UNIVERSITY OF CATANIA

DEPARTMENT OF ELECTRICAL, ELECTRONIC AND COMPUTER ENGINEERING

PhD IN SYSTEM, ENERGY, COMPUTER AND TELECOMMUNICATIONS ENGINEERING

XXXVI CYCLE

Integrating Deep Reinforcement Learning in 6G Edge Environments: Towards Intelligent Network Optimization

---

Raoul Raftopoulos

Coordinator: Prof. Paolo Pietro Arena

Tutor: Prof. Giovanni Schembra

ABSTRACT

The rapid evolution of wireless communication technologies has led to the emergence of 6G networks, which promise unprecedented levels of connectivity, capacity, and intelligence. Edge Intelligence, powered by Artificial Intelligence (AI) techniques, is considered one of the key missing elements in 5G networks and will most likely represent a key enabler for future 6G networks to support their performance, new functions, and services. To fully realize the potential of 6G networks, intelligent network optimization techniques are required. This thesis presents studies on integrating AI in 6G environments toward achieving intelligent network optimization.

Deep Reinforcement Learning (DRL) has shown remarkable success in various domains, such as robotics and gaming, by enabling agents to learn optimal decision-making policies through interactions with their environments. In the context of 6G networks, the integration of DRL offers a promising approach to address complex challenges, such as network resource allocation, dynamic spectrum management, and energy efficiency.

6G networks promise low-latency and high-bandwidth connectivity, enabling a wide range of applications and services. However, the dynamic nature of these environments poses significant challenges in terms of network optimization. Traditional optimization methods struggle to adapt to the dynamic and complex nature of 6G edge environments.

The traditional approach to solving resource allocation problems is through mathematical modeling and optimization. In such a view, the first step is to model the dynamics and performance of the network and to accordingly solve a mathematical problem. While optimal, this approach proves impractical and/or unfeasible. If old-fashioned wireless networks could be easily described and evaluated through mathematical models, the same does not apply to modern networks, which can only be described with either complex but intractable models or simple but inaccurate models. If the model-based approach is doomed to fail, what could instead cope with the increased complexity and variability of wireless networks is AI.

By leveraging the power of neural networks combined with reinforcement learning algorithms, the frameworks proposed in this thesis allow network agents to learn and adapt their behavior autonomously, leading to improved network performance and efficiency. The integration of DRL in 6G environments opens up possibilities for intelligent network planning, self-optimizing networks, and autonomous resource allocation.

The study investigates various aspects of integrating DRL in 6G networks, including the design and training of DRL agents, the definition of suitable reward structures, and the exploration of multi-agent systems for collaborative decision-making. Furthermore, it addresses challenges associated with scalability, convergence, and real-time decision-making in large-scale 6G network environments.

Through extensive simulations and evaluations, the proposed integration of DRL in 6G networks demonstrates promising results in terms of network performance, optimization, and adaptability. By enabling intelligent decision-making and autonomous network optimization, the presented research contributes towards unlocking the full potential of 6G networks and paves the way for intelligent, efficient, and self-adaptive communication systems.

Part I of this thesis explores integrating DRL in Flying ad-hoc Networks (FANETs) 6G Edge Environments for intelligent network optimization. In particular, single-agent, multi-agent, and federated DRL techniques are exploited to enhance resource utilization, horizontal offloading, and positioning.

In Part II we propose a distributed edge-computing multi-agent framework based on Multi-Player Multi-Armed Bandit (MP-MAB) algorithms for latency- and energy-aware job offloading in green vehicular networks.

Part III delves into the design of DRL agents in the context of Open Radio Access Network (O-RAN) able to meet the requirements of different Service Level Agreements (SLAs) while also enabling efficient resource allocation. The designed DRL agent has been validated on the Colosseum platform, the biggest channel emulator in the world.

Keywords: *6G*, *Deep Reinforcement Learning*, *Network Slicing*, *Resources Orchestration*, *O-RAN*.

List of publications

- C. Grasso, R. Raftopoulos and G. Schembra, "Deep Q-Learning for Job Offloading Orchestration in a Fleet of MEC UAVs in 5G Environments," 2021 IEEE 7th International Conference on Network softwarization (Netsoft), Tokyo (Virtual), Japan, 2021, pp 186-190.

- C. Grasso, R. Raftopoulos and G. Schembra, "Smart Zero-Touch Management of UAV-Based Edge Network," Published in IEEE Transactions on Network and Service Management, March 2022, Volume 19, Issue 4.

- L. Galluccio, C. Grasso, G. Maier, R. Raftopoulos, M. Savi, G.Schembra, S. Troia, "Reinforcement Learning for Resource Planning in Drone-Based Softwarized Networks," 2022 Mediterranean Communication and Computer Networking Conference (MedComNet), Paphos, Cyprus, 2022.

- C. Grasso, R. Raftopoulos and G. Schembra, "Tailoring FANET-based 6g network slices in remote areas for low-latency applications," 2022 IFIP Workshop: NI 2022: Network Intelligence, Catania, Italy, 2022.

- C. Grasso, R. Raftopoulos and G. Schembra, "Multi-Agent Deep Reinforcement Learning in Flying Ad-Hoc Networks for Delay-Constrained Applications," 17th International Conference on Future Networks and Communications (FNC), Niagra Falls, Ontario, Canada, 2022.

- C. Grasso, R. Raftopoulos and G. Schembra, "Slicing a FANET for heterogeneous delay-constrained applications," Published in Computer Communications, September 2022, Volume 195, Issue 8.

- C. Grasso, R. Raftopoulos and G. Schembra, "Comparison of Deep Reinforcement Learning Approaches for FANET Optimization," 2022 61st FITCE International Congress Future Telecommunications: Infrastructure and Sustainability (FITCE), Rome, Italy, 2022.

- C. Grasso, R. Raftopoulos and G. Schembra, "A FANET to Provide Blockchain On Demand at the Extreme Edge of a 6G Network," 2022 61st FITCE International Congress Future Telecommunications: Infrastructure and Sustainability (FITCE), Rome, Italy, 2022.

- C. Grasso, R. Raftopoulos, G. Schembra, S. Serrano, "H-HOME: A learning framework of federated FANETs to provide edge computing to future delay-constrained IoT systems," Published in Computer Networks, November 2022, Volume 2019, Issue 1.

- C. Grasso, R. Raftopoulos, G. Schembra, "OSCAR: a Contention Window Optimization approach using Deep Reinforcement Learning," 2023 International Conference on Communications (ICC), Rome, Italy, 2023.

- F. Busacca, S. Palazzo, R. Raftopoulos, G. Schembra, "MANTRA: an Edge-Computing Framework based on Multi-Armed Bandit for Latency- and Energy-aware Job Offloading in Vehicular Networks," IEEE 9th International Conference on Network softwarization (Netsoft), Madrid, Spain, 2023.

- R. Raftopoulos, G. Schembra, "Multi-Armed Bandit for Contention Window Optimization", Submitted to European Wireless (EW) 2023

- R. Avanzato, F. Beritelli, R. Raftopoulos, G. Schembra, "An DRL-based UAV-Smallcell System for Efficiently Localizing Hidden Mobile Devices via RSRP Measurements," Submitted to IEEE Access

Contents

List of Terms and Abbreviations

3GPP	3rd Generation Partnership Project
AI	Artificial Intelligence
AP	Access Point
API	Application Programming Interface
BS	Base Station
CE	Computing Element
CL	Centralized Learning
DL	Deep Learning
DRL	Deep Reinforcement Learning
DT	Digital Twin
DDQN	Double Deep Q-Network
E2E	End-to-End
eMMB	Enhanced Mobile Broadband
EN	Edge Node
eNB	eNodeB
ETSI	European Telecommunication Standards Institute
FANET	Flying Ad-hoc NETwork
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array

FL	Federated Learning
FO	FANET Orchestrator
FRL	Federated Reinforcement Learning
GD	Ground Device
GDG	Ground Device Group
IID	Independent and Identically Distributed
IoT	Internet of Things
IIoT	Intelligent Internet of Things
IIoIT	Intelligent Internet of Intelligent Things
LDO	Local Drone Only
LSTM	Long Short-Term Memory
LTE	Long Term Evolution
LXC	Linux Container
MAC	Medium Access Control
MEC	Multi-Access Edge Computing
MCHEM	Massive Channel Emulator
MGEN	Multi-Generator
ML	Machine Learning
mMTC	Massive Machine Type Communications
MNO	Mobile Network Operator
O-RAN	Open Radio Access Network
PCO	Probabilistic Computation Offloading
PQ	Priority Queueing
QoE	Quality of Experience

QoS	Quality of Service
RAN	Radio Access Network
RF	Radio Frequency
RIC	RAN Intelligent Controller
RL	Reinforcement Learning
RSU	Road Side Unit
SDR	Software Defined Radio
SLA	Service Level Agreement
TGEN	Traffic Generator
TL	Transfer Learning
TO	Telecommunications Operator
UE	User Equipment
UAV	Unmanned Aerial Vehicles
UVS	Unmanned Vehicles System
URLLC	Ultra-reliable and Low Latency Communications
US	Uniform Selection
V2V	Vehicular-to-Vehicular
V2X	Vehicular-to-Everything
VF	Virtual Function
WFQ	Weighted Fair Queuing
ZSM	Zero-touch network and Service Managenent
ZTM	Zero-Touch Management

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, the proliferation of new technologies has transformed the way networks are built, accessed, and data is transmitted and stored. The Internet of Things (IoT) paradigm has led to the widespread adoption of smart terminals, devices, sensors, and video cameras, generating vast amounts of data for processing. As a result, there has been a growing need to extend the centralized cloud computing model toward the edge of the network [1].

The Edge computing paradigm, by bringing service-specific processing and data storage closer to the data generation sources and end-users, has garnered significant interest in the context of 5G networks. Initiatives such as Multi-access Edge Computing (MEC) within European Telecommunication Standards Institute (ETSI) are now focusing on performance improvements, traffic optimization, and ultra-low-latency services that were previously unfeasible. However, meeting these service requirements in an efficient and flexible manner still remains a challenge [2]. Among the several advantages of the edge computing approach, the most noticeable ones are the following [3]:

1. an increase in the system responsiveness and latency reduction due to network resources being moved closer to the users, thus resulting in smaller communication delays, as well as in reduced network congestion (because traffic is kept at the edge servers rather than processed at a single central cloud node);

2. the ability to supply customized services based on the live user's experience, commonly referred to as *context awareness*. The edge servers can exploit the physical proximity to capture useful real-time information, thus enabling a plethora of applications at different levels of the stack, from the application level (i.e., virtual reality [4, 5], content caching [6, 7] and computation

offloading [8, 9]), to the network level (i.e., load balancing [10, 11], and virtual network function provisioning [12, 13]), down to the physical level (i.e., spectrum hole detection [14, 15] and radio fingerprinting [16, 17]).

Nevertheless, the edge infrastructure presents multiple constraints on several levels. For instance, edge servers are not nearly as powerful as cloud servers and can represent a computational bottleneck if the network resources are not properly managed. For example, in mobile networks extended by Unmanned Aerial Vehicles (UAVs), overloading one specific drone may lead to fast battery depletion, thus causing service outages. This issue is also critical in IoT networks, where the energy, computational, and communication resources are extremely constrained, and proper management is required. In other words, the problem of resource management and allocation at the edge is of utmost importance, as it can drastically improve the network throughput, efficiency, and lifetime.

Moreover, the concept of network slicing has emerged as a promising paradigm to revolutionize the way modern communication networks operate. Network slicing offers the ability to partition a physical network into multiple logically isolated segments, each tailored to cater to specific services, applications, or user groups. This approach holds the potential to deliver unprecedented levels of customization, flexibility, and efficiency, thereby paving the way for the realization of diverse and specialized use cases within a single physical infrastructure.

Central to the successful implementation of network slicing is the seamless orchestration of resources to meet the stringent Service Level Agreements (SLAs) demanded by the distinct slices. These SLAs encompass a myriad of performance metrics, with latency being a paramount concern in today's real-time applications. Achieving ultra-low latency is a fundamental requirement for various mission-critical use cases such as industrial automation, augmented reality, and vehicular communication. Reducing network latency while concurrently ensuring the Quality of Service (QoS) commitments for multiple slices remains an intricate challenge that necessitates innovative and adaptive techniques.

One of the fundamental prerequisites of 6G networks is the realization of a fully automated network and service management framework to rapidly deliver services while ensuring the sustainability of heterogeneous service offerings. The ETSI Zero-touch network and Service Managenent (ZSM) group was established to address this requirement and accelerate the definition of the necessary architecture and solutions. Self-configuration, self-monitoring, and self-optimization, without human intervention, form the foundation of all 6G networks.

6G is therefore expected to emerge with support from AI, ultra-reliability, and

zero-touch network management [18]. Edge Intelligence, powered by AI techniques, is already recognized as a crucial element in 5G networks and will likely serve as a key enabler for future 6G networks to support their performance, new functions, and services. This evolving landscape presents new opportunities for edge computing and Edge Intelligence in various industry domains [19]. The evolution towards 6G networks entails distributing AI capabilities and moving intelligence from the central cloud to edge computing resources. This transition represents the ultimate shift towards the paradigm of the Intelligent Intelligent Internet of Intelligent Things (IIoT), enabled by Edge AI [20, 21].

Among various machine learning techniques, Deep Reinforcement Learning (DRL) stands out as the most widely employed approach for tackling intricate optimization challenges within the context of networking in 5G/6G networks. This is due to several key reasons:

- *Adaptability to Dynamic Environments*: 5G and 6G networks operate in highly dynamic and evolving environments where network conditions, user demands, and system configurations change rapidly. DRL's ability to adapt to these changes through learning and decision-making makes it well-suited for such scenarios.

- *Model-free approach*: DRL is a model-free technique, meaning it doesn't rely on predefined models of the network or its components. This is advantageous in situations where it's challenging to create accurate mathematical models due to the complexity and variability of the network.

- *Handling Large State Spaces*: 5G and 6G networks involve vast state spaces, considering the numerous network parameters, devices, and connections. DRL algorithms can handle large state spaces efficiently, enabling them to make informed decisions in complex network settings.

However, in many scenarios, deploying 5G and beyond networks poses challenges in areas where networking and computing infrastructures are scarce or absent. Unmanned Aerial Vehicles (UAVs) have emerged as a viable solution, given their flexibility, on-demand provisioning, and ability to access remote and challenging locations. In fact, fleets of UAVs can be grouped into Flying ad-hoc Networks to further enhance the coverage, capacity, and reliability of wireless cellular networks, as well as provide edge and fog computing capabilities. Other key advantages of employing a FANET lie in its ease of on-demand deployment and its flexibility to adapt to varying runtime needs. Moreover, the number of

Unmanned Aerial Vehicles in the FANET can be adjusted to accommodate the extension and device density of the supported area.

However, ensuring efficient network and service management automation for Flying ad-hoc Networks remains a crucial requirement for 6G networks. Although one or more UAVs may have connections to the structured Internet, these connections are often insufficient to support the timely transmission of large-sized jobs to data centers. Hence, the FANET assumes the role of the network edge, with each UAV functioning as a 6G network access point that offers edge computing facilities. To address this, the research is showing an increasing interest in zero-touch management frameworks for edge networks, aiming at providing computing and networking facilities to remote geographic areas hosting delay-sensitive applications, considering challenges such as limited flight duration, job scheduling, and network optimization.

One of the most challenging application scenarios for the above technologies is constituted by the Intelligent Transport System (ITS), enabled by the use of vehicular networks for various applications, such as traffic management, safety and entertainment [22–24]. These applications often have low latency and high-reliability targets, whose achievement may be challenging in vehicular networks due to their highly dynamic and resource-constrained nature. In fact, the limited processing and storage capabilities of the On-Board Units (OBUs), i.e. the processing equipment installed on-board of the vehicles, are often insufficient to guarantee such requirements.

One approach to address these challenges is to use job offloading, which involves transferring some or all of the required computation to more powerful and/or better-connected devices, typically the so-called Road Side Units (RSUs) installed along the roadway. However, using job offloading in vehicular networks introduces additional challenges, including balancing latency and energy consumption.

When deployed in remote roads and rural areas, RSUs do not have access to a fixed Internet connection and/or the power grid. In such cases, the RSUs are stand-alone, battery-powered devices that can only count on the local computing units for processing and green energy harvesting as their power source.

In this perspective, for load balancing in the network, in order to reduce peaks of latency and energy consumption, each RSU should also be able to further offload the received jobs to nearby RSUs [25].

With this in mind, a new, challenging scenario emerges, where each RSU has to autonomously find a proper trade-off between the energy consumption and the processing delay requested by vehicular services. On the one hand, this goal can

be achieved by adequately tuning the amount of processing power employed and, on the other, by choosing the optimal amount of jobs to offload to the nearby RSUs.

Another hot area of research terrestrial network-wise lies in O-RAN, an innovative network architecture that promotes programmability, virtualization, and open interfaces to enhance the flexibility and agility of cellular networks [26]. The O-RANs architecture leverages a disaggregated approach for the Radio Access Network (RAN), splitting the base stations into functional units. Moreover, O-RAN introduces machine learning (ML)-based network control and automation algorithms through the so-called xApps, running on RAN Intelligent Controllers. However, in spite of the new opportunities brought about by the Open RAN, advances in ML-based network automation have been slow, mainly because of the unavailability of large-scale datasets and experimental testing infrastructure. This slows the development and widespread adoption of DRL agents on real networks, delaying progress in intelligent and autonomous RAN control [27].

Despite its remarkable potential, the application of DRL techniques to address the intricate problem of network slicing optimization in O-RAN remains conspicuously underexplored.

In this context, this thesis aims to address critical aspects of network optimization and management in three different scenarios: (1) the design and testing of DRL-based zero-touch management framework in Flying ad-hoc Networks, (2) the development of a distributed edge-computing framework for latency- and energy-aware job offloading in green vehicular networks, and (3) to harness the power of DRL in devising dynamic and data-driven resource allocation strategies that satisfy network latency requirements for different network slices in O-RANs. The challenges in deploying DRLs-based control solutions at scale are numerous. Collecting datasets representative of real-world network behavior, testing the robustness of ML-based control at scale, designing efficient ML agents with unreliable input and constrained output, enabling generalization capabilities, and selecting meaningful features are all critical considerations to ensure the successful integration and deployment of DRLs in FANETs, VANETs, and O-RANs.

To achieve these objectives, this thesis proposes innovative solutions that address the challenges faced in practical network deployments. By collecting datasets at scale, accurately representing the intrinsic randomness and behavior of real-world networks, we aim to develop robust ML-based control solutions. Rigorous testing of these solutions at scale is vital to ensure their stability and prevent suboptimal performance or outages. Additionally, the ability of ML agents to

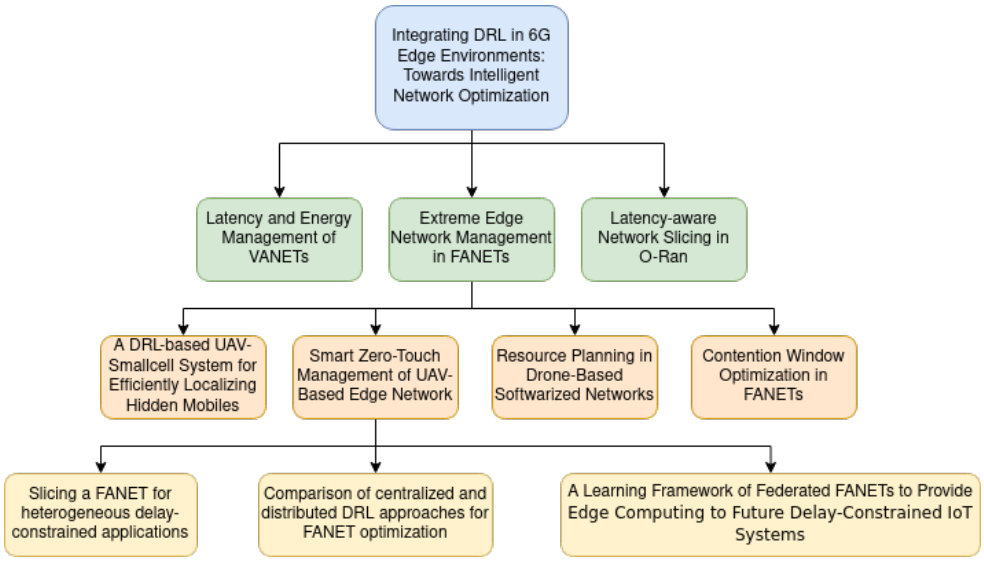


Figure 1.1: High-level Diagram of the Research

generalize and adapt to unseen deployment configurations will be a key focus.

This thesis is structured into three parts. Part I focuses on the design and implementation of a zero-touch management framework for FANETs, enabling efficient network and service management in remote areas.

Part II delves into the design of a distributed framework based on multi-player multi-armed bandit (MP-MAB) algorithms for latency- and energy-aware task offloading in vehicular networks, with the main goal of supporting procedures of job offloading in green vehicular networks in order to achieve a target trade-off between energy consumption and the job processing latency trade-off.

Part III explores the application of DRLs in closed-loop network control scenarios within the O-RANs architecture, aiming to improve network performance and adaptability. Throughout the thesis, extensive simulations and evaluations will be conducted to assess the proposed frameworks' effectiveness and compare their performance against existing literature. The contributions of this thesis lie in the development of novel methodologies and algorithms that enable efficient and adaptive network management in FANETs and O-RANs through the integration of DRLs techniques.

In the course of this extensive research journey, our initial foray into the utilization of Deep Reinforcement Learning (DRL) techniques revolved around addressing an optimization challenge within the context of Flying Ad-Hoc Networks (FANETs), specifically focusing on the optimization of horizontal offload among drones within a FANET. We embarked upon this endeavor by initially employing the well-established DQN technique. This initial step provided us with valuable

insights into engaging with DRL scenarios and underscored the existence of algorithms far more potent than the conventional DQN [28], including its variants [29].

Building upon this foundational knowledge, we ventured into more intricate DRL approaches, such as actor-critic methods (e.g., A2C [30] and PPO [31]), while also undergoing a paradigm shift from centralized orchestration to a distributed framework through the adoption of multi-agent DRL approaches, including MAPPO [32] and A3C [33]. This made us realize that in the current state of the art, purely centralized DRL approaches for orchestrating 6G networks do not scale well when dealing with complex problems. Zero-touch automated orchestration will most likely be achieved by the cooperation of different intelligent agents interacting with different parts of the networks, each with different scopes and objectives. We have also found out that applying DRL in the context of networking requires careful thinking to make the right decisions throughout the design of the reinforcement learning problem, which includes the design of the agents, the formulation of the Markov Decision Process, the choice of the data (observations) required to make smart decisions (actions), and the design of the reward function. Subtle changes in each of these elements may result in terrific changes in the performance and stability of the training process. Piqued by the prospect of integrating Federated Learning and DRL, we proceeded to explore the realm of Federated Deep Reinforcement Learning (FDRL) [34] within an environment where multiple FANETs pooled their collective knowledge to enhance performance in uncharted scenarios. This trajectory led us to revisit the roots of Reinforcement Learning, notably Multi-Armed Bandits (MAB), stimulated in part by publications such as [35], in which authors demonstrated that MAB agents could learn and achieve near-optimal performance even in non-stationary and non-i.i.d. settings. In particular, we were enticed to leverage the expertise garnered from DRL problems to approach a system made up of multiple RSUs with a Multi-Player Multi-Armed Bandit (MP-MAB) [36] framework, resulting in insightful findings regarding the speed of convergence of MAB algorithms in multi-agent non-stationary environments. Finally, adhering to the O-RAN specifications, we embarked on the exploration of offline DRL, as mandated by O-RAN for pre-training agents, which could subsequently be fine-tuned for real-world deployments. This holistic journey allowed us to comprehensively survey various DRL approaches and their integration in 5G and beyond systems, delving into the primary techniques (DQN, A2C, PPO) within both single-agent and multi-agent contexts while also experimenting with federated methodologies in both

online and offline training domains.

Fig. 1.1 shows a high-level diagram of the research discussed in this thesis. The remainder of this thesis is organized as follows: Section II presents the research works specific to FANETs. Section III delves into the design of an energy- and latency-aware framework for VANETs. Section IV presents the latest work on DRL-enabled Network Slicing SLA satisfactions in O-RAN. Finally, Section V concludes the thesis, summarizing the key findings, discussing their implications, and outlining future research directions in the field of DRLs-based network optimization in FANETs, VANETs and O-RANs.

Chapter 2

Extreme Edge Network Management in FANETs

In numerous instances, the envisioned 6G application scenarios pertain to areas lacking nearby structured networks [37], resulting in the absence of networking and computing infrastructures in close proximity to data-generating objects. These scenarios encompass both extreme and remote zones, which are exceedingly challenging to access via conventional structured networks. Additionally, they include areas where structured networks are vulnerable to disruptions caused by natural disasters, terrorist attacks, and wartime actions.

In such scenarios, Flying Ad-hoc Networks (FANETs) present an effective solution to enhance the coverage, capacity, reliability, and energy efficiency of wireless cellular networks [38–40]. The recent reduction in UAV costs has made this approach feasible, offering flexibility through on-demand service provisioning and the ability to access extreme and remote areas that are otherwise challenging to reach through conventional means.

The dynamic and decentralized nature of FANETs presents a unique set of challenges and opportunities in the realm of network management and orchestration. As these aerial networks continue to find application in surveillance, disaster response, and environmental monitoring, their efficient operation necessitates intelligent and autonomous resource allocation strategies. This chapter explores the utilization of DRL to enhance the management and orchestration of FANETs. We delve into the intricacies of problem formulation, discuss the potential AI/ML techniques, and highlight the key considerations for leveraging DRL in optimizing resource allocation, offloading, and positioning in FANETs.

However, ensuring the *continuous availability* and activity of a FANET remains a persistent challenge due to the limited autonomy of its UAVs [41]. Besides

its engines, each UAV within the FANET is equipped with two key elements: a battery and a Computing Element (CE). The CE is responsible for executing one or more Virtual Functions (VFs), delivering 5G services to ground users. Notably, the power consumption of the CE is comparable to that of the UAV's engines, consisting of two primary contributions: 1) a constant power to maintain CE activity and 2) a variable power linked to the amount of data processed by each VF, expressed in terms of CPU usage [42–44].

Hence, a pivotal challenge to address is the minimization of power consumption. Increased CE *resource utilization* leads to faster battery depletion, thereby reducing the overall FANET service availability. Particularly critical is the scenario where a UAV's battery charge falls below a certain threshold, necessitating temporary withdrawal from the FANET to access the nearest charging station. During this period, the Virtual Functions running on the unavailable UAV must be redistributed among the remaining UAVs, causing a surge in their energy consumption and reducing their flight duration. If the number of operational UAVs is insufficient, the FANET's ability to deliver services to ground devices could be compromised. Thus, effective power management and resource allocation strategies are vital to ensure prolonged and reliable FANET operation in 6G networks.

The thesis rigorously delves into the multifaceted problem of power management and resource allocation within the FANET framework, employing advanced methodologies and innovative techniques to optimize the CE's power consumption, enhance overall network availability, and address the challenges posed by UAV battery limitations, thereby ensuring the sustainable and efficient operation of the FANET in the dynamic landscape of 6G networks.

Furthermore, several studies have dedicated their efforts to devising techniques that enhance UAV performance while simultaneously minimizing energy consumption. As UAVs often operate in unknown or partially observable environments, *flight path planning* emerges as a critical challenge, significantly influencing the vehicle's level of autonomy.

Given the absence of an exact mathematical model, researchers have turned to integrating Deep Learning (DL) and Reinforcement Learning (RL) methodologies. By combining these approaches, UAVs can autonomously learn their optimal flight paths, enabling them to navigate through dynamic environments without the risk of collisions [45–48]. This thesis delves into the exploration and evaluation of DL and RL integration in the context of 6G FANET networks, investigating how these technologies can significantly bolster UAV autonomy and performance, thereby paving the way for safer and more efficient operations in complex and uncertain

environments.

Moreover, *load balancing* techniques, such as *horizontal offload*, can be effectively employed among the UAVs within a FANET. Horizontal offload involves transferring jobs from one UAV to another, enabling overloaded UAVs to offload tasks to less-loaded counterparts. This load balancing mechanism ensures that the computation delay of the FANET for each received job remains nearly independent of the current activity state of the area covered by the receiving UAV [42, 49, 50]. Through an in-depth exploration of load balancing strategies, this thesis investigates the impact of horizontal offload on enhancing the overall performance and efficiency of 6G networks within FANETs, aiming to achieve seamless task distribution and minimize computation delays across the network.

All of this requires a new architecture framework designed for closed-loop automation and optimized with data-driven machine learning and artificial intelligence algorithms. The target of this research is to design a zero-touch management framework for an edge network provided through a FANET constituted by a set of UAVs.

2.1 Related Work

In the past, applications of single UAVs or fleets of UAVs have been widely used in order to enhance coverage, capacity, reliability, and energy efficiency of wireless cellular networks with terrestrial equipment [51, 52], and to provide fog-computing facilities to IoT systems in remote areas [53–59].

2.1.1 FANET Edge Computing Frameworks

A multitude of academic research papers have been dedicated to the refinement of FANET Edge computing frameworks. These research endeavors are primarily focused on the optimization of resource allocation efficiency and the establishment of continuous functionality, particularly within dynamic and unpredictable environments. A complex structured fleet of UAVs to provide edge computing services by means of a FANET was studied in [49, 60, 61], but with a management infrastructure that is too specific to the scenario in which it is used. In [62], the authors give an overview of UAV-aided wireless communications, presenting a basic networking architecture (with insights about channel characteristics and system design) that allows the use of UAVs as wireless forwarders to improve connectivity and coverage of GDs. In [63], coverage and rate performance using UAVs as flying base stations are analyzed, providing downlink connectivity to

users placed in a zone in which other devices communicate with each other using device-to-device (D2D) communications. The same authors, in [64], analyze the possibility of using a framework for an optimized deployment and mobility of UAVs to receive data from ground IoT devices, finding the optimal UAVs' location, the association of devices to UAVs and the uplink transmission power that each device has to use to communicate with the UAV. In [57], the authors propose a scenario in which the combined use of UAVs and fog computing techniques allows to support IoT applications. Using one or more UAVFogs, a fog-computing platform is created, providing storage, networking, and processing capabilities. Each UAVFog also acts as a gateway between the UAV infrastructure and the remote cloud. In [65], the authors propose a three-tier supply chain network model to provide 5G network slices on demand to ground users using a FANET. In particular, ground users send service requests to a fleet of UAV controllers, which redistribute these requests in an optimal way to a FANET in which the services are active. To this purpose, the authors implement a constrained optimization problem for which they derive the associated variational inequality formulation.

Computation offloading consists of migrating tasks generated by the applications to more powerful computing devices in order to satisfy some performance requirements or simply speed up the applications. Computation offloading has recently become of particular relevance in the context of mobile computing, where devices may not have enough hardware capabilities to sustain the ever-increasing application requirements [66]. In coarse-grained offloading or full offloading, the whole computation is offloaded and processed by a cloud or MEC server. On the contrary, in fine-grained or partial offloading, only a part of the computation is processed locally, while the rest is offloaded to a cloud or some MEC servers. When and where to offload some jobs is a very crucial decision in order to meet the stringent requirements of some applications. In order to perform this intelligent decision, recently, some AI-based algorithms have been introduced at the network edge [67, 68]. When applied to scenarios of Mobile Ad-Hoc Networks (MANET) or Vehicular Ad-Hoc Networks (VANET), these algorithms usually perform an unbalanced offloading to devices with higher computational resources. In the context of FANETs, this approach is not applicable because entities have all the same features. This kind of horizontal offloading was just applied in several previous works. In [69], for each task, based on some performance parameters, the source UAV has to make the choice of whether to process the task locally or offload it to a nearby UAV with available resources. In [70], horizontal offloading is applied in a MEC-assisted vehicular network.

Indeed, for each possible activity state of the covered zone, the intra-FANET layer of the framework decides the probability that a job will be processed locally or offloaded to the UAV with the least loaded queue. This way, the FANET is able to adapt to the variability of the underlying environment in the short term, also reducing the periods of under-use of some UAVs.

2.1.2 Flight Path Planning

Several studies have focused on devising techniques to improve drone performance while minimizing energy consumption. UAVs are often used for operations in unknown or partially observable environments. For this reason, flight path planning is an essential issue in the use of UAVs as it is directly related to the level of autonomy of the vehicle.

DL and RL are combined and used to allow UAVs to learn their paths autonomously, allowing them to traverse changing environments without the risk of collision [45–48].

In [71], an RL algorithm is introduced that enables UAVs to have direct and continuous interaction with their surroundings. In particular, a combination of Deep Reinforcement Learning (DRL) and a Long Short-Term Memory (LSTM) network is proposed to increase the speed of the used learning algorithm. In addition, the authors in [72] propose the RL algorithm to circumvent obstacles with a reward function and a penalty action to have a smoother trajectory. In [73], several RL algorithms are used to improve UAV navigation. Moreover, UAVs can provide wireless connectivity without network infrastructure or complement conventional base stations (BSs), whose coverage may suffer from severe blockage due to tall buildings or damages caused by natural disasters. Owing to the mobility of UAVs, recent years have seen significant research progress on integrating UAVs with MEC [74].

The emergence of 6G technology will bring UAVs to play an important role in wireless networks and future smart cities with features like massive connectivity, ubiquitous coverage, embedded artificial intelligence, efficient energy usage, and adaptive network security. UAVs can act as vertical components in these networks, enhancing coverage, reliability, and energy efficiency. They are capable of communicating with ground stations and satellites, creating a space-air-ground network that fully integrates heterogeneous 6G networks [75].

However, for UAVs to fully leverage their potential in 6G networks, it is necessary to extend their communication capabilities further. Traditional UAV networks are evolving into enhanced networks, where UAVs connect with each other

in an ad-hoc manner and can perform computation, communication, and control functions. The integration of technologies such as quantum processing, terahertz communication [76], fog/edge computing [77], and wireless optical communication [78] can contribute to improving the performance of UAV networks, including service quality, energy efficiency, security, and fault management.

To integrate 6G networks with UAVs, it is essential to ensure security and privacy. UAVs are susceptible to various security vulnerabilities due to their limited onboard processing and energy capacity. Therefore, the wireless research community must pay particular attention to these aspects in the design of UAV networks [79].

In summary, the utilization of UAVs in combination with emerging technologies of 6G wireless networks offers significant opportunities to enhance the performance, efficiency, and security of civil protection interventions and communication services in emergency and natural disaster scenarios [80].

2.1.3 Contention Window in FANET

As demonstrated in [81], the contention window (CW) is one of the parameters that plays a key role in the MAC layer behavior, and as such has a significant impact on the efficiency of Wi-Fi networks. In the basic channel access method each station waits a certain number of time slots before accessing the channel. This number is chosen at random between 0 and the CW value. To reduce the probability of multiple stations selecting the same random number, the CW, starting from a minimum value, CW_{min}, is doubled after each collision, up until a maximum value, CW_{max}. In IEEE 802.11, the CW_{min} is set to 16, while the CW_{max} is set to 1024 [82]. This approach, while having low complexity in terms of computation, can lead to poor network performance, especially in dense networks. The optimization of CW has, therefore, a major impact on network performance and as such is currently been the subject of multiple research activities.

Analytical approaches can provide optimal CW values. However, these approaches require a lot of assumptions, rarely encountered in real networks, and quasi-static settings. For this reason, these approaches perform poorly when deployed in real networks. For this reason, in the last few years, there has been an increasing research effort to enhance network performance using machine learning (ML) [83]. In the CW optimization problem, the most obvious ML technique to use is Reinforcement Learning (RL). In RL, an agent takes actions in an environment while trying to maximize the long-term reward. RL, and especially Deep Reinforcement Learning (DRL), has proved numerous times to be able to solve

complex problems in highly dynamic environments [84].

The CW optimization problem using Deep Reinforcement Learning has already been investigated in [81], in which a centralized contention window optimization approach (CCOD) predicts the best CW values to improve throughput in 802.11 wireless networks. CCOD uses the legacy binary backoff algorithm combined with DRL. Although it succeeded in decreasing the collision rate, the amount of iterations required to converge is very high (in the order of $1 \cdot 10^4$ iterations).

In [85], authors investigated the CW optimization problem in a multi-agent scenario and designed SETL-DQN, in which agents cooperate to avoid network performance degradation. However, despite achieving slightly better results than CCOD in saturated network conditions, its solution requires an extremely higher number of iterations ($5 \cdot 10^4$ episodes) before giving acceptable results. This means that its solution must be trained offline and deployed only after being fully trained. Moreover, as soon as the underlying environment changes (for example, if the number of stations increases), their approach has to be retrained. Moreover, realistic network conditions have not been evaluated.

2.2 Smart Zero-Touch Management of UAV-Based Edge Network

The next generation of wireless communications networks, namely 6G, will be aimed at realizing a fully connected world and at providing ubiquitous connectivity to people and objects, even in remote areas that are very far from the structured Internet core network. These goals include the definition and the design of intelligent communications environments mainly characterized by pervasive artificial intelligence and large-scale automation. The target of this research is the design of a management framework for edge networks realized with FANETs consisting of a set of UAVs to provide a remote geographic area with computing and networking facilities for delay-sensitive applications.

2.2.1 System Description

As sketched in Fig. 2.1, we focus on geographic areas that are badly connected or not directly connected with a core network infrastructure. To this purpose, when an area requires edge-computing services, a FANET is used to provide it with this service. The area covered by a FANET is subdivided into zones, each one served by one of its UAVs. Each UAV is equipped with a CE to provide MEC

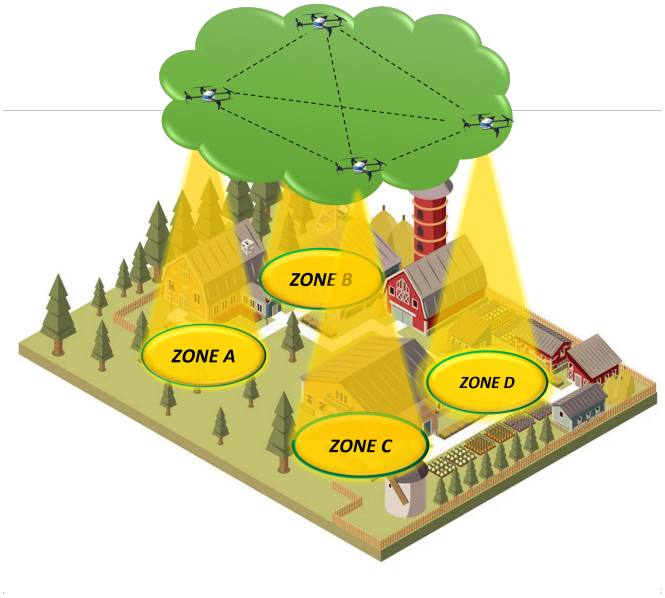


Figure 2.1: Reference System.

facilities to GDs belonging to the ground area it is serving. Each GD inside a zone can generate data that have to be processed in order to perform specific tasks. In this context, a job can be regarded as either a set of data, generated by a single GD, to be processed together, or the collection of several sets of data generated by many GDs, which are collected and aggregated by a sink. Each GD, taking into account its amount of computation facilities, the energy that it should spend to process the job locally, and the current state of its battery charge, decides whether to offload the execution of each job to the FANET or not. In the sequel, offload from GDs to UAVs of the FANET is referred to as *vertical offload*.

The definition of policies for decisions regarding whether offloading jobs to the FANET or not are considered out of the scope of this research. For our purposes, we limit ourselves to characterize vertical offload as a job arrival process to each UAV of the FANET from the zone it is serving.

Undoubtedly, FANET performance hinges on two crucial factors: firstly, the total count of Ground Devices (GDs) within the area and the volume of offload traffic they generate, and secondly, the number of UAVs deployed in the FANET. While the former remains beyond the control of the FANET designer, the latter allows the designer to influence the required number of UAVs to attain the desired performance levels. Additionally, the computation power of the CE and the battery capacity aboard each UAV significantly influence the mission duration of the FANET, accentuating the importance of thoughtful design in both aspects.

The job-offloading rate to each UAV depends on the current state of the zone:

the higher the activity of a zone, the higher the job arrival rate. Time variation of the job-offloading rate on each zone can occur because the number of GDs populating that zone changes over time, for example, due to the regular traveling related to working hours and/or holidays. In addition, even if the number of GDs remains constant over time, the traffic generated by each GD can change, for example, for the occurrence of a specific event or according to normal daylight and night activities, but also to specific solicitations from social networks or event broadcasting by live streaming. This dynamic behavior of each zone suggests we introduce the feature of *horizontal offload*. It consists of intra-FANET job offloading from one UAV to another one, giving overloaded UAVs the possibility to offload jobs to other, less-loaded, UAVs, with a resulting load balancing.

For each job, we call *Dwell UAV* the UAV that has received it from the ground, and *Processing UAV* the UAV where the job is processed. Each job that is locally processed in its Dwell UAV suffers the delay for processing there. On the other hand, an offloaded job suffers an overall delay that is the sum of the time spent for transmission from the Dwell UAV to the Processing UAV and the time spent in the latter for processing.

Job offloading has the following advantages:

- Although horizontal offload introduces an additional delay due to the transmission from a UAV to another one, the overall delay can be reduced if job processing is moved to less loaded UAVs;
- The delay jitter (i.e., the standard deviation of the delay) can be reduced thanks to load balancing;
- The overall wasted time for the inactivity of the FANET CEs can be reduced if offloaded jobs are moved to UAVs with idle CEs.

A scheduler running in each UAV, according to the above considerations, decides whether an incoming job has to be processed locally or offloaded to another UAV. This is a crucial task that has to be optimized by taking into account the state of the whole FANET and, at the same time, forecasting the behavior of the job generation process of all the zones in the short- and medium-time horizon.

Jobs to be processed locally are enqueued in the *Processing Queue* Q_P waiting to be served by the local CE. On the other hand, jobs to be offloaded to other UAVs are enqueued in the *Offloading Queue* Q_O that is served by the wireless transmission link towards the other UAVs. Without loss of generality, we will assume that all the UAVs inside a FANET have identical CEs.

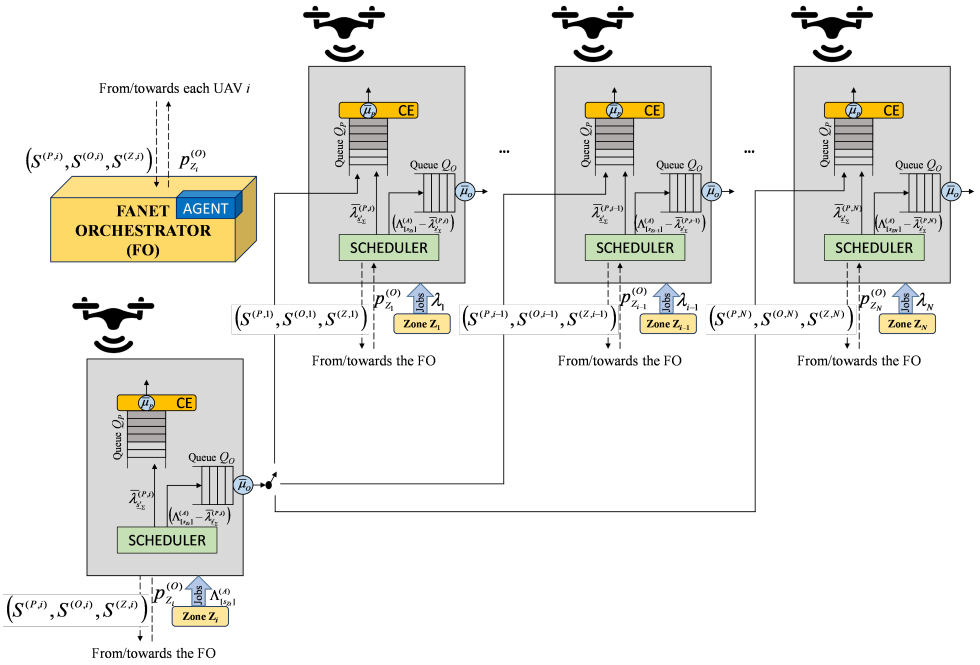


Figure 2.2: Smart Zero-Touch FANET Framework Architecture

The architecture proposed in this research, represented in Fig. 2.2, allows using any scheduling techniques for jobs waiting for processing in the Q_P queue, for example, to manage jobs with different computation times and different priorities. However, for the sake of simplicity, we assume a First-In-First-Out (FIFO) policy, leaving the introduction of other scheduling techniques as future work. Likewise, we assume that jobs enqueued in Q_O for horizontal offload towards other UAVs are managed by a FIFO policy as well.

These decisions are not trivial because horizontal offload causes an additional delay due to the time spent inside Q_O waiting for transmission and the propagation time from the Dwell UAV to the Processing UAV. In the sequel, without loss of generality, we assume that all the channels between each pair of UAVs are characterized by a negligible propagation delay and the same transmission rate. This assumption is supported by the small difference in relative distance between UAVs and by the stationary conditions of the related links (equal channel bandwidth and noise conditions). Therefore, the overall delay suffered by each offloaded job is the sum of the time spent in Q_O of its Dwell UAV (*Offloading delay*) and the time spent in Q_P of its Processing UAV (*Processing delay*). Instead, jobs that are not offloaded only suffer the *Processing delay*.

Decisions regarding horizontal offload are taken in each UAV by the local Scheduler according to a policy received by the orchestrator of its FANET, which uses information on all the UAVs of the FANET regarding the state of their Processing

Queues.

Horizontal offload is managed via a *probabilistic approach*: each Scheduler has an array of probabilities, one for each possible activity state of the served zone. Therefore, according to this state, the UAV Scheduler offloads jobs to another UAV with the probability associated with that state. This set is periodically broadcasted by the Fanet Orchestrator (FO) to all the UAVs of the FANET to be applied according to the current activity state of the zone they are covering. To this purpose, a *Zone Activity Monitor* is in charge of monitoring the state of the served zone by observing the job arrival rate process coming from the ground and communicating this information to the Scheduler that, as said so far, uses this information to perform offload according to the current state of the zone activity.

Let us note that the values of the above probabilities play a crucial role in the performance of the system. Indeed, high probability values cause an excessive exchange of jobs among UAVs, with a consequent increase in the FANET processing time due to the introduction of the time spent by the offloaded jobs in the Offloading Queues. On the other hand, low probability values prevent the system to leverage horizontal offload for load balancing, hence decreasing both mean delay and delay jitter.

2.2.2 FANET Orchestrator

The FANET Orchestrator (FO) of each FANET resides in the *Master UAV* of the FANET and has the objective of deciding the horizontal offloading probabilities to be broadcasted to all the Schedulers in the FANET.

In order to solve the optimization problem of finding the best offloading probabilities, a DRL agent is deployed inside the FO. As the number of UAVs inside the FANET increases, traditional table-based RL becomes infeasible in solving the problem due to its huge state space and the multidimensional discrete action space. Therefore, DRL agents become mandatory to make the FO learn in unknown environments and overcome the prohibitive computational requirements.

In this framework, the agent inside the FO interacts with its environment at each decision epoch by observing the state of the environment and executing an action that alters that state. This action is then evaluated against a performance objective, and a reward is received by the agent. The reward is designed so that maximizing the reward would achieve the desired goal. At each decision epoch n, the agent selects an action a_n in the action space $\mathscr{A}$. The state of the environment perceivable by the agent at the decision epoch n is $\underline{s}_n \in \mathscr{S}^{(\Sigma)}$, where $\mathscr{S}^{(\Sigma)}$ denotes the observable state space of the agent. After executing an action, the agent will

receive a reward r_n. The goal of the agent is to maximize the total reward G_n it receives with a discount factor γ. It is calculated as follows:

$$G_n = \sum_{k=0}^{+\infty} \gamma^k \cdot r_{n+k+1}. \tag{2.1}$$

The mapping between the state and the action is called the policy π of the agent. Formally, the probability of the agent to choose an action a_n when it is in the state $\underline{s}_n$ is called the policy of the agent over all the action and state pairs:

$$\pi(a|\underline{s}_\Sigma) = \Pr\{a_n = a|\underline{s}_n = \underline{s}_\Sigma\} \quad \forall a \in \mathscr{A},\ \forall \underline{s}_\Sigma \in \mathscr{S}^{(\Sigma)}. \tag{2.2}$$

The *state-value function* $v_\pi(\underline{s}_\Sigma)$ is defined as the expectation of future rewards given the current state, that is:

$$v_\pi(\underline{s}_\Sigma) = E\left\{G_n|\underline{s}_n = \underline{s}_\Sigma\right\} \quad \forall \underline{s}_\Sigma \in \mathscr{S}^{(\Sigma)}. \tag{2.3}$$

The *action-value function* $q_\pi(\underline{s}_\Sigma, a)$ is defined as the expectation of the future rewards given the current state $\underline{s}_\Sigma$ if the action a is executed

$$q_\pi(\underline{s}_\Sigma, a) = E\left\{G_n|\underline{s}_n = \underline{s}_\Sigma, a_n = a\right\} \quad \forall a \in \mathscr{A},\ \forall \underline{s}_n \in \mathscr{S}^{(\Sigma)}. \tag{2.4}$$

The relationship between the action-value and state-value functions is derived from the Bellman optimality equation as:

$$q_\pi(\underline{s}_\Sigma, a) = r_n + \gamma v_\pi(\underline{s}_\Sigma) \quad \forall a \in \mathscr{A},\ \forall \underline{s}_n \in \mathscr{S}^{(\Sigma)}, \tag{2.5}$$

where r_n is the reward obtained when executing the action a in the state $\underline{s}_n$.

In our framework, in order to provide more transition samples for the whole training, an experience buffer D has been introduced inside the FO of the FANET.

Any DRL algorithm can be used in this framework to support the FO operations. One step of the local training of a generic DRL algorithm is summarized in Algorithm 1. First, the experience buffer D is instantiated (lines 1-3). For each epoch, we distinguish between two different phases, which are the start (lines 6-7) and the end (lines 9-10) of the decision epoch. At the start of the decision epoch, the agent observes the state and then executes an action based on the current policy. Line 8 considers the environment evolving from one state to another one until the next decision epoch is triggered. At the end of the epoch, as expressed in lines 9-10, the agent receives the reward from the environment and observes the

next state. The experience is then stored in the experience buffer D (line 11), and a gradient descent step is executed on a random minibatch of experience (lines 12-13).

Algorithm 1 Horizontal Offloading Training via DRL

1: Initialize Experience Buffer D to capacity M
2: Initialize $Q(s, a)$ with random weights θ
3: Initialize $\hat{Q}(s, a)$ with weights $\theta^- = \theta$
4: **for** each episode **do**
5: **for** each decision epoch n **do**
6: Observe state s_n as in (2.7)
7: Execute random action with probability ϵ, otherwise execute $a_n = max\{Q(s_n, a; \theta)\}$
8: Let the environment evolve
9: Calculate reward r_n as in (2.11)
10: Observe state s_{n+1}
11: Store state transition (s_n, a_n, r_n, s_{n+1}) in Z
12: Get a random minibatch of state transition of size B from Z
13: Perform a gradient descent step on the loss with respect to the evaluation network parameters θ
14: Soft update target network $\theta^- = \theta \cdot \tau + \theta^- \cdot (1 - \tau)$
15: **end for**
16: **end for**

The Markov Decision Process (MDP), including the state space $\mathscr{S}^{(\Sigma)}$, the action space $\mathscr{A}$, the reward function r_n, the state transition model, as well as details on the environment, will be described in more detail in the next section.

2.2.3 Horizontal Offload Optimization

In this section, we introduce the optimization model used by the FO to decide the horizontal offloading probabilities to be broadcasted to all the Schedulers in the FANET. To this purpose, we first present the MDP used to support the FO operations, and then we introduce the derivation of the transition rate matrix of the MDP.

As said so far, each FANET implements horizontal offload for load balancing in order to minimize average delay and delay jitter of job processing. More specifically, its FO is in charge of deciding the set of offloading probabilities that have to be broadcasted to the UAV Schedulers to perform offload according to the activity state of their zone. To this purpose, the FO applies an RL approach to be able to follow the dynamics of the system it is orchestrating. The FO behaves

as an RL agent that iteratively interacts with the environment at each decision epoch.

Let N be the number of UAVs of the FANET, and Ψ the number of possible activity states of each zone covered by the FANET.

The environment of the agent of this FANET is represented by four types of events:

1. *job arrival*: when a job arrives at one of the N UAVs of the FANET, it can be either enqueued in the Processing Queue to be processed locally or in the Offloading Queue, to be horizontally offloaded to the least-loaded UAV;

2. *job processing*: a job leaves the CE of the UAV where it has been processed;

3. *job transmission for horizontal offload*: a job leaves the Offloading Queue of the Dwell UAV and arrives to the Processing UAV, entering its Processing Queue;

4. *activity-state change*: one of the zones changes its activity state.

Therefore, the state of the environment, as seen by the agent in the FO of a given FANET, is represented by the state of the UAV Processing Queues, the state of the UAV Offloading Queues, and the activity state of the zones served by the UAVs of the considered FANET:

$$\begin{aligned} \underline{S}^{(P)}(t) &= \left(S^{(P,1)}(t), S^{(P,2)}(t), \ldots, S^{(P,N)}(t)\right); \\ \underline{S}^{(O)}(t) &= \left(S^{(O,1)}(t), S^{(O,2)}(t), \ldots, S^{(O,N)}(t)\right); \\ \underline{S}^{(Z)}(t) &= \left(S^{(Z,1)}(t), S^{(Z,2)}(t), \ldots, S^{(Z,N)}(t)\right). \end{aligned} \tag{2.6}$$

Thus, the behavior of the FANET environment is modeled with a multi-dimension Markov chain defined as follows:

$$\underline{S}^{(\Sigma)}(t) = \left(\underline{S}^{(P)}(t), \underline{S}^{(O)}(t), \underline{S}^{(Z)}(t)\right). \tag{2.7}$$

Let us indicate the state space of the Markov chain $\underline{S}^{(\Sigma)}(t)$ as $\mathscr{S}^{(\Sigma)}$. The state transition, described in the next section, is considered stochastic because the next state does not depend only on the actions decided by the agent but also on external factors that are not controlled by the agent, such as the arrival of new jobs and changes of the zone activity states.

The agent observes the environment state defined in (2.7) at the beginning of each epoch. Let t_n be the starting instant of the epoch n, and $\underline{s}_n$ be the state

observed at that instant, that is:

$$\underline{s}_n = \underline{S}^{(\Sigma)}(t_n). \tag{2.8}$$

Let $\mathscr{P}^{(O)}$ be the set of horizontal offloading probabilities to be optimized by the :

$$\mathscr{P}^{(O)} = \left\{p_{\sigma_1}^{(O)}, p_{\sigma_2}^{(O)}, \ldots, p_{\sigma_\Psi}^{(O)}\right\}, \tag{2.9}$$

where $\{\sigma_1, \ldots, \sigma_\Psi\}$ is the set of all possible activity states, while $p_{\sigma_i}^{(O)}$, for each $i \in [1, \Psi]$, represents the offloading probability to be used by the UAV Schedulers when the zone they are covering is in the activity state σ_i.

The action, that is, the choice of the offloading probabilities, needs to meet the following requirements. Since the higher the mean job arrival rate, the higher the need for horizontal offload to avoid local overload and to balance the load in the FANET, we impose that the higher the zone activity, the higher the offloading probability. More, specifically, for two different activity states σ_i and σ_j , we apply the following rules

$$\begin{array}{l} \text{if } \Lambda_{[\sigma_i]}^{(A)} = \Lambda_{[\sigma_j]}^{(A)} \Rightarrow p_{\sigma_i}^{(O)} = p_{\sigma_j}^{(O)}, \\ \text{if } \Lambda_{[\sigma_i]}^{(A)} > \Lambda_{[\sigma_j]}^{(A)} \Rightarrow p_{\sigma_i}^{(O)} \geq p_{\sigma_j}^{(O)}, \end{array} \tag{2.10}$$

where $\Lambda_{[\sigma_i]}^{(A)}$ is the mean job arrival rate when the zone activity state is equal to σ_i.

Finally, we recall that the objective of the proposed framework is to maintain the FANET processing delay as low as possible and as equal as possible for all the jobs, independently of the activity state of the zones where they have been generated and thus independent of their access point to the FANET. To this purpose, we define the random process array $\underline{\delta}(n)$ whose generic element, $\delta_\nu(n)$, for each $\nu \in [1, N]$, represents the mean value of the FANET delays suffered during the decision epoch n by jobs processed by UAV ν. In order to minimize both the mean delay and the jitter of the elements of $\underline{\delta}(n)$, the *reward* is defined as a function of the maximum delay value observed by the UAVs at the end of the decision epoch n. Therefore, the reward function is defined as follows:

$$r_n = -\max_\nu E\{\delta_\nu(n)\}. \tag{2.11}$$

In order to calculate the term $E\{\delta_\nu(n)\}$ in (2.11), for each $\nu \in [1, N]$, the UAV ν retrieves, for each job it has processed, the information regarding the time spent by it in the FANET before being processed; this information is sent to the FO with a beacon message so that the latter can keep track of the UAV with the

lowest processing queue, and update the probability distribution $f_\nu^{(\delta)}(d)$ of the FANET delay suffered by jobs served by the same UAV. At the end of the epoch, the FO uses $f_\nu^{(\delta)}(d)$ to derive $E\{\delta_\nu(n)\}$, for each $\nu \in [1, N]$, as follows:

$$E\{\delta_\nu(n)\} = \sum_{\forall d} d \cdot f_\nu^{(\delta)}(d) \tag{2.12}$$

and therefore, using (2.11), it calculates r_n.

Likewise, the delay jitter, defined as the standard deviation of the overall delay suffered in the FANET, can be derived as follows:

$$\bar{\Gamma} = \frac{1}{N} \sum_{\nu=1}^{N} \sqrt{\sum_{\forall d} \left[d - E\{\delta_\nu^n\}\right]^2 \cdot f_\nu^{(\delta)}(d)} \tag{2.13}$$

In order to better analyze the behavior of the system, two more parameters have been taken into account: the average delays suffered in the Processing and in the Offloading Queues. These values can be derived from the average lengths of these queues and applying the Little theorem as follows:

$$\bar{D}_P = \frac{1}{N} \sum_{\nu=1}^{N} \left[\bar{Q}_\nu^{(P)} / \bar{\lambda}_\nu^{(P)}\right] \tag{2.14}$$

$$\bar{D}_O = \frac{1}{N} \sum_{\nu=1}^{N} \left[\bar{Q}_\nu^{(O)} / \bar{\lambda}_\nu^{(O)}\right] \tag{2.15}$$

where the values $\bar{\lambda}_\nu^{(P)}$ and $\bar{\lambda}_\nu^{(O)}$ are the actual arrival rates to the Processing and Offloading Queues of the UAV ν. Of course, their sum is equal to the whole mean arrival rate $\bar{\lambda}_\nu^{(IN)}$ from the zone the UAV ν is covering. Moreover, we define the job offloading ratio for the UAV ν as the portion of the job flow the UAV ν offloads. It can be calculated as follows:

$$\omega_\nu = \frac{\bar{\lambda}_\nu^{(O)}}{\bar{\lambda}_\nu^{(IN)}} \tag{2.16}$$

Let us stress that ω_ν is not exactly equal to the average value of the offloading probabilities provided by the FO to the UAV ν because, when the length of the Processing Queue of the UAV ν is the lowest among all the UAVs in the FANET, it does offload any job. Finally, a KPI that measures the system's unreliability is the probability that a given delay threshold, D_{max}, is violated. It can be easily derived from the pdf of the overall delays for flows leaving the FANET through the UAV ν as follows:

$$\Theta(D_{max}) = \frac{1}{N}\sum_{\nu=1}^{N}\sum_{\forall d > D_{max}} f_{\nu}^{(D_f)}(d) \tag{2.17}$$

The behavior of each zone ν of a FANET, for each $\nu \in [1, N]$, is modeled by a Markov Modulated Poisson Process (MMPP) $A_\nu(t)$ [86]. The process changes the vertical job-offloading rate according to the state of an underlying Markov chain. The MMPP permits us to represent both first- and second-order statistics of the real job arrival process. These statistics are sufficient to capture the behavior of queueing systems loaded by the real job arrival process [87]. The MMPP is characterized by a 2-uple $\left(\mathbf{Q}^{(A)}, \underline{\Lambda}^{(A)}\right)$, where:

- $\mathbf{Q}^{(A)}$ is the transition rate matrix of the underlying Markov chain; considering two generic states σ_i and σ_j, its generic element, $Q^{(A)}_{[\sigma_i,\sigma_j]}$, $\forall i, j \in [1, \Psi]$, represents the transition rate from σ_i to σ_j;
- $\underline{\Lambda}^{(A)}$ is the array of the job arrival rates; its generic element, $\Lambda^{(A)}_{[\sigma_i]}$, $\forall i \in [1, \Psi]$, represents the mean value of the Poisson process modeling the job arrival rate when the underlying Markov chain of $A_\nu(t)$ is in the state σ_i.

Defining $\underline{\Pi}^{(A)}$ as the steady-state probability array of the underlying Markov chain, the mean job arrival rate for each UAV can be evaluated as

$$\overline{\lambda}_A = \underline{\Lambda}^{(A)} \cdot \left(\underline{\Pi}^{(A)}\right)^T. \tag{2.18}$$

Indicating the average transmission bit rate of each link as r_L (bit/s), and the mean job size, expressed in bits, as $\overline{J}$, the mean transmission rate for horizontal offload between two UAVs of the FANET is:

$$\overline{\mu}_O = \frac{\overline{J}}{\overline{T}_O}(\text{job/s}). \tag{2.19}$$

Likewise, the mean processing rate in the CE of a UAV depends on the clock frequency of the CE CPU, φ_{CE}, and the mean number of CPU operations required to process a job, $\overline{\gamma}$, as follows:

$$\overline{\mu}_P = \frac{\varphi_{CE}}{\overline{\gamma}}(\text{job/s}). \tag{2.20}$$

The above two rates are the serving rates of the two queues Q_O and Q_P, respectively.

Now, let us derive the transition rate matrix $\mathbf{Q}^{(\Sigma)}$ of the Markov chain modeling

the FANET behavior. To this purpose, let us consider two generic states:

$$\begin{aligned} \underline{s}'_{\Sigma} &= (\underline{s}'_P, \underline{s}'_O, \underline{s}'_Z) \in \mathscr{S}^{(\Sigma)}, \\ \underline{s}''_{\Sigma} &= (\underline{s}''_P, \underline{s}''_O, \underline{s}''_Z) \in \mathscr{S}^{(\Sigma)}, \end{aligned} \tag{2.21}$$

where $\underline{s}'_P$, $\underline{s}'_O$ and $\underline{s}'_A$ are three arrays whose generic elements, $s'_{(P,\nu)}$, $s'_{(O,\nu)}$ and $s'_{(A,\nu)}$, for each $\nu \in [1, N]$, respectively represent the state of the Processing Queue, the Offloading Queue and the zone activity of the UAV ν. The same holds for the arrays $\underline{s}''_P$, $\underline{s}''_O$ and $\underline{s}''_Z$.

The generic element of $\mathbf{Q}^{(\Sigma)}$, representing the transition rate from $\underline{s}'_{\Sigma}$ to $\underline{s}''_{\Sigma}$, can be expressed as follows:

$$Q^{(\Sigma)}_{[\underline{s}'_{\Sigma}, \underline{s}''_{\Sigma}]} = \begin{cases} \overline{\lambda}^{(P,\nu)}_{\underline{s}'_{\Sigma}} & \text{if } s''_{(P,\nu)} = s'_{(P,\nu)} + 1 \\ \overline{\mu}_P & \text{if } s''_{(P,\nu)} = s'_{(P,\nu)} - 1 \\ \overline{\lambda}^{(O,\nu)}_{\underline{s}'_{\Sigma}} & \text{if } s''_{(O,\nu)} = s'_{(O,\nu)} + 1 \\ \overline{\mu}_O & \text{if } s''_{(O,\nu)} = s'_{(O,\nu)} - 1 \\ Q^{(A)}_{[s'_{(A,\nu)}, s''_{(A,\nu)}]} & \text{if } s''_{(A,\nu)} \neq s'_{(A,\nu)} \\ -\sum_{\forall \underline{s}_{\Sigma} \neq \underline{s}'_{\Sigma}} Q^{(\Sigma)}[\underline{s}'_{\Sigma}, \underline{s}_{\Sigma}] & \text{if } \underline{s}''_{\Sigma} = \underline{s}'_{\Sigma} \\ 0 & \text{otherwise} \end{cases} \tag{2.22}$$

In the above equation, for the sake of simplicity, we have indicated only the variables of the whole system state that have changed. So, for example, the first condition indicates that the variable $S^{(P,\nu)}(t)$ has changed from $s'_{(P,\nu)}$ to $s''_{(P,\nu)}$, while $s''_{(P,z)} = s'_{(P,z)}, \forall z \neq \nu$, $s''_{(O,\nu)} = s'_{(O,\nu)}, \forall \nu$, and $s''_{(A,\nu)} = s'_{(A,\nu)}, \forall \nu$.

The terms $\overline{\lambda}^{(P,\nu)}_{\underline{s}'_{\Sigma}}$ and $\overline{\lambda}^{(O,\nu)}_{\underline{s}'_{\Sigma}}$ represent, respectively, the mean job arrival rate to the Processing and to the Offloading Queues of the UAV ν. The first one can be calculated as the sum of the following three terms:

- the arrival rate from the zone ν of jobs that are not horizontally offloaded according to the non-offloading probability $1 - p^{(O)}_{s'_{(A,\nu)}}$:

$$\alpha_1 = \left(1 - p^{(O)}_{s'_{(A,\nu)}}\right) \cdot \lambda^{(A_\nu)}_{s'_{(A,\nu)}} \tag{2.23}$$

- the arrival rate of the jobs from the zone ν that, although belonging to the set of jobs that should be offloaded according to the probability $p^{(O)}_{s'_{(A,\nu)}}$

imposed by the FO, are enqueued in the local Processing Queue of the UAV ν because this queue has the shortest length among the UAVs in the FANET:

$$\alpha_2 = I^{(P)}(\nu) \cdot p^{(O)}_{s'_{(A,\nu)}} \cdot \lambda^{(A_\nu)}_{s'_{(A,\nu)}} \tag{2.24}$$

- the arrival rate of jobs coming from other UAVs in the case the Processing Queue of the UAV ν has the shortest length among the UAVs in the FANET:

$$\alpha_3 = I^{(P)}(\nu) \cdot \sum_{\substack{z=1 \\ z \notin \Delta^{(P)}(\nu)}}^{N} \frac{p^{(O)}_{s'_{(A,z)}}}{\Xi^{(P)}(\nu)} \cdot \lambda^{(A_z)}_{s'_{(A,z)}} \tag{2.25}$$

,

where:

- $I^{(P)}(\nu)$ is an indicator function which is equal to 1 if the Processing Queue length of the UAV ν is the shortest one among the N, otherwise it is equal to 0. Formally, we have:

$$I^{(P)}(\nu) = \begin{cases} 1 & \text{if } s_{(P,\nu)} \leq s_{(P,z)} \ \forall z \neq \nu \\ \\ 0 & \text{otherwise} \end{cases} \tag{2.26}$$

- $\Delta^{(P)}(\nu)$ is the set of UAVs which have the same Processing Queue length of the UAV ν:

$$\Delta^{(P)}(\nu) = \bigcup_{\substack{z=1 \\ s_{(P,z)} = s_{(P,\nu)}}}^{N} \{z\} \tag{2.27}$$

- $\Xi^{(P)}(\nu)$ is the number of UAVs that have the same Processing Queue length of the UAV ν. It matches the number of items inside the set $\Delta^{(P)}(\nu)$, that is:

$$\Xi^{(P)}(\nu) = \left|\Delta^{(P)}(\nu)\right| \tag{2.28}$$

Therefore, the overall mean arrival rate of jobs to the Processing Queue of the UAV ν is:

$$\overline{\lambda}^{(P,\nu)}_{\underline{s'_\Sigma}} = \alpha_1 + \alpha_2 + \alpha_3. \tag{2.29}$$

The mean job arrival rate to the Offloading Queue of the UAV ν can be derived as the rate of the remaining part of the job arrival flow that is not sent to the

Processing Queue, that is:

$$\overline{\lambda}^{(O,\nu)}_{\underline{s}'_{\Sigma}} = \left(1 - I^{(P)}(\nu)\right) \cdot p^{(O)}_{s'_{(A,\nu)}} \cdot \lambda^{(A_\nu)}_{s'_{(A,\nu)}} \tag{2.30}$$

2.2.4 Numerical Results

In this section, we present some numerical results to evaluate the performance of the proposed FANET management framework and to compare it to some techniques previously proposed in other works in the literature. More specifically, we first present the reference scenario we considered for performance evaluation, and we describe the setup of the parameters we used to simulate its behavior. Then we illustrate numerical results collected in the considered scenario, describing some analyses against three parameters that play a key role in the behavior of the system and deriving some guidelines for system design.

We consider a FANET with 6 UAVs that cover an area where a number of video surveillance cameras are installed as the main use case. The vertical application to support consists in elaborating video streams for image recognition purposes, and this must be performed in less than $D_{MAX} = 50ms$. Moreover, let us suppose that this requirement has to be fulfilled 0.9999% of the time. Each job is an image with an average size of $J = 900$ kbytes. To process each image, we considered the MobileNet convolutional neural network [88] that, trained on the CIFAR-10 dataset [89], uses an average number $\epsilon = 12 \cdot 106$ of CPU operations. In order to statistically characterize the arrival rate process of jobs vertically offloaded by GDs, for one week, we collected measurements in a real video surveillance testbed deployed at the University of Catania Campus with video cameras generating high-definition video streams. Processing the data trace constituted by the stream of jobs offloaded to the FANET by each zone, each covered by one UAV, we identified two main activity states of each zone, hereinafter referred to as *low-activity* (L) and *high-activity* (H), i.e., the set of activities is $\mathfrak{S} = \{\sigma_L, \sigma_H\}$. The transition rate matrix and the job-arrival rate array calculated through this data analysis for each zone as the solution of an inverse eigenvalue problem [90, 91] from the traces, are:

$$Q^{(A)} = \begin{bmatrix} -5.5 \cdot 10^{-3} & 5.5 \cdot 10^{-3} \\ 5.1 \cdot 10^{-2} & -5.1 \cdot 10^{-2} \end{bmatrix} \tag{2.31}$$

$$\underline{\Lambda}^{(A)} = [1,\ 6]\,\text{kjob/s} \tag{2.32}$$

Table 2.1: Setup Parameters for the Four Scenarios

	Analysis 1	Analysis 2	Analysis 3
N	[4, 9]	6	6
μ_p (kjob/s)	2.35	[1.6, 2.35]	2.35
μ_o (kjob/s)	4.2	4.2	[1.2, 7.2]
$K_P = K_O$	500	500	500
μ_L (kjob/s)	[0.67, 1.5]	1	1
μ_H	[4, 9]	26	6

According to (2.18), this means that the mean job arrival rate is $\bar{\lambda_A}$ =1.5 kjob/s. UAVs are equipped with an INTEL NUC 10 Barebone Core i7 CE with a clock frequency φ_{CE} = 28.2 GHz, meaning that the job processing rate is μ_P = 2.35 kjob/s, and energy consumption required to process one job is equal to ς = $59.1mJ$. Let us notice that we neglect the computation load due to the presence of the FO running on the *Master-UAV* because it requires a number of operations of about $3.5 \cdot 104$ (due to the forward pass in the FO neural network), which is negligible compared to the average number $\epsilon = 12 \cdot 106$ of operations needed to process a job. Let the link transmission rate of the UAV-2-UAV links be equal to r_L = 30 Mbit/s, consequently, the job transmission rate is μ_O = 4.2 kjob/s. Beacon messages sent by each UAV to the Master-UAV have a payload size of 2 bytes. These messages are generated each time a job is enqueued in the processing queue and each time a job is served by the CE. Therefore, the transmission rate required to transmit beacon messages from each UAV, including the overhead due to the protocol headers (assuming that some header compression technique is used [92, 93] is equal, on average, approximately to 50 kbit/s. Let us notice that this value is negligible if compared with the link transmission rate r_L of the UAV-2-UAV links, and therefore we can neglect any effect of beacon messages on the UAV-2-UAV data transmission for horizontal offload.

The resulting utilization coefficient of the FANET is $\rho^{(F)}$ = 0.638. Let K_P = K_O = 500 jobs be the size of the Processing and the Offloading Queues of each UAV. On top of that, we have performed three different analyses, each based on the reference scenario described so far, by varying some relevant FANET configuration parameters, specifically the number of UAVs covering the area, the computation power of the CE mounted onboard UAV, and the link transmission rate of the UAV-2-UAV links. The simulation parameters for each performance analysis are summarized in Table 2.1. For performance comparison purposes, we

consider the management policy that is, to the best of our knowledge, the most appropriate one in the current literature to use for performance comparison. This is the Probabilistic Computation Offloading (PCO) proposed in [70]. Moreover, we compare our framework with two heuristic algorithms, i.e., Local Drone Only (LDO) and Uniform Selection (US), commonly used in similar works for performance comparison. The details are described as follows:

- *PCO* tries to achieve the optimal scheduling solution by enabling each UAV to independently make online offloading decisions based on a certain offloading probability, which is determined by parameter settings and adapted to different service scenarios. The optimal probabilities are achieved in an iterative way using the ADMM optimization method [94]. Using the PCO policy in scenarios like the one considered in this research, in which the system behavior is highly dynamic because the activity of zones changes frequently, requires that offload probabilities have to be recalculated at each change by using the ADMM method. Since the ADMM method is computationally and timely demanding, the only solution that allows the PCO to work in real-time is an offline calculation of the optimal offloading probabilities for each UAV. This, of course, is suitable for our comparison purposes but is not efficient in real scenarios where the FO has to follow system dynamics to achieve good performances;

- LDO policy never offloads any job to other UAVs, hence each job is computed by the Dwell UAV;

- US policy randomly chooses the horizontal offloading probability each time a job is received from a UAV.

The experiments were performed using a simulator based on OpenAI Gym [95] and Pytorch [96]. We trained each DDQN model for 10.000 steps. Let us point out that this value ensured that the neural network received enough experience regardless of the dynamics of the environment on which the network has been trained on. In some of the trained models, the DDQN loss converges long before the end of the training phase. To diversify the experience obtained and drastically reduce the time required for the network convergence, each Agent was trained simultaneously on 10 environments with different random seeds but the same settings in terms of the number of UAVs, CE processing rate, and link transmission rate. The DDQN neural networks have two fully connected layers, each with 128 neurons. To balance exploration and exploitation, we applied an adaptive ϵ-greedy policy.

The ratio of exploration is initialized to 1 and gradually decreased down to 0.01. We use the Adam optimizer with a learning rate $\alpha = 10^{-3}$, and with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ to update θ. To avoid a correlation between the action values and the target values, instead of copying the weights of the evaluation network θ to the target network, we performed a soft update of the weights of the target network by having them slowly track the learned networks, with $\tau << 1$. This means that the target values are constrained to change slowly, greatly improving the stability of learning. We also set the replay buffer capacity to $5 \cdot 10^5$, and the batch size to $B = 32$.

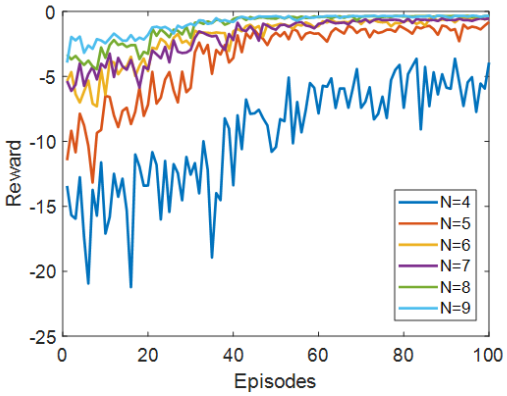


Figure 2.3: Reward convergence behavior

In order to evaluate the convergence of the proposed algorithm, in Fig. 2.3 we show the cumulative episode reward as the number of UAVs increases from 4 to 9. In Fig. 2.3 we can note that the reward converges in about 50 episodes. Since each episode lasts 10 s, we can deduce that a DDQN trained from scratch on random initial weights needs about 8 mins to fully adapt to the environment dynamics. However, transfer learning techniques could also be applied to train each DDQN model using pre-trained weights of models trained on different environments to speed up training. Therefore, we can conclude that the proposed management model, thanks to the application of DDQN, is able to easily adapt to highly dynamic systems that change their behavior in the time scale of a few seconds.

The system performance of ZTM is collected during the evaluation phase of the DQN. For performance evaluation, we consider three different analyses. In the first analysis, presented in Section VI-B1, we compare the performance of the proposed ZTM approach as the number of FANET UAVs increases from 4 to 9. Subsequently, we evaluate the performance of the FANET as the processing rate of the UAV CE increases. This strongly affects the power consumption, the maximum time of each UAV battery charge lifetime, and, consequently, the max-

imum duration of the FANET mission. Finally, in the third analysis, we evaluate how the performance of UAV-2-UAV channels affects the FANET behavior. For all the scenarios, we compare the performance of the proposed ZTM approach against the LDO, US, and PCO policies previously described. We also provide some considerations to provide the system designer with some guidelines.

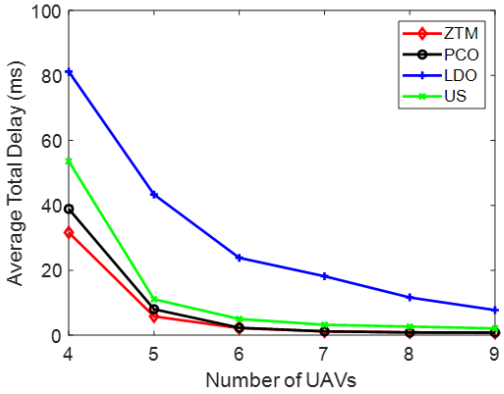
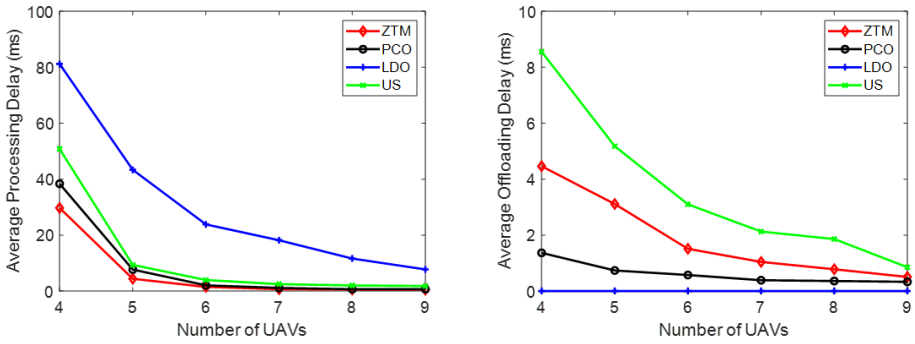


Figure 2.4: Average Total Delay vs. the number of UAVs.

1. *Impact of the Number of UAVs*: This scenario is used to evaluate the performance of the proposed framework as the number of FANET UAVs varies in the range $[4, 9]$. The other parameters that characterize the FANET are left as described so far. This analysis can also be used to size the number of UAVs in the entire fleet. Let us note that the average input-flow rate for each UAV decreases as the number of UAVs increases, and consequently, the utilization coefficient of each UAV decreases as well. In this analysis, we compare our framework with the PCO, LDO, and US policies. Fig. 2.4 shows the average total delay, derived as in (2.12). As expected, it decreases as the number of UAVs increases. This figure allows us to get a general idea of the FANET performance. The number of UAVs needed to obtain certain performances can be obtained by looking at the minimum number of UAVs whose performances satisfy the requested ones. Notice how using the proposed ZTM algorithm, by efficiently leveraging the horizontal offloading probabilities based on the state of the FANET, improves the average delay compared to PCO and the other heuristics.

   Figs. 2.5a and 2.5b present the average values of the delay suffered in the Processing Queues and in the Offloading Queues, respectively, calculated as in (2.14) and (2.15). The total delay measured for the ZTM policy is lower than the other policies, thanks to the ZTM's ability to identify and



(a) Average Processing Delay (b) Average Offloading Delay

Figure 2.5: Components of the Average Total Delay vs. number of UAVs.

prevent critical states. This explains why, on average, the offload delay is higher than PCO while the processing delay is the lowest. In this framework, PCO cannot achieve better performance than ZTM as (i) it cannot follow the time-varying dynamics of the system and (ii) it makes offloading decisions based on partial information of the system. PCO needs the average job arrival rate to optimize the offloading probabilities: this means that the average job arrival rate must be periodically estimated; moreover, it is desirable that the estimated average job arrival rate accurately matches the real network dynamics as much as possible. On the other hand, our approach does not need to know the average job arrival rate, hence it achieves better results in scenarios with real network dynamics and real data to process, without requiring any previous knowledge.

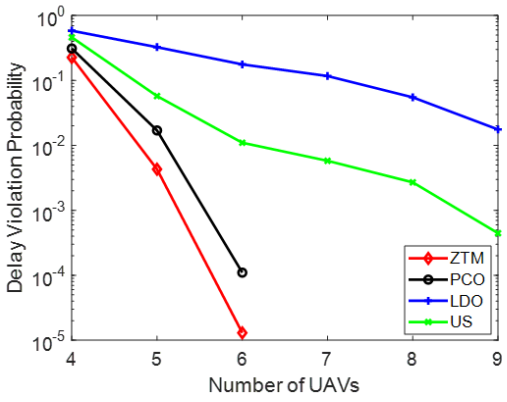


Figure 2.6: Delay Violation Probability vs. the number of UAVs.

The total average delay, however, does not give us any information regarding the probability of matching certain delay requirements. For this reason, Fig. 2.6 depicts the delay violation probability, i.e., the probability that the

delay of the system exceeds the delay threshold of D_{max}, already defined in (2.17) as a measure of the system unreliability. In our case, considering that $D_{max} = 50$ ms and the probability of it being violated must be not higher than $1 \cdot 10^{-4}$, then the appropriate minimum number of UAVs would be $N = 6$.

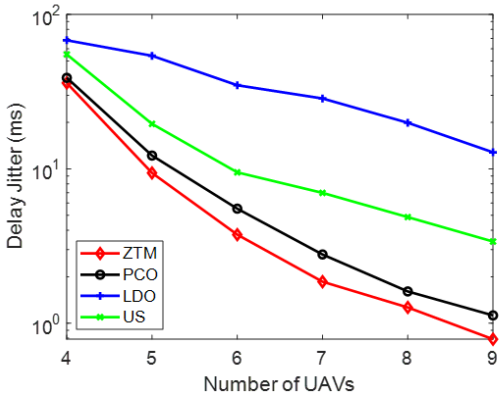
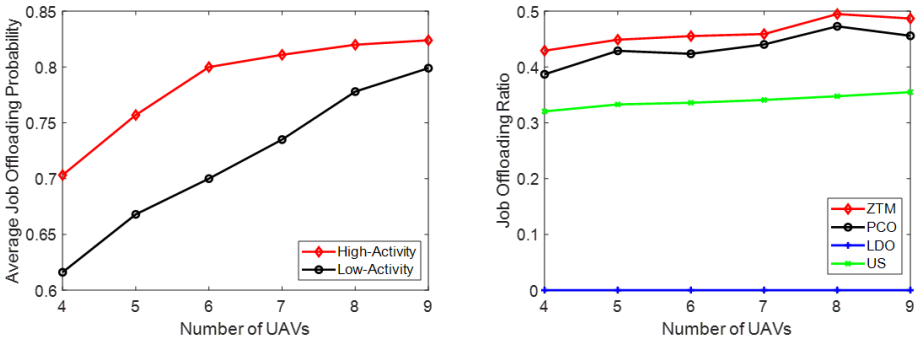


Figure 2.7: Delay Jitter vs. the number of UAVs.

Fig. 2.7 shows the measured delay jitter. We note how ZTM is not only the one that achieves the best delay performance but also the framework that presents the lowest jitter, and this allows to almost guarantee, for certain values of N, that the delay perceived by the jobs at a given moment is almost independent from the activity of the zone, that is, from the GD job generation frequency in the area where the job is generated.



(a) Job offloading probabilities (b) Measured job offloading ratio

Figure 2.8: Average offloading statistics vs. the number of UAVs.

Figs. 2.8a and 2.8b depict the average offloading probabilities used by the FO Agent and the average offloading ratio. The values of the two graphs do not coincide as the scheduler inside each UAV cannot offload if its Processing Queue is the smallest among all the UAVs. This means that as the number

of UAVs increases and the utilization coefficient of each UAV decreases, the load is almost balanced among all the UAVs, and it becomes more likely that more UAVs do not need to offload to other UAVs because their Processing Queue is often the lowest one. This is the reason why, despite the probability of offloading increasing, as shown in Fig. 2.8a, the average performed offload ratio remains almost constant. Furthermore, by using higher offloading probabilities than the other policies, the ZTM achieves the goal of not only maintaining low total delay values but also decreasing the jitter. If, on the one hand, as the number of UAVs increases, the utilization coefficient of the system decreases so that avoiding offloading could reduce the total delay, in practice, this is not the case. In fact, the presence of more UAVs guarantees more computational capacity that would remain unused if offloading is not frequently performed: this is, in fact, observable from the performance of the LDO algorithm, which tends to improve as the number of UAVs increases, but still performs worse than all the other policies.

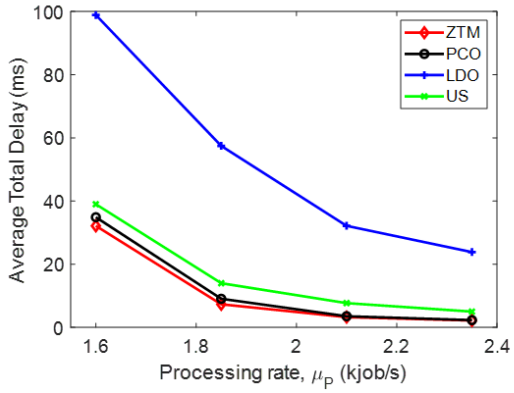
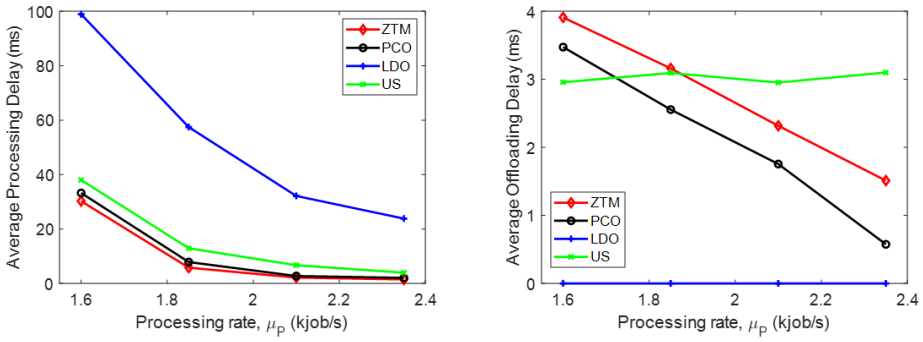


Figure 2.9: Average Total Delay vs. job processing rate $\bar{\mu}_P$.

2. *Impact of the CE Processing Rate*: In this section, we analyze how the FANET performance is influenced by the CE processing rate μ_P ranging in the interval $[1.60, 2.35]$ kjob/s, while the number of UAVs in the FANET is maintained constant and equal to $N = 6$. Increasing the CE processing rate implies decreasing the lifetime of the battery charge of each UAV in the FANET, and, therefore, the maximum duration of the FANET mission itself, as discussed later. Obviously, as the available processing rate increases, the overall system performance improves, as shown in Fig. 2.9 for the average total delay. It will therefore be necessary to find the right tradeoff between performance and power consumption.

   In Figs. 2.10a and 2.10b we show the performance of the FANET in terms of



(a) Average Processing Delay (b) Average Offloading Delay

Figure 2.10: Components of the Average Total Delay.

processing and offloading delays. As already seen in the previous analysis, the ZTM policy achieves better results by leveraging the offloading probabilities in the most critical moments, thus achieving a better processing delay among all at the cost of a greater offloading delay.

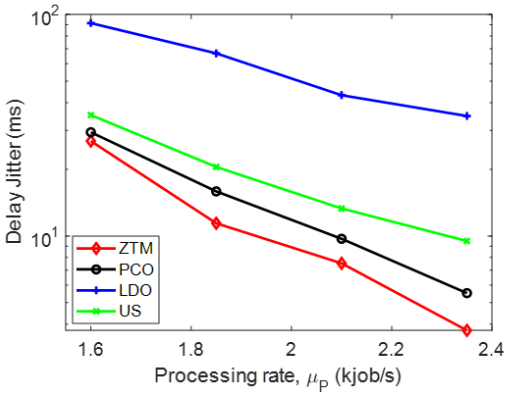
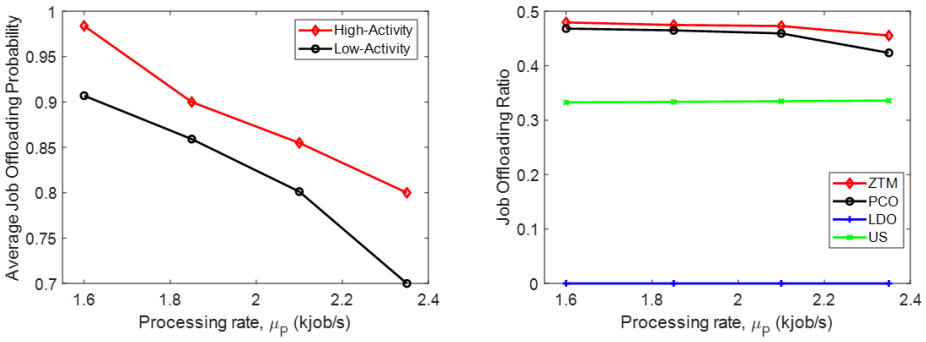


Figure 2.11: Delay Jitter vs. job processing rate $\bar{\mu}_P$

The delay jitter of the ZTM turns out to be the lowest of all policies, too, as shown in Fig. 2.11. We then show in Figs. 2.12a and 2.12b the offloading probabilities used by the Agent and the actual offloading ratio. As expected, the offloading probabilities increase as the CE processing rate decreases and, therefore, as the utilization coefficient of each UAV increases. This is a different behavior compared to the one considered in the previous analysis, where instead, the offloading probabilities increased as the utilization factor decreased because in the latter, the presence of a greater number of UAVs, and therefore of overall processing queues and computing elements, allowed the ZTM to intelligently take advantage of the greater computational capacity available, which instead in this case does not increase nor



(a) Job Offloading Probabilities (b) Measured Job Offloading Ratio

Figure 2.12: Components of the Average Total Delay vs. job processing rate $\bar{\mu}_P$.

decrease, since the number of UAVs is fixed at $N = 6$.

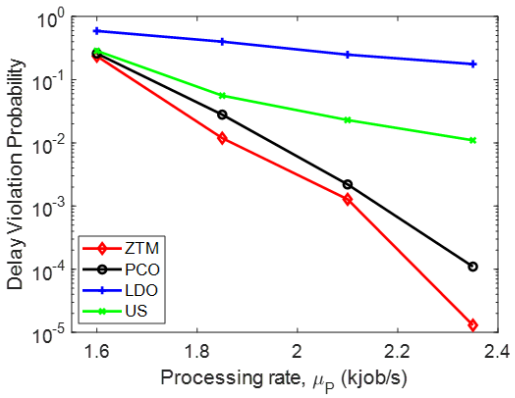
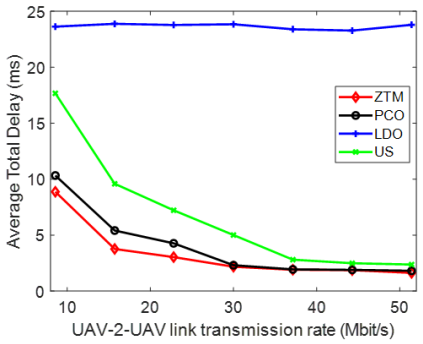


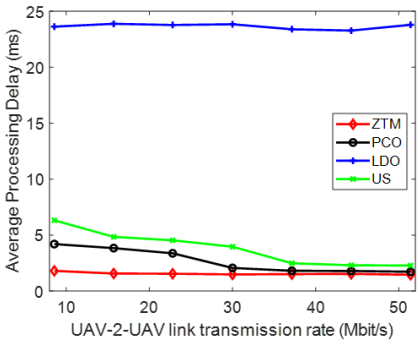
Figure 2.13: Delay Violation Probability vs. job processing rate $\bar{\mu}_P$.

We then show in Fig. 2.13 the delay violation probability. More specifically, it increases as the processing rate decreases. However, since as the processing rate decreases, the battery charge duration increases, the most appropriate processing rate should be the smallest for which the delay requirements are fulfilled with a certain probability so that both the battery duration is maximized while fulfilling delay requirements.

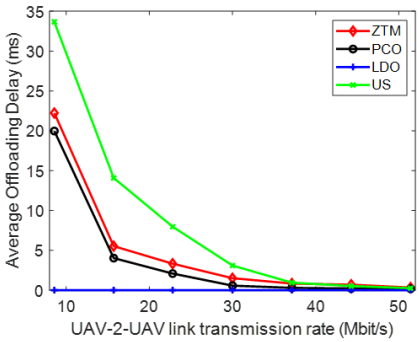
3. *Impact of the UAV-2-UAV Link Transmission Rate*: The third analysis we carried out on the reference system regards the impact of the UAV-2-UAV link transmission rate on the overall performance. As expected, the delay decreases as channel conditions improve. The ZTM manages to obtain better performances than the other approaches, as shown in Figs. 2.14a, 2.14b and 2.14c. The processing delay for the ZTM remains almost the same even for low link transmission rate values; on the other hand, the performance of the other policies suffers from the poor capacity of the bit rate.



(a) Average Total Delay



(b) Average Processing Delay



(c) Average Offloading Delay

Figure 2.14: Delay performance vs. UAV-2-UAV link transmission rate.

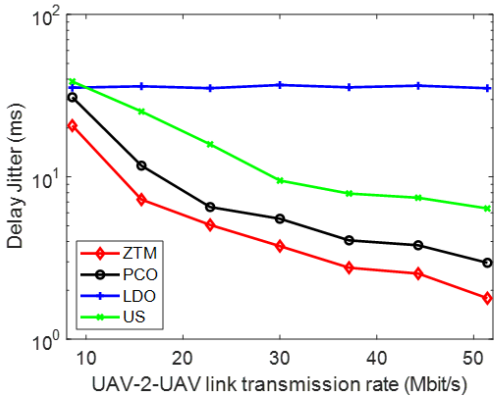
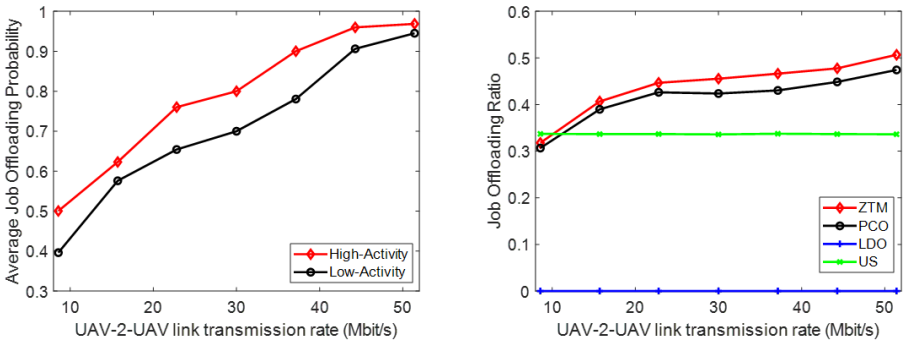


Figure 2.15: Delay Jitter vs. UAV-2-UAV link transmission rate.

The lower channel capacity is also reflected in the performance of the delay jitter, which, as expected, increases as the transmission rate decreases, as shown in Fig. 2.15. Also, this metric shows better performance for the proposed ZTM policy. As expected, the average job offloading probabilities (Fig. 2.16.a) and the average offloading ratio (Fig. 2.16.b) decreases as the link transmission rate decreases both in the cases of ZTM and PCO, demonstrating how a worse channel capacity makes the delay perceived by



(a) Job Offloading Probabilities (b) Measured Job Offloading Ratio

Figure 2.16: Average offloading probabilities vs. UAV-2-UAV link transmission rate.

the offload queues more relevant to the total job delay, which therefore causes to lessen the offloading queues usage w.r.t the reference scenario.

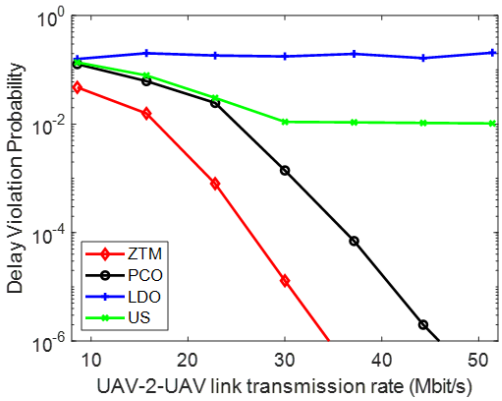


Figure 2.17: Delay Violation Probability vs. UAV-2-UAV link transmission rate.

Finally, in Fig. 2.17 we show how the delay violation probability drastically decreases for the ZTM as the link transmission rate increases; however, the same is not valid for the other policies, in which the probabilities remain relatively high.

4. *Further Considerations*: We conclude the numerical analysis by presenting some final considerations that can provide a system designer with some guidelines to design the FANET in order to fulfill user requirements. To this purpose, let us refer again to the reference system described in Table 2.1, i.e., with $N = 6$, $\mu_P = 2.35$ kjob/s and a UAV-2-UAV link bit rate of 30 Mbit/s. First, we analyze the impact of the latency-requirement level of vertical applications that use the proposed system, specifically the impact of the maximum value of tolerated delay, D_{max}, on the delay violation probability.

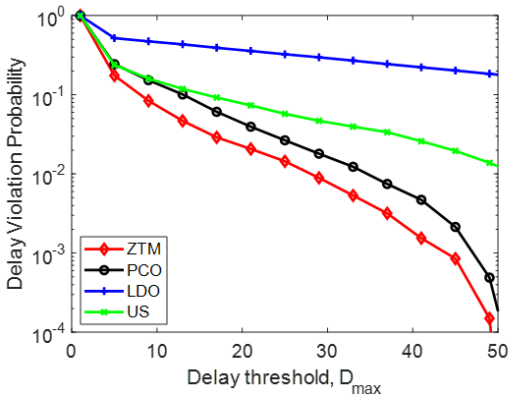


Figure 2.18: Delay Violation Probability vs. delay threshold.

To this purpose, Fig. 2.18 depicts delay violation probability as a function of D_{max} for the considered setup. As expected, reliability guaranteed by the system increases for applications that tolerate higher delays, given that the violation probability decreases. We can observe that the best tradeoff is achieved by ZTM. If this tradeoff does not satisfy applications, then some configuration parameters should be modified, as for example the number of UAVs in the FANET. If this is not compliant with the design constraints, increasing the processing rate of the CEs mounted onboard may be another viable solution. However, this has a strong impact on the maximum duration of the FANET mission, because this increases power consumption. In order to analyze this issue, as an example, we assumed that the FANET is realized with quadcopter UAVs with an engine power consumption $\varphi^{(EN)} = 66$ W, and with a Lithium battery with a capacity of 60 Wh.

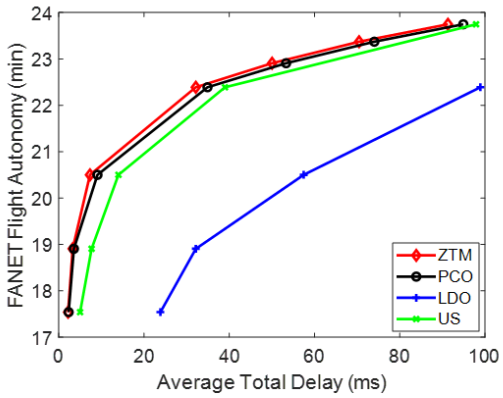


Figure 2.19: FANET flight autonomy vs. average total delay.

The total power consumption of each UAV, $\varphi^{(TOT)}$, is calculated as the sum of $\varphi^{(EN)}$ and the power consumption of the CE, $\varphi^{(CE)}$, which can be

calculated as follows:

$$\varphi^{(CE)} = \varsigma \cdot \bar{\mu}_P \tag{2.33}$$

The maximum FANET flight autonomy depends on $\varphi^{(TOT)}$ and the battery capacity, β_C , as follows:

$$\chi = \frac{\beta}{\varphi^{(TOT)}} \tag{2.34}$$

Fig. 2.19 combines the average total delay (for example, refer to the one shown in Fig. 2.4, for $N = 6$) with the maximum duration of the FANET mission, calculated as in (2.5.5), to analyze how the maximum duration of the FANET mission is related to the average total delay suffered in the FANET. Choosing a point in this figure means deciding the policy (ZTM, PCO, LDO, or US) and a specific CE computation rate, $\bar{\mu}_P$. First, we can observe that our proposed ZTM policy maximizes the FANET mission duration for a given average total delay requirement. Indeed, the ZTM policy improves the average delay over PCO by 23% and over US by 47% on average. Then, for example, for the ZTM, if the vertical application tolerates a FANET delay of 80 ms, we can use a CE with a $\bar{\mu}_P = 1.47$ kjob/s, which is with a frequency clock of 17.64 GHz. In this case, the FANET will have a flight autonomy of about 24 minutes. However, if the vertical application requires the FANET delay to be not higher than 15 ms, then the CE clock frequency should be increased at least to the value of 21.12 GHz to be able to process jobs with a rate of $\bar{\mu}_P = 1.76$ kjob/s; however, this leads to an increase of the CE power consumption and thus reducing the FANET flight autonomy to about 21 minutes.

2.3 Slicing a FANET for heterogeneous delay-constrained applications

Slicing is a key enabler to provide the network with the flexibility needed to address the requirements of all the possible vertical services supported by 5G. In [30], Yang et al. address the problem of how to satisfy different types of services using a unique UAV network, avoiding the creation of an individual network solution for each service. For this reason, the UAV network resources are virtually isolated and divided into two types of slices (i.e. URLLC and MBB). In [31],

Xilouris et al. discuss on the extension of network slicing in the case of UAV-based 5G-network deployment, while in [32] the problem of jointly allocating CPUs and VNFs for network slicing applications is addressed. In [33] Van Huynh et al. present a framework that maximizes a provider's reward and allows reducing cases in which an over-provisioning of the resources is performed, choosing which slices have to be admitted. In [34] Castellano et al. propose a framework with the purpose of implementing a heterogeneous resource orchestration. Liu et al. in [35] study the cross-domain resource orchestration and management problems regarding the implementation of a dynamic network slice in mobile networks. By developing a distributed cross-domain resource orchestration protocol, they optimize the cross-domain resource orchestration and provide the performance and functional isolations among network slices. Finally, in [36], D'Oro et al. describe Sl-EDGE, a MEC slicing framework to help network operators to create heterogeneous slice services on common edge devices.

Given the requirement for future networks to provide a full e2e automation of network and service management, the literature is still missing studies regarding the possibility of using a FANET to provide MEC facilities to users interested in the services offered by two or more slices, finding a tradeoff between the requirements characterizing each slice and the MEC resources installed on board UAV.

A possible solution to improve performance inside a slice dedicated to time-critical vertical applications could be increasing the amount of computing power of the CEs dedicated to that slice, but this worsens the performance of the other slices. So, besides deciding the best inter-slice sharing of computation resources, it is necessary to optimize intra-slice performance. By means of horizontal offloading, overloaded UAVs have the possibility to send jobs, received for processing from ground devices, to other UAVs, hence avoiding unbalanced situations in which some UAVs introduce high processing delays, while others are just flying and not contributing with their computing capabilities. However, offloading jobs between UAVs introduces an additional delay due to the limited capacities of the wireless links. Therefore, achieving a tradeoff between energy consumption and processing delays and balancing performance among different slices according to user requirements is critical and needs to be carefully considered.

With all this in mind, we propose a zero-touch management framework for the above FANET. An inter-slice orchestrator is introduced to split the computation power of the CE of each UAV between the different slices, while an intra-slice orchestrator is in charge of managing horizontal offload among UAVs to obtain

load balancing among UAVs, so minimizing both mean delays and delay jitter. We introduce an extensive simulation campaign for performance evaluation, in order to demonstrate the power of the proposed framework and provide designers with some guidelines in optimizing some key system parameters

2.3.1 System Description

We propose a FANET-based framework providing 5G network slicing to a remote area for heterogeneous vertical applications. As already defined in the previous sections, we consider a huge number of devices, referred to as GD, that generate jobs to be processed with stringent requirements in terms of delay. We assume that the geographic area where these devices are deployed is not covered by any network infrastructure that is able to provide them with edge computing facilities. For this reason, a FANET is used to provide this area with edge computing on demand. This way, GDs can offload jobs to the FANET for processing. In the sequel, for the sake of simplicity, we refer to a group of homogeneous devices as a Ground Device Group (GDG). Therefore, each GDG is characterized by a specific job-offloading rate and a specific set of QoS requirements, expressed in terms of mean delay and delay jitter. For this reason, the FANET is required to provide different network slices, one for each GDG.

One of the main challenges in the proposed system is constituted by the duration of the battery charge of each UAV. To this purpose, let us notice that increasing the capacity of the battery, on the one hand, tends to extend the UAV flight autonomy but, on the other hand, increases the UAV payload, and therefore may cause an increase in power consumption during flight. Therefore, the choice of the battery to be mounted on board is crucial but also out of the scope of this research. Moreover, let us note that to mitigate the above problem of short UAV flight duration and obtain a sufficient duration of the mission, the latter is not entrusted to a single UAV but to a fleet of UAVs organized in FANET. In order to make the system zero-touch, i.e. with no human intervention, and even make it work in areas not supplied by the electrical grid, a Charge Station is installed close to the geographic area where the network service is requested. The Charge Station could even be mobile, to be easily installed when needed, and easily removed when the FANET ends its mission. With the aim of minimizing the service downtime due to flat batteries, we assume that the Charge Station is equipped with an automatic battery swap mechanism to replace the batteries of UAVs very quickly and with no human intervention. Moreover, we assume that a sufficient number of charged batteries is always present in the Charge Station, such that a landed UAV

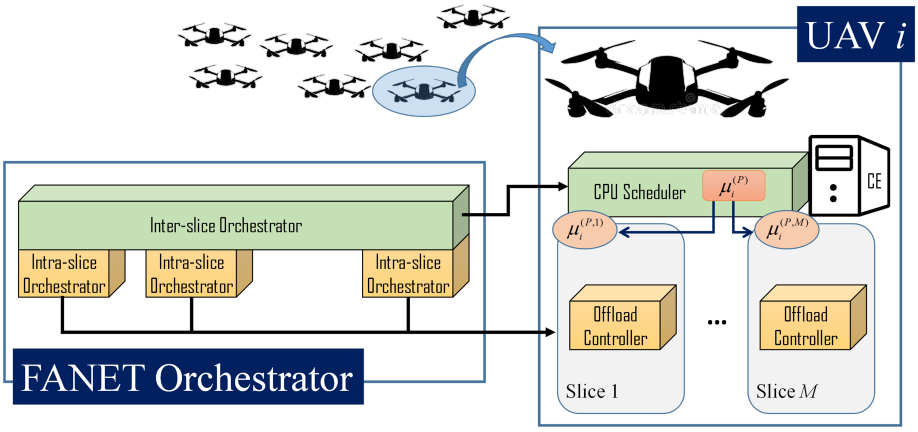


Figure 2.20: FANET management architecture.

can replace its battery and return to the FANET immediately. The Charge Station is also equipped with a renewable energy generator (e.g., a wind power microeolic system or a photovoltaic power generator) to supply pads where flat batteries are connected for recharging. In addition, if a connection with the structured Internet is needed, a communication module is installed, for example, for satellite communications. The Charge Station is also the place where UAVs are loaded with the edge capabilities they offer (i.e. edge computing applications they run). Let N be the number of UAVs in the FANET. The geographic area covered by the considered FANET is subdivided in zones, each assigned to one UAV of the FANET. Therefore, N is also the number of zones to be served by the FANET. In order to provide GDs with edge computing, each UAV is equipped with a CE. The vertical offloading process is time-variant according to the current state of activity of each GDG in each zone: the higher the activity, the higher the emission rate of jobs offloaded to the FANET to be processed. Let M be the number of network slices the FANET has to provide. Therefore, each CE is sliced in M portions, each dedicated to serving one slice. Wireless links between UAVs are sliced in M portions as well with the aim of providing communication channels that are isolated from each other. Moreover, in order to guarantee isolation in accessing the CE of each UAV, one queue is associated with each CE slice. In this way, jobs of a given slice that find the slice CE busy to serve previous jobs are accommodated in the specific queue of that slice.

The management architecture of the FANET is shown in Fig. 2.20. The CE Scheduler is in charge of assigning a portion of the whole CE computing rate to each slice. In order to provide slices with priorities, the CE Scheduler can apply Weighted Fair Queuing (WFQ) and Priority Queuing (PQ) techniques [40]. As shown in Fig. 2.20, for the generic UAV i, we will indicate the whole CE computing

rate of its CE as $\mu_i^{(P)}$, while $\mu_i^{(P,\sigma)}$ represents the portion of CE computing rate assigned to the slice σ, for $\sigma \in [1, M]$. Of course, we have: $\sum_{\sigma=1}^{M} \mu_i^{(P,\sigma)} = \mu_i^{(P)}$.

For each job, we name the UAV that has received it from the ground as the Dwell UAV, and the UAV where the job is processed as the Processing UAV. The decision on whether to process a job received in a slice by a UAV locally or offload it to another UAV is made according to an offloading scheduling policy that is in charge of a local entity named Offload Controller. If it decides to offload a job, it has also to select the Processing UAV for it. M Offload Controllers, i.e. one for each slice, are available in each UAV.

As shown in Fig. 2.20, the FANET Orchestrator is hierarchically structured in two layers:

- Inter-slice Orchestrator;
- Intra-slice Orchestrators.

The *Inter-slice Orchestrator* is in charge of deciding the amount of the computing rate, $\mu_i^{(P,\sigma)}$, of the CE in the UAV i that has to be assigned to the slice σ. This decision is communicated to the CE Scheduler of UAV i for all the slices. Each *Intra-slice Orchestrator*, on the other hand, is in charge of deciding the offloading probabilities to be used by the Offload Controllers of all the UAVs for the slice it manages.

The decision problem of the horizontal offloading probabilities being sent by the Intra-slice Orchestrator of each slice to the Offload Controllers running in all the UAVs for the same slice is the same as the one defined in the previous sections.

Fig. 2.21 shows the data-plane model of the generic UAV i for a given slice σ and the settings it receives by the FANET Orchestrator. More specifically, for the generic slice σ, let us define:

- $S_i^{(P,i,\sigma)}$: the state of the Processing Queue of the UAV i, representing the current number of jobs in the Processing Queue, including the one being served in the CE;
- $S_i^{(O,i,\sigma)}$: the state of the Offloading Queue of the UAV i, representing the current number of jobs in the Offloading Queue, including the one being transmitted on the wireless TX interface;
- $S_i^{(A,i,\sigma)}$: the activity state of the zone the UAV i is covering.

The Intra-slice Orchestrator of the considered slice is in charge of deciding a set of offloading probabilities, one for each possible zone activity state.

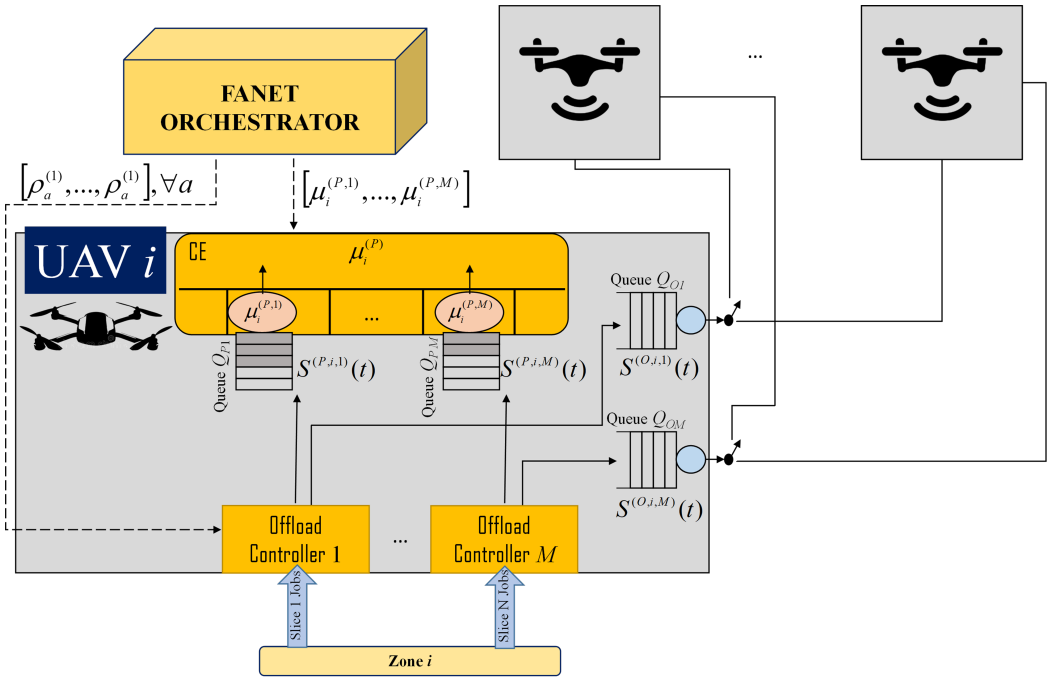


Figure 2.21: Data-plane UAV model.

During each epoch, each UAV will offload jobs coming from the GDG of a given slice σ with probability p_a^σ that depends on the activity state a of its zone. By so doing, the job arrival rate entering the FANET through the UAV i is split by the Offload Controller between the Processing Queue and the Offloading Queue.

2.3.2 KPI Description

In this section, we define the main KPIs that can be used to evaluate the proposed framework against other baselines. The first set of KPIs regards delay performance for each slice. Then, we define a profit parameter to evaluate how the choices of the system designer, implemented in the Inter-slice Orchestrator, affect the overall revenue achieved from all the slices.

The average total delay suffered (2.12), in the FANET by the jobs offloaded by the GDG, the delay jitter (2.13), and the delays of the Processing (2.14) and Offloading queues (2.15) for each slice σ are considered.

We also introduce a KPI aimed at quantifying the behavior of the system as a whole, joining the performance achieved on the single slices. To this purpose, we define a profit parameter that depends on the ability to satisfy a given target of delay for each slice. For this reason, we assume that the FANET providing service to slice σ gains a profit that increases as the average delay in this slice decreases. Therefore, the FANET Manager has to decide the amount of computing rate to

allocate to each of the slices in order to maximize its profit. Let us define $\bar{D}^{(\sigma)}_{F,TGT}$ as the *target average delay* required by the slice σ. It represents the maximum average delay that it can tolerate. The profit achieved for this slice is proportional to the distance of the actual average delay from the target average delay:

$$\varphi_\sigma = u_\sigma \cdot \left(\bar{D}^{(\sigma)}_{F,TGT} - \bar{D}^{(\sigma)}_{F} \right) \tag{2.35}$$

where the multiplicative constant u_σ, measured in PU/ms (PU: price unit), is the profit gained by the FANET Manager to provide its service to the slice σ. Notice that, as the value of u_σ increases, the FANET will be more disposed to allocate more resources to slice σ. The *total profit* obtained from the FANET can be calculated as follows:

$$\varphi_F = \sum_{\sigma=1}^{M} \varphi_\sigma \tag{2.36}$$

2.3.3 Use Case Description

We consider a FANET with $N = 6$ UAVs that covers an area where $M = 2$ different slices must be provided. The transition rate matrix and the job-arrival rate array calculated through this data analysis for each zone as the solution of an inverse eigenvalue problem from the traces for the first slice are:

$$Q^{(A)}_1 = \begin{bmatrix} -5.48 \cdot 10^{-3} & 5.48 \cdot 10^{-3} \\ 5.17 \cdot 10^{-2} & -5.17 \cdot 10^{-2} \end{bmatrix} \tag{2.37}$$

$$\underline{\Lambda}^{(A)}_1 = [0.98,\ 6.03]\,\text{kjob/s} \tag{2.38}$$

The transition rate matrix and the job-arrival rate array for the second slice are:

$$Q^{(A)}_2 = \begin{bmatrix} -5.51 \cdot 10^{-3} & 5.51 \cdot 10^{-3} \\ 1.49 \cdot 10^{-2} & -1.49 \cdot 10^{-2} \end{bmatrix} \tag{2.39}$$

$$\underline{\Lambda}^{(A)}_2 = [0.6,\ 4.0]\,\text{kjob/s} \tag{2.40}$$

This means that the mean job arrival rate is $\bar{\lambda}_{A1} = 1.46$ kjob/s for the first

slice, and $\bar{\lambda}_{A2} = 1.51$ kjob/s for the second slice. Let $K_P = K_O = 500$ jobs be the size of the Processing and the Offloading Queues of each UAV. As far as the CE computing power, we have carried out two different analyses. In the first analysis, we have studied the two slices separately, by varying the CE computing amount assigned to each slice, $\mu_1^{(P)}$ and $\mu_2^{(P)}$, respectively, in the interval [1.75, 2.75] kjob/s, and taken equally for all the UAVs. The utilization coefficient of each slice of the FANET is defined as the ratio between the whole job arrival rate from all the zones and the total computing rate of all the UAV CEs. Therefore, since the computing rate assigned to each FANET is varied in our analysis in the interval [1.75, 2.75] kjob/s, the utilization coefficient of slice 1 has ranged in the interval [0.53, 0.83], while the one of the slice 2 has ranged in the interval [0.55, 0.86].

A second analysis has been done against the total computing rate of each CE ranging in the interval [4, 5] kjob/s, and assuming a target average delays of $\bar{D}_{F,TGT}^{(1)} = 40$ PUs for slice 1 and $\bar{D}_{F,TGT}^{(2)} = 80$ PUs for slice 2. When the fraction of the CE computing rate for one slice increases, the computing rate for the other slice decreases. Therefore, in this case, the analysis has regarded the impact of the overall computing rate and of its distribution between the two slices on the profit gained by the FANET Manager, calculated for $u_1 = 2$ PUs and $u_2 = 1$ PU. As far as the transmission link between UAVs is concerned, we assumed a total job transmission rate $mu_O = 5.4$ kjob/s, which is subdivided to the two slices as follows: $mu_1^{(O)} = 1.2$ kjob/s for the first slice and $mu_2^{(O)} = 4.2$ kjob/s for the second slice.

We compare our framework with two heuristic algorithms, i.e., Local Drone Only (LDO), commonly used in similar works for performance comparison [19,20], and Uniform Selection (US), already described in the previous sections.

To ensure that each model converged to its maximum value, we trained each A2C model for 100.000 steps. To diversify the experience obtained and drastically reduce the time required for network convergence, we leveraged the capability of the A2C method to train multiple agents on multiple environments with different random seeds simultaneously. This enabled us to speed up the convergence time and to avoid the need for a replay buffer. Moreover, the final policy is more robust to changes in the environment, especially when compared to agents trained on a single scenario. Moreover, pre-trained models can be deployed to further speed up the convergence of the agent in new, previously unseen scenarios.

To evaluate the convergence of the proposed method, in Fig. 2.22 we show the smoothed losses of both the Actor and Critic networks, calculated for slice 1 in

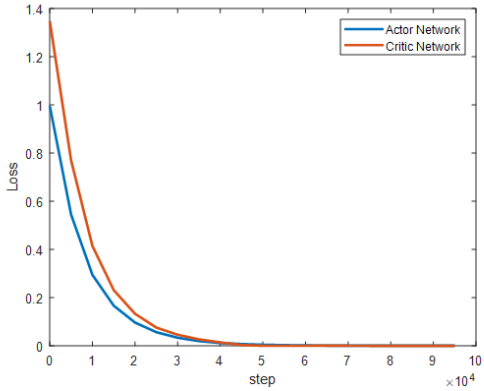
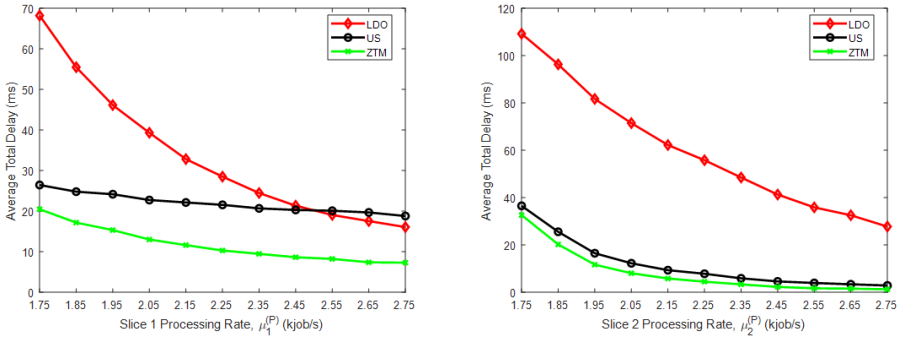


Figure 2.22: Actor and Critic Loss.

an episode of 10.000 s, with $mu_1^{(P)} = 2.35$ kjob/s, $mu_1^{(O)} = 1.2$ kjob/s. As we can see in the figure, the loss is able to converge, and this happens only in about $5 \cdot 10^4$ steps. System performance, collected during the evaluation phase of the network, will be presented in the next section.



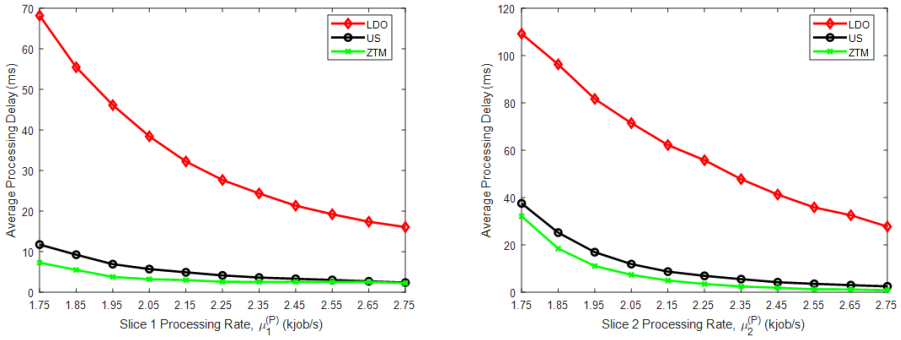
(a) Slice 1 (b) Slice 2

Figure 2.23: Average total delay.

2.3.4 Performance Evaluation

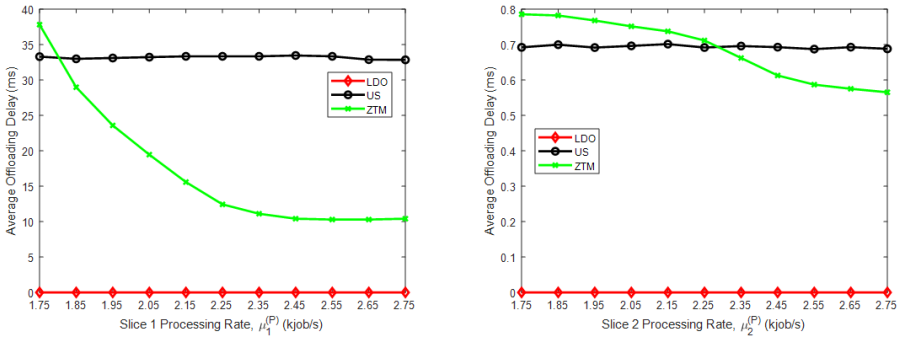
In this section, we evaluate the performance of the proposed FANET management framework, referred to in the figures as the Zero-Touch Management (ZTM) framework, through some simulation experiments. The system performance of ZTM is collected during the evaluation phase of the A2C method. For performance evaluation, we measured the performance the FANET provides to each slice as the computing rate of the UAV CE allocated to that slice increases in the range [1.75 2.75] kjob/s. As expected, as the available computing rate for each slice increases, the overall performance improves, as shown in Fig. 2.23, where the average total per-slice delay is exhibited. The computing rate needed for each

slice to obtain a certain target performance can be obtained by looking at the minimum computing rate whose performance satisfies the requested ones. Considering the two heuristics LDO and US for performance comparison, notice how using the proposed ZTM algorithm, the average delay is improved by efficiently leveraging on the horizontal offloading probabilities based on the state of the FANET. It is also possible to notice in Fig. 2.23a, i.e., for slice 1, that LDO starts to perform better than US for computing rates higher than 2.55 kjob/s, which reveals how, for high computing rate values, offloading may not be the right answer for reducing the overall delay.



(a) Slice 1 (b) Slice 2

Figure 2.24: Average computing delay.

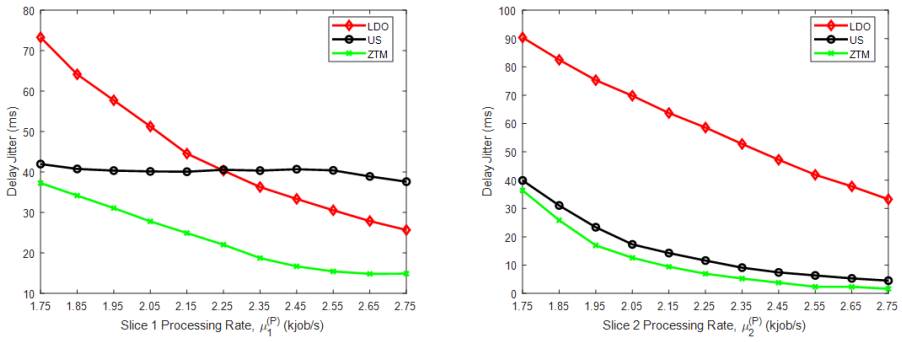


(a) Slice 1 (b) Slice 2

Figure 2.25: Average offloading delay.

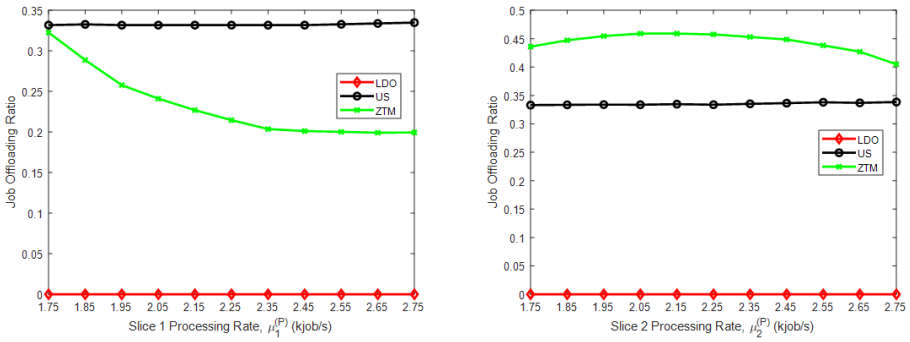
Figs. 2.24 and 2.25 present the average values of the delay suffered in the Processing Queues and in the Offloading Queues, respectively, calculated as in (2.14) and (2.15). The total delay measured for the ZTM policy is lower than the other policies thanks to the ZTM's ability to identify and to prevent critical states. On one hand, the processing delay of the ZTM policy is always lower than the other policies. On the other hand, the offloading delay is higher when the available

computing rate is low, and starts to decrease as the computing rate increases.



(a) Slice 1 (b) Slice 2

Figure 2.26: Delay Jitter.



(a) Slice 1 (b) Slice 2

Figure 2.27: Job offloading ratio.

Fig. 2.26 shows the measured delay jitter. As we can see in the figure, ZTM is not only the one that achieves the best performance in terms of average delay but also the policy that presents the lowest jitter. Hence, the delay perceived by the jobs at a given time is almost independent from the GD job generation rate where the jobs are generated, that is, from the activity of the zone. We then show, in Fig. 2.27, the job offloading ratio. As expected, the offloading ratio decreases using the ZTM policy as the CE computing rate increases and, therefore as the utilization coefficient of each UAV decreases. This happens because as the CE computing rate increases, UAVs process the incoming jobs on their own more easily, without the help of other UAVs. On the other hand, both the offloading rate for the US and LDO policy do not change as the CE computing rate increases.

Finally, we present some results regarding the impact of both increasing the overall CE computing rate and its distribution between the two slices.

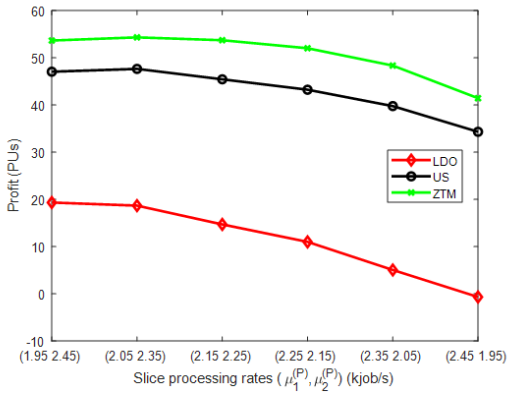


Figure 2.28: FANET profit.

First, in Fig. 2.28, we considered a CE computing rate of $\mu^{(P)} = 4.4$ kjob/s, and presented the FANET profit, calculated as in 2.36, by varying the CE computing rate assigned to each slice, $\mu_1^{(P)}$ and $\mu_2^{(P)}$, in the interval [1.95, 2.45] kjob/s. The non-monotonic trend is due to the fact that, increasing the CE computing rate for a slice improves profit gain for that slice, but reduces profit gain for the other slice. Therefore, for a given overall CE computing rate, there is a 2-uple $(\mu_1^{(P)}, \mu_2^{(P)})$ that maximizes the profit for the FANET Manager.

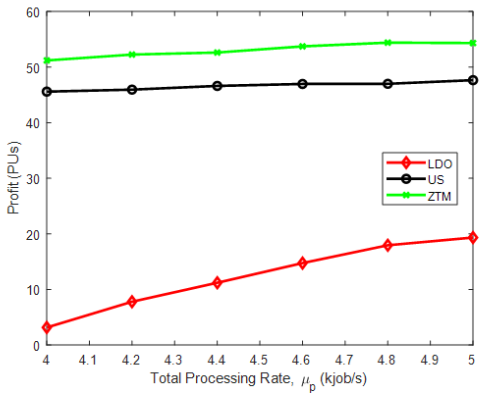


Figure 2.29: Maximum profit gained with the best allocation of the CPU computation power to the two slices, compared with the two heuristics.

Fig. 2.29 compares the maximum profit gained with the best allocation of the CE computing rate to the two slices achieved by ZTM with the two considered heuristics. In this figure, we can appreciate another time the gain achieved by the proposed ZTM framework. The CE computing rate strongly affects the power consumption, consequently the maximum time of each UAV battery charge lifetime, and therefore the maximum duration of the FANET mission. For this reason, in Fig. 2.30 we show the FANET flight autonomy as a function of the computing

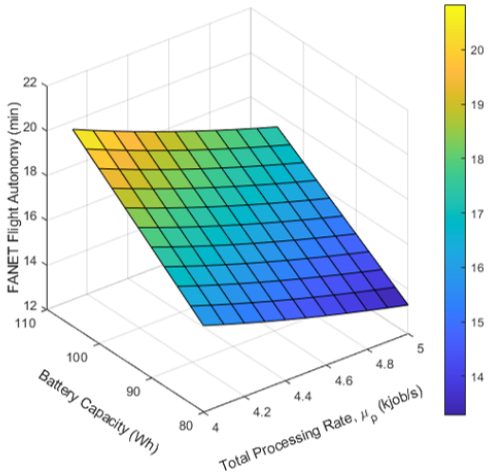


Figure 2.30: FANET flight autonomy.

rate of the UAV CE. As expected, the FANET flight autonomy increases as the battery capacity increases and the CE computing rate decreases. From a joint analysis of Figs. 2.28–2.30, the system designer can decide the CE computing rate that maximizes profits for the FANET owner and the battery capacity that, given the CE computing rate, allows missions with a given time duration. Then, the analysis carried out in Fig. 2.28, allows the Inter-slice Orchestrator to share the total CE computing rate with the slices to obtain the target maximum profit.

2.4 Comparison of centralized and distributed DRL approaches for FANET Optimization

The model-free feature of DRL makes it easier to map the high-dimension action space to a specific action without the convexity of objective or reward functions. However, these DRL algorithms do not perform all in the same way. Adopted in several optimization problems for FANET systems, from trajectory optimization in three-dimensional space to channel-powered joint optimization with pre-defined trajectories [97, 98], Multi-Agent Q-learning usually performs poorly in complex environments [99]. As the number of UAVs in the FANET increases, sometimes Single-Agent DRL approaches fall short of achieving reasonable performance due to the exponential increase in the state and action space. On the other hand, Multi-Agent DRL approaches have already demonstrated their capabilities on training sets of AI agents that can collaborate to solve complex tasks [100, 101].

For this reason, we investigate both centralized and decentralized DRL approaches, i.e., Single-Agent and Multi-Agents frameworks, based on DQN [102],

A2C [30] and PPO [103], in which each agent chooses the best offloading probabilities to forward incoming jobs to neighboring UAVs. The horizontal offload decision problem is defined as a Markov Decision Process (MDP) that is solved via Multi-Agent DRL (MADRL).

2.4.1 Framework

In this framework, N agents interact with the environment, one in each UAV of the FANET, and are characterized by a set of states, S, and each of them periodically take an action from a set, A. The state of the environment as a whole at time t, that is $s_t \in S$, consists of the state of the Processing and Offloading Queues of all the UAVs and the activity state of all the N zones served by the FANET.

At the beginning of each decision epoch e (let us indicate that instant as t_e), each agent obtains its private observation o_{n,t_e} and takes its own action a_{n,t_e}. Then, the environment will evolve into a new state. Finally, at the end of the same epoch, each agent sends its own new observation to the neighboring agents and obtains a reward $r_{n,e}$.

Thus, we define the observation, action and reward function for each agent during decision epoch e as follows:

1. *Observation* o_{n,t_e}: the observation is constituted by the Processing and Offloading queues of the local UAV, that is, $Q_n^{(P)}(t_e)$ and $Q_n^{(O)}(t_e)$, and the Processing queues $\{Q_m^{(P)}(t_e), \forall m \in N, m \neq n\}$ and Offloading queues $\{Q_m^{(O)}(t_e), \forall m \in N, m \neq n\}$ of the neighboring UAVs. Additionally, we also include the set of the total number of job requests arrived to each UAV during the previous decision epoch.

2. *Action* a_{n,t_e}: the action is the choice of the n-th UAV's *offloading probability*, that is the probability to offload jobs received from the zone that it is serving to a neighboring UAV. It is defined as $a_{n,t_e} \in \mathfrak{P}$, where $\mathfrak{P}$ is the discrete set of possible offloading probabilities. Of course, we have $0 \leq a_{n,t_e} \leq 1, \ \forall n, \forall e$.

3. *Reward* $r_{n,e}$: we use the same reward function already defined in 2.11

Then, we can define the state of the entire environment, s_{t_e}, and the whole action a_{t_e} at decision epoch e, which are also the state space and action space in the single-agent scenario, as follows:

1. State s_{t_e}: the state consists of the observations of all the agents, which is expressed as $s_{t_e} = \{o_{n,t_e}, \forall n \in N\}$

Table 2.2: DRL hyperparameters

Algorithm	Learning Rate	Layers	Batch Size	Runtime
DQN	1e-4	2x128	32	8m55s
A2C	7e-4	2x128	40	11m19s
PPO	3e-4	2x128	64	14m42s

2. Action a_{t_e}: the action consists of the actions of all the agents, which is $a_{t_e} = \{a_{n,t_e}, \forall n \in N\}$

2.4.2 Numerical Results

In this section, we evaluate the performance of the proposed framework by showing some numerical results.

We consider a FANET with $N = 4$ UAVs, that provide on-demand MEC services. To characterize the job offload process by GDs in the area covered by the FANET, we collected measurements at the University of Catania campus, in which different GDs are deployed and require different services. By processing the measured job generation traces, we identified two main activity states, hereinafter referred to as low-activity (L) and high-activity (H), i.e. the set of possible activity states for each zone is $\mathfrak{S} = \{\sigma_L, \sigma_H\}$. The transition rate matrix and the job-arrival rate array calculated through this data analysis for each zone as the solution of an inverse eigenvalue problem [104] from the traces for the geographic area are:

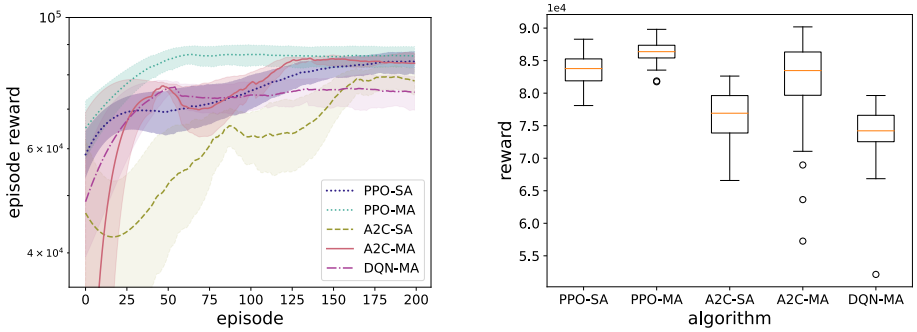
$$\mathbf{Q}^{(A_1)} = \begin{bmatrix} -5.5 \cdot 10^{-3} & 5.5 \cdot 10^{-3} \\ 1.6 \cdot 10^{-2} & -1.6 \cdot 10^{-2} \end{bmatrix}$$

$$\underline{\Lambda}^{(A_1)} = [1.3,\ 2.3]\,\text{kjob/s}$$

We present two different analyses. In the first analysis, we show the gain of distributed DRL approaches compared to a centralized approach, in terms of both convergence speed and average latency. Then, in the second analysis, we compare the model performance in an evaluation phase of 100 episodes, in which we keep track of the cumulative episode reward and mean delay.

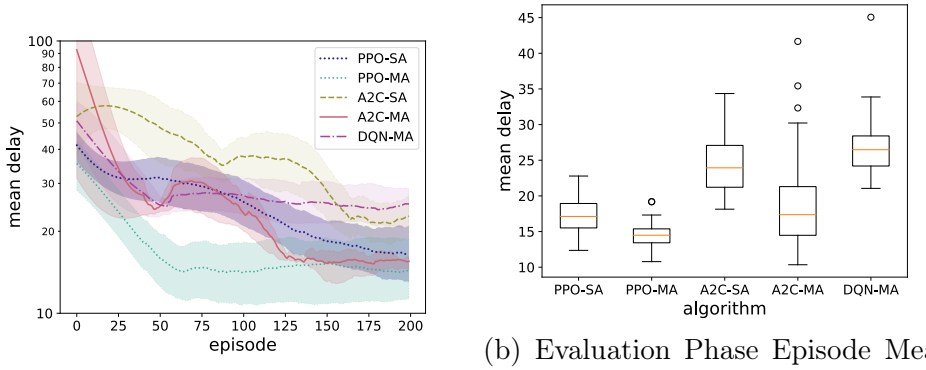
The main hyperparameters of the DRL algorithms are summarized in Table 2.2. We trained each agent for $5 \cdot 10^3$ epochs in a server with two NVIDIA GPU RTX 3070. We used the Adam [105] optimizer to train the neural networks. The experiments were performed via a simulator based on OpenAI Gym [95] and

Stable Baselines3 [106]. After the training phase, both the model and the network parameters are saved for testing.



(a) Episode Reward (b) Evaluation Phase Episode Reward

Figure 2.31: Episode Reward comparison among different DRL algorithms



(a) Episode Mean Delay (b) Evaluation Phase Episode Mean Delay

Figure 2.32: Delay comparison among different DRL algorithms

First, in Figs. 2.31a and 2.32a, we compare the training performance of Single-Agent approaches based on A2C and PPO with Multi-Agent approaches based on DQN, A2C and PPO. DQN could not be deployed in the Single-Agent mode since it is not able to emit multiple actions each time step. Observe from Fig. 2.31 that the cumulative episode reward achieved by PPO, and especially the Multi-Agent version, achieves better performance compared to the other algorithms in both terms of convergence speed and mean episode reward at convergence. As shown from the shaded region in Fig. 2.31a, PPO gives better and stable results when compared to other algorithms. PPO also resulted to be more robust to the hyperparameters choice, while on the other hand A2C results varies drastically with minor hyperparameters changes. Fig. 2.32a shows that the average delay decreases as the reward increases, indicating that the reward function is well

designed. The convergence speed-up of the Multi-Agent algorithms, especially PPO, is appreciable compared to the Single-Agent approaches, which are not able to reach the same performance of Multi-Agent algorithms. This may be due to the fact that Single-Agent approaches have a much bigger action space, whereas each agent in the Multi-Agent scenario has fewer actions to emit every step (one action per agent, compared to N actions in the Single-Agent scenario). Moreover, PPO seems to achieve good performance even in the Single-Agent scenario. This may be due to the PPO feature of having a clipped loss function, that has a very strong impact on the overall robustness and stability of the algorithm. Specifically, PPO clips the affect of the advantage such that an actor's action distribution for a particular state doesn't move too much during training. On the other hand, in A2C there can be issues where a particular training trajectory can significantly influence an actor's preferred action, causing it to be bad at exploration. PPO solves this issue by preventing itself from being too much influenced from any particular training round.

2.5 A Learning Framework of Federated FANETs to Provide Edge Computing to Future Delay-Constrained IoT Systems

In realistic scenarios, each time a FANET is deployed to provide service provisioning, the dynamics of the network traffic in the underlying area covered by the UAVs could be very different from the ones experienced in previous deployments. In the example shown in Fig. 2.33, we have two FANETs, the former deployed in a rural area, and the latter deployed in a city area. Since the two geographic areas are characterized by different vertical offloading activity behaviors, the neural networks optimizing horizontal offload in the two above FANETs are deeply different. If one of these FANETs will be involved in another mission, to provide edge computing to a new geographical area presenting different activity behavior, or in the case that the same area a FANET is covering changes its activity behavior abruptly for an unexpected event, the neural network trained on the previous behavior has to be re-trained to readjust the policy to the new scenario. This would be done from scratch, based on never seen underlying activity conditions. This, in turn, can lead to unacceptable and unstable performance. This is also due to the fact that the RL agents overfit their policies to their environments and therefore are usually not able to generalize to different environments.

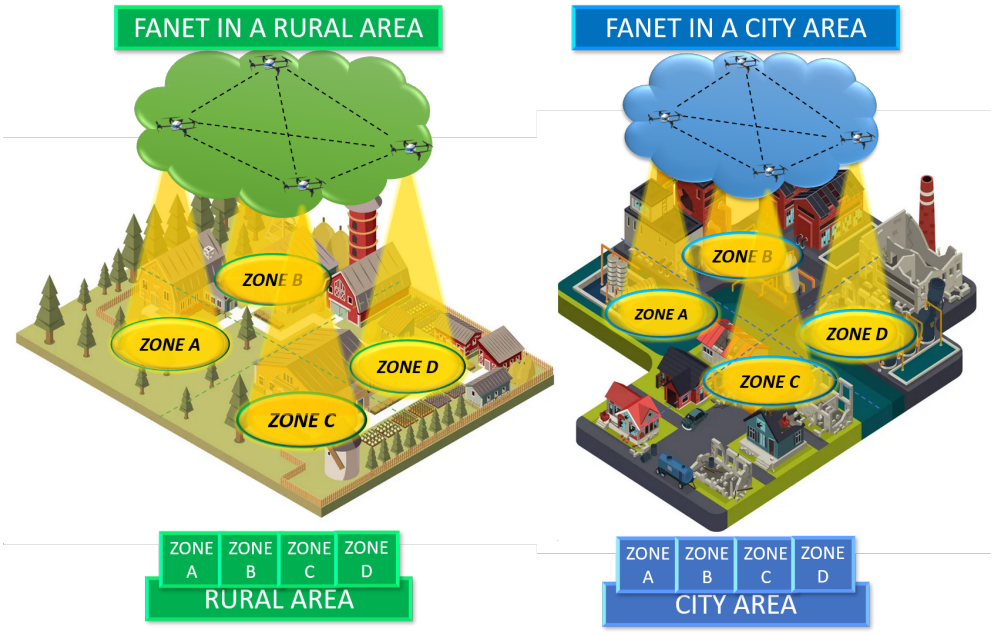


Figure 2.33: Reference System

Instead, it would be more efficient if each FANET could benefit from the knowledge acquired by other FANETs in order to: 1) react faster to changes of the underlying activity conditions, especially when these are similar to other dynamics already experienced in the past, even in different geographical areas covered by a different FANET; 2) quickly adapt to new underlying activity conditions of different geographical areas. Indeed, the same FANET can carry out its missions in areas with substantial differences in terms of traffic conditions of the covered zones: consider, for example, a FANET, usually covering a rural area, that has to be deployed in a metropolitan area due to sudden destruction of the infrastructure network of the city.

In order to perform insightful analytics of several underlying activity conditions, Centralized Learning (CL) algorithms could be applied. CL algorithms were recently successfully used in the context of intelligent end-to-end management and orchestration of network resources. CL requires the aggregation of operational data from various data sources belonging to single or multiple domains. In the context of this research, multi-domain can either mean multiple FANETs managed by a single Mobile Network Operator (MNO) or multiple FANETs (e.g., dealing with inter-operator Service Level Agreements) managed by several MNOs. Either way, in order to use the same approach with FANETs owned by single or different MNOs, a common solution is required.

In both contexts, aggregating the data in a central network entity able to execute

the CL algorithm causes several drawbacks due to regulatory restrictions, sharing of sensitive data by the MNO, high bandwidth resources required to transfer raw data, and increased risk associated with a single point of failure. Specifically, in a multi-domain ecosystem, it is crucial to ensure isolation among domains with the purpose of ensuring sensitive data secrecy when one domain possesses access to data of other domains.

On the other hand, Federated Learning (FL) [107] is a distributed machine learning approach where data from multiple domains can be processed and analyzed in a distributed manner in order to obtain the sharing of knowledge. This way, each federated entity can benefit from the acquired knowledge of the other ones. FL algorithm exchanges model updates among the participants. This entails two main advantages over CL: (i) privacy among the participating domains and (ii) significant reduction of the occupied network bandwidth, an aspect that has great importance in scenarios where communications between different FANETs are very hard. Both advantages occur because only the model updates are sent to the centralized aggregator during the training process rather than the stream of raw data like in CL.

2.5.1 System Description

The training phase of the DRL model is heavily affected by the dynamics of the area covered by the FANET. If a FANET will be involved in another mission or if the same area covered by a FANET changes its activity behavior abruptly for an unexpected event, the neural network trained on the previous behavior has to be re-trained. To this purpose we propose a two-layer Hierarchical Horizontal-Offload ManagEment (H-HOME) framework based on a FRL approach, which leverages experience obtained by all the FANETs during their missions, in order to obtain a federated model working in any new situation.

H-HOME is a two-layer framework for a scalable and efficient management of horizontal job offload performed by each FANET. It is based on a FRL approach to leverage experience obtained by all the FANETs during their missions to obtain a federated model that is suitable to any new situation. The H-HOME architecture is represented in Fig. 2.34, and is made up of a Job Processing Layer and an Orchestration Layer.

The Job Processing Layer of the H-HOME architecture is constituted by a number M of FANETs that provide edge-computing service to their geographical areas. As already described in previous sections, each area is subdivided into zones, each assigned to one UAV of the FANET. Therefore, each zone generates a

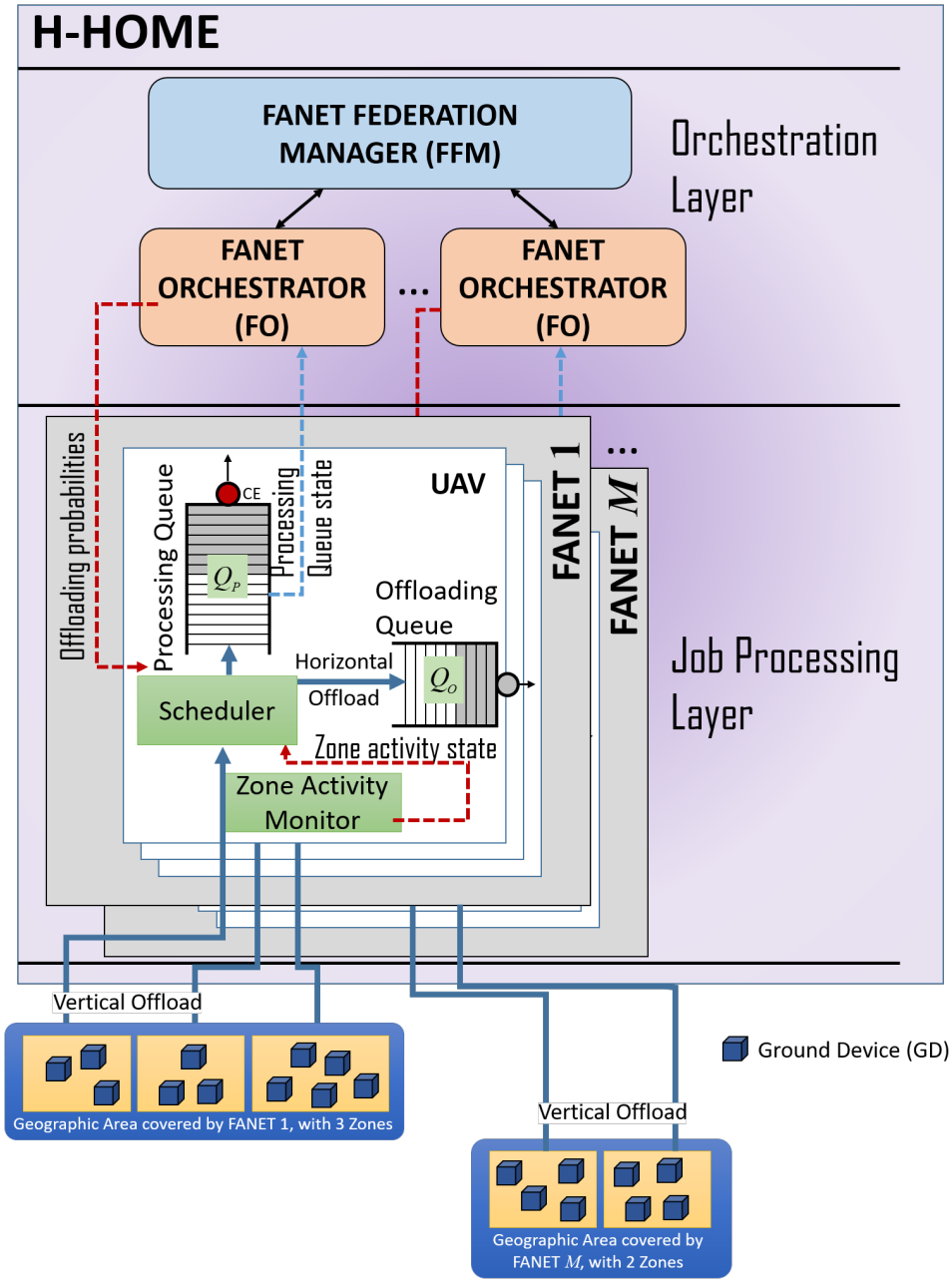


Figure 2.34: H-HOME Framework Architecture

job offloading flow directed to the UAV associated to it. We assume that all GDs are homogeneous in terms of both requirements and priorities. As already said so far, we introduce horizontal offload among UAVs to balance the load, offloading jobs to less-loaded UAVs. The decision whether processing a job locally in the Dwell UAV that has received it from the ground, or offloading it to another UAV, here referred as Processing UAV, is in charge of the Scheduler, as depicted in Fig. 2.34. In case of offload decision for a job, the Scheduler has also to decide the Processing UAV for it.

Jobs to be processed locally are enqueued in the *Processing Queue* Q_P waiting to be served by the local CE. On the other hand, jobs to be offloaded to other UAVs are enqueued in the *Offloading Queue* Q_O that is served by the wireless transmission link towards the other UAVs.

The H-HOME Orchestration layer is represented in Fig. 2.35. According to a Federated Learning approach, it is constituted by the FO of all the federated FANETs and a FANET Federation Manager. The FOs train their DRL models locally using the federated model received by the FANET Federation Manager as their starting point and then exploiting the data received by each UAV of their FANET. On the other hand, the FANET Federation Manager trains a federated model to be periodically sent to the FOs of the federated FANETs. The behavior of these entities will be detailed in the following subsections.

2.5.2 FANET Federation Manager

The *FANET Federation Manager* (FFM) is a centralized entity whose objective is to periodically compute a new federated model by executing a gradient descent step with the average of the gradients received by the FO of each FANET, as sketched in Fig. 2.35. To this purpose, the FFM can be run, for example, in a low-orbit satellite, a high altitude long endurance (HALE) UAV, a UAV Balloon, etc.

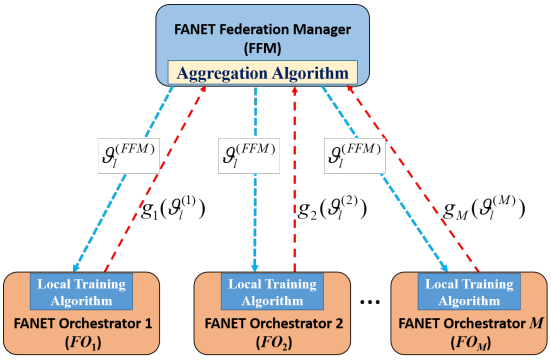


Figure 2.35: Interaction between the FANET Federated Manager and the FANET Orchestrators

In more detail, we consider M FANETs working in parallel in M different geographic areas.

This means that there are M local FOs, indicated as $FO_1, FO_2, ..., FO_M$ in Fig. 2.35. They perform a DRL algorithm, each one in its area, and share their parameters with the FFM. The FFM, for its part, collects information received

by each FO and updates the global model parameters via Federated Stochastic Gradient Descent (FedSGD).

The FO of each FANET and the FFM work at two different timescales, as shown in Fig. 2.36. The behavior of the FFM evolves according to the so-called *FFM episodes*, each with a duration equal to T. At the beginning of the generic episode l, in the FO timescale, during the first short interval, with duration τ_C, the FFM sends $\theta_l^{(FFM)}$, i.e. the parameters of the federated model, to all the FOs that, at the end of the previous episode, have sent their parameters to the FFM. Then, in the timescale of the FOs, each FO starts its *FO Training Phase* that is constituted by U consecutive epochs, each of duration τ, as described in Section 2.2. In the last short interval of the episode l, with time span τ_C, the FFM waits for gradients $g_m(\theta_l^{(m)})$ of the model trained by each FO_m, for each $m \in [1, M]$, during the episode l.

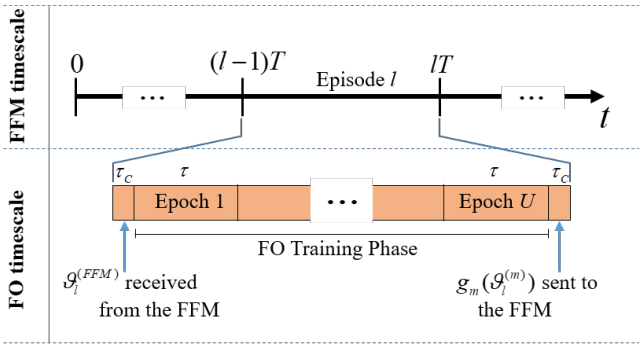


Figure 2.36: Timescales of FFM and FO inside H-HOME framework

Algorithm 2 shows the pseudocode of the proposed H-HOME framework to manage the generic episode l. In more detail, at the beginning of the episode, the FFM starts from the model $\theta_l^{(FFM)}$ calculated at the end of the previous episode and broadcasts it to all the FOs (Line 1), as shown in Fig. 2.35. In the first episode, the FFM broadcasts a pre-trained model, if available, otherwise a new federated model will be initialized.

After that (Lines 3-7), each FO simultaneously receives the federated model in the first short interval of duration τ_C, and runs the Algorithm 2 for all the U epochs. In this way, each FO updates the federated model received by the FFM and, after the end of the last FO epoch, sends the gradient $g_m(\theta_l^{(m)})$ to the FFM. The gradient equation is dependent on the algorithm used by each local in the framework, which will be described in detail in Section 2.2.

Finally (Lines 9-10), the FFM calculates the new gradient as follows:

$$\Gamma = \frac{1}{M} \sum_{m=1}^{M} g_m(\theta_l^{(m)}). \tag{2.41}$$

This allows the FFM to derive the new model $\theta_{l+1}^{(FFM)}$ following a Stochastic Gradient Descent (SGD) approach:

$$\theta_{l+1}^{(FFM)} = \theta_l^{(FFM)} - \eta \cdot \Gamma, \tag{2.42}$$

where η is the FFM model learning rate.

The complexity of Algorithm 2 is $O(M+M\cdot U\cdot O_{A2}+M+1+1) \approx O(M\cdot U\cdot O_{A2})$, where M is the number of UAVs, U is the number of epochs inside an FFM episode, and O_{A2} is the complexity of the Algorithm 3, which will be described in the next section.

Algorithm 2 FRL-Based H-HOME Episode Management

Input: Initial model $\theta_l^{(FFM)}$
Output: New model $\theta_{l+1}^{(FFM)}$
Data: $g_m(\theta_l^{(m)})$, $\forall m \in [1, M]$

1: FFM sends $\theta_l^{(FFM)}$ to the FO_m, $\forall m \in [1, M]$;
2: *#region Parallel Tasks executed by* $\forall FO_m, m \in [1, M]$
3: *In the first short interval:* $\theta_l^{(m)} \leftarrow \theta_l^{(FFM)}$;
4: ***for*** *epoch* $n \in [1, U]$ ***do***
5: *Training according to* ***Algorithm 2;***
6: ***end for***
7: *In the last short interval,* FO_m *sends* $g_m(\theta_l^{(m)})$ *to FFM*
8: *#endregion*
9: *FFM calculates the new gradient* Γ *as in (2.41)*
10: *FFM calculates the new model* $\theta_{l+1}^{(FFM)}$ *as in (2.42)*
11: ***return*** $\theta_{l+1}^{(FFM)}$

2.5.3 FANET Orchestrator

As already described in the previous chapters, the *FANET Orchestrator* (FO) of each FANET has the objective of deciding the horizontal offloading probabilities to be broadcasted to all the Schedulers in the FANET.

In our framework, in order to provide more transition samples for the whole training, an experience buffer D_m has been introduced into each local model inside the FO of each FANET m, indicated as FO_m, for $m \in [1, M]$.

Any DRL algorithm can be used in this framework to support the FO operations. One step of the local training of a generic DRL algorithm is summarized in Algorithm 3. First the experience buffer D_m is instantiated (lines 1-3). For each epoch, we distinguish between two different phases, which are the start (lines 4-5) and the end (lines 7-8) of the decision epoch. At the start of the decision epoch, the observes the state and then executes an action based on the current policy. Line 6 considers the environment evolving from one state to another one until the next decision epoch is triggered. At the end of the epoch, as expressed in lines 7-8, the of FO_m receives the reward from the environment, and observes the next state. The experience is then stored in the experience buffer D_m (line 9). Every U epochs, the FO sends the gradient to the FMM, calculated as follows:

$$g_m(\theta) = \frac{1}{|D_m|} \sum_{d \in D_m} \nabla f(\theta, d), \tag{2.43}$$

where $\nabla f(\theta, d)$ depends on the DRL algorithm chosen to solve the MDP. Details on how to calculate $\nabla f(\theta, d)$ for the most popular DRL algorithm, used in the simulation campaign to support the FO decisions, are depicted in Appendix A.

Accordingly, the complexity of Algorithm 3 is $O(M + K\ + 1 + 1 + 1 + 1 + 1) \approx O(K)$, where K is the number of operations executed in the forward pass of the DRL neural network, and is much greater than M, that is the number of UAVs in the FANET. Therefore, we can rewrite the complexity of Algorithm 2 as $O(M \cdot U \cdot O_{A2}) \approx O(M \cdot U \cdot K)$

Algorithm 3 Local Intra-FANET Online Training via DRL

Input: Initial model $\theta_l^{(m)}$
Output: Gradient $g_m(\theta_l^{(m)})$
1: **if** $n == 1$ **then**
2: Initialize experience buffer D_m
3: **end if**
4: Observe state $\underline{s}_n$ as in (**??**)
5: Generate and execute action a_n from $\pi_\theta(\underline{s})$
6: Let the environment evolve
7: Calculate reward r_n as in (2.11)
8: Observe state $\underline{s}_{n+1}$
9: Store $\underline{s}_n$, a_n, r_n in D_m
10: **if** $n == U$ **then**
11: Calculate $g_m(\theta_l)$ as in (2.43)
12: Discard the experience in D_m
13: **end if**

2.5.4 A Use Case for Performance Evaluation

In this section, we establish and conduct extensive simulations to evaluate the performance of the H-HOME framework proposed in this research. First, we describe the simulation setup, and then present our evaluation metrics. Finally, we illustrate the results of performance comparison between H-HOME and other approaches.

We consider three different FANETs, each with $N = 6$ UAVs, that provide on-demand MEC services in three different geographic areas. To characterize these geographic areas, we collected measurements in three different scenarios deployed at both the University of Catania and University of Messina campuses, in which different GDs required different services. By processing the measured job generation traces, we identified for each scenario two main activity states, hereinafter referred to as low-activity (L) and high-activity (H), i.e. the set of activities is $\mathfrak{S} = \{\sigma_L, \sigma_H\}$. The transition rate matrix and the job-arrival rate array calculated through this data analysis for each zone as the solution of an inverse eigenvalue problem [104, 108] from the traces for the first geographic area are:

$$\mathbf{Q}^{(A_1)} = \begin{bmatrix} -5.5 \cdot 10^{-3} & 5.5 \cdot 10^{-3} \\ 1.6 \cdot 10^{-2} & -1.6 \cdot 10^{-2} \end{bmatrix}$$

$$\underline{\Lambda}^{(A_1)} = [0.5,\ 4]\,\text{kjob/s}$$

The transition rate matrix and the job-arrival rate array for the second geographic area are:

$$\mathbf{Q}^{(A_2)} = \begin{bmatrix} -1.1 \cdot 10^{-2} & 1.1 \cdot 10^{-2} \\ 3.3 \cdot 10^{-2} & -3.3 \cdot 10^{-2} \end{bmatrix}$$

$$\underline{\Lambda}^{(A_2)} = [1.3,\ 3.3]\,\text{kjob/s}$$

Finally, the transition rate matrix and the job-arrival rate array for the third geographic area are:

$$\mathbf{Q}^{(A_3)} = \begin{bmatrix} -1.6 \cdot 10^{-2} & 1.6 \cdot 10^{-2} \\ 5.1 \cdot 10^{-2} & -5.1 \cdot 10^{-2} \end{bmatrix}$$

$$\underline{\Lambda}^{(A_3)} = [1.5,\ 2]\,\text{kjob/s}$$

We have carried out two different analyses. In the first analysis, we study the effectiveness of the proposed H-HOME federated training approach against the traditional local training approach. The DRL algorithm used to support the FO decisions is the A2C algorithm. Each UAV in the FANET has a CPU processing

rate $\overline{\mu}_P = 2.15$ kjob/s and a job transmission rate $\overline{\mu}_O = 3.2$ kjob/s. Let $K_P = K_O = 250$ jobs be the size of the Processing and Offloading Queues inside each UAV. We deploy three FANETs to cover the above three different geographic areas. Each FO has locally trained its model by interacting with the environment for 10^5 epochs. We therefore refer to as LT-SCx s every time a FO has been trained via local train in the geographic area x, with $x \in \{1, 2, 3\}$.

Afterwards, we deploy the H-HOME framework, in which FANETs collaborate with each other by joining a federation, orchestrated by the FFM, and cooperatively train a new federated model, for 10^3 federated episodes, each with $U = 10$ epochs.

The epoch duration τ is set to 0.1 s. We considered that all the UAVs are equipped with an INTEL NUC 10 Barebone Core i7 with a clock frequency of 2.8 GHz. The neural network forward pass requires $3.5 \cdot 10^{-4}$ Floating Point Operations (FLOP). Therefore, the time required for the forward pass is in the order of 10^{-5}s, which is far smaller than the epoch duration $\tau = 0.1$s, and this ensures that all local computation is completed within the epoch. On the other hand, each model is saved as a *.pth* file, and uploaded to the FFM every U epochs. The size of this file is 25 KB. Therefore, considering a 1 MB/s transmission rate, the time required for uploading and downloading the model is equal to 25 ms. Finally, the minimum time slot T will then have to be equal to $T_{download} + T_{upload} + (\tau \cdot U - 1) = 25 + 25 + (100 \cdot 9)$ ms $= 0.95$ s

We then compare these two different approaches by deploying four FANETs, i.e. three FANETs with the locally-trained LT-SCx model and one FANET using the federated H-HOME framework model, in a new evaluation scenario, where the activity state of the geographic area is described by the following transition rate matrix and job-arrival rate array:

$$\mathbf{Q}^{(A_4)} = \begin{bmatrix} -5.5 \cdot 10^{-3} & 5.5 \cdot 10^{-3} \\ 1.25 \cdot 10^{-2} & -1.25 \cdot 10^{-2} \end{bmatrix}$$

$$\underline{\Lambda}^{(A_4)} = [0.8,\ 2.5]\,\text{kjob/s}$$

During deployment we allow each model to be re-trained using the newly collected experience, and we compare how much experience each model requires before reaching the optimal performance in the new scenario. In this analysis we will then compare the performances against another algorithm, *LOCAL*, that represents the agent that has been specifically trained only in the evaluation scenario, and is supposed to be the best agent in terms of performance.

In the second analysis, we analyze the performance of the H-HOME framework in the scenario characterized by the matrices $\mathbf{Q}^{(A_4)}$ and $\underline{\Lambda}^{(A_4)}$, taken as reference, against the CPU processing rate of each UAV increasing in the range $[1.95, 2.35]$ kjob/s. We compare it with the performance of the LT-SCx and LOCAL FANETs. In contrast to the first analysis, the layers in each FO neural network are frozen. This allows us to analyze the generalization capabilities of the H-HOME approach against the LT-SCx. Even in this case, all UAVs have a total job transmission rate $\overline{\mu}_O = 3.2$ kjob/s, and let $K_P = K_O = 250$ jobs be the size of the Processing and Offloading Queues inside each UAV.

The experiments were performed via a simulator based on OpenAI Gym [95] and Pytorch [96]. We used the A2C algorithm, and implemented both Actor and Critic networks with two fully connected layers, each with 128 neurons. As opposed to a value-based algorithm, such as DQN, where some policy has to be defined and applied to balance exploration and exploitation (such as the ϵ-greedy policy), the A2C has a stochastic policy function. This means that, by default, it will occasionally explore all the actions since, at every decision epoch, each action will have a non-zero probability to be executed; hence, there is no need for other additional techniques to enable exploring. We also used the Adam optimizer with a learning rate $\eta = 10^{-4}$ and set a discount factor $\gamma = 0.9$.

For performance evaluation, we consider three evaluation metrics, i.e., loss function, average total delay (2.12) and delay jitter (2.13).

The *loss function* is used to indicate the convergence performance of each model. The smaller the loss function is, the better the estimation performance the model will have.

2.5.5 Simulation Results

In this section, we evaluate the performance of the proposed H-HOME framework through two different simulation analyses.

In the first analysis, we show the effectiveness of the proposed Federated Learning training approach against the traditional local training approach by comparing the training speedup obtained using the H-HOME approach against three LT-SCx FANETs. We therefore show in Fig. 2.37 the loss function of the H-HOME model, the three LT-SCx models and the LOCAL model, in the reference scenario, which is the one characterized by $\mathbf{Q}^{(A_4)}$ and $\underline{\Lambda}^{(A_4)}$. This figure shows that the loss function of H-HOME decreases closely to 0 after 120 FFM episodes. This convergence indicates that this model has rapidly learned the hidden rules for evaluating actions. In contrast, the first of the other models to converge is

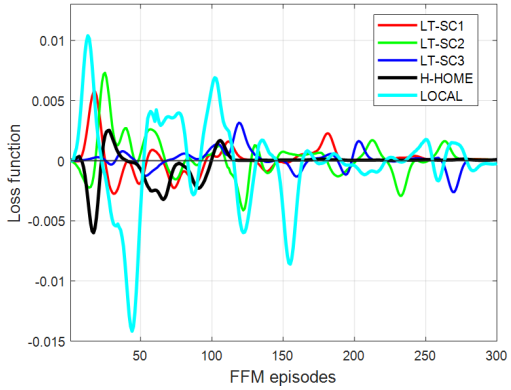


Figure 2.37: Loss Function Convergence

the LT-SC1 after other 150 FFM episodes, which means that the other models, compared to the H-HOME framework, require as much as double experience. The LOCAL model is the one, among all the models, that is having the most difficulty to converge: this is due to the fact that it has not been previously trained on another scenario, and therefore has just started collecting its first MDP experiences. We can conclude that H-HOME is more robust to sudden changes of the job arrival rate with respect to the other algorithms.

In the following we will refer to a FANET mission as a RL episode, or simply episode. The length of the FANET mission depends on the flight autonomy of each UAV. Fig. 2.38 shows the cumulative reward at the end of each episode, calculated as in (2.1).

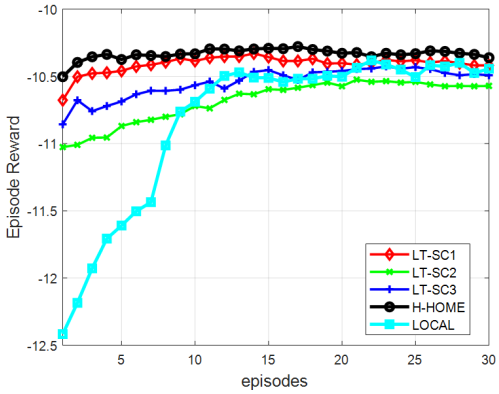


Figure 2.38: Episode Reward

We can notice how the H-HOME is able to obtain the highest cumulative reward among all the s. Since the reward has been shaped to lower both the average delay and the jitter, we expect the H-HOME framework to achieve the lowest average delay and jitter. Fig. 2.39 illustrates the average delay in the reference scenario. It

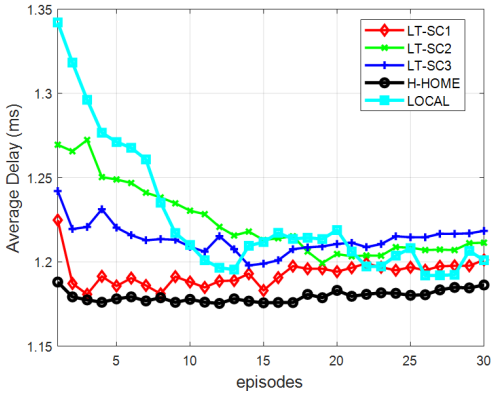


Figure 2.39: Average Delay

is clear to see that the H-HOME framework, by having a faster convergence speed, quickly achieves the lowest average delay among all the models. This also means that the learning speed is adequate. Fig. 2.40 shows that H-HOME achieves the lowest jitter in most episodes as desired, meaning that the federated model has been trained to well utilize the limited resources of UAVs and avoiding computational waste by offloading the right amount of jobs among the UAVs.

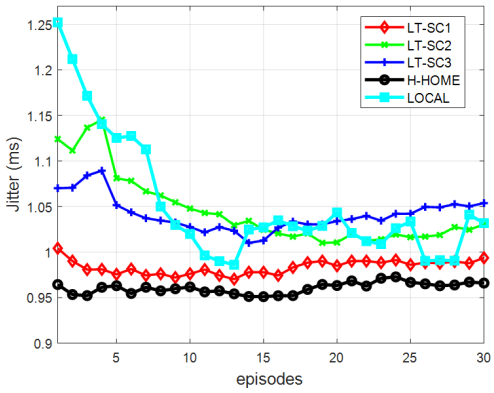


Figure 2.40: Jitter

In the second analysis, we exhibit the performance of the H-HOME as the CPU processing rate inside each UAV increases in the interval $[1.75, 2.55]$ kjob/s in terms of average total delay, jitter and flight autonomy. Each LT-SCx model is trained in its own geographical area, whereas the H-HOME model is trained in a federated fashion.In contrast to the first analysis, we do not allow each model to be re-trained in the reference evaluation scenario. Instead, we evaluate each model performance by freezing all the layers in the network. This allows us to analyze the generalization of each model. Contrary to the LT-SCx and H-HOME models, the LOCAL model has been trained in the evaluation scenario, but with

a restricted number of iterations, equal to 30 episodes.

As expected, as the available processing rate for each slice increases, the overall performance of each FANET improves, as shown in Fig. 2.41. However, the

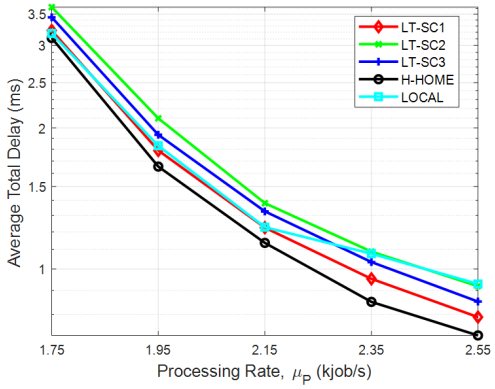


Figure 2.41: Average Total Delay vs Processing Rate

FANET that reaches the best values is the one with the H-HOME model: this is due to the fact that, since it has been trained on different scenarios via federated learning, it is able to generalize better in new scenarios.

Fig. 2.42 shows the measured delay jitter. As we can see in the figure, H-HOME

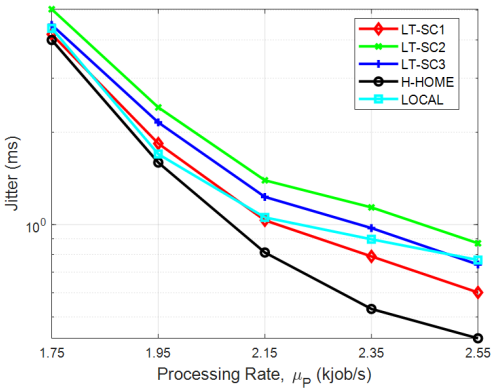


Figure 2.42: Jitter vs. Processing Rate

is not only the one that achieves the best performance in terms of average delay, but also the model that presents the lowest jitter for considered each processing rate. Moreover, let us note that, thanks to the FANET federation approach, the delay perceived by the jobs at a given time is almost independent of the GD job generation rate where the jobs are generated, that is, from the activity of the zone.

The CPU computation power strongly affects the power consumption, consequently the maximum time of each UAV battery charge lifetime, and therefore

the maximum duration of the FANET mission. For this reason, in Fig. 2.43 we show the FANET flight autonomy as a function of the CPU computation power of the UAV CE. We assumed that the FANET is realized with quadcopter UAVs

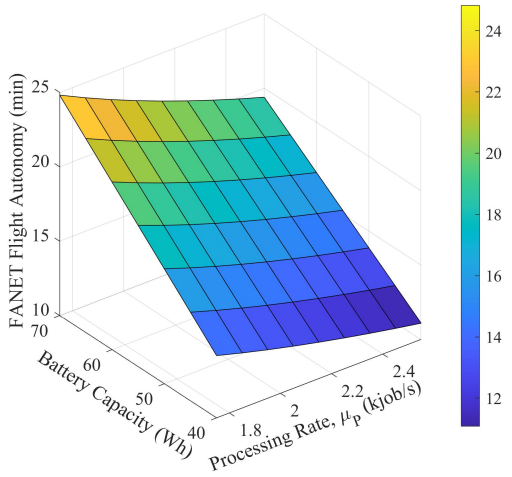


Figure 2.43: FANET Flight Autonomy

with an engine power consumption $\mathcal{P}^{(EN)} = 66$ W, and we consider different Lithium batteries with capacity β_c increasing in the range $[40, 70]$ Wh. The total power consumption of each UAV, $\mathcal{P}^{(TOT)}$, is calculated as the sum of $\mathcal{P}^{(EN)}$ and the power consumption of the CE, $\mathcal{P}^{(CE)}$, already derived in 2.33 From measurements on a real testbed, we have used $\overline{\omega} = 59.1$ mJ. The FANET flight autonomy depends on $\mathcal{P}^{(TOT)}$ and the battery capacity, β_c, and has already been derived in : As expected, the FANET flight autonomy increases as the battery capacity increases and the CPU processing rate decreases. From a joint analysis of Figs. 2.41, 2.42 and 2.43, the system designer can decide the CPU computation power that maximizes profits for the FANET owner and the battery capacity that, given the CPU computation power, allows missions with a given time duration.

As an example, Fig. 2.44 combines the average total delay with the maximum duration of the FANET mission, calculated as in (2.5.5), to analyze how the maximum duration of the FANET mission is related to the average total delay suffered in the FANET, with $\beta_c = 60$ Wh. Choosing a point in this figure means deciding the framework to use and a specific CE computation rate, $\overline{\mu}_P$. We can observe that our proposed H-HOME framework maximizes the FANET mission duration for a given average total delay requirement. Finally, we show in Fig. 2.45 that the H-HOME framework manages to reduce the FANET energy consumption by achieving lower average delay with less CE power consumption compared to the other models.

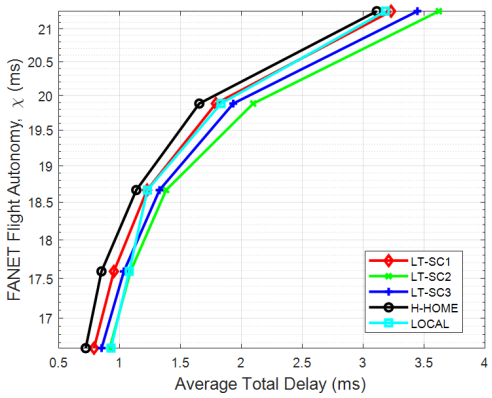


Figure 2.44: FANET Flight Autonomy vs Average Total Delay

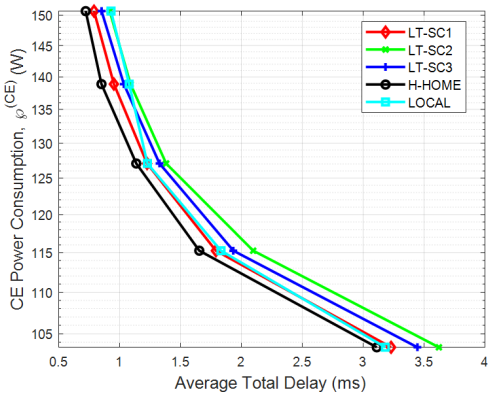


Figure 2.45: Power Consumption

2.6 A DRL-based UAV-Smallcell System for Efficiently Localizing Hidden Mobile Devices via RSRP Measurements

The last decades years have been characterized by various catastrophic events such as earthquakes, floods, etc. In these scenarios, civil protection interventions have to be fast and precise. This can be enabled by improving the current intervention techniques by exploiting the latest technological innovations. In the last few years, UAVs have been increasingly used in various scenarios, especially when communications infrastructure is not available [42, 50, 109–113].

For this reason, several studies have focused on devising techniques to improve drone performance while minimizing energy consumption. UAVs are often used for operations in unknown or partially observable environments. For this reason, flight path planning is an essential issue in the use of UAVs as it is directly related to the level of autonomy of the vehicle.

Since no exact mathematical model is available, DL and RL are combined and used to allow UAVs to learn their paths autonomously, allowing them to traverse changing environments without the risk of collision [45–48].

In [71], an RL algorithm is introduced that enables UAVs to have direct and continuous interaction with their surroundings. In particular, a combination of DRL and a Long Short-Term Memory (LSTM) network is proposed to increase the speed of the used learning algorithm. In addition, the authors in [72] propose the RL algorithm to circumvent obstacles with a reward function and a penalty action to have a smoother trajectory. In [73], several RL algorithms are used to improve UAV navigation. Moreover, UAVs can provide wireless connectivity without network infrastructure or complement conventional base stations (BSs), whose coverage may suffer from severe blockage due to tall buildings or damages caused by natural disasters. Owing to the mobility of UAVs, recent years have seen significant research progress on integrating UAVs with MEC [74].

In this work, we consider a case study where several missing people have to be localized in the shortest amount of time possible in earthquake-affected areas. In particular, it is assumed that the missing persons have a mobile device switched on, and the terminal's location can be estimated using UAV-Smallcell systems, i.e., a low-range low-powered base station installed on a flying UAV. This approach has recently been proposed in [114–116]. In these studies, several localization techniques and algorithms have been applied to locate mobile devices, such as the "Cluster-based Fast Proximity Algorithm" in [116] and a cooperative localization technique in [114] in which two UAVs leverage Game Theory (GT) techniques.

In this research, we design a DRL-based UAV-Smallcell system for localizing hidden mobile devices using RSRP measurements and propose RADAR, a Reinforcement leArning based mobile Device locAlization approach via RSRP measurements algorithm that can quickly and efficiently localize hidden mobile devices in large areas. RADAR also exploits Transfer Learning (TL) techniques to leverage the knowledge acquired in simpler missions to solve complex, challenging environments.

The rest of this chapter is structured as follows: Section 2.6.1 discusses the synergy between 6G technology and drones. In Section 2.6.2, the main principles of free-space radio signal propagation are reviewed and how the power received by mobile devices is calculated in scenarios where there is isotropic attenuation in free space. The proposed architecture is described in Section 2.6.3, where the UAV-Smallcell System and Markov Decision Process (MDP) are illustrated. The

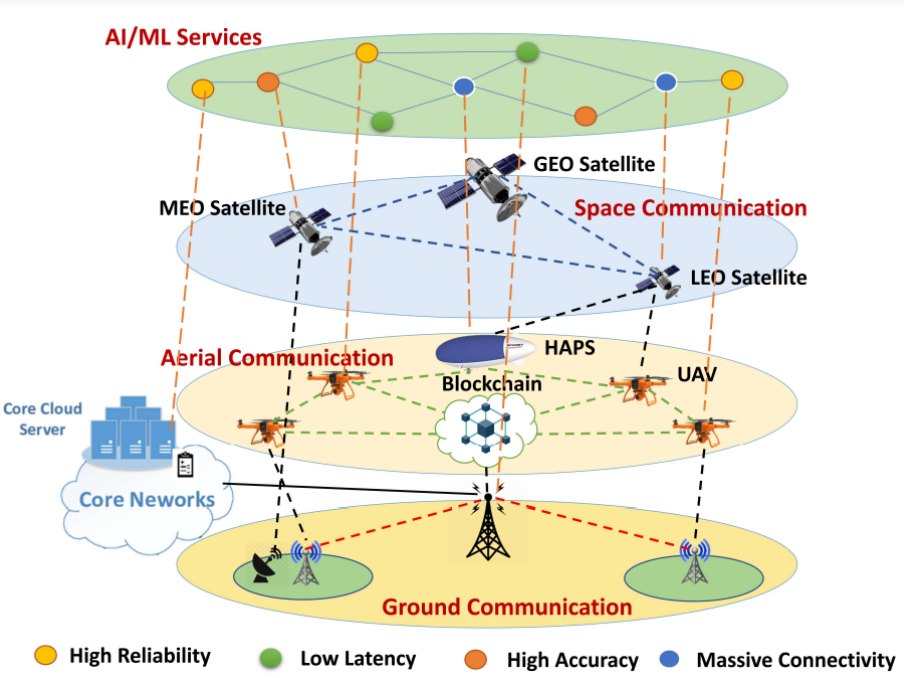


Figure 2.46: Architecture for 6G-enabled UAV Network [75].

system setup is described in Section 2.6.6. Finally, in Section 2.6.7, we evaluate the performance of the proposed system through an extensive simulation campaign and demonstrate that our approach can significantly improve the effectiveness of mobile device localization compared to other state-of-the-art approaches.

2.6.1 Exploring the Synergy: Integrating Drones with 6G Networks

6G offers a range of advantages compared to previous generations of mobile networks, such as ultra-high data density, high-speed and low-latency communications, and ubiquitous ultra-wideband mobile connectivity [117]. In the context of UAV networks, 6G enables drones to interconnect and communicate with the fixed infrastructure. With cloud computing capabilities enabled by 6G technology, drones can perform complex tasks, overcoming onboard resource limitations. The integration of satellites in 6G enables global coverage and precise positioning, while advanced antenna techniques address challenges related to UAV's three-dimensional mobility.

6G also provides additional positioning services through beamforming and triangulation. Overall, 6G offers significant opportunities for UAV networks, improving their efficiency and enabling the introduction of new revolutionary applications and services [75] (see Figure 2.46).

In terms of applications, drones can be divided into three main categories: UAV as Base Station (BS), UAV as relay, and UAV as data collector/disseminator [79]. In the first category, the drone serves as a communication infrastructure, allowing ground nodes to communicate and connect to 5G/6G core networks through the onboard BS module. The fleet of drones can self-organize to form a network and provide seamless coverage. In case of terrestrial BS failures, the UAV can act as a BS and quickly restore service after infrastructure failure. Some common applications in this category include emergency support, temporary coverage for terrestrial users, and high-density hotspot applications.

UAV as a relay utilizes a UAV with a radio access node that connects to the terrestrial BS and the main 5G/6G network. The UAV acts as a relay to extend communication infrastructure and can connect to nearby UAVs to expand its coverage.

Lastly, UAVs as data collector/disseminator is used for data collection and dissemination in remote locations where human presence is challenging. These drones enable wireless sensors to gather and transmit data, providing an inexpensive and easily deployable solution for data transfer. This application is particularly useful for periodic sensing and information multicasting for ground sensors and vehicles, as well as data collection operations in hostile and difficult terrains.

Among all the discussed application contexts, this study focuses on implementing an algorithm based on Unmanned Vehicles Systems (UVSs), RL, and localization algorithms to identify the position of devices under rubble in earthquake scenarios.

2.6.2 Isotropic Signal Propagation

Electromagnetic signal propagation can be hindered and/or limited by several factors. There are different types of signal propagation:

1. Signal propagation in a vacuum;
2. Signal propagation within the Earth's atmosphere;
3. Signal propagation within the Earth's atmosphere and in the presence of materials.

In this study, the first case is considered in which the attenuation of the signal (A_0) occurs due to empty space and is expressed via the formula in (2.44):

$$A_0 = 20 \cdot \log(d) + 20 \cdot \log(f) + 92.45, \tag{2.44}$$

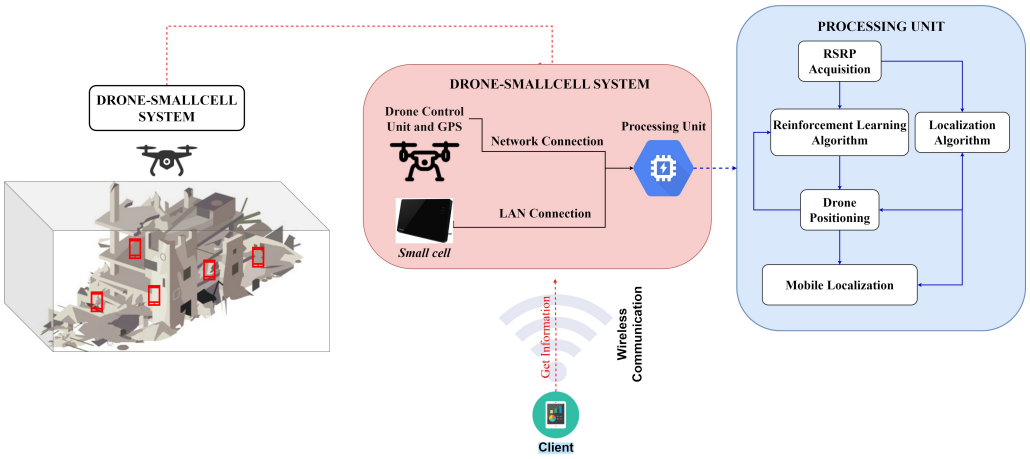


Figure 2.47: UAV-Smallcell System: Connection/communication scheme.

where d is the distance between the UAV-Smallcell system and the mobile device or, more generally, the distance between the transmitting and receiving antenna, expressed in kilometers, while f is the carrier frequency expressed in GHz. Thus, the transmitting power of the signal is expressed by the formula in (2.45) (in dB), and referred to as the "Friis Transmission equation":

$$P_R = P_T + G_T + G_R - A_0, \tag{2.45}$$

where P_T [dBm] is the transmission power of the Smallcell (or any transmitter); P_R [dBm] is the receiving power of the mobile device (or any type of receiver); G_T and G_R [dB] are the transmitting and receiving antenna gain, respectively.

In our system, vo P_T, G_T, G_R and A_0 are known values. In LTE systems, P_R is known as the RSRP value. A device is considered as localized when the measured RSRP reaches a predefined threshold.

2.6.3 Proposed Architecture

The system illustrated in Fig. 2.47 involves the use of one (or more) UAVs carrying a Smallcell and a processing unit (e.g., a Raspberry Pi). The RSRP values measured by each terminal are received by the Smallcell via a common control channel and sent to a processing unit using a point-to-point link. The UAV has to reach different positions within the monitoring area to make RSRP measurements. The processing unit is connected to the UAV's GPS coordinate system using an Ethernet cable. This way, it is possible to associate the position assumed by the UAV with the RSRP value that the Smallcell receives at that particular location. Within the processing unit, an RL algorithm is executed. The RL agent, com-

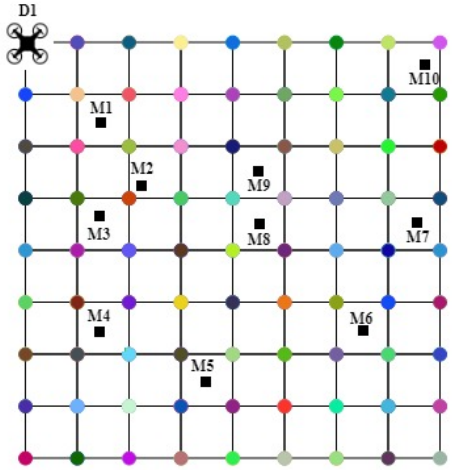


Figure 2.48: Example of the simulated scenario.

bined with a localization algorithm based on the proximity criterion, is able to localize the mobile devices in the shortest amount of frame.

The estimated location and total time required to locate all devices are then sent to a client via a wireless network obtained by the WiFi module installed into the processing unit, which also acts as an access point for the clients. An example of an area to be monitored is shown in Figure 2.48. We divide the area into several 3×3 meters smaller areas. This division results in a 9×9 grid where the UAV can move. Each time a UAV has to change its position, it will move from one center of a small area to the next one.

2.6.4 Markov Decision Process

In the RADAR system, there is a set of hidden mobile stations that needs to be located. In the UAV, an RL agent is deployed to choose, at runtime, the optimal movement to be executed by the UAV to track and localize all the devices. At the beginning of the mission, we consider the hidden device with the highest received RSRP as the *tracked* device, i.e., the device that the UAV tries to localize. If and when the tracked device is localized, the (old) tracked device will be considered to be fully localized, and the (newly) tracked device will be chosen between the yet-to-be-localized devices with the highest RSRP.

To optimize the movements, the RL agent learns the optimal policies using an RL approach. In RL, an MDP provides a framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker, which in our system is the RL agent. Formally, an MDP is a mathematical framework for modeling decision-making problems under

uncertainty. It consists of a set of states, actions, and a transition model that specifies the probabilistic transitions between states as a function of the current state and the chosen action. The MDP also includes a reward function that assigns a reward to each state-action pair.

A MDP is formally defined as a tuple (S, A, T, R, γ) in which S is the state space, A is the action space, T is the set of transition probabilities among states, r is the reward function, and γ is the discount factor.

In the context of a UAV localization system that finds hidden mobile devices by measuring their RSRP, we define the MDP as follows.

States: The state s_n at decision epoch n of the MDP, $s_n \in S$, represents the location of the UAV and the RSRP of the tracked device. Moreover, to increase the agent's performance, we stack the last k observations together. Each time a UAV has to take a new action, a decision epoch n is triggered. Therefore, at each decision epoch n, the most recent agent observation is the following:

$$o_n = [x_n^{(UAV)}, y_n^{(UAV)}, RSRP_n^{(DEV-TR)}] \tag{2.46}$$

where $x_n^{(UAV)}$ and $y_n^{(UAV)}$ are the coordinates of the UAV in the area, whereas $RSRP_n^{(DEV-TR)}$ is the RSRP of the tracked device. After stacking the observations, the new agent state will have the following shape:

$$s_n = [o_n, o_{n-1}, ..., o_{n-k-1}] \tag{2.47}$$

Actions: The action of the MDP $a_n \in A$ includes the UAV's movement commands, such as go left, go right, go up, go down. Therefore, the action space A is defined as $A = \{\text{left, right, up, down}\}$. After moving, the UAV measures the devices' RSRPs at its current location.

Transition model: The transition model T specifies the probability of transitioning from one state to another based on the current state and the chosen action. However, if, the UAV location at the next step may be easily modeled, the same cannot be said for the RSRP of the devices. This is why we adopt a model-free RL approach, in which the transition probabilities are unknown.

Reward function: The reward function r_n assigns a reward to each state-action pair based on the success of the UAV in finding the hidden mobile devices. Specifically, if the UAV measures an RSRP higher than a predefined threshold, the agent receives a positive reward, and the mobile device is considered localized; otherwise, the reward is zero. Therefore, we can write the reward the agent receives at decision epoch n as:

$$r_n = \begin{cases} 1 & \text{a device has been localized during decision epoch } n \\ 0 & \text{otherwise} \end{cases} \tag{2.48}$$

2.6.5 RADAR Transfer Learning

TL techniques are leveraged in the RADAR framework to transfer knowledge gained from solving previous missions and applying it to later ones. TL in RL refers to applying knowledge acquired in a source task, $T^{(S)}$, to a target task, $T^{(T)}$. In mathematical terms, this can be achieved by using the learned representation in $T^{(S)}$ to initialize the parameters of the policy or value function in $T^{(T)}$. In RADAR, this is accomplished by transferring the model and optimized parameters of both PPO Actor and Critic networks between consecutive missions. We can therefore use the knowledge learned from a previous mission, M^m, and transfer it to speed up the learning process in the next mission, M^{m+1}. This way, the agent can leverage the knowledge gained in the source mission to quickly adapt to the new mission, reducing the amount of data and training time. In the following sections, we refer to the RADAR agents whose models have been pre-trained on another mission as *RADAR-TL*. In contrast, we refer to the RADAR agents without a pre-trained model as *RADAR*.

2.6.6 Simulation Setup

We implemented the RADAR agent in Python using the stable-baselines3 library [106]. The UAVs have been modeled in a simulator written in Python to include their physical properties (such as size, weight, and propulsion system) and their sensor and communication capabilities (such as their range and accuracy for measuring RSRP). The simulator also includes a model of the environment in which the UAV operates, based on a customized version of minigrid [118]. The simulator also has a user interface that allows one to specify the parameters of the simulation (such as the starting location of the UAV, the locations of the hidden mobile devices, and any obstacles), monitor the progress of the simulation, and view the results. Randomness was incorporated into both agent behavior and the location of the devices. As previously mentioned, TL techniques have leveraged prior knowledge from previous UAV missions to help reduce the amount of training data and computational resources required to train the model and improve its overall accuracy and robustness.

In our simulation campaign, we consider a $48m \times 48m$ area, which results in

Table 2.3: Simulation Parameters.

Parameter	Value
Number devices to localize	[10, 15, ..., 30]
δ_f	10 s
δ_p	30 s
Matrix Size	16×16
Area size	$48m \times 48$ m
Cells size	$3m \times 3$ m
RSRP threshold	-70.5 dBm
Transmission antenna gain, G_T	2 dB
Receive antenna gain, G_R	1 dB
Carrier frequency, f	2.4 GHz
Smallcell transmission power, P_T	20 dBm

Table 2.4: PPO Parameters.

Parameter	Value
Optimizer	AdamOptimizer
Actor Learning rate, α_θ	$3 \cdot 10^{-4}$
Actor Hidden Layers	[128,128]
Critic Learning rate, α_w	$4 \cdot 10^{-3}$
Critic Hidden Layers	[128,128]
Batch size, b	64
Discount Factor, γ	0.99
Number of previous observations to stack, k	4

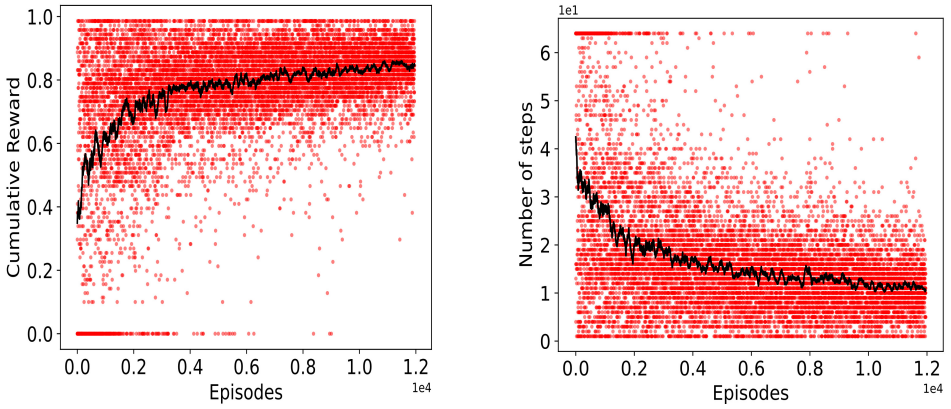
a 16×16 grid. The initial position of the drone is in the top left vertex of the monitoring area.

The main simulation parameters are summarized in Table 2.3, where the flight time, δ_f, is the time required by the drone to move along one side of a basic cell, while the processing time, δ_p, is the time required by the drone to acquire a stable power value.

The main DRL parameters are summarized in Table 2.4.

2.6.7 Numerical Results

In this section, we analyze the performance of the RADAR framework. It has been evaluated for different numbers of hidden devices to localize. We first compare



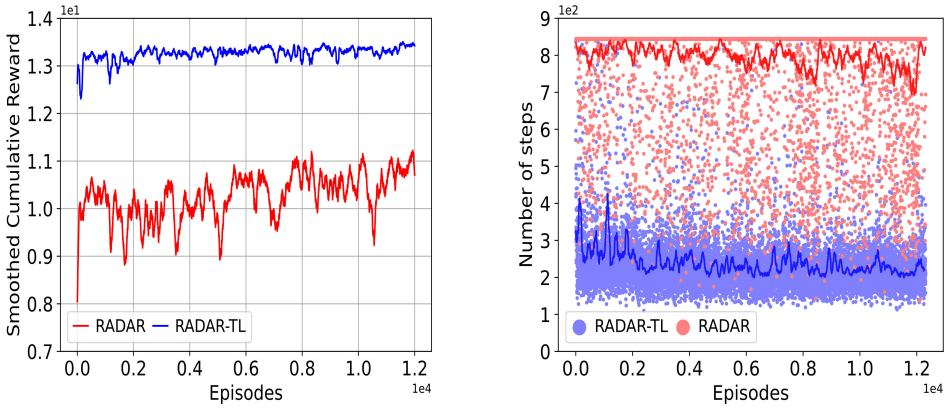
(a) (b)

Figure 2.49: RADAR agent training phase: (a) RADAR agent cumulative reward, (b) RADAR agent mission length.

the performance of the systems with and without leveraging TL. Specifically, we consider one pre-trained RADAR-TL model that has been trained in an area with one hidden device for 5 million steps. This agent is then deployed on the evaluation scenarios where 5 to 30 devices have to be localized. We refer to this agent as RADAR-TL. Whereas the agent that does not use any pre-trained model is referred to simply as RADAR. This agent is only trained on the evaluation scenario for 15 million steps.

First, we show in Fig. 2.49a and Fig. 2.49b that a RADAR agent can track and localize the devices by plotting its cumulative reward during the training phase in a scenario where one hidden device has to be localized as quickly as possible. Each red data point in the figure corresponds to the Cumulative Episode Reward for one mission, whereas the black curve is the smoothed Cumulative Episode Reward obtained with a Simple Moving Average with a window size equal to 100. This scenario also produces the neural network models whose knowledge is used to pre-train the RADAR-TL models in the consequent scenarios.

As illustrated in Fig. 3.2a and Fig. 2.50b, RADAR performance, represented by the smoothed cumulative reward curves, is substantially higher when leveraging the pre-trained RADAR-TL model that leverages TL than the RADAR agents that do not use it. Even if in the RADAR agent some learning seems to be occurring, its cumulative reward after 15 million is nowhere near the cumulative reward of RADAR-TL, as shown in Figure 3.2a. These performances are directly related to the amount of time required to complete a mission. A mission is considered completed once all the hidden devices have been localized. Figure 2.50b shows that the amount of time required to complete a mission, in terms of the



(a) (b)

Figure 2.50: RADAR agent training phase: (a) RADAR agents Cumulative Reward comparison, (b) RADAR agents mission length comparison.

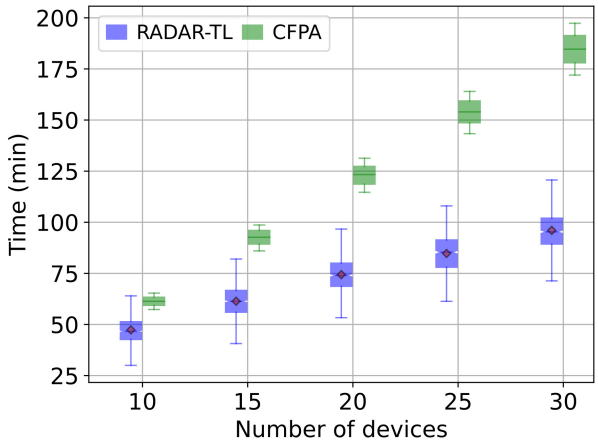


Figure 2.51: Performance comparison.

number of RL steps, is much lower for RADAR-TL than RADAR. Hence, TL can be considered mandatory for the proposed framework to learn to track and localize all the hidden devices.

Finally, in Figure 3.4, we show the performance comparison among RADAR-TL and the *Cluster-based Fast Proximity Algorithm* approach proposed in [116] in terms of minutes required to localize all the devices. The *Cluster-based Fast Proximity Algorithm* (CFPA) leverages the assumption that the mobile devices are not uniformly distributed within the monitoring area, and, therefore, there will be one section of the area with higher mobile devices density. By doing so, it iteratively chooses the best path to localize all the devices in the subsection of the area in the least amount of time.

The results show that CFPA performance linearly degrades as the number of hidden devices increases. On the other hand, RADAR-TL optimizes its move-

ments to localize all the devices quickly. Specifically, the gap between RADAR-TL and CFPA, in terms of the reducing the average time all the devices, ranges between 24% for ten devices to 52% for 30 devices.

2.7 Resource Planning in Drone-Based Softwarized Networks

As said in the previous sections, keeping a FANET available and active is an ongoing challenge as the autonomy of its UAVs is limited [41].

The minimization of the power consumption represents the main challenge to be addressed. The more resources the CE consumes, the faster the battery runs out, thus reducing the overall FANET service availability.

Specifically, when the battery charge of a UAV is below a certain threshold, the UAV must temporarily leave the FANET to reach the nearest charging station. During this period, the VFs that were running on that UAV need to be placed within the remaining UAVs, causing an increase of their energy consumption and the consequent reduction of the flight duration. If the number of flying UAVs is not sufficient, the FANET could not be able to deliver services to the ground devices.

In this research, we present an optimization framework capable of increasing the overall duration of the FANET. To do this, we act on two fronts: the first, in the long-term, is to optimize the percentage of available CPU resources for VF computing; the second, in the short-term, consists in optimizing the VF placement inside the active UAVs of the FANET.

The proposed optimization framework uses a DRL approach, based on Double Deep Q-Networks (DDQN) [119], with the goal of setting the amount of available CPU, and an Integer Linear Programming (ILP) algorithm to optimize the VF placement.

The rest of the research is organized as follows. Section 2.7.1 gives an overall description of the system, specifically of the considered scenario and the functional architecture of the main elements. Then, in Section 2.7.3, we introduce the proposed orchestration framework, describing the long-term FANET behavior optimization, which is based on RL, and the short-term VF placement optimization. Section 2.7.6 proposes a use case for performance evaluation. Finally, Section **??** draws conclusions.

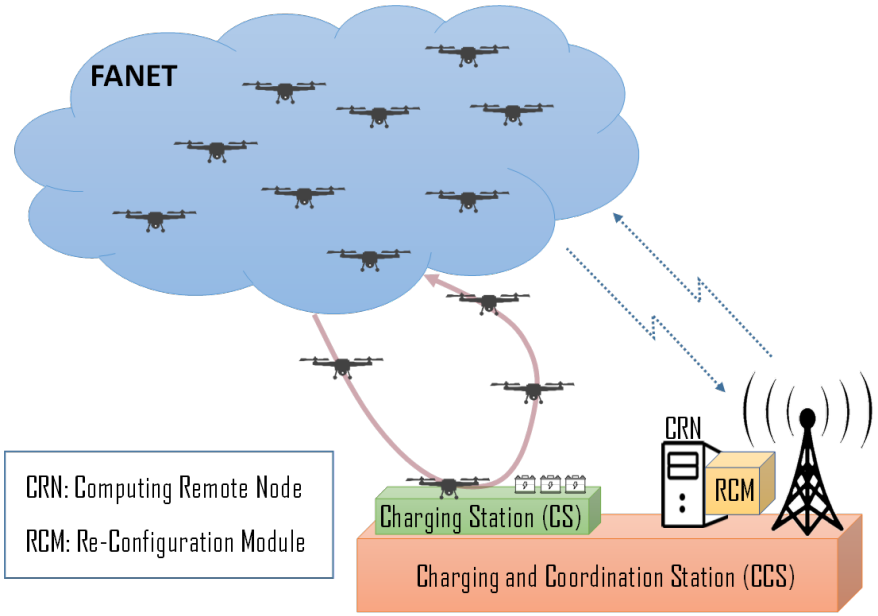


Figure 2.52: System Description

2.7.1 System Description

The system we consider in this research is depicted in Fig. 2.52. We consider a fleet of 5G-enabled UAVs that is used to create a FANET. The FANET is deployed in support to Application Services (ASs) that are dynamically requested by users in a given ground area, and typically spans up to few squared kilometers. More specifically, each UAV acts as an NFV Infrastructure Point-of-Presence (NFV-PoP), that is, as a *micro-datacenter* where both virtual network functions and virtual application functions (related to given ASs) are executed as Virtualized Functions (VF). Each UAV can be interconnected to other UAVs and to users by means of standalone 5G technology [120]. This way, each UAV provides application flows generated by ground devices with the required VFs. To this purpose, each VF has to be run on a UAV to manage the aggregated flow coming from ground (specifically, from the devices that require it) taking into account the performance requirements specified for that VF.

Being the battery charge of UAVs limited, a *Charging and Coordination Station* (CCS) is assumed to be deployed on the ground, within a range of few kilometers from the FANET. When the battery charge of a UAV is below a given threshold, the UAV has to temporarily leave the FANET and fly to the CCS for recharging. Every time a UAV leaves the FANET, the FANET has to be re-configured, meaning that the VFs currently executed by that UAV must be migrated to other UAVs, with the goal of guaranteeing service continuity to the users. However, if not done appropriately, such a re-configuration may impact on battery consump-

tion and, consequently, on flight duration of the UAVs involved in VF migration, with a potential impossibility for the FANET to guarantee service continuity if too few UAVs are present in the FANET at the same time. A *energy-aware FANET re-configuration* needs thus to be performed. A *Re-Configuration Module* (RCM) that runs on a dedicated computing node placed in the CCS and named *Computing Remote Node* (CRN) is in charge of this task.

CCS-UAVs communication is guaranteed by the most appropriate wireless network technology, such as a *Private 5G Network* [121] or a *Low-Power Wide-Area Network* (LPWAN) (e.g. based on LoRaWAN) [122]. The latter case is preferable from a UAV energy consumption minimization perspective, but *(i)* it requires that each UAV is equipped with a LoRaWAN TX/RX module and *(ii)* the available bandwidth is limited. The former case is more energy-hungry but does not need any additional TX/RX hardware, as the already-available 5G antenna is enough to enable the UAV-CCS communication. A thorough analysis aimed at identifying the most appropriate solution is out the scope of this research, and strongly depends on the considered context.

2.7.2 Functional architecture

The main architectural elements of the system, as shown in Fig. 2.52, are the UAVs and the CCS. Their functional architecture and components are described in the following. The four main components of any UAV are (see Fig. 2.53) the *Engines*, the *Computing Element*, the *TX/RX Module* and the *Battery*. A *UAV Local Manager* is in charge of coordinating the behavior of the above modules, as specified below.

The *Battery* is used to supply all the three former modules. Its current charge level is communicated to the UAV Local Manager such that it can take its decisions regarding the amount of CE computing power using and when landing for battery recharging.

The CE is the core component of the UAV, since it ensures that the UAV can act as a NFVI-PoP, being able to execute VFs inside Virtual Machines (VM). For this reason, a hypervisor (e.g. CentOS) is installed on each CE to provide VMs hosting VFs with a virtualization of the underlying hardware resources.

The *TX/RX Module* includes a 5G antenna, and may also include a LoRaWAN module as specified in the previous subsection. Since power consumption of this module is few hundred mW, while power consumption of engines and CE is dozens of Watt, we can neglect the former, considering in the sequel only the latter in the overall power load of the UAV Battery.

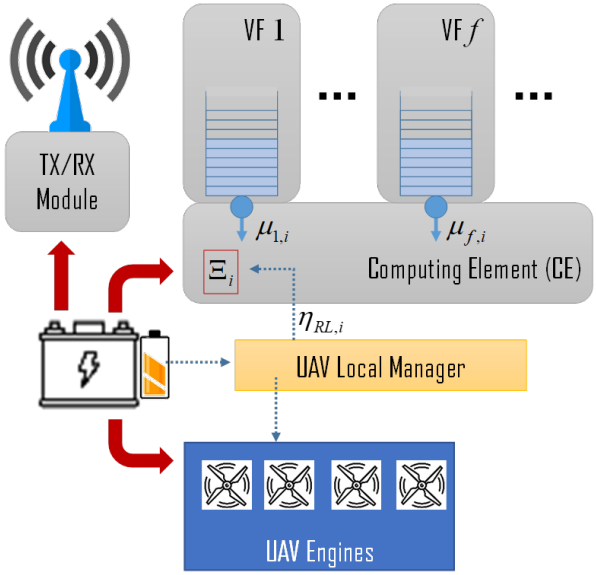


Figure 2.53: Functional architecture of a UAV

As already pointed out in previous work [42–44], the amount of energy consumed by the CE is comparable with that consumed by the engines. For this reason, a careful optimization of the *CE power management* and *VF placement* (i.e., FANET (re)-configuration) is needed to avoid an unsustainable reduction of flight duration. This task is one of the main duties of the CCS. In addition, it is needed that FANET management guarantees that the number of UAVs does not decrease too much in order to limit the possibility of not being able to guarantee the service to the requesting flows.

The UAV Local Manager of each UAV interacts with the CCS at each re-configuration instance by sending the current level of battery of all the UAVs that are active in the FANET, and receiving updates on both the computation power to be set in the CE and the VFs to be run until the next re-configuration instance.

The CCS is the place where UAV batteries are re-charged and where FANET re-configuration decisions are taken. It includes two main components: the *Charging Station* and a *Computing Remote Node*. With respect to the Charging Station, the CCS is designed in such a way that the outage period of UAVs with residual low battery power is minimized. There, for this reason, backup batteries are over-provisioned and a sufficient number of electrical plugs is available to charge all the spare batteries. When a UAV lands, it thus always finds an available battery that can replace the discharged one. The replacement is done fast by an automated system without human intervention [123], and the UAV can fly to its spot in the FANET back right after. In this way, the outage period for a UAV is only related

to the time needed for replacing the battery at the CCS and the time to fly to the CCS and back.

The *Computing Remote Node* (CRN) is a powered computing device, with TX/RX capabilites, that runs the Re-Configuration Module. RCM is able to periodically retrieve all the relevant information from the UAVs (i.e. the battery status cited so far, together with the currently-executed VFs and the amount of traffic towards VFs) needed for taking energy-aware decisions on the *best* FANET re-configuration, and for communicating its decisions to the UAVs for re-configuration execution. Typically, the best re-configuration is that minimizing the FANET energy consumption as a whole while ensuring service continuity to all Application Services over time. RCM is re-ran each time the FANET status changes. This may happen for some different reasons: *(i)* The FANET topology changes after that a UAV leaves the FANET to fly to CCS or a UAV comes back after battery replacement at the CCS; *(ii)* The input traffic towards one or more VFs has considerably changed, and current configuration is not energy efficient anymore; *(iii)* New VFs have to be executed by the FANET upon ground devices' requests. In all these cases, the RCM uses its TX/RX module (a 5G antenna + virtualized 5G core capabilities, or a LoRaWAN Gateway) to send to UAVs the new configuration to be implemented.

2.7.3 FANET Resource Orchestration

Orchestration of FANET resources aims at minimizing the service downtime. To this purpose, it is realized according two successive steps that are executed at the occurrence of each event that modifies the FANET composition. This happens when one UAV leaves the FANET for battery recharging or comes back to the FANET after battery recharging, when a flow requests a new VF service, or when a flow already using a VF service changes its behavior (e.g. its mean bitrate). Artificial intelligence for decision-making is applied by the RCM running in the CRN to take the following decisions, which are communicated to the active UAVs in the FANET:

1. the fraction $\eta_{RL,i}$ of the computing power $\mu_i^{(P)}$, to be made available by each UAV i on its CE;

2. the set F_i of VFs that each UAV i has to execute.

The target is twofold. On one side, in the short term, the overall service request, that is, all aggregated flows, each requesting a given VF with a given Quality of

Service (QoS) requirement in terms of maximum tolerated latency, should be satisfied. On the other side, service downtimes due to a too low number of active UAVs in the FANET should be minimized in the long term.

These objectives are strictly correlated to each other. In fact, a myopic placement of the VFs on any UAV should minimize energy consumption and service downtime at the short time, but could cause that some UAVs consume their battery charge very soon, in the same time interval. In this case, they are forced to land almost simultaneously, so leaving the FANET with too few UAVs to provide all the flows with the required services.

For this reason, the RCM makes the following two decision steps at each event: first, the fraction $\eta_{RL,i}$ of the computing power $\mu_i^{(P)}$ is calculated for each UAV i by means of RL, in order to minimize service downtime at the long-term. These values are used as input of an (instantaneous) optimization problem to decide VF placement with the constraint that latency for packet processing for the aggregated flow requesting the VF f be less a given delay threshold D_f.

The long-term optimization based on RL will be described in Section 2.7.4, while the short-term optimization, which is approached by means of Integer Linear Programming (ILP), will be introduced in Section 2.7.5.

2.7.4 Long-term FANET behavior optimization

In this section we describe the MDP used to support the RCM in taking the best energy-aware decisions that minimize the FANET service downtime. Specifically, a RL agent is deployed inside the RCM to choose the best reduction factor $\eta_{RL,i}$ of each UAV computing hardware mounted on board.

Let us define the MDP used by the RL agent to achieve this goal. A MDP can be defined by its environment, space state, action space and reward function.

The *environment* in which the agent takes action is the system described so far, as seen by the RCM. It is constituted by the state of charge of the battery of each UAV in the FANET, which is used to supply the engines, the computing element and the TX/RX Module. Its current charge level is communicated to the UAV Local Manager such that it can take its decisions regarding the amount of CE computing power using and when landing for battery recharging.

The *state observations* are the system state observed by the agent at the beginning of each decision epoch. Specifically, each time a UAV leaves the FANET to fly to the CCS or a UAV comes back after battery replacement, a decision epoch is triggered, and each UAV computing hardware reduction factor $\eta_{RL,i}$ is chosen.

The *action space* is the set of reduction factors to be applied to each UAV in the

FANET, H_{RL}. Specifically, at each decision epoch the agent chooses one reduction factor $\eta_{RL,i}$ for each UAV in the FANET, from a discrete set of values ranging from 0 to 1 with a constant step of 0.25:

$$\mathscr{H}_{RL} = \{\eta_{RL,0}, \ldots, \eta_{RL,N}\}, \text{ for } \eta_{RL,i} \in [0,1], \forall i. \tag{2.49}$$

Finally, we recall that the objective of the long-term reconfiguration is to ensure service continuity to all Application Services over time. For this reason, we design the *reward* so that its maximization would minimize the total service downtime. Therefore, at the end of each decision epoch n, the RCM will retrieve the amount of time in which some VF could not be deployed, and therefore users experienced service downtime, $S_{down}(n)$. Let us set the reward r_n as the opposite of the service downtime:

$$r_n = -S_{down}(n) \tag{2.50}$$

Recall that the goal of the agent is to maximize the total reward G_n it receives with a discounting factor γ, calculated as:

$$G_n = \sum_{k=0}^{+\infty} \gamma^k \cdot r_{n+k+1} \tag{2.51}$$

Therefore, the long-term goal of the agent is to minimize the amount of service downtime experienced over a large time horizon.

2.7.5 VF Placement short-term Optimization

Let V be the set of UAVs, including both the ones that are flying in FANET, so providing service, and the ones that are temporarily left the FANET for battery recharging. Let $i \in V$ be the generic UAV, and F the set of available VF instances to be deployed on the FANET as a whole. We refer to an aggregated flow as the superposition of all the flows that require the same VF with the same QoS requirement. Let λ_f be the packet rate of the aggregated flow requiring the VF f.

The short-term optimization problem consists in deciding the VF placement while minimizing the total power consumption and the percentage of flows not served because the required VF has not been placed. To this purpose, we define an optimization problem as follows.

Let u_i be the variable representing whether the UAV i is used to host at least

one VF or not:

$$u_i = \begin{cases} 1 & \text{if the UAV } i \text{ hosts at least one VF} \\ 0 & \text{otherwise} \end{cases} \tag{2.52}$$

Let $v_{f,i}$ be the variable representing whether the VF f is assigned to the UAV i or not:

$$v_{f,i} = \begin{cases} 1 & \text{if the VF } f \text{ is assigned to the UAV } i \\ 0 & \text{otherwise} \end{cases} \tag{2.53}$$

Let p_f be the variable representing whether the VF f is placed on a UAV or not. Since each VF is placed at most on one UAV only, this variable can be defined as follows:

$$p_f = \sum_{i=1}^{N} v_{f,i}. \tag{2.54}$$

The objective function weighing the contribution of the overall power consumption, P_T, and the lack of service, L_T, is defined as follows:

$$\Psi = \gamma_F \cdot P_T + \gamma_L \cdot L_T; \tag{2.55}$$

where γ_F and γ_L are two constants that are used to give different importance to the two components of the objective function.

The term P_T is the overall power consumption of the FANET, defined as follows:

$$\begin{aligned} P_T = \sum_{i=1}^{N} u_i \Big[& P_i^{(E)} + \eta_{RL,i} \cdot P_i^{(CE)} + \\ & + \sum_{f \in F} v_{f,i} \left(P_f^{(VM)} + P_f^{(VF)} \right) \Big]; \end{aligned} \tag{2.56}$$

where:

- $P_i^{(E)}$ is the power consumption due to the engines of the UAV i; it is aways present if the UAV i is involved in the FANET mission;
- $P_i^{(CE)}$ is the power absorbed by the computing hardware mounted on board of UAV i, independently of the number of running VFs; it is reduced by a factor $\eta_{RL,i}$ decided by the RCM, which calculates it by RL to minimize service downtime, as described in Section 2.7.4.
- $P_i^{(VM)}$ is the additional power consumption due to the execution of the VM where the VF f is executed;

- $P_f^{(VF)}$ is the power consumption absorbed to process the flow requiring the VF f. It depends on the packet flow rate, λ_f, and the energy $e_f^{(P)}$ needed by the VF f to process one packet as follows:

$$P_f^{(VF)} = \lambda_f \cdot e_f^{(P)} \tag{2.57}$$

The term L_T in (2.55) represents the deployment capability of all the VFs, defined as the fraction of flows that the FANET is able to serve. It can be calculated as follows:

$$L_T = \sum_{\forall f \in F} p_f \cdot \lambda_f \tag{2.58}$$

Each queue associated with a VF instance (see Fig. 2.52) is modeled as a M/M/1 system, which is a single-server queueing system with Poisson-distributed arrivals and exponentially-distributed packet service times.

Let Ξ_i be the processing rate of the CE installed onboard UAV i, expressed in FLoating point Operations Per Second (FLOPS), and $\eta_{RL,i}$ the reduction factor decided by the RCM installed on the CCS on ground. Assuming that all VFs are provided with no priority, i.e. the overall processing power is equally shared among the VFs running on the UAV i, the packet processing rate for the generic VF f, coinciding with the service rate of the M/M/1 queue assigned to it, is:

$$\mu_{f,i} = \frac{\Xi_i \cdot \eta_{RL,i}}{\Phi_i} \cdot \frac{1}{\Omega_f} \tag{2.59}$$

where Ω_f is the number of floating-point operations required by the VF f to process a packet, while Φ_i is the number of VFs placed on the UAV i, that is:

$$\Phi_i = \sum_{\forall f \in F} v_{f,i} \tag{2.60}$$

According to the M/M/1 queueing theory, the average response time or sojourn time (i.e. total time a packet spends in the M/M/1 queueing system) suffered by packets of the flow requiring the VF f, if deployed on the UAV i, is:

$$d_f = \frac{1}{\mu_{f,i} - \lambda_f}. \tag{2.61}$$

Now, we can formulate the optimization problem that maximizes the objective function defined in (2.55). We find the optimum set of UAVs to be included in the placement, $U = [u_i]_{1 \times N}$, and the optimum placement, i.e. $V = [v_{f,i}]_{|F| \times N}$, $|F|$ being the cardinality of F, that is, the number of all the VFs requested to the

FANET.

The optimization problem is formulated as follows:

$$\min_{U,V} \Psi \tag{2.62}$$

$$\text{s.t. C1: } \widetilde{i} = 0, \ \forall \widetilde{i} \in \widetilde{S} \tag{2.63}$$

$$\text{C2: } 0 \leq u_i \leq min\{1, \Phi_i\}, \ \forall i \in V \tag{2.64}$$

$$\text{C3: } \sum_{\forall f \in F} \lambda_f \cdot v_{f,i} < \mu_{f,i}, \ \forall i \in V \tag{2.65}$$

$$\text{C4: } \delta_f < D_f^{(MAX)} \ \forall f \in F \tag{2.66}$$

where $\widetilde{S}$ is a subset of S containing non-available UAVs. Constraint C1 imposes that no VF can run in non-available UAVs. Constraint C2 shows that the Boolean variable u_i, for each UAV i, is upper-bounded by the number Φ_i of VFs running on the UAV i. In other words, u_i cannot assume the value 1 if $\Phi_i = 0$. Constraint C3 imposes that the overall packet rate arriving to any VFs deployed on each UAV i is less than the fraction $\mu_{f,i}$ assigned to the VF f on the UAV i. Finally, constraint C4 imposes that the processing delay suffered by packets of the flow using the VF f is less than the maximum tolerable delay for that flow, $D_f^{(MAX)}$.

2.7.6 Numerical Results

In this section, we will present a use case to evaluate performance of the proposed framework.

The experiments were performed in a simulator based on OpenAI Gym [124] and Stable-Baselines3 [106]. We used a Proximal Policy Optimization (PPO) [125] agent, which is based on the Actor-Critic architecture, and is one of the most powerful RL algorithms in the current literature, by providing consistency and stability with little parameter tuning. In our agent both actor and critic networks have two fully connected layers with 64 neurons each. We also used the Adam Optimizer with a learning rate of $3 \cdot 10^{-4}$ and set the discount factor $\gamma = 0.95$.

Let us stress that, for network sizes like the ones considering in this research and characterizing FANET scenarios, derivation of the solution of the optimization problem defined in (2.62) by means of an entry-level computing hardware requires few minutes. Also, consider that it does not need to be executed at runtime, but it can be run on the Computing Remote Node offline to save the results on a table that should be modified each time the FANET behavior changes, on a larger timescale.

Let us consider a FANET composed by $|V| = 5$ UAVs, and $|F| = 10$ VFs that

have to be allocated in the FANET to provide functionalities to end users or ground devices.

The packet flow rate λ_f, as well as the additional power consumption $P_f^{(VM)}$ to maintain the VMs hosting the VFs switched on, and the energy $e_f^{(P)}$ needed by the VF f to process one packet, are listed in Table 2.5. These parameters were estimated on a real deployment of a softwarized network at the *UniCT 5G&B Lab* of the University of Catania, where a set of VNFs implemented by students for teaching purposes are running to execute different functions. Flow statistics have been achieved during some measurement experiment on flows generated by groups of users during their normal academic activities.

Regarding the UAVs, we consider five small equal quadcopters, consuming a total power for the engine $P_i^{(E)}$ equal to 66.56W, and each UAV is equipped with an INTEL NUC miniPC working as CE with a maximum execution capacity Ξ_i equal to 8.32 GigaFLOP/s. Finally, all the considered VFs have to guarantee the same requirement in terms of maximum tolerable processing delay suffered by each packet $D_f^{(MAX)}$. In our experiments, we set this delay equal to 1ms [126]. Finally, the two weights of the objective function Ψ are set as $\gamma_F = 0.5$ for the overall power consumption of the FANET, and $\gamma_L = 0.1$ for the deployment capabilities of all VFs.

Performance of the framework is evaluated by varying the mean value t_{CS} of the round-trip time needed by a UAV to reach the Charging Station, replace the battery and come back to the place of the mission, and the maximum battery capacity B. The actual value of round-trip time is randomly generated as a Gaussian random variable with average t_{CS} and standard deviation equal to 10% of t_{CS}. Results are provided with a 95% confidence interval on the third decimal place. In Table 2.6, there is a summary of all the variables and the corresponding value considered during the experiments.

In Fig. 2.54, the average number of flying UAVs is represented. It is easy to understand that, as the round-trip time increases, each UAV remains more time away from the FANET, and therefore there are fewer flying UAVs. Obviously, the maximum battery capacity has an influence on the average number of flying UAVs, since the greater the battery capacity, the less time each UAV is forced to return to the Charging Station for battery replacement.

Another consequence of the longer round-trip time is the need to place the VFs hosted by the UAV that has temporarily left the FANET on those UAVs remained active in flight. This causes an increase in the average number of VFs that each flying UAV will have to host to ensure that the FANET is able to satisfy the

Table 2.5: Packet flow rate

Function	λ_f **(packet/s)**	$P_f^{(VM)}$ **(W)**	$e_f^{(P)}$ **(mJ)**
f_1	4688	8.63	6.04
f_2	4276	10.78	4.24
f_3	3896	1.22	4.63
f_4	3164	15.41	5.55
f_5	5100	20.01	4.9
f_6	4521	3.69	5.31
f_7	3786	1.22	8.74
f_8	3520	2.78	7.44
f_9	5021	4.13	6.18
f_{10}	3762	10.16	4.1

Table 2.6: Simulation Parameters

Parameter	**Value**
$\|V\|$	5
$\|F\|$	10
$P_i^{(E)}$	66.5 W, $\forall i \in V$
Ξ_i	8,32 gigaFlop/s, $\forall i \in V$
t_{CS}	[40,90,140,190,240,290,340] s
B	[40,60,80]Wh
η_{RL}	$0, 0.25, 0.5, 0.75, 1$
γ_F	0.5
γ_L	0.1
$D_f^{(MAX)}$	1ms
Layer of the networks	2
Number of neurons for each layer	64
Learning Rate	$3 \cdot 10^{-4}$
γ	0.95

requests coming from users as much as possible (see Fig. 2.55). Also in this case, the maximum battery capacity has its impact: as the battery capacity increases, more UAVs are active in the FANET simultaneously, each one having to host a smaller number of VFs.

Since the total energy consumption also depends on the consumption due to the instantiation of the VMs that host the VFs ($P_f^{(VM)}$) and the processing load due to the input flow of each VFs, it is evident that the higher the number of VFs allocated on the same UAV, the greater the average consumption of it. The UAV will be forced to use a higher percentage of CPU in order to process a higher input load if compared to the case in which the UAVs are all present within the FANET and the VFs are distributed more uniformly. In Fig. 2.56, the average

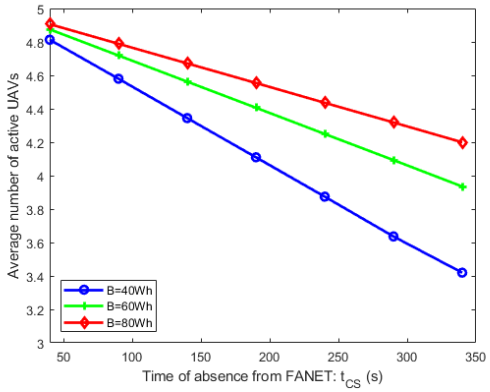


Figure 2.54: Average number of flying UAVs

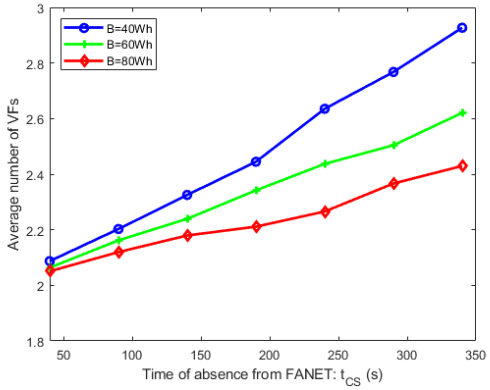


Figure 2.55: Average number of VFs running on each UAV

power consumption suffered by each UAV remaining active in the FANET is shown. The curves clearly justify what has just been said.

Coming back to what is shown in the previous figures, the possibility of having a higher battery capacity means that the UAVs are forced to return to the Charging Station less frequently. Accordingly, remaining more UAVs in flight, the VFs do not weigh down individual UAVs, which are not forced to increase the percentage of CPU to use, and therefore consume less.

As already said in the Introduction, the more resources the CE consumes, the faster the battery of the UAV runs out, thus reducing the overall FANET service availability. In Fig. 2.57, the average flight time of each UAV is represented. This trend is obviously correlated with the power consumption shown in Fig. 2.56, since the more a UAV consumes, the less its battery charge will be, and therefore the less its flying time inside the FANET.

Fig. 2.58 shows the average processing delay offered by the FANET to each packet. As we can see, in all the performed simulations, the FANET is able to satisfy the maximum acceptable delay requirement $D_f^{(MAX)}$. However, we can see

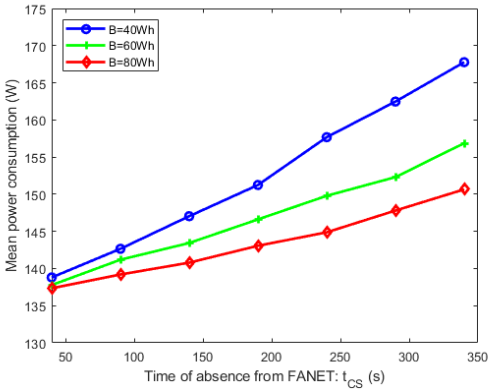


Figure 2.56: Mean power consumption of active UAVs

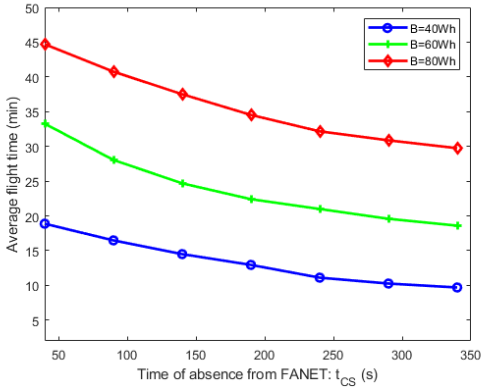


Figure 2.57: Average flight time of each UAV

that as the time of absence of the UAV from the FANET increases, the delay undergoes a gradual increase. This is due to the fact that, as there are fewer UAVs available, the VFs will be concentrated on those that remain active. This implies that even if the UAV increases the percentage of used CPU, it will have to partition this CPU to a higher number of VFs. Consequently, as the computational resources available to the single VF are smaller, the delay introduced during packet processing will increase. The difference in behavior depending on the maximum battery capacity is justified by what has already been said so far.

2.8 Contention Window Optimization in FANET

From the communication point of view, the design of a FANET should involve the definition of the *communication architecture*, the *routing protocol* and the *MAC protocol*. The *communication architecture* specifies the rules and mechanisms that determine how information flows between GCS and multiple UAVs or between

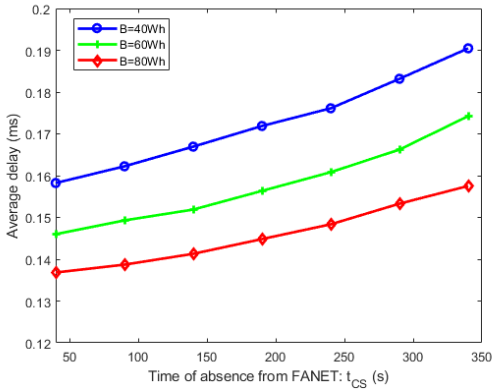


Figure 2.58: Average FANET processing delay

UAVs. We can distinguish:

- UAV direct communication: each UAV is connected to the GCS; communication between UAVs occurs via the GCS (by means of a double hop); this centralized scheme is not robust as it suffers the vulnerability of GCS;
- UAV communication via satellite networks: each UAV is connected to the GCS by a satellite link (or network); communication between UAVs can occur via the satellite network; the major drawback of this approach is the high latency of satellite communications but costs and transceiver power consumption should be considered too;
- UAV communication via cellular networks: each UAV is connected to a base station (BS) of a cellular network; the GCS is a specific node reachable from the cellular network; communication between UAVs can occur via the cellular network; this approach is not applicable when FANET missions includes natural disaster, war scenarios or terrorist attacks where the operativity of cellular infrastructure can not be guaranteed;
- UAV communication via Ad-Hoc networks: this network architecture predict that nodes communicate between them without the need for a central infrastructure; UAV to UAV and UAV to GCS communications can occur directly or by means of multi-hop; this architecture appears the most flexible for FANETs.

In order to satisfy the specific characteristics of nodes in FANET like transmission bandwidth, shortage of energy and rapid changes in links between them, most *routing protocols* used in MANET/VANET cannot be directly applied. Adapted

protocols based on static, proactive, on-demand, hybrid and geographic categories have been proposed in the literature. An exhaustive description of routing protocols mostly used in FANET can be found in [127].

The planning of *FANET MAC layer protocol* is primarily affected by the possible varying distances between the different UAVs and, secondarily, by them high mobility. The target is to guarantee a high throughput also in poor link quality and a bounded packet latency (specifically for real time applications). Accordingly to [128], the possible MAC protocols used in FANET can be selected based on communication range: for distances shorter than 10 m IEEE 802.15.1 (Bluetooth) can be used; for higher distances shorter than 250 m, one of the different amendment of IEEE 802.11 is usually applied; when higher distances should be covered (shorter than fifty km) the choice can involve IEEE 802.16 (WiMAX) or cellular 3G/4G/LTE; finally to cover higher distances a satellite access is mandatory. Anyway, the IEEE 802.11 technology appears the mostly used in FANETs involving the use of small UAVs which constitutes the majority of currently used installations. The IEEE 802.11 employs a Carrier Sense Multiple Access with collision avoidance CSMA/CA mechanism with binary exponential backoff (BEB) rules, called Distributed Coordination Function (DCF). DCF defines a basic access method, and an optional four-way handshaking technique, known as request-to-send/clear-to send (RTS/CTS) method [129]. The DCF makes use one of the parameters that plays a key role in the MAC layer behavior: the contention window size (CW). As demonstrated in [81], CW has a significant impact on the efficiency of Wi-Fi networks.

Therefore, in this research we propose the Online Smart Collision Avoidance Reinforcement learning (OSCAR) algorithm, a DRL approach that can be deployed online to quickly and efficiently find the best CW value that maximizes the throughput in the context of a FANET which implements an IEEE 802.11 protocol at MAC layer. For this purpose, it is based on Deep Deterministic Policy Gradient (DDPG) [130], that significantly enhances the performance of the network, with a fast convergence speed that does not require any previous knowledge of the system.

2.8.1 System Model

In our system, we consider a set of UAVs, denoted as M, that communicate to each other through the same Wi-Fi Access Point (AP), sharing the same channel. In the UAV, inside the AP, an OSCAR agent is deployed to choose, at runtime, the optimal CW to be used by all the stations. In order to optimize the CW, the

OSCAR agent learns the optimal policies using an RL approach. In RL, a MDP is used to provide a framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision maker, that in our system is the OSCAR agent.

In order to optimize the CW using RL, we, therefore, need to frame the problem as a MDP. Specifically, we need to define the environment state, the action space, and the rewards that the agent will receive upon executing the actions. A MDP is formally defined as a tuple (S, A, T, R, γ) in which:

- S is the state space,
- A is the action space,
- T is the set of transition probabilities among states,
- R is the reward function,
- γ is the discount factor.

In a MDP, the agent has access to all the information of the environment state, which means that the environment state and the agent state coincide. However, in our problem, the environment state consists in the state of all of the UAVs connected to the network. This is a piece of information impossible to collect. For this reason, we shift from MDPs to Partially Observable Markov Decision Processes (POMDP), which do not assume that the agent can observe the environment's state perfectly. A POMDP is formally a (S, O, A, T, R, γ) tuple in which O is the observation space. In POMDP, an observation is what the agent sees from the environment in order to execute actions.

Each time the agent has to make a decision, a new *decision epoch* n is triggered. The agent *observation* $o_n \in O$ is the frame loss rate calculated at the end of the decision epoch $n-1$, defined as:

$$\rho_n^c = \frac{F_{n-1}^s}{F_{n-1}^s + F_{n-1}^r} \tag{2.67}$$

in which F_{n-1}^s is the number of frames sent by all the UAVs to the AP during the decision epoch $n-1$, and F_{n-1}^r is the number of frames correctly received by the AP during the decision epoch $n-1$.

In particular, F_n^s can be calculated as:

$$F_n^s = \sum_{\forall t \in n-1} f_t^s \tag{2.68}$$

where f_t^s is:

$$f_t^s = \begin{cases} 1 & \text{a frame has been transmitted at time } t \\ 0 & \text{otherwise} \end{cases} \tag{2.69}$$

On the other hand, F_n^r can be calculated as:

$$F_n^r = \sum_{\forall t \in n-1} f_t^r \tag{2.70}$$

where f_t^r is:

$$f_t^r = \begin{cases} 1 & \text{a frame has been received by the AP at time } t \\ 0 & \text{otherwise} \end{cases} \tag{2.71}$$

For each decision epoch n, the agent *action* consists in choosing a new CW value, CW_n. Each UAV uses this value for the whole duration of the epoch as the upper bound of the range in which the random numbers represent the waiting times before transmissions are extracted. To this purpose, the agent uses the following standard equation:

$$CW_n = \lfloor 2^{a_n} - 1 \rceil \tag{2.72}$$

However the IEEE 802.11 standard mechanism increments a_n at each collision in the discrete range $[4, 10]$, such that the CW is doubled in the interval between 15 and 1023. On the other hand, the OSCAR agent chooses a_n in the continuous interval $A = [4, 10]$ and keeps it constant for the whole duration of the epoch. The mechanism is then implemented by setting the minimum and maximum values of the CW range for the whole decision epoch both equal to the CW_n value decided by the agent, i.e., $CW_{n,min} = CW_{n,max} = CW_n$.

Since the agent in the system is model-free, the *transition probabilities* of the environment state are unknown to the agent. This reflects what happens in real network scenarios, and therefore allows our agent to be deployed in real networks. Finally, recalling that the objective of the system is to decrease the number of collisions to increase the network throughput, the *reward* is defined as the number of frames correctly received during the current decision epoch, that is:

$$r_n = F_n^r = \sum_{\forall t \in n-1} f_t^s \tag{2.73}$$

To summarize, at the beginning of each *decision epoch*, the agent in the AP

observes the current frame loss rate, then chooses an action, transmits the chosen CW to all the connected UAVs, and then receives the reward right before the start of the next decision epoch.
The goal of the agent is to maximize the total received reward G_n discounted by γ, calculated as:

$$G_n = \sum_{k=0}^{+\infty} \gamma^k \cdot r_{n+k+1} \tag{2.74}$$

The parameter $\gamma \in [0, 1]$ is the discount factor. It determines the importance of future rewards over immediate rewards.

2.8.2 The OSCAR algorithm

In the definition of OSCAR, we use a continuous action space with a policy-based approach. As regards the first choice, in a discrete action space, the agent decides which action to perform from a finite action set, while in a continuous action space, actions are expressed as a single real-valued vector. In our system, a continuous action space can lead to better performance, since it allows the agent to choose the CW from a much larger set of values than the one in a discrete action space. The second choice i.e. using a policy-based method is due to value-based methods not performing well in continuous action space.

Moreover, we cannot allow the agent to take random actions through random exploration since it might lead to unexpected and undesired network performance. If, on the one hand, this might not be an issue in a simulated network, especially during the training phase, our goal is to design an agent able to be immediately deployed in a real network, even during the agent's training phase.

Therefore, in our system, we use a state-of-the-art deterministic policy-based method called DDPG.

2.8.3 OSCAR Execution Phases

Usually, DRL agents for Contention Window Optimization operate in two distinct phases: the learning phase and the operational phase.

In the *learning phase* the agent chooses the CW value according to what the agent has seen so far in terms of experience and, to enable exploration, each action is usually modified by a noise factor, sampled from a Gaussian distribution, which decays over the course of the learning phase. This learning phase is usually executed in controlled simulations. Occasionally, a pre-learning phase is required, in which the Wi-Fi network is controlled by legacy 802.11: this is done in order

to start storing some experience in the Experience Replay Buffer.

After the algorithm converges, the *operational phase* starts, the noise factor is set equal to zero, and the agent is deployed in the real network. The algorithm convergence is usually determined by a user-set time limit, which usually depends on the number of training steps required by the agent to usually converge.

However, in real networks, the agent cannot simply stop to learn, because the underlying environment (the network) is always evolving. The operational phase may cause some issues due to the environment being different from the one the agent has been trained on, and may lead to bad performance. For this reason, in our system, we do not distinguish between a learning phase and an operational one. In fact, the agent is always learning to be able to fastly adapt to new network conditions, triggered, for example, by new UAVs joining or leaving the FANET. To make this possible, we need to make sure that the agent learns good policies in a very short amount of training steps. We, therefore, consider the OSCAR agent execution phase in Algorithm 4.

The performance of DRL algorithms depends on the hyperparameters of the neural networks, such as their learning rates α_θ and α_w, the number of hidden layers, and the number of neurons in each hidden layer. Since the learning is done using mini-batch stochastic gradient descent, choosing the correct batch size b is also critical. Moreover, an Experience Replay Buffer B records every interaction between the agent and the environment (up to a size limit Z_{max}, that is another hyperparameter to be chosen), and serves as a base for mini-batch sampling. Finally, since in both Actor and Critics we have two neural networks, the target and the main networks, another important hyperparameter is the soft-update period τ.

After an initial phase in which the agent and the environment are initialized, the agent observes the current state of the environment, as in line 11, and chooses an action, in line 12, that is then converted into a CW using (2.72) and broadcasted to all the stations in the network, as in line 13. Then, all the UAVs in the network use that CW until a new decision epoch is triggered, that is, after d_e seconds. The agent then receives a reward, calculated as in (2.73), then observes the environment, pushes the experience tuple into the Experience Replay Buffer B, and then trains both the Actor and Critic networks, as in lines 14-19. Lines 9-20 loop indefinitely, or until a specific user-set time limit that can be manually set in line 5 by substituting the *while True* command line with the desired control loop.

Algorithm 4 OSCAR Algorithm

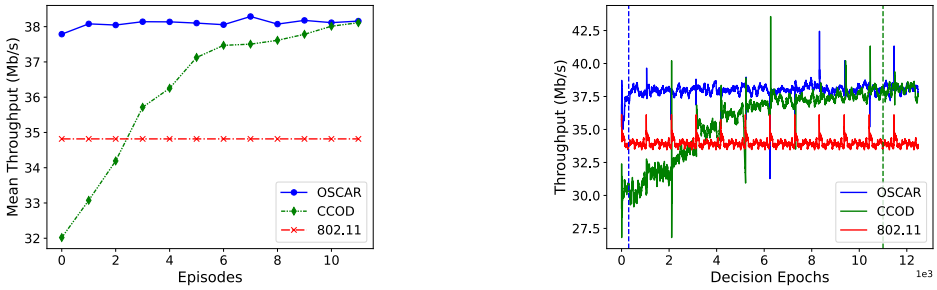
Initialize Experience Replay Buffer B with max size Z_{max}
Initialize batch size, b, and learning rate, γ
Initialize Actor and Critic main networks with weights θ and w
Initialize Actor and Critic target networks with weights θ' and w'
Initialize Actor and Critic learning rates, α_θ and α_w
Initialize decision epoch length, d_e
Initialize decision epoch n
while True do
    $F_n^s \leftarrow$ number of transmitted frames
    $F_n^r \leftarrow$ number of received frames
    $o_n \leftarrow P_n^c$ calculated as in (2.67)
    $a_n \leftarrow$ actor_target.predict(o_n)
    $o_{n+1}, r_{n+1} \leftarrow env.step(a_n)$ let the environment evolve for d_e milliseconds
    $B.push(o_n, a_n, r_{n+1}, o_{n+1})$
    $b \leftarrow$ sampled minibatch from Experience Replay Buffer
    $actor.train(b)$ train the Actor main network
    $critic.train(b)$ train the Critic main network
    $actor\_target.update()$
    $critic\_target.update()$
    $n \leftarrow n + 1$
end while

2.8.4 System Setup

We implemented OSCAR in ns3-gym [131], a framework that allows the network simulator 3 (ns3) [132] environment to be compatible with the OpenAI Gym [124] interface. Gym is the interface commonly used by RL algorithms. We implemented DDPG in Pytorch [133]. The ns-3 simulation settings were the following: error-free radio channels, IEEE 802.11ax, 1024-QAM modulation with a 5/6 coding rate, single-user transmissions, a 20 MHz channel, frame aggregation disabled, and constant bit rate UDP uplink traffic to a single AP with 1500B packets and equal offered load calibrated to saturate the network. The transfer of data required by the observation to the agent requires an overhead of 100B/s, sent from each UAV to the UAV-AP, and the dissemination of the newly chosen CW values by the AP is done through periodic beacon frames. The DRL agent hyperparameters are described in Table 2.7. Randomness was incorporated into both agent behavior and network simulation. Each experiment was run for 12 rounds of 15 seconds simulations.

Table 2.7: DRL Parameters

Parameter	Value
Decision Epoch length, d_e	10 ms
Actor Learning rate, α_θ	$3 \cdot 10^{-4}$
Actor Hidden Layers	[128,128]
Critic Learning rate, α_w	$4 \cdot 10^{-3}$
Critic Hidden Layers	[128,128]
Batch size, b	64
Discount Factor, γ	0.9
Experience Replay Buffer size, B_{max}	1×10^6
Soft update period, τ	1×10^{-1}



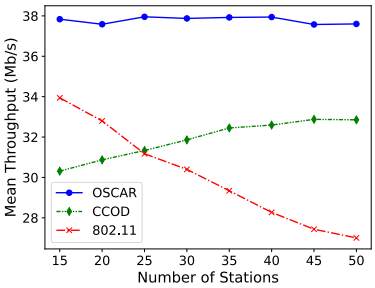
(a) Mean Episode Network Throughput (b) Instantaneous Network Throughput

Figure 2.59: Throughput in the learning phase

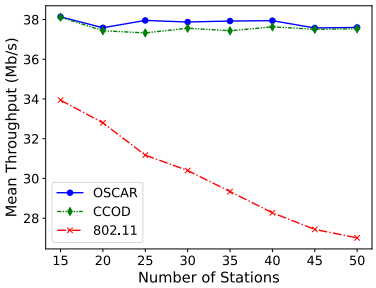
2.8.5 Numerical Results

OSCAR was evaluated for different numbers of UAVs in the FANET, in order to assess various performance aspects. We used two baselines for comparison: (a) the current operation of 802.11ax, denoted as standard 802.11, and CCOD w/DDPG [81], that is currently, at the best of our knowledge, the best DRL approach for optimizing the CW. The first baseline represents the current operation of 802.11ax, while the latter estimates the current upper bound in terms of performance. In CCOD, authors also analyzed the performance of CCOD w/DQN, but since its performance was worst than CCOD w/DDPG, we did not consider it in our performance evaluation campaign.

CCOD uses Recurrent Neural Networks with long-short term memory layers to address the possibility that older observation might be useful to the agent to make intelligent decisions to optimize the long-term network throughput. To do this, a wide history window of size 300 allowed the algorithms to take previous observations into account. Then a preprocessing phase is executed, which consists



(a) Mean Network Throughput as the number of UAVs increases at OSCAR convergence



(b) Mean Network Throughput as the number of UAVs increases at CCOD convergence

Figure 2.60: Throughput as the number of stations increase

of calculating the mean and standard deviation of the history of recently observed collision probabilities $H(P_c)$ of length h using a moving window of a fixed size and stride. This operation changes the data's shape from one- to two- dimensions, since each step of the moving window yields two data points. This collection can then be interpreted as a time series, which means it can be analyzed by a Recurrent Neural Network. The preprocessing window length was set to $h/2$ with a stride of $h/4$, where h is the history length.

Twelve rounds of OSCAR's 15-second execution phase, simulated on two NVIDIA 3070 RTX, lasted approximately 10 minutes. However, GPUs are not required to run the OSCAR algorithm since, as opposed to CCOD, OSCAR does not require complex networks such as Recurrent Neural Networks (RNN), which are usually much slower than traditional neural networks, i.e., Feed-Forward Neural Networks (FNN), especially when not supported by GPU acceleration. On top of that, OSCAR does not need any additional buffer nor data preprocessing, that is instead used in CCOD to store and preprocess the previously observed collision probabilities $H(P_c)$ in the history window, that will then be fed to the agent.

First, we evaluate how fast the OSCAR algorithm is able to recognize the optimal CW value with respect to the current state-of-the-art algorithm and what is the improvement over the standard 802.11 approach. The results show that 802.11 performance degrades as the number of UAVs in the FANET increase, while on the other hand, both OSCAR and CCOD can optimize the CW value in static network conditions, as shown in Fig. 2.59. However, as shown by the blue dashed vertical line in Fig. 2.59b, the OSCAR algorithm converges to the optimal solution in about 300 decision epochs (iterations), while CCOD requires 11000 decision epochs, as depicted by the green vertical line. Considering that each decision epoch lasts 0.01 seconds, this means that our approach is able to adapt to

the current dynamics of the network in about 3 seconds, while CCOD requires about two minutes. This means that OSCAR learns the optimal values 36 times faster than the current state-of-the-art DRL approach, while still reaching the same optimal results.

In Fig. 2.60, for each data point, a fixed number of UAVs has been connected to the UAV-AP throughout the simulations. In theory, a constant value of CW should be optimal in these conditions. Fig. 2.60 shows the mean network throughput right after the algorithms' convergence. Specifically, Fig. 2.60a shows the network performance after 3 seconds of the learning phase, that is when OSCAR converges to the optimal policy. On the other hand, Fig. 2.60b shows the network performance after 110 seconds of learning, that is at CCOD convergence. As shown by Fig. 2.60a, after 3 seconds, the OSCAR improvement over standard 802.11 ranges between 11% for 15 UAVs and 40% for 50 UAVs, while the improvement over CCOD ranges between 30% for 15 UAVs and 15% for 50 UAVs. As the number of UAVs increases, CCOD mean network throughput improves: this is due to the fact that CCOD convergence tends to be quicker for a higher number of UAVs. Nonetheless, CCOD is still not able to reach the same performance as OSCAR. After 110 seconds, as shown in Fig. 2.60b, CCOD converges, and thus the performance of OSCAR and CCOD is basically the same. However, this higher convergence time means that CCOD cannot reach optimal values if the number of UAVs changes every less than 2 minutes, while on the other hand, OSCAR is able to adapt to new network conditions in about 3 seconds.

experiments have shown that OSCAR achieves state-of-the-art results 36 times faster than the other DRL approaches, while also lowering the agent' computational cost.

Chapter 3

Latency and Energy Management of VANETs

In recent years, thanks to the advent of 5G technologies, network softwarization and programmability [61, 134–136], which have guaranteeing ultra-low latency, high reliability and energy saving as one of the main objectives, there has been a significant increase in deployment delay-constrained scenarios not achievable with the previous generations of mobile networks [52, 137–140]. One of the most challenging application scenarios for the above technologies is constituted by the Intelligent Transport System (ITS), enabled by the use of vehicular networks for various applications, such as traffic management, safety and entertainment [22–24]. These applications often have low latency and high-reliability targets, whose achievement may be challenging in vehicular networks due to their highly dynamic and resource-constrained nature. In fact, the limited processing and storage capabilities of the OBUs, i.e. the processing equipment installed on-board of the vehicles, are often insufficient to guarantee such requirements.

One approach to address these challenges is to use job offloading, which involves transferring some or all of the required computation to more powerful and/or better-connected devices, typically the so-called RSUs installed along the roadway. However, using job offloading in vehicular networks introduces additional challenges, including balancing latency and energy consumption.

When deployed in remote roads and rural areas, RSUs do not have access to a fixed Internet connection and/or the power grid. In such cases, the RSUs are stand-alone, battery-powered devices that can only count on the local computing units for processing and green energy harvesting as their power source.

In this perspective, for load balancing in the network, in order to reduce peaks of latency and energy consumption, each RSU should also be able to further offload

the received jobs to nearby RSUs [25].

With this in mind, a new, challenging scenario emerges, where each RSU has to autonomously find a proper trade-off between the energy consumption and the processing delay requested by vehicular services. On the one hand, this goal can be achieved by adequately tuning the amount of processing power employed and, on the other, by choosing the optimal amount of jobs to offload to the nearby RSUs.

Hence, the main motivation of this work is to develop a distributed edge-computing framework based on Multi-Player Multi-Armed Bandit (MP-MAB) algorithms for latency- and energy-aware job offloading in green vehicular networks.

Multi-Armed Bandit (MAB) algorithms [141] are a class of online learning algorithms widely used to solve resource allocation problems in various contexts, including wireless communication networks and online advertising. MAB algorithms work by allowing a decision-making agent to explore different options (or "arms") to learn which option is the best in a given context. MP-MAB algorithms [142] extend this idea to the case where multiple decision-making agents must coordinate their actions to achieve a common goal. In the context of our work, these agents reside in the RSUs and are responsible for the offloading decisions and the proper tuning of the employed processing power.

Distributed multi-armed bandit frameworks have been recently proposed as a practical approach for optimizing job offloading in vehicular networks [143–145]. These algorithms aim to maximize the network's overall performance by intelligently allocating tasks to the most suitable RSUs, taking into account factors such as latency and energy consumption.

The distributed nature of the framework allows the RSUs to share information with each other and coordinate their actions to collectively optimize the network's performance. This can lead to significant improvements in the effectiveness of job offloading in terms of energy consumption and job processing latency.

Overall, distributed bandit frameworks offer a promising solution for addressing the challenges of job offloading in vehicular networks and have the potential to impact a wide range of applications significantly.

We hereby summarize the main contributions of our work:

- we derive the mathematical model of a vehicular network infrastructure based on green, stand-alone RSUs and supported by two-level offloading capabilities, as well as of the system performance in terms of outage probability and job processing latency;

- we design MANTRA (Multi-AgeNT Rsu mAnagement), a distributed framework based on MP-MAB, to support autonomous offloading and energy-saving decisions on the RSU devices;

Moreover, we prove the effectiveness of MANTRA through an extensive numerical analysis, and demonstrate how our solution can quickly converge towards the global optimum despite its decentralized and fully-scalable nature.

The chapter is organized as follows. Section 3.1 is devoted to the revision of existing literature in the field of job offloading in vehicular networks. In Section 3.2, we explain the system model and formulate the problem. In Section 3.3, we introduce the mathematical formulation of the problem. Section 3.4 describes the proposed MANTRA framework. In Section 3.5, we discuss the setup of the simulation campaign, while in Section 3.6, some numerical results are presented.

3.1 Related Work

Authors in [144] propose a MAB-based offloading framework to support job offloading in vehicular networks. In particular, edge servers in the network can exploit Utility-table based Learning (UL) algorithms to optimally offload the received jobs to the other edge servers in the network or a remote cloud. In such a way, the authors leverage the offloading mechanism to minimize the service delay and balance the computation-intensive jobs among MEC/cloud servers in a distributed manner. However, the authors do not consider the possibility of having battery-powered, flexible edge servers and only focus on the problem of latency minimization and load balancing. The issue of energy saving, instead, is not taken into account.

In [145], the authors devise a two-level offloading scheme for vehicular networks, where both vehicles and RSUs act as MAB agents. In particular, the former can offload their jobs to the RSUs and cloud servers installed near the network's 5G base stations (gNB). Instead, the RSUs can choose to process the received jobs locally or perform a further offloading towards the available gNBs. Hence, the whole offloading framework relies on the MAB algorithm to explore and learn the best actions to minimize the average completion latency for the jobs. As in the previous case, the authors do not take the energy consumption of the system into account and work with traditional, fixed RSUs.

As opposed to the works mentioned above, this research explores the usage of flexible, portable, battery-powered RSUs. For this reason, not only do we consider the problem of latency minimization, but we also consider the crucial trade-off

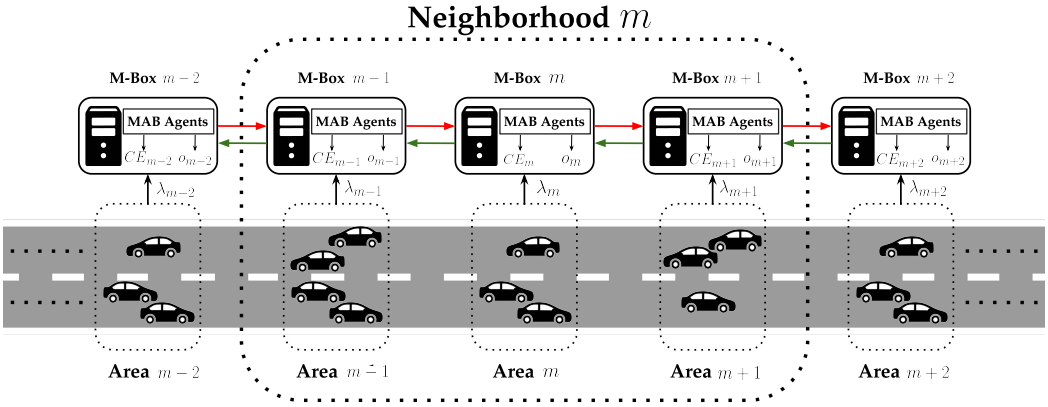


Figure 3.1: Reference system

between latency and energy consumption in the system. The idea of resorting to portable RSUs stems from our previous work in [25]. However, this research also explores an entirely new idea as i) it resorts to a lightweight, dynamic MAB framework to optimize the system performance, whereas the work in [25] leverages on model-based reinforcement learning which, on the one hand, is more precise and accurate, but may fail in adapting to a dynamic vehicular system; ii) This research introduces the idea of fully-distributed decision-making systems with local information only, as opposed to the previous work [25], where the RSUs had full knowledge over the system state.

3.2 The Reference System

In the following, we describe the scenario addressed in this work. Here, intelligent vehicles flow along both directions of a stretch of road. Each vehicle is equipped with several sensors and runs one or more smart tasks, ranging from safety to infotainment applications, from traffic improvement to cooperative, and autonomous driving [146, 147].

The tasks can be split into one or more *jobs*. Each job can be either processed by the local OBU installed on the vehicles where that job is generated or, leveraging on edge computing facilities, offloaded to the MEC-in-a-box (M-Box) stations placed along the road [25]. Each M-Box includes a RSU to communicate and process the jobs offloaded by the vehicles, a battery to supply the communication and processing devices, and a microeolic turbine acting as a power generator to recharge the battery.

As depicted in Fig. 3.1, the stretch of road is covered by a set $\mathcal{M} = \{0, 1, \ldots, M-1\}$ of M-Boxes, where each M-Box m manages a specific portion of the road, or

Area, that is, its radio coverage area. Hence, each M-Box m will receive and process the jobs generated by the vehicles in the coverage Area m.

Clearly, the possibility of offloading jobs can help relieve the processing burden from the vehicles. Still, in case of intense traffic conditions in specific areas, the corresponding M-Boxes may be overloaded by the high volume of generated jobs, with a severe impact on the system performance (quick battery drain and/or high processing latency).

For this reason, together with the traditional offloading procedure from the smart vehicles to the M-Box (*vertical offloading*), in the reference system, the M-Box devices are provided with the possibility to further offload the jobs to the other M-Boxes in the system (*horizontal offloading*).

The horizontal offloading procedure proves fundamental, as it introduces the possibility to *balance* the load in the network. Overloaded M-Boxes can leverage the horizontal offloading procedure to transfer part of the received jobs to their *neighbors*. In particular, two M-Boxes are considered neighbors whenever they are in radio visibility with each other. Hence, we introduce the definition of *Neighborhood*, where a Neighborhood m is correspondingly centered on the M-Box m. Accordingly, all the M-Boxes included in the Neighborhood m belong to the set N_m, i.e., the set of M-Boxes directly reachable from the M-Box m for horizontal offloading. For the sake of simplicity, we assume that each M-Box has only two neighbors, i.e., the next and previous M-Boxes along the road. The first and the last M-Boxes represent an exception, as they only have one neighbor (the next and the previous M-Box, respectively).

Finally, to offload their jobs, the M-Boxes have to incur one or more costs of heterogeneous nature (an energy cost, a monetary cost, or both).

The horizontal offloading can, therefore, crucially improve the load distribution in the system, with a significant impact on the processing delay and, thus, the system responsiveness.

Energy saving is another crucial aspect for the mobile and battery-powered M-Boxes. In such a perspective, M-Boxes are provided with the possibility to turn on and off their processors, here referred to as *Computing Elements* (CEs). Hence, whenever the job load is minimal, the M-Boxes can turn off most of their CEs, thus saving energy. Vice versa, in case of intense loads, the M-Boxes can turn on all the available CEs in response to the massive amount of processing requests.

Therefore, a trade-off emerges between the system energy consumption and the system latency. Indeed, depending on the specific system application and the system state, the M-Boxes could decide to either accept a more significant job

latency to save energy or, instead, reduce the system latency as much as possible by turning on more CEs.

With all this in mind, each M-Box can decide i) the number of CEs to keep active and ii) the percentage of jobs to offload to its neighbors. The decision depends on several constraints/requirements, such as the residual system autonomy, the desired job processing latency, and the offloading cost. These requirements must be appropriately formalized and modeled in the form of specific *reward functions*.

3.3 Analytical Model

In this section, we provide the mathematical formulation of the system performance. For the sake of scalability and simplicity, we treat the decision on offloading (*offloading problem*) and the one on the number of active CEs (*CE problem*) as separate, though intertwined issues.

As regards the CE problem, the reward function for a generic M-Box m can be defined as follows:

$$\begin{aligned} F_{RW,m}^{(CE)} = \quad & \alpha \frac{\log p_{L,m} - \log \epsilon_L}{\log \epsilon_L} H(\log p_{L,m} - \log \epsilon_L) \; + \\ & (1-\alpha) \frac{\log p_{D,m} - \log \epsilon_D}{\log \epsilon_D} H(\log p_{D,m} - \log \epsilon_D) \end{aligned} \tag{3.1}$$

where:

- $p_{L,m}$ is the so-called *latency threshold violation probability*, that is, the probability that the waiting time for a job in the M-Box m queue exceeds a given threshold τ_{th};
- ϵ_L is the target value for $p_{L,m}$;
- $p_{D,m}$ is the battery depletion probability or, equivalently, the outage probability for the M-Box m;
- ϵ_D is the target value for $p_{D,m}$;
- α is a weighing parameter to set the relative importance of the function components;
- $H(\cdot)$ is the Heaviside unit step function.

Note how both the components of the reward function are properly normalized in the interval [0,1] to be properly comparable. Since both $p_{L,m}$ and $p_{D,m}$ depend

on the number of active processors on the M-Box m, a proper choice of the number of active CEs proves crucial to achieving the desired trade-off between energy consumption and system latency. Before defining the mathematical expression for $p_{L,m}$ and $p_{D,m}$, let us first introduce the reward function for the offloading problem. Again, for a generic M-Box m, we define the reward function for the offloading problem as:

$$F_{RW,m}^{(OL)} = \beta \frac{\log p_{L,m} - \log \epsilon_L}{\log \epsilon_L} H(\log p_{L,m} - \log \epsilon_L) + (1-\beta)\frac{\sigma_m}{\sigma_{m,max}} \tag{3.2}$$

where:

- σ_m is the sum of energy and/or monetary costs that the M-Box m has to pay whenever a job is offloaded. In particular, it can be calculated as $\sigma_m = [\sigma^{(e)} + \sigma^{(f)}] \cdot \lambda_m o_m$, where λ_m is the job generation rate for Area m, o_m is the offloading probability for the M-Box m, and $\sigma^{(e)}$ and $\sigma^{(f)}$ are the offloading energy cost and the monetary offloading fee, respectively.

- β is a weighing parameter to set the relative importance of the function components.

As in the previous case, all the components are normalized in the interval [0,1].

Let us delve into the calculations of $p_{L,m}$ and $p_{D,m}$. To simplify the notation, in the following, we will omit the subscript m. Without loss of generality, we assume job generation and job processing to be Poisson processes. The latter is distributed with parameter μ_{CE}, identical for all the M-Boxes in the system (i.e., all the M-Boxes are equipped with identical hardware).

The job generation process strictly depends on the underlying vehicular traffic distribution. In particular, let us note how very different time scales characterize the job generation and traffic variation processes. Indeed, while jobs are usually generated and processed in time scales of tens or hundreds of milliseconds, the traffic distribution can take minutes, and even hours, to change over time. For this reason, we assume the vehicular traffic distribution to be "static". Hence, we model the M-Boxes as stationary M/M/CE/Q_C systems, with Poisson-distributed arrival and service processes, a variable number of servers equal to the active CEs on the M-Box, and a maximum job queue length equal to Q_C.

The mean rate of arrivals to the queue of the M-Box m is equal to the sum of the job generation rate of the Area m and the rates of jobs offloaded by the neighboring M-Boxes, minus the rate of jobs offloaded by the M-Box m.

Accordingly, the overall arrival rate to the queue of the M-Box m can be calculated as:

$$\lambda_{in,m} = \lambda_m(1 - o_m) + \sum_{i \in N_m} \lambda_{i \to m} \tag{3.3}$$

The term $\lambda_{i \to m}$ in (3.3) represents the job offloading rate from the M-Box i to the M-Box m and depends on the offloading mechanism. In principle, the offloading decision specifies the portion of incoming jobs to be offloaded but does not specify the destination M-Box. For the sake of fairness and load balancing, offloaded jobs are proportionally split among the neighboring M-Boxes. Accordingly, $\lambda_{m \to n}$ can be derived as:

$$\lambda_{m \to n} = \lambda_m o_m \Big(1 - \frac{\rho_n}{\sum_{i \in N_m} \rho_i}\Big) \quad \text{with} \quad n \in N_m \tag{3.4}$$

In other words, the idea is to evaluate the average load for each neighboring node and to distribute the offloaded jobs in a balanced way, where the most underloaded M-Boxes receive the most significant portion of offloaded jobs.

Now, the term p_L, representing the probability that waiting time W in the job queue is higher than the latency threshold τ_{th}, can be derived as the sum of the probabilities of the queue states causing this condition, that is $p_L = \sum_{q=0}^{Q_C} g_q$, where:

$$g_q = \Pr\{W > \tau_{th}\} = \begin{Bmatrix} \pi_q \text{ if } \dfrac{1}{\mu_{CE}}(\lceil \dfrac{q}{CE} \rceil + 1) > \tau_{th} \\ 0 \quad \text{otherwise} \end{Bmatrix} \tag{3.5}$$

Here, π_q is the stationary probability of having a queue length of q jobs. It can be derived using the well-known queueing theory of the M/M/m/K systems. The above condition can be explained as follows: when a job enters the queue, it has to wait for $\lceil \frac{q}{CE} \rceil$ service times before being processed, plus an additional service time for the processing operation. Hence, the waiting time in the system for a job is equal to the total number of service times divided by the average processing rate.

Likewise, p_D can be derived using the queueing theory. As done in [25], the M-Box battery can be modeled as an M/M/CE/Q_B queue, where each element in the queue represents a battery charge quantum, namely u_B. In particular, the battery queue length can be calculated as $Q_B = \frac{B_c}{u_B}$, where B_c is the battery capacity in Ah. The queue arrivals are determined by the battery recharge process and strictly depend on the wind conditions. The service process, instead, depends on the number of active CEs: the more active CEs, the bigger the energy consumption and the faster the battery discharge rate.

The condition $q_B = 0$, where q_B is the current queue length, corresponds to a fully depleted battery. Hence, p_D is equal to π_0, i.e., the stationary probability of being in the queue state 0. Once again, π_0 can be calculated from the queueing theory applied to the system M/M/m/K systems.

3.4 The MANTRA Framework

In this section, we introduce MANTRA, the proposed distributed framework based on MP-MAB to address the challenge of balancing latency and energy consumption in the reference system. In MANTRA, each M-Box acts as a decision-making agent that chooses which job offloading strategy to use according to both the current network conditions and the performance of past offloading decisions. Contextually, each M-Box is also responsible for choosing the number of CEs to be turned on.

3.4.1 Multi-player Multi-armed Bandit

MP-MAB algorithms [142] are online learning algorithms widely used to solve resource allocation problems in various contexts, including wireless communication networks and online advertising. MAB algorithms work by allowing a decision-making agent to explore different options (or "arms") in order to learn which is the best in a given context. MP-MAB algorithms extend this idea to the case where multiple decision-making agents must coordinate their actions in order to achieve a common goal. From a high-level perspective, MP-MAB algorithms can be thought of as a *game* played between the decision-making agents and the environment. The decision-making agents take actions by selecting arms, and the environment responds by providing a reward. The game's goal is to maximize the total reward of the agents over time.

More formally, we can define an MP-MAB algorithm as:

- there are K arms, and each arm has an associated reward distribution; there are N decision-making agents, and each agent has a set of actions, A, that it can take;
- at each time step t, each agent i selects an action a_i from its set of actions;
- the environment responds by providing a reward r_i to each agent based on its action and the reward distribution of the selected arm.

Algorithm 5 MANTRA Learning Algorithm

$UCB_{k,m}^{(CE)} = 0,\ \forall m \in \mathcal{M}, \forall k \in \{1, 2, ..., K_m^{(CE)}\}$ <Initialize computing agents UCB values for each M-Box m>
$UCB_{k,m,CE_m}^{(OL)} = 0,\ \forall m \in \mathcal{M}, \forall k \in \{1, 2, ..., K_m^{(OL)}\}, \forall CE_m$ <Initialize offloading agents UCB values for each M-Box m>
$t = 0$ <Initialize time step>
env.reset() <Initialize environment>
while True do
  $a_m^{(CE)} = [\ ]$ <Initialize computing agents action vector>
  $a_m^{(OL)} = [\ ]$ <Initialize offloading agents action vector>
  for $m \in \mathcal{M}$ **do**
    select $k_m^{(CE)}$ as in (3.7) <the computing agent pulls an arm>
    push $k_m^{(CE)}$ into $a_m^{(CE)}$ <the selected arm is inserted into the computing action vector>
  end for
  for $m \in \mathcal{M}$ **do**
    Retrieve context CE_m from $a_m^{(CE)}$ <the offloading agent retrieves the number of active CEs for each M-Box in its neighborhood>
    select $k_m^{(OL)}$ as in (3.9) based on CE_m <the offloading agent pulls an arm based on the context>
    push $k_m^{(OL)}$ into $a_m^{(OL)}$ <the selected arm is inserted into the offloading action vector>
  end for
  $r_m^{(CE)}, r_m^{(OL)} = \text{env.step}(a_m^{(CE)}, a_m^{(OL)})$ as in (3.6) and (3.8) <execute actions into the environment and retrieve the rewards>
  for $m \in \mathcal{M}$ **do**
    Update $\overline{r}_{k,m}^{(CE)}$ and $\overline{r}_{k,m,CE_m}^{(OL)}$ of each $k_m^{(CE)}$ and $k_m^{(OL)}$ arm in $a_m^{(CE)}$ and $a_m^{(OL)}$ <the pulled arm mean rewards are updated>
  end for
  $t \leftarrow t + 1$
end while

One variant of the MP-MAB algorithm is the Multi-Player Independent MAB (MP-IMAB) algorithm. In MP-IMAB, each agent is treated as an independent decision-maker, and the rewards of the different agents are assumed to be independent.

Actually, in the addressed reference system, the rewards of different agents cannot be assumed to be independent. This requires leveraging another class of MP-MAB algorithms, called Multi-Player Correlated MAB (MP-CMAB) algorithms, which take into account the correlations between the rewards of the different agents. In MP-CMAB algorithms, each agent is still treated as an independent decision-maker, but the rewards of the agents are no longer assumed to be independent.

3.4.2 MANTRA Agents

In the following, we describe the two types of agents that are deployed in each M-Box $m \in \mathcal{M}$: one that optimizes the number of computing elements, called the *CE* agent, and one that optimizes the offloading strategy, the *offloading* agent.

The *CE* agent is responsible for determining the optimal number of computing elements for each task based on the performance of past decisions. This agent exploits the Upper Confidence Bound 1 (UCB1) [148] algorithm to learn the relationship between the number of CEs used and the resulting latency and energy consumption.

The *offloading* agent is responsible for determining the best offloading strategy to use according to the current network conditions, i.e., the number of active CEs, and the performance of past offloading decisions. This agent implements the Contextual UCB1 algorithm to learn the relationship between the different offloading strategies and the corresponding latency and energy consumption.

Specifically, each M-Box is provided with a certain number of CEs that can be turned on. Let C be the maximum number of CEs in an M-Box. Therefore, $K_m^{(CE)} = C$ is the number of arms of the CE agent in the M-Box m. Then, at each time step t, when arm $k_m^{(CE)} \in \{1, 2, ..., K_m^{(CE)}\}$ is pulled, $k_m^{(CE)}$ CEs are accordingly powered on.

Afterward, each CE agent receives a reward based on the latency of the whole neighborhood and its own outage probability. For this reason, we can define the reward of each CE agent in the M-Box m as follows:

$$\begin{aligned} r_m^{(CE)} = \min_{i \in N_m} \quad & \alpha \frac{\log p_{L,i} - \log \epsilon_L}{\log \epsilon_L} H(\log p_{L,i} - \log \epsilon_L) \; + \\ & (1-\alpha) \frac{\log p_{D,m} - \log \epsilon_D}{\log \epsilon_D} H(\log p_{D,m} - \log \epsilon_D) \end{aligned} \tag{3.6}$$

At each timestep t, each CE agent in M-Box m pulls the arm with the highest UCB, that is:

$$k_m^{(CE)} = \text{argmax}_{k \in \{1,...,K_m^{(CE)}\}} \left(\overline{r}_{k,m}^{(CE)} + c \cdot \sqrt{\frac{log(t)}{n_k}} \right) \tag{3.7}$$

where $\overline{r}_{k,m}^{(CE)}, k \in \{1, ..., K_m^{(CE)}\}$, is the mean reward of each arm k, n_k is the number of times arm k has been selected, and c is a positive constant that gauges the exploration/exploitation trade-off.

On the other hand, the offloading agents determine the probability of offloading

incoming jobs to the other neighbors, as explained in Section **??**. However, since MAB algorithms do not work well when provided with a large number of arms, we limit the set of offloading probabilities to $\mathcal{O} = \{0\%, 10\%, ..., 100\%\}$. More specifically, when the offloading agent in M-Box m pulls an arm $k_m^{(OL)} \in \{0, 1, ..., 10\}$, it accordingly sets the offloading probability to $o_m = (k_m^{(OL)} \cdot 10)\%$. Contrary to the CE agents, the offloading agents resort to contextual information to choose the arm to pull. In our system, the context consists of the number of CEs active in each M-Box of the neighborhood. This means that the offloading agent in the M-Box m is aware of the set of active CEs of its neighborhood N_m, i.e., $CE_{N_m} = \{k_i^{(CE)}\}, \forall i \in N_m$. For the sake of conciseness, we will afterward refer to CE_{N_m} as CE_m. Similarly to what happens with the CE agents, the offloading agents receive a reward based on their neighborhood performance and the amount of offloading they perform. We define the reward of the offloading agent m as follows:

$$\begin{aligned} r_m^{(OL)} = \min_{i \in N_m} \beta \frac{\log p_{L,i} - \log \epsilon_L}{\log \epsilon_L} H(\log p_{L,i} - \log \epsilon_L) + \\ + (1 - \beta) \frac{\sigma_m}{\sigma_{m,max}} \end{aligned} \tag{3.8}$$

Then, at each time step t, each offloading agent in M-Box m pulls the arm which offers the highest UCB according to the current context CE_m, that is:

$$k_m^{(OL)} = \text{argmax}_{k \in \{1,2,...,K_m^{(OL)}\}} \left(\overline{r}_{k,m,CE_m}^{(OL)} + c \cdot \sqrt{\frac{log(t)}{n_{k,CE_m}}} \right) \tag{3.9}$$

where $\overline{r}_{k,m,CE_m}^{(OL)}$ and n_{k,CE_m} are respectively the mean reward of each arm and the number of times each arm has been selected in context CE_m.

By having both of these agents running in each M-Box, the MANTRA framework can dynamically optimize the number of used CEs and the offloading strategy, leading to improved performance in terms of latency and energy consumption. We show the MANTRA learning phase in Algorithm 5.

3.5 Simulation Setup

This section describes the simulation setup for evaluating the proposed MANTRA framework. Our simulation includes several intertwined key components, namely the network model, the set of MAB algorithms, the task offloading model, and the evaluation framework.

The network model describes the topology and characteristics of the vehicular

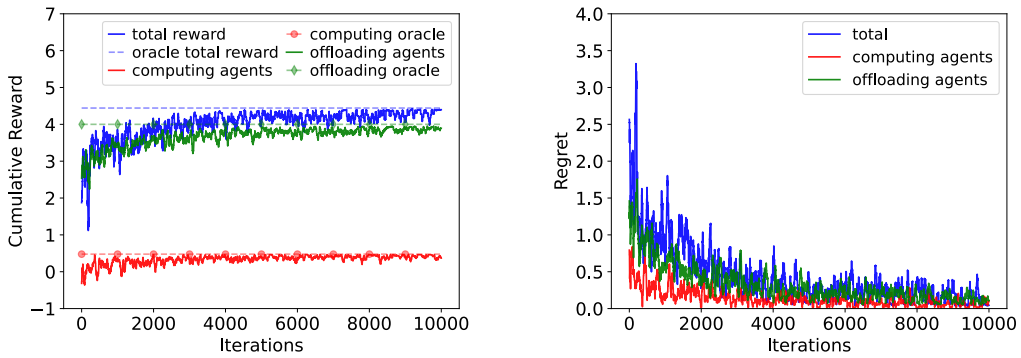
Table 3.1: Simulation Parameters

Parameter	Value
Number of M-Boxes	4
Computing Agents UCB1 constant, $c^{(CE)}$	$\frac{1}{2}$
Offloading Agents UCB1 constant, $c^{(OL)}$	$\frac{1}{2}$
Computing Agents max number of arms, $K_m^{(CE)}$	varied in [1, 6]
Offloading Agents max number of arms, $K_m^{(OL)}$	10
Computing reward weighing parameter, α	0.75
Offloading reward weighing parameter, β	0.75
CE service rate, μ_{CE}	1250 packets/s
Vehicular traffic distribution, λ_m	$[1.4, 0.4, 2.2, 0.8] \cdot \mu_{CE}$
Queue waiting time target, τ_{th}	8 ms
Queue waiting time violation probability, ϵ_L	$1 \cdot 10^{-8}$
Outage target probability, ϵ_D	$4 \cdot 10^{-2}$
Maximum job queue size, Q_C	100
Battery Capacity, B_c	6.2 Ah
Battery size, Q_B	100
Battery recharge rate, λ_B	0.186 Ah/s
Battery discharge rate, μ_B	$0.062 \cdot k_m^{(CE)}$ Ah/s
Energy cost, $\sigma^{(e)}$	0.125
Monetary cost, $\sigma^{(f)}$	0.125

network, including the capabilities of the M-Boxes and the offloading rate of the vehicles. In particular, in our simulation campaign, we consider $M = 4$ M-Boxes deployed in a straight road.

The second component is the set of multiple bandit algorithms that learn through a trial and error process based on the network's feedback and the task offloading model. In our framework, we deploy 4 UCB1 computing agents and 4 Contextual UCB1 offloading agents, i.e., one UCB1 computing agent and one Contextual UCB1 offloading agent for each M-Box. We study the system's performance for a variable number of $K_m^{(CE)}$, i.e., the maximum number of available CEs. In particular, we let $K_m^{(CE)}$ vary from 1 to 6. On the other hand, we fix the maximum number of arms for the offloading agent to $K_m^{(OL)} = 10$. The reward weights α and β in (3.1) and (3.2) are both set to 0.75. We also set the gauge constant that controls the exploration/exploitation trade-off, c, to 0.5.

The task offloading model simulates the process of offloading tasks from one M-Box to another, taking into account factors such as the available bandwidth, the distance between the M-Boxes, and the cost of offloading. Generally speaking, each M-Box can communicate with the other M-Boxes in its neighborhood. In



(a) MANTRA Agents' Cumulative Reward (b) MANTRA Agents' Regret

Figure 3.2: MANTRA learning phase

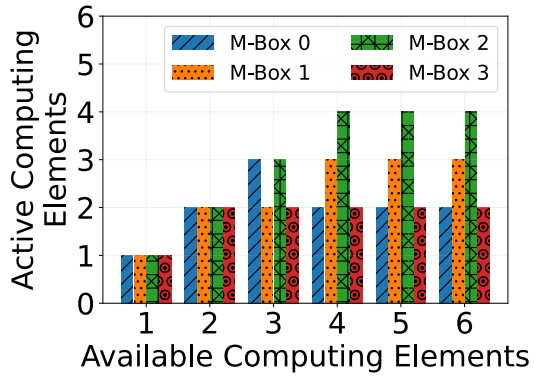
our specific setting, each M-Box m is, therefore, able to communicate with M-Boxes $m-1$ and $m+1$, with $m=0$ and $m=M-1$ being the only exceptions. In fact, $m=0$ is the first M-Box, and can only offload tasks to the M-Box 1. Similarly, $m=M-1$ is the last M-Box and can only offload to M-Box $M-2$. The battery capacity, B_C, is set to 6.2 Ah, while a battery charge quantum u_B is equal to 0.062 Ah. Therefore, the battery queue length, Q_B, is equal to 100 energy quanta. Moreover, the maximum job queue length, Q_C, is set to 100 jobs. As regards the energy and monetary costs $\sigma^{(e)}$ and $\sigma^{(f)}$, they are both set to 0.5. The M-Box are recharged with a rate equal to $\lambda_B = 3u_B = 0.186$ Ah per second, and discharged with a rate $\mu_B = u_B = 0.062$ Ah per second for each active CE.

Regarding the latency requirement, we take Ultra-Reliable Low-Latency Communication (URLLC) [149] as the reference case study. In URLLC use cases, the maximum probability that the latency requirements are not met is usually set between 1×10^{-5} and 1×10^{-8}. For this reason, we set the target latency threshold violation probability to $\epsilon_L = 1 \cdot 10^{-8}$. Moreover, according to the typical URLLC requirements, we set a maximum latency threshold equal to 8 ms. Finally, we set the outage target probability, ϵ_D, to $4 \cdot 10^{-2}$.

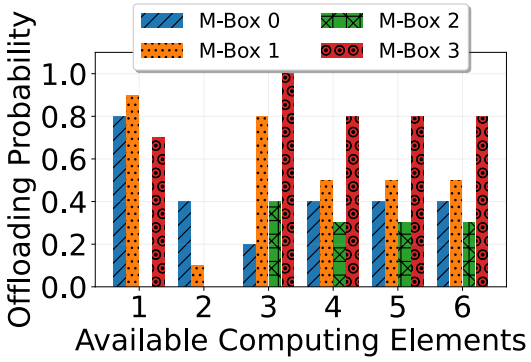
The job generation process, which depends on the vehicular traffic distribution, λ_m, is different for each Area m: $[\lambda_0, \lambda_1, \lambda_2, \lambda_3] = [1.4, 0.4, 2.2, 0.8] \cdot \mu_{CE}$.

Finally, the evaluation framework leverages the Weights and Biases [150] and Stable Baselines3 [151] libraries to run simulations and collect data on the system's performance. The evaluation framework implements algorithms for measuring latency, energy efficiency and offloading cost metrics throughout the simulation. In general, these components enable the evaluation of the performance of the bandit algorithms and provide feedback to these components.

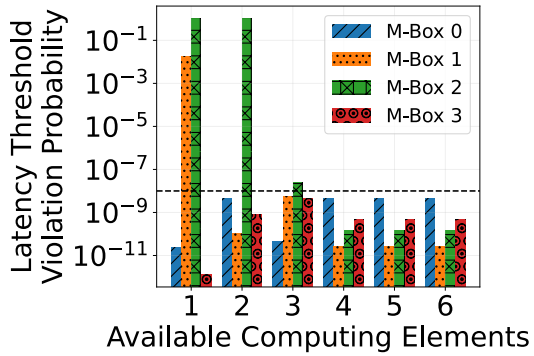
These components are implemented in Python using appropriate data structures



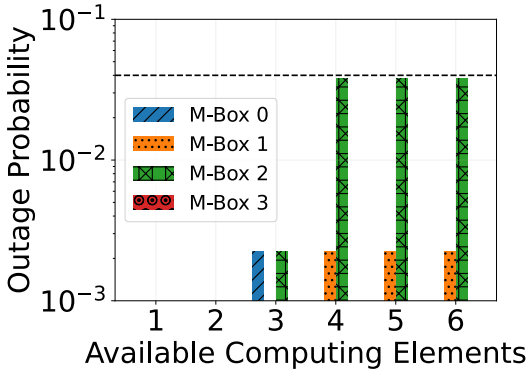
(a) MANTRA Computing Agents Action



(b) MANTRA Offloading Agents Action



(c) MANTRA M-Boxes Latency Threshold Violation Probabilities



(d) MANTRA M-Boxes Outage Probabilities

Figure 3.3: MANTRA Strategies and Performance

and algorithms. They are integrated into a proper interface that allows the user to specify the simulation's parameters, monitor the simulation's progress, and view the results. Table 3.1 summarizes the simulation parameters.

3.6 Numerical Results

MANTRA has been evaluated for different numbers of available CEs in each M-Box to assess various performance aspects. We compare the performance of MANTRA against an ideal *oracle* policy, that is, the optimal policy that could be achieved if the M-Boxes had a perfect knowledge of the system, thus making a fully centralized decision possible. Figure 3.2 shows that the agents achieve near-optimal performance.

More specifically, as visible in Figure 3.2a, the performance of the MANTRA agents steadily approaches the one offered by the oracle. In other words, the rewards achieved by the fully decentralized MANTRA algorithms (solid lines) are almost as efficient as the oracle rewards (dashed lines), and converge after

a relatively short learning period. Notably, the agents do not need to communicate directly with each other and do not possess any knowledge of the system parameters. Note how the total reward is the sum of the computing agent's and offloading agents' rewards. Similarly, in Figure 3.2b, we show the so-called *regret*, which is the difference between the expected reward of the best action, i.e. the oracle action, and the expected reward of the action chosen by the algorithm. It is clearly shown that MANTRA agents' regret quickly approaches zero, proving that the agents can learn almost optimal policies.

Figure 3.3a and 3.3b show the strategies adopted by the M-Boxes for an increasing number of the available CEs. Let us now analyze the behavior of each M-Box. As depicted in Figure 3.3a, M-Box 0 only needs up to two CEs, with the only exception being the case where the number of available computing elements is equal to three. At a first glance, this behavior seems counterintuitive, as the underlying job generation rate due to the vehicular traffic distribution in Area 0 is the second highest in the system. However, in most cases, the amount of jobs offloaded from M-Box 1 to 0 is quite low. Consequentially, two CEs are sufficient to handle the incoming traffic in almost all cases. As already stated, the only exception is represented for three available CEs. In that case, in fact, M-Box 1 exhibits a high offloading probability, and offloads most of the incoming jobs towards M-Box 0. The latter, in turn, reduces its offloading probability, and chooses instead to turn on three CEs to cope with the increased job load.

As regards M-Box 1, let us note that, despite its high offloading probability (see Fig 3.3b), its actual offloading rate is small. The job incoming rate λ_1 is in fact the lowest among all areas, as visible in Table 3.1 On the other hand, we can see how M-Box 1 tends to increase the number of active CEs as the CEs availability increases. In fact, M-Box 1 is the main recipient of the jobs offloaded from M-Box 2. This is due to the high amount of jobs generated by the vehicles in Area 3; as a consequence, M-Box 3 is highly overloaded and, therefore, cannot affort to support M-Box 2.

M-Box 2, on the contrary, has the highest incoming job rate among all the M-Boxes and is therefore highly loaded. Accordingly, this M-Box tends to power the maximum amount of available CEs, up to four CEs. On the other hand, it does not set very high offloading probabilities, as this would inevitably deteriorate the performance of its neighbor nodes.

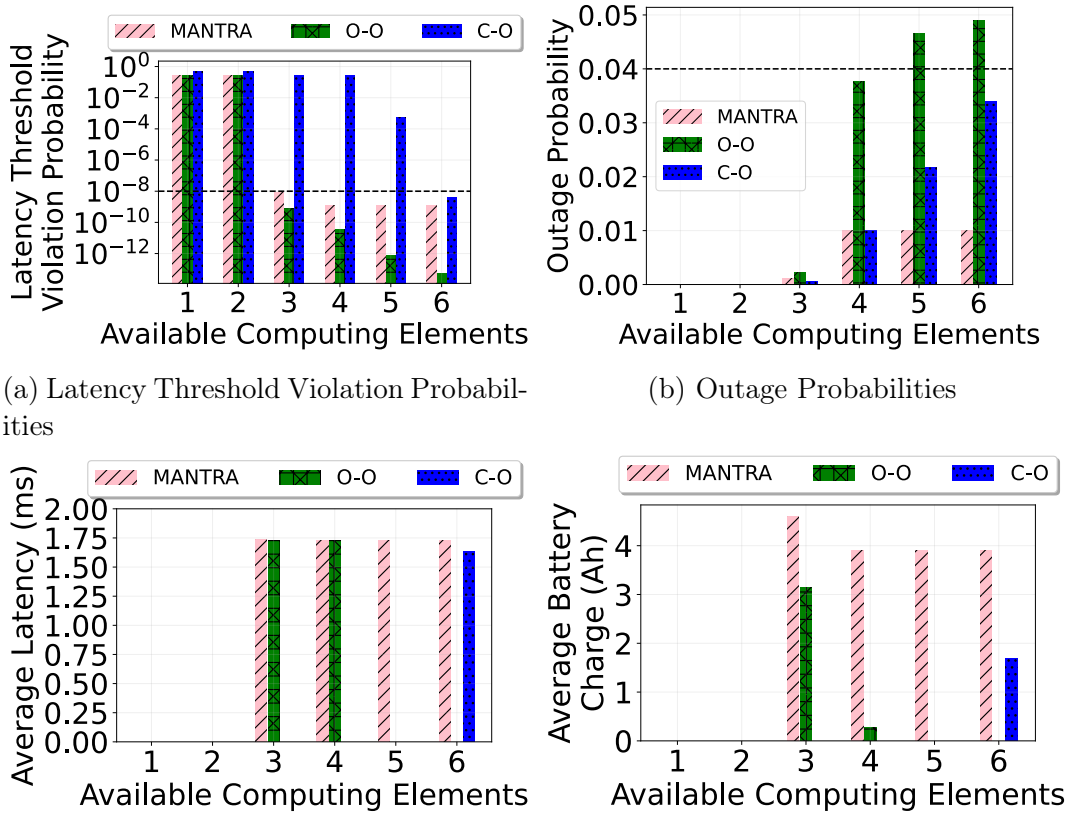
Note how M-Box 3 already has to deal with a high job rate incoming from its area and, for this reason, cannot afford to fully support M-Box 2. The latter, in turn, relies on M-Box 1 to offload most of its jobs.

For what concerns M-Box 3, it powers up to 2 CEs but tends to keep the offloading probabilities high. The reason for this behavior is the following: as already stated, M-Box 2 is overburdened by a high volume of incoming jobs, but tends to turn on a lot of CEs, and can rely on the help of the M-Box with the lowest incoming job rate, M-Box 1. Hence, M-Box 3 offloads most of the incoming jobs to M-Box 2, and can accordingly keep most of its CEs off, while M-Box 2, in turn, further offloads part of those jobs to M-Box 1.

Finally, Figures **??** and 3.3d depict the latency threshold violation probability, $p_{L,m}$, and the M-Boxes battery outage probability, $p_{D,m}$. As expected, the system cannot satisfy the latency requirements of $\epsilon_l = 1 \times 10^{-8}$ when there are less than 4 CEs in each M-Box. This is due to the system being unable to handle the demand without requiring additional hardware. You can also clearly see how the agents are cooperating by the fact that the M-Box with the highest incoming job rate, i.e., M-Box 2, is not always the one with the highest latency threshold violation probability since it is frequently helped by the other M-Boxes. Similarly, the M-Box with the lowest incoming job rate, i.e., M-Box 1, is not always the one with the lowest latency threshold violation probability since it assists other M-Boxes by offering its CEs to process their jobs.

Figure 3.4 depicts the comparison between the performance of MANTRA versus two main baseline heuristics. In particular, the first baseline, *Offloading-only* (O-O), acts over the amount of offloaded jobs while keeping all the available CEs active. Performance improvements over this baseline are achievable only if the CE MANTRA agents learn the appropriate amount of CES to keep active. On the contrary, the second baseline, *Computing-only* (C-O), can dynamically power on and off the available CEs, but does not rely on any offloading procedure and therefore keeps all the processing local instead. The comparison between MANTRA and C-O highlights the capabilities of the MANTRA offloading agents to effectively cooperate with each other, and also demonstrates that, at least in certain scenarios, the latency requirements can only be satisfied by leveraging the offloading procedure.

MANTRA outperforms both baseline strategies, as shown in Figures 3.4a and 3.4b. On the one hand, the first baseline offers generally low latency threshold violation probabilities. It is also characterized by a much higher outage probability, as compared with the other two approaches. MANTRA, instead, is capable of keeping the latency threshold violation probability low with substantially lower energy consumption. On the other hand, the second baseline forces the M-Boxes to power on most of the available CEs, due to the fact that it doesn't exploit any



(a) Latency Threshold Violation Probabilities

(b) Outage Probabilities

(c) Average Latencies

(d) Average Battery Charge

Figure 3.4: MANTRA vs Offloading-Only and Computation-Only

offloading among M-Boxes. This results in both high outage probabilities and high latency threshold violation probabilities, especially in the case of overloaded M-Boxes, such as M-Box 2. MANTRA can instead adequately balance the network load among the available M-Boxes, thus reducing the energy consumption in the network and the latency threshold violation probability. Finally, Figures 3.4c and 3.4d depict the average delay and battery charge of the different strategies. Specifically, for the sake of readability, we choose to show the performance of the strategies only when both the latency and outage requirements are satisfied. For this reason, for instance, the boxes for the case with only one and two computing elements are set to zero, as two CEs are never sufficient to satisfy the latency requirement. Clearly, as soon as the number of available computing elements is above two, MANTRA is the only strategy that is able to consistently satisfy the requirements. On the other hand, O-O is only able to satisfy the requirements in the case of three and four available computing elements. Moreover, while the latency performances of MANTRA and O-O average delays are comparable, the average battery charge in O-O is instead 36% and 90% lower than MANTRA for three and four available computing elements, respectively. Conversely, C-O is

able to meet both the target requirements only when six computing elements are available. However, in this scenario, MANTRA's battery charge is, on average, 57% higher than the one in C-O. This shows how MANTRA agents can effectively reduce the M-Boxes energy consumption

Chapter 4

Latency-aware Network Slicing in O-RAN

The advent of O-RAN architectures has redefined the way cellular networks are organized and deployed. Network slicing within the context of O-RAN holds the promise of delivering tailored services to diverse use cases, each with distinct Service Level Agreements (SLAs). This section focuses on the application of DRL techniques to address the intricate challenge of satisfying different SLAs for network slices in an O-RAN environment. We delve into the complexities of ensuring low latency and optimal resource allocation across multiple network slices.

4.0.1 O-RAN Overview

The forthcoming iterations of cellular networks must possess intrinsic adaptability and adaptiveness to effectively cater to a wide spectrum of application-specific and user-centric demands. To accomplish these aspirations, the future Radio Access Networks must combine three pivotal elements [153]: (i) programmable and virtualized protocol stacks with clearly defined open interfaces; (ii) closed-loop network control mechanisms; and (iii) data-driven modeling and ML. Programmability will facilitate agile RAN adaptation, enabling the provisioning of tailor-made services to cater to the precise requirements of distinct deployments. Closed-loop control will leverage telemetry measurements from the RAN to reconfigure cellular nodes, adapting their behavior to current network conditions and traffic. In tandem, data-driven modeling will leverage recent advances in ML and large-scale data analysis, thereby enabling real-time, closed-loop, and dynamic decision-making processes grounded in techniques like DRL. It is noteworthy that these very principles constitute the foundational principles of the

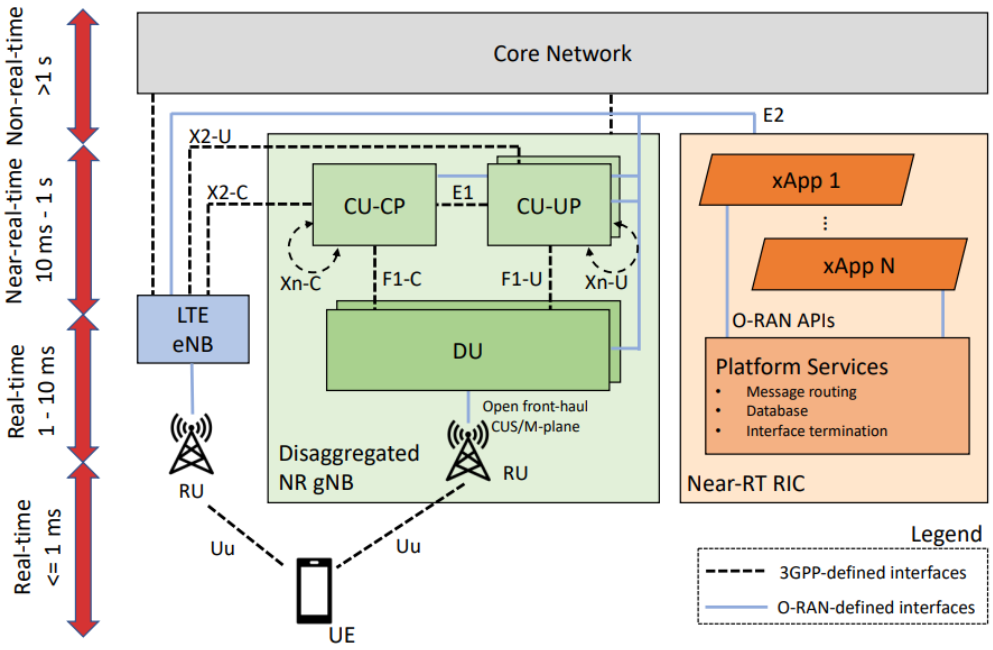


Figure 4.1: O-RAN architecture with the near-RT RIC functions, aside packet core [152]

emergent Open RAN paradigm, which has recently garnered substantial momentum as a pragmatic enabler of algorithmic and hardware innovation in the future of cellular networks [154, 155].

In the pursuit of catalyzing the evolution towards open RAN architectures, the 3rd Generation Partnership Project (3GPP) has standardized the disentanglement of base stations into discrete functional units, as shown in 4.1: the Central Unit (CU), Distributed Unit (DU), and Radio Unit (RU). The O-RAN Alliance, a consortium of industry stakeholders, is actively engaged in standardizing open interfaces that serve to connect the various disaggregated functional units to a shared control overlay, denoted as the RAN Intelligent Controller (RIC). This RIC is capable of executing custom control logic via so-called *xApps*. These efforts will render the monolithic RAN "black-box" obsolete, favoring open, programmable and virtualized solutions that expose status and offer control knobs through standardized interfaces [153].

However, while developing new software solutions for open architectures might be considerably easier than before, demonstrating their effectiveness, efficiency, reliability, and robustness in a host of varying scenarios becomes a necessity. This imperative arises from the essential need to safeguard the operational performance, stability, and security of both the network itself and the services dispensed to a vast user base. It is therefore imperative for Telecommunications Operators

(TOs) to require that all networks algorithms and software components are extensively tested prior to actual deployment on the commercial infrastructure.

In the capacity of infrastructure owners, Telecommunications Operators theoretically possess the prerogative to subject new software solutions to experimentation within the established network framework. However, this decision is fraught with complexities and cost implications, as the deployment of novel solutions within the commercial network introduces the potential for inadvertent and unfavorable behavioral consequences, leading to unanticipated service disruptions and financial ramifications. As an alternative, new solutions could be first tested in smaller laboratory setups. These trials, however, can only capture a limited number of radio frequency (RF) scenarios and would merely model small-scale deployment configurations, limiting their effectiveness and extent.

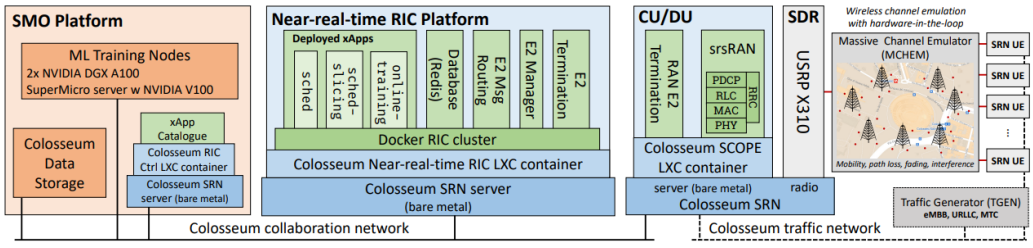


Figure 4.2: Integration of the O-RAN infrastructure in Colosseum [27]

Consequently, the imperative arises for methodical and expansive testing on a broader scale. Notably, endeavors in the realm of wireless testing at scale are progressively advancing. Initiatives such as the Colosseum, the world's largest network emulator [156], provides researchers with testing at scale through a fully controlled, programmable, and observable environment with hardware in the loop. Colosseum comprises a substantial matrix of 256-by-256-channel RF simulation, incorporating programmable Software Defined Radios (SDRs) that facilitate the emulation of end-to-end communications inclusive of abundant computational capabilities. In Colosseum, each node, or Standard Radio Node (SRN), consists of a GPU-endowed server connected to one USRP X310. SRNs are fully programmable and serve as virtualized environments running LXC. This makes it possible to use them as either compute-only (e.g., edge or cloud servers) or compute-and-transmit (e.g., UE or BS) nodes. A high-level diagram showing how O-RAN is integrated inside the Colosseum emulator is shown in Fig. 4.2

Intelligent, dynamic network optimization via xApps is clearly a key enabler for future network automation. However, it also introduces novel practical challenges concerning, for instance, the deployment of data-driven ML control solutions at

scale. Domain-specific challenges stem from considering the constraints of standardized RANs, the very nature of the wireless ecosystem, and the complex interplay among different elements of the networking stack. These challenges, all yet to be addressed in practical RAN deployments, include:

1. *Collecting datasets at scale*: Datasets for ML training at scale need to be collected and curated to accurately represent the intrinsic randomness and behavior of real-world RANs

2. *Designing ML agents capable of generalizing*: Agents should be able to generalize and adapt to unseen deployment configurations not part of the training set.

3. *Selecting meaningful features*: Features should be accurately selected to provide a meaningful representation of the network status without incurring dimensionality issues.

To address these key challenges, in this research, we describe the design of DRL-based xApps for closed-loop control in O-RAN with the objective of satisfying different SLAs for different network slices while also minimizing the amount of PRBs to be allocated to do so.

We develop an xApp for closed-loop control of RAN slicing policies. We propose an innovative xApp design based on the combination of a unified interface to the near-real-time RIC for data and control messaging and a data-driven unit with the DRL agent. This simplifies the design and prototyping of xApps, which share the same interface but are equipped with different intelligent logic. We train the agents over a 1.5 GB dataset with more than 20 hours of live RAN performance traces. The results of our large-scale experimental evaluation include new understandings of data analysis, feature selection, and modeling of control actions for DRL agents, and insights on design strategies to train ML algorithms that generalize over different SLAs.

4.0.2 Related Work

The concept of Network Slicing has gained significant attention in recent years as a pivotal enabler for the efficient deployment of 5G and beyond networks. In the context of O-RAN, the realization of network slicing holds paramount importance due to its potential to revolutionize how mobile networks are designed, deployed, and managed. This section provides an overview of the existing research and developments pertaining to network slicing within the O-RAN ecosystem.

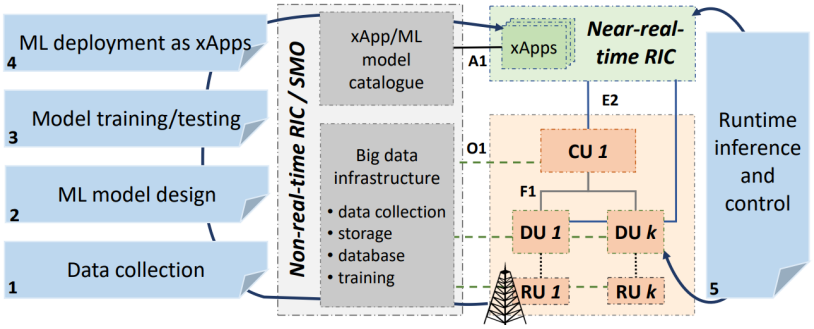


Figure 4.3: The O-RAN architecture and the workflow for the design, development and deployment of ML applications [27]

The foundation for network slicing within O-RAN builds upon earlier research in 5G and future-generation networks. Researchers have extensively explored the fundamental principles and architectural frameworks for network slicing. Several surveys discuss research directions and challenges on network slicing, discussing various aspects, including architecture, orchestration, and resource management [157, 158]. These studies serve as valuable references for understanding the broader context of network slicing deployment in O-RAN.

The O-RAN Alliance has played a pivotal role in advancing the adoption of open and intelligent RAN technologies. Their technical specifications and whitepapers provide critical insights into the integration of network slicing within the O-RAN architecture. Several O-RAN Alliance documents outline how network slicing is envisaged as a part of the O-RAN framework. It highlights the importance of flexibility, scalability, and adaptability of slices to meet diverse service requirements.

Effective orchestration and management of network slices are key challenges in realizing the potential of O-RAN network slicing. Several works have focused on orchestration frameworks and methodologies. Several novel approaches like the one in [159] have been proposed to orchestrate and manage network slices in multi-vendor O-RAN environments efficiently. These studies emphasize the need for standardized interfaces and intelligent orchestration algorithms to optimize resource allocation.

Ensuring the security and isolation of network slices is a critical concern in O-RAN network slicing. Some research [160, 161] addresses security challenges associated with network slicing, proposing mechanisms to protect slice instances from unauthorized access and malicious attacks. These studies are essential for establishing the trustworthiness of O-RAN network slicing.

Evaluating the performance of network slices and optimizing resource allocation

is crucial for delivering high-quality services. Several studies, such as in [162, 163], have focused on performance evaluation metrics and optimization techniques for O-RAN network slicing, enabling efficient resource utilization and QoS assurance.

4.0.3 RIC Data collection and training

The O-RAN specifications include guidelines for the management of ML models in cellular networks. The specifications describe the ML workflow for O-RAN through five steps (Fig. 4.3): (1) data collection; (2) model design; (3) model training and testing; (4) model deployment as xApp, and (5) runtime inference and control. First, data are collected for different configurations and setups of the RAN, e.g., large/small scale, different traffic (step 1). Data are generated by the RAN nodes, i.e., CUs, DUs, and RUs, and streamed to the non-real-time RIC through the O1 interface, where it is organized in large datasets. After enough data have been collected, an ML model is designed (step 2). This entails the following: (i) identifying the RAN parameters to input to the model (e.g., throughput, latency, etc.); (ii) identifying the RAN parameters to control as output (e.g., RAN slicing and scheduling policies); and (iii) the actual ML algorithm implementation. Once the model has been designed and implemented, it is trained and tested on the collected data (step 3). This involves selecting the model hyperparameters (e.g., the depth and number of layers of the neural network) and training the model on a portion of the collected data until a (satisfactory) level of convergence of the model has been reached. After the model has been trained, it is tested on an unseen portion of the collected data to verify that it is able to generalize and react to potentially unforeseen situations. Then, the model is packaged into an xApp ready to run on the near-real-time RIC (step 4). After the xApp has been created, it is deployed on the O-RAN infrastructure. In this phase, the model is first stored in the xApp catalog of the non-real-time RIC and then instantiated on demand on the near-real-time RIC, where it is interfaced with the RAN through the E2 interface to perform runtime inference and control based on the current network conditions (step 5).

In this research, we focus on steps 1 to 3 for the design and development an xApp able to satisfy specific SLA requirements in terms of latency.

4.1 System Model

In our system, we consider a radio access network consisting of a set of BSs, denoted as B, and a set of UEs for each BS, denoted as U_B that send data

through the BSs. Each BS b has capacity C_b, i.e., a discrete number of physical resource blocks (PRBs) with a fixed bandwidth. Moreover, in each BS there are deployed several network slices, and we denote the set of network slices as I. The available resources are divided into subsets of PRBs and dynamically allocated to each network slice in accordance with their real-time traffic demands and SLA requirements. Let us assume a system that operates in time slots, each representing a "decision epoch", denoted by $n \in \mathbf{N} = 1, 2, ..., N$. In this system, decisions about PRB allocation can only be made at the start of each decision interval. The length of each decision interval, denoted by ϵ, can be determined based on the policies of the infrastructure provider and can range from a few seconds to several minutes.

We see the allocation of radio resources to end-users as a two-step process, as described in [164]. First, once network slices are admitted into the system, the infrastructure provider schedules the assignment of radio resource slots for each tenant. Then, based on the available resources, each tenant may choose to implement their own scheduling solutions for their end-users based on their specific needs and requirements. In light of the various user-to-base station associations and scheduling algorithms available for allocating resources to end users, our focus is on correctly and fairly dimensioning the allocation of resources between slices rather than addressing the issue of intra-slice scheduling.

In each BS, the optimal amount of PRBs to allocate to each slice in $i \in I$ is chosen by the RIC. In order to optimize the number of PRBs to allocate to each slice, the RIC learns the optimal policies using an RL approach. In RL, an MDP is used to provide a framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker.

To achieve this goal, we utilize the variable a $a^{(b,i)}(n)$ to indicate the decision regarding the allocation of PRBs to the i-th slice under the b-th BS during the n-th decision time interval. Altogether, we formulate the local optimization task of satisfying some SLA requirements in terms of latency inside BS b for slice i as:

$$\min \sum_{n=1}^{N} \left[\sum_{i \in I} d^{(b,i)}(n) \right] \tag{4.1}$$

which is subject to the capacity constraint, which means that the sum of allocated PRBs cannot exceed the total capacity:

$$\sum_{i \in I} a^{(b,i)}(n) \leq C_b \tag{4.2}$$

We also want to make it so that each slice incurred latency is less or equal to the latency requirements of that slice, Λ_i.

The abovementioned optimization task can be solved by invoking by means of DRL. Therefore, need to frame the problem as a specific class of MDP, called

Partially Observable Markov Decision Processes (POMDP), which do not assume that the agent can observe the environment's state perfectly. A POMDP is formally a (S, O, A, T, R, γ) tuple in which S is the state space (not fully observable by the agent)), O is the observation space (what the agent sees from the environment in order to execute actions), A is the action space, T is the set of transition probabilities among states (can be unknown), R is the reward function and γ is the discount factor.

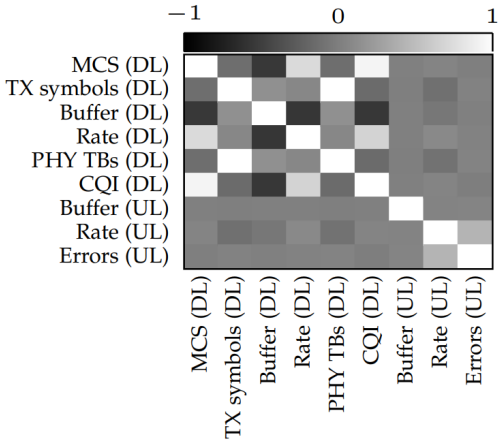


Figure 4.4: Correlation analysis of several UE-specific metrics

O-RAN does not indicate which KPMs should be considered for the design of ML algorithms. The O-RAN E2SM KPM specifications allow the generation of more than 400 possible KPMs, listed in [165]. These KPMs range from physical layer metrics to base station monitoring statistics. Therefore, the bulk set of data may not be useful to represent the network state for a specific problem. Additionally, reporting or collecting all the metrics via the E2 (fronthaul interface between the RU and the DU) or O1 (control and management interface in which the interactions between various components within the O-RAN architecture occur) interfaces introduces a high overhead and a high dimensional input may lead to suboptimal performance for ML-driven xApps.

To choose the most valuable metrics, it is necessary to analyze the metrics through a correlation analysis [27], as shown in Fig. 4.4. Specifically, while downlink and uplink metrics exhibit a low correlation, most downlink KPMs positively or negatively correlate with each other (the same holds for uplink KPMs). For

example, the downlink Modulation and Coding Scheme (MCS) and buffer occupancy have a negative correlation. Similarly, the number of Transport Blocks (TBs) and symbols in the downlink have a strong positive correlation. Two downlink metrics that do not correlate well, instead, are the number of TBs and the buffer occupancy. Indeed, the amount of data transmitted in each TB varies with the MCS and, therefore, cannot be used as an indicator of how much the buffer will empty after each transmission.

To summarize, since the number of downlink symbols and TBs, or the MCS and the buffer occupancy, are highly correlated, using them to represent the state of the network only increases the dimensionality of the state without introducing additional information. Conversely, the buffer occupancy and the number of TBs enrich the representation with low redundancy. Therefore, the DRL agents for the xApps in this research consider as input metrics, for each slice i, the number of TBs $tb^{(b,i)}$, the ratio of PRB granted and requested $rt^{(b)}$, and the downlink rate $dl^{(b,i)}$. Moreover, since the objective of the agent is to satisfy some latency requirements, we also include the minimum, maximum, and average latency of the packets for all the UEs of BS b, $d^{(b,i)}_{min}, d^{(b,i)}_{max}, d^{(b,i)}_{mean}$. These metrics are periodically sent from the BSs and UEs to the RIC before the start of a new decision interval t. More information on how these metrics have been collected and processed before being fed to the DRL agent will be discussed in the next section.

Each time the agent has to make a decision for a BS $b \in B$ and a slice $i \in I$, a new *decision epoch* n is triggered. The agent *observation* $o^{(e,b,i)} \in O$ at decision epoch n is defined as:

$$o^{(e,b,i)}(n) = \left[tb^{(b,i)}(n), rt^{(b,i)}(n), dl^{(b,i)}(n), d^{(b,i)}_{min}(n), d^{(b,i)}_{max}(n), d^{(b,i)}_{mean}(n)\right] \tag{4.3}$$

For each decision epoch, the agent *action* consists in choosing a new PRB value for each slice and for each base station, $a_{b,i}(n)$. Each BS $b \in B$ uses this value for the whole duration of the epoch.

Since the agent in the system is model-free, the *transition probabilities* of the environment state are unknown to the agent. This reflects what happens in real network scenarios and therefore allows our agent to be deployed in real networks. Finally, recalling that the objective of the system is to satisfy the SLAs in terms of latency while also ensuring that the minimum amount of PRBs required to do so is allocated, we define φ_{sla} as the ratio of packets whose latency is less than Λ, and corresponds to the target ratio the agent needs to maintain in order to satisfy the SLA requirements. For example, $\varphi_{sla} = 0.99$ means that 99% of the packets need to have their latency less than a predefined latency threshold. Moreover,

we define the actual ratio of packets that satisfy the SLA as φ_{env}. The latency threshold to be satisfied is Λ_i.

With all of this in mind, the *reward* for slice i and BS b is defined as:

$$r_{i,b}(n) = \begin{cases} \dfrac{1}{1 + e^{k \cdot (\varphi_{sla}(n) - \varphi_{env}(n))}} & \text{if } \varphi_{sla}(n) > \varphi_{env}(n) \\ \dfrac{1}{1 + e^{k \cdot (\varphi_{sla}(n) - \varphi_{env}(n))}} + \left(1 - \dfrac{a^{(b,i)(n)}}{C_b}\right) & \text{otherwise} \end{cases} \tag{4.4}$$

We use a sigmoid-like function of the difference between the SLA requirements, φ_{sla}, and the actual ratio of packets that satisfy the SLA, i.e., φ_{env}. This allows the agent to learn how to satisfy the SLA first, no matter the number of PRBs used in the process. Then, as soon as the agents learn how to satisfy the SLA, the agent is rewarded even more if he is able to allocate fewer PRBs while still satisfying the latency requirements. To normalize the components of the reward function between 0 and 1, we divide the number of PRBs allocated by the maximum amount of PRBs that can be allocated.

To summarize, at the beginning of each *decision epoch*, the agent observes, for each BS, a set of relevant metrics collected through the E2 and O1 interface, then chooses an action, transmits the chosen amount of PRBs for each slice to the BSs, and then receives the reward right before the start of the next decision epoch.

4.2 Experiment Setup

The system architecture consists of a distributed multi-agent optimization framework. A DRL agent runs on each cellular BS and makes control decisions on its slicing policies. Fig. 4.5 shows the architecture of the DRL-based framework. This is formed by three components: (i) The agent; (ii) the BS protocol stack, and (iii) the BS connector.

Each *agent* is equipped with the DRL implementation of a state-of-the-art algorithm like [31], a Data Processing module, and a Reward Calculator module. Periodically the agent receives MGEN reports from the UEs (step 1). Then, at each decision epoch t, these reports, together with other metrics collected from the O1 and E2 interfaces, are processed from the Data Processing module to obtain the observation of the DRL agent (steps 2 and 3), which reflects the performance of the BS on which the agent is deployed, and of the UEs served by it. Then (step 4), the DRL network(s) outputs the action to be executed in the BS

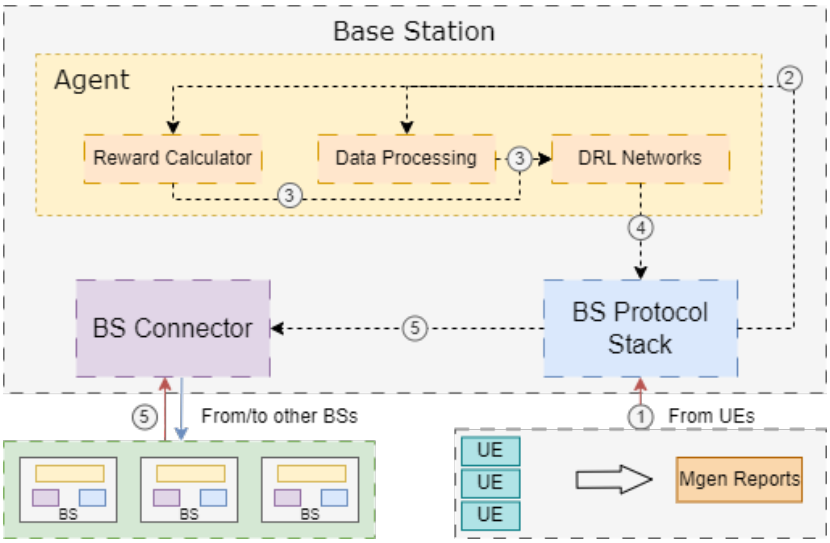


Figure 4.5: The System Architecture

(i.e., the number of PRBs to allocate for each slice). The DRL tuple, made up of the observation, action, and reward *can* periodically forwarded to the other BSs (step 5). This can be useful if the DRL agents are required to be retrained on fresh data coming from multiple BSs.

The *BS connector* allows communication between the BSs of the network, while The *BS protocol stack* implements the softwarized cellular BS, including layers such as PHY, MAC, RLC, PDCP, and RRC. This enables communication with the UEs of the network and can be implemented through open-source software solutions, as discussed in the next section. Regardless of the specific implementation, this element provides the network data that is fed to the Data Processing and Reward Calculator modules to obtain the current observation (from decision epoch n) and the last reward (from the previous decision epoch $n-1$).

4.2.1 SCOPE: a Softwarized Cellular Open Prototyping Environment

In our research, both BSs and UEs have been run inside Colosseum as customized SCOPE containers. SCOPE [166] is an accessible and software-driven prototyping platform designed for the advancement of Next Generation (NextG) systems. It encompasses (i) a ready-to-use transportable open-source container, facilitating the instantiation of programmable cellular network elements such as base stations and user devices; (ii) an emulation module that accommodates a diverse array of authentic real-world deployments, channels, and traffic conditions, thereby enabling the rigorous evaluation of novel solutions; (iii) a data collection module

tailored to the support of applications rooted in artificial intelligence and machine learning paradigms; and (iv) a suite of open Application Programming Interfaces (APIs) which endow users with the capability to exercise real-time control over network element functionalities.

In particular, SCOPE includes an open-source implementation of a 3GPP-compliant softwarized cellular BS, which includes capabilities such as RAN slicing and Medium Access Control (MAC) and Physical (PHY)-layer functions. To make SCOPE platform-independent, Scope is developed as ready-to-deploy Linux Containers (LXCs).

The SCOPE network slicing implementation facilitates the concurrent existence of multiple distinct slices, each meticulously tailored to accommodate specific traffic classes and User Equipments (UEs) within a shared infrastructure. Our implementation empowers the division of the available spectrum at each Base Station (BS) and exercises authority over the allocation of resources for each individual network slice. Slicing is implemented in the SCOPE framework by applying PRB masks during the scheduling process, and it is possible to control the number of PRBs for each slice. As slices represent a specific type of service the operators agree to provide to their subscribers (e.g., as part of SLAs), they are pre-instantiated on the base stations, and users are statically assigned to one of such slices based on the purchased service level.

SCOPE also includes a data collection module for automatically recording the performance of the network. Network run-time metrics are saved in CSV format in the metrics and performance dataset. These can either be queried at run time (e.g., using it as a feedback loop) or downloaded at the end of the experiment to refine the control logic. Detailed statistics (e.g., throughput, MCS, buffer size, slice PRBs) on the performance of each BS and UE are periodically logged by the SCOPE data collection module and stored in a CSV-formatted dataset. These metrics have been used, in conjunctions with the packet-related metrics collected from MGEN reports (which will be discussed in the next section), to create the dataset used to train the agents.

4.2.2 Cellular Scenarios in Colosseum

During the experiment configuration, we set up the experiment to run on Colosseum by selecting the desired RF and traffic scenarios and the duration of the experiment. This means specifying which nodes act as BSs and which as UEs, and creating a customized instance of SCOPE with user-defined control logic. Once the experiment begins, the user-customized SCOPE container is automatically

deployed on the corresponding Software Radio Node (SRN), and SCOPE RF and traffic scenarios of choice are executed by Colosseum Massive Channel Emulator (MCHEM) and Traffic Generator (TGEN).

Each Colosseum scenario consists of two macroblocks: he RF scenario and the traffic scenario.

The RF scenario specifies the channel conditions that each node experiences in the experiment. For each SRN, the scenario defines channel impulse responses that model path loss, fading, and multipath effects. These channel coefficients are updated every millisecond and can be generated in different ways. Colosseum supports scenarios with channel coefficients generated via analytical models, ray tracing software, or obtained via real-world measurements/channel sounders. Coefficients are then fed to MCHEM, which applies the corresponding channel taps to signals to/from each SRN.

The Traffic scenario specifies and configures the traffic flows among BSs and UEs. Traffic scenarios are handled by the Colosseum TGEN, which is built on top of Multi-Generator (MGEN), an open-source software that generates and controls realistic TCP/UDP traffic [167]. MGEN supports a variety of different classes of traffic with diverse QoS requirements, probability distributions, data rates, and types of service. MGEN is the tool that also allowed us to collect packet-related metrics in order to calculate the latencies.

4.2.3 Cellular Scenario

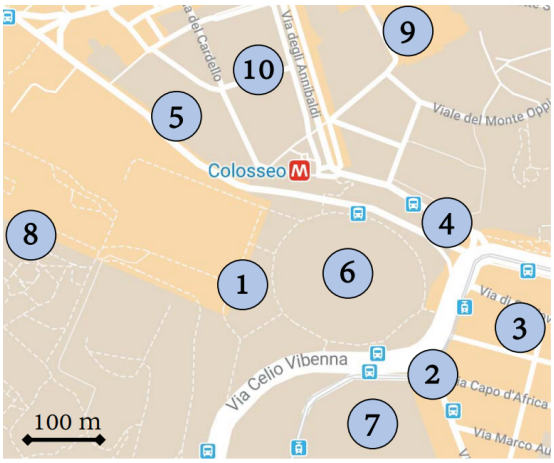


Figure 4.6: Rome cellular scenario map.

In this section, we give an overview of the SCOPE cellular scenario from which the metrics used to train the DRL agents have been collected. In particular,

SCOPE contains the Rome urban setup (as shown in Fig. 4.6), in which the locations of the BSs reflect real cell tower deployments extracted from the OpenCelliD [168] database. The numbered blue circles in the figure mark the locations of the BSs on the map

The *Rome scenario* captures the dynamics of the city center of Rome, Italy. A total of 50 nodes are involved: 10 BSs and 40 UEs. This is the densest Colosseum scenario, and it covers an area of 0.5 km^2. We leveraged TGEN and MGEN to generate dedicated traffic scenarios that model uplink and downlink video streaming traffic flows among UEs and BSs.

We consider the default SCOPE configuration where all UEs generate the same type of traffic, belong to the same slice, and are served via a round-robin scheduling algorithm.

We considered a network with two different slices. The agent is required to select how many PRBs to allocate to each slice. The actions taken by the agent are then enforced via SCOPE, which reconfigures the BSs in real-time. The state of the agent is generated by periodically reading the dataset entries corresponding to the most recent 250ms of the experiment.

To train the DRL agents, we performed large-scale data collection experiments on Colosseum. The large-scale RF scenario mimics a real-world cellular deployment in downtown Rome, Italy, with the positions of the BSs derived from the OpenCelliD database. We instantiated a softwarized cellular network with ten base stations through the SCOPE framework. Each base station operates on a 10 MHz channel (50 PRBs) which can be dynamically assigned to the two slices. We also considered a uniform traffic scenario in which the bitrate is 1.5 Mbit/s for all users. Finally, the base stations serve a total of 100 users equally divided among the two slices. In our data collection campaign, we gathered 1.5 GB of data, for a total of more than 20 hours of experiments. In each experiment, the BSs (and UEs through the BSs) periodically report RAN KPMs to the non-real-time RIC. The complete dataset features more than 30 metrics that can be used for RAN analysis and ML training. Following O-RAN specifications, training is performed offline on the dataset. In our case, this is achieved by randomly selecting instances in which the network reaches the state s_1 that results from the combination of the previous state s_0 and the action to explore a_0.

Table 4.1: Action Index to PRBs allocation

Action Index	Number of PRBs
0	6
1	9
2	12
3	15
4	18
5	21
6	24
7	27
8	30

4.3 Numerical Results

In this section, we show some preliminary results. Our agents have been evaluated for different slices with different SLA requirements in terms of latency in order to assess various performance aspects. Specifically, we consider two slices, with $\Lambda_1 = 200ms$ and $\Lambda_2 = 100ms$ respectively. We consider two different scenarios: one with $\varphi_1 = \varphi_2 = 0.90$ and the second with $\varphi_1 = \varphi_2 = 0.99$. The training of each agent lasted about 13 hours on an NVIDIA V100 32GB. We show that the DRL agents are able to converge to good policies and satisfy the SLA requirements. In the following figures, we show some histograms representing the actions' id taken by the DRL agents to clearly understand what each agent is doing and to what policy it converges, if any. Specifically, each action index corresponds to a specific amount of PRBs that are then allocated, as shown in Table 4.1

We first show the convergence of the agent for the first network slice, with latency requirements $\Lambda = 200ms$ and $\varphi = 0.9$. Recall that, at the beginning of the training, the agent is initialized with random weights. Figure 4.7 reports the evolution of the distribution of the actions chosen by the DRL agent for the Colosseum offline training. During training, the distribution of the actions evolves from uniform (in yellow) to more skewed, multi-modal distributions at the end of the offline training (in red). This means that actions computed in the first few epochs are taken randomly and are generally sub-optimal. As the training goes on, the agent learns how to select strategies that satisfy the slice-specific SLA requirements effectively (in this scenario, the action with a lower index seems to be the one that gives good results). After 8000 episodes of training, for instance, the

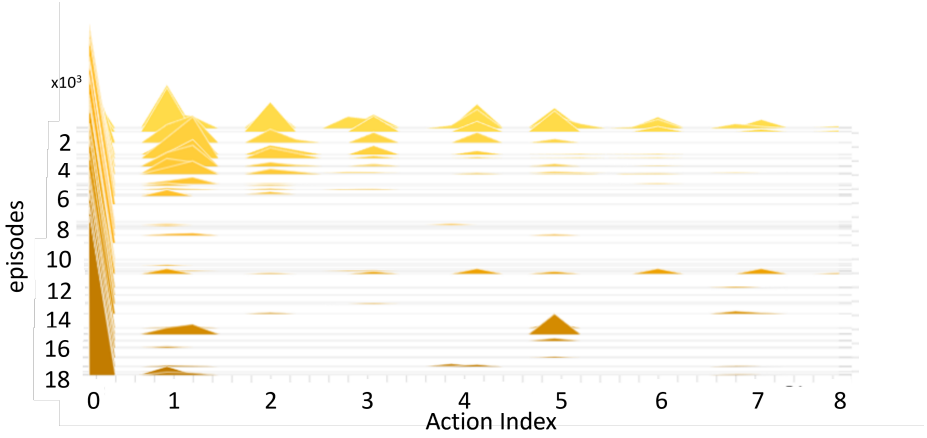


Figure 4.7: Distribution of the actions during the training ($\Lambda_1 = 200ms$, $\varphi_1 = 0.9$)

agent is already capable of selecting actions that result in improved performance and is prone to not choosing actions with a higher index (which corresponds to actions that allocate a higher amount of PRBs).

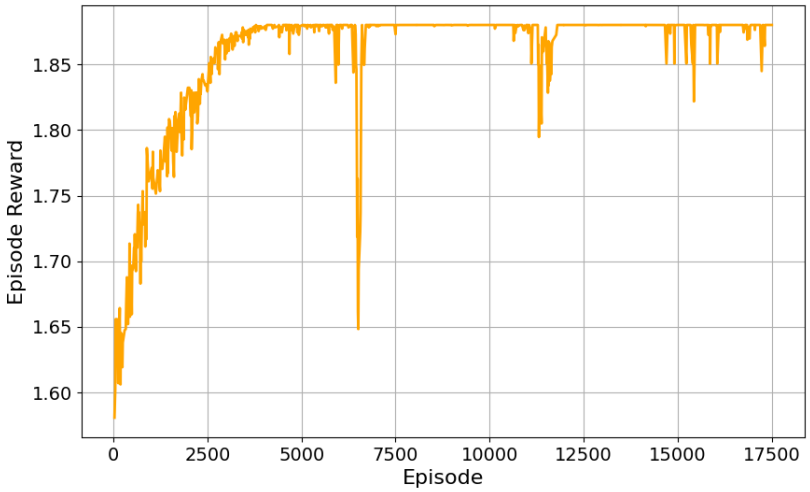


Figure 4.8: Convergence of the Episode Reward ($\Lambda_1 = 200ms$, $\varphi_1 = 0.9$)

Figures 4.8 and 4.9 show how quickly the agent is able to converge. Specifically, Fig. 4.9 reports the entropy regularization loss as a function of the training step of the agent. This metric correlates with the convergence of the training process: the smaller the absolute value of the entropy, the more likely the agent has converged to a set of actions that maximize the long-term reward [169]. We stop the training when this metric (and the mean reward, Fig. 4.10) plateaus.

We now show some results for the other network slice, i.e., the one with $\Lambda = 100ms$ and $\varphi = 0.9$. This has been a much harder task to accomplish since the

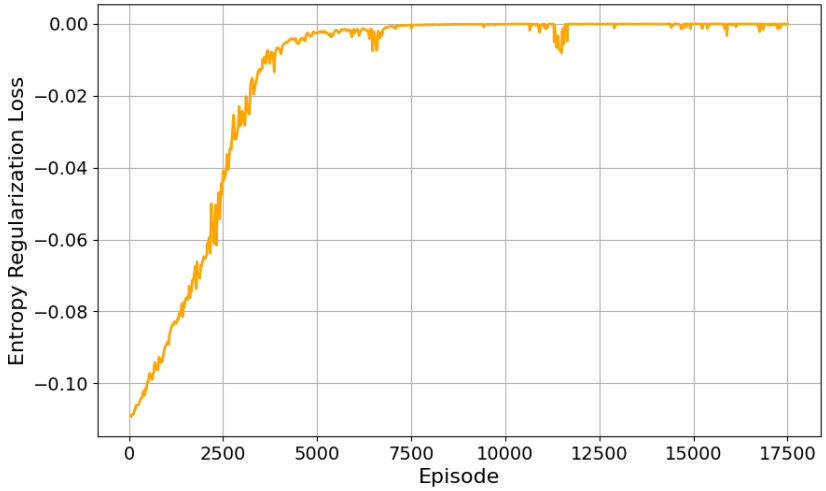


Figure 4.9: Convergence of the Entropy Regularization Loss ($\Lambda_1 = 200ms$, $\varphi_1 = 0.9$)

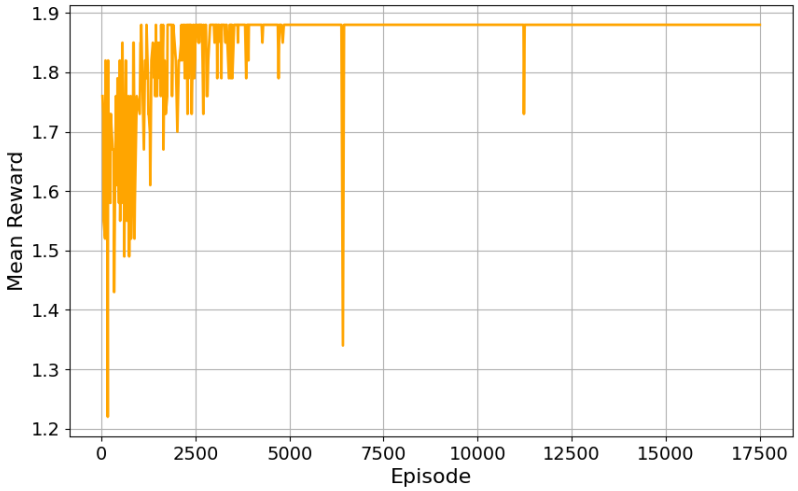


Figure 4.10: Mean Reward ($\Lambda_1 = 200ms$, $\varphi_1 = 0.9$)

latency requirements are much more strict than the first slice.

Fig. 4.11 shows the episode reward. This clearly shows that, compared to the first network slice, the agent is much slower at finding a good policy. In fact, after a few good episodes during the first 1000 episodes (in which the agent policy is still mostly random but apparently achieved somewhat good results), the agent episode reward stagnates in the range 0 from to 500 for about 12000 episodes. Then, starting at episode 16000 the episode reward starts to increase rapidly. This coincides with the agents being able to satisfy the SLA constraints, as shown in Fig. 4.12, where it is clear that the ratio of packets that doesn't satisfy the latency

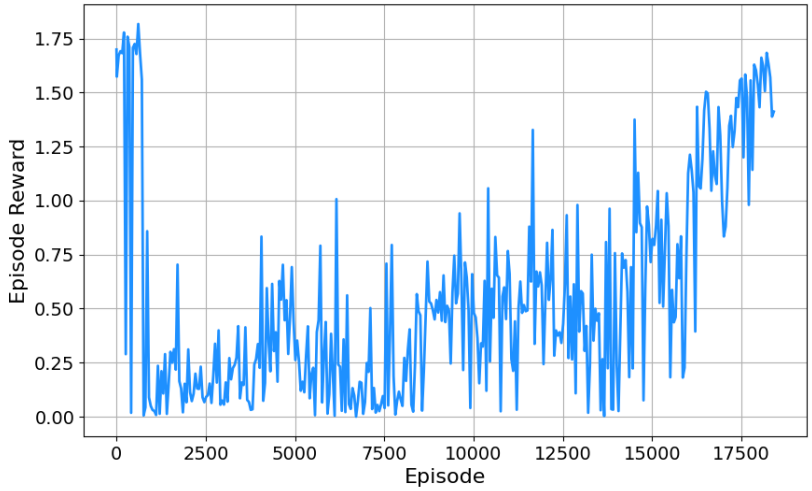


Figure 4.11: Episode Reward ($\Lambda_2 = 100ms$, $\varphi_2 = 0.9$)

requirements, i.e., $1 - \varphi$, is less than the requirements, φ_2.

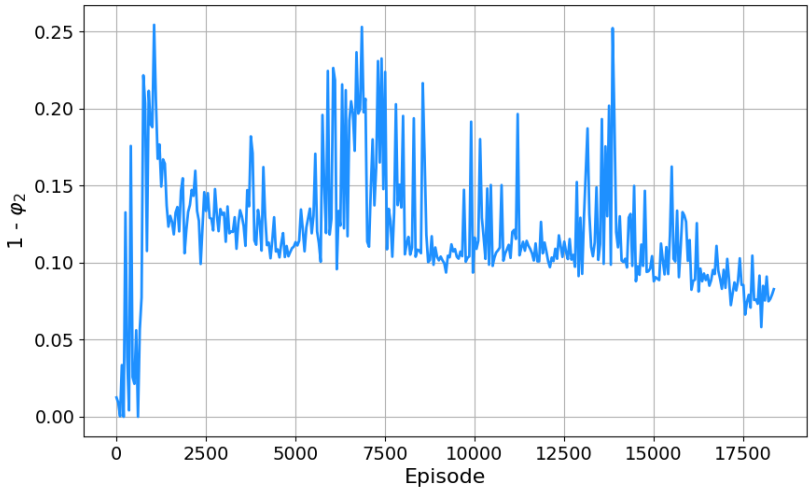


Figure 4.12: Ratio of packets that do not satisfy the latency requirements ($\Lambda_2 = 100ms$, $\varphi_2 = 0.9$)

Accordingly, notice how the action distribution, shown in Fig. 4.13 is much more complex than the one of the first slice, shown in Fig. 4.7. This reflects the fact that achieving the target latency requirements is much more complicated in this second scenario and that the PRB allocation is required to change often (in contrast to the first slice in which one specific PRB allocation, to which the policy converged, seemed to be enough to satisfy the requirements).

We now show the results of the training phase when the SLA requirements

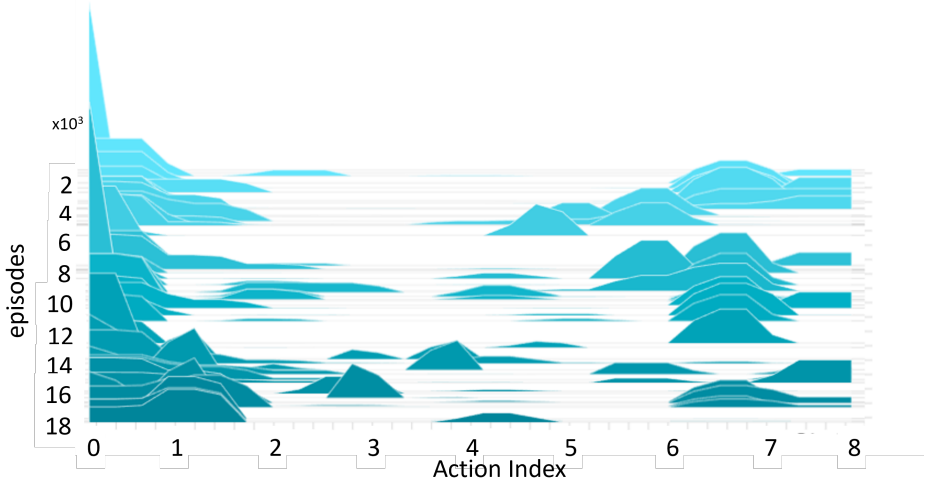


Figure 4.13: Distribution of the actions during the training ($\Lambda_2 = 100ms$, $\varphi_1 = 0.9$)

decrease from $\varphi_1 = \varphi_2 = 0.9$ to $\varphi_1 = \varphi_2 = 0.99$.

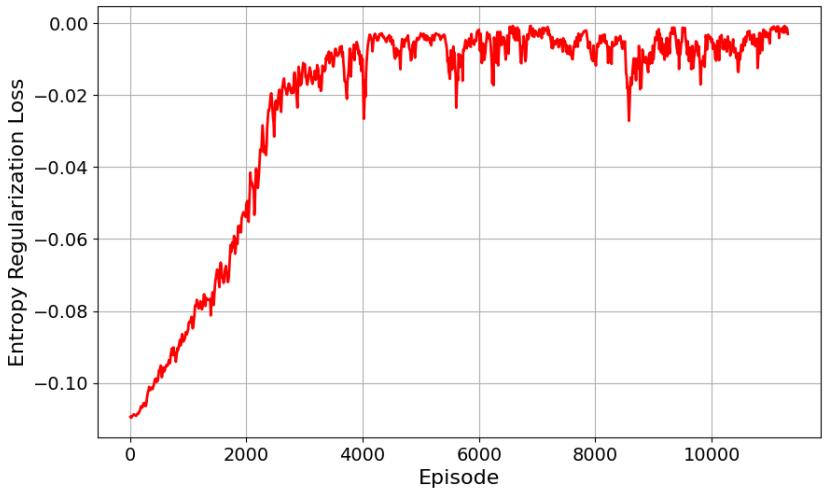


Figure 4.14: Convergence of the Entropy Regularization Loss ($\Lambda_1 = 200ms$, $\varphi_1 = 0.99$)

For both network slices, both the entropy regularization loss and episode reward converge to good values, as shown in Figs. 4.14 and 4.15. As expected, the training process for the second slice seems to be more unstable than for the first slice.

Fig. 4.17 shows the distribution of actions during the training process for the second slice. Compared to Fig. 4.13, the policy that the agent learns is more focused towards a lower amount of PRBs right from the start of the training process, while in Fig. 4.13 a higher amount of PRBs has been allocated even

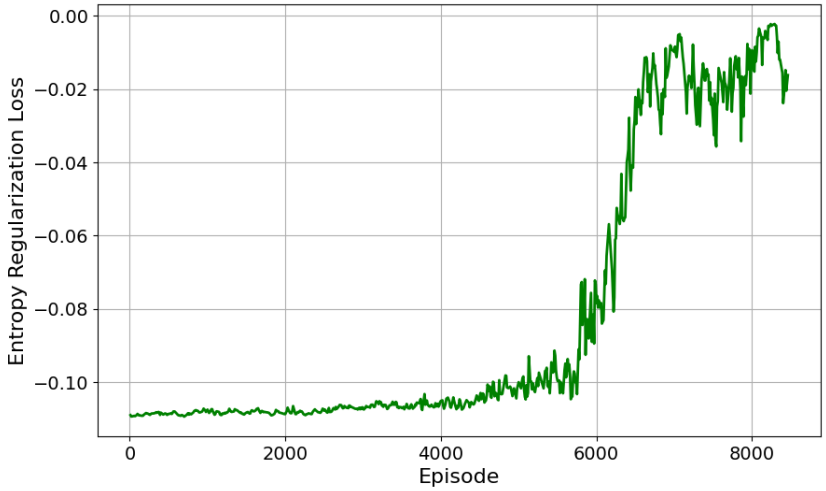


Figure 4.15: Convergence of the Entropy Regularization Loss ($\Lambda_1 = 100ms$, $\varphi_1 = 0.99$)

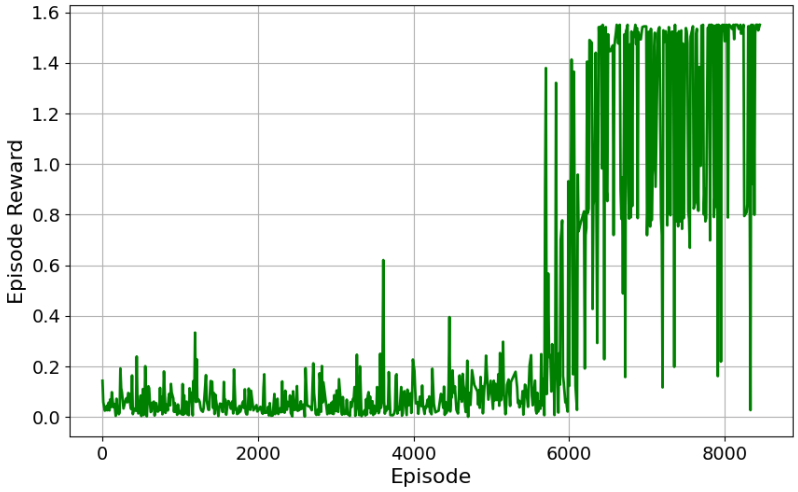


Figure 4.16: Convergence of the Episode Reward ($\Lambda_1 = 100ms$, $\varphi_1 = 0.99$)

during later stages of the training.

However, Fig 4.18 clearly shows that the SLA requirements are met even for the second slice after around 8000 training episodes, after which the training is stopped. This means that the agent is able to satisfy the requirements even though the training phase is much more unstable when faced with harder constraints.

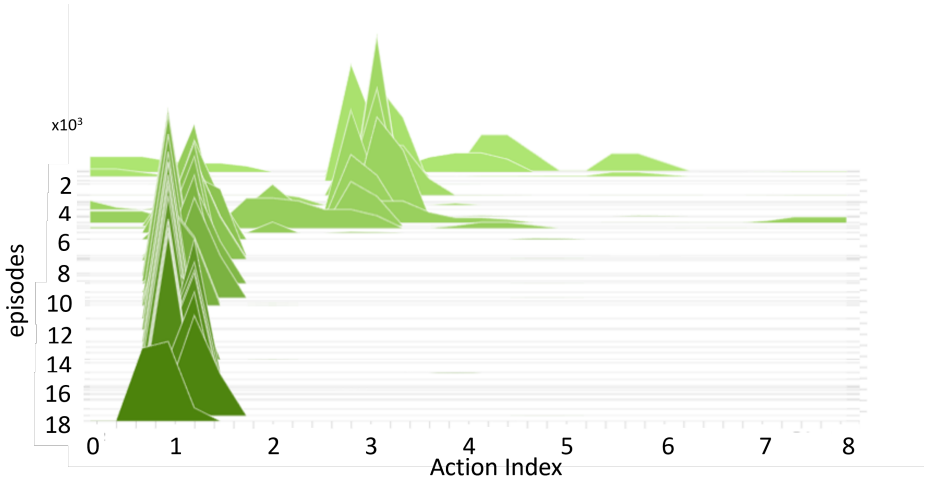


Figure 4.17: Distribution of the actions during the training ($\Lambda_2 = 100ms$, $\varphi_1 = 0.9$)

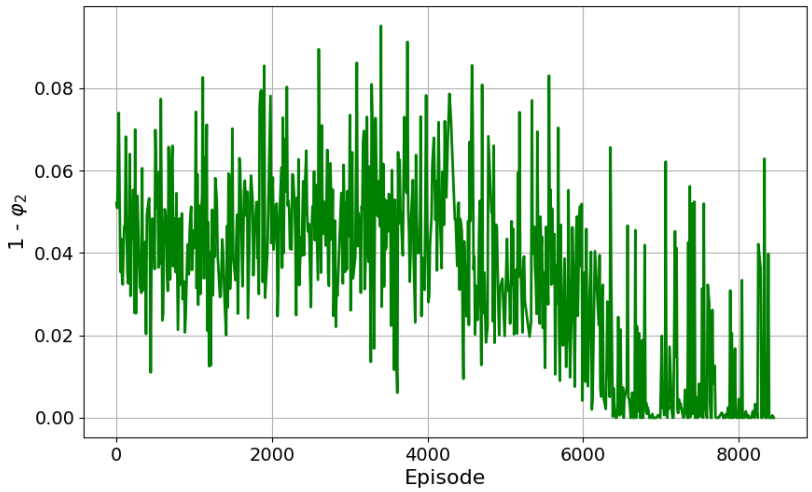


Figure 4.18: Ratio of packets that do not satisfy the latency requirements ($\Lambda_2 = 100ms$, $\varphi_2 = 0.99$)

Chapter 5

Conclusion

Throughout this thesis, we embarked on a journey to address critical aspects of network optimization and management in diverse scenarios, namely: (i) DRL-based Zero-Touch Management in FANETs, in which We proposed frameworks that leverage Deep Reinforcement Learning to automate management tasks in Flying ad-hoc Networks. These frameworks showcased significant potential in improving network efficiency and adaptability; (ii) Distributed Edge-Computing Framework for Green Vehicular Networks, in which we developed a distributed framework utilizing Multi-Player Multi-Armed Bandit (MP-MAB) algorithms to optimize task offloading, achieving a crucial trade-off between energy consumption and job processing latency; (iii) DRL for Network Slicing in O-RAN, where we explored the application of DRL in closed-loop network control scenarios, with the aim of enhancing network performance and adaptability, especially in the context of Network Slicing Service Level Agreement (SLA) satisfaction.

The contributions of this thesis extend beyond the development of these frameworks. We address several challenges that arise when deploying DRL-based control solutions at scale, including dataset collection, robustness testing, agent design, generalization, and feature selection. These challenges are paramount to ensure the successful integration and deployment of DRLs in FANETs, VANETs, and O-RANs.

Moreover, our work demonstrates the potential of DRL techniques in reshaping the landscape of network management and optimization. The novel methodologies and algorithms developed here enable more efficient, adaptive, and autonomous network management, with profound implications for the reliability and performance of future wireless networks.

Overall, this thesis has demonstrated why AI and ML will play a key role in 5G and beyond networks. These techniques will indeed enable fully-distributed

and model-free management of the resources available at the network edge, to the benefit of the network latency, throughput, and lifetime. The importance of AI and ML will become crucial and will therefore stimulate a shift towards the paradigm of AI-centric. networks.

It is also important to acknowledge the ever-evolving nature of the field and to identify avenues for future research:

1. Real-World Deployments: Transitioning from simulations to real-world deployments remains a significant challenge. Future research can focus on practical implementations and the development of frameworks that bridge the gap between theory and practice.

2. Security and Robustness: As networks become increasingly autonomous, ensuring the security and robustness of DRL-based solutions is paramount. Future work should delve into fortifying these solutions against adversarial attacks and unforeseen scenarios.

3. Multi-Domain Optimization: Network management often spans multiple domains. Future research can explore methods to optimize networks that traverse FANETs, VANETs, and O-RANs, considering inter-domain dynamics.

4. Ethical Considerations: The ethical implications of autonomous network management deserve careful consideration. Research into the ethical aspects of DRL in network management is crucial as these technologies become more pervasive.

In closing, this thesis represents a significant step forward in the integration of DRL techniques for network optimization. It is our hope that the findings and methodologies presented here will serve as a foundation for future research and contribute to the continued evolution of network management in FANETs, VANETs, and O-RANs.

References

[1] Corrado Rametta et al. "An open framework to enable NetFATE (Network Functions at the edge)". In: Apr. 2015. DOI: 10.1109/NETSOFT.2015.7116179.

[2] Mostafa Zaman Chowdhury et al. *6G Wireless Communication Systems: Applications, Requirements, Technologies, Challenges, and Research Directions*. 2019. arXiv: 1909.11315 [cs.NI].

[3] Wazir Zada Khan et al. "Edge Computing: A Survey". In: *Future Gener. Comput. Syst.* 97.C (Aug. 2019), pp. 219–235. ISSN: 0167-739X. DOI: 10.1016/j.future.2019.02.050. URL: https://doi.org/10.1016/j.future.2019.02.050.

[4] Lin Wang et al. "Service Entity Placement for Social Virtual Reality Applications in Edge Computing". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, pp. 468–476. DOI: 10.1109/INFOCOM.2018.8486411.

[5] Xiao Yang et al. "Communication-Constrained Mobile Edge Computing Systems for Wireless Virtual Reality: Scheduling and Tradeoff". In: *IEEE Access* PP (Mar. 2018), pp. 1–1. DOI: 10.1109/ACCESS.2018.2817288.

[6] Sunitha Safavat, Naveen Naik Sapavath, and Danda B. Rawat. "Recent advances in mobile edge computing and content caching". In: *Digital Communications and Networks* 6.2 (2020), pp. 189–194. ISSN: 2352-8648. DOI: https://doi.org/10.1016/j.dcan.2019.08.004. URL: https://www.sciencedirect.com/science/article/pii/S2352864819300227.

[7] Ke Zhang et al. "Cooperative Content Caching in 5G Networks with Mobile Edge Computing". In: *IEEE Wireless Communications* 25 (June 2018), pp. 80–87. DOI: 10.1109/MWC.2018.1700303.

[8] Zhiqing Tang et al. "Dependent Task Offloading for Multiple Jobs in Edge Computing". In: Aug. 2020, pp. 1–9. DOI: 10.1109/ICCCN49398.2020.9209593.

[9] Haichuan Wang et al. "Joint Job Offloading and Resource Allocation for Distributed Deep Learning in Edge Computing". In: *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 2019, pp. 734–741. DOI: 10.1109/HPCC/SmartCity/DSS.2019.00109.

[10] Chen Khong Tham and Rajarshi Chattopadhyay. "A load balancing scheme for sensing and analytics on a mobile edge computing network". In: June 2017, pp. 1–9. DOI: 10.1109/WoWMoM.2017.7974307.

[11] Shichao Li, Gang Zhu, and Siyu Lin. "Joint Radio and Computation Resource Allocation with Predictable Channel in Vehicular Edge Computing". In: Nov. 2018, pp. 3736–3741. DOI: 10.1109/ITSC.2018.8569271.

[12] Richard Cziva, Christos Anagnostopoulos, and Dimitrios P. Pezaros. "Dynamic, Latency-Optimal VNF Placement at the Network Edge". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. Honolulu, HI, USA: IEEE Press, 2018, pp. 693–701. DOI: 10.1109/INFOCOM.2018.8486021. URL: https://doi.org/10.1109/INFOCOM.2018.8486021.

[13] Panpan Jin et al. "Latency-Aware VNF Chain Deployment with Efficient Resource Reuse at Network Edge". In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. Toronto, ON, Canada: IEEE Press, 2020, pp. 267–276. DOI: 10.1109/INFOCOM41043.2020.9155345. URL: https://doi.org/10.1109/INFOCOM41043.2020.9155345.

[14] Haoran Qi, Xingjian Zhang, and Yue Gao. "Low-Complexity Subspace-Aided Compressive Spectrum Sensing Over Wideband Whitespace". In: *IEEE Transactions on Vehicular Technology* PP (Oct. 2019), pp. 1–1. DOI: 10.1109/TVT.2019.2937649.

[15] Surendra Solanki, Vasudev Dehalwar, and Jaytrilok Choudhary. "Deep Learning for Spectrum Sensing in Cognitive Radio". In: *Symmetry* 13.1 (2021). ISSN: 2073-8994. DOI: 10.3390/sym13010147. URL: https://www.mdpi.com/2073-8994/13/1/147.

[16] Amani Al-Shawabka et al. "DeepLoRa: Fingerprinting LoRa Devices at Scale Through Deep Learning and Data Augmentation". In: *Proceedings of the Twenty-Second International Symposium on Theory, Algorithmic*

Foundations, and Protocol Design for Mobile Networks and Mobile Computing. MobiHoc '21. Shanghai, China: Association for Computing Machinery, 2021, pp. 251–260. ISBN: 9781450385589. DOI: 10.1145/3466772.3467054. URL: https://doi.org/10.1145/3466772.3467054.

[17] Amani Al-Shawabka et al. "Exposing the Fingerprint: Dissecting the Impact of the Wireless Channel on Radio Fingerprinting". In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 2020, pp. 646–655. DOI: 10.1109/INFOCOM41043.2020.9155259.

[18] Helin Yang et al. "Artificial Intelligence-Enabled Intelligent 6G Networks". In: *IEEE Network* PP (Oct. 2020), pp. 1–9. DOI: 10.1109/MNET.011.2000195.

[19] Ella Peltonen et al. "6G white paper on edge intelligence". In: *arXiv preprint arXiv:2004.14850* (2020).

[20] Bharti Nathani and Rekha Vijayvergia. "The Internet of Intelligent things: An overview". In: *2017 International Conference on Intelligent Communication and Computational Techniques (ICCT)*. 2017, pp. 119–122. DOI: 10.1109/INTELCCT.2017.8324031.

[21] Harish Viswanathan and Preben E. Mogensen. "Communications in the 6G Era". In: *IEEE Access* 8 (2020), pp. 57063–57074. DOI: 10.1109/ACCESS.2020.2981745.

[22] Lei Liu et al. "Vehicular edge computing and networking: A survey". In: *Mobile networks and applications* 26.3 (2021), pp. 1145–1168.

[23] Ignacio Soto et al. "A survey on road safety and traffic efficiency vehicular applications based on C-V2X technologies". In: *Vehicular Communications* 33 (2022), p. 100428.

[24] Lion Silva et al. "Computing paradigms in emerging vehicular environments: A review". In: *IEEE/CAA J. of Automatica Sinica* 8.3 (2021), pp. 491–511.

[25] Fabio Busacca et al. "A smart road side unit in a microeolic box to provide edge computing for vehicular applications". In: *IEEE Trans. on Green Comm. and Networking* (2022).

[26] Michele Polese et al. "Understanding O-RAN: Architecture, interfaces, algorithms, security, and research challenges". In: *IEEE Communications Surveys & Tutorials* (2023).

[27] Michele Polese et al. "ColO-RAN: Developing machine learning-based xApps for open RAN closed-loop control on programmable experimental platforms". In: *IEEE Transactions on Mobile Computing* (2022).

[28] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. DOI: 10.48550/ARXIV.1312.5602. URL: https://arxiv.org/abs/1312.5602.

[29] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. DOI: 10.48550/ARXIV.1509.06461. URL: https://arxiv.org/abs/1509.06461.

[30] Vijay Konda and John Tsitsiklis. "Actor-Critic Algorithms". In: *Society for Industrial and Applied Mathematics* 42 (Apr. 2001).

[31] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[32] Chao Yu et al. "The surprising effectiveness of ppo in cooperative multi-agent games". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 24611–24624.

[33] Mohammad Babaeizadeh et al. "GA3C: GPU-based A3C for deep reinforcement learning". In: *CoRR abs/1611.06256* (2016).

[34] Hankz Hankui Zhuo et al. "Federated deep reinforcement learning". In: *arXiv preprint arXiv:1901.08277* (2019).

[35] Rémi Bonnefoi et al. "Multi-Armed Bandit Learning in IoT Networks: Learning helps even in non-stationary settings". In: *International Conference on Cognitive Radio Oriented Wireless Networks*. Springer. 2017, pp. 173–185.

[36] Chengshuai Shi et al. "Decentralized multi-player multi-armed bandits with no collision information". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 1519–1528.

[37] Marco Giordani et al. "Toward 6G networks: Use cases and technologies". In: *IEEE Communications Magazine* 58.3 (2020), pp. 55–61.

[38] İlker Bekmezci, Ozgur Koray Sahingoz, and Şamil Temel. "Flying Ad-Hoc Networks (FANETs): A survey". In: *Ad Hoc Networks* 11.3 (2013), pp. 1254–1270. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2012.12.004. URL: https://www.sciencedirect.com/science/article/pii/S1570870512002193.

[39] Muhammad Asghar Khan et al. "Flying ad-hoc networks (FANETs): A review of communication architectures, and routing protocols". In: *2017 First International Conference on Latest trends in Electrical Engineering and Computing Technologies (INTELLECT)*. 2017, pp. 1–9. DOI: 10.1109/INTELLECT.2017.8277614.

[40] Corrado Rametta and Giovanni Schembra. "Designing a Softwarized Network Deployed on a Fleet of Drones for Rural Zone Monitoring". In: *Future Internet* 9.1 (Mar. 2017), p. 8. ISSN: 1999-5903. DOI: 10.3390/fi9010008. URL: http://dx.doi.org/10.3390/fi9010008.

[41] Giuseppe Faraci et al. "Green wireless power transfer system for a drone fleet managed by reinforcement learning in smart industry". In: *Applied Energy* 259 (Feb. 2020). DOI: 10.1016/j.apenergy.2019.1.

[42] Giuseppe Faraci, Christian Grasso, and Giovanni Schembra. "Reinforcement-Learning for Management of a 5G Network Slice Extension with UAVs". In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2019, pp. 732–737. DOI: 10.1109/INFCOMW.2019.8845316.

[43] Giuseppe Faraci, Christian Grasso, and Giovanni Schembra. "Fog in the Clouds: UAVs to Provide Edge Computing to IoT Devices". In: *ACM Trans. Internet Technol.* 20.3 (2020).

[44] Giuseppe Faraci, Christian Grasso, and Giovanni Schembra. "Design of a 5G Network Slice Extension With MEC UAVs Managed With Reinforcement Learning". In: *IEEE Journal on Selected Areas in Communications* 38.10 (2020), pp. 2356–2371.

[45] Shaowei Li et al. "Collaborative Decision-Making Method for Multi-UAV Based on Multiagent Reinforcement Learning". In: *IEEE Access* 10 (2022), pp. 91385–91396. DOI: 10.1109/ACCESS.2022.3199070.

[46] Zeyi Xi et al. "Energy-Optimized Trajectory Planning for Solar-Powered Aircraft in a Wind Field Using Reinforcement Learning". In: *IEEE Access* 10 (2022), pp. 87715–87732. DOI: 10.1109/ACCESS.2022.3199004.

[47] Ghada Afifi and Yasser Gadallah. "Cellular Network-Supported Machine Learning Techniques for Autonomous UAV Trajectory Planning". In: *IEEE Access* 10 (2022), pp. 131996–132011. DOI: 10.1109/ACCESS.2022.3229171.

[48] Jihoon Lee et al. "Multi-level Indoor Path Planning and Clearance-Based Path Optimization for Search and Rescue Operations". In: *IEEE Access* (2023), pp. 1–1. DOI: 10.1109/ACCESS.2023.3269981.

[49] Christian Grasso and Giovanni Schembra. "A fleet of MEC UAVs to extend a 5G network slice for video monitoring with low-latency constraints". In: *Journal of Sensor and Actuator Networks* 8.1 (2019), p. 3.

[50] Christian Grasso, Raoul Raftopoulos, and Giovanni Schembra. "Deep Q-Learning for Job Offloading Orchestration in a Fleet of MEC UAVs in 5G Environments". In: *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*. 2021, pp. 186–190. DOI: 10.1109/NetSoft51509.2021.9492638.

[51] Mohammad Mozaffari et al. "A tutorial on UAVs for wireless networks: Applications, challenges, and open problems". In: *IEEE communications surveys & tutorials* 21.3 (2019), pp. 2334–2360.

[52] Christian Grasso and Giovanni Schembra. "Design of a UAV-based video-surveillance system with tactile internet constraints in a 5G ecosystem". In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE. 2018, pp. 449–455.

[53] Giuseppe Faraci, Christian Grasso, and Giovanni Schembra. "Fog in the Clouds: UAVs to Provide Edge Computing to IoT Devices". In: *ACM Transactions on Internet Technology (TOIT)* 20.3 (2020), pp. 1–26.

[54] Fuhui Zhou et al. "UAV-enabled mobile edge computing: Offloading optimization and trajectory design". In: *2018 IEEE International Conference on Communications (ICC)*. IEEE. 2018, pp. 1–6.

[55] Lei Wang and Meng Hua. "Optimal bit allocation for UAV-enabled mobile communication". In: *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. IEEE. 2017, pp. 474–478.

[56] Gilsoo Lee, Walid Saad, and Mehdi Bennis. "Online optimization for UAV-assisted distributed fog computing in smart factories of industry 4.0". In: *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2018, pp. 1–6.

[57] Nader Mohamed et al. "UAVFog: A UAV-based fog computing for Internet of Things". In: *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City In-*

novation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). IEEE. 2017, pp. 1–8.

[58] Giuseppe Faraci, Christian Grasso, and Giovanni Schembra. "Design of a 5G network slice extension with MEC UAVs managed with reinforcement learning". In: *IEEE Journal on Selected Areas in Communications* 38.10 (2020), pp. 2356–2371.

[59] Giuseppe Faraci et al. "Green wireless power transfer system for a drone fleet managed by reinforcement learning in smart industry". In: *Applied Energy* 259 (2020), p. 114204. ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2019.114204. URL: https://www.sciencedirect.com/science/article/pii/S0306261919318914.

[60] Corrado Rametta and Giovanni Schembra. "Designing a softwarized network deployed on a fleet of drones for rural zone monitoring". In: *Future Internet* 9.1 (2017), p. 8.

[61] Fabio D'Urso et al. "The Tactile Internet for the flight control of UAV flocks". In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE. 2018, pp. 470–475.

[62] Yong Zeng, Rui Zhang, and Teng Joon Lim. "Wireless communications with unmanned aerial vehicles: Opportunities and challenges". In: *IEEE Communications Magazine* 54.5 (2016), pp. 36–42.

[63] Mohammad Mozaffari et al. "Unmanned aerial vehicle with underlaid device-to-device communications: Performance and tradeoffs". In: *IEEE Transactions on Wireless Communications* 15.6 (2016), pp. 3949–3963.

[64] Mohammad Mozaffari et al. "Mobile unmanned aerial vehicles (UAVs) for energy-efficient Internet of Things communications". In: *IEEE Transactions on Wireless Communications* 16.11 (2017), pp. 7574–7589.

[65] Gabriella Colajanni and Daniele Sciacca. "An Optimization Model for Service Requests Management in a 5G Network Architecture". In: *Optimization and Data Science: Trends and Applications*. Ed. by Adriano Masone, Veronica Dal Sasso, and Valentina Morandi. Cham: Springer International Publishing, 2021, pp. 81–98. ISBN: 978-3-030-86286-2.

[66] Li Lin et al. "Computation Offloading Toward Edge Computing". In: *Proceedings of the IEEE* 107 (July 2019), pp. 1584–1607. DOI: 10.1109/JPROC.2019.2922285.

[67] Bin Cao et al. “Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework”. In: *IEEE Communications Magazine* 57.3 (2019), pp. 56–62.

[68] Gonçalo Carvalho et al. “Computation offloading in Edge Computing environments using Artificial Intelligence techniques”. In: *Engineering Applications of Artificial Intelligence* 95 (2020), p. 103840.

[69] Abdelhamied A Ashraf Ateya et al. “Energy-and latency-aware hybrid offloading algorithm for UAVs”. In: *IEEE Access* 7 (2019), pp. 37587–37600.

[70] Penglin Dai et al. “A probabilistic approach for cooperative computation offloading in MEC-assisted vehicular networks”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.2 (2020), pp. 899–911.

[71] Yun Lin et al. “Dynamic spectrum interaction of UAV flight formation communication with priority: A deep reinforcement learning approach”. In: *IEEE Transactions on Cognitive Communications and Networking* (2020), pp. 892–903.

[72] Bohao Li and Yunjie Wu. “Path planning for UAV ground target tracking via deep reinforcement learning”. In: *IEEE Access* (2020), pp. 29064–29074.

[73] Sang-Yun Shin, Yong-Won Kang, and Yong-Guk Kim. “Obstacle avoidance drone by deep reinforcement learning and its racing with human pilot”. In: *Applied sciences* 9.24 (2019), p. 5571.

[74] Christian Grasso, Raoul Raftopoulos, and Giovanni Schembra. “Smart Zero-Touch Management for 6G UAV-Based Network Slices”. In: *IEEE* (2021).

[75] Muhammad Asghar Khan et al. “Swarm of UAVs for Network Management in 6G: A Technical Review”. In: *IEEE Transactions on Network and Service Management* 20.1 (2023), pp. 741–761. DOI: 10.1109/TNSM.2022.3213370.

[76] Theodore S. Rappaport et al. “Wireless Communications and Applications Above 100 GHz: Opportunities and Challenges for 6G and Beyond”. In: *IEEE Access* 7 (2019), pp. 78729–78757. DOI: 10.1109/ACCESS.2019.2921522.

[77] Shangwei Zhang et al. "Envisioning Device-to-Device Communications in 6G". In: *IEEE Network* 34.3 (2020), pp. 86–91. DOI: 10.1109/MNET.001.1900652.

[78] Nan Chi et al. "Visible Light Communication in 6G: Advances, Challenges, and Prospects". In: *IEEE Vehicular Technology Magazine* 15.4 (2020), pp. 93–102. DOI: 10.1109/MVT.2020.3017153.

[79] Rojeena Bajracharya et al. "6G NR-U Based Wireless Infrastructure UAV: Standardization, Opportunities, Challenges and Future Scopes". In: *IEEE Access* 10 (2022), pp. 30536–30555. DOI: 10.1109/ACCESS.2022.3159698.

[80] Amartya Mukherjee et al. "DisastDrone: A Disaster Aware Consumer Internet of Drone Things System in Ultra-Low Latent 6G Network". In: *IEEE Transactions on Consumer Electronics* 69.1 (2023), pp. 38–48. DOI: 10.1109/TCE.2022.3214568.

[81] Witold Wydmański and Szymon Szott. "Contention window optimization in IEEE 802.11 ax networks with deep reinforcement learning". In: *2021 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2021, pp. 1–6.

[82] B.P. Crow et al. "IEEE 802.11 Wireless Local Area Networks". In: *IEEE Communications Magazine* 35.9 (1997), pp. 116–126. DOI: 10.1109/35.620533.

[83] Mowei Wang et al. "Machine Learning for Networking: Workflow, Advances and Opportunities". In: *IEEE Network* 32.2 (2018), pp. 92–99. DOI: 10.1109/MNET.2017.1700200.

[84] OpenAI et al. *Dota 2 with Large Scale Deep Reinforcement Learning*. 2019. arXiv: 1912.06680 [cs.LG].

[85] Chih-Heng Ke and Lia Astuti. "Applying multi-agent deep reinforcement learning for contention window optimization to enhance wireless network performance". In: *ICT Express* (2022). ISSN: 2405-9595. DOI: https://doi.org/10.1016/j.icte.2022.07.009. URL: https://www.sciencedirect.com/science/article/pii/S2405959522001060.

[86] Mahdokht A Farahani and Mohsen Guizani. "Markov modulated Poisson process model for hand-off calls in cellular systems". In: *2000 IEEE Wireless Communications and Networking Conference. Conference Record (Cat. No. 00TH8540)*. Vol. 3. IEEE. 2000, pp. 1113–1118.

[87] S-Q Li and C-L Hwang. "Queue response to input correlation functions: Discrete spectral analysis". In: *[Proceedings] IEEE INFOCOM'92: The Conference on Computer Communications*. IEEE. 1992, pp. 382–394.

[88] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[89] Alex Krizhevsky and Geoff Hinton. "Convolutional deep belief networks on cifar-10". In: *Unpublished manuscript* 40.7 (2010), pp. 1–9.

[90] San-qi Li and Chia-Lin Hwang. "On the convergence of traffic measurement and queueing analysis: a statistical-matching and queueing (SMAQ) tool". In: *IEEE/ACM transactions on networking* 5.1 (1997), pp. 95–110.

[91] Alfio Lombardo, Giacomo Morabito, and Giovanni Schembra. "An accurate and treatable Markov model of MPEG-video traffic". In: *Proceedings. IEEE INFOCOM'98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No. 98*. Vol. 1. IEEE. 1998, pp. 217–224.

[92] John B Kenney. "Dedicated short-range communications (DSRC) standards in the United States". In: *Proceedings of the IEEE* 99.7 (2011), pp. 1162–1182.

[93] Khiem Le et al. "Efficient and robust header compression for real-time services". In: *2000 IEEE Wireless Communications and Networking Conference. Conference Record (Cat. No. 00TH8540)*. Vol. 2. IEEE. 2000, pp. 924–928.

[94] Stephen Boyd et al. "Distributed optimization and statistical learning via the alternating direction method of multipliers". In: *Foundations and Trends® in Machine learning* 3.1 (2011), pp. 1–122.

[95] *Gym: a toolkit for developing and comparing reinforcement learning algorithms*. `https://gym.openai.com/`. Accessed Jan 2022.

[96] *Pytorch: an open source machine learning framework*. `https://pytorch.org/`. Accessed Jan 2022.

[97] Nan Zhao, Zehua Liu, and Yiqiang Cheng. "Multi-Agent Deep Reinforcement Learning for Trajectory Design and Power Allocation in Multi-UAV Networks". In: *IEEE Access* 8 (2020), pp. 139670–139679.

[98] Yu Zhang et al. "UAV-Enabled Secure Communications by Multi-Agent Deep Reinforcement Learning". In: *IEEE Transactions on Vehicular Technology* 69.10 (2020), pp. 11599–11611. DOI: 10.1109/TVT.2020.3014788.

[99] Ryan Lowe et al. *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments*. 2017. DOI: 10.48550/ARXIV.1706.02275. URL: https://arxiv.org/abs/1706.02275.

[100] Oriol Vinyals *et al.* "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* (2019), pp. 1–5.

[101] Christopher Berner *et al. Dota 2 with Large Scale Deep Reinforcement Learning*. 2019. DOI: 10.48550/ARXIV.1912.06680. URL: https://arxiv.org/abs/1912.06680.

[102] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. DOI: 10.48550/ARXIV.1707.06347. URL: https://arxiv.org/abs/1707.06347.

[103] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. DOI: 10.48550/ARXIV.1509.06461. URL: https://arxiv.org/abs/1509.06461.

[104] A. Lombardo, G. Morabito, and G. Schembra. "An accurate and treatable Markov model of MPEG-video traffic". In: *Proceedings IEEE INFOCOM'98 Conference on Computer Communications Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies Gateway to the 21st Century*. Vol. 1. Los Alamitos, CA, USA: IEEE Computer Society, Apr. 1998, pp. 217–224. DOI: 10.1109/INFCOM.1998.659657.

[105] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[106] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[107] Qiang Yang et al. "Federated machine learning: Concept and applications". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.2 (2019), pp. 1–19.

[108] Chia-Lin Hwang and San-qi Li. "On the convergence of traffic measurement and queueing analysis: a Statistical-MAtch Queueing (SMAQ) tool". In: *Proceedings of INFOCOM'95*. Vol. 2. 1995, 602–612 vol.2. DOI: 10.1109/INFCOM.1995.515927.

[109] Christian Grasso and Giovanni Schembra. "Design of a UAV-Based Video-surveillance System with Tactile Internet Constraints in a 5G Ecosystem". In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. 2018, pp. 449–455. DOI: 10.1109/NETSOFT.2018.8460024.

[110] Yassine Yazid et al. "UAV-enabled mobile edge-computing for IoT based on AI: A comprehensive review". In: *Drones* 5.4 (2021), p. 148.

[111] Fabio D'Ursol et al. "The Tactile Internet for the flight control of UAV flocks". In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. 2018, pp. 470–475. DOI: 10.1109/NETSOFT.2018.8458493.

[112] Giuseppe Faraci et al. "A 5G platform for Unmanned Aerial Monitoring in Rural Areas: Design and Performance Issues". In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. 2018, pp. 237–241. DOI: 10.1109/NETSOFT.2018.8459960.

[113] Mohammad Mozaffari et al. "A Tutorial on UAVs for Wireless Networks: Applications, Challenges, and Open Problems". In: *IEEE Communications Surveys & Tutorials* 21.3 (2019), pp. 2334–2360. DOI: 10.1109/COMST.2019.2902862.

[114] Roberta Avanzato et al. "Optimization of UAV-Femtocell Systems Positioning via Game Theory to Geolocate Mobile Terminals in a Post-Earthquake Scenario". In: *2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*. Vol. 2. 2021, pp. 785–790. DOI: 10.1109/IDAACS53288.2021.9660873.

[115] Roberta Avanzato and Francesco Beritelli. "An innovative technique for identification of missing persons in natural disaster based on drone-femtocell systems". In: *Sensors* 19.20 (2019), p. 4547.

[116] Roberta Avanzato and Francesco Beritelli. "A smart UAV-femtocell data sensing system for post-earthquake localization of people". In: *IEEE Access* 8 (2020), pp. 30262–30270.

[117] Vu Khanh Quy et al. "Innovative Trends in the 6G Era: A Comprehensive Survey of Architecture, Applications, Technologies, and Challenges". In: *IEEE Access* 11 (2023), pp. 39824–39844. DOI: 10.1109/ACCESS.2023.3269297.

[118] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. "Minimalistic Gridworld Environment for Gymnasium". In: (2018). URL: https://github.com/Farama-Foundation/Minigrid.

[119] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double q-Learning". In: *AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[120] Guangyi Liu et al. "5G Deployment: Standalone vs. Non-Standalone from the Operator Perspective". In: *IEEE Communications Magazine* 58.11 (2020), pp. 83–89.

[121] Adnan Aijaz. "Private 5G: The Future of Industrial Wireless". In: *IEEE Industrial Electronics Magazine* 14.4 (2020), pp. 136–145.

[122] "A Comparative Study of LPWAN Technologies for Large-scale IoT Deployment". In: *ICT Express* 5.1 (2019), pp. 1–7.

[123] Danny Lee, Joe Zhou, and Wong Tze Lin. "Autonomous Battery Swapping System for Quadcopter". In: *IEEE International Conference on Unmanned Aircraft Systems (ICUAS)*. 2015, pp. 118–124.

[124] Greg Brockman et al. "OpenAI Gym". In: *CoRR* abs/1606.01540 (2016). arXiv: 1606.01540. URL: http://arxiv.org/abs/1606.01540.

[125] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[126] Dinh C. Nguyen et al. "6G Internet of Things: A Comprehensive Survey". In: *IEEE Internet of Things Journal* 9.1 (2022), pp. 359–383. DOI: 10.1109/JIOT.2021.3103320.

[127] Amira Chriki et al. "FANET: Communication, mobility models and security issues". In: *Computer Networks* 163 (2019), p. 106877.

[128] Ashish Srivastava and Jay Prakash. "Future FANET with application and enabling techniques: Anatomization and sustainability issues". In: *Computer Science Review* 39 (2021), p. 100359.

[129] IEEE Computer Society LAN/MAN Standards Committee and others. "IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications". In: *IEEE Std 802.11^* (2007).

[130] Lillicrap et al. *Continuous control with deep reinforcement learning*. 2015. DOI: 10.48550/ARXIV.1509.02971. URL: https://arxiv.org/abs/1509.02971.

[131] Piotr Gawłowicz and Anatolij Zubow. "ns-3 meets OpenAI Gym: The Playground for Machine Learning in Networking Research". In: *ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*. Miami Beach, USA, Nov. 2019. URL: http://www.tkn.tu-berlin.de/fileadmin/fg112/Papers/2019/gawlowicz19_mswim.pdf.

[132] George F. Riley and Thomas R. Henderson. "The ns-3 Network Simulator". In: *Modeling and Tools for Network Simulation*. Ed. by Klaus Wehrle, Mesut Güneş, and James Gross. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34. ISBN: 978-3-642-12331-3. DOI: 10.1007/978-3-642-12331-3_2. URL: https://doi.org/10.1007/978-3-642-12331-3_2.

[133] Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[134] Giuseppe Faraci, Christian Grasso, and Giovanni Schembra. "Reinforcement-learning for management of a 5G network slice extension with UAVs". In: *IEEE INFOCOM 2019*. IEEE. 2019, pp. 732–737.

[135] H Koumaras et al. "A network programmability framework for vertical applications in the beyond 5G era". In: *2022 EuCNC/6G Summit*. IEEE. 2022, pp. 375–380.

[136] Dimitrios Fragkos et al. "5G Vertical Application Enablers Implementation Challenges and Perspectives". In: *IEEE MeditCom 2021*, pp. 117–122.

[137] Francesco Licandro, Alfio Lombardo, and Giovanni Schembra. "Multipath routing and rate-controlled video encoding in wireless video surveillance networks". In: *Multimedia Systems* 14 (2008), pp. 155–165.

[138] Sahrish Khan Tayyaba et al. "5G vehicular network resource management for improving radio access through machine learning". In: *IEEE Access* 8 (2020), pp. 6792–6800.

[139] Shashank K Gupta, Jamil Y Khan, and Duy T Ngo. "A 5G-Based Vehicular Network Architecture to Enhance Road Safety Applications". In: *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*. IEEE. 2021, pp. 1–7.

[140] Giuseppe Faraci et al. "A 5G platform for unmanned aerial monitoring in rural areas: Design and performance issues". In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE. 2018, pp. 237–241.

[141] Aleksandrs Slivkins. *Introduction to Multi-Armed Bandits*. 2019. DOI: 10.48550/ARXIV.1904.07272. URL: https://arxiv.org/abs/1904.07272.

[142] Rémi Bonnefoi et al. "Multi-Armed Bandit Learning in IoT Networks: Learning helps even in non-stationary settings". In: *Intern. Conference on Cognitive Radio Oriented Wireless Networks*. Springer. 2017.

[143] Yuxuan Sun et al. "Task Replication for Vehicular Edge Computing: A Combinatorial Multi-Armed Bandit Based Approach". In: *2018 IEEE Global Communications Conference (GLOBECOM)*. 2018, pp. 1–7. DOI: 10.1109/GLOCOM.2018.8647564.

[144] Penglin Dai et al. "Multi-armed bandit learning for computation-intensive services in MEC-empowered vehicular networks". In: *IEEE Transactions on Vehicular Technology* 69.7 (2020), pp. 7821–7834.

[145] Nang Hung Nguyen et al. "Multi-Agent Multi-Armed Bandit Learning for Offloading Delay Minimization in V2X Networks". In: *2021 IEEE 19th International Conference on Embedded and Ubiquitous Computing (EUC)*. IEEE. 2021, pp. 47–55.

[146] Nehad Hameed Hussein et al. "A Comprehensive Survey on Vehicular Networking: Communications, Applications, Challenges, and Upcoming Research Directions". In: *IEEE Access* 10 (2022), pp. 86127–86180. DOI: 10.1109/ACCESS.2022.3198656.

[147] Santi Agatino Rizzo, Giovanni Susinni, and Francesco Iannuzzo. "Intrusiveness of Power Device Condition Monitoring Methods: Introducing Figures of Merit for Condition Monitoring". In: *IEEE Industrial Electronics Magazine* 16.1 (2022), pp. 60–69. DOI: 10.1109/MIE.2021.3066959.

[148] Volodymyr Kuleshov and Doina Precup. *Algorithms for multi-armed bandit problems*. 2014. DOI: 10.48550/ARXIV.1402.6028. URL: https://arxiv.org/abs/1402.6028.

[149] Petar Popovski et al. "Wireless access in ultra-reliable low-latency communication (URLLC)". In: *IEEE Trans. on Comm.* 67.8 (2019).

[150] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[151] Ashley Hill et al. *Stable Baselines*. https://github.com/hill-a/stable-baselines. 2018.

[152] Andrea Lacava et al. "Programmable and Customized Intelligence for Traffic Steering in 5G Networks Using Open RAN Architectures". In: *IEEE Transactions on Mobile Computing* (2023), pp. 1–16. DOI: 10.1109/TMC.2023.3266642.

[153] Leonardo Bonati et al. "Open, programmable, and virtualized 5G networks: State-of-the-art and the road ahead". In: *Computer Networks* 182 (2020), p. 107516.

[154] Solmaz Niknam et al. "Intelligent O-RAN for beyond 5G and 6G wireless networks". In: *2022 IEEE Globecom Workshops (GC Wkshps)*. IEEE. 2022, pp. 215–220.

[155] Hao Zhou, Medhat Elsayed, and Melike Erol-Kantarci. "RAN resource slicing in 5G using multi-agent correlated Q-learning". In: *2021 IEEE 32nd Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. IEEE. 2021, pp. 1179–1184.

[156] *Colosseum*. .https://www.colosseum.net/. [Online; accessed 2023].

[157] Shalitha Wijethilaka and Madhusanka Liyanage. "Survey on Network Slicing for Internet of Things Realization in 5G Networks". In: *IEEE Communications Surveys Tutorials* 23.2 (2021), pp. 957–994. DOI: 10.1109/COMST.2021.3067807.

[158] Yulei Wu et al. "A survey of intelligent network slicing management for industrial IoT: Integrated approaches for smart transportation, smart energy, and smart factory". In: *IEEE Communications Surveys & Tutorials* 24.2 (2022), pp. 1175–1211.

[159] Mojdeh Karbalaee Motalleb et al. "Resource allocation in an open ran system using network slicing". In: *IEEE Transactions on Network and Service Management* 20.1 (2022), pp. 471–485.

[160] Ramraj Dangi et al. “Ml-based 5g network slicing security: A comprehensive survey”. In: *Future Internet* 14.4 (2022), p. 116.

[161] CT Shen et al. “Security threat analysis and treatment strategy for ORAN”. In: *2022 24th International Conference on Advanced Communication Technology (ICACT)*. IEEE. 2022, pp. 417–422.

[162] Nasim Kazemifard and Vahid Shah-Mansouri. “Minimum delay function placement and resource allocation for Open RAN (O-RAN) 5G networks”. In: *Computer Networks* 188 (2021), p. 107809.

[163] Han Zhang, Hao Zhou, and Melike Erol-Kantarci. “Federated deep reinforcement learning for resource allocation in O-RAN slicing”. In: *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE. 2022, pp. 958–963.

[164] Xenofon Foukas, Mahesh K Marina, and Kimon Kontovasilis. “Orion: RAN slicing for a flexible and cost-effective multi-service mobile network architecture”. In: *Proceedings of the 23rd annual international conference on mobile computing and networking*. 2017, pp. 127–140.

[165] *5G performance measurements*. Tech. rep. June 2021.

[166] Leonardo Bonati et al. “SCOPE: An open and softwarized prototyping platform for NextG systems”. In: *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. 2021, pp. 415–426.

[167] *MGEN*. `.https://www.nrl.navy.mil/itd/ncs/products/mgen`. [Online; accessed 2023].

[168] *OpenCelliD*. `.https://opencellid.org/`. [Online; accessed 2023].

[169] Tuomas Haarnoja et al. “Reinforcement learning with deep energy-based policies”. In: *International conference on machine learning*. PMLR. 2017, pp. 1352–1361.

[170] Christopher J. C. H. Watkins and Peter Dayan. “Q-learning”. In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. DOI: `10.1007/BF00992698`. URL: `https://doi.org/10.1007/BF00992698`.

[171] Vijay Konda and John Tsitsiklis. “Actor-Critic Algorithms”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: `https://proceedings.neurips.cc/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf`.

Appendix

Appendix A: Markov Decision Processes

Markov Decision Processes (MDPs) are a mathematical framework for modeling sequential decision-making problems under uncertainty. They are a fundamental concept in RL, as they provide a formal way to represent the environment, the agent's actions, and the rewards associated with those actions.

An MDP is defined by the tuple (S, A, T, R, γ), in which:

- States (S): A set of possible states of the environment.
- Actions (A): A set of possible actions that the agent can take.
- Transition probabilities (T): A set of probabilities that determine the likelihood of transitioning from one state to another given an action.
- Rewards (R): A function that assigns a reward to each state-action pair.
- Discount factor (γ): A value between 0 and 1 that determines the importance of future rewards relative to immediate rewards.

The goal of an agent in an MDP is to learn an optimal policy, which is a mapping from states to actions that maximizes the expected cumulative reward over time.

In an MDP, the agent has complete information about the current state of the environment. This means that the agent knows exactly where it is and what its options are. In a Partially Observable Markov Decision Processes (POMDPs), the agent does not have complete information about the current state of the environment. This means that the agent has to make decisions based on its observations of the environment, which may be noisy or incomplete.

POMDPs are therefore a generalization of MDPs to environments where the agent does not have perfect information about the current state of the environment. Instead, the agent only has access to observations, which are noisy and unreliable indicators of the true state.

A POMDP is defined by the the tuple S, A, T, R, γ, O, B, in which:

- States (S): A set of possible states of the environment.
- Actions (A): A set of possible actions that the agent can take.
- Transition probabilities (T): A set of probabilities that determine the likelihood of transitioning from one state to another given an action.
- Rewards (R): A function that assigns a reward to each state-action pair.
- Discount factor (γ): A value between 0 and 1 that determines the importance of future rewards relative to immediate rewards.
- Observations (O): A set of possible observations that the agent can make.
- Observation probabilities (B): A set of probabilities that determine the likelihood of making an observation given the true state and the chosen action.

The goal of an agent in a POMDP is to learn an optimal policy, which is a mapping from observations to actions that maximizes the expected cumulative reward over time.

MDPs and POMDPs are important in RL for the following reasons:

1. They provide a formal framework for modeling sequential decision-making problems under uncertainty. This makes them well-suited for modeling a wide range of RL problems.
2. They allow for the use of powerful mathematical techniques to solve RL problems. These techniques can be used to find optimal policies for MDPs and POMDPs, even in complex environments.
3. They provide a basis for developing new RL algorithms. Many RL algorithms are based on the principles of MDPs and POMDPs.

Appendix B: Deep Reinforcement Learning algorithms

In this section, we describe several DRL algorithms that are used to solve the MDPs described in the previous chapters.

Q-Learning and Deep Q-Networks

In RL, the notion of *state-value function*, often denoted as $V^{\pi}(s_n)$, represents how good is a state for an agent to be in, and is equal to the expected total reward that the agent will get starting from state s_n and then following policy π. Similarly, the *action-value function*, denoted as $Q^{\pi}(s_n, a_n)$, represents the expected total reward that the agent should receive when executing action a_n in state s_n. In traditional Q-Learning algorithms [170], the action-value function is updated as follows:

$$\begin{aligned} Q^{\pi}(s_n, a_n) \leftarrow Q^{\pi}(s_n, a_n) + \alpha(r_{n+1} + \\ + \gamma \max Q^{\pi}(s_{n+1}, a_{n+1}) - Q^{\pi}(s_n, a_n)) \end{aligned} \tag{1}$$

where $\alpha \in (0, 1]$ is the learning rate, while $Q(s_n, a_n)$ and $Q(s_{n+1}, a_{n+1})$ are the Q-values of the current state and the next state, respectively. However, as the number of states and actions in the MDP increases, the Q-table becomes too large and the algorithm fails to converge.

Instead of approximating Q-values in the Q-table, Deep Neural Networks can be used to estimate $Q^{\pi}(s_n, a_n)$, i.e., $Q^{\pi}(s_n, a_n) \approx Q^{\pi_\theta}(s_n, a_n) \approx Q^{\theta}(s_n, a_n)$, where θ represents the weights of the neural network that is used as a function approximator of the Q-values, replacing the Q-table. The network, called *Deep Q-Network* (DQN) [28], is then trained by minimizing the loss via Temporal Difference (TD) Learning, which requires retrieving the loss gradient, which is expressed as follows:

$$\begin{aligned} \nabla_{\theta} J(\pi_{\theta})^{DQN} = E[r_{n+1} + \gamma \max Q^{\theta}(s_{n+1}, a_{n+1}) - \\ Q^{\theta}(s_n, a_n)] \end{aligned} \tag{2}$$

Therefore, the loss depends on the difference between the target Q-value, which is calculated using the Q-learning formula [170], and the estimated Q-value. The goal is to get the estimate Q-value as close as possible to the target Q-value.
To obtain these two Q-values, two independent neural networks are deployed, with the same identical structure; they are called the evaluation network, with weights θ, and the target network, with weights θ'. The former generates the estimate Q-value according to the current state and changes its parameters every iteration to decrease the loss, while the latter updates its parameters, θ', by simply copying the evaluation network parameters, θ, into its own network every k steps, where k is a hyperparameter.
To provide training samples, DQN has a replay memory that stores historical experience tuples. These are selected randomly from the replay memory to train the neural network. This allows to enhance the stability of the training phase.

DQN is a value-based method in which the policy is not stored explicitly, but it can be rather derived directly from the value function by simply picking the action with the best value.

Advantage Actor Critic (A2C)

The A2C [30] method is a Policy Gradient approach where the policy of an RL Agent is parameterized and learned directly from experience. Policy gradient methods attempt to compute an estimator of a parameterized policy function using a gradient descent algorithm rather than an action-value or a state-value function. Thus, they avoid the convergence problems that occur with estimation functions due to non-linear approximation and partial observation. The policy gradient, and therefore the loss function, is based on the expectation over the probabilities of the policy actions and an estimate of the advantage function at time step t, and is expressed as follows:

$$L(\theta)^{PG} = \hat{E}[\nabla_\theta \log \pi_\theta(a|s)\hat{A}] \tag{3}$$

where $\hat{E}$ is the expectation operator over a finite batch of samples, π_θ indicates a stochastic policy, $\hat{A}$ is defined by the discounted sum of rewards and a baseline estimate, and a and s express the action and state, respectively. While it might be appealing and straightforward to perform multiple steps of optimization on this loss function $L^{PG}(\theta)$, many challenges can arise from the prevalence of sample inefficiency, the balance between exploration and exploitation, and the undesirable high variance of the learned policy. Empirically, it often leads to destructively large policy updates, which are destructive since they can also affect the observation and reward distribution at future time steps. Actor-Critic methods use two neural networks, the Policy Network, also called the Actor, and the Value Network, also called the Critic, with θ and ω being the parameters of the two networks, respectively. The Critic is usually used to approximate the parameterized value function $q_\pi(s, a)$ or $v_\pi(s)$. On the other hand, the Actor updates the policy distribution in the direction suggested by the Critic. A2C introduces the *advantage* function, defined as the difference between the action-value function and the state-value function:

$$\alpha_\pi(s, a) = q_\pi(s, a) - v_\pi(s) \tag{4}$$

This is a measure of how much better it is to take a specific action compared to the average, general action at the given state. To avoid constructing two neural

networks to approximate both the action-value and the state-value, we can use the relationship between the state-value and the action-value dictated from (4). We can therefore rewrite the advantage as:

$$\alpha_\pi(s,a) = r + \gamma v_\pi(s') - v_\pi(s) \tag{5}$$

where s' is the value of the state of the environment at the next step. This allows us to calculate the advantage by having to approximate only the state values. Hence, we only need another neural network, besides the policy network, to retrieve the advantage by approximating the state-value. The parameter update rule of the Actor is given by:

$$\theta \longleftarrow \theta + \alpha^\theta \cdot \nabla_\theta \log_{10}\left(\Pr\left\{a|\pi_\theta(s)\right\}\right) \cdot \alpha_\omega(s,a) \tag{6}$$

while the parameter update rule of the Critic is given by:

$$\omega \longleftarrow \omega + \alpha^\omega \cdot \nabla_\omega \alpha_\omega(s,a)^2 \tag{7}$$

The variables α^θ and α^ω in (6) and (7) are the step sizes for Actor and Critic networks, respectively.

The Actor and the Critic can share both the input layers and the hidden layers. For this reason, the gradient can be calculated as:

$$\nabla f(\theta) = \nabla_\theta\left(\Pr\left\{a|\pi_\theta(s)\right\}\right) \cdot \alpha_\theta(s,a) + \nabla_\theta \alpha_\theta(s,a)^2 \tag{8}$$

The loss function is then given by:

$$L(\theta)^{A2C} = \alpha_\theta(s,a) \cdot \left(-\log_{10}\left(\Pr\left\{a|\pi_\theta(s)\right\}\right) + \alpha_\theta(s,a)\right) \tag{9}$$

Deep Deterministic Policy Gradient

In policy-based methods, a representation of a policy $\pi : s \rightarrow a$ is explicitly created and updated during the training phase. For this reason, in the Deep Policy Gradient (DPG) method, as in all the policy-based methods, the gradient of the loss is calculated differently than the value-based methods, that is, as the expected value over all the possible states and actions, as in (3). This means that a lot of experience tuples $(s_n, a_n, r_{n+1}, s_{n+1})$ have to be collected from the environment to make an accurate estimation of the gradient.

On the other hand, in Deterministic DPG (DDPG), the mapping from states to actions is fixed, so we do not need to integrate over the whole action space. In this

case, the gradient is the expected gradient of the action-value function, which can be estimated much more efficiently than its stochastic version. Therefore, DDPG needs fewer data samples to converge over stochastic policy gradient algorithms.

To explore the environment, the DDPG algorithm achieves off-policy learning by borrowing ideas from Actor-Critic methods [171]. Actor-Critic methods use two neural networks, the Policy network, also called the Actor, and the Value network, also called the Critic, with θ and ω being the parameters of the two networks, respectively. The Critic is usually used to approximate the action-value function $Q^w(s, a)$. On the other hand, the Actor updates the policy distribution in the direction suggested by the Critic.

Essentially, the Actor produces the action, a_n, given the current state of the agent, s_n, while the Critic produces a signal that criticizes the actions made by the Actor. The Actor network outputs the actions, while the Critic network takes the actions and observation as input and outputs the estimated rewards for each action. Then the Critic is updated via TD learning as in (2), and the Actor is updated via Policy Gradient as in (3). Therefore, recalling that the parameters of the Critic are w and the parameters of the Actor are θ, then the gradient for DDPG is:

$$\nabla_\theta J(\mu_\theta)^{DDPG} = E\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s, a)|a = \mu_\theta(s)\right] \tag{10}$$

Where the weights of the Actor, w, and of the Critic, θ, are updated as in (6) and (7): In order to improve the stability of the Actor training process, two techniques are borrowed by DDQN [29]: Experience Replay and target networks.
Experience Replay Buffer aims at satisfying one of the fundamental requirements in neural network optimization, that is, that the training data should be independent and identically distributed. This is to prevent action values from oscillating or diverging catastrophically. This can happen because the sequence of experience tuples $(s_n, a_n, r_{n+1}, o_{n+1})$ can be highly correlated. The learning algorithm that learns from each of these experience tuples in sequential order runs the risk of getting swayed by the effects of this correlation.

The other idea that has been adopted by DQN is the use of *target networks*. The Bellman equation provides us with the value of $Q(s_n, a_n)$ via $Q(s_{n+1}, a_{n+1})$, and this can potentially lead to harmful correlations. In fact, state s_n and s_{n+1} are only one step far from each other. This makes them potentially very similar, therefore making it very hard for a neural network to distinguish between them. When we perform an update of our neural networks' parameters to make $Q(s_n, a_n)$ closer to the desired result, we can indirectly alter the value produced for $Q(s_{n+1}, a_{n+1})$ and other similar states. This can make our training very unstable. To make

training more stable, DDQN introduced the concept of target network, in which a copy of the neural network is used to get the $Q(s_{n+1}, a_{n+1})$ value in the Bellman equation. This means that the predicted Q-values of this target network are used to backpropagate through and train the main network.

It is important to highlight that the target network's parameters are not trained via neural network optimization, but are rather periodically synchronized with the parameters of the main network. This update is called "hard-update". The idea is that using the target network's Q-values to train the main Q-network will improve the stability of the training. By using target networks, we prevent the training process from spiraling around because we are fixing the targets for multiple time steps, thus allowing the online network weights to move consistently toward the targets before an update changes the optimization problem, and a new one is set. On the other hand, in this way the learning process is slowed down, because the main network is no longer training on up-to-date values: the frozen weights of the target network can be lagging for up-to 10,000 steps at a time. Therefore, it is essential to balance stability and speed and tune this hyperparameter.

In DDPG, target networks are used for both Actor and Critic. However, if on DDQN the weights of the main network are periodically copied to the target network, in DDPG the weights of these target networks are updated by having them slowly track the learned networks. This is done with a fixed frequency via "soft-update". For Actor and Critic networks, the network weights θ and w are soft-updated respectively by:

$$\theta' = \tau\theta + (1 - \tau)\theta' \tag{11}$$

$$w' = \tau w + (1 - \tau)w' \tag{12}$$

Proximal Policy Optimization (PPO)

PPO [103] is an advanced Policy-based algorithm that builds upon the Actor-Critic structure, which is able to combine advantages of traditional value-based and policy-based approaches. The PPO algorithm is also much simpler to implement, has less computation burden, and has better sample complexity (empirically). Specifically, PPO proposes a clipped surrogate loss function and combines the policy surrogate and a value function error term:

$$L^{PPO}(\theta) = \hat{E}[L^{CLIP}(\theta) - c_1 L^{VF}(\theta) + c_2 S[\pi_\theta](s)] \tag{13}$$

where L^{CLIP} is the clipped surrogate objective, c_1 and c_2 are coefficients, L^{VF} is the squared-error loss of the value function, and S denotes the entropy loss. Specifically, the clipped surrogate objective L^{CLIP} takes the following form:

$$L^{CLIP}(\theta) = \hat{E}[\min(r(\theta)\hat{A}, clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}] \tag{14}$$

where ϵ is an hyperparameter, $r(\theta)$ denotes the probability ratio $r(\theta) = \pi_\theta(a|s)/\pi_{\theta_{old}}(a_t|s_t)$. This way, the probability ratio r is clipped at $1-\epsilon$ or $1+\epsilon$ depending on whether the advantage is positive or negative, which forms the clipped objective after multiplying the advantage approximator $\hat{A}$. The final value of L^{CLIP} takes the minimum of this clipped objective and the unclipped objective $r(\theta)\hat{A}$, which can effectively avoid taking large policy updates.