network resource utilization in heterogeneous networks by selecting the radio access technology and allocating resources for individual users. The model-free reinforcement learning approach is used in [51] to learn the optimal policy for user scheduling in heterogeneous networks to maximize the network energy efficiency. Resource allocation in non-orthogonal-multiple-access based F-RAN architecture with selective interference cancellation is investigated in [52] to maximize the spectral efficiency while considering the co-channel interference. With the help of task scheduler, resource selector, and history analyzer, [53] introduced an FN resource selection algorithm in which the selection and allocation of the best FN to execute an IoT task depends on the predicted run-time, where stored execution logs for historical performance data of FNs provide realistic estimation of it.

A comprehensive study of network slicing in 5G system is considered in [36, 54]. Issues and challenges of network slicing in Fog RAN is investigated in [36], where authors presented key techniques and solutions in regards to radio and cache resource management as well as social-aware slicing. [54] provides a comprehensive survey on network slicing which embraces the key principles, enabling technologies, challenges, standardization, and solutions including slicing solutions for 5G system, and illustrates the requirements and diverse use cases of network slicing considering RAN sharing, end-to-end orchestration and management involving the radio access, transport and core networks. Radio resource allocation for different network slices is exploited in [55–57] to support various quality-of-service (QoS) requirements and minimize the queuing delay for low latency requests, in which network is logically partitioned into a high-transmission-rate slice for mobile broadband (MBB) applications, and a low-latency slice which supports ultra-reliable low-latency communication (URLLC) applications. While [55] focuses on efficiently allocating radio resources and satisfying various QoS requirements, [56] investigates a joint radio and caching resource allocation problem.The authors in [57] proposed a hierarchical radio resource allocation architecture for network slicing in which a global radio resource manager (GRRM) allocates subchannels to local radio resource managers (LRRMs) in slices, which then assign resources to their end users. Two-level resource management in C-RAN is explored in [58, 59]: an upper level for allocating fronthaul capacity and computing resources of C-RAN among multiple wireless operators, and a lower level for controlling the allocation of C-RAN radio resources to individual operators.

Reinforcement learning (RL) is embraced as a powerful tool to deal with dynamic network slicing for adaptive resource allocation in F-RAN. In [1–3], the RL methods of Q-learning (QL), Monte Carlo, SARSA, expected SARSA, and dynamic programming are utilized to learn the optimal resource allocation policy for a single fog node. The work [60] follows the problem formulation in [3] with an extension to spectrum sharing between 5G users and incumbent users. As the complexity of the control problem increases with more fog nodes, deep RL (DRL), which integrates deep neural networks (DNN) with RL, is more advantageous to cope with the large state and action spaces [61].

Applying DRL as a solution for core network slicing is investigated in [62, 63]. In [62], a particular scenario with three service types (VoIP, video, URLLC) and hundred users is considered. Resource reconfiguration of core network slices is studied in [63] with the aim of minimizing long-term resource consumption. DRL-based centralized agent for C-RAN slicing is investigated in [64–68]. In [64], Deep Q-network (DQN) is utilized by a cloud server to optimally manage centralized caching and radio resources and support two transmission-mode network slices, hotspot slice which supports high-transmission-rate users for MBB applications, and vehicle-to-infrastructure slice for delay-guaranteed transmission. For better radio resource management, a DRL agent is used as a slice manager in [66] to schedule users into three slice types, best effort rate, constant bit rate, and minimum bit rate slices. In [67], a DRL agent at a centralized controller manages the allocation of shared radio resources (bandwidth) among multiple base stations and different network slices (VoLTE, eMBB, and URLLC) to maximize spectrum efficiency and service level agreements. A general network slicing model is proposed by [68] in which an owner of a network provides physical resources to tenants (service providers) to meet the service demand of their end users. With the aim of maintaining high quality of service and maximizing the long-term revenue of service providers through minimizing the reconfiguration cost, a centralized DRL agent reconfigures the allocated resources for two network slices, eMBB and URLLC. Single base station slicing model is considered in [65, 69–72]. C-RAN with single base station is investigated in [65], where Generative-adversarialnetwork distributed DQN (GAN-DDQN) is examined for dynamic bandwidth slicing among network slices, each of which supports users of a particular service type. The dependency of radio resource allocation and the number of slices supported by a single BS is studied by [69], in which distributed DRL is utilized to achieve optimal and flexible radio resource allocation regardless of the number of slices. The work in [70] follows [62] where a single base station provides three service types (video,VoLTE, URLLC) and decides the bandwidth to allocate for each user request. In a vehicular network, [71] employs a DRL agent at a single base station to allocate radio resources for users that belong to four slices, cellular high-definition television slice, cellular ultra-low latency slice, and two device-to-device slices, one from each cellular slice. In [72], a DRL agent at a single base station is exploited to allocate uplink bandwidth to mobile users from various slices with the aim of maximizing uplink throughput and minimizing energy cost. [73] utilizes DQN to dynamically select the best slicing configuration in WiFi networks. DRL slicing in a hierarchical network architecture for dynamic resource reservation is studied in [74], in which the infrastructure provider assigns unutilized resources to network slices maintaining a minimum resource requirement and demand in each slice, where DRL then is employed to efficiently manage reserved resources and maximize QoS.

However, the resource allocation problem considered in the network slicing literature focuses on the dynamics of resource allocation among various network slices and layers, i.e., decides on allocation of resources between FNs for URLLC applications and RRUs for MBB applications while it is infeasible for mobile operators and service providers to keep changing the distribution of resources in the network due to the huge accompanying operational expenditure (OPEX) to cover all required hardware, software and license swaps as well as the implementation of any necessary frequency reuse plans and fronthaul links capacity upgrade, and the impact of outages and prolonged testing, optimizing and fine tuning the network performance follow every single change. Hence, deciding on resource allocation among network slices should be so thoughtful and deliberate, and only after assuring that each slice utilizes its allocated resources efficiently. In this dissertation, we zoom in to the allocated limited resources to FNs to optimize and guarantee their efficient utilization.

1.4 Research Gaps and Contributions of this Dissertation

Despite the growing literature, still there exists a significant research gap: adaptively satisfying the QoS requirements of the URLLC applications while efficiently utilizing the local resources in F-RAN. Motivated by this problem, we provide a novel network slicing framework for sequentially allocating the FNs' limited computing and processing resources at the network edge to various IoT, vehicular, and smart city applications with heterogeneous latency needs. In this dissertation, we develop Markov Decision Process (MDP) formulations for finite-horizon and infinite-horizon scenarios for the considered resource allocation problem. We provide analytical solution for the finite-horizon MDP using dynamic programming. For the infinite-horizon MDP, we consider two cases: (i) a single uncoordinated FN, for which we exploit tabular RL algorithms like Monte Carlo, SARSA, Expected SARSA, and Q-Learning for learning the optimum decision-making policy adaptive to the IoT environment. (ii) an edge cluster of coordinating FNs with an edge controller, where a DRL-based network slicing technique is proposed to ensure the efficient utilization of the edge computing resources in dynamic IoV and smart city traffic profiles and task loads. Specifically, the DRL-based network slicing technique in this dissertation contributes to resolving the following limitations of the existing network slicing works.

- Firstly, an uncoordinated RL-based network slicing and a single fog node slicing model as in [60, 69–72] are not an ideal network slicing approach for 5G and future technologies. Especially in dynamic environments, coordination among fog nodes is needed for more efficient utilization of edge resources while satisfying the QoS needs of users. In this dissertation, we present a coordinated network slicing model based on multiple fog nodes cooperating through an edge controller.

- Secondly, a centralized RL-based cloud controller for an entire network to manage resource allocation among various network slices as in [58, 59, 63–68] will have latency limitations, especially for URLLC-based IoV, V2X, and smart city applications, such as autonomous driving. Whereas, distributed and independent edge controllers (ECs), which are fog nodes that serve as cluster heads, as proposed in this dissertation can avoid large transmission delays and satisfy the desired level of QoS at FNs by making local real-time decisions for the received service requests in a cluster.

- Thirdly, a fixed RL-based network slicing approach as in [55–57] with dedicated fog access point resources for the URLLC slice and dedicated remote radio heads for the MBB slice can cause inefficient utilization of edge resources, especially in a dynamic environment. The nature of many smart city and IoV applications (e.g., autonomous vehicles) requires continuous edge capabilities everywhere in the service area, hence radio, caching and computing resources need to be available at the edge. In practice, the population of delay-sensitive and high-data-rate services dynamically varies over time, and as a result a fixed URLLC or MBB slice will be underutilized during light demand for low-latency or high-speed services, respectively. A more flexible network slicing method as proposed in this dissertation would smartly adapt to the environment. We provide finite- and infinite-horizon Markov decision process (MDP) formulations and RL algorithms to adaptively learn the optimal network slicing policy by closely interacting with the IoT and smart city environment.

- Lastly, a hard RL-based network slicing and hierarchical slicing architecture as in [57, 62, 63, 68, 74] require frequent physical shifting of resources, and hence cannot address dynamic environments with changing demands in a cost-efficient manner. With hard slicing, it will be costly and impractical for cellular operators and service providers to keep adding and transferring further infrastructural assets, i.e., capital expenditure which includes transceivers (TRX) and other radio resources, computing and signal processing resources such as, BBUs, CPUs and GPUs, as well as caching resources and storage data centers. Such major network changes could be considered as part of network expansion plans over time. In this dissertation, we propose a cost-efficient, soft (i.e., virtual) network slicing method in F-RAN. We present extensive simulation results to examine the performance and adaptivity of the proposed RL-based network slicing methods in diverse intelligent vehicular systems and smart city environments and different performance objectives.

1.5 Dissertation Organization

This dissertation is comprised of five chapters, including the current chapter, which introduces the dissertation topics, C-RAN and F-RAN, network slicing for heterogeneous IoT demands, related work, research gaps and dissertation contributions. The rest of the dissertation chapters are organized as follow.

Chapter 2 considers a single non coordinating fog node and proposes a finite-horizon Markov Decision Process (MDP) formulation for the network slicing problem of allocation the limited resource at the single FN for IoT services with heterogeneous latency requirements. The optimum solution (decision policy) for the finite-horizon MDP problem is provided using dynamic programming.

In Chapter 3, provides infinite-horizon MDP formulations for the considered network slicing problem for a single uncoordinated FN. Diverse RL tabular methods, specifically, Monte Carlo, SARSA, Expected SARSA, and Q-learning (QL) are investigated for learning the optimum decision-making policies adaptive to the IoT environment.

Chapter 4 considers the network slicing problem of allocating the limited resources at the network edge (fog nodes) to vehicular and smart city users with heterogeneous latency and computing demands in dynamic environments. A network slicing model is developed based on a cluster of fog nodes (FNs) coordinated with an edge controller (EC) to efficiently utilize the limited resources at the network edge. The network slicing problem is formulated as infinite-horizon MDP and a deep reinforcement learning (DRL) solution to adaptively learn the optimal slicing policy is proposed.

Finally, Chapter 5 summarizes this dissertation and presents the recommendations for future research directions.

1.6 Publications Resulting from this Dissertation

The following publications are associated with this dissertation.

1. Almuthanna Nassar, and Yasin Yilmaz, "Deep Reinforcement Learning for Adaptive Network Slicing in 5G for Intelligent Vehicular Systems and Smart Cities," In review, submitted to *IEEE Internet of Things Journal*, Dec. 2021, revised, Mar. 2021. arXiv preprint arXiv:2010.09916.

2. Almuthanna Nassar, and Yasin Yilmaz, "Dynamic Network Slicing for Fog Radio Access Networks," The 7th IEEE Global Conf. on Signal and Information Processing (GlobalSIP2019), Ottawa, Canada, Nov. 2019.

3. Almuthanna Nassar, and Yasin Yilmaz, "Reinforcement Learning for Adaptive Resource Allocation in Fog RAN for IoT with Heterogeneous Latency Requirements," *IEEE Access*, Sep. 2019.

4. Almuthanna Nassar, and Yasin Yilmaz, "Resource Allocation in Fog RAN for Heterogeneous IoT Environments based on Reinforcement Learning," 2019 IEEE International Conf. on Communications (ICC2019), Shanghai, China, May 2019.

Chapter 2: Dynamic Programming for Adaptive Network Slicing

Fog radio access network (F-RAN) has been recently proposed to satisfy the quality-of-service (QoS) requirements of the ultra-reliable-low-latency-communication (URLLC) IoT applications, hence fog nodes are empowered with computing and storage resources to independently deliver network functionalities at the edge of network without referring the users to the cloud.[2] However, due to their limited resources, fog nodes should utilize their resources intelligently for low latency IoT applications to leverage the complementarity with cloud computing. We consider the problem of sequentially allocating fog node's limited resources to various IoT applications with heterogeneous latency needs. We formulate the problem as a finite-horizon Markov Decision Process (MDP), and present the optimal solution, known as the optimal policy, through dynamic programming. The fog node learns the optimal policy through interaction with the IoT environment, which enables adaptive resource allocation in different IoT environments. Comprehensive simulation results for various IoT environments corroborate the theoretical basis of the proposed MDP method.

2.1 System Model

We consider the F-RAN structure shown in Fig. 2.1, in which FNs are connected through the fronthaul to the cloud controller (CC), where a massive computing capability, centralized baseband units (BBUs) and cloud storage pooling are available. To ease the burden on the fronthaul and the

[2]Portions of this chapter were published in IEEE GlobalSIP [1] and IEEE Access [3]. Copyright permissions from the publishers are included in Appendix A.

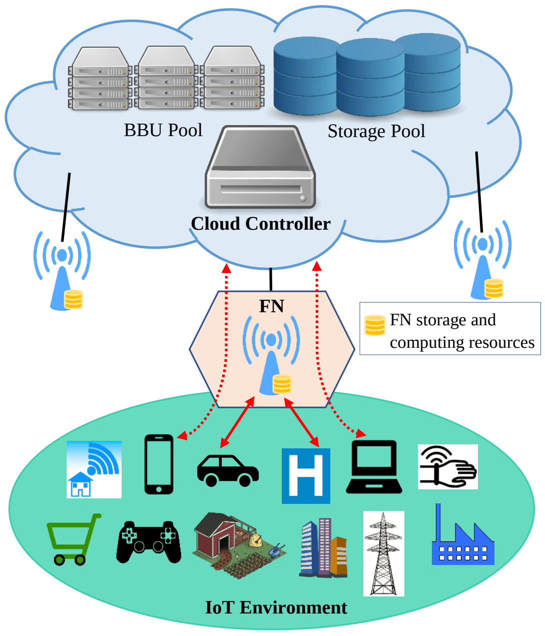


Figure 2.1: Fog-RAN system model. The FN serves heterogeneous latency needs in the IoT environment, and is connected to the cloud through the fronthaul links represented by solid lines. Solid red arrows represent local service by FN to satisfy low-latency requirements, and dashed arrows represent referral to the cloud to save FN's limited resources.

cloud, and to overcome the challenge of the increasing number of IoT devices and low-latency applications, FNs are empowered with capability to deliver network functionalities at the edge. Hence, they are equipped with caching capacity, computing and signal processing capabilities. However, these resources are limited, and therefore need to be utilized efficiently. An end user attempts to access the network by sending a request to the nearest FN. The FN takes a decision whether to serve the user locally at the edge using its own computing and processing resources or refer it to the cloud. We consider the FN's computing and processing capacity to be limited to N resource blocks (RBs). User requests arrive sequentially and decisions are taken quickly, so no queuing occurs.

The QoS requirements of a wireless user are typically given by the latency requirement and throughput requirement. IoT applications have various levels of latency requirement, hence it is sensible for the FN to give higher priority for serving the low-latency applications. To differentiate between similar latency requirements we also consider the risk of failing to satisfy the throughput requirement. This risk is related to the ratio of the achievable throughput to the throughput requirement. The achievable throughput is characterized by the signal-to-noise ratio (SNR) through Shannon channel capacity. Shannon's fundamental limit on the capacity of a communications channel gives an upper bound for the achievable throughput, as a function of available bandwidth (B) in Hz and SNR in dB, $C = B \log_2 (1 + \text{SNR})$. Hence, we define the utility of an IoT user request to be a function of latency requirement, l (in milliseconds), throughput requirement, ω (in bits per second), and channel capacity, C (in bits per second), i.e., $u = f(l, \omega, C)$. Since the utility should be inversely proportional to the latency requirement, and directly proportional to the achievable throughput ratio, $\mu = C/\omega$, we define utility as

$$u = \kappa(\mu^{\zeta}/l^{\beta}), \tag{2.1}$$

where $\kappa, \zeta, \beta > 0$ are mapping parameters. This provides a flexible model for utility. By selecting the parameters κ, ζ, β a desired range of u and importance levels for latency and throughput requirements can be obtained. Since F-RAN is intended for satisfying low-latency requirements, typically, more weight should be given to latency by choosing larger β values.

FNs should be smart to learn how to decide (serve/refer to the cloud) for each IoT request (i.e., how to allocate its limited resources), so as to achieve the conflicting objectives of maximizing the average total utility of served users over time and minimizing its idle (no-service) time. The system objective can be stated as a constrained optimization problem,

$$\begin{gathered}\max_{a_0,\ldots,a_{T-1}} \sum_{t=0}^{T} \mathbb{1}_{\{a_t=serve\}} u_t \quad \text{and} \quad \min_{a_0,\ldots,a_{T-1}} \sum_{t=0}^{T} \mathbb{1}_{\{a_t=reject\}} \\ \text{subject to} \quad \sum_{t=0}^{T} \mathbb{1}_{\{a_t=serve\}} = N, \end{gathered} \tag{2.2}$$

where a_t denotes the action taken at time t (either serves the request locally or rejects it and refers to cloud), T denotes the termination time when all RBs are filled, N denotes the number of RBs, and $\mathbb{1}_{\{\cdot\}}$ is the indicator function taking value 1 if its argument is true and 0 if false. The goal is to find the optimum decision policy $\{a_0, a_1, \ldots, a_{T-1}\}$ for an IoT environment which randomly generates $\{u_t\}$. Note that the final decision is always $a_T = serve$ by definition, hence omitted in the policy representation.

One straightforward approach to deal with this resource allocation problem is to apply network slicing [36, 54] based on the user utility, in which the network is logically partitioned into two slices [55–57], a fog slice handles high-utility IoT requests of low-latency demand, and cloud slice considers low-utility users. Hence, a filtering standard is required for the FN to direct users' requests to their corresponding network slices. For instance, we can define a threshold rule, such as "serve locally if $u > 5$", if we classify all applications in an IoT environment into ten different utilities $u \in \{1, 2, ..., 10\}$, 10 being the highest utility. However, such a policy is sub-optimum since the FN will be waiting for a user to satisfy the threshold, which will increase the FN's idle time and make the CC busier. The main drawback of this policy is that it cannot adapt to the dynamic IoT environment to achieve the objective. For instance, when the user utilities are almost uniformly distributed, a very selective policy with a high threshold will stay idle most of the time, whereas an impatient policy with a low threshold will in general obtain a low average served utility.

A mild policy with threshold 5 may in general perform better than the extreme policies, yet it will not be able adapt to different IoT environments. A better solution for the F-RAN resource allocation problem is to use RL techniques which can continuously learn the environment and adapt the decision rule accordingly.

2.2 Finite-Horizon MDP Formulation

We formulate the F-RAN resource allocation problem in the form of finite-horizon Markov decision process (MDP). In finite-horizon MDP, there is a hard constraint on the FN in terms of the value of time as it must terminate within a limited time T_f regardless whether the N slots are filled or not. This means that the MDP will terminate either at the termination time T_f or before if all slots are filled earlier. The utility of a user, denoted with u, has the same value as user's priority level, e.g., $u \in \{1, 2, 3, ..., 10\}$. We consider that the FN has N slots of serving resources. The state of FN is denoted by $s \in \{s_{t,n}\}$, where t and n represent the time and the number of occupied slots, respectively. We start at $t = 0$ with all slots available, thus the initial state is $s_{0,0}$. The finite-horizon MDP has multiple terminal states. All the FN states $\{s_{T_f,n} : n = 0, 1, ..., N\}$ and $\{s_{t,N} : t = N, N+1, ..., T_f\}$ are terminal states. For a request from a user with utility u_t, at time t, if the FN decides to take the action $a_t = serve$, which means to serve the user at the edge, then it will gain its utility value as a reward $r_t = u_t$, and one slot of the FN's resources will be occupied. Otherwise, for the action $a_t = wait$, which means to refer the current user to the cloud and wait for a better future utility, the FN will maintain its resources but it will get a reward $r_t = -\eta$, where η is the penalty of waiting, whose role is to encourage less idle time. We define the state s of the FN at any time as the number of available slots at that time, where future state is independent of past states given the current state, i.e., Markov state. At every time step t, the FN receives a request from a user of utility u_t, and FN takes an action $a_t \in \{serve, wait\}$. Based on its decision, the FN receives an immediate reward $r_t \in \{u_t, -\eta\}$, and moves to a successor state s'. We define the return G_t as the total discounted reward received from time t till the termination time which is limited by T_f, $G_t = \sum_{j=0}^{T-1} \gamma^j r_{t+j}$, where $\gamma \in [0, 1]$ is the discount factor which represents the importance of future rewards with respect to the immediate reward, and after termination $r = 0$. $\gamma = 0$ ignores future rewards, whereas $\gamma = 1$ means that future rewards are of the same importance as the immediate rewards.

Starting at the initial state $s = s_{0,0}$, the objective is to find the optimal decision policy which maximizes the expected initial return $\mathbb{E}[G_0]$. The state-value function $V(s_{0,0})$, where $V(s)$ is shown in (2.3), is equal to the objective function $\mathbb{E}[G_0]$. Similarly, we define the action-value function $Q(s, a)$ as the expected return that can be achieved after taking the action a at state s, as shown in (2.4). The expressions in (2.3) and (2.4) are known as the Bellman expectation equations for state value and action value, respectively [75],

$$V(s) = \mathbb{E}[G_t|s] = \mathbb{E}[r_t + \gamma V(s')|s], \tag{2.3}$$

$$Q(s, a) = \mathbb{E}[G_t|s, a] = \mathbb{E}[r_t + \gamma Q(s', a')|s, a], \tag{2.4}$$

where a' denotes the successor action at the successor state s'.

The value of state $s_{0,0}$ is the expected return considering all dynamics and episodes, i.e., $V(s_{0,0} = \mathbb{E}[G_0]$. The objective of the FN is to utilize the N resource slots for high priority IoT applications in a timely manner. This can be done through maximizing the value of initial state

$V(s_{0.0})$. To achieve this objective an optimal decision policy is required, which is discussed in the following section.

2.3 Optimal Policy

A decision-making policy π is a way of selecting actions. It can be defined as the set of probabilities of taking a particular action given the state, i.e., $\pi = \{P(a|s)\}$ for all possible state-action pairs. The policy π is said to be optimal if it maximizes the value of all states, i.e., $\pi^* = \arg\max_{\pi} V_\pi(s), \forall s$. Hence, to solve the considered MDP problem, the FN needs to find the optimal policy through finding the optimal state-value function $V^*(s) = \max_{\pi} V_\pi(s)$, which is similar to finding the optimal action-value function $Q^*(s,a) = \max_{\pi} Q_\pi(s,a)$ for all state-action pairs. From (2.3) and (2.4), we can write the Bellman optimality equations for $V^*(s)$ and $Q^*(s,a)$ as,

$$V^*(s) = \max_{a\in\mathcal{A}} Q^*(s,a) = \max_{a\in\mathcal{A}} \mathbb{E}[r_t + \gamma V^*(s')|s,a], \tag{2.5}$$

$$Q^*(s,a) = \mathbb{E}[r_t + \gamma \max_{a'\in\mathcal{A}} Q^*(s',a')|s,a], \tag{2.6}$$

where a' denotes the action at $t+1$ and $\mathcal{A}$ is the set of all possible actions. Since the goal of maximizing the expected future rewards is already taken care of by the optimal value of the successor state, $V^*(s')$ can be taken out of the expectation in (2.5). Hence, the optimal policy is given by the best local action at each state,

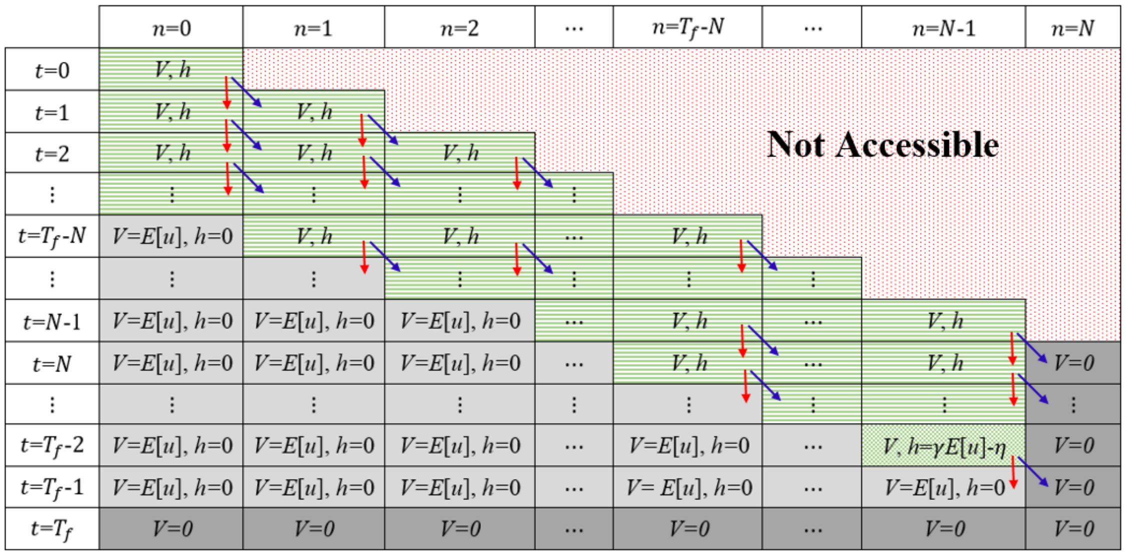
$$a^* = \arg\max_{a\in\mathcal{A}} Q^*(s,a) = \arg\max_{a\in\mathcal{A}} \mathbb{E}[r_t + \gamma V^*(s')|s,a]. \tag{2.7}$$

This solution approach is known as Dynamic Programming.

In our problem, first the user arrives, then we make a decision to serve or wait (refer to the cloud), meaning that the reward u for serving and the reward $-\eta$ for waiting are known at the time of decision making. Thus, from (2.7), the optimal action at state $S_{t,n}$ is given by

$$a^*_{t,n} = \begin{cases} Serve \text{ if } u > h_{t,n}, \\ \\ Wait \text{ otherwise}, \end{cases} \tag{2.8}$$

where $h_{t,n} = \gamma[V^*(s_{t+1,n}) - V^*(s_{t+1,n+1})] - \eta$. Since the number of states is finite in the finite-horizon case, we can use the backward induction technique to compute the optimal thresholds $\{h_{t,n}\}$ assuming some training data $\{u_t\}$ is available to learn some key statistics of the IoT environment. Starting with the terminal states, which have zero value, we can compute the optimal state values and consequently the optimal thresholds for all states by moving backwards. Firstly, note that not all states $\{s_{t,n}\}$ are accessible for all t and n. Even if one slot is filled at each t, the states with $n > t$ are not accessible, as shown in Fig. 2.2. Secondly, note that there are $T_f + 1$ terminal states with zero value ($T_f - N$ from early stopping with $T < T_f$ and $N + 1$ from $T = T_f$, $n = 0, 1, ..., N$), which do not require threshold, as shown with dark gray in Fig. 2.2. Next, note that for all the non-terminal states at time $T_f - 1$, both *serve* and *wait* actions result in a terminal state with zero value, thus the decision is made based on only the immediate rewards (u vs. $-\eta$). That is, at those states the optimal action is always *serve*, hence the threshold on u is zero and the state value is $\mathbb{E}[u]$, as shown in Fig. 2.2. Similarly, for $t = T_f - N, ..., T_f - 2$, there is a number of non-terminal states for which both actions yield the same future value, hence have zero threshold and value $\mathbb{E}[u]$. Specifically, the states $\{s_{t,n} : t = T_f - l, n = 0, ..., N - l, l = 1, ..., N\}$ have state value $\mathbb{E}[u]$ and threshold 0, as shown with light gray in Fig. 2.2. Finally, for the $(T_f - N)N$ remaining states in a diagonal



	n=0	n=1	n=2	⋯	n=T_f-N	⋯	n=N-1	n=N
t=0	V, h							
t=1	V, h	V, h						
t=2	V, h	V, h	V, h				Not Accessible	
⋮	⋮	⋮	⋮	⋮				
t=T_f-N	V=$E[u]$, h=0	V, h	V, h	⋯	V, h			
⋮	⋮	⋮	⋮	⋮	⋮	⋮		
t=N-1	V=$E[u]$, h=0	V=$E[u]$, h=0	V=$E[u]$, h=0	⋯	V, h	⋯	V, h	
t=N	V=$E[u]$, h=0	V=$E[u]$, h=0	V=$E[u]$, h=0	⋯	V, h	⋯	V, h	V=0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
t=T_f-2	V=$E[u]$, h=0	V=$E[u]$, h=0	V=$E[u]$, h=0	⋯	V=$E[u]$, h=0	⋯	V, h=$\gamma E[u]$-η	V=0
t=T_f-1	V=$E[u]$, h=0	V=$E[u]$, h=0	V=$E[u]$, h=0	⋯	V= $E[u]$, h=0	⋯	V=$E[u]$, h=0	V=0
t=T_f	V=0	V=0	V=0	⋯	V=0	⋯	V=0	V=0

Figure 2.2: FH state values and thresholds that need to be computed via backward induction (green diagonal band). Start with the farthest state $s_{T_f-2,N-1}$ (checked green box) and traverse backwards the diagonal band until the initial state $s_{0,0}$ (see Algorithm 1). The terminal states (dark gray), the trivial states whose optimal action is always serve (light gray), and the not accessible states (red dotted) are also shown.

band, shown with green in Fig. 2.2, the state values and the corresponding thresholds need to be computed backwards starting with the farthest state $s_{T_f-2,N-1}$ from the initial state $s_{0,0}$. The total reward is u if served, whereas it is $\gamma\mathbb{E}[u]-\eta$ if waited, giving the threshold $h_{T_f-2,N-1}=\gamma\mathbb{E}[u]-\eta$, as shown by the checked green box in Fig. 2.2. Then, its state value is written as

$$\begin{aligned} V(s_{T_f-2,N-1}) &= P(u>h_{T_f-2,N-1})\mathbb{E}[u|u>h_{T_f-2,N-1}] \\ &\quad +\{1-P(u>h_{T_f-2,N-1})\}\{\gamma\mathbb{E}[u]-\eta\}, \end{aligned} \tag{2.9}$$

where the first and second terms correspond to the *serve* and *wait* actions, respectively. Note that the probability $P(u>h_{T_f-2,N-1})$ and the expectation $\mathbb{E}[u|u>h_{T_f-2,N-1}]$ can be computed through some observations $\{u\}$ from the IoT environment. With $V(s_{T_f-2,N-1})$ computed, we can now find the threshold for the two undiscovered neighboring states above it, namely $s_{T_f-3,N-2}$ and

$s_{T_f-3,N-1}$ using

$$h_{t,n} = -\eta + \gamma[V^*(s_{t+1,n}) - V^*(s_{t+1,n+1})], \tag{2.10}$$

from (2.8). Then, using the thresholds the state values are computed similarly to (2.9) as follows

$$\begin{aligned} V(s_{t,n}) = P(u > h_{t,n})\{\mathbb{E}[u|u > h_{t,n}] + \gamma V(s_{t+1,n+1})\} \\ +\{1 - P(u > h_{t,n})\}\{\gamma V(s_{t+1,n}) - \eta\}. \end{aligned} \tag{2.11}$$

Algorithm 1 Learning Optimum Policy for FH MDP

```
1: Select: $\gamma \in [0, 1]$, $\eta \in \mathbb{R}$;
2: for $i = N - 1 : -1 : 0$ do
3:    for $j = T_f - 1 : -1 : i$ do
4:       $h_{j,i} = \gamma[V(s_{j+1,i}) - V(s_{j+1,i+1})] - \eta$;
5:       Compute $a \leftarrow \mathbb{E}[u|u > h_{j,i}]$ and $p \leftarrow P(u > h_{j,i})$;
6:       $V(s_{j,i}) = p\{a + \gamma V(s_{j+1,i+1})\}$
         $+ (1 - p)\{\gamma V(s_{j+1,i}) - \eta\}$;
7:    end for
8: end for
9: Return $\{h_{j,i}\}$.
```

In the same way, by computing first the threshold and then the state value via (2.10) and (2.11), respectively, the remaining states in the diagonal band are traversed backwards until the initial state $s_{0,0}$. The key statistics $P(u > h_{t,n})$ and $\mathbb{E}[u|u > h_{t,n}]$ are to be found from the IoT environment. The procedure for finding the optimal policy is summarized in Algorithm 1, where by default the terminal states have zero value, and $x : -1 : y$ denotes the decrement by 1 from x to y. For notational simplicity, the trivial states are also included in the loops at lines 2 and 3. The range for the loops can be modified to exclude the trivial states.

Recall that the FN objective is to maximize the expected total served utility and minimize the expected termination time. Hence, to compare the performance of dynamic network slicing provided in Algorithm 1 with the performance of a threshold-based static network slicing, which does not learn from the interactions with environment, we define an objective performance metric R as

$$R = \mathbb{E}\left[\sum_{m=1}^{M} u_m - \theta(T - M)\right], \tag{2.12}$$

where a served utility is denoted with u_m, the number of served IoT requests in an episode is denoted with M, $(T - M)$ represents the total idle time, and θ is a penalty for being idle.

2.4 Simulations

We next provide simulation results to compare the performance of the FN when implementing the MDP-based method, given in Algorithm 1, with that when a threshold-based slicing is employed. We consider that the FN is empowered with computing and storage resources of five slots, i.e., $N = 5$. We evaluate the performances in various IoT environments with different compositions of latency requirements. Specifically, we consider 10 utility classes with different latency requirements to exemplify the variety of IoT applications in an F-RAN setting. By changing the composition of utility classes, we generate 19 scenarios, 6 of which are summarized in Table 2.1. Higher percentages of high-utility users make the IoT environment richer. Denoting an IoT environment of particular statistics with $\mathcal{E}$, in Table 2.1 we show the statistics of $\mathcal{E}_1$, $\mathcal{E}_4$, $\mathcal{E}_7$, $\mathcal{E}_{10}$, $\mathcal{E}_{15}$, and $\mathcal{E}_{19}$. The last two rows in Table 2.1 show the probability ρ of utility being greater than 5, and the expected value of u, respectively. The first 10 rows in the table provide detailed information given by the probability of each utility value in an IoT environment. In the considered 19 scenarios, ρ increases by 0.05 from 5% to 95% for $\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_{19}$ respectively. The remaining 13 scenarios have

Table 2.1: Utility distributions corresponding to a variety of latency requirements of IoT applications in various environments

	$\mathcal{E}_1$	$\mathcal{E}_4$	$\mathcal{E}_7$	$\mathcal{E}_{10}$	$\mathcal{E}_{15}$	$\mathcal{E}_{19}$
$P(u=1)$	0.015	0.012	0.01	0.008	0.004	0.001
$P(u=2)$	0.073	0.062	0.05	0.038	0.019	0.004
$P(u=3)$	0.365	0.308	0.25	0.192	0.096	0.019
$P(u=4)$	0.292	0.246	0.2	0.154	0.077	0.015
$P(u=5)$	0.205	0.172	0.14	0.108	0.054	0.011
$P(u=6)$	0.014	0.057	0.1	0.142	0.214	0.271
$P(u=7)$	0.013	0.051	0.09	0.129	0.193	0.244
$P(u=8)$	0.011	0.046	0.08	0.114	0.171	0.217
$P(u=9)$	0.009	0.034	0.06	0.086	0.129	0.163
$P(u=10)$	0.003	0.012	0.02	0.029	0.043	0.055
$\rho = P(u>5)$	5%	20%	35%	50%	75%	95%
$\bar{u} = \mathbb{E}[u]$	3.82	4.4	4.97	5.55	6.5	7.27

statistics proportional to their ρ values. We started with a general scenario given by $\mathcal{E}_7$ for the following IoT applications: smart farming, smart retail, smart home, wearables, entertainment, smart grid, smart city, industrial Internet, autonomous vehicles, and connected health, which correspond to the utility values $1, 2, ..., 10$, respectively.

We consider the objective performance metric given in (2.12) to compare the performance of the FN with $N = 5$ when applying the proposed dynamic programming algorithm (Algorithm 1) with the threshold-based slicing algorithm. We consider a finite horizon of $T_f = 10$, which serves as a strict termination time. Since we already have a time constraint on the FN which represents the value of time, we consider $\eta = \theta = 0$. As shown in Fig. 2.3, the dynamic network slicing algorithm exhibits the best performance as it adaptively learns how to balance early termination with higher utilities. It never terminates too early or too late ($T \approx 7.6$ for all environments), as opposed to the threshold-based slicing algorithm which is not adaptive to the environment. Dynamic slicing

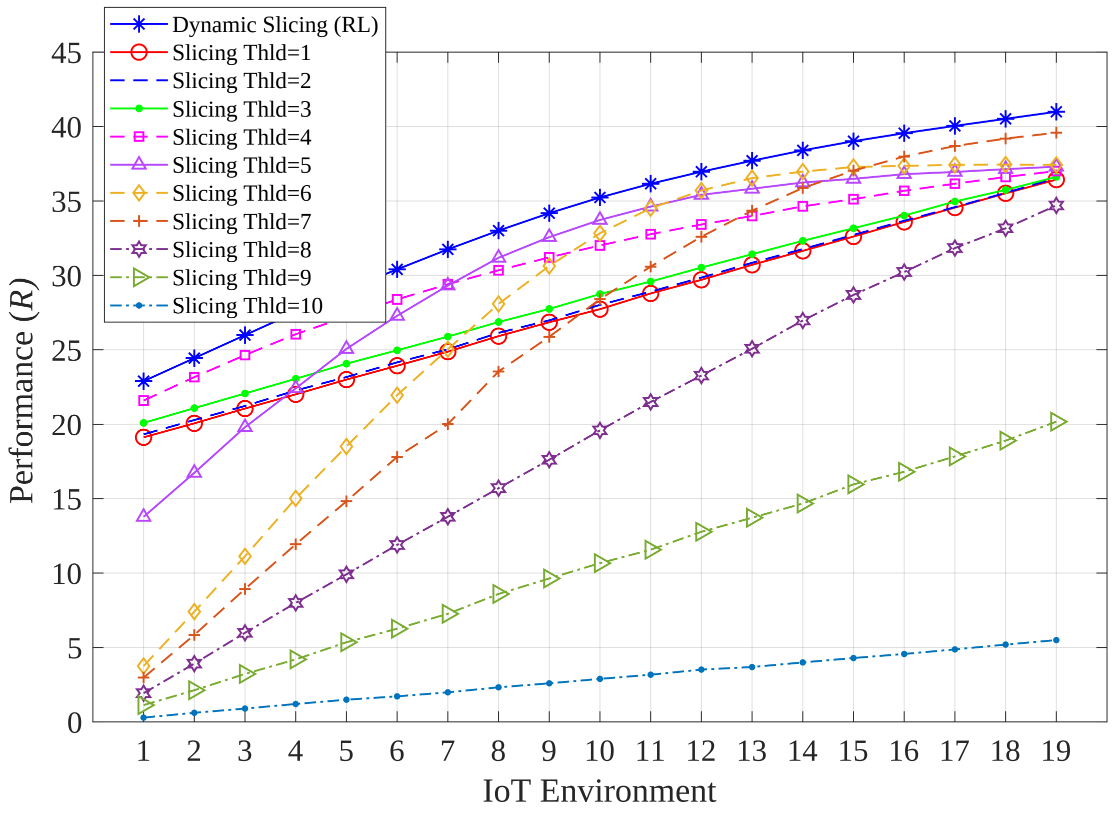


Figure 2.3: Performance of FN with $N = 5$ and $T_f = 10$ in various IoT environments when applying Algorithm 1 with $\eta = 0$, $\gamma = 1$, and the threshold-based network slicing algorithm with different slicing thresholds.

algorithm adaptively learns how to achieve the objective for all IoT environments under a strict termination time constraint.

2.5 Summary

In this chapter, we proposed a finite-horizon Markov Decision Process (MDP) formulation for the resource allocation problem in Fog RAN for IoT services with heterogeneous latency requirements. We provided the optimum solution (decision policy) for the MDP problem using dynamic programming. Various IoT environments with different latency compositions were considered in the simulations to evaluate the performance of the proposed dynamic network slicing approach. The numerical results corroborated the fact that MDP methods adapt to the environment by learning the optimum policy from experience. We showed that the dynamic MDP-based slicing method always dominates the static threshold-based slicing method, which does not learn from the environment.

Chapter 3: Adaptive Network Slicing Using Tabular RL Methods

This chapter considers the F-RAN structure shown in Fig. 2.1 of Chapter 2 and the network slicing problem of sequentially allocating the FN's limited resources to IoT applications of heterogeneous latency requirements.[3] For each access request from an IoT user, the FN needs to decide whether to serve it locally at the edge utilizing its own resources or to refer it to the cloud to conserve its valuable resources for future users of potentially higher utility to the system (i.e., lower latency requirement). We formulate the Fog-RAN resource allocation problem in the form of a Markov decision process (MDP), and employ several reinforcement learning (RL) methods, namely Q-learning, SARSA, Expected SARSA, and Monte Carlo, for solving the MDP problem by learning the optimum decision-making policies. We verify the performance and adaptivity of the RL methods and compare it with the performance of the network slicing approach with various slicing thresholds. Extensive simulation results considering various IoT environments of heterogeneous latency requirements corroborate that RL methods always achieve the best possible performance regardless of the IoT environment.

3.1 Infinite-Horizon MDP Formulation

RL can be thought as the third paradigm of machine learning in addition to the other two paradigms, supervised learning and unsupervised learning. The key point in the proposed

[3]Portions of this chapter were published in IEEE ICC [2] and IEEE Access [3]. Copyright permissions from the publishers are included in Appendix A.

RL approach is that FN learns about the IoT environment by interaction and then adapts to it. FN gains rewards from the environment for every action it takes, and once the optimum policy of actions is learned, FN will be able to maximize its expected cumulative rewards, adapt to the IoT environment, and achieve the objective.

For an access request from a user with utility u_t, at time t, if the FN decides to take the action $a_t = serve$, which means to serve the user at the edge, then it will gain an immediate reward r_t and one of the RBs will be occupied. Otherwise, for the action $a_t = reject$, which means to reject serving the user at the edge and refer it to the cloud, the FN will maintain its available RBs and get a reward r_t. The value of r_t depends on a_t and u_t. For tractability, we consider quantized utility values, $u_t \in \{1, 2, \dots, U\}$.

We define the state s_t of the FN at any time t as

$$s_t = 10\, b_t + u_t, \tag{3.1}$$

where $b_t \in \{0, 1, 2, \dots, N\}$ is the number of occupied RBs at time t. Note that the successor state s_{t+1} depends only on the current state s_t, the utility u_{t+1} of the next service request, and the action taken (*serve* or *reject*), satisfying the Markov property $P(s_{t+1}|s_0, ..., s_{t-2}, s_{t-1}, s_t, a_t) = P(s_{t+1}|s_t, a_t)$, i.e., Markov state. Hence, we formulate the Fog-RAN resource allocation problem in the form of a Markov decision process (MDP), which is defined by the tuple $(\mathcal{S}, \mathcal{A}, P^a_{ss'}, R^a_{ss'})$, where $\mathcal{S}$ is the set of all possible states, i.e., $s_t \in \mathcal{S}$, $\mathcal{A}$ is the set of actions, i.e., $a_t \in \mathcal{A} = \{serve, reject\}$, $P^a_{ss'}$ is the transition probability from state s to s' when the action a is taken, i.e., $P^a_{ss'} = P(s'|s, a)$, where s' is a shorthand notation for the successor state, and $R^a_{ss'}$ is the immediate reward received when the action a is taken at state s which ends up in state s', e.g., $r_t = R^{a_t}_{s_t s_{t+1}} \in \mathcal{R}$. The return

G_t is defined as the cumulative discounted rewards received from time t onward and given by

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{j=0}^{\infty} \gamma^j r_{t+j}, \tag{3.2}$$

where $\gamma \in [0, 1]$ is the discount factor. γ represents the weight of future rewards with respect to the immediate reward, $\gamma = 0$ ignores future rewards, whereas $\gamma = 1$ means that future rewards are of the same importance as the immediate rewards. The objective of the MDP problem is to maximize the expected initial return $\mathbb{E}[G_0]$.

In the presented MDP, for an FN that has a computing and processing capacity of N RBs, there are $U(N + 1)$ states, $s_t \in \mathcal{S} = \{1, 2, 3, \ldots, U(N + 1)\}$, where U is the greatest discrete utility level. At the initiation time $t = 0$, all RBs are available, i.e., $b = 0$, hence from (3.1), there are U possible initial states $s_0 \in \{1, 2, \ldots, U\}$ dependent on u_0. The MDP terminates at time T when all RBs are occupied, i.e., $b_T = N$, hence similarly there are U terminal states $s_T \in \{UN + 1, UN + 2, \ldots, U(N + 1)\}$. Note that a policy treating the MDP problem can continue operating after T as in-use RBs become available in time by taking actions similarly to its operation before T.

The reward mechanism $R^a_{ss'}$ is typically chosen by the system designer according to the objective. We propose a reward mechanism based on the received utility and the action taken for it. Specifically, at time t, based on u_t and a_t, the FN receives an immediate reward $r_t \in \mathcal{R} = \{r_{sh}, r_{sl}, r_{rh}, r_{rl}\}$, and moves to the successor state s_{t+1}, where r_{sh} is the reward for serving a high-utility request, r_{sl} is the reward for serving a low-utility request, r_{rh} is the reward for rejecting a high-utility request, and r_{rl} is the reward for rejecting a low-utility request. A request is determined as high-utility or low-utility relative to the environment based on a threshold u_h, which is a design parameter dependent on the utility distribution in IoT environment. For instance, u_h can be selected as a certain percentile, such as the 50th percentile, i.e., median, of the utilities in the environment. Hence, the proposed reward function is given by

$$r_t = \begin{cases} r_{sh} & \text{if } a_t = serve, \quad u_t \geq u_h \\ r_{rh} & \text{if } a_t = reject, \ u_t \geq u_h \\ r_{sl} & \text{if } a_t = serve, \quad u_t < u_h \\ r_{rl} & \text{if } a_t = reject, \ u_t < u_h. \end{cases} \tag{3.3}$$

Remark 1: Note that the threshold u_h does not have a definitive meaning with respect to the system requirements, i.e., there is no requirement saying that requests with utility lower/greater than u_h must be rejected/served. The goal here is to introduce an internal reward mechanism for the RL approach to facilitate learning the expected future gains, as will be clear later in this section and the following section. For an effective learning performance, the reward mechanism should be simple enough to guide the RL algorithm towards the system objective (see (2.2)) [75]. That is, its role is not to imitate the system objective closely to make the algorithm achieve it at once, but to resemble it in a simple manner to let the algorithm iteratively achieve a high performance.

Remark 2: Although a threshold u_h is utilized in the proposed reward mechanism, its use is fundamentally different than the utility filtering-based policy in network slicing approach which always accepts/rejects requests with utility greater/lower than a threshold. While the utility filtering-based policy considers only the immediate gain from the current utility, the algorithms tackling the MDP problem, such as the RL algorithms, consider the expected return $\mathbb{E}[G_0]$ which includes the immediate reward and expected future rewards. Hence, the threshold u_h does not

Table 3.1: State transitions of 5-RB FN for a sample of IoT requests and random actions with $U = 10$ and $u_h = 6$

t	u_t	b_t	s_t	a_t	r_t	s_{t+1}
0	5	0	5	*reject*	r_{rl}	9
1	9	0	9	*serve*	r_{sh}	13
2	3	1	13	*reject*	r_{rl}	13
3	3	1	13	*serve*	r_{sl}	28
4	8	2	28	*serve*	r_{sh}	36
5	6	3	36	*reject*	r_{rh}	31
6	1	3	31	*reject*	r_{rl}	40
7	10	3	40	*serve*	r_{sh}	47
8	7	4	47	*reject*	r_{rh}	49
9	9	4	49	*serve*	r_{sh}	54
10	4	5	54	·	·	·

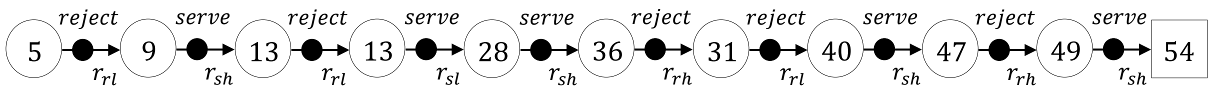


Figure 3.1: State transition graph for the MDP episode given in Table 3.1 for an FN with $N = 5$, $U = 10$, and $u_h = 6$. Non-terminal states and terminal state are represented by circles and squares, respectively, and labeled by the states names. Filled circles represent actions, and arrows show the transitions with corresponding rewards.

necessarily cause the algorithm to accept/reject requests with utility greater/lower than u_h; it only plays an internal role in learning the expected future rewards.

State transitions for an FN with 5 RBs ($N = 5$), 10 utility levels ($U = 10$), and $u_h = 6$, a sample of IoT requests with utilities u_t, and random actions a_t are shown in Table 3.1. At time t, being at state s_t, and taking the action a_t will result in getting an immediate reward r_t and moving to the successor state s_{t+1}. The state transitions in Table 3.1 represent an episode of the MDP, it starts at $t = 0$ and terminates at $T = 10$ with the states $5 \rightarrow 9 \rightarrow 13 \rightarrow 13 \rightarrow 28 \rightarrow 36 \rightarrow 31 \rightarrow 40 \rightarrow 47 \rightarrow 49 \rightarrow 54$. The dynamics of this episode is shown through a state transition graph in

Fig. 3.1, in which non-terminal states and terminal state are represented by circles and squares, respectively, and labeled by the states names, filled circles represent actions, and arrows show the transitions with corresponding rewards.

3.2 Optimal Policies

The state-value function $V(s)$, shown in (2.3), represents the long-term value of being in state s in terms of the expected return which can be collected starting from this state onward till termination. Hence, the terminal state has zero value since no reward can be collected from that state, and the value of initial state is equal to the objective function $\mathbb{E}[G_0]$. The state value can be viewed also in two parts: the immediate reward from the action taken and the discounted value of the successor state where we move to. Similarly, the action-value function $Q(s, a)$ is the expected return that can be achieved after taking the action a at state s, as shown in (2.4). The action value function tells how good it is to take a particular action at a given state. The expressions in (2.3) and (2.4) are known as the Bellman expectation equations for state value and action value, respectively [75], where a' denotes the successor action at the successor state s'.

Since (2.3) shows the relationship between the value of a state and its successor states, similarly for the value of an action in (2.4), it is useful to show the dynamics of the MDP in a backup diagram, as shown in Fig. 3.2 for a 2-RB FN ($N = 2$). The backup diagram provides an overview for the possible episodes of the considered MDP, where the minimum termination time required to reach a terminal state, at which all RBs are occupied ($b = N$), is $T = 2$ through the episode $u_0 \rightarrow 10 + u_1 \rightarrow 20 + u_2$, i.e., *serve* all the time. Note that early termination does not necessarily maximize the return.

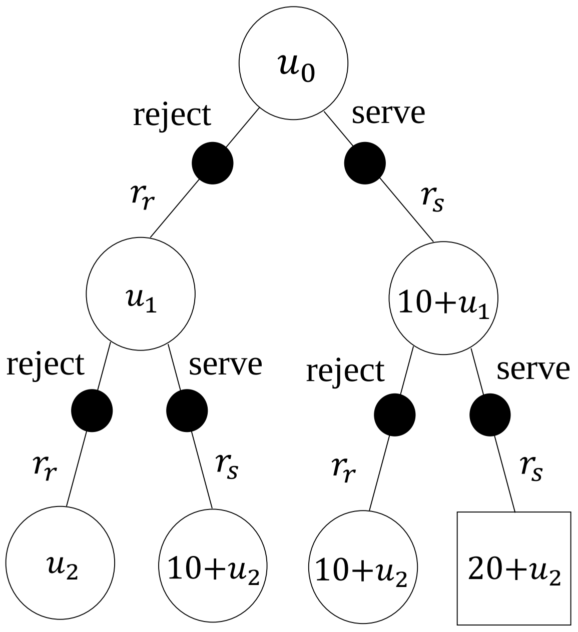


Figure 3.2: The first 3 levels of the backup diagram for the MDP with 2-RB FN ($N = 2$). Non-terminal states and terminal states are represented by open circles and squares, respectively, and labeled by the states according to (3.1). $r_s \in \{r_{sh}, r_{sl}\}$ and $r_r \in \{r_{rh}, r_{rl}\}$ are the rewards for serving and rejecting, respectively, and depend on u_h.

The objective of the FN in the presented MDP is to utilize the N resource blocks for high-utility IoT applications in a timely manner. This can be done through maximizing the value of initial state, which is equal to the MDP objective $\mathbb{E}[G_0]$. To this end, an optimal decision policy is required, which is discussed next.

A policy π is a way of selecting actions. It can be defined as the set of probabilities of taking a particular action given the state, i.e., $\pi = \{P(a|s)\}$ for all possible state-action pairs. The policy π is said to be optimal if it maximizes the value of all states, i.e., $\pi^* = \arg\max_{\pi} V_\pi(s), \forall s$. Hence, to solve the considered MDP problem, the FN needs to find the optimal policy through finding the optimal state-value function $V^*(s) = \max_{\pi} V_\pi(s)$, which is similar to finding the optimal action-value function $Q^*(s,a) = \max_{\pi} Q_\pi(s,a)$ for all state-action pairs. The Bellman optimality equations for $V^*(s)$ and $Q^*(s,a)$ are shown in (2.5) and (2.6), respectively.

The notion of optimal state-value function $V^*(s)$ greatly simplifies the search for optimal policy. Since the goal of maximizing the expected future rewards is already taken care of the optimal value of the successor state, $V^*(s')$ can be taken out of the expectation in (2.5). Hence, the optimal policy is given by the best local actions at each state. Dealing with $Q^*(s,a)$ to choose optimal actions is even easier, because with $Q^*(s,a)$ there is no need for the FN to do the one-step-ahead search and instead it picks the best action that maximizes $Q^*(s,a)$ at each state. Optimal actions are defined as in (2.7).

After discretizing the utility into U levels, the state space becomes tractable with cardinality $|\mathcal{S}| = U(N+1)$, hence in this case the optimal policy can be learned by estimating the optimal value functions (either (2.5) or (2.6)) using tabular methods such as model-free RL methods (e.g., Monte Carlo, SARSA, Expected SARSA, and Q-learning), which are also called approximate dynamic programming methods [75]. Since the expectations involved in value functions are not tractable to find in closed form, we resort to model-free RL methods in this chapter instead of exact dynamic programming. Continuous utility values (see (2.1)) would yield infinite dimensional state space, and thus require function approximation methods, such as deep Q-learning known as DQN [61], for predicting the value function at different states, which we discuss in Chapter 4.

In our MDP problem, firstly FN receives a request from an IoT application of utility u, then it makes a decision to *serve* or *reject*, meaning that the reward for serving $r_s \in \{r_{sh}, r_{sl}\}$ and the reward for rejecting $r_r \in \{r_{rh}, r_{rl}\}$ are known at the time of decision making. Thus, from (2.3) and

Algorithm 2 Learning Optimum Policy using Monte Carlo

```
1: Select: $\gamma \in [0,1]$, $\{u_h, r_{sh}, r_{sl}, r_{rh}, r_{rl}\} \in \mathbb{R}$;
2: Input: $N$ (number of RBs);
3: Initialize: $V(s) \leftarrow 0, \forall s$; *Returns(s)* (an array to save states' returns in all iterations);
4: for $iteration = 0, 1, 2, ...$ do
5:     Initialize: $b \leftarrow 0$;
6:     Generate an episode: Take actions using (3.4) until termination;
7:     $G(s) \leftarrow$ sum of discounted rewards from $s$ till terminal state for all states appearing in the
       episode;
8:     Append $G(s)$ to *Returns(s)*;
9:     $V(s) \leftarrow$ average(Returns(s));
10:    if $V(s)$ converges for all $s$ then
11:        break
12:        $V^*(s) \leftarrow V(s), \forall s$;
13:    end if
14: end for
15: Use the estimated $V^*(s)$ to find optimal actions using (3.4).
```

(2.7), the optimal action at state s is given by

$$a^* = \begin{cases} \textit{serve} \text{ if } r_s + \gamma \mathbb{E}_u[V^*(s'_{serve} = 10(b+1) + u_{t+1})] > r_r + \gamma \mathbb{E}_u[V^*(s'_{reject} = 10b + u_{t+1})], \\ \\ \textit{reject} \text{ otherwise,} \end{cases} \tag{3.4}$$

where s'_{serve} is the successor state when $a = serve$, s'_{reject} is the successor state when $a = reject$, and $\mathbb{E}_u$ is the expectation with respect to the utilities u in the IoT environment.

A popular way to compute the optimal state values, required by the optimal policy as shown in (3.4), is through value iteration by Monte Carlo computations. The procedure to learn the optimal policy from the IoT environment using Monte Carlo is given in Algorithm 2. Given the parameters N, γ, $\{u_h, r_{sh}, r_{sl}, r_{rh}, r_{rl}\}$, and the data of IoT users $\{u_t\}$, Algorithm 2 shows how to learn the optimal policy for the considered MDP problem. Note that $\{u_t\}$ can be real data from the IoT environment, as well as from simulations if the probability distribution is known. The

Returns array at line 2 represents a matrix to save the return of each state at every episode, which corresponds to an iteration. At line 3, we initialize all state values with zeros. Starting from the initial state in each iteration $b = 0$, the current state values, which constitutes the current policy, are used to take actions until the terminal state is reached. To promote exploring different states randomized actions can be taken sometimes at line 6 [75]. $G(s)$ in lines 7 and 8 represents a vector of returns of all states appearing in the episode. Inserting these values into the *Returns* array, the state values are updated by taking the average as shown in line 9. The algorithm stops when all state values converge, the converged values are then used to determine actions as in (3.4).

Similar to (3.4), we can write the optimal action at state s in terms of $Q^*(s, a)$ as follows,

$$a^* = \begin{cases} serve \text{ if } Q^*(s, serve) > Q^*(s, reject), \\ reject \text{ otherwise.} \end{cases} \tag{3.5}$$

The optimal action-value functions, required by the optimal policy as shown in (3.5), can be also computed through the value iteration technique using different RL algorithms. The procedure to learn the optimal policy from the IoT environment using the model-free SARSA, E-SARSA, and Q-learning methods is given in Algorithm 3.

Algorithm 3 shows how FN learns the optimal policy for the MDP by estimating $Q^*(s, a)$ using QL, E-SARSA, and SARSA methods. The step size parameter α represents the weight we give to the change in our experience, i.e., the learning rate, ϵ is the probability of making a random action for exploration, and the batch size n represents the number of time steps after which we update the $Q(s, a)$ values. The $\mathbb{Q}$ array at line 3 represents a matrix to save the updated values of the action-value functions of all states and actions in each iteration. In each iteration, we take an

Algorithm 3 Learning Optimum Policy using QL, E-SARSA, and SARSA

```
1: Select: ${\gamma, \epsilon} \in [0, 1]$, $\alpha \in (0, 1]$, $n \in \{1, 2, ...\}$;
2: Input: $N$ (number of RBs);
3: Initialize: $Q(s, a)$ arbitrarily in $\mathbb{Q}$, $\forall(s, a)$;
4: Initialize: $b \leftarrow 0$;
5: for $t = 0, 1, 2, ...$ do
6:     Take action $a_t$ according to $\pi$ (e.g., $\epsilon$-greedy), and store $r_t$ and $s_{t+1}$;
7:     if $t \geq n - 1$ then
8:         $\tau \leftarrow t + 1 - n$;
9:         QL: $G \leftarrow \sum_{j=\tau}^{t+1} \gamma^{(j-\tau)} r_j + \gamma^n \max_a Q(s_{t+1}, a)$;
10:        E-SARSA: $G \leftarrow \sum_{j=\tau}^{t+1} \gamma^{(j-\tau)} r_j + \gamma^n \mathbb{E}_a[Q(s_{t+1}, a)]$;
11:        SARSA: $G \leftarrow \sum_{j=\tau}^{t+1} \gamma^{(j-\tau)} r_j + \gamma^n Q(s_{t+1}, a_{t+1})$;
12:        $Q(s_\tau, a_\tau) \leftarrow Q(s_\tau, a_\tau) + \alpha[G - Q(s_\tau, a_\tau)]$;
13:        Update $\mathbb{Q}$ with $Q(s_\tau, a_\tau)$;
14:    end if
15:    if $Q(s, a)$ converges for all $(s, a)$ then
16:        $Q^*(s, a) \leftarrow Q(s, a)$;
17:        break
18:    end if
19: end for
20: Use $Q^*(s, a)$ estimated in $\mathbb{Q}$ for $\pi^*$ using (3.5)
```

action, observe and store the collected reward and the successor state. Actions are taken according to a policy π such as the ϵ-greedy policy in line 6, in which a random action with probability ϵ is taken to explore new rewards, and an optimal action (see (3.5)) is taken with probability $(1 - \epsilon)$ to maximize the rewards; with $\epsilon = 0$, the policy becomes greedy. The condition at line 7 represents the time, in terms of the batch size, at which we start updating the Q values of the actions taken in the previously visited states. The way target G is computed for QL, E-SARSA and SARSA is shown at lines 9-11. G represents the return collected starting from time $(t + 1 - n)$ to n time-steps ahead, and it contains two parts, the discounted collected rewards and a function of the action-value for future rewards. The latter part changes for QL, E-SARSA and SARSA. For QL, the maximum action-value is used considering all possible actions which can be taken from the state at $t + 1$. Whereas, E-SARSA uses the expected value of $Q(s_{t+1}, a)$ over possible actions at state s_{t+1}, and

SARSA uses $Q(s_{t+1}, a_{t+1})$ considering the action that will be taken at time $t+1$ according to the current policy. The way to update the action-value is shown at line 12, where τ is the time whose Q estimate is being updated. At line 13, the matrix $\mathbb{Q}$ is updated with the new Q value and used to make future decisions. The algorithm stops when all Q values converge. The converged values represent the optimal action values Q^* which are then used to determine optimal actions as in (3.5).

Recall that the FN objective is to maximize the expected total served utility and minimize the expected termination time, as shown in (2.2). Hence, to compare the performance of FN when using QL, SARSA, E-SARSA and MC provided in Algorithms 2 and 3 with the performance of a fixed-threshold algorithm in the utility filtering-based network slicing, which does not learn from the interactions with environment, we consider the objective performance metric R as defined in (2.12), where a served utility is denoted with u_m, the number of served IoT requests in an episode is denoted with M, $(T-M)$ represents the total idle time for RBs, and θ is a penalty for being idle.

3.3 Simulations

We next provide simulation results to evaluate the performance of FN when implementing the RL methods, Q-learning, SARSA, Expected-SARSA, and Monte Carlo, given in Algorithms 2 and 3. We also compare the RL-based FN performance with the FN performance when utility filtering-based network slicing is employed with a fixed thresholding algorithm. We evaluate the performances in various IoT environments with different compositions of IoT latency requirements. For brevity, we do not consider the effect of ratio of the achievable throughput to the throughput requirement in assessing the utility of a service request. Specifically, we consider 10 utility classes with different latency requirements to exemplify the variety of IoT applications in an F-RAN setting.

That is, we consider $\zeta = 0, \beta = 1, \kappa = 1$ in (2.1), and discretize the latency-based utility to 10 classes ($U = 10$). The utility values $1, 2, ..., 10$ may represent the following IoT applications, respectively: smart farming, smart retail, smart home, wearables, entertainment, smart grid, smart city, industrial Internet, autonomous vehicles, and connected health. By changing the composition of utility classes, we generate 19 scenarios of IoT environments, 6 of which are summarized in Table 2.1. Higher density of high-utility users makes the IoT environment richer in terms of low-latency IoT applications.

Denoting an IoT environment of a particular utility distribution with $\mathcal{E}$, we show in Table 2.1 the statistics of $\mathcal{E}_1$, $\mathcal{E}_4$, $\mathcal{E}_7$, $\mathcal{E}_{10}$, $\mathcal{E}_{15}$, and $\mathcal{E}_{19}$. The first 10 rows in the table provide detailed information about the proportion of each utility class in an IoT environment corresponding to a latency requirement. The last two rows illustrate the quality or richness of IoT environments, where ρ is the probability of a utility being greater than 5, and $\bar{u}$ is the mean value of utilities in the environment. In the considered 19 scenarios, ρ increases in steps of 0.05 from 5% to 95% for $\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_{19}$ respectively. The remaining 13 scenarios have statistics proportional to their ρ values. We started with a general scenario given by $\mathcal{E}_7$, and changed ρ to obtain the other scenarios.

The simulation parameters shown in Table 3.2 are used for the presented results in this section. The rewards $\mathcal{R} = \{r_{sh}, r_{sl}, r_{rh}, r_{rl}\}$ are chosen to facilitate learning the optimal policy. We consider that the FN is equipped with computing, signal processing and storage resources of 15 resource blocks (RBs), i.e., $N = 15$. In a particular environment $\mathcal{E}$, the threshold that defines "high utility" is set to the mean of all utilities, i.e., $u_h = \bar{u}$. We applied the greedy policy in our simulations, hence $\epsilon = 0$.

Table 3.2: Summary of simulation parameters and their values

Parameter	Description	Value
γ	discount factor	0.7
α	learning rate	0.01
ϵ	probability of random action	0
θ	penalty of idle time	1
n	batch/step size	1
N	total number of resource blocks of FN	15
r_{sh}	reward for serving high-utility user	2
r_{sl}	reward for serving low-utility user	-1
r_{rh}	reward for rejecting high-utility user	-2
r_{rl}	reward for rejecting low-utility user	1
u_h	the threshold for "high-utility"	mean

We firstly consider the MDP formulation for the IoT environment given by scenario $\mathcal{E}_7$ shown in Table 2.1. By interaction with the environment, the FN updates the state value functions which converge to the optimum policy. Fig. 3.3, shows how the FN learns the optimal policy using the Monte Carlo (MC) method given in Algorithm 2 to estimate the optimal state values. With 15 RBs, there are 160 states, the last 10 of which are terminal states with $b = 15$ for which $V(s) = 0$. The state-value functions of 16 states are given in 3.3. The remaining states have values within a standard deviation $\sigma = 0.5$ of the selected 16 states. It is seen that for most of the states the state values converges to the optimal value $V^*(s)$ after about 5000 iterations. This number can be easily exceeded by the number of requests received by FN during a busy hour from a variety of IoT applications [6].

We next apply SARSA, Expected SARSA and QL in the IoT environment $\mathcal{E}_7$, for learning the optimal policy in (3.5) using the estimated $Q^*(s,a)$ in Algorithm 3. The convergence of $Q(s, serve)$ and $Q(s, reject)$ when using QL is shown in Figs. 3.4 and 3.5, respectively. In our MDP problem, QL converges slightly faster than E-SARSA, SARSA and MC since it implements a greedy approach by

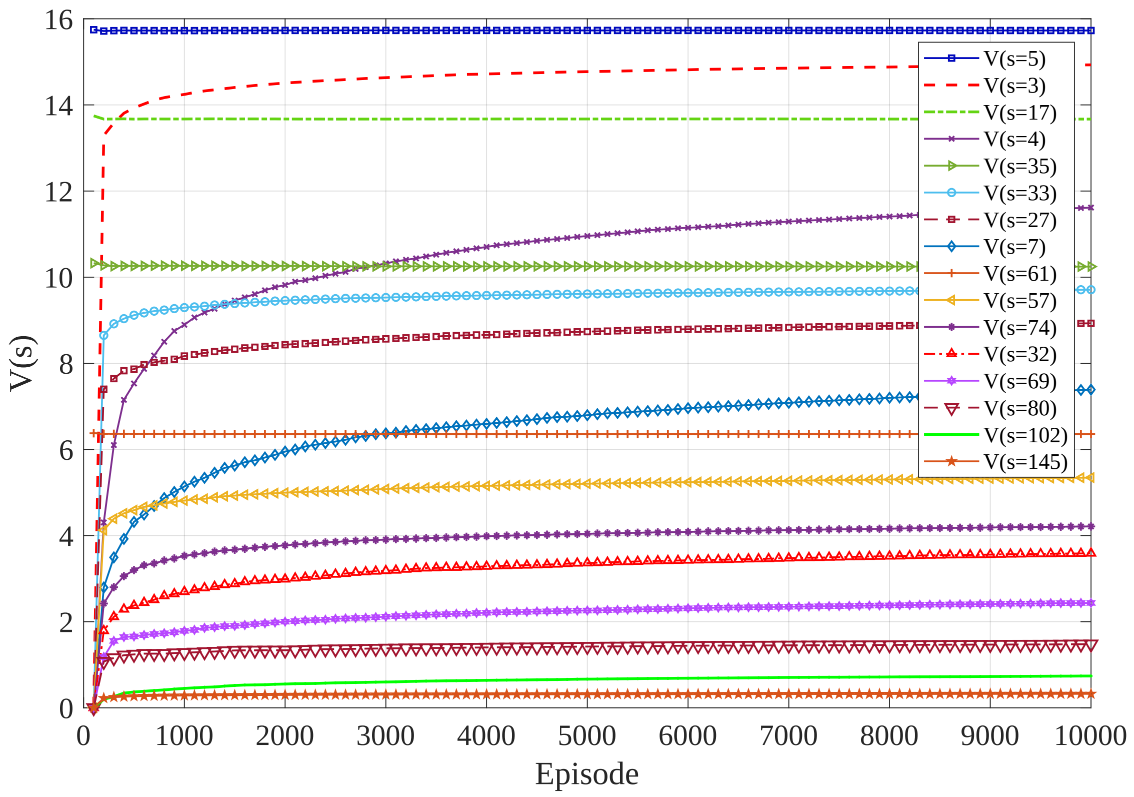


Figure 3.3: Learning optimum policy of the MDP by applying the Monte Carlo method given by Algorithm 2 to obtain the optimal state values required in (3.4). The IoT environment $\mathcal{E}_7$ is considered, and the FN is equipped with 15 RBs. The 16 state values shown in the figure are a sample of the 150 non-terminal state values.

selecting the maximum $Q(s', a')$ when updating the return G_t as shown in Algorithm 3. However, this is not a general rule as it depends on the nature of each problem. There are many factors affecting the convergence rate, e.g., large values of the learning rate α make the Q-values bounce around a mean value, whereas small values causes it to converge slowly.

Unnecessary exploration makes the convergence slower, controlled by the ϵ value in the ϵ-greedy policy. The step size n after which we update the state values or Q-values affects also the convergence dependent on the problem. For instance, MC updates the state values at the end of an episode regardless of how long it is, which makes it slower to exploit the updated state values

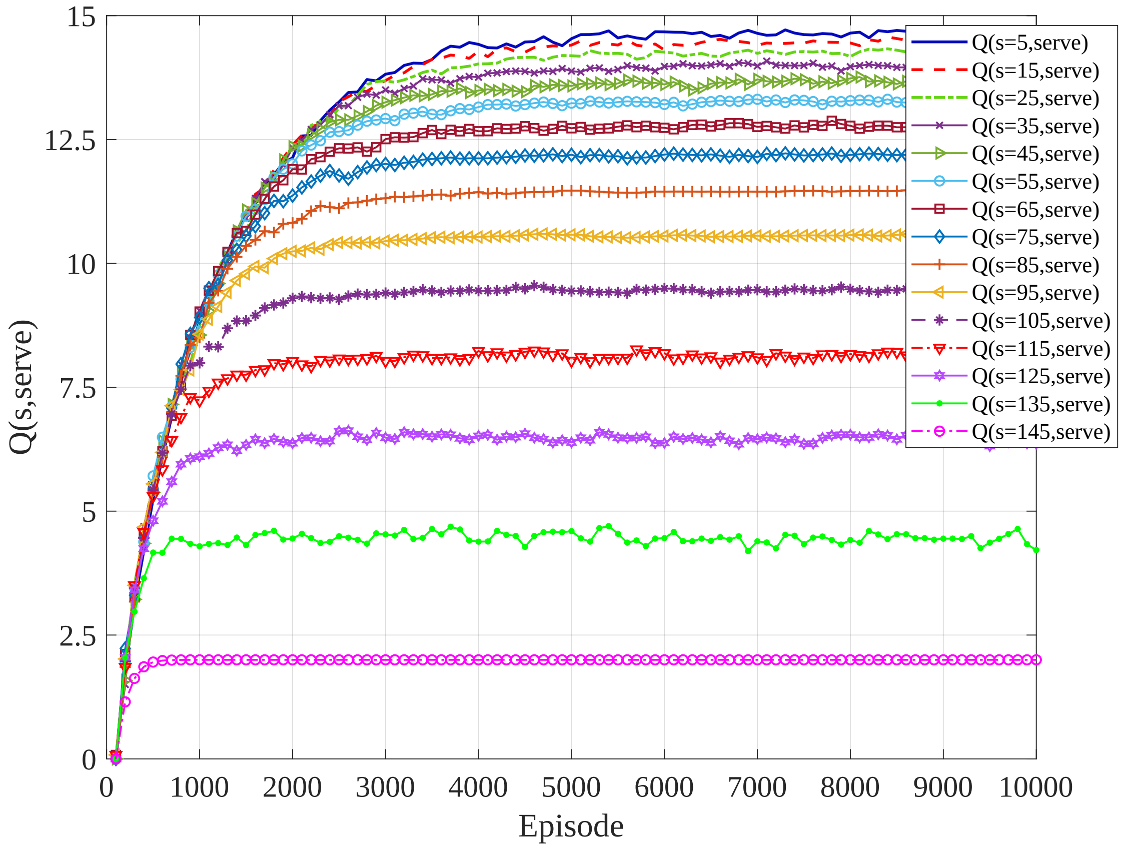


Figure 3.4: Learning the optimal action-value function $Q^*(s, serve)$ required in (3.5) using the Q-learning method given by Algorithm 3. Q-values converge to the optimal values after around 4000 episodes. The IoT environment $\mathcal{E}_7$ is considered, and the FN is equipped with 15 RBs.

in making better actions, whereas QL, SARSA and E-SARSA using $n = 1$ update the Q-value every time step. Unlike MC, the FN needs to keep updating two Q-values for each state instead of updating one state value. Hence, we have 300 Q-values to update in order to learn the optimal policy.

We compare the performance of the RL methods, in terms of R, as shown in (2.12), with that of the utility filtering-based network slicing with various slicing thresholds in the 19 IoT environments. The utility filtering algorithm uses the same threshold for network slicing regardless of the environment. For the RL methods, we consider the simulation setup shown in Table 3.2, and

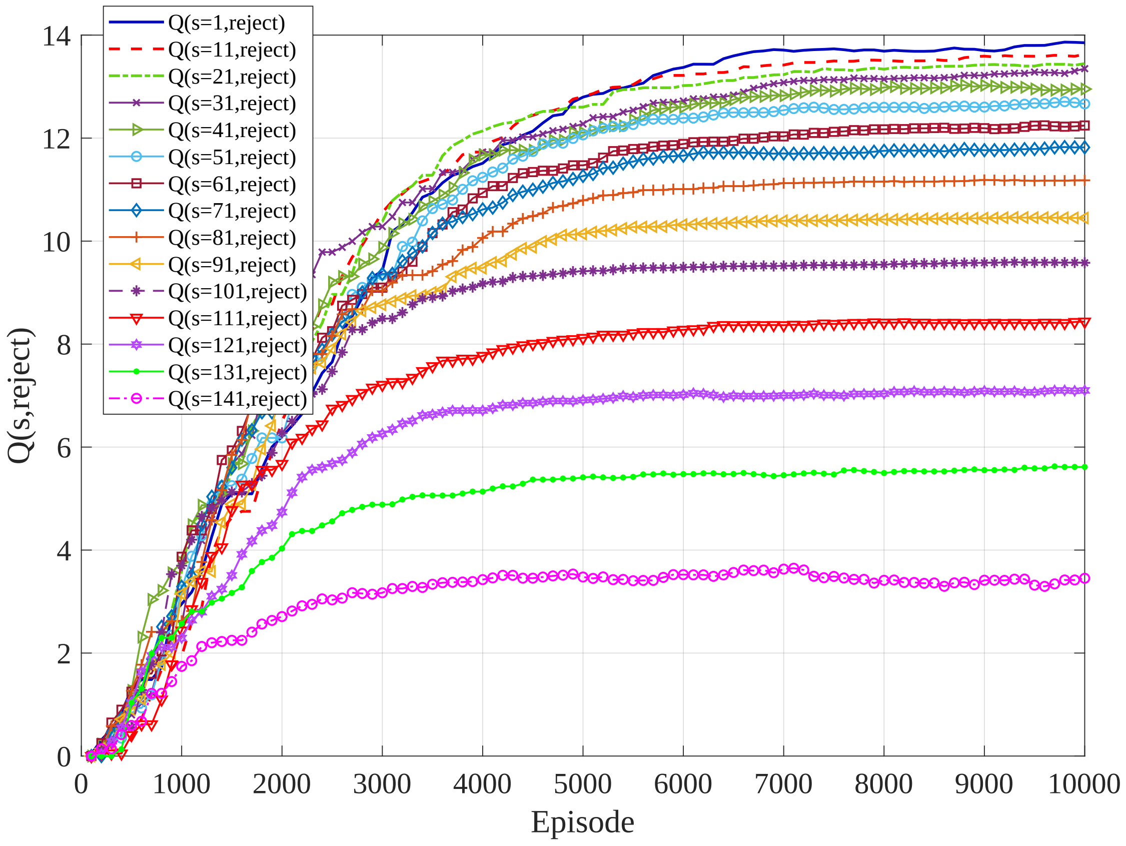


Figure 3.5: Learning the optimal action-value function $Q^*(s, reject)$ required in (3.5) using the Q-learning method given by Algorithm 3. Q-values converge to the optimal values after around 5000 episodes. The IoT environment $\mathcal{E}_7$ is considered, and the FN is equipped with 15 RBs.

for the utility filtering-based network slicing we consider all possible slicing thresholds $1, 2, ..., 10$. As shown in Figs. 3.6 and 3.7, the RL methods exhibit the best performance as they learn how to balance early termination with higher total served utilities. It never terminates too early or too late ($T \approx 27$ for all environments as seen in Fig. 3.7), as opposed to the utility filtering-based network slicing which is not adaptive to the environment. As seen in Fig. 3.6, the performance of the utility filtering algorithm with slicing thresholds $1, 2, 3, 8, 9$ are steadily below that of the RL algorithms. The average termination time for slicing thresholds 1, 2, and 3 is about 15 which is the minimum termination time, though they could not achieve good performance. Slicing threshold

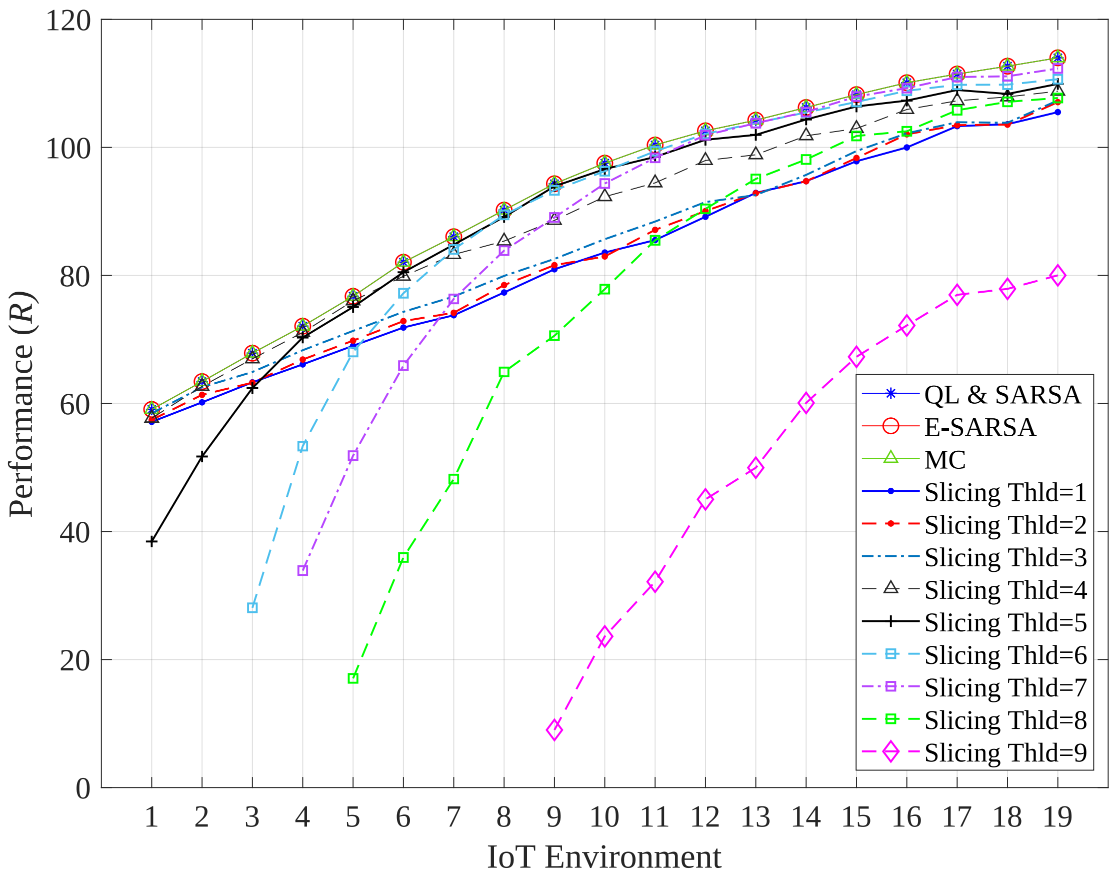


Figure 3.6: The performance in terms of R for the FN with $N = 15$, in various IoT environments when applying the RL methods (QL, SARSA, E-SARSA and MC) given in Algorithms 2 and 3, and the utility filtering algorithm in network slicing with different slicing thresholds. RL methods' performances are indistinguishable here, and better than the conventional-filtering based network slicing in all environments thanks to their learning and adaptation capability.

4 has a comparable performance to RL for the environments $\mathcal{E}_2 - \mathcal{E}_5$, after which its performance starts to decline. Although slicing thresholds $5, 6, 7$ have good performances close to RL for environments with medium to high ρ, they perform far from RL for IoT environments with small ρ. The performance of slicing threshold 10 is much worse than threshold 9 for all environments due to the long termination time which exceeds 280 time-steps, thus it does not appear in Figs. 3.6 and 3.7.

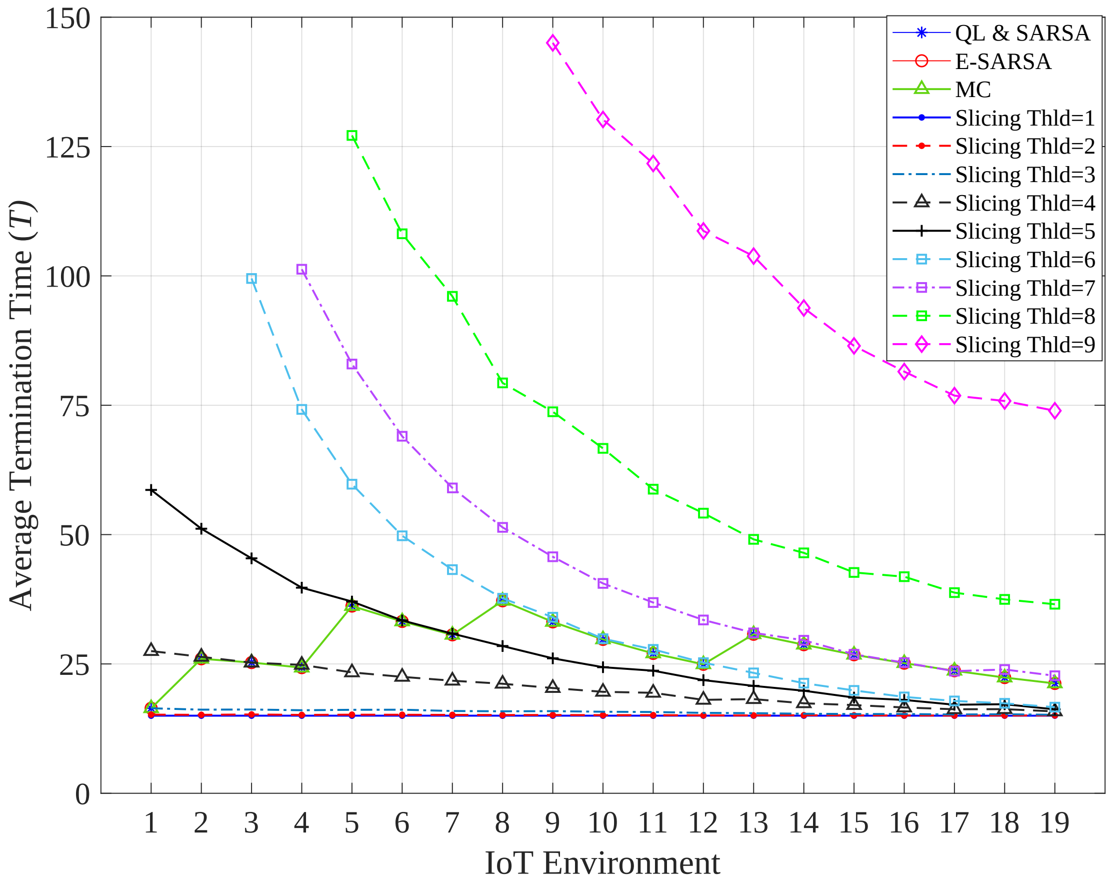


Figure 3.7: The average termination time T in time-steps for FN with $N = 15$ in various IoT environments when applying the RL methods (QL, SARSA, E-SARSA and MC) given by Algorithms 2 and 3, and the utility filtering algorithm in network slicing with different slicing thresholds. RL methods manage to have a steady termination time in all environments.

The performance of the RL methods is very close to each other, hence it is not easy to distinguish them in Figs. 3.6 and 3.7. For a clearer view, Fig. 3.8 compares the performance of the four RL methods in terms of the performance ratio with respect to performance of slicing threshold 4, i.e., (R_{RL}/R_{Thld4}). QL has the best performance with an average performance ratio of 104% in all IoT environments with a peak of 106% in $\mathcal{E}_9$, followed by E-SARSA and MC. SARSA has the same performance as QL because greedy policy, i.e., $\epsilon = 0$, was used.

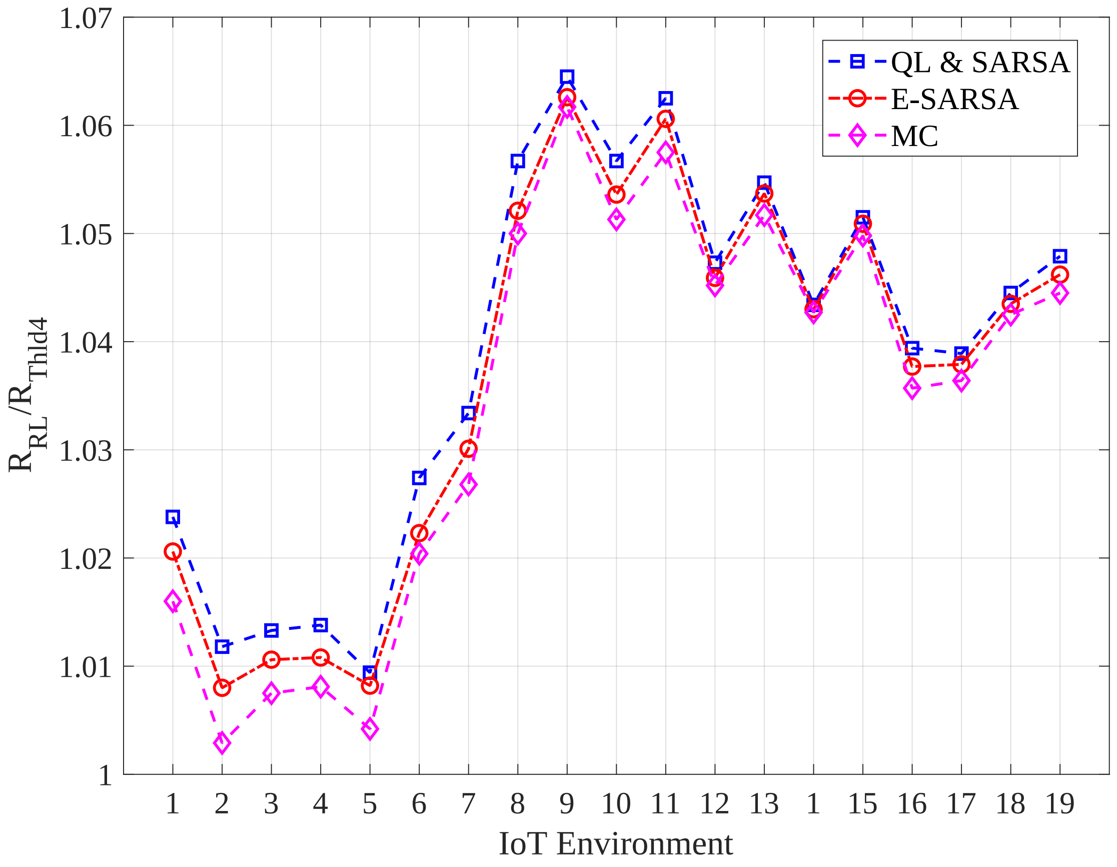


Figure 3.8: Comparison between the performance of RL methods in terms of relative performance with respect to the utility filtering algorithm in network slicing with slicing threshold 4. QL and SARSA coincide due to the greedy policy used in the simulations.

3.4 Summary

In this chapter, we proposed an infinite Markov Decision Process (MDP) formulation for the resource allocation problem in Fog RAN for IoT services with heterogeneous latency requirements. Several reinforcement learning (RL) methods, namely Q-learning, SARSA, Expected SARSA, and Monte Carlo, were discussed for learning the optimum decision-making policy adaptive to the IoT environment. Their superior performance over utility filtering-based network slicing methods, and adaptivity to the IoT environment were verified through extensive simulations. The RL methods strike a right balance between the two conflicting objectives, maximize the average total served utility vs. minimize the fog node's idle time, which helps utilize fog node's limited resource blocks efficiently. In the next chapter, we consider expanding the presented resource allocation framework to more challenging scenarios such as dynamic resource allocation with heterogeneous service times and number of resource blocks needed, and collaborative resource allocation with multiple fog nodes.

Chapter 4: Deep RL for Adaptive Network Slicing in IoT Communications

Intelligent vehicular systems and smart city applications are the fastest growing Internet of things (IoT) implementations at a compound annual growth rate of 30%.[4] In view of the recent advances in IoT devices and the emerging new breed of IoT applications driven by artificial intelligence (AI), fog radio access network (F-RAN) has been recently introduced for the fifth generation (5G) wireless communications to overcome the latency limitations of cloud-RAN (C-RAN). In this chapter, we consider the network slicing problem of allocating the limited resources at the network edge (fog nodes) to vehicular and smart city users with heterogeneous latency and computing demands in dynamic environments. We develop a network slicing model based on a cluster of fog nodes (FNs) coordinated with an edge controller (EC) to efficiently utilize the limited resources at the network edge. For each service request in a cluster, the EC decides which FN to execute the task, i.e., locally serve the request at the edge, or to reject the task and refer it to the cloud. We formulate the problem as infinite-horizon Markov decision process (MDP) and propose a deep reinforcement learning (DRL) solution to adaptively learn the optimal slicing policy. The performance of the proposed DRL-based slicing method is evaluated by comparing it with other slicing approaches in dynamic environments and for different scenarios of design objectives. Comprehensive simulation results corroborate that the proposed DRL-based EC quickly learns the optimal policy

[4]Portions of this chapter have appeared in arXiv preprint [4]. Copyright permissions from the publishers are included in Appendix A.

through interaction with the environment, which enables adaptive and automated network slicing for efficient resource allocation in dynamic vehicular and smart city environments.

4.1 Network Slicing Model

We consider the F-RAN network slicing model for IoV and smart city shown in Fig. 4.1. The two logical network slices, cloud slice and edge slice, support multiple radio access technologies and serve heterogeneous latency needs and resource requirements in dynamic IoV and smart city environments. The hexagonal structure represents the coverage area of fog nodes (FNs) in the edge slice, where each hexagon exemplifies an FN's footprint, i.e., its serving zone. An FN in an edge cluster is connected through extremely fast and reliable optical links with its adjacent FNs whose hexagons have a common side with it. FNs in the edge slice are also connected via high-capacity fronthaul links to the cloud slice which includes a powerful cloud controller (CC) of massive computing capabilities, a pool of huge storage capacity, centralized baseband units (BBUs), and an operations and maintenance center (OMC) which monitors the key performance indicators (KPIs) and generates network reports. To ensure the best QoS for the massive smart city and IoV service requests, especially the URLLC applications and to mitigate the onus on the fronthaul and the cloud, FNs are equipped with computing and processing capabilities to independently deliver network functionalities at the edge of network. However, the edge resources at FNs are limited, and therefore need to be used efficiently.

In an environment densely populated with low-latency service requests, it is rational for the FNs to route delay-tolerant applications to the cloud and save the limited edge resources for delay-sensitive applications. However, in practice, smart city and IoV environments are dynamic,

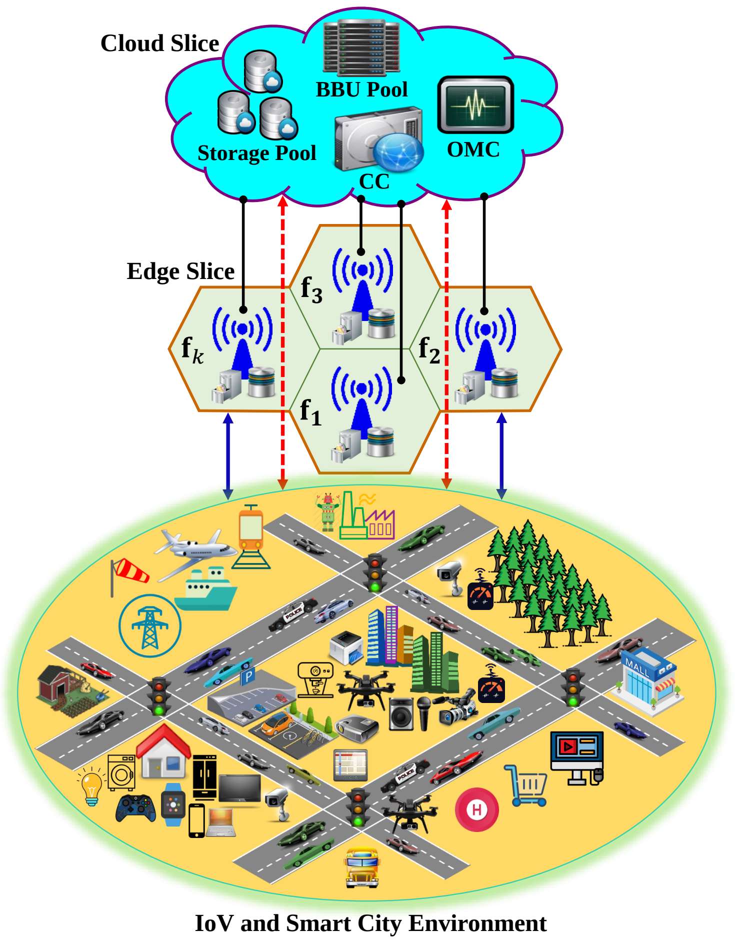


Figure 4.1: Network slicing model. Edge slice is connected to cloud slice through high-capacity fronthaul links represented by solid lines. Solid arrows represent edge service to satisfy QoS, and dashed arrows represent task referral to the cloud-slice to save the limited resources of edge slice.

i.e., a typical environment will not be always densely populated with delay-sensitive applications. A rule-based network slicing policy cannot ensure efficient use of edge resources in dynamic environments as it will under-utilize the edge resources when delay-sensitive applications are rare. On the other hand, a statistical learning policy can adapt its decisions to the changing environment characteristics. Moreover, it can learn to prioritize low-load delay-sensitive applications over high-load delay-sensitive ones.

We propose using edge controllers (ECs) to efficiently manage the edge resources by enabling cooperation among FNs. In this approach, FNs are grouped in clusters, each of which covers a particular geographical area, and manages the edge resources in that area through a cluster head called EC. The cluster size k is a network design parameter which represents the number of coordinated FNs in an edge cluster. An FN in each cluster is appointed as EC to manage and coordinate edge resources at FNs in the cluster. The EC is nominated by the network designer mainly based on its central geo location among the FNs in the cluster, like f_1 and f_3 in Fig. 4.1. Note that unlike the cloud controller, the edge controller is close to the end users as it is basically one of the FNs in a cluster. Also, the cluster size k is constrained by the neighboring FNs that cover a limited service area such as a downtown, industrial area, university campus, etc.

All FNs in an edge cluster are connected together and with the EC through super-speedy reliable optical links. The EC monitors all individual FN internal states, including resource availability and received service requests, and decides for each service request received by an FN in the cluster. For each received request, the EC chooses one of the three options: serve at the receiving FN (primary FN), serve at a neighboring FN, or serve at the cloud. Each FN in the cluster has a predefined list $\mathcal{N}_i$ of neighboring FNs, which can help serving a received service request. For instance, $\mathcal{C} = \{\mathrm{f}_1, \mathrm{f}_2, \ldots, \overset{\star}{\mathrm{f}}_i, \ldots, \mathrm{f}_k\}$ is an edge cluster of size k, where $\overset{\star}{\mathrm{f}}_i$ denotes the EC which can be any FN in the cluster $\mathcal{C}$. The network designer needs to define a neighboring list $\mathcal{N}_i \subseteq \{\mathcal{C} - \mathrm{f}_i\}$ for each FN in the cluster. An FN can hand-over service tasks only to its neighbors. Dealing with IoV and smart city service requests, we call the FN which receives a request the primary FN $\hat{\mathrm{f}}$, and call the FN which actually serves the request utilizing its resources the serving FN $\bar{\mathrm{f}}$. Depending on the EC decision, the primary FN or one of its neighbors can be the serving FN, or there can be no serving FN (for the decision to serve at the cloud).

An IoV or smart city application attempts to access the network by sending a service request to the primary FN , which is usually the closest FN to the user. The primary FN checks the utility $u \in U = \{1, 2, \ldots, u_{max}\}$, i.e., the priority level of executing the service task at the edge, analyzes the task load by figuring the required amount of resources $c \in C = \{1, 2, \ldots, c_{max}\}$ and holding time of resources $h \in H = \{1, 2, \ldots, h_{max}\}$, and sends the EC the task input (u_t, c_t, h_t) at time t. We consider the resource capacity of the i^{th} FN $\text{f}_i \in \mathcal{C}$ is limited to N_i resource blocks. Hence, the maximum number of resource blocks to be allocated for a task is constrained by the FN resource capacity, i.e., $c \leq c_{max} \leq \text{N}_i$. We partition the time into very small time steps $t = 1, 2, ...$, and assume a high-rate sequential arrival of IoV and smart city service requests, one task at a time step. ECs should be intelligent to learn how to decide (which FN to *serve* or *reject*) for each service request, i.e., how to sequentially allocate limited edge resources, to achieve the objective of efficiently utilizing the edge resources while maximizing the grade-of-service (GoS) defined as the proportion of served high-utility requests to the total number of high-utility requests received.

A straightforward approach to deal with this network slicing problem is to filter the received service requests by comparing their utility values with a predefined threshold. For instance, consider ten different utilities $u \in \{1, 2, 3, ..., 10\}$ for all received tasks in terms of the latency requirement, where $u = 10$ represents the highest-priority and lowest-latency task such as the emergency requests from the driver distraction alerting system, and $u = 1$ is for the lowest-priority task with highest level of latency such as a service task from smart waste management system. Then, a straightforward non-adaptive solution for network slicing is to dedicate the edge resources to high-utility tasks, such as $u \geq u_h$, and refer to the cloud the tasks with $u < u_h$, where the threshold u_h is a predefined network design parameter. However, such a policy is strictly sub-optimum since the EC will execute any task which satisfies the threshold regardless of how demanding the task load is. For instance, while FNs are busy with serving a few high-utility requests of high load, i.e., low-latency tasks which require large amount of resources c and long holding times h, many high-utility requests with low load demand may be missed. In addition, this straightforward policy increases the burden on the cloud unnecessarily, especially when the environment is dominated by low-utility tasks with $u < u_h$. A better network slicing policy would consider the current resource utilization and expected reward of each possible action while deciding, and also adapt to changing utility and load distributions in the environment. To this end, we next propose a Markov Decision Process (MDP) formulation for the considered network slicing problem.

4.2 MDP Formulation at EC

MDP formulation enables the EC to consider expected rewards of all possible actions in its network slicing decision. Since closed form expressions typically do not exist for the expected reward of each possible action at each system state in a real-world problem, reinforcement learning (RL) is commonly used to empirically learn the optimum policy for the MDP formulation. The RL agent (the EC in our problem) learns to maximize the expected reward by trial and error. That means the RL agent sometimes exploits the best known actions, and sometimes, especially in the beginning of learning, explores other actions to statistically strengthen its knowledge of best actions at different system states. Once, the RL agents learns an optimum policy (i.e., the RL algorithm converges) through managing this exploitation-exploration trade-off, the learned policy can be exploited as long as the environment (i.e., the probability distribution of system state) remains the same. In dynamic IoV and smart city environments, an RL agent can adapt its decision policy to the changing distributions.

As illustrated in Fig. 4.2, for each service request in an edge cluster at time t from an IoV or smart city application with utility u_t, the primary FN computes the number of resource blocks c_t and the holding time h_t which are required to serve the task locally at the edge. Then, the primary FN shares (u_t, c_t, h_t) with the EC , which keeps track of the available resources at all FNs in the cluster. If neither the primary FN nor its neighbors has c_t available resource blocks for a duration of h_t, the EC inevitably rejects serving the task at the edge and refers it to the cloud. Note that in the proposed cooperative structure enabled by the EC, such an automatic rejection will be much less frequent compared to the non-cooperative structure considered in [1, 3, 60, 69], where each FN decides for its resources on its own. If the requested resource blocks c_t for the requested duration h_t are available at the primary FN or at least one of the neighbors, then the EC uses the RL algorithm given in the next section to decide either to serve or reject. In any case, as a result of the taken action a_t, the EC will observe a reward r_t and the system state s_t will transition to s_{t+1}. We next explain the KPIs in an F-RAN to guide the design of the proposed MDP formulation.

4.2.1 Key Performance Indicators

Considering the main motivation behind F-RAN we define the Grade of Service (GoS) as a key performance indicator (KPI). GoS is the proportion of the number of served high-utility service tasks to the total number of high-utility requests in the cluster, and given by

$$\text{GoS} = \frac{m_h}{M_h} = \frac{\sum_{t=0}^{T-1} \mathbb{1}_{\{u_t \geq u_h\}} \mathbb{1}_{\{a_t \in \{1,2,\ldots,k\}\}}}{\sum_{t=0}^{T-1} \mathbb{1}_{\{u_t \geq u_h\}}}, \tag{4.1}$$

where u_h is a utility threshold which differentiates the low-latency (i.e., high-utility) tasks such as URLLC from other tasks, $a_t \in \{1, 2, \dots, k\}$ means *serve* the requested task at the i^{th} FN in the cluster, $\mathrm{f}_i \in \mathcal{C} = \{\mathrm{f}_1, \mathrm{f}_2, \dots, \mathrm{f}_k\}$, and $\mathbb{1}_{\{\cdot\}}$ is the indicator function taking the value 1 when its argument is true, and 0 otherwise.

Naturally, a network operator would want the edge resources to be efficiently utilized. Hence, the average utilization of edge resources over a time period T gives another KPI:

$$\text{Utilization} = \frac{1}{T} \sum_{t=0}^{T-1} \frac{\sum_{i=1}^{k} b_{it}}{\sum_{i=1}^{k} \mathrm{N}_i}, \tag{4.2}$$

where b_{it} and N_i are the number of occupied resources at time t, and the resource capacity of the FN f_i in the cluster, respectively. Another KPI to examine the EC performance is cloud avoidance which is given by the proportion of all IoV and smart city service requests that are served by FNs in the edge cluster to all requests received. Cloud avoidance is reported over a period of time T, and it is given by

$$\text{Cloud Avoidance} = \frac{m}{M} = \frac{\sum_{t=0}^{T-1} \mathbb{1}_{\{a_t \in \{1,2,\dots,k\}\}}}{M}, \tag{4.3}$$

where $m = m_h + m_l$ is the number of high-utility and low-utility served requests at the edge cluster, and $M = M_h + M_l$ is the total number of high-utility and low-utility received requests. Note that $M - m$ is the portion of IoV and smart city service tasks which is served by the cloud, and one of the objectives of F-RAN is to lessen this burden especially during busy hours. Cloud avoidance shows a general overview about the contribution of edge slice to share the load. It gives a similar metric as resource utilization, which is more focused on resource occupancy rather than dealing with service requests in general. While we use the resource utilization together with GoS to define an overall performance metric below, cloud avoidance is still used as a performance evaluation metric in Sec. 4.4.

To evaluate the performance of an EC over a particular period of time T, we consider the weighted sum of the main two KPIs, the GoS and edge-slice average resource utilization as

$$\text{Performance} = \omega_g \text{GoS} + \omega_u \text{Utilization}. \tag{4.4}$$

In Sec. 4.4, different performance scenarios are considered to evaluate the proposed DRL scheme with respect to other slicing approaches in various vehicular and smart city environments.

4.2.2 MDP Formulation

An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P^a, R^a, \gamma)$, where $\mathcal{S}$ is the set of states, i.e., $s_t \in \mathcal{S}$, $\mathcal{A}$ is the set of actions, i.e., $a_t \in \mathcal{A} = \{1, 2, \ldots, k, k+1\}$, $P^a(s, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$ is the transition probability from state s to s' when action a is taken, $R^a(s, s')$ is the reward received by taking action a in state s which ends up in state s', i.e., $r_t \in R^a(s, s')$, and $\gamma \in [0, 1]$ is the discount factor in computing the return which is the cumulative reward as shown in (3.2). γ represents how much weight is given to the future rewards compared to the immediate reward. For $\gamma = 1$, future rewards are of equal importance as the immediate reward, whereas $\gamma = 0$ completely ignores future rewards. The objective in MDP is to maximize the expected cumulative reward starting from $t = 0$, i.e., $\max_{\{a_t\}} \mathbb{E}[G_0 | s_0]$, where G_t is given by (3.2), by choosing the actions $\{a_t\}$.

Before explaining the (state, action, reward) structure in the proposed MDP, let us first define the task load $L_t = c_t \times h_t$ as the number of resource blocks required to execute a task

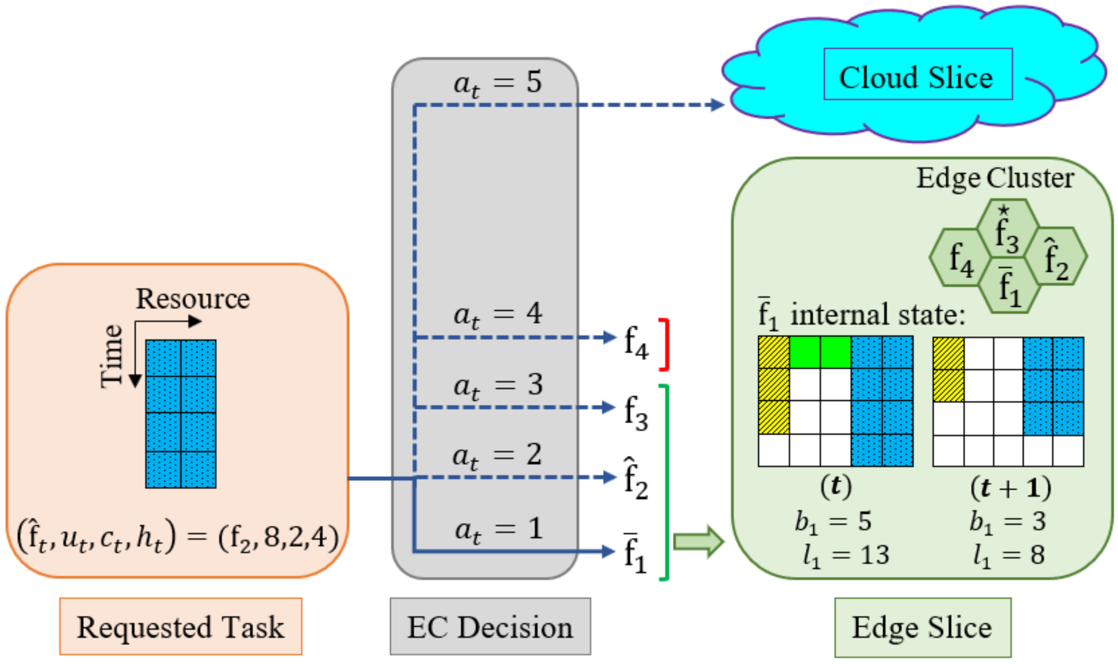


Figure 4.2: EC decision for a sample service request received by f_2, and the internal state of the serving FN f_1 with $N_1 = 5$ and $h_{max} = 4$ (see (4.5)). The edge cluster size is $k = 4$ and f_3 is the EC.

completely, and similarly the existing load l_{it} of FN i as the number of already allocated resource blocks (see Fig. 4.2).

- *State:* We define the system state in a cluster of size k at any time t as

$$s_t = (b_{1t}, l_{1t}, b_{2t}, l_{2t}, \ldots, b_{kt}, l_{kt}, \hat{f}_t, u_t, c_t, h_t), \tag{4.5}$$

  where b_{it} denotes the number of resource blocks in use at FN i at time t. Note that $b_{i(t+1)}, l_{i(t+1)}$ and in turn the next state s_{t+1} are independent of the past values given the current state s_t, satisfying the Markov property $P(s_{t+1}|s_0, s_1, s_2, ..., s_t, a_t) = P(s_{t+1}|s_t, a_t)$.

- *Action:* The EC decides, as shown in Fig. 4.2, for each service request by taking an action $a_t \in \mathcal{A} = \{1, 2, \ldots, k, k+1\}$, where $a_t = i \in \{1, 2, \ldots, k\}$ means *serve* the requested task at the i^{th} FN in the cluster, $\mathrm{f}_i \in \mathcal{C} = \{\mathrm{f}_1, \mathrm{f}_2, \dots, \mathrm{f}_k\}$, whereas $a_t = k+1$ means to *reject* the job and refer it to the cloud. Note that for a request received by f_i, the feasible action set is a subset of $\mathcal{A}$ consisting of f_i, its neighbors $\mathcal{N}_i$, and the cloud. Fig. 4.2 illustrates the decision of the EC for a sample service request received by f_2 at time t in an edge cluster with $k=4$ FNs. Note that in this example the action $a_t=4$ is not feasible as $\mathrm{f}_4 \notin \mathcal{N}_2$, and the EC took the action $a_t=1$, which means serve the task by f_1. Hence, f_1 started executing the task at t while another two tasks (striped yellow and green) are in progress. At $t+1$, two resource blocks are released as the job in clear-green is completed. Note that resource utilization of f_1 decreased from 100% at t, i.e., internal busy state with $b_{1t}=5$, to 60% at $t+1$.

- *Reward:* In general, a proper rewarding system is crucial for an RL agent to learn the optimum policy of actions that maximizes the KPIs. The RL agent at the EC collects an immediate reward $r_t \in R^a(s, s')$ for taking action a at time t from state s which ends in the successor state s' in the next time step $t+1$. We define the immediate reward

$$r_t = r_{(a_t, u_t)} \pm r_{L_t} \tag{4.6}$$

using two components. The first term $r_{(a_t,u_t)} \in \{r_{sh}, r_{sl}, r_{rh}, r_{rl}, r_{bh}, r_{bl}\}$ corresponds to the reward portion for taking an action $a \in \{1, 2, \dots, k, k+1\}$ when a request of specific u is received, and the second term

$$r_{L_t} = c_{max} \times h_{max} + 1 - L_t, \tag{4.7}$$

considers the reward portion for handling the new job load $L_t = c_t \times h_t$ of a requested task. For instance, serving low-load task such as $L = 3$ is awarded more than serving a task with $L = 18$. Similarly, rejecting a low-load task such as $L = 3$ should be more penalized, i.e., negatively rewarded especially when $u \geq u_h$, than rejecting a task with the same utility and higher load such as $L = 18$. The two reward parts are added when $a_t = serve$, and subtracted if $a_t = reject$. We define six different reward-component $r_{(a,u)} \in \{r_{sh}, r_{sl}, r_{rh}, r_{rl}, r_{bh}, r_{bl}\}$, where r_{sh} is the reward for serving a high-utility request, r_{sl} is the reward for serving a low-utility request, r_{rh} is the reward for rejecting a high-utility request, r_{rl} is the reward for rejecting a low-utility request, r_{bh} is the reward for rejecting a high-utility request due to being busy, and r_{bl} is the reward for rejecting a low-utility request due to being busy. Note that having a separate reward for rejecting due to a busy state makes it easier for the RL agent to differentiate between similar states for the *reject* action. A request is determined as high-utility or low-utility based on the threshold u_h, which is a design parameter that depends on the level of latency requirement in an IoV and smart city environment.

4.3 Optimal Policies and DQN

The state value function $V(s)$ represents the long-term value of being in a state s. That is, starting from state s how much value on average the EC will collect in the future, i.e., the expected total discounted rewards from that state onward. Similarly, the action-value function $Q(s, a)$ tells how valuable it is to take a particular action a from the state s. It represents the expected total reward which the EC may get after taking the particular action a from the state s onward. The state-value and the action-value functions are given by the Bellman expectation equations (2.3) and (2.4), respectively [75], where the state value $V(s)$ and the action value $Q(s, a)$ are recursively presented in terms of the immediate reward r_t and the discounted value of the successor state $V(s')$ and the successor state-action $Q(s', a')$, respectively. a' denotes the action at the next state s'.

Starting at the initial state s_0, the EC objective can be achieved by maximizing the expected total return $V(s_0) = \mathbb{E}[G_0|s_0]$ over a particular time period T. To achieve this goal, the EC should learn an optimal decision policy to take proper actions. However, considering the large dimension of sate space (see (4.5)) and the intractable number of state-action combinations, it is infeasible for RL tabular methods to keep track of all state-action pairs and continuously update the corresponding $V(s)$ and $Q(s, a)$ for all combinations in order to learn the optimal policy. Approximate DRL methods such as DQN is a more efficient alternative for the high-dimensional EC MDP to quickly learn an optimal decision policy to take proper actions, which we discuss next.

A policy π is a way of selecting actions. It can be viewed as a mapping from states to actions as it describes the set of probabilities for all possible actions to select from a given state, i.e., $\pi = \{P(a|s)\}$. A policy helps in estimating the value functions in (2.3) and (2.4). π_1 is said to be better than another policy π_2 if the state value function following π_1 is greater than that following π_2 for all states, i.e., $\pi_1 > \pi_2$ if $V_{\pi_1}(s) > V_{\pi_2}(s), \forall s \in \mathcal{S}$. A policy π is said to be optimal if it maximizes the value of all states, i.e., $\pi^* = \arg\max_{\pi} V_\pi(s), \forall s \in \mathcal{S}$. Hence, to solve the considered MDP problem, the DRL agent needs to find the optimal policy through finding the optimal state-value function $V^*(s) = \max_{\pi} V_\pi(s)$, which is similar to finding the optimal action-value function $Q^*(s, a) = \max_{\pi} Q_\pi(s, a)$ for all state-action pairs. From (2.3) and (2.4), we can write the Bellman optimality equations for $V^*(s)$ and $Q^*(s, a)$ as in (2.5) and (2.6), respectively. The expression of optimal state-value function $V^*(s)$ greatly simplifies the search for optimal policy as it subdivides the targeted optimal policy into local actions: take an optimal action a^* from state s

which maximizes the expected immediate reward followed by the optimal policy from successor state s'. Hence, the optimal policy is simply taking the best local actions from each state considering the expected rewards. Dealing with $Q^*(s,a)$ to choose optimal actions is even easier, because with $Q^*(s,a)$ there is no need for the EC to do the one-step-ahead search and instead it picks the best action that maximizes $Q^*(s,a)$ at each state. The optimal action for each state s is given by (2.7).

The optimal policy can be learned by solving the Bellman optimality equations (2.5) and (2.6) for a^*. This can be done for tractable number of states by estimating the optimal value functions using tabular solution methods such as dynamic programming, and model-free RL methods which include Monte Carlo, SARSA, expected SARSA, and Q-Learning (QL) [3]. However, for high-dimensional state space, such as ours given in (4.5), tabular methods are not tractable in terms of computational and storage complexity. Deep RL (DRL) methods address the high-dimensionality problem by approximating the value functions using deep neural networks (DNN).

Deep Q-Network (DQN) is a powerful DRL method for addressing RL problems with high-dimensional input states and output actions [61]. DQN extends QL to high-dimensional problems by using DNN to approximate the action-value functions without keeping a Q-table to store and update the Q-values for all possible state-action pairs as in QL. Fig. 4.3 demonstrates the DQN method for EC in the network slicing problem, in which the DQN agent at EC learns about the IoV and smart city environment by interaction. The DQN agent is basically a DNN that consists of an input layer, hidden layers, and an output layer. The number of neurons in the input and output layers is equal to the state and action dimensions, respectively, whereas the number of hidden layers and the number of neurons in each hidden layer are design parameters to be chosen. Feeding the current EC state s to the DNN as an input and regularly updating its parameters, i.e., the weights

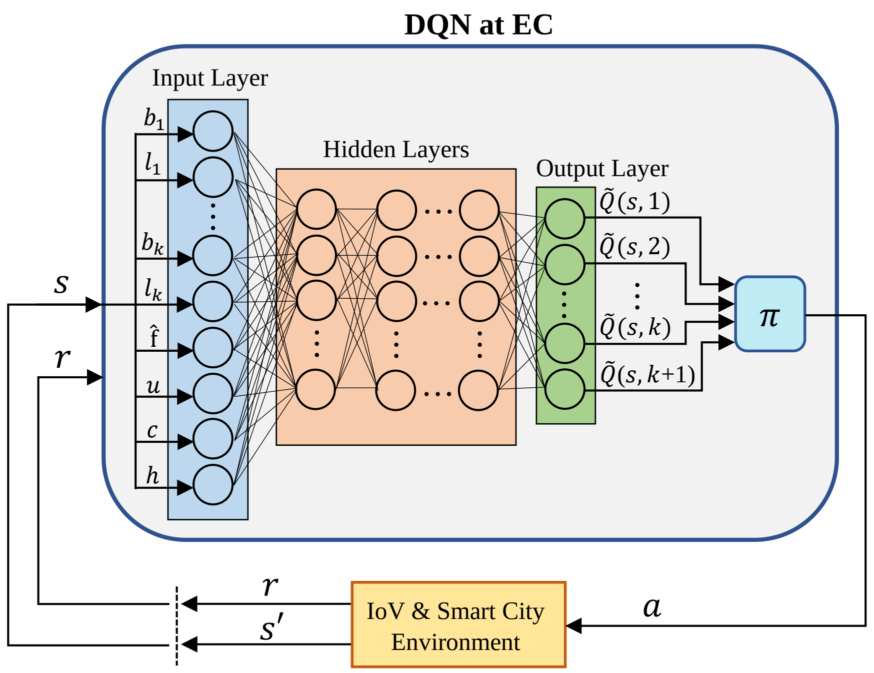


Figure 4.3: The interaction of the DQN-based EC with the IoV and smart city environment. Given the EC input state $s = (b_1, l_1, \dots, b_k, l_k, \hat{\mathrm{f}}, u, c, h)$, the DQN agent predicts the action-value functions and follows a policy π to take an action a which ends up in state s', and collects a reward r accordingly.

of all connections between neurons, DNN is able to predict the Q-values at the output for a given input state. The DRL agent at EC sometimes takes random actions to explore new rewards, and at other times exploits its experience to maximize the discounted cumulative rewards over time and keeps updating the DNN weights. Once the DNN weights converge to the optimal values, the agent learns the optimal policy for taking actions in the observed environment.

For a received service request, if the requested resources are affordable, i.e., $c_t \leq (\mathrm{N}_i - b_{it})$ for any $\mathrm{f}_i \in \{\hat{\mathrm{f}}_i, \mathcal{N}_i\}$, the EC makes a decision whether to *serve* the request by the primary FN or one of its neighbors, or *reject* and refer it to the cloud. From (2.7), the optimal action at state s is given by,

$$a^* = \begin{cases} i \in \tilde{\mathcal{A}} \ \text{ if } Q^*(s,i) = \max\limits_{a \in \tilde{\mathcal{A}}} Q(s,a), \\ \\ k+1 \ \text{ otherwise,} \end{cases} \tag{4.8}$$

where $\tilde{\mathcal{A}}$ denotes the set of possible *serve* actions to execute the service task by $f_i \in \{\hat{f}_i, \mathcal{N}_i\}$. The procedure to learn the optimal policy from the IoV and smart city environment using the model-free DQN algorithm is given in Algorithm 4.

Algorithm 4 Learning Optimum Policy Using DQN

```
1:  Select: $\{\gamma, \epsilon\} \in [0,1]$, $\rho \in (0,1]$, $n \in \{1,2,\ldots,D\}$;
2:  Create DNN model and target model with weights $\mathrm{w}$ and $\hat{\mathrm{w}}$, respectively;
3:  Initialize: $\mathrm{w}$, $\hat{\mathrm{w}}$, the replay memory $\mathbb{M}$ with size $D$;
4:  Initialize: $s$;
5:  for $t = 0,1,2,\ldots,T$ do
6:     Take action $a_t$ according to $\pi = \epsilon$-greedy, and observe $r_t$ and $s_{t+1}$;
7:     Append the observation $(s_t, a_t, r_t, s_{t+1})$ to $\mathbb{M}$;
8:     $s \leftarrow s_{t+1}$;
9:     Sample a random minibatch of $n$ observations from $\mathbb{M}$;
10:    for $j = 1,2,\ldots,n$ do
11:       Predict $\hat{Q}_j(s_j|\hat{\mathrm{w}})$;
12:       $\mathrm{y}_j = \begin{cases} r_j \ \text{ if } t+1 = T, \\ r_j + \gamma \max\limits_{a'} \hat{Q}_j(s', a'|\hat{\mathrm{w}}) \ \text{ otherwise.} \end{cases}$
13:       Fit the DNN model for $(s_j, \mathrm{y}_j)$ by applying gradient descent step on $\left(\mathrm{y}_j - \tilde{Q}_j\right)^2$ with
          respect to $\mathrm{w}$;
14:    end for
15:    if $(t \bmod \tau) = 0$ then
16:       $\hat{\mathrm{w}} \leftarrow \rho\mathrm{w} + (1-\rho)\hat{\mathrm{w}}$;
17:    end if
18:    if $\mathrm{w}$ converges then
19:       $\mathrm{w}^* \leftarrow \mathrm{w}$;
20:       break
21:    end if
22: end for
23: Use $\mathrm{w}^*$ to estimate $Q^*(s,a)$ required for $\pi^*$ using (4.8).
```

Algorithm 4 shows how the EC learns the optimal policy π^* for the considered MDP. It requires the EC design parameters k, $\mathcal{N}$, N_i, and u_h, and selecting the DNN hyper-parameters γ, the target update rate ρ, the probability ϵ of making a random action for exploration, the replay memory capacity D to store the observations (s, a, r, s'), the minibatch size n of samples used to train the DNN model and update its weights w, and the data of the IoV and smart city users u, c, h. Note that u, c and h can be real data from the IoV and smart city environment, as well as from simulations if the probability distributions are known. The DNN target model at line 2 is used to stabilize the DNN model and avoid divergence by reducing the correlation between the action-values $Q(s, a)$ and the targets $\mathrm{y} = r + \gamma \max_{a'} Q(s', a')$ through only periodical updates of the target model weights $\hat{\mathrm{w}}$. In each iteration, we take an action and observe the collected reward and the successor state. Actions are taken according to a policy π such as the ϵ-greedy policy in which a random action with probability ϵ is taken to explore new rewards, and an optimal action (see (4.8)) is taken with probability $(1 - \epsilon)$ to maximize the rewards. Model is trained using experience replay as shown in lines 9-14. At line 9, a minibatch of n random observations is sampled from $\mathbb{M}$. The randomness in selecting samples eliminates correlations in the observations to avoid model overfitting. At line 11, we estimate the output vector $\hat{Q}$ of the target model for a given input state s in each experience sample using the target model weights $\hat{\mathrm{w}}$. $\tilde{Q}$ and $\hat{Q}$ are the predicted vectors of the $k+1$ Q-values for a given state s with respect to w and $\hat{\mathrm{w}}$, respectively. The way to compute the target for sample j is shown at line 12. At line 13, we update the model weights w by fitting the model for the input states and the corresponding targets. A gradient decent step is applied to minimize the squared loss $\left(\mathrm{y}_j - \tilde{Q}_j\right)^2$ between the target and the model predictions. The gradient descent converges to the global minima of the quadratic loss function when the DNN is over-parametrized, i.e., with large training data and large number of hidden neurons [76–78]. The target model weights are periodically updated every τ time steps as shown at line 16, where the update rate ρ exemplifies how much we believe in our experience. The algorithm stops when the

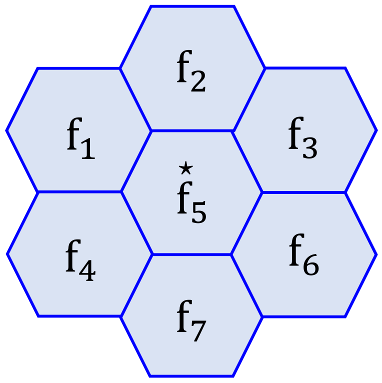


Figure 4.4: The structure of the edge cluster considered in the simulations. The neighboring lists $\mathcal{N}_1, \mathcal{N}_2, \dots, \mathcal{N}_7$ include the adjacent FNs only.

DNN model weights w converge. The converged values are then used to determine optimal actions, i.e., π^* as in (4.8).

4.4 Simulations

We next provide simulation results to evaluate the performance of the proposed network slicing approach in dynamic IoV and smart city environments under different performance evaluation criteria. We compare the DRL algorithm given in Algorithm 4 with the serve-all-utilities (SAU) algorithm in which the EC serves all coming tasks when requested resources are available, serve-high-utilities (SHU) algorithm where the EC filters high-utility requests and serve them if the available resources are enough, and the QL algorithm independently running at each FN following a local version of our MDP formulation [3]. The QL algorithm at each FN corresponds to the non-cooperative scenario, hence this comparison will help evaluate the importance of cooperation among FNs. In the non-cooperative scenario, each FN operates as a standalone entity with no neighbors to handover tasks when busy, and no EC to manage the edge resources.

Table 4.1: Utility distributions corresponding to a variety of latency requirements of IoV and smart city applications in various environments

	$\mathcal{E}_1$	$\mathcal{E}_2$	$\mathcal{E}_3$	$\mathcal{E}_4$	$\mathcal{E}_5$
$P(u = 1)$	0.015	0.012	0.008	0.004	0.001
$P(u = 2)$	0.073	0.058	0.038	0.019	0.004
$P(u = 3)$	0.365	0.288	0.192	0.096	0.019
$P(u = 4)$	0.292	0.230	0.154	0.077	0.015
$P(u = 5)$	0.205	0.162	0.108	0.054	0.011
$P(u = 6)$	0.014	0.071	0.142	0.214	0.271
$P(u = 7)$	0.013	0.064	0.129	0.193	0.244
$P(u = 8)$	0.011	0.057	0.114	0.171	0.217
$P(u = 9)$	0.009	0.043	0.086	0.129	0.163
$P(u = 10)$	0.003	0.015	0.029	0.043	0.055
$P(u \geq u_h = 8)$	2.3%	11.5%	22.9%	34.3%	43.5%
$\bar{u}$	3.82	4.589	5.55	6.5	7.27

Table 4.2: Simulation setup

Parameter	Description	Value
N_i	resource capacity of FN f_i	7
C	set of possible resource blocks	$\{1, 2, 3, 4\}$
H	set of possible holding times	$5\times\{1, 2, 3, 4, 5, 6\}$
ω_g	weight for GoS	$\{0.7, 0.5, 0.3\}$
ω_u	weight for resource utilization	$\{0.3, 0.5, 0.7\}$
u_h	threshold for a “high-utility"	8
D	capacity of DNN replay memory	2000
γ	reward discount factor	0.9
α	learning rate	0.01
ϵ	probability of random action	1.0 with 0.9995 decay
n	batch size	32
τ	$\hat{\mathrm{w}}$ update interval	1000
ρ	$\hat{\mathrm{w}}$ update rate	0.2

Table 4.3: Considered rewarding systems

Scenario	ω_g	ω_u	R	$\{r_{sh},\ r_{rh},\ r_{bh},\ r_{sl},\ r_{rl},\ r_{bl}\}$	r_L
1	0.7	0.3	R_1	$\{24, -12, -12, -3, 3, 12\}$	(see (4.7))
2	0.5	0.5	R_2	$\{24, -12, -12, 0, 0, 12\}$	(see (4.7))
3	0.3	0.7	R_3	$\{50, -50, -50, 50, -50, -25\}$	0

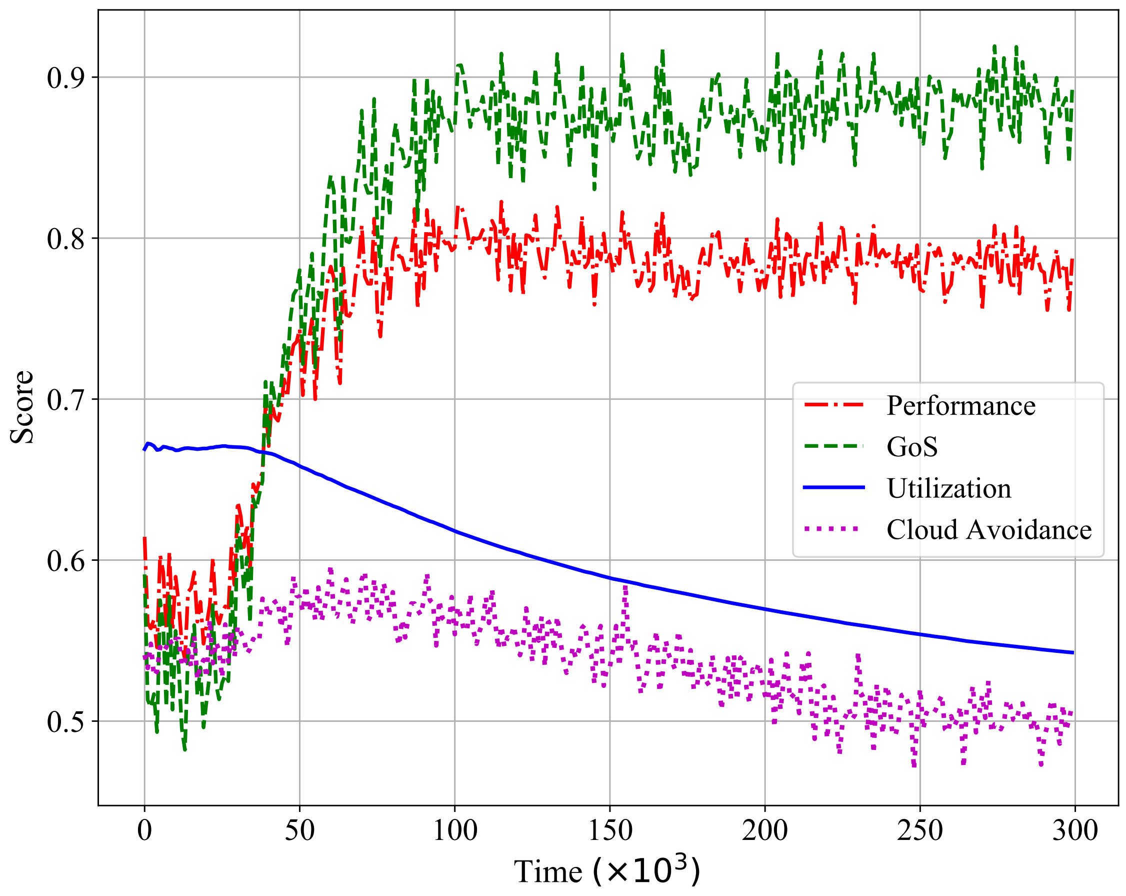


Figure 4.5: The performance and main network KPIs for DQN-based EC while learning the optimum policy in the IoV and smart city environment $\mathcal{E}_3$ under scenario 1 of Table 4.3.

4.4.1 Simulation Environments

We evaluate the performances in various IoV and smart city environments with different compositions of user utilities. Specifically, we consider 10 utility classes that represent different latency requirements to exemplify the variety of IoV and smart city applications in an F-RAN setting. Considering Table 2.1 and by changing the distribution of utility classes we generate 5 IoV and smart city environments as summarized in Table 4.1. Higher density of high-utility applications makes the IoV and smart city environment richer in terms of URLLC applications. Denoting an IoV and smart city environment of a particular utility distribution with $\mathcal{E}$, we show in Table 4.1

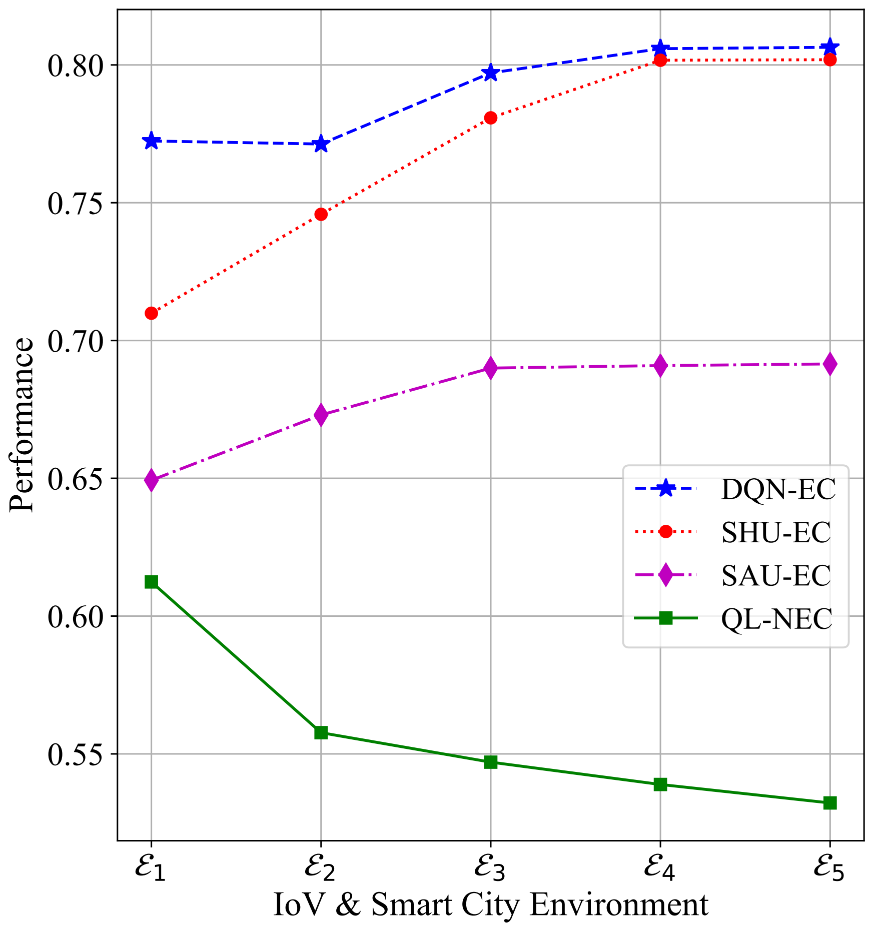


Figure 4.6: The performance of the edge slice when the EC applies the DRL Algorithm 4, SAU and SHU for the coordinated FNs, and the uncoordinated QL based FNs case with NEC considering scenario 1 in Table 4.3 with $\omega_g = 1 - \omega_u = 0.7$.

the statistics of $\mathcal{E}_1$, $\mathcal{E}_2$, $\mathcal{E}_3$, $\mathcal{E}_4$, and $\mathcal{E}_5$. The probabilities in the first 10 rows in Table 4.1 present detailed information about the proportion of each utility class in the environment corresponding to the latency requirement of diverse IoV and smart city applications. The last two rows interpret the quality or richness of IoV and smart city environments, where $\bar{u}$ is the mean of utilities in an environment, and $P(u \geq u_h)$ is the percentage of high-utility population. We started with a general environment given by $\mathcal{E}_3$ for the following IoV and smart city applications corresponding to the utility values $1, 2, \ldots, 10$, respectively: smart lighting and automation of public buildings, air quality

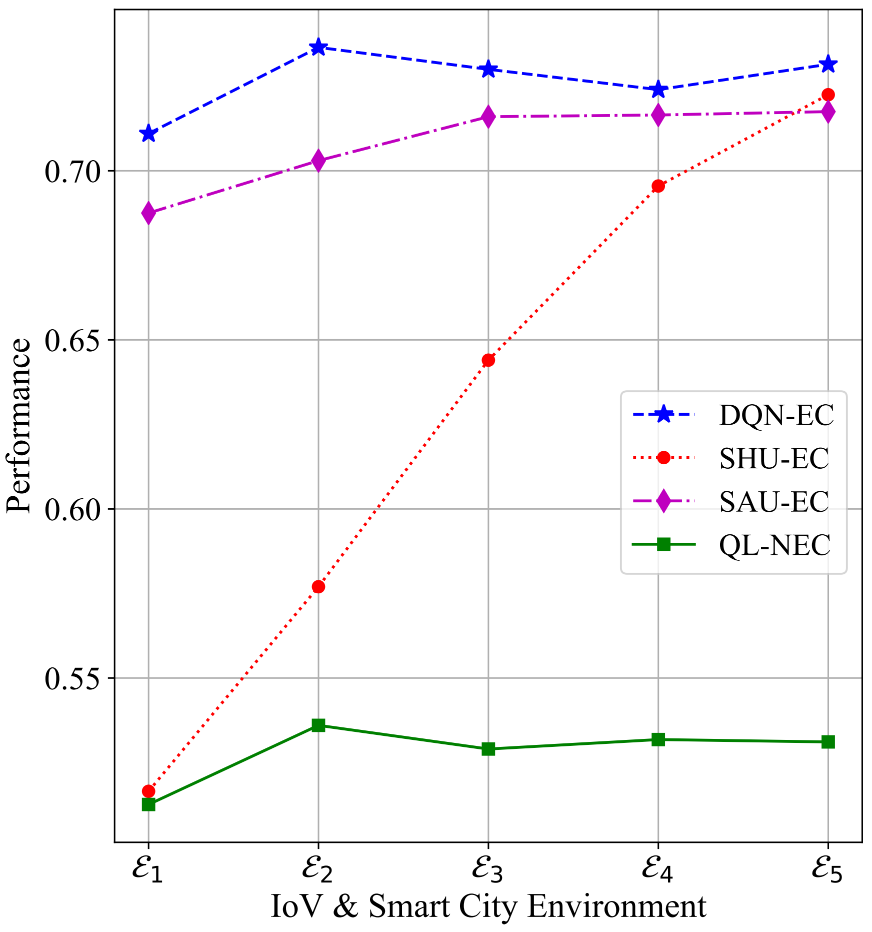


Figure 4.7: The performance of the edge slice when the EC applies the DRL Algorithm 4, SAU and SHU for the coordinated FNs, and the uncoordinated QL based FNs case with NEC considering scenario 2 in Table 4.3 with $\omega_g = 1 - \omega_u = 0.5$.

management and noise monitoring, smart waste management and energy consumption management, smart parking assistance, in-vehicle audio and video infotainment, driver authentication service, structural health monitoring, safe share rides, smart amber alerting system and AI-driven and video-analytics tracking services, driver distraction alerting system and autonomous driving. Then, we changed the utility distribution to obtain the other environments.

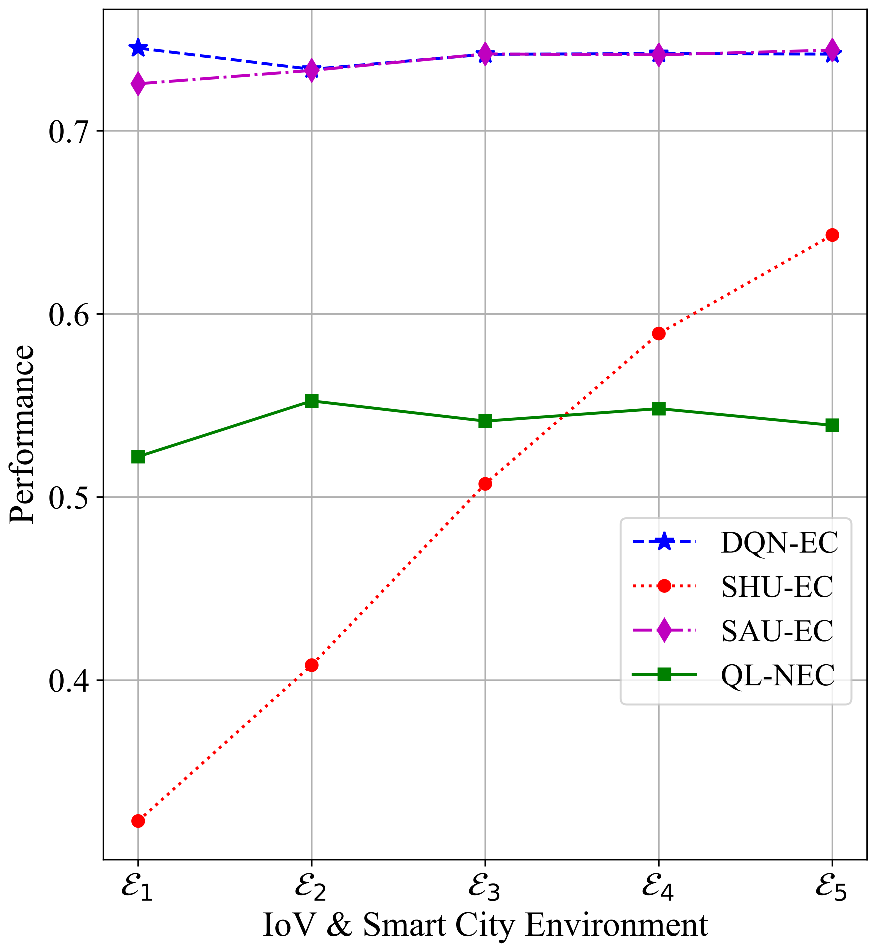


Figure 4.8: The performance of the edge slice when the EC applies the DRL Algorithm 4, SAU and SHU for the coordinated FNs, and the uncoordinated QL based FNs case with NEC considering scenario 3 in Table 4.3 with $\omega_g = 1 - \omega_u = 0.3$.

4.4.2 Simulation Parameters

The simulation parameters used in this section are summarized in Table 4.2. We consider an edge cluster of size $k = 7$, where each FN has a computing and processing resource capacity of seven resource blocks, i.e., $N = 7$. The central FN f_5 acts as the EC, and the neighboring relationships are shown in Fig. 4.4. In a particular IoV and smart city environment $\mathcal{E}$, the threshold that defines “high utility" is set to $u_h = 8$, i.e., $u \in \{8, 9, 10\}$ is a high-utility application with higher priority for edge service. To make the resource allocation of the network slicing problem more challenging, we consider a request arrival rate of at least five times the task execution rate, i.e., holding times increment by five times the arrival interval. The probabilities of $c \in C = \{1, 2, 3, 4\}$ are 0.1, 0.2, 0.3, and 0.4, respectively, whereas the probabilities of $h \in H = \{5, 10, 15, 20, 25, 30\}$ are 0.05, 0.1, 0.1, 0.15, 0.2, and 0.4, respectively.

We consider a fully connected DNN structure for DQN with an input layer of 18 neurons, 2 hidden layers of 64 and 24 neurons, respectively, and an 8-neuron output layer. Linear activation function is used at the output layer and ReLU activation is considered for the other layers. The Huber loss function and the RMSprop optimizer are considered with 0.01 learning rate, 10^{-4} learning decay, and momentum of 0.9. The ϵ-greedy policy is adopted in DNN training where ϵ starts at 1.0 for 10% of the time in training and then decays at a rate of 0.9995 to a minimum value of 10^{-3} to guarantee enough exploration over time. As it depends on the nature of the problem, there is no rule of thumb to tune DNNs. However, the key factors and main DNN hyper-parameters to optimize for quick convergence include the loss function, the optimizer, the interval τ to update target weights, the update rate ρ, the exploration rate ϵ, the learning rate α, the discount factor γ, the randomness of the samples and the batch size n, replay memory size D, and a proper rewarding system to expedite the learning. The values of all hyper-parameters in this section are chosen based on extensive experiments.

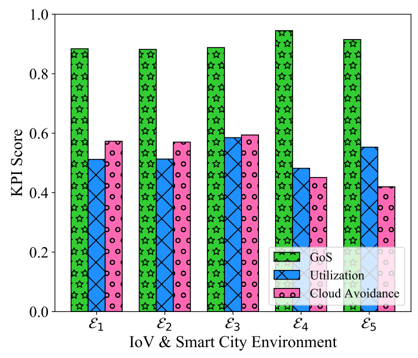
We examine the KPIs explained in Sec. 4.2.1, GoS, resource utilization, cloud avoidance, as well as the overall performance (see (4.4)-(4.3)) considering the three scenarios shown in Table 4.3 with the weights $\omega_g = 1 - \omega_u = 0.7$, $\omega_g = \omega_u = 0.5$, and $\omega_g = 1 - \omega_u = 0.3$. Each scenario in Table 4.3 represents a new problem, hence the rewarding systems R_1, R_2, and R_3 are chosen to facilitate learning the optimal policy in each scenario. The two reward components, $r_{(a,u)} \in$

$\{r_{sh}, r_{rh}, r_{bh}, r_{sl}, r_{rl}, r_{bl}\}$ and r_L for each rewarding system are provided in Table 4.3. Note that unlike R_2 and R_3, R_1 encourages rejecting low-utility requests with higher loads to accommodate the performance requirement of scenario 1, which puts higher weight on GoS with $\omega_g = 0.7$. On the other hand, R_3 promotes serving regardless of the request utility and the task load as the performance in scenario 3 is in favor of achieving higher resource utilization with $\omega_u = 0.7$.

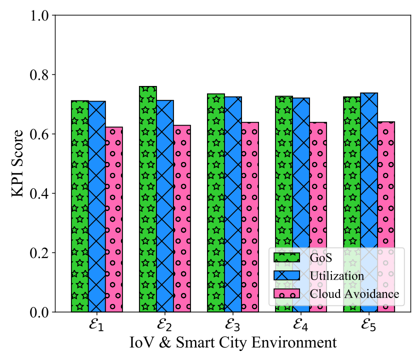
4.4.3 Simulation Results

We train the DRL agent at the EC in various environments and under different performance scenarios provided in Tables 4.1 and 4.3, respectively. By interaction with the environment as illustrated in Fig. 4.3, the EC learns the optimal policy using the DQN method given in Algorithm 4. Considering the environment $\mathcal{E}_3$ and the performance scenario 1, Fig. 4.5 shows an example for the learning curve of the proposed DQN-based EC in terms of the overall performance and KPIs which quickly converge to the optimal scores. Starting with exploration through taking random actions for 30k time steps, the EC initially performs improperly and provides a relatively low GoS (i.e., many high-utility requests are missed) while utilizing the resources mainly for low-utility requests. However, as the algorithm learns the optimum actions from reward feedback, the exploration rate decays and the performance starts to improve. As a result, the EC quickly aligns with the objectives of scenario 1 putting more emphasis on GoS by prioritizing high-utility users for edge service.

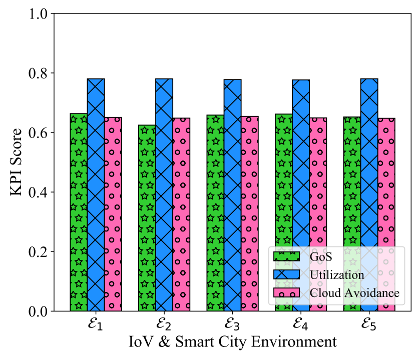
Next, we compare DQN-EC given in Algorithm 4 with SAU-EC, SHU-EC, and QL with no EC (QL-NEC) under the three scenarios given in Table 4.3. Figs. 4.6-4.8 show that the DRL-based EC adapts to each scenario and outperforms the other algorithms in all IoV and smart city environments. For scenario 1 in Fig. 4.6, SHU-EC has a comparable performance to DQN-EC because SHU algorithm promotes serving high-utility requests all the time, which matches with the



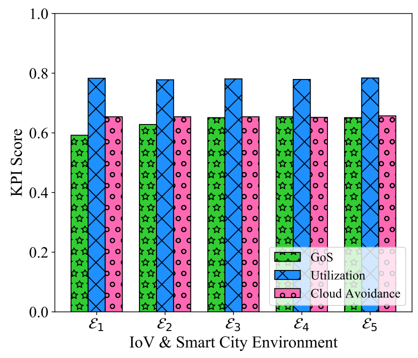
(a) DQN-EC, $\omega_g = 0.7$ and $\omega_u = 0.3$.



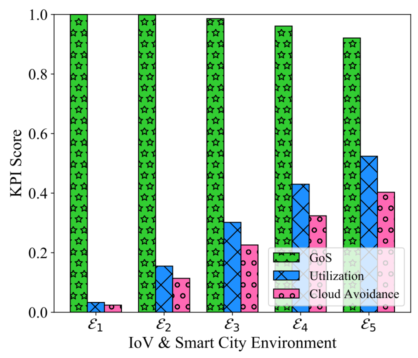
(b) DQN-EC, $\omega_g = \omega_u = 0.5$.



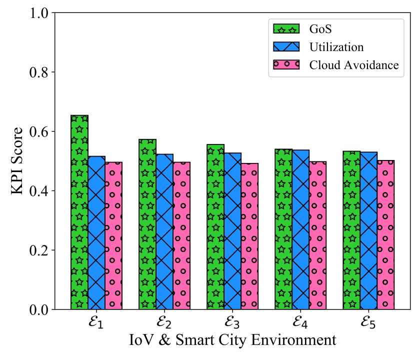
(c) DQN-EC, $\omega_g = 0.3$ and $\omega_u = 0.7$.



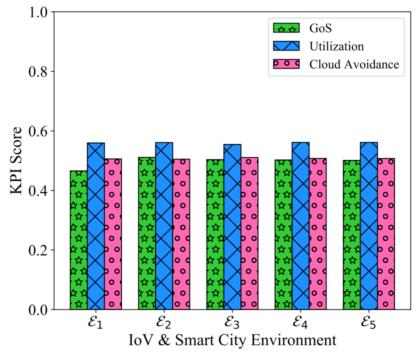
(d) SAU-EC, all scenarios.



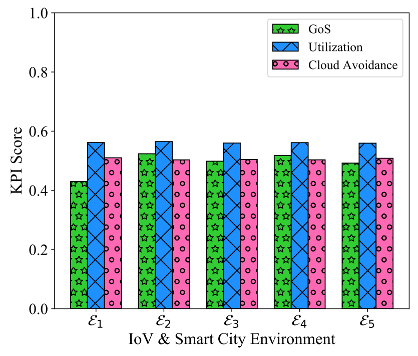
(e) SHU-EC, all scenarios.



(f) QL-NEC, $\omega_g = 0.7$ and $\omega_u = 0.3$.



(g) QL-NEC, $\omega_g = \omega_u = 0.5$.



(h) QL-NEC, $\omega_g = 0.3$ and $\omega_u = 0.7$.

Figure 4.9: The score of the main three individual KPIs, GoS, resource utilization, and cloud avoidance when the EC applies Algorithm 4, SAU and SHU for coordinated FNs, and the uncoordinated QL at FNs with no EC, under the 3 scenarios in Table 4.3.

focus on GoS in scenario 1 design objective with $\omega_g = 0.7$. However, in poor IoV and smart city environments with less high-utility population such as $\mathcal{E}_1$ the performance gap increases. This gap shrinks as environment becomes richer and SHU-EC achieves a performance as high as the DQN-EC score in $\mathcal{E}_4$ and $\mathcal{E}_5$. The performance of SAU-EC slightly increases while moving from $\mathcal{E}_1$ to $\mathcal{E}_3$ and becomes stable afterwards even for the richer environments $\mathcal{E}_4$ and $\mathcal{E}_5$ since SAU-EC does not prioritize high-utility tasks. Unlike the other algorithms, QL-NEC shows a declining trend since the network slicing problem becomes more challenging with uncoordinated FNs while moving towards richer environments in this scenario. Fig. 4.7 represents scenario 2 with equal weights for GoS and resource utilization, where SAU-EC is the second performing algorithm following DQN-EC. With less importance for GoS, the performance of SHU-EC is as low as the QL-NEC in $\mathcal{E}_1$ and although it grows while moving to richer environments, it does not reach a comparable level until $\mathcal{E}_4$ and $\mathcal{E}_5$. The uncoordinated FNs with QL-NEC is more steady in scenario 2. Fig. 4.8 shows the performances in scenario 3 in which more emphasis is put on resource utilization than GoS with $\omega_u = 0.7$. It is observed that SHU-EC fails to achieve a comparable level of performance compared to DQN-EC while SAU-EC does.

Fig. 4.9 provides the detailed KPI scores for GoS, resource utilization and cloud avoidance for all algorithms considering the three design scenarios in all environments. DQN-EC always adapts to the design objective and the IoV and smart city environment. It maximizes GoS in scenario 1 as shown in Fig. 4.9a, balances GoS and utilization for scenario 2 as observed in Fig. 4.9b, and promotes resource utilization for scenario 3 as shown in Fig. 4.9c. QL-NEC in Figs. 4.9f-4.9h tries to behave similarly as it learns by interaction, but unfortunately the uncoordinated FNs in the edge slice cannot achieve that. Note that, DQN-EC learns the right balance between GoS and resource utilization in each scenario. For instance, even though SHU-EC is the second performing in Fig. 4.6 following DQN-EC, it has lower utilization and cloud avoidance scores, i.e., less edge-slice contribution to handle service requests as shown in Fig. 4.9e. Similarly, SAU-EC is well-performing in scenario 2 compared to DQN-EC as shown in Fig. 4.7, however, it does not learn to balance GoS and utilization as DQN-EC does in Fig. 4.9b.

Finally, we test the performance of the proposed DQN algorithm in a dynamic IoV and smart city environment. In Fig. 4.10, we consider the design objectives of scenario 1 in Table 4.3 and a sampling rate of 5×10^{-4}. To generate a dynamic IoT environment we start with 40 samples for the initial environment and then change $\mathcal{E}$ every 30 samples. More samples is considered for the initial $\mathcal{E}$ since we start with vacant resource blocks for all FNs in the edge slice. We consider a dynamic IoV and smart city environment whose composition of high-utility requests, i.e., low-latency tasks, changes over a day. Starting in the morning busy hours with $\mathcal{E}_4$, the density of high-utility requests drops over time to $\mathcal{E}_1$ during the late morning hours. It starts growing to reach $\mathcal{E}_2$ by noon and $\mathcal{E}_3$ in the evening, and then peaks again during the night busy hours with $\mathcal{E}_5$. These 5 environments represent different distributions over diverse levels of utilities, i.e., different latency requirements of the various IoV and smart city applications. Hence, they can be thought as different traffic profiles during the day in terms of the required QoS. The changes during the day directly affect the overall distribution of the environment over time and makes it dynamic. In the proposed algorithm, once the EC detects the traffic profile, i.e., the environment, it applies the corresponding optimal policy π_4^* given in (4.8) to maximize the expected rewards in $\mathcal{E}_4$. Right after the density of low-latency tasks drops over time to $\mathcal{E}_1$, i.e., at $t = 80$K, the EC keeps following π_4^* until it detects the change from the statistics of task utilities, which results in a slight degradation in its performance since π_4^*

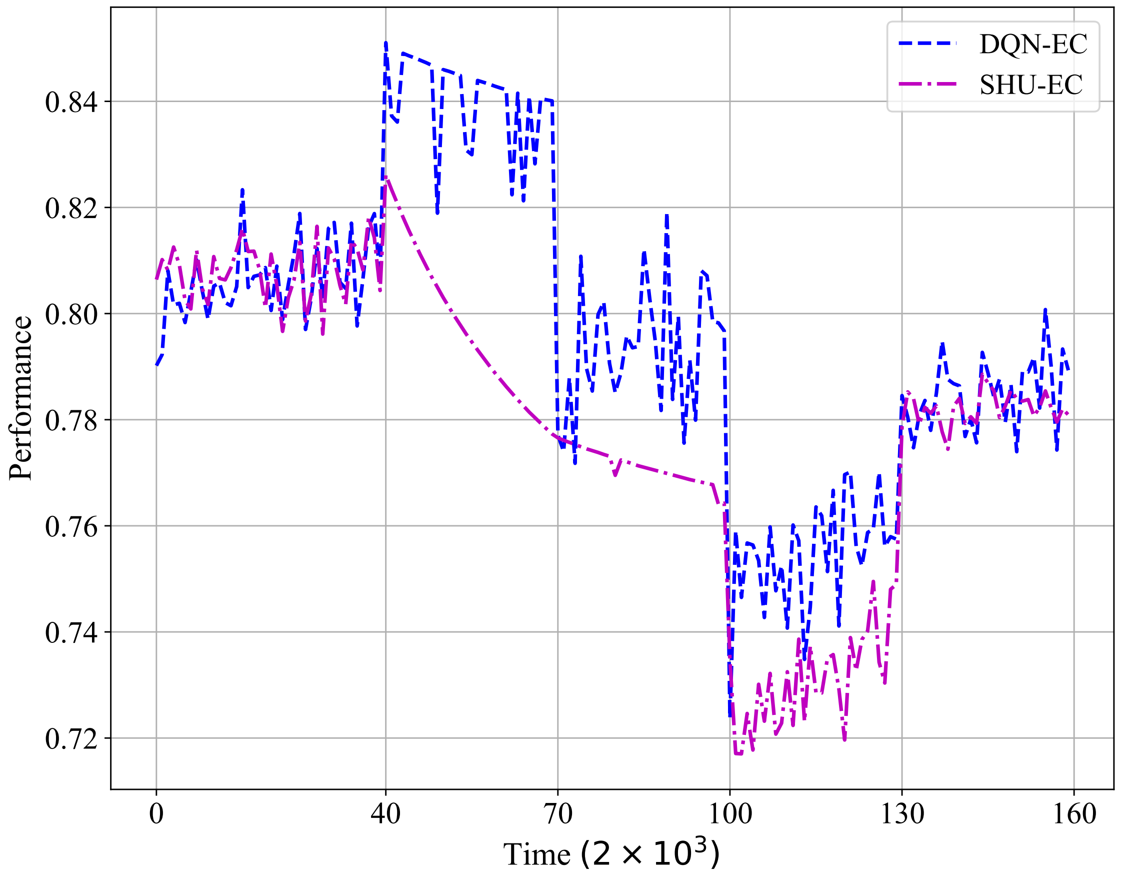


Figure 4.10: The performance of the proposed DQN and the straightforward SHU policy for network slicing in a dynamic IoV and smart city environment considering the design objective of scenario 1 in Table 4.3. Although SHU performs well in rich environments, it cannot adapt to the other environments as expected. The proposed DQN policy on the other hand learns to adapt to different environments.

is no longer optimal for the new environment $\mathcal{E}_1$. However, after a short learning period, the EC adapts to the dynamics, and switches to the new optimal policy π_1^*. Similarly, as seen for the other transitions from $\mathcal{E}_1$ to $\mathcal{E}_2$, $\mathcal{E}_2$ to $\mathcal{E}_3$, and $\mathcal{E}_3$ to $\mathcal{E}_5$, DQN-EC successfully adapts to the changing IoV and smart city environments. Whereas, the straightforward SHU-EC policy performs well in only the rich environments for which it was designed, and cannot adapt to the changes in the environment as expected.

4.5 Summary

In this chapter, we developed an infinite-horizon Markov decision process (MDP) formulation for the network slicing problem in a realistic fog-RAN with cooperative fog nodes. A deep reinforcement learning (DRL) solution is proposed for the edge controllers (ECs), which are the fog nodes that serve as cluster heads, to learn the optimal policy of allocating the limited edge computing and processing resources to vehicular and smart city applications with heterogeneous latency needs and various task loads. The deep Q-Network (DQN) based EC quickly learns the dynamics through interaction with the environment and adapts to it. DQN-EC dominates the straightforward and non-cooperative RL approaches as it always learns the right balance between GoS and resource utilization under different performance objectives and environments. In a dynamic environment with changing distributions, DQN-EC adapts to the dynamics and updates its optimal policy to maximize the performance.

Chapter 5: Conclusions and Future Directions

This dissertation is an effort towards improving the quality of service (QoS) for Internet of Things (IoT) users with various latency and computing demands through intelligently utilizing the edge resources in fog RAN (F-RAN). The considered network slicing problem of sequentially allocating the computing, signal processing, radio, and storage resources at the fog nodes (FNs) to intelligent vehicular and smart city applications with heterogeneous requirements is formulated as finite-horizon and infinite-horizon Markov decision processes (MDPs). A single FN slicing model is first developed with a tractable number of states, where tabular solution methods such as dynamic programming and model-free RL methods, in particular Monte Carlo, SARSA, expected SARSA, and Q-Learning (QL), were used to learn the optimum slicing policy.

The presented resource allocation framework in Chapter 2 and Chapter 3 is then expanded to more challenging scenarios such as collaborative resource allocation with multiple fog nodes for heterogeneous service times and number of resource blocks. To this end, a network slicing model based on a cluster of FNs coordinated with an edge controller (EC) with high-dimensional state space is developed. Deep RL (DRL) methods such as deep Q-networks (DQN) is adopted to address the high-dimensionality in learning the optimal policy of allocating the limited edge computing and processing resources to vehicular and smart city applications with heterogeneous latency needs and various task loads.

Various IoT environments with different latency and diverse task loads compositions were considered in the simulations to evaluate the performance of the proposed dynamic network slicing approaches. The numerical results corroborated the fact that MDP methods adapt to the environment by learning the optimum policy from experience. We showed that the dynamic MDP-based slicing methods always dominate the static threshold-based slicing methods, which do not learn from the environment. The RL methods can strike a right balance between the two conflicting objectives, maximizing the average total served utility vs. minimizing the FN idle time, which helps utilize limited FN resources efficiently. The deep Q-Network (DQN) based EC quickly learns the dynamics through interaction with the environment and adapts to it. DQN-EC dominates the straightforward and non-cooperative RL approaches as it enables collaboration among a set of neighboring FNs. In a dynamic environment with changing distributions, DQN-EC adapts to the dynamics and updates its optimal policy to maximize the performance.

As future work, the presented DRL based network slicing framework can be expanded by developing novel multi-agent algorithms to achieve the desired self-optimized network structure which can autonomously, timely, and softly configure its end-to-end resources to satisfy the needs for diverse IoT applications and maximize the overall network performance. The multi-agent DRL solutions for the interactions between intelligent vehicles and AI-enabled communication infrastructure can be extended to other areas like robotics, industry, agriculture, transportation, and various smart city applications. Another direction is to develop novel algorithms to improve the learning performance in multi-objective problems.

References

[1] A. Nassar and Y. Yilmaz. Dynamic network slicing for fog radio access networks. In *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1–5, 2019.

[2] A. Nassar and Y. Yilmaz. Resource allocation in fog ran for heterogeneous iot environments based on reinforcement learning. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–6, 2019.

[3] A. Nassar and Y. Yilmaz. Reinforcement learning for adaptive resource allocation in fog ran for iot with heterogeneous latency requirements. *IEEE Access*, 7:128014–128025, 2019.

[4] Almuthanna Nassar and Yasin Yilmaz. Deep reinforcement learning for adaptive network slicing in 5g for intelligent vehicular systems and smart cities. Accepted, May 2021, *IEEE Internet of Things Journal*, submitted, Dec. 2020, revised, Mar. 2021. *arXiv* preprint *arXiv*:2010.09916.

[5] Cisco Syst. Cisco annual internet report (2018—2023) white paper, Updated: Mar, 2020. Cisco Syst., Corporate Headquarters, San Jose, CA, USA, White Paper. Accessed: Jun. 10, 2021. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf?dtid=osscdc000283.

[6] Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 2016-–2021, 2017. White Paper, [Online]. Available: https://www.ramonmillan.com/documentos/bibliografia/VisualNetworkingIndexGlobalMobileDataTrafficForecastUpdate2016_Cisco.pdf (Accessed: Jun. 10, 2021).

[7] Kathy Winter. For self-driving cars, there's big meaning behind one big number: 4 terabytes. Intel.com. https://www.intc.com/news-events/press-releases/detail/237/intel-editorial-for-self-driving-cars-theres-big (Accessed Jun. 10, 2021).

[8] AlMuthanna Turki Nassar, Ahmed Iyanda Sulyman, and Abdulhameed Alsanie. Achievable rf coverage and system capacity using millimeter wave cellular technologies in 5g networks. In *Electrical and Computer Engineering (CCECE), 2014 IEEE 27th Canadian Conference on*, pages 1–6. IEEE, 2014.

[9] Ahmed Iyanda Sulyman, AlMuthanna T Nassar, Mathew K Samimi, George R MacCartney, Theodore S Rappaport, and Abdulhameed Alsanie. Radio propagation path loss models for 5g cellular networks in the 28 ghz and 38 ghz millimeter-wave bands. *IEEE Communications Magazine*, 52(9):78–86, 2014.

[10] Binqi Yang, Zhiqiang Yu, Ji Lan, Ruoqiao Zhang, Jianyi Zhou, and Wei Hong. Digital beamforming-based massive mimo transceiver for 5g millimeter-wave communications. *IEEE Transactions on Microwave Theory and Techniques*, 2018.

[11] Sundeep Rangan, Theodore S Rappaport, and Elza Erkip. Millimeter-wave cellular wireless networks: Potentials and challenges. *Proceedings of the IEEE*, 102(3):366–385, 2014.

[12] Jianhua Zhang, Zhe Zheng, Yuxiang Zhang, Jie Xi, Xiongwen Zhao, and Guan Gui. 3d mimo for 5g nr: Several observations from 32 to massive 256 antennas based on channel measurement. *IEEE Communications Magazine*, 56(3):62–70, 2018.

[13] Mugen Peng, Yaohua Sun, Xuelong Li, Zhendong Mao, and Chonggang Wang. Recent advances in cloud radio access networks: System architectures, key techniques, and open issues. *IEEE Communications Surveys and Tutorials*, 18(3):2282–2308, 2016.

[14] Zhongyuan Zhao, Mugen Peng, Zhiguo Ding, Wenbo Wang, and H Vincent Poor. Cluster content caching: An energy-efficient approach to improve quality of service in cloud radio access networks. *IEEE Journal on Selected Areas in Communications*, 34(5):1207–1221, 2016.

[15] Mugen Peng, Chonggang Wang, Vincent Lau, and H Vincent Poor. Fronthaul-constrained cloud radio access networks: Insights and challenges. *IEEE Wireless Communications*, 22(2):152–160, 2015.

[16] Wei Wang, Vincent KN Lau, and Mugen Peng. Delay-aware uplink fronthaul allocation in cloud radio access networks. *IEEE Transactions on Wireless Communications*, 16(7):4275–4287, 2017.

[17] Shuo Wang, Xing Zhang, Yan Zhang, Lin Wang, Juwo Yang, and Wenbo Wang. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access*, 5:6757–6779, 2017.

[18] Yuan-Yao Shih, Wei-Ho Chung, Ai-Chun Pang, Te-Chuan Chiu, and Hung-Yu Wei. Enabling low-latency applications in fog-radio access networks. *IEEE network*, 31(1):52–58, 2017.

[19] Seok-Hwan Park, Osvaldo Simeone, and Shlomo Shamai. Joint optimization of cloud and edge processing for fog radio access networks. In *Information Theory (ISIT), 2016 IEEE International Symposium on*, pages 315–319. IEEE, 2016.

[20] Gerhard P Fettweis. The tactile internet: Applications and challenges. *IEEE Vehicular Technology Magazine*, 9(1):64–70, 2014.

[21] Qiang Zheng, Kan Zheng, Haijun Zhang, and Victor CM Leung. Delay-optimal virtualized radio resource scheduling in software-defined vehicular networks via stochastic learning. *IEEE Transactions on Vehicular Technology*, 65(10):7857–7867, 2016.

[22] Philipp Schulz, Maximilian Matthe, Henrik Klessig, Meryem Simsek, Gerhard Fettweis, Junaid Ansari, Shehzad Ali Ashraf, Bjoern Almeroth, Jens Voigt, Ines Riedel, et al. Latency critical iot applications in 5g: Perspective on the design of radio interface and network architecture. *IEEE Communications Magazine*, 55(2):70–78, 2017.

[23] T. Lin, H. Rivano, and F. Le Mouël. A survey of smart parking solutions. *IEEE Transactions on Intelligent Transportation Systems*, 18(12):3229–3253, 2017.

[24] T. Anagnostopoulos, A. Zaslavsky, K. Kolomvatsos, A. Medvedev, P. Amirian, J. Morley, and S. Hadjieftymiades. Challenges and opportunities of waste management in iot-enabled smart cities: A survey. *IEEE Transactions on Sustainable Computing*, 2(3):275–289, 2017.

[25] P. Barsocchi, P. Cassara, F. Mavilia, and D. Pellegrini. Sensing a city's state of health: Structural monitoring system by internet-of-things wireless sensing devices. *IEEE Consumer Electronics Magazine*, 7(2):22–31, 2018.

[26] Z. Hu, Z. Bai, K. Bian, T. Wang, and L. Song. Real-time fine-grained air quality sensing networks in smart city: Design, implementation, and optimization. *IEEE Internet of Things Journal*, 6(5):7526–7542, 2019.

[27] L. Ruge, B. Altakrouri, and A. Schrader. Soundofthecity - continuous noise monitoring for a healthy city. In *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 670–675, 2013.

[28] P. T. Daely, H. T. Reda, G. B. Satrya, J. W. Kim, and S. Y. Shin. Design of smart led streetlight system for smart city with web-based management system. *IEEE Sensors Journal*, 17(18):6100–6110, 2017.

[29] M. Teliceanu, G. C. Lazaroiu, and V. Dumbrava. Consumption profile optimization in smart city vision. In *2017 10th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*, pages 876–881, 2017.

[30] F. Heimgaertner, S. Hettich, O. Kohlbacher, and M. Menth. Scaling home automation to public buildings: A distributed multiuser setup for openhab 2. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6, 2017.

[31] 3GPP. Study on scenarios and requirements for next generation access technologies (release 14), v14.2.0, Mar., 2017. pp. 23-25. 3GPP, Sophia Antipolis, France, Rep. TR 38.913. Accessed: Jun. 10, 2021. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2996.

[32] Gordon J Sutton, Jie Zeng, Ren Ping Liu, Wei Ni, Diep N Nguyen, Beeshanga A Jayawickrama, Xiaojing Huang, Mehran Abolhasan, Zhang Zhang, Eryk Dutkiewicz, et al. Enabling technologies for ultra-reliable and low latency communications: from phy and mac layer perspectives. *IEEE Communications Surveys & Tutorials*, 21(3):2488–2524, 2019.

[33] J. Wang, J. Liu, and N. Kato. Networking and communications in autonomous driving: A survey. *IEEE Communications Surveys Tutorials*, 21(2):1243–1274, 2019.

[34] R. H. Goudar and H. N. Megha. Next generation intelligent traffic management system and analysis for smart cities. In *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*, pages 999–1003, 2017.

[35] L. Lin, X. Liao, H. Jin, and P. Li. Computation offloading toward edge computing. *Proceedings of the IEEE*, 107(8):1584–1607, 2019.

[36] Hongyu Xiang, Wenan Zhou, Mahmoud Daneshmand, and Mugen Peng. Network slicing in fog radio access networks: Issues and challenges. *IEEE Communications Magazine*, 55(12):110–116, 2017.

[37] Yuvraj Sahni, Jiannong Cao, Shigeng Zhang, and Lei Yang. Edge mesh: A new paradigm to enable distributed intelligence in internet of things. *IEEE access*, 5:16441–16458, 2017.

[38] Ai-Chun Pang, Wei-Ho Chung, Te-Chuan Chiu, and Junshan Zhang. Latency-driven cooperative task computing in multi-user fog-radio access networks. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 615–624. IEEE, 2017.

[39] GM Shafiqur Rahman, Mugen Peng, Kecheng Zhang, and Shanzhi Chen. Radio resource allocation for achieving ultra-low latency in fog radio access networks. *IEEE Access*, 6:17442–17454, 2018.

[40] Jessica Oueis, Emilio Calvanese Strinati, and Sergio Barbarossa. The fog balancing: Load distribution for small cell cloud computing. In *Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st*, pages 1–6. IEEE, 2015.

[41] Te-Chuan Chiu, Wei-Ho Chung, Ai-Chun Pang, Ya-Ju Yu, and Pei-Hsuan Yen. Ultra-low latency service provision in 5g fog-radio access networks. In *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2016 IEEE 27th Annual International Symposium on*, pages 1–6. IEEE, 2016.

[42] Eren Balevi and Richard D Gitlin. Optimizing the number of fog nodes for cloud-fog-thing networks. *IEEE Access*, 6:11173–11183, 2018.

[43] Tianmu Gao, Mingzhe Chen, Hanzhou Gu, and Changchuan Yin. Reinforcement learning based resource allocation in cache-enabled small cell networks with mobile users. 2017.

[44] Duc-Nghia Vu, Nhu-Ngoc Dao, and Sungrae Cho. Downlink sum-rate optimization leveraging hungarian method in fog radio access networks. In *Information Networking (ICOIN), 2018 International Conference on*, pages 56–60. IEEE, 2018.

[45] Yu-Jui Liu, Shin-Ming Cheng, and Yu-Lin Hsueh. enb selection for machine type communications using reinforcement learning based markov decision process. *IEEE Transactions on Vehicular Technology*, 66(12):11330–11338, 2017.

[46] Massimo Condoluci, Toktam Mahmoodi, Eckehard Steinbach, and Mischa Dohler. Soft resource reservation for low-delayed teleoperation over mobile networks. *IEEE Access*, 5:10445–10455, 2017.

[47] Haruna Abdu Manis Name, Francisca O Oladipo, and Ezendu Ariwa. User mobility and resource scheduling and management in fog computing to support iot devices. In *2017 Seventh International Conference on Innovative Computing Technology (INTECH)*, pages 191–196. IEEE, 2017.

[48] Mugen Peng and Kecheng Zhang. Recent advances in fog radio access networks: Performance analysis and radio resource allocation. *IEEE Access*, 4:5003–5009, 2016.

[49] Taehyeun Park, Nof Abuzainab, and Walid Saad. Learning how to communicate in the internet of things: Finite resources and heterogeneity. *IEEE Access*, 4:7063–7073, 2016.

[50] Mu Yan, Gang Feng, and Shuang Qin. Multi-rat access based on multi-agent reinforcement learning. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE, 2017.

[51] Yifei Wei, F Richard Yu, Mei Song, and Zhu Han. User scheduling and resource allocation in hetnets with hybrid energy supply: An actor-critic reinforcement learning approach. *IEEE Transactions on Wireless Communications*, 17(1):680–692, 2018.

[52] Haijun Zhang, Yu Qiu, Keping Long, George K Karagiannidis, Xianbin Wang, and Arumugam Nallanathan. Resource allocation in noma based fog radio access networks. *arXiv preprint arXiv:1803.05641*, 2018.

[53] Nour Mostafa, Ismaeel Al Ridhawi, and Moayad Aloqaily. Fog resource selection using historical executions. In *Fog and Mobile Edge Computing (FMEC), 2018 Third International Conference on*, pages 272–276. IEEE, 2018.

[54] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis, Adlen Ksentini, and Hannu Flinck. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys & Tutorials*, 20(3):2429–2453, 2018.

[55] Tian Dang and Mugen Peng. Delay-aware radio resource allocation optimization for network slicing in fog radio access networks. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2018.

[56] Liya Tang, Xian Zhang, Hongyu Xiang, Yaohua Sun, and Mugen Peng. Joint resource allocation and caching placement for network slicing in fog radio access networks. In *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–6. IEEE, 2017.

[57] Yaohua Sun, Mugen Peng, Shiwen Mao, and Shi Yan. Hierarchical radio resource allocation for network slicing in fog radio access networks. *IEEE Transactions on Vehicular Technology*, 2019.

[58] V. N. Ha and L. B. Le. End-to-end network slicing in virtualized ofdma-based cloud radio access networks. *IEEE Access*, 5:18675–18691, 2017.

[59] Y. L. Lee, J. Loo, T. C. Chuah, and L. Wang. Dynamic network slicing for multitenant heterogeneous cloud radio access networks. *IEEE Transactions on Wireless Communications*, 17(4):2146–2161, 2018.

[60] Y. Shi, Y. E. Sagduyu, and T. Erpek. Reinforcement learning for dynamic resource optimization in 5g radio access network slicing. In *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–6, 2020.

[61] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[62] R. Li, Z. Zhao, Q. Sun, C. I, C. Yang, X. Chen, M. Zhao, and H. Zhang. Deep reinforcement learning for resource management in network slicing. *IEEE Access*, 6:74429–74441, 2018.

[63] Fengsheng Wei, Gang Feng, Yao Sun, Yatong Wang, Shuang Qin, and Ying-Chang Liang. Network slice reconfiguration by exploiting deep reinforcement learning with large action space. *IEEE Transactions on Network and Service Management*, 17(4):2197–2211, 2020.

[64] H. Xiang, S. Yan, and M. Peng. A realization of fog-ran slicing via deep reinforcement learning. *IEEE Transactions on Wireless Communications*, 19(4):2515–2527, 2020.

[65] Y. Hua, R. Li, Z. Zhao, X. Chen, and H. Zhang. Gan-powered deep distributional reinforcement learning for resource management in network slicing. *IEEE Journal on Selected Areas in Communications*, 38(2):334–349, 2020.

[66] Behnam Khodapanah, Ahmad Awada, Ingo Viering, Andre Noll Barreto, Meryem Simsek, and Gerhard Fettweis. Slice management in radio access network via deep reinforcement learning. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–6. IEEE, 2020.

[67] Rongpeng Li, Chujie Wang, Zhifeng Zhao, Rongbin Guo, and Honggang Zhang. The lstm-based advantage actor-critic learning for resource management in network slicing with user mobility. *IEEE Communications Letters*, 24(9):2005–2009, 2020.

[68] Wanqing Guan, Haijun Zhang, and CM Victor Leung. Slice reconfiguration based on demand prediction with dueling deep reinforcement learning. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pages 1–6. IEEE, 2020.

[69] Y. Abiko, T. Saito, D. Ikeda, K. Ohta, T. Mizuno, and H. Mineno. Flexible resource block allocation to multiple slices for radio access network slicing using deep reinforcement learning. *IEEE Access*, 8:68183–68198, 2020.

[70] Yongshuai Liu, Jiaxin Ding, and Xin Liu. A constrained reinforcement learning based approach for network slicing. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–6. IEEE, 2020.

[71] Guolin Sun, Gordon Owusu Boateng, Daniel Ayepah-Mensah, Guisong Liu, and Jiang Wei. Autonomous resource slicing for virtualized vehicular networks with d2d communications based on deep reinforcement learning. *IEEE Systems Journal*, 14(4):4694–4705, 2020.

[72] Yizhen Xu, Zhengyang Zhao, Peng Cheng, Zhuo Chen, Ming Ding, Branka Vucetic, and Yonghui Li. Constrained reinforcement learning for resource allocation in network slicing. *IEEE Communications Letters*, 2021.

[73] S. de Bast, R. Torrea-Duran, A. Chiumento, S. Pollin, and H. Gacanin. Deep reinforcement learning for dynamic network slicing in ieee 802.11 networks. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 264–269, 2019.

[74] G. Sun, Z. T. Gebrekidan, G. O. Boateng, D. Ayepah-Mensah, and W. Jiang. Dynamic reservation and deep reinforcement learning based autonomous resource slicing for virtualized radio access networks. *IEEE Access*, 7:45758–45772, 2019.

[75] R. Sutton, and A. BartoMack. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, USA, 1998.

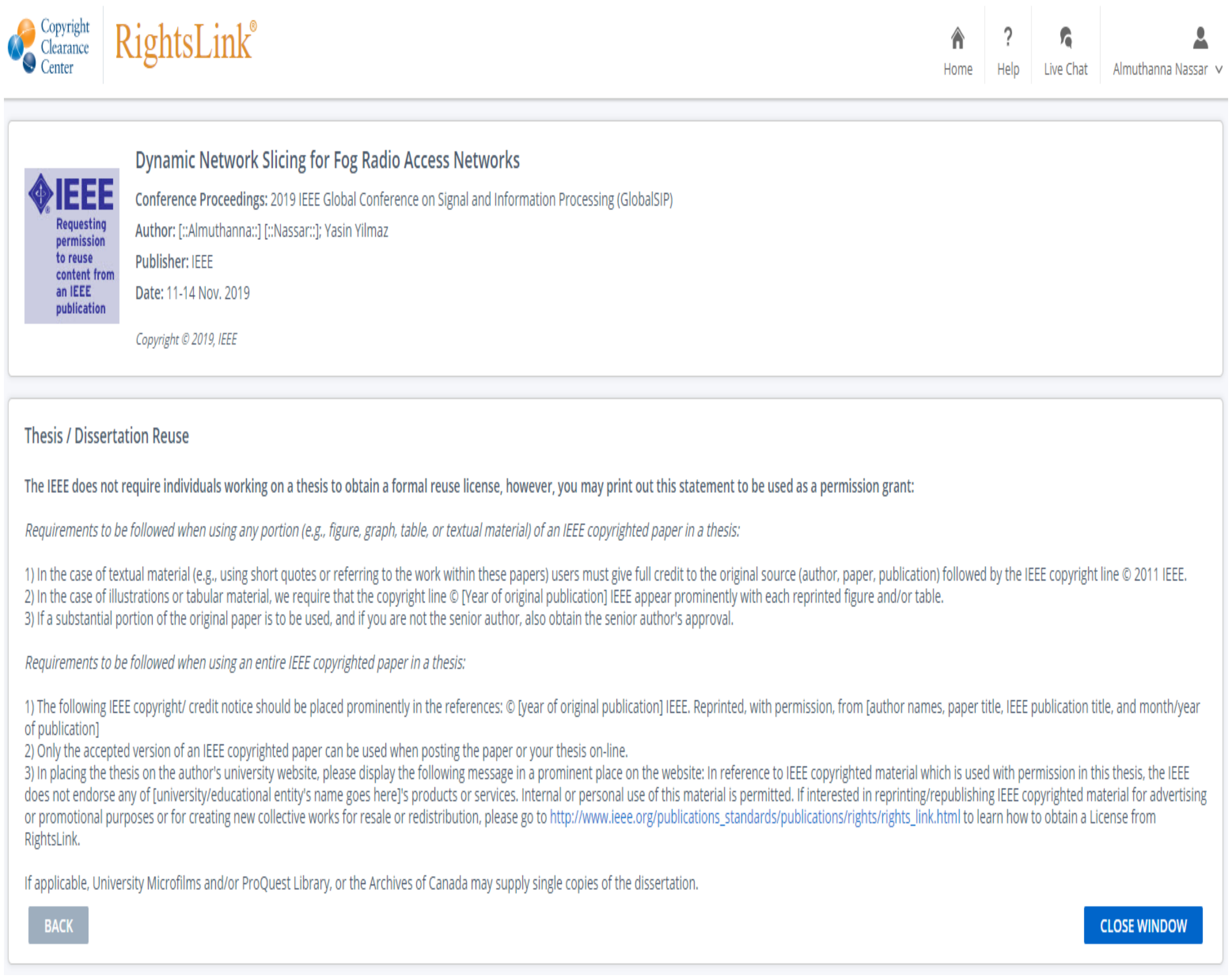
[76] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep q-learning. In *Learning for Dynamics and Control*, pages 486–489. PMLR, 2020.

[77] Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning*, pages 1675–1685. PMLR, 2019.

[78] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*, 2018.

Appendix A: Copyright Permissions

The following copyright permission is for the use of material in Chapter 1 and Chapter 2 which have appeared in [1].


Copyright Clearance Center
RightsLink®
Home
Help
Live Chat
Almuthanna Nassar
IEEE Requesting permission to reuse content from an IEEE publication
BACK
CLOSE WINDOW

Dynamic Network Slicing for Fog Radio Access Networks

Conference Proceedings: 2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)

Author: [::Almuthanna::] [::Nassar::]; Yasin Yilmaz

Publisher: IEEE

Date: 11-14 Nov. 2019

Copyright © 2019, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

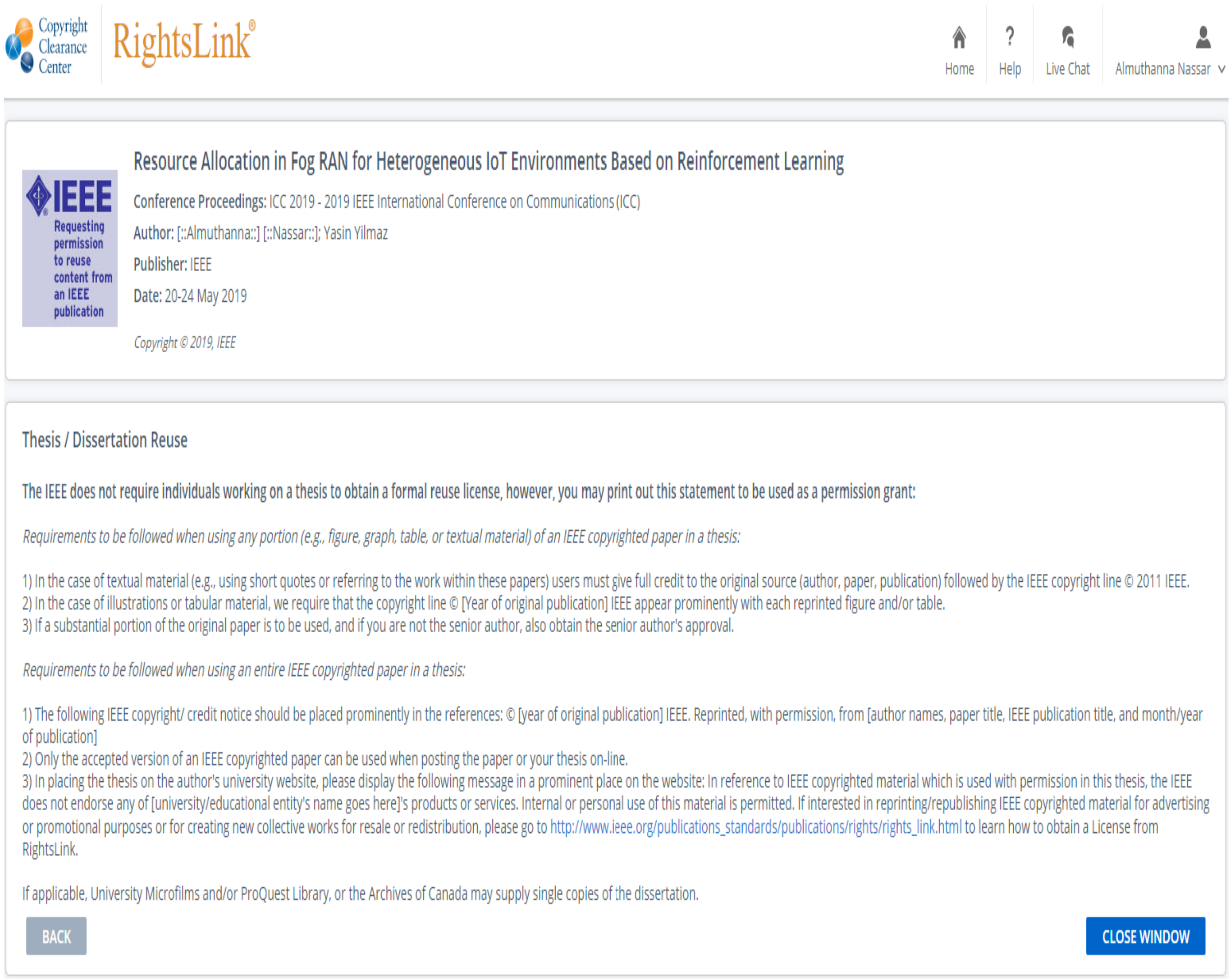
1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

The following copyright permission is for the use of material in Chapter 1 and Chapter 3 which have appeared in [2].


Copyright Clearance Center
RightsLink®
Home
Help
Live Chat
Almuthanna Nassar
Requesting permission to reuse content from an IEEE publication
BACK
CLOSE WINDOW

Resource Allocation in Fog RAN for Heterogeneous IoT Environments Based on Reinforcement Learning

Conference Proceedings: ICC 2019 - 2019 IEEE International Conference on Communications (ICC)

Author: [::Almuthanna::] [::Nassar::]; Yasin Yilmaz

Publisher: IEEE

Date: 20-24 May 2019

Copyright © 2019, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

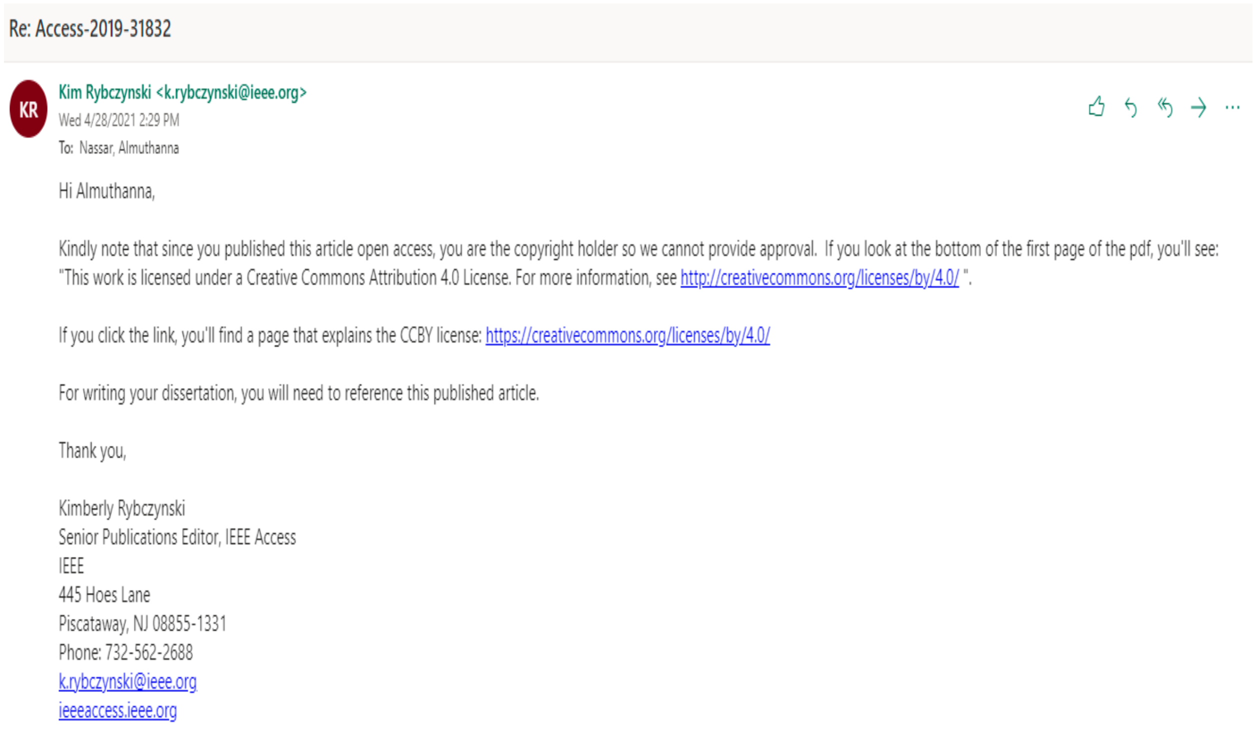
1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

The following copyright permission is for the use of material in Chapter 1, Chapter 2, and Chapter 3 which have appeared in [3].



Re: Access-2019-31832

Kim Rybczynski <k.rybczynski@ieee.org>
Wed 4/28/2021 2:29 PM
To: Nassar, Almuthanna

Hi Almuthanna,

Kindly note that since you published this article open access, you are the copyright holder so we cannot provide approval. If you look at the bottom of the first page of the pdf, you'll see: "This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/ ".

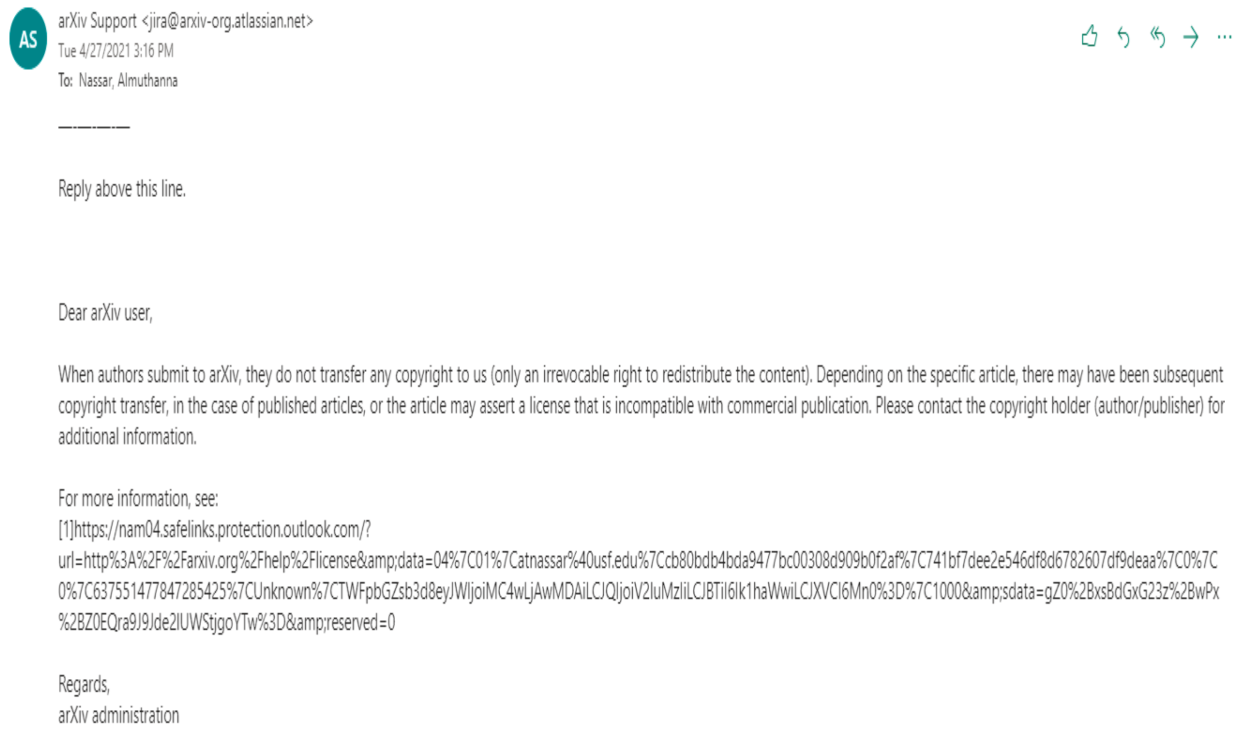
If you click the link, you'll find a page that explains the CCBY license: https://creativecommons.org/licenses/by/4.0/

For writing your dissertation, you will need to reference this published article.

Thank you,

Kimberly Rybczynski
Senior Publications Editor, IEEE Access
IEEE
445 Hoes Lane
Piscataway, NJ 08855-1331
Phone: 732-562-2688
k.rybczynski@ieee.org
ieeeaccess.ieee.org

The following copyright permission is for the use of material in Chapter 1 and Chapter 4 which have appeared in [4].



arXiv Support <jira@arxiv-org.atlassian.net>
Tue 4/27/2021 3:16 PM
To: Nassar, Almuthanna

—-—-—

Reply above this line.

Dear arXiv user,

When authors submit to arXiv, they do not transfer any copyright to us (only an irrevocable right to redistribute the content). Depending on the specific article, there may have been subsequent copyright transfer, in the case of published articles, or the article may assert a license that is incompatible with commercial publication. Please contact the copyright holder (author/publisher) for additional information.

For more information, see:
[1]https://nam04.safelinks.protection.outlook.com/?
url=http%3A%2F%2Farxiv.org%2Fhelp%2Flicense&amp;data=04%7C01%7Catnassar%40usf.edu%7Ccb80bdb4bda9477bc00308d909b0f2af%7C741bf7dee2e546df8d6782607df9deaa%7C0%7C
0%7C637551477847285425%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&amp;sdata=gZ0%2BxsBdGxG23z%2BwPx
%2BZ0EQra9J9Jde2lUWStjgoYTw%3D&amp;reserved=0

Regards,
arXiv administration