SECURE AND ENERGY-EFFICIENT RESOURCE ALLOCATION IN NETWORK SLICING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

UMUT CAN GÜLMEZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2022

Approval of the thesis:

SECURE AND ENERGY-EFFICIENT RESOURCE ALLOCATION IN NETWORK SLICING

submitted by **UMUT CAN GÜLMEZ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Date: 02.09.2022

Name, Surname: Umut Can Gülmez

Signature :

ABSTRACT

SECURE AND ENERGY-EFFICIENT RESOURCE ALLOCATION IN NETWORK SLICING

Gülmez, Umut Can
M.S., Department of Computer Engineering
Supervisor: Assoc. Prof. Dr. Pelin Angın

September 2022, 66 pages

The one-size-fits-all idea of the previous telecommunication generations is no longer suitable for current applications. The current network systems need to satisfy the Quality of Service requirements of the different types of use cases such as enhanced mobile broadband, ultra-reliable and low latency communications and massive machine type communications in the same physical infrastructure. 5G telecommunication networks aim to provide a solution to this problem through the network slicing concept. Virtual Network Functions (VNF) play an essential role in the network slicing concept and embedding these functions into the network is an important task to achieve. As state-of-the-art research focuses on allocating these functions in-network taking only energy efficiency into consideration, this research proposes a solution that considers the security aspects too. We propose a VNF placement strategy using an integer linear programming (ILP) model for 5G network slicing under strict security requirements, which optimizes energy consumption by the core network nodes. As an improvement to this approach, we also propose using Deep Reinforcement Learning (DRL) methods to provide a dynamic, energy-efficient, resilient, and secure resource allocation framework for network slicing. Hence, in this research, we compared various state-of-the-art DRL methods to find a suitable algorithm for energy-efficient resource allocation under stringent security constraints. Simulation results demonstrate that the proposed DRL-based models achieve significant energy optimization compared to the ILP-based optimization model performing VNF placement under the same QoS and security constraints. The results of the study show that DRL-based methods provide faster allocation than ILP-based methods. Also, the results show that DRL-based methods provide faster allocation than ILP-based methods.

Keywords: 5G and Beyond Mobile Networks, Network Slicing, Deep Reinforcement Learning, Integer Linear Programming, Network Security, Energy Efficiency

ÖZ

AĞ DİLİMLEMEDE GÜVENLİ VE ENERJİ-VERİMLİ KAYNAK TAHSİSİ

Gülmez, Umut Can
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Pelin Angın

Eylül 2022 , 66 sayfa

Önceki telekomünikasyon nesillerinin tek çeşit şebeke ile tüm kullanıcılara aynı hizmeti verme fikri artık günümüz uygulamaları için yeterli olmamaktadır. Mevcut şebeke sistemlerinin, aynı fiziksel altyapıda Gelişmiş Mobil Geniş Bant, Ultra Güvenilir ve Düşük Gecikmeli İletişim ve Büyük Makine Tipi İletişim gibi farklı kullanım şekillerinin şebeke iletişimi hizmet kalite (QoS) şartlarını karşılaması bir zorunluluk haline gelmiştir. 5G telekomünikasyon ağları bu soruna bir çözüm önerisinde bulunuyor: Ağ Dilimleme kavramı. Ağ fonksiyonlarının sanallaştırılması (VNF), ağ dilimleme konseptinde önemli bir rol oynamakta ve bu sanallaştırılan fonksiyonların şebeke üzerinde yerleştirilmesi önemli bir görev teşkil etmektedir. Literatürdeki araştırmalar, bu fonksiyonları ağ üzerinde yalnızca enerji verimliliğini dikkate alarak tahsis etmeye odaklandığından, bu tez çalışması güvenlik yönlerini de dikkate alan bir çözüm sunmaktadır. Bu çalışmada, güvenlik gereksinimleri olan 5G ağ dilimleme konsepti için çekirdek ağ fonksiyonlarının enerji tüketimini, tamsayılı doğrusal programlama (ILP) modeli kullanarak sanal ağ fonksiyonlarını verimli bir şekilde yerleştirme stratejisi ile optimize edilmektedir. Bu yaklaşım üzerine bir iyileştirme olarak, ağ dilimleme için dinamik, enerji açısından verimli, esnek ve güvenli bir kaynak tahsis çerçevesi sağlamak için Derin Güçlendirmeli Öğrenme (DRL) yöntemlerinin kullanılması eklenmiştir. Çalışmada, katı güvenlik kısıtlamaları altında enerji verimli kaynak tahsisi için uygun bir algoritma bulmak için çeşitli son teknoloji Derin Güçlendirmeli Öğrenme yöntemleri karşılaştırmıştır. Simülasyon sonuçları önerilen DRL modellerinin, aynı QoS ve güvenlik kısıtlamaları altında VNF yerleştirme gerçekleştiren ILP tabanlı yaklaşıma göre önemli ölçüde güç tasarrufu sağladığını göstermektedir. Aynı zamanda, DRL tabanlı yöntemlerin ILP tabanlı yöntemlerden daha hızlı kaynak tahsis sağladığını da göstermektedir.

Anahtar Kelimeler: 5G ve Ötesi Mobil Ağlar, Ağ Dilimleme, Derin Pekiştirmeli Öğrenme, Tam Sayılı Doğrusal Programlama, Ağ Güvenliği, Enerji Verimliliği

To my family and friends

ACKNOWLEDGMENTS

I want to acknowledge and give my warmest thanks to my advisor Dr. Pelin Angin for her helpful comments and feedback throughout the writing process. Her patience, motivation, and immense knowledge in the area led me to point in the right direction and improved the quality of work I produced. I would also like to thank my thesis committee members, Dr. İbrahim Körpeoğlu and Dr. Serkan Sarıtaş.

A debt of gratitude is also owed to my mother Nuran, father Ferhat, and significant other Dilara for believing in me throughout the thesis process and supporting me spiritually throughout my life.

This study has been supported by Vodafone within the framework of 5G and Beyond Joint Graduate Support Programme coordinated by Information and Communication Technologies Authority. Being a scholar of this programme enabled this research to be conducted at a level where both industry and academic contributions produced the desired outcome most efficiently.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

2G	Second Generation
3G	Third Generation
3GPP	Third Generation Partnership Project
5G	Fifth Generation
5GCN	5G Core Network
AI	Artificial Intelligence
AMF	Access and Mobility Management Function
API	Application Programming Interface
AUSF	Authentication Server Function
CapEx	Capital Expenditure
CN	Core Network
DDQN	Double Deep Q-Network
DoS	Denial of Service
DQN	Deep Q-Network
DQL	Deep Q-Learning
DRL	Deep Reinforcement Learning
eMBB	Enhanced Mobile Broadband
ETSI	The European Telecommunications Standards Institute
EPC	Evolved Packet Core
ILP	Integer Linear Programming
IoT	Internet of Things
KPI	Key Performance Indicator
LTE	Long Term Evolution
MANO	Management and Orchestration

mMTC	Massive Machine Type Communication
NF	Network Function
NFV	Network Function Virtualization
NN	Neural Networks
ONAP	Open Network Automation Platform
OpEx	Operating Expenses
OSM	Open Source MANO
OSS/BSS	Operations Support System and Business Support System
QoS	Quality of Service
RAN	Radio Access Network
RL	Reinforcement Learning
SBA	Service Based Architecture
SDN	Software Defined Network
SLA	Service Level Agreements
SMA	Session Management Function
UDM	Unified Data Management Function
UDR	Unified Data Repository
UE	User Entity
URLLC	Ultra Reliable Low Latency Communication
UPF	User Plane Function
vMME	virtual Mobility Management Entity
VNF	Virtual Network Function
VNFI	Virtual Network Function Infrastructure
VNSF	Virtual Network Security Function

CHAPTER 1

INTRODUCTION

As the number of connected devices and use cases telecommunication networks aim to support is increasing at a tremendous speed, legacy mobile networks are falling short of achieving the bandwidth, throughput, and delay requirements of various network applications. Given the potential of the Internet of Things (IoT) to create smarter systems as "the next major economic and societal innovation wave" after Internet's evolution and expectations that worldwide 5G subscribers will exceed 4.4 billion by 2027 [1], it is essential to develop 5G infrastructures that will support massive IoT. While 5G has immense potential to transform the world's digital economy and help create significant opportunities for sustainable computing, it is expected to incur up to 170% increase in network energy consumption in macro, node, and network data center areas by 2026 [2]. When considering the high costs of deploying the 5G infrastructure with high coverage globally and the fact that standards are still being developed, the creation of frameworks for efficient management and resource provisioning for 5G networks becomes paramount.

The network slicing concept is considered one of the enablers of 5G telecommunication networks using virtualization technologies to provide flexible, scalable, and effective end-to-end operation in 5G networks. With the combination of Network Function Virtualization (NFV) and Software-defined Networks (SDN), network slices support various network applications that have different service level agreements (SLAs) to run on the same physical infrastructure. Allocating resources efficiently for these different use cases is a critical issue to be resolved, as poorly allocated assets could cost a fortune to the operators. Network slice embedding onto network topology must be handled in a self-organized way to reduce the capital (CapEx) and operational (OpEx) expenses of the operators. In addition to that, one of the essential purposes of automated resource allocation is to decrease energy consumption. The research interest in establishing an automated resource allocation scheme in network slicing generally focuses on this purpose. However, state-of-the-art research does not consider the security aspects of the virtual network functions. Due to the different vendors' co-existence in the same physical environment, isolation between the slices needs to be handled by the operators while keeping energy consumption at a minimum.

1.1 Motivation and Problem Definition

While network slicing is seen as a compulsory technology for the successful operation of next-generation mobile networks, its effective implementation faces many challenges to be solved before widespread adoption by telecommunication operators can take place:

- **Security** is a critical problem [3], [4] due to sharing of physical resources between slices, each of which may have different security requirements [5]. The security requirements of one slice could affect the overall performance of other network slices.
- Given the concerns regarding the vast energy consumption expectations for 5G networks, one of the key performance indicators will be **energy efficiency** [6] as in other wireless networks [7]. Jointly optimizing energy consumption and QoS parameters like delay, bandwidth, and throughput while meeting **strict security requirements** is a challenge, as these are conflicting in most cases.
- In the network slicing concept, network slice instances share the same physical infrastructure; hence **isolation** of security-critical network functions needs to be handled by the operators. Optimizing the embedding of these functions is an issue that needs to be solved.
- Network environment could change stochastically [8] as the demands of the end-user types could change in time. Resource allocation methods should con-

sider this **dynamicity**. Instead of using mathematical models like ILP, data-driven models that have proven themselves in stochastic environments such as DRL methods should be used in highly dynamic environments.

1.2 Contributions

This thesis proposes a novel energy-efficient and secure network slice embedding model for 5G core networks with Integer Linear Programming (ILP) based optimization and Deep Reinforcement Learning based optimization to address the above challenges.

The proposed model contributes to the literature on energy-efficient secure network slicing as follows:

- We propose a DRL-based optimization model that embeds virtual network slices in a 5G network while maintaining QoS of different use cases. The proposed architecture use cases (slice types) have different security constraints, and the proposed method embeds these slice types in a secure and energy-efficient way.
- The proposed method decreases energy consumption via disabling underutilized virtual network functions. Apart from disabling VNFs, DRL agents and the ILP-based model also deactivate the servers that do not run any VNFs.
- We compare promising state-of-the-art DRL algorithms to find an optimal solution to the secure network embedding problem. Furthermore, we compare the proposed method with our previous ILP-based work [9] in order to demonstrate the difference between data-driven and mathematical optimization-based approaches for energy-efficient and secure resource allocation in network slicing.
- This thesis also shows that security requirements affect energy consumption in 5G networks. Trained agents were used to optimize energy consumption in both networks where security requirements were considered and those that did not consider any security requirements.

- We propose an optimization model for the virtual network embedding problem in 5G core network slicing that achieves significant energy savings in overall core network energy consumption under strict security requirements for virtual network function placement.

- We provide a network topology generator that can be used to test optimization models on various topologies with different requirements.

1.3 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2 explains the network slicing concept, network slicing use cases, and network slicing enablers, including NFV and SDN. In addition to that, this chapter summarizes the Deep Reinforcement Learning algorithms that we used to allocate resources. Finally, this chapter mentions state-of-the-art resource allocation schemes in network slicing and compares them.

Chapter 3 introduces the architecture of the proposed approach for energy-efficient, secure core network slicing for the ILP-based optimization model. Network constraints like bandwidth, latency, and how to generate the network topology are explained in this part. The mathematical details of the constraints given in the ILP model are explained here.

Chapter 4 starts with the problem explanation and its solution using DRL-based optimization. Furthermore, environment generation for Deep Reinforcement Learning algorithms is explained in detail. A heuristic method implemented for mapping traffic into the network graph is presented. Security constraints of the servers and slice types are given in detail, and slice types and their limitations regarding QoS are briefly explained.

Chapter 5 includes the evaluation of the proposed method and compares the proposed DRL-based method with the ILP-based model. This chapter illustrates the energy efficiency of the different models, the proposed models' effect on reducing energy consumption on secure and non-secure versions of the architectures, and the convergence of the deep reinforcement learning algorithms.

Chapter 6 concludes the paper with future work directions.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 An Overview of 5G Core Networks

As in the previous generations of telecommunications technologies, 5G networks consist of Radio Access Network (RAN) and Core Network (CN). While the RAN is responsible for connecting user entities (UE) with the CN in a wireless fashion, CN is the backbone of the telecommunications network, which handles voice and data routing. Network slicing can be applied on both RAN and CN. We focus on CN slicing in this thesis and RAN slicing will be out of the scope of this work. In order to better understand 5G CN, basic 5G CN concepts are explained below based on [10].

Until 5G, each generation of mobile networks (2G/3G/4G) had developed new protocols and interfaces for connecting radio and core networks and the network layer itself. 5G CN, on the other hand, to prevent further complexities, aims to define an access-independent interface that can be used by any relevant access technology, including those that have not been specified in the standards yet. Therefore, it also aims to address future access technologies.

As with the previous generations, 5G will be built on top of the nearest of its predecessors, which is 4G. 4G is made up of Long Term Evolution (LTE) on RAN and Evolved Packet Core (EPC) on CN. Since LTE is the most widespread radio access technology today, 5G architecture has been developed so that the new 5G RAN can cooperate with the existing LTE. On the CN side, comparing EPC and 5G Core Network (5GCN) would show what improvements we expect from 5G. User data processing and integration with RAN are very similar in EPC and 5GCN. On the signaling functionality side, however, 5GCN will introduce the **Service Based Architecture**

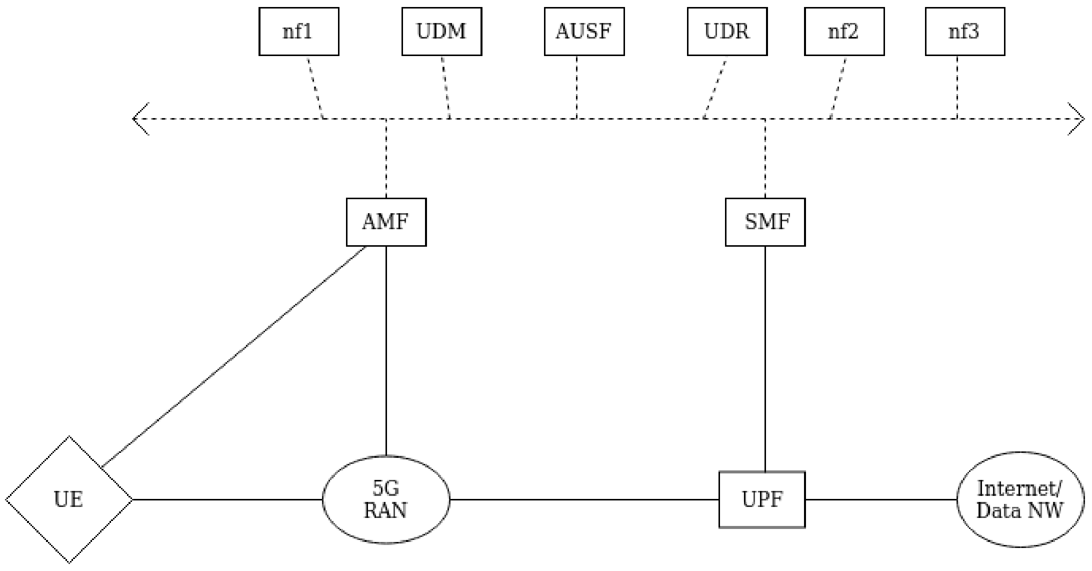

Figure 2.1: 5G Core Network Architecture

(SBA), which marks the main difference in the networking paradigm. In order to understand the SBA, we must understand the core network's functional view and the main Network Functions (NF). Many more NFs can be defined; however, without diving into too much detail, the most fundamental core network functionality can be summarized in six main NFs:

- Access and Mobility Management Function (AMF)
- Session Management Function (SMF)
- User Plane Function (UPF)
- Unified Data Management Function (UDM)
- Unified Data Repository (UDR)
- Authentication Server Function (AUSF)

Instead of network nodes connected via specific interfaces, SBA now allows different NFs to offer services to other NFs via Application Programming Interfaces (API). NFs which provide some services take the role of **Service Producer**, and NFs which make requests to get those services will be **Service Consumer**.

Figure 2.1 explains the basic 5GCN. NFs whose abbreviations were specified are the essential NFs. Other possible NFs are specified with nf*n* labels. AMF is responsible for signaling between the User Entity (UE) and the RAN. UPF's task is to process and forward user data. SMF controls the UPF. All other NFs publish their services and subscribe to each other's services through the service-based interface (dashed channel). Arrows on both sides of the horizontal dashed line indicate that various other NFs can be present on this service-based interface.

2.1.1 Trusted Execution Environment

Trusted Execution Environments (TEEs) are secure, isolated areas of processing that use special hardware to provide a high level of data protection. As stated in [11], there has been much research conducted on TEEs and their attestation techniques in both academia [12–14] and industry [15, 16]. **Attestation** of a Trusted Execution Environment is the technique that measures the status of the TEE. In our work, we assume some of the cloud nodes where NFVs are placed contain a TEE, a subset of which have the attestation technique. TEE and attestation are added to the proposed architecture because some slice owners would require to run their applications on the servers that would provide TEE and even attestation of a TEE for security purposes.

2.2 Network Slicing in 5G

Network slicing can be described as the placement of various end-to-end virtual networks on top of shared physical infrastructure. Standardization of the network slicing concept is underway by several institutions like 3GPP, ETSI, and IETF. According to the standards accepted by the 3GPP foundation, **a network slice** is defined as a logical end-to-end network that can be created on-demand, and users can access these multiple slices over the same radio interface [17]. These slices contain **network slice instances** that consist of several network functions. **Network functions** have detailed functional behavior and well-defined 3GPP interfaces [17]. These logical networks will provide various use cases like enhanced mobile broadband, ultra-reliable low latency communications, and massive machine-type communications. Figure 2.2

shows the network slicing concept with several use cases of the slices and network slice instances.

Network Function Virtualization (NFV) and Software Defined Networks (SDN) are the critical technologies for network slicing [18]. **Network Function Virtualization** is the virtualization of network functions on the shared hardware, examples of which include firewalls, VPNs, and 5G Evolved Packet Core functions among others. ETSI has published a standard [19] for NFV, which is widely accepted in both academia and the industry. In Figure 2.3, three layers of network slices are shown. In the lowest layer, which is called VNFI, there is the network's physical infrastructure. This layer consists of storage, computing and network hardware, hypervisors, and virtual machines that run on these physical infrastructures. The middle layer is virtual network functions (VNFs) that provide various functionalities in the network [17]. The uppermost layer is Operational Support Systems and Business Support System (OSS/BSS). The OSS/BSS layer consists of management functions for network operators, such as inventory, service provisioning, network configuration, and fault management.

These layers are managed by the network's Management and Orchestration (MANO) component. This component manages the lifecycle of the components of these layers. Both 3GPP and ETSI have published standards for these tools. In 3GPP standards, lifecycle management, performance monitoring, and self-organized network concepts in network slicing are mentioned [20]. There are several tools and projects to manage and orchestrate the network. However, there are two projects with a highly active community and developer support. The first project, which is called ONAP, was developed by Linux Foundation. The second one by ETSI is an open-source project called OSM. [21] compares these projects and concludes that OSM has far less resource footprint than ONAP.

Software-defined networks (SDN) decouple the data plane from the control plane. While the control plane decides the path of packets over the network, the data plane provides the flow of the packets in this path. One of the essential benefits of using SDN is that it significantly decreases the capital expenses of the operator. Since the control plane is separated into the cloud, the operator can use less expensive switches in the transport network. SDN also has advantages while managing the network.

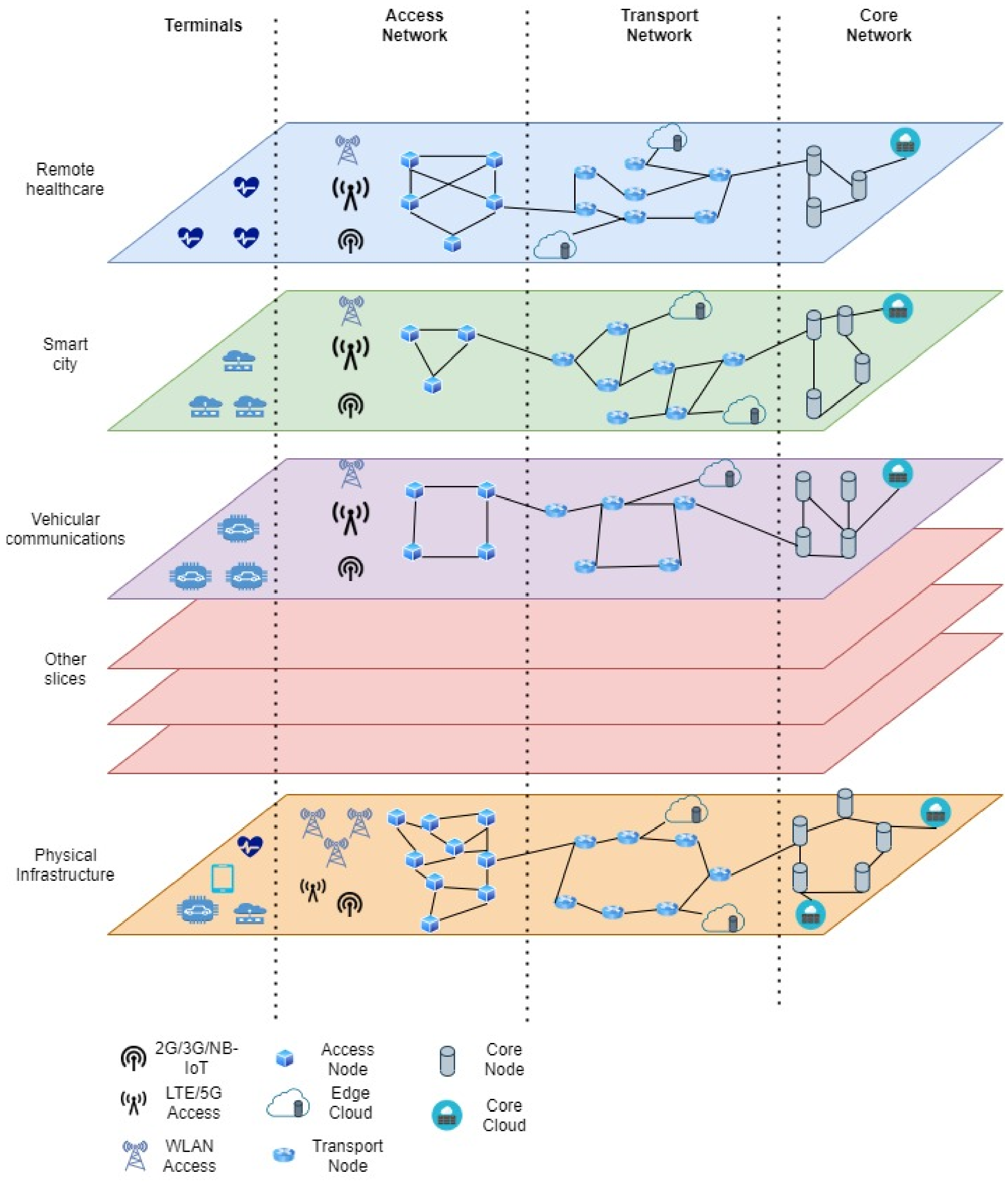

Figure 2.2: Network Slicing Architecture

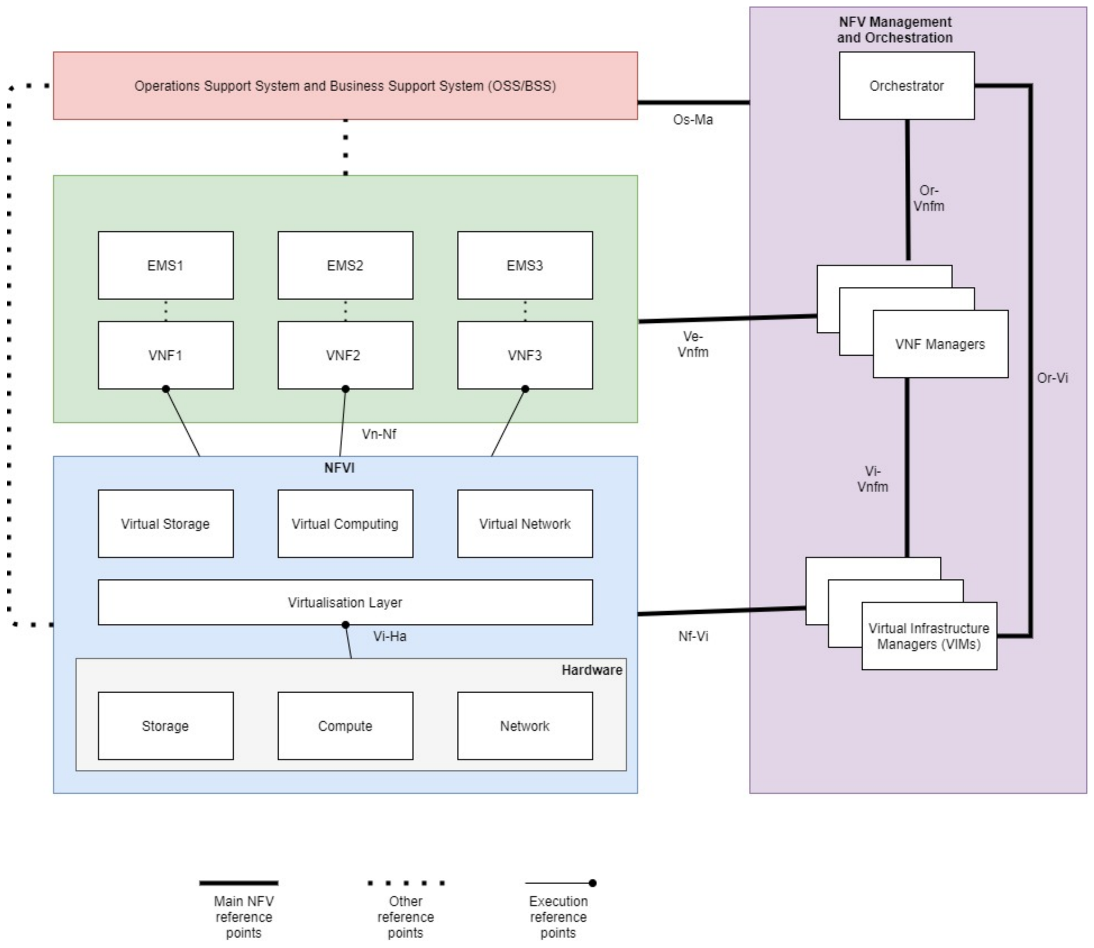

Figure 2.3: ETSI NFV Architecture

Since the controller can monitor the traffic in the network, it can interfere with any suspicious movement and hence maintains security. In addition, due to only one controller in the SDN architecture, any policy configuration could be deployed more efficiently compared to the classical network system that has multiple controllers [18].

2.3 Deep Reinforcement Learning

The advances in deep learning have accelerated the progress in Reinforcement Learning (RL), and the use of deep-learning algorithms within RL has emerged as a new concept entitled deep reinforcement learning (DRL). DRL is considered a leveraging technology for artificial intelligence (AI) that assists autonomous systems through better interaction with the surrounding environment [22]. DRL algorithms have been utilized in fields ranging from helping robots learn behavioral skills with minimum human interaction [23] to new generation networks for more optimized resource allocation [24]. Below we first introduce the basics of RL and continue with Deep Q-Learning concepts.

2.3.1 Reinforcement Learning Basics

Kai et al. [22] introduced the basics of an RL setup. Machine learning-driven autonomous agents learn to modify their behaviors according to the received rewards through observations and interactions with the environment. An RL agent observes a state s_t from the environment at time t. Then, the agent reacts to the environment by taking an action a_t, and the agent changes its state as s_{t+1} according to the action. An agent takes the most reasonable action according to the state, which contains all of the necessary information related to the environment. The agent's action is determined by the scalar rewards r_{t+1}, which are given as feedback by the environment at a certain state. The agent aims to adopt a strategy (optimal policy, π) that gives the maximum expected cumulative discounted return (value function) calculated by the

Bellman equation :

$$V^{\pi}(\hat{s}) = E_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k}R(s^{(k)},\pi(s^{(k)}))|s^{(0)}=\hat{s})\right] \tag{2.1}$$

$$= E_{\pi}\left[R(\hat{s},\pi(\hat{s}))) + \gamma\sum_{s'\in\mathcal{S}}P(s'|\hat{s},\pi(\hat{s}))V^{\pi}(s')\right] \tag{2.2}$$

2.3.2 Deep Q-Learning

As Li et al. discussed in [24], Q-learning is an off-policy and model-free RL algorithm. It consists of three steps. First, the RL agent takes action under the current state according to a greedy policy ϵ, so that the agent chooses the action with the maximum Q-value with a probability of ϵ and chooses other actions with a probability of $\frac{1-\epsilon}{|A|}$, where |A| is the cardinality of the action space. Then the agent takes its reward as $R(s,a)$ and transits to the next state s'. Finally, the agent updates the Q-value function as: $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s,a) + \gamma\max_{a'}Q(s',a') - Q(s,a))$

For a large state space, function approximation methods are used to store the estimated value function, that is, the Q-value function approximated by a linear combination of n orthogonal bases:

$$\boldsymbol{\psi}(s,a) = \{\psi_1(s,a),\cdots\psi_n(s,a)\} \tag{2.3}$$

$$Q(s,a) = \theta_0 + \theta_1\cdot\psi_1(s,a) + \cdots + \theta_n\cdot\psi_n(s,a) \tag{2.4}$$

$$Q(s,a) = \boldsymbol{\theta}^T\boldsymbol{\psi}(s,a) \tag{2.5}$$

Q-learning means $Q(s,a) = \boldsymbol{\theta}^T\boldsymbol{\psi}(s,a)$ must be as close as possible to the learnt target value $Q^{+}(s,a) = \sum_s P(s'|s,a)\big[R(s,a) + \gamma\max_{a'}Q^{+}(s',a')\big]$ for all state/action pairs.

The suitability of deep Q-learning (DQL) for both Experience Replay and Network Cloning functionality of Neural Networks (NN) makes it a perfect candidate for the resource allocation problem. For an Experience Replay operation, the agent stores experience parameters (state, action, next state, reward) at a specific episode t in a dataset. Then, it can use these experiences to update the Q-value NN. On the other hand, the agent uses a separate network $\hat{Q}$, and it is replaced with Q at a certain period of episodes, where $\hat{Q}$ is used for the selection of an action a in a state s.

DQL can gather the pairs of states, rewards, and actions and train its policy in the background [24]. Also, the stored policies can be applied to similar scenarios. That is, DQL can make the resource allocation according to its past experiences, i.e., already learned policies [24].

2.4 Deep Reinforcement Learning Algorithms

There are several DQN agents in the state-of-the-art. These agents proved themselves by solving discrete action-based reinforcement learning problems. Therefore, we implemented four different versions of DQN agents in our proposed solution to compare their results. Table 2.1 summarizes the hyperparameters used in these algorithms. We selected them after several tries on training agents and chose them according to both convergence of the algorithms and the convergence rate shown in the results section.

Table 2.1: Agent Hyperparameters

Agent Hyperparameter	**Value**
learning rate	0.001
replay buffer batch size	256
replay buffer size	1000000
initial epsilon	1.0
discount rate	0.999
tau	0.1
linear hidden nodes	256, 1024 ,512
optimizer	Adam
loss function	L2
gradient_clipping_norm	0.7
incremental td error	1e-8
alpha prioritised replay	0.6
beta prioritised replay	0.1

2.4.1 Deep Q-Learning with Fixed Q-targets

The first agent we implemented in order to test the written environment is from the Deepmind paper [25]. Algorithm 1 shows the process of DQN with a fixed Q target agent. Improvement of this algorithm to the classic DQN uses a second network called the target network. Using a second network resolves the problem known as moving Q-targets. In classic DQN, only one network is used for training, and the result of this approach is that there will be a high correlation between the Temporal Difference target and the parameters we are using. Thus, we change Q values and the target value when we train our data, causing oscillations during the training process. Using a second target network with different weights and calculating the Temporal Difference target with this network is proven to solve this issue. Weights of this target network are updated every *K* step with the initial (evaluation) network's weights. We implemented our DQN with a fixed Q-targets agent according to this approach.

Algorithm 1 DQN with fixed Q targets

```
Initialize network Q with random weights $\omega$
Initialize target network $Q^*$ with random weights $\psi$
Initialize replay memory M with a given capacity
episode = 0
while episode < episode count do
    episode += 1
    Initialize environment with randomly allocated vnfs on the server
    while not done do
        With probability $\epsilon$ select a random action $a_t \in A_t$
        otherwise select $a_t$ = $argmax_a$Q($s_t$,a; $\omega$ )
        Conduct action $a_t$ on environment retrieve reward $r_t$ and next state s'
        Store $(s_t, a_t, r_t, s', done)$ on replay buffer
        Sample random minibatch from replay buffer M
        for all experience in minibatch do
            $$y_i = \begin{cases} r_i + \gamma max_{a'} Q^*(s_{t+1}, a'; \omega), & \text{if not done} \\ r_i, & \text{otherwise} \end{cases}$$
            Loss = 1/N $\sum_0^N (Q(s_i, a_i) - y_i)^2$
            Update Q using the SGD algorithm minimizing loss
            every K step copy weights from Q to $Q^*$
        end for
    end while
end while
```

2.4.2 Double DQN (DDQN)

As an update to the DQN algorithm, Double DQN implements the following: The DQN agent, which we implemented earlier, leads to over-optimistic value estimates due to the max operator using the same values to both select and evaluate action. DDQN overcomes this problem by using a second neural network that has a copy of the last episode's weights. With (2.6), while one network chooses the action, the other could evaluate the action and become a solution to the maximization bias [26]. Algorithm 2 shows the update to the DQN with Fixed Q Targets we implemented.

$$Q^*(s_t, a_t) = r_t + \tau Q_\omega(s_{t+1}, argmax_{a'} Q_\psi(s_{t+1}, a')) \tag{2.6}$$

Algorithm 2 Double DQN

```
Initialize network Q with random weights $\omega$
Initialize target network $Q^*$ with random weights $\psi$
Initialize replay memory M with a given capacity
episode = 0
**while** episode < episode count **do**
    episode += 1
    Reset environment with randomly allocated vnfs on servers
    **while** not done **do**
        With probability $\epsilon$ select a random action $a_t \in A_t$
        otherwise select $a_t$ = $argmax_a$Q($s_t$,a; $\omega$ )
        Conduct action $a_t$ on environment retrieve reward $r_t$ and next state s'
        Store $(s_t, a_t, r_t, s', done)$ on replay buffer
        Sample random minibatch from replay buffer M
        **for** all experience in minibatch **do**
            Compute target Q value:
                $Q^*(s_t, a_t) = r_t + \gamma Q_\omega(s_{t+1}, argmax_{a'} Q_\psi(s_{t+1}, a'))$
            Loss = 1/N $\sum_0^N$ ($Q^*(s_t, a_t)$ - $Q_\omega(s_t, a_t))^2$
            Update Q using the SGD algorithm minimizing loss
            update $\psi$ = $\tau$ * $\omega$ + (1-$\tau$)*$\psi$
            every K step copy weights from Q to $Q^*$
        **end for**
    **end while**
**end while**
```

2.4.3 DDQN with Prioritised Experience Replay

Another update to the DDQN agent is using prioritized experience replay as a replay memory. As one might guess, some experiences would have more importance in training than others. So we retrieved the more valuable experiences with the probability calculated in the paper [27]. Algorithm 3 shows the updated approach.

Algorithm 3 DDQN with prioritised replay memory

```
Initialize network Q with random weights $\omega$
Initialize target network $Q^*$ with random weights $\psi$
Initialize replay memory M with replay period K, alpha value $\alpha$, beta value $\beta$
episode = 0
while episode < episode count do
    episode += 1
    Reset environment with randomly allocated vnfs on the server
    while not done do
        With probability $\epsilon$ select a random action $a_t \in A_t$
        otherwise select $a_t$ = $argmax_a$Q($s_t$,a; $\omega$ )
        Conduct action $a_t$ on environment retrieve reward $r_t$ and next state s'
        Store $(s_t, a_t, r_t, s', done)$ on replay buffer with M maximal priority $p_t$ = $max_{i<t}p_i$
        if t = 0 mod K then
            for all experience in minibatch do
                Sample transition i P(i) = $\frac{p_i^\alpha}{\sum_i p_i^\alpha}$
                Compute importance-sampling weight $w_i = \frac{(N*P(i))^{-\beta}}{max_i w_i}$
                Compute TD-error:
                $\delta_i = r_i + \gamma_i Q^*(s_{i+1}, argmax_a Q(s_{i+1}, a)) - Q(s_i, a_i)$
                Update transition priority $p_i$ = $|\delta_i|$
                Accumulate weight-change $\Delta$ = $\Delta + w_i * \delta_i * \nabla_\omega Q(s_i, a_i)$
            end for
            Update weights $\omega + = \tau * \Delta$
            every K step copy weights from Q to $Q^*$
        end if
    end while
end while
```

2.4.4 Dueling DDQN

The Dueling DDQN agent calculates both the state value function and advantage function, as can be seen in 2.7. Q values are calculated by the goodness of being in a state and the advantage of taking an action in that state. When all actions have a similar advantage, the Q value would be equal to the value of the state. Thus, actions that do not have a particular effect on the Q value will not be taken into consideration [28]. Algorithm 4 illustrates the described approach.

$$
\begin{aligned}
Q^*(s_t, a_t; \omega, \alpha, \beta) = V(s; \omega, \beta) + (A(s, a; \omega, \alpha) \\
-\frac{1}{|A|} \sum_{a'} A(s, a'; \omega, \alpha))
\end{aligned}
\tag{2.7}
$$

Algorithm 4 Dueling DDQN

```
Initialize network Q with random weights $\omega$
Initialize target network $Q^*$ with random weights $\psi$
Initialize replay memory M with a given capacity
episode = 0
while episode < episode count do
    episode += 1
    Initialize environment with randomly allocated vnfs on the server
    while not done do
        With probability $\epsilon$ select a random action $a_t \in A_t$
        otherwise select $a_t$ = $argmax_a$Q($s_t$,a; $\omega$ )
        Conduct action $a_t$ on environment retrieve reward $r_t$ and next state s'
        Store $(s_t, a_t, r_t, s', done)$ on replay buffer
        Sample random minibatch from replay buffer M
        for all experience in minibatch do
            Compute target Q value
            $Q^*(s_t, a_t; \omega, \alpha, \beta)$ = $V(s; \omega, \beta)$ + $(A(s, a; \omega, \alpha)$-$\frac{1}{|A|} \sum_{a'} A(s, a'; \omega, \alpha))$
            Loss = 1/N $\sum_0^N$ $(Q^*(s_t, a_t)$ - $Q_\omega(s_t, a_t))^2$
            Update Q using the SGD algorithm minimizing loss
            update $\psi$ = $\tau$ * $\omega$ + (1-$\tau$)*$\psi$
            every K step copy weights from Q to $Q^*$
        end for
    end while
end while
```

We implemented all of the described DQN agents in this work, and they all converged into a model that could decrease the energy consumption in a network slicing problem with stringent security requirements. In the next chapter, we describe the proposed architecture and explain how we used these agents to increase energy efficiency.

2.5 Related Work

Network slicing is a new technology proposed to support effective management of 5G networks and beyond, for which many research and development efforts are still in progress, with no mature standards in place. 5G is expected to provide increased performance as compared to the previous generation of mobile networking technologies. As the utilized bandwidth gets wider, connected devices get more extensive in number, and data rates become faster, the energy needed for these operations will also increase. Therefore, it is a significant research problem to optimize the energy consumption of 5G networks while providing the required services with the required QoS. Below we provide an overview of existing approaches that address the aspects of network slicing that we focus on in this work.

In [29], a security-aware slice instance allocation model for 5G core networks was proposed. Security limitations, such as some of the VNFs having to be hosted on the same server and some of the VNFs not being able to coexist on the same server, were given to the ILP solver as constraints. This work showed a trade-off between slice security and embedding performance metrics such as execution time and average revenue cost ratio for accepted requests. Although the authors considered security aspects of resource allocation, they did not look at energy efficiency. [30] considered virtual network security functions (VNSFs) placement as an ILP problem while considering the security and QoS requirements of the network slices. For this purpose, the authors gave total maximum end-to-end latency as a QoS constraint and VNSF execution order, VNSF network position, and operational mode as security constraints. However, energy efficiency optimization was not considered in the model. Guan et al. [31] also implemented an algorithm that places VNSFs onto a network topology using routing characteristics instead of ILP and tested their security performance in a simulation that mimics computer virus and worm attacks on the network. In [8], an Integer Linear Programming model was presented for mobile network slice embedding, resource allocation, and link mapping. Their work aimed to implement a network slicing model maximizing the weights on network slices. In addition to that, resource allocation and link mapping were provided by adding capacity, latency, and graph constraints.

Research has been conducted on energy efficiency in 5G networks in every layer of the 5G architecture, from base stations to radio access networks (RAN) and core 5G networks. In this research, we focus on achieving an energy-efficient secure network slicing scheme on the core network. Energy efficiency in networking is usually considered a fractional programming problem since providing more service with as little energy as possible is, by its very nature, a trade-off problem. Therefore, researchers have approached the energy efficiency issue as a "fraction" to be maximized, where the numerator and denominator are two sides of a trade-off. Nguyen proposed in [32] a hybrid resource allocation scheme that considers spectrum allocation, interference alignment, and energy efficiency simultaneously since all three are essential for providing good performance in the network. In [33], Matthiesen et al. developed a QoS framework for a sliced radio network (RAN), where two network parameters were considered: throughput and energy efficiency. They built Pareto boundaries of two different algorithms based on utility profile and scalarization, respectively. In [34], the researchers focused on the energy efficiency vs. delay trade-off problem in wireless network virtualization. They modeled the issue as a stochastic optimization problem with predefined delay constraints, where users are queued on virtual base stations.

Mathematical optimization is not the only approach used in optimizing energy consumption in networking. With the advancements in hardware technology and data science in recent years, reinforcement learning-based models have proven successful in network resource allocation and optimization. [35] proposed an algorithm that considers both energy efficiency and spectral efficiency of the network while using Dueling Deep Q-Network and shows successful results compared to Q-learning and DQN. In [36], base stations' sleep modes were optimized with the help of Q-Learning, a variant of reinforcement learning. The aim was to find the optimal operating duration of base stations concerning the delay and energy consumption requirements and activate and deactivate them accordingly. Laroi et al. [37] developed a VNF slice placement algorithm for core networks using Deep Reinforcement Learning (DRL) algorithms as well as an ILP algorithm and compared their performances. Results showed that the DRL model consumes less energy and time than ILP and reinforcement learning algorithms. However, they did not consider the security aspects of VNF placement. Particularly in network slicing, there have been many works optimizing the resource allocation between network slices [38], [39], [40], though not particularly on optimizing the energy consumption in network slices.

2.5.1 Open Issues and Challenges

Although there exist approaches focusing on optimal network slicing/virtual network embedding, none consider both energy efficiency and security constraints of the network. As we show later in the following chapters, one should consider security while optimizing network slicing since security comes with a cost. Our approach handles the VNF allocation problem more comprehensively than other approaches in the state-of-the-art. It solves the secure slice embedding problem while minimizing energy consumption.

CHAPTER 3

PROPOSED ILP-BASED OPTIMIZATION MODEL

This chapter introduces the proposed ILP model for secure VNF placement in core network slicing, which aims to minimize energy consumption while meeting the given memory, throughput, and latency requirements. Our model is an extension of the model proposed by Fendt et al. [8]. The primary purpose of the VNF placement problem is to map VNFs into servers in the network in the most effective manner. One of the main challenges in this virtual network embedding problem is solving this embedding problem in a security-aware way. The presented mathematical model can be used with an Integer-Linear Programming Solver, such as Gurobi[1], which is used in this work.

In the thesis study, we started with the ILP-based model and extended it with some changes explained later in this chapter. Thus, in the first part of this chapter, we describe the initial version of the ILP-based model published in [9]. After that, we explain an extended version of this algorithm that is compared against Deep Reinforcement Learning based optimization algorithms.

3.1 Initial Version of the ILP-based Optimization

3.1.1 Definitions

This section covers essential parameters and definitions before delving into the details of the ILP model.

An undirected graph G is an ordered pair (V, E), in which V stands for the set of v

[1] Gurobi - The Fastest ILP Solver https://www.gurobi.com/

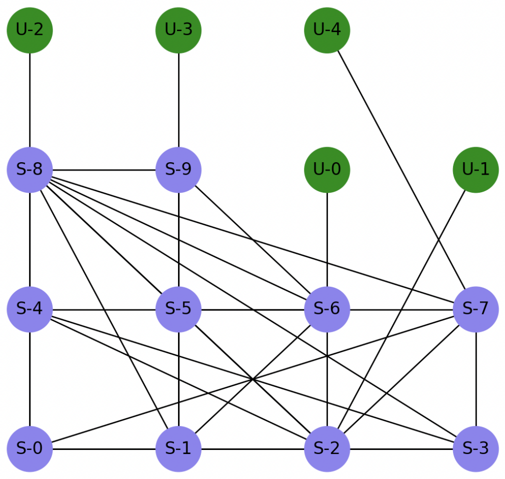

Figure 3.1: Example network that consists of server and user equipment nodes

nodes in the graph, and E stands for the edges. Each e_{ij} represents an edge between two nodes v_i and v_j that are connected with each other.

$$V = \{v_1, v_2, ..., v_k\} \tag{3.1}$$

$$E = \{e_{12}, e_{23}, ..., e_{ij}\} \text{ , where } v_i, v_j \in V \tag{3.2}$$

$$P_{ij} = \{e_{ir}, ..., e_{tj}\}, \text{where } v_i, v_r, v_t, v_j \in V \tag{3.3}$$

P, an ordered set of edges, describes a path in the graph G. Note that P is a subset of V. P_{ij} defines a path between nodes v_i and v_j.

An illustration of an example network graph can be found in Figure 3.1, and Figure 3.2 shows an example placement of a VNF in the network. The colored lines between user equipment nodes and VNF show the paths used for connections between them. The paths are organized using the constraints and requirements, which is why the paths used are not the shortest. For example, although there is a shorter path `U-2, S-8, VNF` in Figure 3.2 between node `U-2` and `VNF`, our algorithm picked the path `U-2, S-8, S-1, VNF` to meet the constraints and requirements.

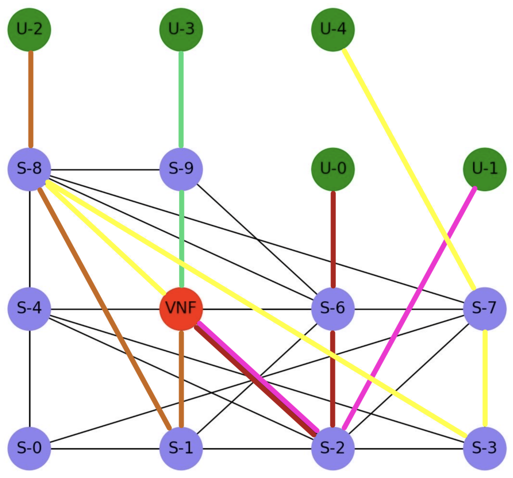

Figure 3.2: Example VNF placement that consists of paths between VNF and user equipment nodes

3.1.2 Model Parameters

In the proposed model, it is assumed that a physical network contains several server nodes and user equipment nodes. The main target of this solution is embedding VNFs into the server nodes to optimize the total energy consumption under the given security requirements and QoS constraints.

The set Δ defines the list of VNFs which will be used in the network slices.

$$\Delta = \{\delta_0, \delta_1, ..., \delta_m\} \tag{3.4}$$

$$L_k = \{l_0, l_1, ..., l_i\} \tag{3.5}$$

$$N_k = (V, E, \Delta, L) \tag{3.6}$$

Let us call a VNF δ, and let Δ be a set of VNFs. We can define the kth network slice, namely $N_k = (V, E, \Delta, L)$, where V stands for nodes, E for physical links, and Δ for virtual network functions, and L for virtual links that connect user equipment nodes with server nodes. A virtual link $l \in L$ is a set of edges (e) that connects a user equipment node to a VNF running in a server node in the graph.

$$F_k = \{\delta_r, ..., \delta_m\} \quad (3.7) \qquad\qquad LS_k = \{\delta_r, ..., \delta_m\} \quad (3.8)$$

The equations 3.7, 3.8 are used for providing security constraints, which will be covered in the following section.

The parameters of the model are given below:

- N_k: kth network slice.
- s_w: wth server node.
- u_v: vth user equipment node.
- e_j: jth physical link.
- F_o: Forbidden set of VNF o.
- LS_m: Locate-Same set of VNF m.
- $\delta_{k,m}$: mth VNF which runs in the kth slice.
- $l_{k,i}$: ith virtual link in kth slice, which connects a user equipment node and a VNF. This value is computed using the the sets s_w, u_v, and $\delta_{k,m}$.
- $P^{\delta}_{k,m}$: the additional energy consumption of the VNF δ_m in the kth network slice, in addition to P^s_w.
- $M^{\delta}_{k,m}$: the memory usage of the VNF m in the kth network slice.
- $R^{\delta}_{k,m}$: the average requests usage of the VNF m in the kth network slice.
- $T^l_{k,i}$: the instantaneous throughput of the ith virtual link in kth slice, $l_{k,i}$.
- $L^l_{k,i}$: the maximum latency of the ith virtual link in the kth slice, $l_{k,i}$.

The parameters below are the constraints that will be used during the optimization process.

- M_w^s: maximum memory of the server node s_w.
- R_w^s: maximum number of requests of the server node s_w.
- P_w^s: energy consumption of a virtual network function running in the server node s_w.
- T_j^e: maximum throughput of the physical link e_j.
- L_j^e: maximum latency of the physical link e_j.

These static parameters are part of the graph and constraints and cannot be changed after the infrastructure has been set. In addition, some related dynamic parameters change in the embedding process.

The variable $\Phi_{r,j}$ defines a physical link e_j used for constructing P_r

$$\Phi_{r,j} = \begin{cases} 1 & \text{if } P_r \text{ uses } e_j \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

The integer variable in 3.9 is not a part of the proposed ILP model. It does not affect the proposed solution. On the other hand, this variable is changed after the model finds the optimized solution. It defines whether path P uses physical edge e_j. By using this variable, the visualized version of the network topology is created, which will be demonstrated below.

3.1.3 Decision Variables

In the proposed ILP model, binary and linear decision variables whose values change during the optimization process are the key points of ILP. By changing the values of decision variables, the ILP model tries to find the optimized solution from among the feasible solutions.

Here are some variables that have been adapted from [8] to our model:

$$\mu_{k,m,w} = \begin{cases} 1 & \text{if } \delta_{k,m} \text{ is mapped on } s_w \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

$$\rho_{k,i,r} = \begin{cases} 1 & \text{if } P_r \text{ is used in } l_{k,i} \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

The integer variable in 3.10 decides whether a virtual network function δ is mapped on a server node s_w. In addition, the variable in 3.11 performs the same operations on virtual links over a physical path.

3.1.4 Objective Function

As mentioned earlier, the purpose behind the objective function is to minimize energy consumption while mapping VNFs and server nodes by meeting the requirements and limitations. In 3.12, there exists a static energy consumption of every VNF if it is mapped to a server node, which increases the overall energy consumption of the network.

$$\min \sum_k \sum_m \sum_w [(P_w^s + P_{k,m}^\delta) \cdot \mu_{k,m,w}] \tag{3.12}$$

3.1.5 Optimization Constraints

3.1.5.1 Graph constraints

$$\sum_w \mu_{k,m,w} = 1 \; \forall k, m \tag{3.13}$$

$$\sum_{P_r} \rho_{k,i,r} = \mu_{k,m,w} \; \forall k, i \;\; \text{where} \;\; l_{k,i} \text{ is a link from } s_w \text{ to } \delta_m \tag{3.14}$$

The constraint in 3.13 maps every virtual network function to a server node in the network slices so that every virtual network function is mapped to that slice at least once.

The constraint in 3.14 ensures that the virtual links that connect virtual network functions in slices and user equipment nodes are synchronized with the paths.

3.1.5.2 Capacity constraints

$$\sum_{k}\sum_{i}[(\sum_{r}\rho_{k,i,r}\cdot\Phi_{r,j})\cdot T^{l}_{k,i}]\leq T^{e}_{j},\forall j \tag{3.15}$$

$$\sum_{k}\sum_{m}(\mu_{k,m,w}\cdot M^{\delta}_{k,m})\leq M^{s}_{w},\forall w \tag{3.16}$$

$$\sum_{k}\sum_{m}(\mu_{k,m,w}\cdot R^{\delta}_{k,m})\leq R^{s}_{w},\forall w \tag{3.17}$$

The constraints above guarantee that if a VNF is mapped on a server node, that server node must meet the QoS needs of the VNF. In addition, these requirements cannot exceed the limitations of server nodes. Every server node has predefined throughput, energy usage, and memory usage limits. These are given as static data. Every VNF has a throughput requirement that needs to be provided by server nodes. The constraint in 3.15 ensures that the throughput on every virtual link does not exceed its maximum value. In addition to that, for every VNF, there exist memory requirements that should be maintained by the mapped server node. This gives the constraint in 3.16, which ensures the maximum memory usage does not exceed its maximum possible value. Similarly, 3.17 ensures that the maximum required number of requests for each server node does not exceed the total number of requests in the server node.

3.1.5.3 Latency constraints

$$\sum_{j}[\sum_{r}(\rho_{k,i,r}\cdot\Phi_{r,j})\cdot L^{l}_{j}]\leq L^{e}_{k,i}\ ,\forall k,i \tag{3.18}$$

One of the critical points in network slicing is that while mapping the server nodes and network functions, the latency requirements must be met. The constraint in 3.18 ensures that the latency in every virtual link is in the required range.

3.1.5.4 Security Constraints

$$\mu_{k,m,w} \cdot \mu_{k,o,w} = 0 \ , \ \forall o \in F_m \tag{3.19}$$

$$\mu_{k,m,w} = \mu_{k,o,w} \ , \ \forall o \in LS_m \tag{3.20}$$

Inspired by [29], two different static sets are defined to meet security requirements, including Forbidden and Locate-Same. If VNF m contains VNF o in its forbidden set, these VNFs will not be placed on the same server node. If VNFs create a security vulnerability for each other or for a server node, then the Forbidden sets in 3.19 of those VNFs will contain other VNFs that create a vulnerability. The Locate-Same set works entirely in the opposite way. VNFs may need other network functions to have a secure environment. For example, VNFs may require a firewall in the same server node to have a secure environment. Then, in the Locate-Same sets in 3.20, these VNFs will contain the network functions that are required for them. These are static sets that never change while the proposed solution is running. These sets will change the placements of VNFs so that they will meet the security requirements given with these Forbidden and Locate-Same sets.

3.1.6 ILP Model

The proposed model uses the constraints in the previous section and creates the network topology by optimizing the objective function. While doing this, graph, capacity, latency, and security constraints are maintained by the optimizer. The network topology generated using the ILP model is a system that meets all the requirements and constraints and also a system that has minimum energy consumption in given parameters. In addition to that, the wanted VNF types, the memory, energy, or latency constraints can be defined for each server node, physical link, or slice separately.

3.2 Extended version of the ILP-based Optimization

There have been a couple of changes in the extended version that compares the ILP-based model to the DRL-based model. First, to have an ILP model closer to real-world applications, the number of times VNFs are mapped on core nodes can be higher than one. As a result, one of the constraints in the ILP model is changed. In the initial model [9], the first graph constraint indicates a VNF can be mapped at most once. We change it to the following constraint in the updated version:

$$\sum_{w} \mu_{k,m,w} \geq 1 \ \forall k, m \tag{3.21}$$

Regarding security constraints, in the proposed solution, the Locate-Same set and constraint are removed from the ILP model to ease comparison with the DRL-based model. In addition, we add TEE and attestation constraints to increase the reliability of the security-aware ILP version. For this purpose, we hold an attestation and TEE variable in every VNF type. Then, we map every VNF's TEE and attestation value for every server and slice.

CHAPTER 4

PROPOSED DEEP REINFORCEMENT LEARNING BASED OPTIMIZATION MODEL

This thesis proposes a DRL model that optimizes energy consumption in network slicing of a 5G core network. The proposed model is implemented using a custom OpenAI Gym environment. We represent the 5G network topology as a connected graph on the networkx[1] library. As we will explain in detail in the experimental evaluation section, the proposed architecture is tested with three different network topologies and configurations. This section will explain the server graph structure and the security constraints we implemented. We will also demonstrate agents' actions on the environment and how we calculate the comparison results.

Starting with the network graph represented in Figure 4.1, each graph vertex is considered as a server with VNFs running on it. Servers in the environment have idle energy usage, and each VNF that works on these servers also has a VNF energy usage. These energy consumption values are taken from [41]. As seen in Table 4.1, servers have limitations regarding their VNF capacity. We set the server capacity value to 40 for all servers in the environment for ease of calculation. Furthermore, virtual network functions that run on these servers also have limitations on how much traffic they can handle in each timestep, which can be seen in the VNF package capacity column in Table 4.2. So, all servers have limitations on how much traffic they can handle in a timestep. In addition, servers are capable of having different types of VNFs running on them. However, we established security constraints on co-locating these VNFs, where only some of them can coexist on the same physical server. For example, VNFs that do not directly need to be isolated in a network slice could be

[1] Software for complex networks - https://networkx.org/

running on the same server. VNFs that could coexist on the same server can be seen in Table 4.2. Also, some of the VNFs could be desired to run on a server with a trusted execution environment and servers that provide attestation. All of the requirements and limitations on VNFs can be examined in Table 4.2. The initial configuration of the servers is stochastically generated. Each VNF count is set to a value between 0 and server VNF capacity if they can coexist with the other VNF types on the server, and the server holds the trusted execution environment and attestation requirements of that VNF type. Servers also hold attestation and trusted execution environment values since some network slices could desire these properties on the server.

Table 4.1: Test Scenarios

	Server Count	**Server Capacities**	**Slice Type Count**	**VNF Type Count**	**Incoming Traffic per timestep**
Small Network	5	40	3	3	0-100
Medium Network	10	40	3	3	0-100
Large Network	30	40	3	5	0-100

Table 4.2: VNF Capacities

VNF Type id	**VNF Package Capacity**	**Coexist**	**TEE requirement**	**Attestation requirement**
VNF1	25	vnf1,vnf2,vnf4	False	False
VNF2	50	vnf1,vnf2	True	False
VNF3	100	vnf3,vnf4	True	True
VNF4	50	vnf1,vnf3,vnf4	False	False
VNF5	50	vnf5	True	True

Edges of this graph represent links between servers. They have throughput and latency attributes, so traffic mapping can be realistically done in terms of throughput and latency-related QoS requirements. These latency and throughput values are taken from [42]. While constructing the network graph, the throughputs of these links are randomly set between 400 and 1000 Gbps [42] as it was claimed to be in this range in the core network. In addition, each link's latency values are randomly set between 1 to 3 ms again based on [43].

One of the main contributions of the proposed model to the state-of-the-art is that

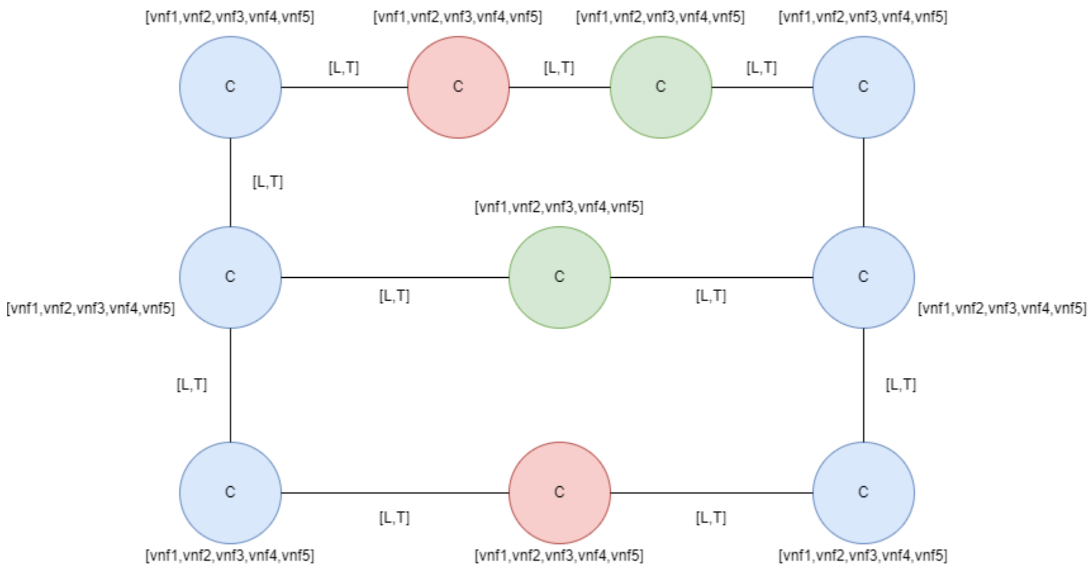

Figure 4.1: Graph Representation

we introduce security-related constraints in the environment. There are three main security constraints we implemented for this architecture. As can be seen in Table 4.2 some of the VNFs cannot coexist on the same servers. We put this constraint in the model because some of the VNFs could be required to have strict hardware isolation requirements. For example, an autonomous vehicles network slice instance could desire to have a firewall VNF to be completely isolated from other slices. Secondly, the environment contains servers with a trusted execution environment. As we explained in Section 2, some network slices could be desired to have a trusted execution environment on the server where VNF will be deployed. Thus, we are generating our graph vertices so that each vertice could have a trusted execution environment with a probability of 0.5. The last security constraint we implemented is that if the server has the trusted execution environment on them, it will also have an attestation property with the probability of 0.5.

The environment maps the incoming traffic with Algorithm 5. Traffic arriving in the environment is mapped to VNFs by this heuristic method. A random amount of traffic is generated and held in the traffic buffer at each timestep. For each packet in this buffer, the heuristic algorithm tries to find an available VNF for each packet's slice type requirements, which can be seen in Table 4.3. For example, eMBB types require vnf1 and vnf3 in one scenario, and traffic mapping algorithms search for those two vnfs in the network graph. For each VNF requirement in a packet, the search algorithm looks at the closest server, and if a VNF is available, it maps the packet to that VNF in that server. If there is no available VNF in the closest server, it searches the neighbors. Hence, the iterative mapping algorithm finds the available VNF at a specific server. Then, if the latency limit is satisfied, it finds the shortest available path from the closest server to the found server. Suppose there are no available paths that satisfy both the links' throughputs and the packet's latency limit. It searches other neighbors until there are no servers left to look at.

Table 4.3: Slice Types and Requirements

Slice Type	**VNF requirements**	**Latency limit**
eMBB	{vnf1,vnf3,vnf4,vnf5}	14 ms
mMTC	{vnf2,vnf4}	10000 ms
URLLC	{vnf2,vnf3,vnf5}	0.5 ms

The DQN agents we showed in the earlier section have a discrete action space, and each discrete value represents how a VNF is deployed or shut down on a specific server. So for each scenario, there would be *server count* $\times$ *VNF count* actions for closing a VNF at a specific server and many actions for opening at a specific server. In each step of the environment, the agent performs an action in the environment concerning the Q values as explained in Section 2. The action predicted by the agent changes the network server structure if and only if it holds all the given security requirements of that VNF. So if a server has VNF types with which the given action VNF type cannot coexist, the action will not be executed, and the server state will remain the same. Therefore, to be conducted, the action should hold TEE and attestation requirements of the action VNF type on an action server, and there should be only VNFs that the action VNF type could coexist with on the action server.

At the end of each step, the reward value is calculated. The reward calculation 4.1 returns 1 if the energy consumption of the overall network is not increasing and 75 percent of the traffic is mapped to the servers. We set 75 percent as a design choice, and this also gives an additional constraint to the models, such as a KPI limitation for traffic mapping. If overall energy consumption increases, we check whether the mapped traffic ratio has increased over the previously mapped traffic ratio. We are letting agents only reward actions that decrease energy consumption by maintaining the mapped traffic ratio or actions that increase the mapped traffic ratio. All other actions would give 0 rewards and finish the current episode. Likewise, in the Atari games that these agents [25–28] proved themselves, agents learn while the episodes are not terminated. We implemented our reward function and how we ended the episodes similar to their approach. If the protagonist is dead or unable to continue the Atari game, they restart the game, or as we can see in the famous cart pole example [44], while the pole does not fall from the cart, the system has rewards, and episodes continue. If energy consumption is increased without changing the mapped traffic ratio, we end the episode. Like in the Atari games and the cart pole problem, the cumulative reward increases while the game is not over. Hence, the done condition for each episode in this architecture is if the current energy consumption is more than the previous energy consumption, the mapped traffic ratio decreases, or the mapped traffic ratio is less than 75 percent. Until the done condition is set or the agent successfully conducts 1000 actions on the environment, the cumulative reward increases by 1 in each step. Thus, the environment's maximum reward is 1000.

$$R_i = \begin{cases} 1, & \text{if not done} \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

So far, we have talked about the DRL approach we implemented. We also implemented an Integer Linear Programming-based optimization algorithm to compare our results. We improved the implemented algorithm in [9] by adding a trusted execution environment and attestation requirement as a constraint. Apart from adding these two conditions, we changed energy consumption values to more realistic ones, as explained in this section. In the ILP version, the model tries to allocate all VNFs to the servers without dynamic environment changes. However, the DRL version we implemented dynamically allocates the VNFs while the environment changes. So, if the traffic ratio is changed in the environment, DRL-based algorithms rapidly react while ILP-based models need to be optimized again.

In the ILP model in [9], we changed the proposed solution so that the DRL version and ILP model meet the exact environment. In the ILP model, the objective function definition and some constraints are changed to calculate energy consumption more accurately. Regarding the objective function, in the objective function in [9], we calculate the energy consumption by checking whether VNFs are mapped on core nodes. The proposed objective function tries to optimize energy consumption by this calculation. Although each server's idle energy consumption values are changed according to the real-life values, 3.12 still calculates the total energy consumption by the same equation.

Algorithm 5 Traffic mapping

```
while traffic_buffer.size()$>$ 0 do
    packet = traffic_buffer.pop()
    for range(required vnf_counts) do
        marked_servers = []
        if closest_server has required vnf and closest_server.vnf_capacity$>$ 0 then
            closest_server.vnf_capacity -= 1
            continue
        else
            marked_servers.append(closest_server)
            non_traversed_graph_nodes = []
            for all neighbors of closest server do
                if marked_servers[neighbor] = false and path_exist(closest server,neighbor) then
                    non_traversed_graph_nodes.append(neighbor)
                end if
            end for
        end if
        while non_traversed_graph_nodes do
            server = non_traversed_graph_nodes.pop()
            if marked_servers[server]=false and server.vnf.capacity $>$ 0 then
                path = shortest_path(closest server, server)
                if valid_path(path,latency limit) then
                    server.vnf_capacity -= 1
                    continue
                else
                    for all paths in the available path(closest_server, server) do
                        if isvalidpath(path,latency_limit) then
                            server.vnf_capacity -= 1
                            continue
                        end if
                    end for
                end if
            end if
            marked_servers[server] = True
            for all neighbors of server do
                if marked_servers[neighbor]=false and path_exist(server,neighbor) then
                    non_traversed_graph_nodes.append(neighbor)
                end if
            end for
            return false
        end while
    end for
end while
```

CHAPTER 5

EXPERIMENTAL EVALUATION

As the earlier sections explain, two different environment versions have been modeled. In this section, first, we will show the results of the initial version of the ILP-based model, and then the results of the extended version of the ILP-based model compared to the DRL-based model explained earlier.

5.1 Simulation Scenarios

5.1.1 Simulation Environment for Initial version of ILP-based Model

To evaluate the ILP algorithm we introduced in the previous section, we have developed a simulation framework using the Gurobi integer linear programming simulation library. We have used Python to implement the algorithm and the simulation environment. We have run the benchmarks on a MacBook Pro 16, 2.6 GHz 6-Core Intel Core i7 with 32 GB RAM. We have also developed a dataset generator to run the simulations on our own generated comprehensive, fully customizable datasets. Below we present the details of the evaluation environment and the results.

In the experimental evaluation, we used various simulation settings with different parameters. The parameters used in the simulation are listed in Table 5.1. Each configuration was used five times to generate 50 different datasets, and we took the average of the metric values we obtained from the simulations.

As a baseline for comparison, we also implemented a greedy approach in Python that performs VNF placement in the network topology. The greedy approach places VNFs in server nodes with the given capacity, security, and graph constraints, without opti-

Table 5.1: Dataset generation parameters used for the simulations

Count of slices	2	Min memory per VNF	16
Count of edge nodes	2	Max memory per VNF	2048
Min number of VNFs per slice	10	Min request count per VNF	100
Max number of VNFs per slice	20	Max request count per VNF	1000
Min additional power usage per VNF	10	Min power usage per server node per VNF	1000
Max additional power usage per VNF	500	Max power usage per server node per VNF	50000
Min latency per physical link	5	Min memory count per server node	2048
Max latency per physical link	25	Max memory count per server node	10240
Min throughput per virtual link	100	Min request count per server node	10000
Max throughput per virtual link	500	Max request count per server node	100000
Min latency per virtual link	50	Min throughput per physical link	10000
Max latency per virtual link	150	Max throughput per physical link	20000

mizing power consumption. Moreover, to measure the cost of the power optimization during the ILP-based optimization process, we have also run our ILP model on 10 of our datasets without an objective function, which does VNF placement under the given constraints.

To verify the correctness of our ILP model and greedy approaches, we have also built a verifier that checks each graph, security, and capacity constraint. The verification code confirms that both the ILP and the greedy algorithms are compatible.

5.1.2 Simulation Environment for DRL-based Model

We tried several scenarios in the environment mentioned above to calculate power consumption over time. The scenarios can be seen in Table 4.1. We constructed three different sets of server counts to calculate the power consumption on small, medium, and large networks. There are only three types of VNFs for simplicity in the small and medium sets. However, in the large set, we put five different VNF types to increase the complexity of the action space of the agent. In the small test set, the agent conducts $2 \times 5 \times 3$ (30) actions on the environment, the medium test set involves conducting $2 \times 10 \times 3$ (60), and the large test set involves $2 \times 30 \times 5$ (300) actions on the environment. These actions are used to decide which VNF will be opened or closed on a specific server.

To show the performance of the trained agent, we also implemented an environment with no security requirements. Small, medium, and large networks were also tested in this environment, which has no security restrictions. Our results show that optimization models led secure versions of the environments to consume less power than those with no security requirements. This is due to the security constraints giving more incentive to agents to disable certain VNFs, leading to better performance for the models.

5.2 Simulation Results

5.2.1 Results for the Initial Version of the ILP-based Model

We have compared the evaluation results of the greedy and ILP algorithms, and the comparison of these two algorithms both from the aspect of time and power can be seen in Figure-5.1 and Figure-5.2. The line plot in Figure-5.1 shows the change in problem-solving time with respect to the server node count. As shown in Figure-5.1, the build time of the ILP algorithm is much more than the optimization time. In addition, when the server node count increases, the build time of this model is also increased. On the other hand, when the server node count increases, the time difference between ILP optimization and the greedy algorithm decreases. The bar plot in Figure-5.2 shows the total power consumption difference between the ILP and greedy algorithms vs. server node count. The greater the value, the better the ILP algorithm outperforms the greedy algorithm.

As seen in the optimization time versus server count in Figure-5.1, both algorithms seem to follow an exponential time complexity for finding the solutions. Also, there is a constant level of difference between the data points of both algorithms, and we can say this difference is because of the model building time of Gurobi, the ILP framework we have used.

On the other hand, the optimization unit difference in Figure-5.2 shows that the differ-

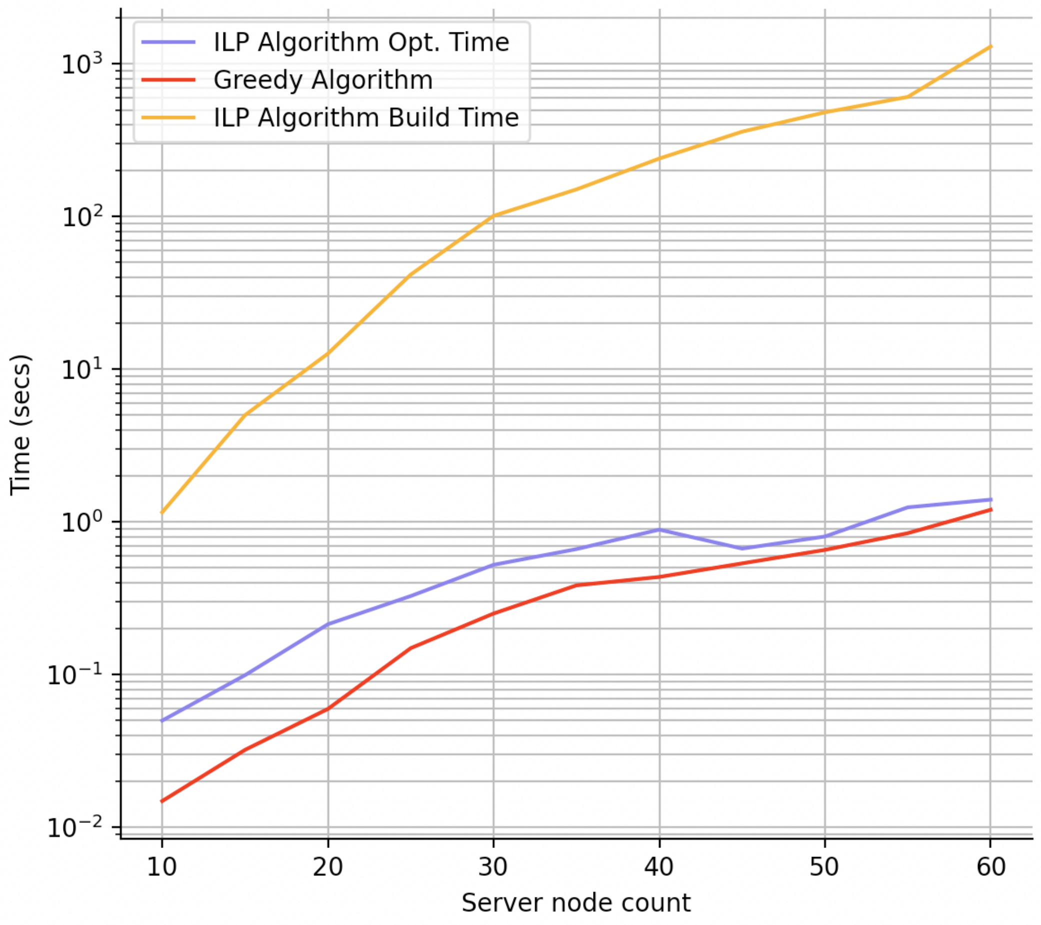

Figure 5.1: Total problem solving time by server node count plot, in logarithmic scale

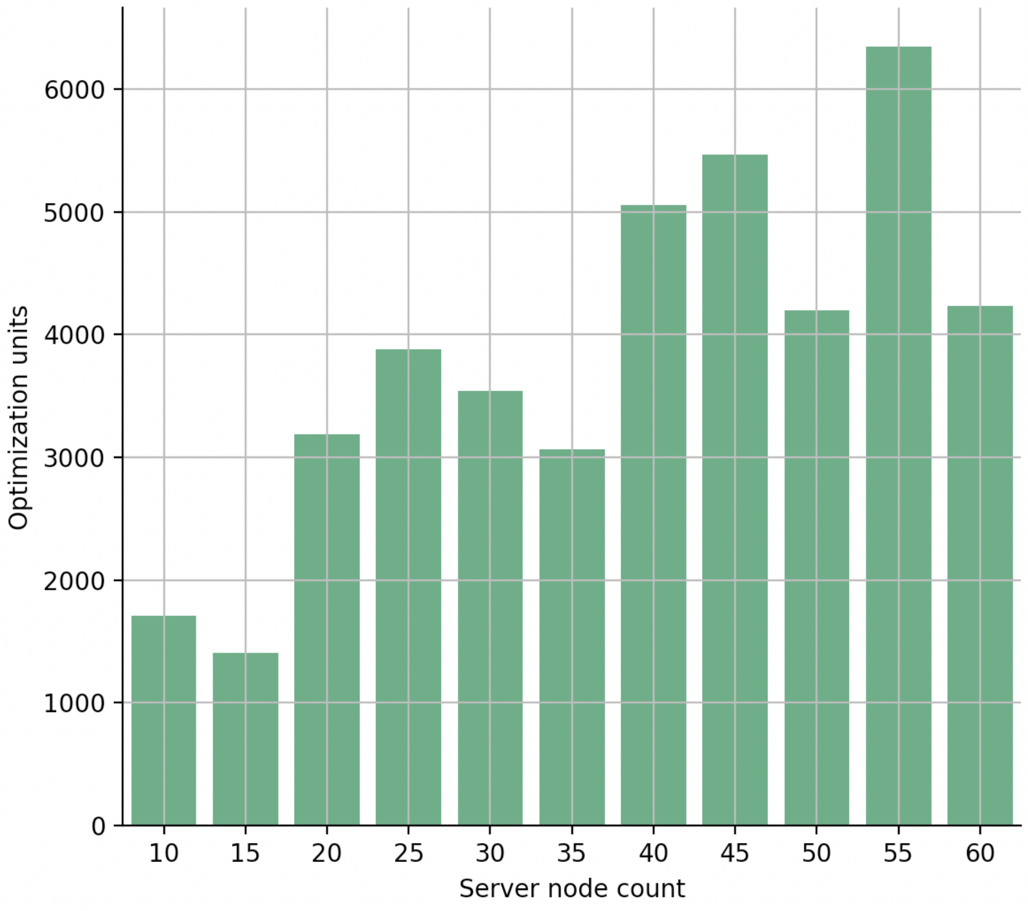

Figure 5.2: Optimization result difference between Greedy Implementation and ILP Implementation by server node count plot

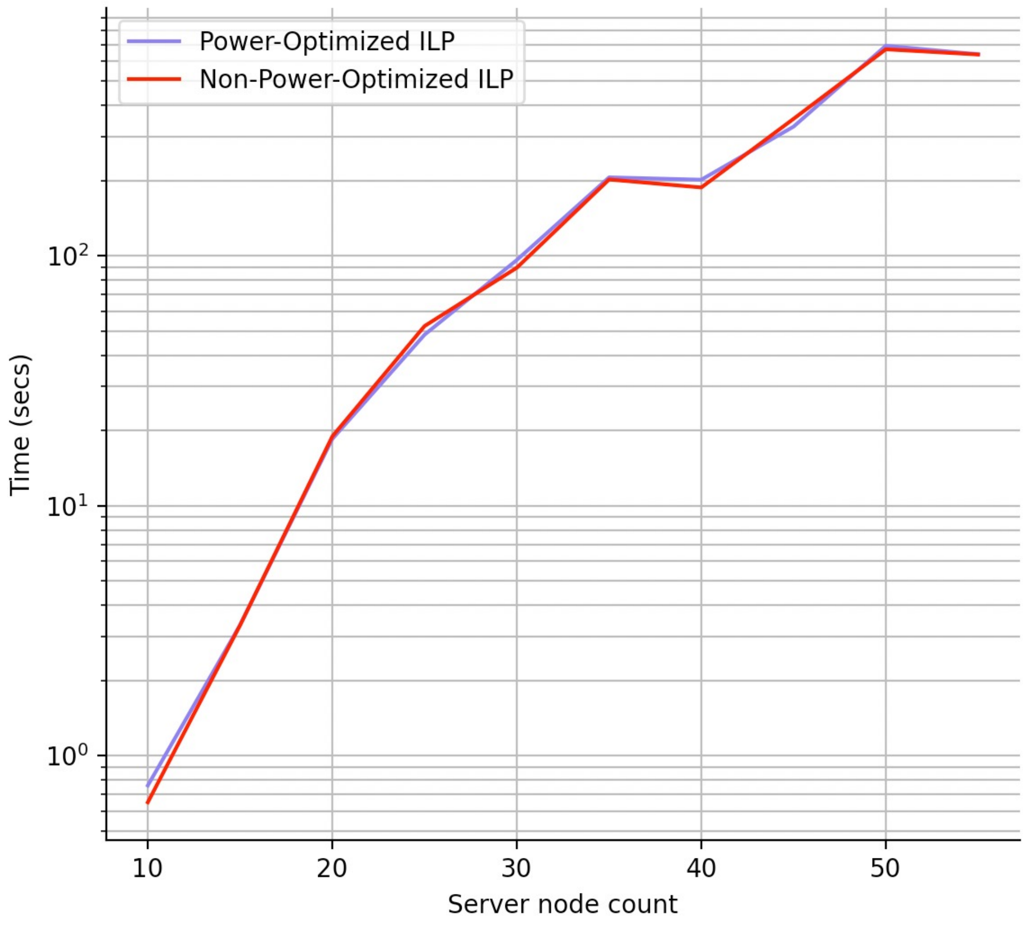

Figure 5.3: Total problem solving times (model build time + model optimization time) for non-power optimized and power-optimized ILP implementation, by server node count plot, in logarithmic scale

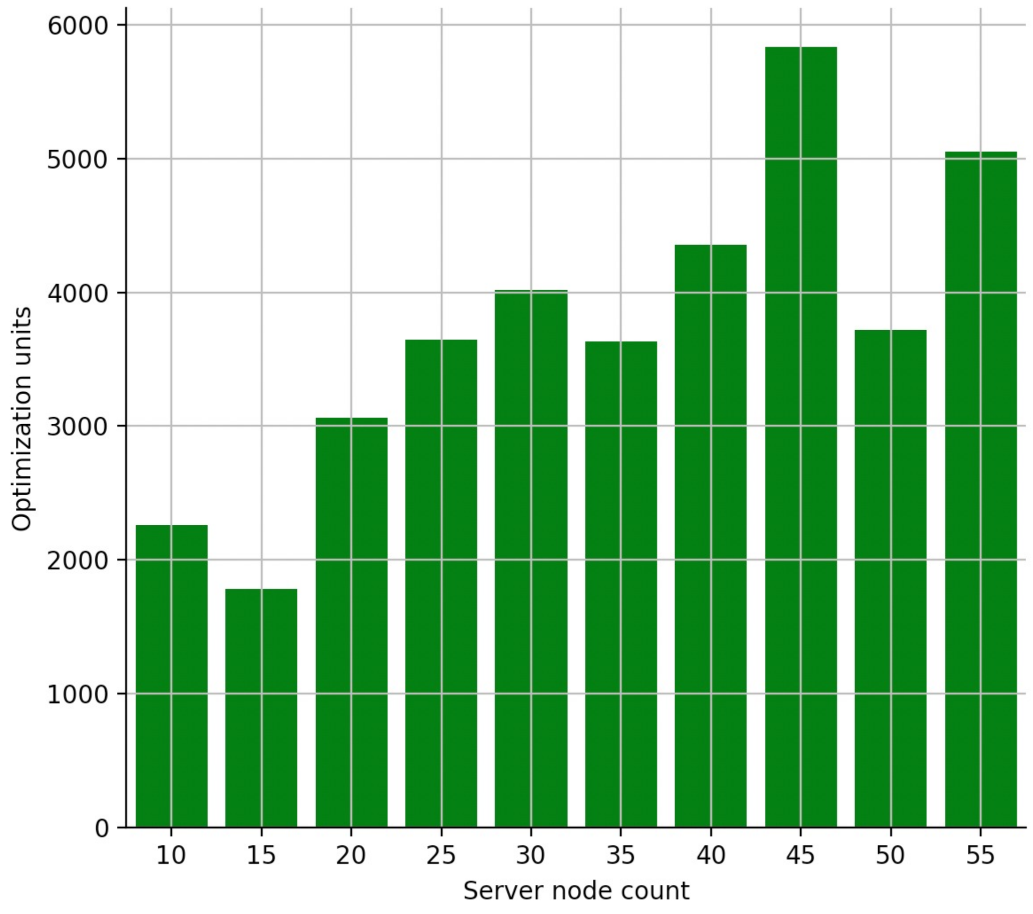

Figure 5.4: Optimization result difference between non-power optimized and power-optimized ILP implementation by server node count plot

ence in power consumption between the two algorithms follows a generally increasing pattern as the server count increases. These units are calculated by our objective function of the ILP model. These results show that our proposed improvements provide **a high value of savings in power consumption**.

Furthermore, as can be seen in Figure-5.3, while there is no considerable amount of optimization time difference between non-optimized and optimized ILP algorithms, Figure-5.4 shows a considerable value of power consumption difference between the two algorithms. Considering the greedy algorithm results from Figure-5.2, we can say that unless there is an optimization target, using the greedy approach rather than ILP could perform better. However, our ILP solution results in substantial power savings from the power optimization perspective.

5.2.2 DRL-based Model Results

In this part, we will illustrate the DRL-based optimization algorithm results. In addition, we will compare these results with the extended version of the ILP-based model explained earlier.

Figures 5.5 5.6 5.7 illustrate average power consumption over episodes. Average power consumption is calculated cumulatively so that for each episode, the cumulative power consumption of the last 100 episodes is divided by 100. Hence, power consumption in the last 100 episodes can be seen. As can be seen in the figures, the average power consumption over episodes is steadily decreasing as the agent trains itself. This proves that DRL-based methods allocate VNFs on servers in an power-efficient way.

The average reward function of the agents over time can be seen in Figure 5.8. As can be observed in the graph, all agents start to converge between 400 to 500 episodes. The rewards in this figure are the average rewards of each agent on both non-secure and secure versions of small, medium, and large network environments. Although there is no significant difference between agents reward wise, DDQN with Prioritised Replay performs better than other agents in reward convergence. On the other hand, the DQN agent with fixed Q Targets seems to converge towards smaller rewards than

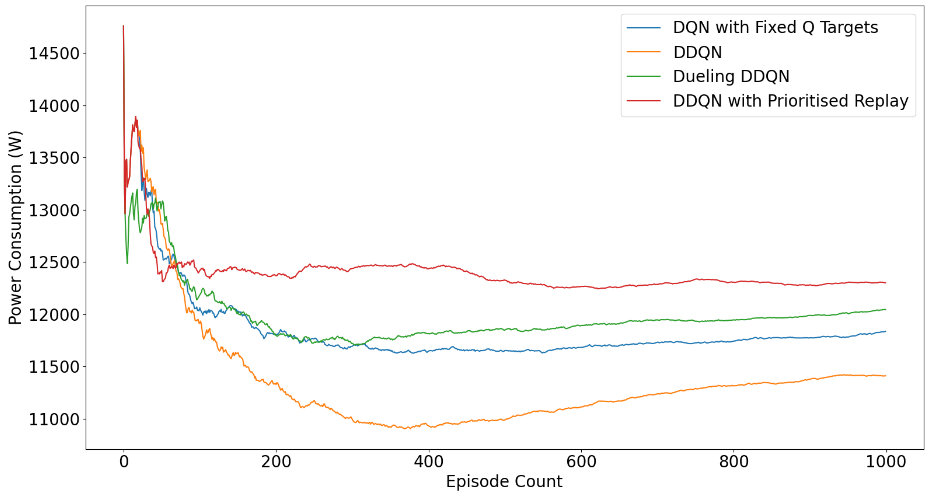

Figure 5.5: Average Power Consumption for 5 Servers

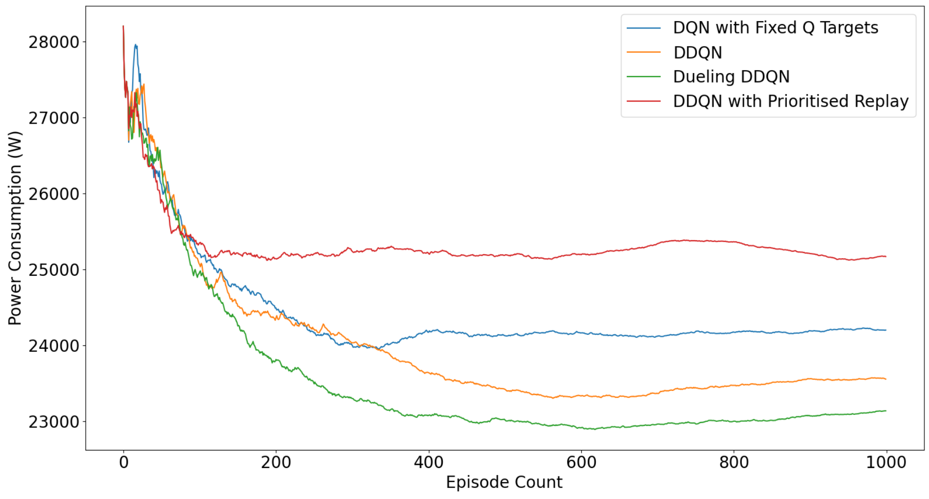

Figure 5.6: Average Power Consumption for 10 Servers

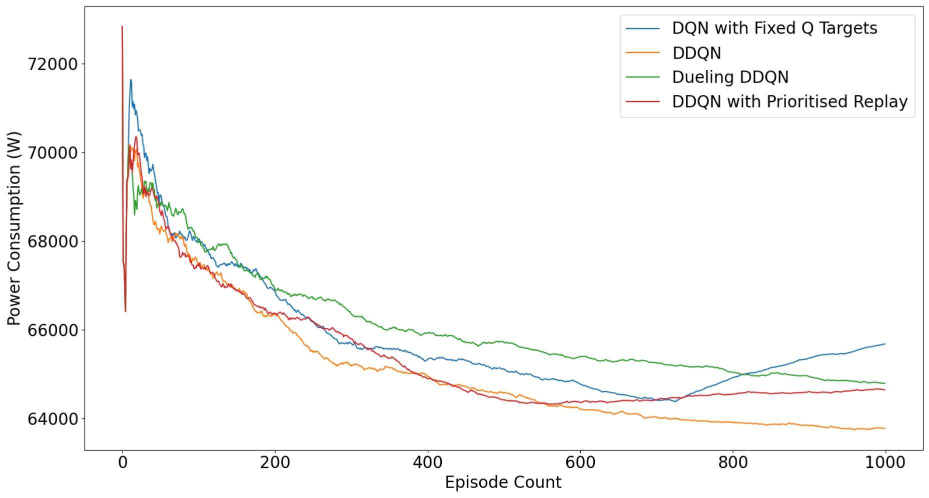

Figure 5.7: Average Power Consumption for 30 Servers

other agents.

Each episode starts with almost half of the VNFs being randomly open on the servers. We start the environment with an initial power consumption level equal to almost utilizing half of the capacity of the network for comparing how agents perform after 1000 episodes. At the end of the 1000 episodes, according to 5.1, we get the power efficiency values of each agent and have a mean for each environment set. In 5.1, E_e represents the power efficiency and δ is the initial power consumption value for the episodes. M_n represents the power consumption value in episode n. In this experiment, we took the average of the best 100 achieved results in the episodes. Figure 5.9 compares how agents performed on small, medium, and large networks and agents' performance comparison on secure and non-secure environment versions. The results in this figure show the average performance of all agents used in this work under secure and non-secure environments. Column bars represent the average power efficiency of the proposed methods in the scenarios. There are two main pieces of information we interpret from this figure. Firstly, DRL agents performed better on small networks due to the small amount of action space. As server count increases, action space increases, and their relationship is exponential. In order to solve this issue, we can utilize the method used in [45]. Since we are not focusing on a novel solution for the implementation of DRL agents, we leave this issue as future work.

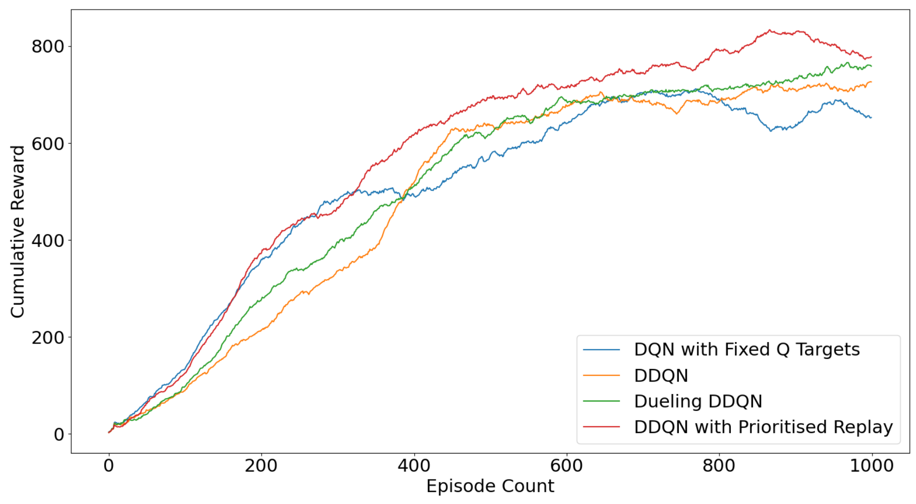

Figure 5.8: Average Reward Graph

Secondly, DRL agents perform better in security-constrained environments than in non-secure ones. We interpret this occasion as security constraints on servers giving DRL agents more information about the environment. Hence, agents learn better than in an environment where no constraint exists. According to our interpretation, when there are no constraints on the environment, the learning process of the agents cannot find a profound directive to decrease power consumption and develops more primitive strategies. On the other hand, when we set more rules for the environment, such as the security constraints we explained earlier, agents develop intelligent strategies related to the system's limitations, leading to better power efficiency.

$$E_e = 100 - (100 * \frac{1}{n}\sum_{i=1}^{n} M_n)/\delta \tag{5.1}$$

In Figure 5.10 the dark blue bar represents the secure environments', whereas the cyan bar represents the non-secure environments' power efficiency. For example, DDQN achieves around 40 percent efficiency on average for all secure environment sets, while non-secure versions achieve 25 percent power efficiency. From the findings, it is hard to say whether a particular agent performs better than others on average. However, we can say that the implemented DRL methods can achieve up to 40 percent power efficiency.

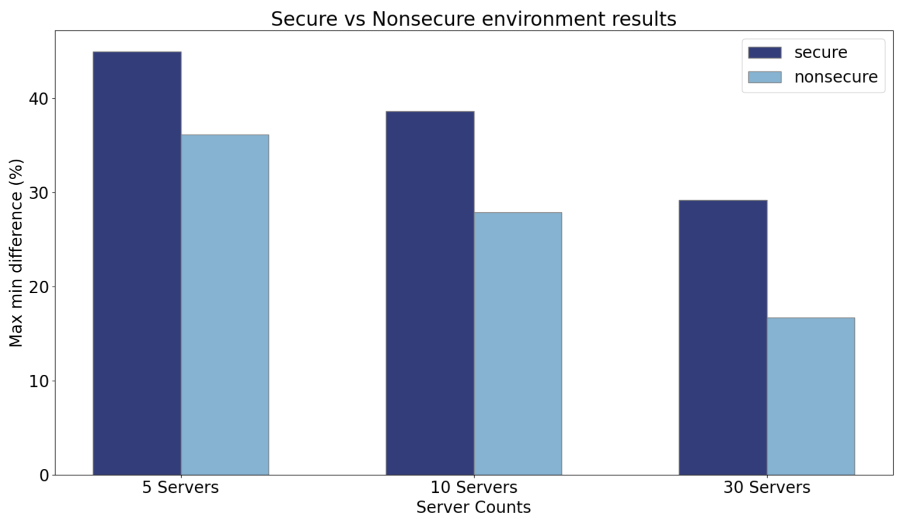

Figure 5.9: Power Efficiencies of DDQN Approach in Secure and Non-secure Environments Based on Server Count

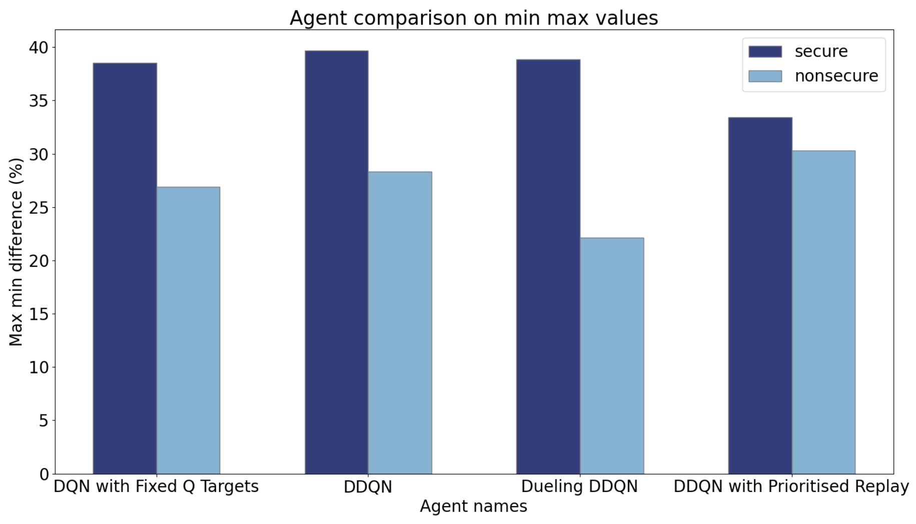

Figure 5.10: Power Efficiencies of DRL Agents for Secure vs Non-secure Environments

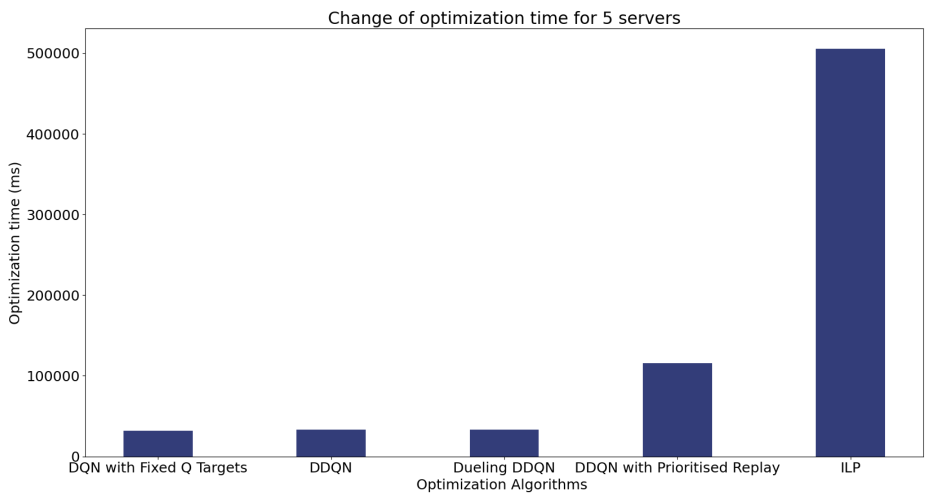

Figure 5.11: Average Optimization Time for 5 Servers

5.2.3 ILP vs DRL

The comparison of ILP vs. DRL-based optimization is made under two sections: time-based and power-based. Figures 5.11, 5.12, 5.13 show the optimization times for the different algorithms used in the prepared secure environments in ms. The experiments were conducted on a computer with Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz and GeForce GTX 1050 Ti Mobile GPU.

DRL-based algorithms' time costs are calculated according to the average episode time of each agent. In each train session, the agent runs 1000 episodes, and in each episode, randomly generated traffic is mapped into the simulated network environment at 1000 steps. As explained before, the agent runs until power consumption increases. Also, in each step, the agent tries to map randomly generated 100 different user equipments in the network simulator. The average episode time is then handled by dividing the total time cost of a train session by 1000.

According to our results, DQN with Fixed Q-targets, DDQN, and Dueling DDQN have almost the same optimization time in ms. However, DDQN with Prioritised Replay costs almost double the time in all small, medium, and large networks. The reason for this could be the extra steps taken when putting experience into the replay memory, such as the computing importance-sampling weight and updating transition

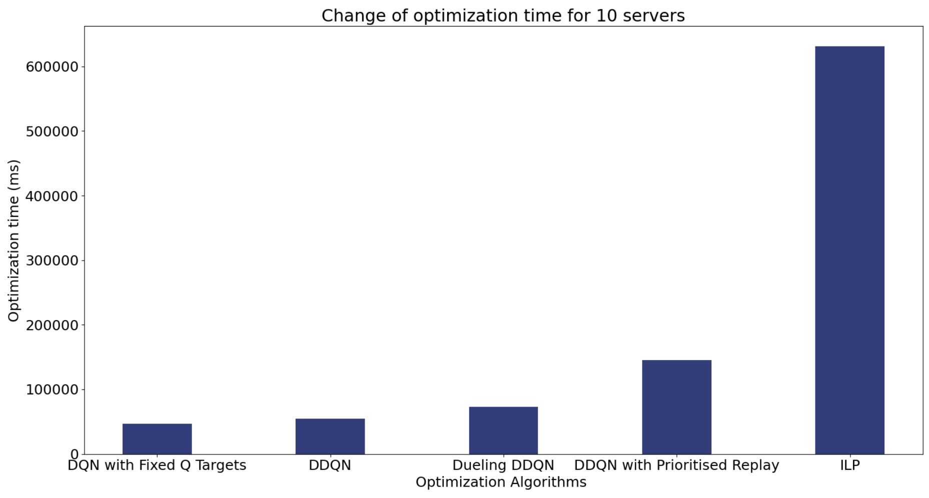

Figure 5.12: Average Optimization Time for 10 Servers

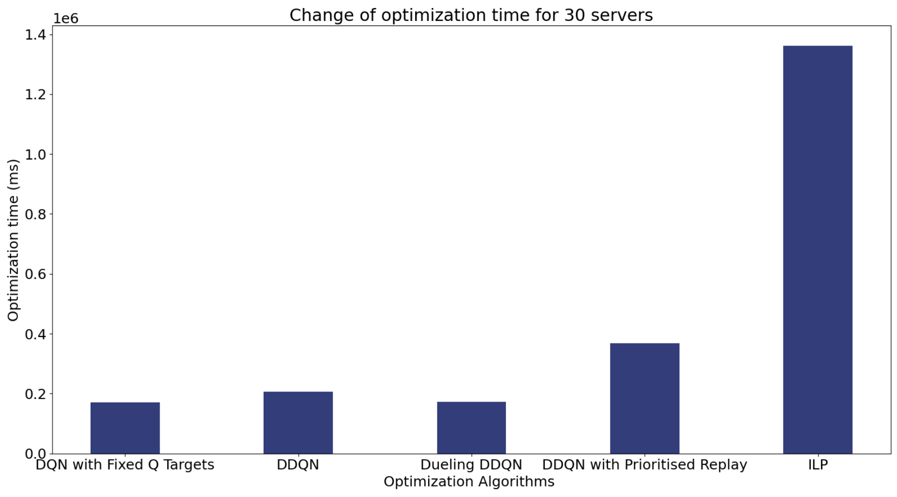

Figure 5.13: Average Optimization Time for 30 Servers

priorities.

The same approach is used to calculate the time cost of the ILP-based optimization algorithm. Randomly generated 100 user equipments are mapped in the network topology. Based on our experiments, ILP-based algorithms cost almost seven times more in terms of time than DRL-based algorithms. However, one must notice that the total time cost of each agent's training is 1000 more than what is seen in the figures. Even though the initial training period is more than that of the ILP-based model, after training is complete, a single run of mapping user equipment into the network is at least five to seven times more efficient than linear optimization. In addition to that, DRL-based agents can handle environment changes like different types of traffic coming into the network, while the ILP-based agent does not. While DRL-based models are working in a dynamic environment, they can retrain themselves in an online manner. On the other hand, ILP-based models need to solve the optimization problem for each changed environment parameter.

As seen in the Figures 5.11, 5.12, 5.13, the optimization time in larger networks increases significantly for the DRL models. For example, while the 5-server optimization time for DDQN is 33530 ms, the 30-server optimization time is 205726 ms, which is more than six times, and this relation is the same for the other DRL algorithms too. When the network sizes are smaller, the increase in the optimization time is sublinear. For instance, while the 5-server optimization time for DDQN is 33530 ms, the 30-server optimization is 54347 ms, which is less than two times, and this relation is the same for other optimization algorithms too.

As can be seen in Figures 5.14, 5.15, 5.16 DRL-based algorithms perform better in terms of power consumption optimization as well. Optimization results are calculated by subtracting the average power consumption result, which is calculated by 5.1 from the average initial power consumption of an environment.

Since the general training strategy is the same for all of the agents, there seems not to be significant power consumption optimization differences between the four DRL agent types. They all converge to similar values. However, comparing ILP-based algorithms vs. DRL-based optimization shows that optimization algorithms could converge on different power efficiency values. ILP-based optimization of power

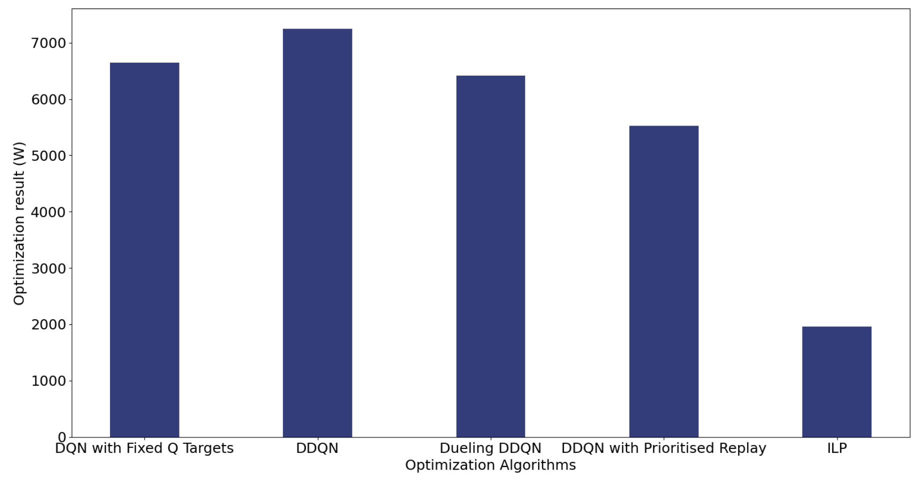

Figure 5.14: Average Power Consumption for 5 Servers

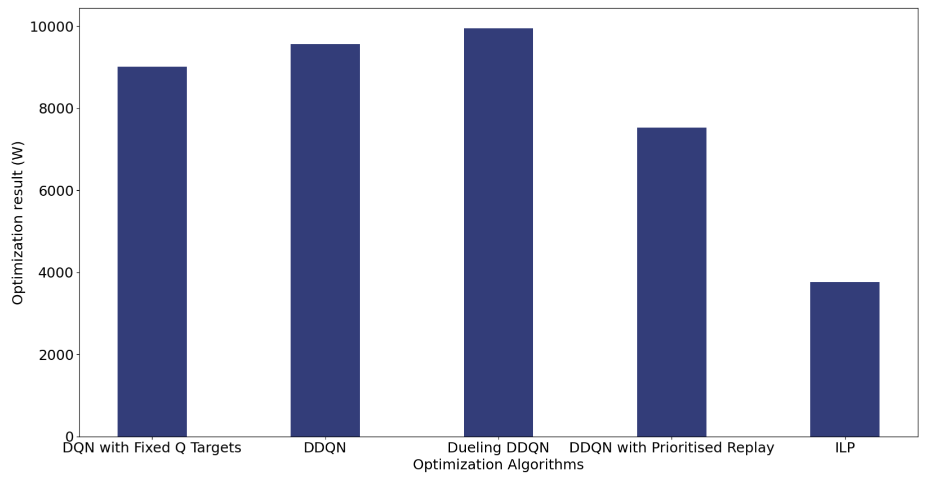

Figure 5.15: Average Power Consumption for 10 Servers

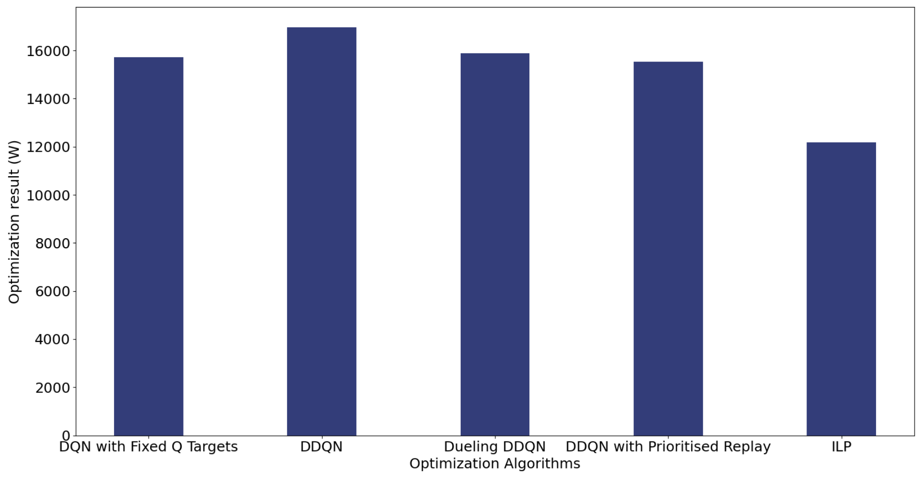

Figure 5.16: Average Power Consumption for 30 Servers

consumption optimization gives results such that DRL-based algorithms almost performed better in all five, ten, and thirty server count environments. However, optimization results for DRL and ILP get closer as server count increases; the reason for that could be that as server count increases, the ILP-based solution could map VNFs into the servers in a better way.

CHAPTER 6

CONCLUSION

In this chapter, we conclude this work and briefly discuss the possible future extensions.

6.1 Conclusion

This thesis proposes DRL-based models for allocating VNFs on a 5G core network in an energy-efficient and secure way. We benchmarked various DRL algorithms with environments requiring strict security and those that do not have any security requirements. We tested our algorithms on small, medium, and large network sets to compare their results. In addition to that, we compared our proposed solution with Integer Linear Programming based optimization algorithms. It can be concluded that DRL approaches are more efficient than integer linear programming-based optimization algorithms since they can work in dynamic environments efficiently and perform better in terms of energy efficiency in the same environment. Furthermore, we showed that the DRL-based algorithms could be used for resource allocation where energy efficiency and security requirements are strict.

6.2 Future Work

Larger action spaces for DRL methods could be implemented in future work to assess their performance on massive 5G telecommunication networks. Dealing with large action spaces is still an open problem in Deep Reinforcement Learning methods. In addition to that, the algorithms we developed could be implemented on an actual

5G network structure to assess their performance on a real-life system. The current approach only works on user-generated traffic and environment models. Lastly, the methods could be extended to handle dynamic slice type additions to the network.

REFERENCES

[1] "Mobile Subscriptions Forecast, Ericsson Mobility Report Q2 2022 Update." https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/mobile-subscriptions-outlook. Accessed: 2022-08-09.

[2] C. Dongxu, "5G power: Creating a green grid that slashes costs, emissions & energy use." https://www.huawei.com/en/technology-insights/publications/huawei-tech/89/5g-power-green-grid--costs-emissions-energy-use, Jul 2020. Accessed: 2022-06-11.

[3] H. Kim, "5G core network security issues and attack classification from network protocol perspective," *Journal of Internet Services and Information Security (JISIS)*, vol. 10, pp. 1–15, May 2020.

[4] S. Nowaczewski and W. Mazurczyk, "Securing future internet and 5G using customer edge switching using DNSCrypt and DNSSEC," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications JoWUA*, vol. 11, pp. 87–106, September 2020.

[5] X. Li, M. Samaka, H. A. Chan, D. Bhamare, L. Gupta, C. Guo, and R. Jain, "Network slicing for 5G: Challenges and opportunities," *IEEE Internet Computing*, vol. 21, pp. 20–27, September 2017.

[6] S. Buzzi, C. I, T. E. Klein, H. V. Poor, C. Yang, and A. Zappone, "A survey of energy-efficient techniques for 5G networks and challenges ahead," *IEEE Journal on Selected Areas in Communications*, vol. 34, pp. 697–709, April 2016.

[7] A. M. Khedr, P. R. P. V, and A. A. Ali, "An energy-efficient data acquisition technique for hierarchical cluster-based wireless sensor networks," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications JoWUA*, vol. 11, pp. 70–86, September 2020.

[8] A. Fendt, S. Lohmuller, L. C. Schmelz, and B. Bauer, "A network slice resource allocation and optimization model for end-to-end mobile networks," in *2018 IEEE 5G World Forum (5GWF)*, (Silicon Valley, CA, USA), pp. 262–267, IEEE, July 2018.

[9] O. Akin, U. C. Gulmez, O. Sazak, O. U. Yagmur, and P. Angin, "GreenSlice: An energy-efficient secure network slicing framework," *Journal of Internet Services and Information Security (JISIS)*, vol. 12, pp. 57–71, February 2022.

[10] S. Rommer, P. Hedman, M. Olsson, L. Frid, S. Sultana, and C. Mulligan, "Chapter 3 - architecture overview," in *5G Core Networks* (S. Rommer, P. Hedman, M. Olsson, L. Frid, S. Sultana, and C. Mulligan, eds.), pp. 15–72, Academic Press, 2020.

[11] M. Morbitzer, B. Kopf, and P. Zieris, "Guarantee: Introducing control-flow attestation for trusted execution environments," 2022. `https://doi.org/10.48550/arxiv.2202.07380` Accessed: 2022-08-10.

[12] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, (USA), p. 857–874, USENIX Association, 2016.

[13] D. Evtyushkin, J. Elwell, M. Ozsoy, D. Ponomarev, N. A. Ghazaleh, and R. Riley, "Iso-x: A flexible architecture for hardware-managed isolated execution," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, (Cambridge,UK), pp. 190–202, IEEE, 2014.

[14] D. Lee, D. Kohlbrenner, S. Shinde, D. Song, and K. Asanović, "Keystone: An open framework for architecting tees," 2019. `https://arxiv.org/abs/1907.10119` Accessed: 2022-02-05.

[15] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 86, 2016.

[16] J. Ménétrey, C. Göttel, M. Pasin, P. Felber, and V. Schiavoni, "An exploratory study of attestation mechanisms for trusted execution environments," 2022. `https://arxiv.org/abs/2204.06790` Accessed: 2022-07-20.

[17] 3GPP, "Technical Specification Group Services and System Aspects; System architecture for the 5G System (5GS); Stage 2," Technical Specification (TS) 23.501, 3rd Generation Partnership Project (3GPP), March 2020. Version 16.4.0.

[18] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges," *Computer Networks*, vol. 167, p. 106984, February 2020.

[19] ETSI, " European Telecommunications Standards Institute," GS NFV 002, European Telecommunications Standards Institute, 10 2013. Version 1.1.1.

[20] 3GPP, "Technical Specification Group Services and System Aspects; Telecommunication management; Study on management and orchestration of network slicing for next generation network," Technical Report(TR) 28.801, 3rd Generation Partnership Project (3GPP), 01 2018. Version 15.1.0.

[21] G. M. Yilma, Z. F. Yousaf, V. Sciancalepore, and X. Costa-Perez, "Benchmarking open source NFV MANO systems: OSM and ONAP," *Computer Communications*, vol. 161, pp. 86–98, 2020.

[22] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[23] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, (Singapore), pp. 3389–3396, IEEE, 2017.

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

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing Atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[26] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, Mar. 2016.

[27] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

[28] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1995–2003, PMLR, 20–22 Jun 2016.

[29] H. Jmila and G. Blanc, "Towards security-aware 5G slice embedding," *Computers & Security*, vol. 100, p. 102075, January 2021.

[30] R. Doriguzzi-Corin, S. Scott-Hayward, D. Siracusa, and E. Salvadori, "Application-centric provisioning of virtual security network functions," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, (Berlin,Germany), pp. 276–279, IEEE, December 2017.

[31] J. Guan, Z. Wei, and I. You, "Grbc-based network security functions placement scheme in SDS for 5G security," *Journal of Network and Computer Applications*, vol. 114, pp. 48–56, July 2018.

[32] L. D. Nguyen, "Resource allocation for energy efficiency in 5G wireless networks," *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, vol. 5, p. 154832, June 2018.

[33] B. Matthiesen, O. Aydin, and E. A. Jorswieck, "Throughput and energy-efficient network slicing," in *WSA 2018; 22nd International ITG Workshop on Smart Antennas*, (Bochum, Germany), pp. 1–6, VDE, June 2018.

[34] Q. Shi, L. Zhao, Y. Zhang, G. Zheng, F. R. Yu, and H. Chen, "Energy-efficiency versus delay tradeoff in wireless networks virtualization," *IEEE Transactions on Vehicular Technology*, vol. 67, pp. 837–841, January 2018.

[35] Z. Liu, X. Chen, Y. Chen, and Z. Li, "Deep reinforcement learning based dynamic resource allocation in 5G ultra-dense networks," in *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, (Tianjin, China), pp. 168–174, IEEE, August 2019.

[36] F. E. Salem, Z. Altman, A. Gati, T. Chahed, and E. Altman, "Reinforcement learning approach for advanced sleep modes management in 5G networks," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, (Chicago, IL, USA), pp. 1–5, IEEE, August 2018.

[37] M. Laroui, M. A. Cherif, H. I. Khedher, H. Moungla, and H. Afifi, "Scalable and cost efficient resource allocation algorithms using deep reinforcement learning," in *2020 International Wireless Communications and Mobile Computing (IWCMC)*, (Limassol, Cyprus), pp. 946–951, IEEE, June 2020.

[38] R. Li, Z. Zhao, Q. Sun, I. Chih-Lin, C. Yang, X. Chen, M. Zhao, and H. Zhang, "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74429–74441, November 2018.

[39] C. Qi, Y. Hua, R. Li, Z. Zhao, and H. Zhang, "Deep reinforcement learning with discrete normalized advantage functions for resource management in network slicing," *IEEE Communications Letters*, vol. 23, pp. 1337–1341, June 2019.

[40] L. Zhao and L. Li, "Reinforcement learning for resource mapping in 5G network slicing," in *2020 5th International Conference on Computer and Communication Systems (ICCCS)*, (Shanghai, China), pp. 869–873, IEEE, June 2020.

[41] ITU, "Measurement method for energy efficiency of network functions virtualization ," Recommendation L.1361, International Telecommunication Union (ITU), 11 2018. ITU-T L.1361.

[42] Huangxi, "Bandwidth needs in core and aggregation nodes in the optical transport network." `https://www.ieee802.org/3/ad_hoc/bwa/public/nov11/huang_01_1111.pdf`, 2011. Accessed: 2022-01-09.

[43] J. Manner, "How to measure real latency, as experienced by your customers?." `https://www.netradar.com/`

`how-to-measure-real-latency-as-experienced-by-your-customers/`, May 2021. Accessed: 2022-01-17.

[44] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[45] G. Dulac-Arnold, R. Evans, P. Sunehag, and B. Coppin, "Reinforcement learning in large discrete action spaces," *CoRR*, vol. abs/1512.07679, 2015.