T.C.

ALTINBAŞ UNIVERSITY

Electrical and Computer Engineering

OPTIMIZING THE LIFETIME OF WIRELESS SENSOR NETWORKS USING DEEP REINFORCEMENT LEARNING IN A SOFTWARE-DEFINED NETWORK ARCHITECTURE

Zainab Ali Abbood

Master Thesis

Supervisor

Cagatay AYDIN

Istanbul, 2019

OPTIMIZING THE LIFETIME OF WIRELESS SENSOR NETWORKS USING DEEP REINFORCEMENT LEARNING IN A SOFTWARE-DEFINED NETWORK ARCHITECTURE

Zainab Ali Abbood

Electrical and Computer Engineering

Submitted to the Graduate School of Science and Engineering

in partial fulfillment of the requirements for the degree of

Master of Science and Engineering

ALTINBAŞ UNIVERSITY

2019

This is to certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science

Dr. Mahmoud Shuker Mahmoud	Asst. Prof. Dr. Cagatay AYDIN
Co-Supervisor	Supervisor

Examining Committee Members (first name belongs to the chairperson of the jury and the second name belongs to supervisor)

Asst. Prof. Dr. Cagatay AYDIN	School of Engineering and Natural Sciences, Altinbas Univerisity	_____
Asst. Prof. Dr. Dogu Cagdas ATILLA	School of Engineering and Natural Sciences, Altinbas Univerisity	_____
Asst. Prof. Dr. Cahit KARAKUS	Faculty of Engineering and Architecture, Esenyurt Univerisity	_____

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____
Asst. Prof. Dr. Cagatay AYDIN
Head of Department

_____
Prof. Dr. Oguz BAYAT
Director

Approval Date of Graduate School of
Science and Engineering: ____/____/____

DECLARATION

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Zainab Ali Abbood

.

ACKNOWLEDGMENTS

My utmost thanks and gratitude must first be offered to Almighty Allah for all his blessing, and in granting me patience throughout the duration of this research.

Profound appreciation and thanks are given to my supervisor, Asst. Prof. Cagatay Aydin and Co-Supervisor Dr. Mahmoud Shuker Mahmoud, for their patient supervision, guidance, constructive suggestions and comments during the entire research period until its completion. Their advice and support throughout the program have been invaluable. Without their tireless help, leadership, and confidence in my ability, the completion of this project would not have been possible. I also offer my gratitude to them for opening my mind to a new world of knowledge, opportunities, and experience, giving me a better understanding throughout.

I would like to thank my brother, Dr. Hayder Ali Abbood, for always being beside me and helping me. Love you, bro!

Finally, I must thank all my family for all their help and continuous support at every stage of my research.

ABSTRACT

OPTIMIZING THE LIFETIME OF WIRELESS SENSOR NETWORKS USING DEEP REINFORCEMENT LEARNING IN A SOFTWARE-DEFINED NETWORK ARCHITECTURE

Zainab Ali Abbood,

M.Sc., Electrical and Computer Engineering, Altınbaş University

Supervisor: Asst. Prof. Dr. Cagatay Aydin

Co-Supervisor: Dr. Mahmoud Shuker Mahmoud

Date: September/2019

Pages: 63

According to the changing topologies of modern networks, the use of static routing rules has become obsolete. Software-Defined Networks (SDNs) are being used to overcome such limitation, where a central controller handles the decision-making role regarding packets routing. This controller collects information about the network, in addition to the packet information, to decide the route a packet should follow to reach its destination. However, with the growing complexity of WSNs topologies and the importance of efficient routing, Machine Learning (ML) techniques are being used to handle the decision making in the SDN controller. In this study, a new method is proposed to optimize the resources consumption in a WSN that uses SDN. The proposed method employs a neural network that is trained using Reinforcement Learning (RL), based on the lifetime of the WSN. To extend the lifetime of WSN the neural network is required to optimize the power consumption of the nodes in that network, in which the optimal routes must be used. Three types of neural networks are evaluated in this thesis; Feed-Forward Neural Network (FF-NN), 2D-Convolutional Neural Network (2D-CNN) and 3D-CNN. The evaluation of these models show that the using the 3D-CNN has achieved the best performance, with an average lifetime of 678251.6 seconds, with an extension of 17% of the 578122.2 seconds using the existing state-of-the-art method. The average number of hops a packet is required

to travel through, to reach its destination, in this model is 9.81 hops with an average Packet Delivery Rate (PDR) of 85.07%. Additionally, the 2D-CNN model has achieved 638169.2 seconds lifetime, with an average of 12.37 hops per packet and 82.47% PDR, whereas the FF-NN has achieved 578381.6 seconds lifetime with 83.37% PDR and 8.31 hops per packet. In addition to the superiority of the 3D-CNN, the results also show that the use of the shortest paths causes an exhaustion to the resources of certain nodes, positioned in locations that handle extensive traffic, which reduces the overall lifetime of the WSN. Thus, the extension of the lifetime requires using alternative, i.e. longer, paths to avoid such exhaustion and extend the lifetime of the network.

Keywords: Software-Defined Network; Internet of Things; Wireless Sensor Network; Reinforcement Learning; Artificial Neural Network.

ÖZET

YAZILIM TANIMLI BİR AĞ MİMARİSİNDE DERİN TAKVİYELİ ÖĞRENMEYİ KULLANARAK KABLOSUZ SENSÖR AĞLARININ ÖMÜR BOYU OPTİMİZASYONUNU SAĞLAMA

Zainab Ali Abbood,
Yüksek Lisans Elektrik ve Bilgisayar Mühendisliği, Altınbaş Üniversitesi
Danışman : Asst. Prof. Dr. Çağatay Aydın
Eş Danışman: Dr. Mahmoud Shuker Mahmoud
Tarih: Eylül/2019
Sayfa Sayısı: 63

Farklı uygulamalardaki verilerin günümüzde gittikçe artan önemi ile birlikte, çevreden söz konusu verileri toplayabilmek adına Kablosuz Sensör Ağları (Wireless Sensor Networks/WSN) yaygın bir biçimde kullanılmaktadır. Bu ağların görevlerini yerine getirebilmek için ihtiyaç duymuş oldukları esnekliği sağlamak adına, küçük boyutlarda ve bu sayede yüksek hareketliliğe sahip sensörler kullanılır. Ancak, bahsedilen sensörlerin küçük boyutlu olması, güç ve enerji kaynaklarının işlenmesi gibi, devrelerin donatılabilecek oldukları kaynak sayısını sınırlandırmaktadır. Ayrıca, internet erişiminin kolaylaşması ile birlikte, söz konusu ağlar, Nesnelerin İnternetini (Internet of Things) oluşturan bir merkez birimde toplamış oldukları verileri gönderebilmek için internete bağlanmaktadır. Kablosuz Sensör Ağlarının (WSN) değişmekte olan devre topolojilerine adapte olabilmesi adına, devreler, bir sunucudan diğer sunucuya, kendilerinden gönderilen paketlere ek paket göndermeye yönlendirilirler. Dolayısıyla, kendi görevini yerine getirebilmek ve paketlerini iletebilmek adına ihtiyaç duyduğu kaynaklara ek olarak, devreden ihtiyaç duyulan kaynaklar, iletilmesi gereken paketlerin sayısına bağlıdır. Böylelikle, devrelerdeki kaynak tüketiminin optimizasyonu, paketlerin hedeflerine ulaşabilmek için kullandıkları yolların optimizasyonunu gerektirmektedir.

Modern ağların değişmekte olan topolojisine göre, sabit ağ veri yönlendirme kuralları artık kullanılmamaktadır. Söz konusu kısıtlandırmanın aşılması adına, merkezi bir denetçinin paket verilerinin yönlendirilmesinde karar veren role sahip olduğu Yazılım Tanımlı Ağlar (SDN) kullanılır. Söz konusu denetleyici, bir paketin hedefine ulaşmak için takip etmesi gereken yolu belirlemek adına, paketin bilgilerine ek olarak, ağ hakkında da veri toplar. Fakat, WSN'nin topolojisinde gelişen karmaşıklık ve etkili yönlendirmenin önemi ile birlikte, SDN'nin denetçisinde karar verme sürecini kontrol etmek adına ML (Makine Öğrenimi) teknikleri kullanılmaktadır. Bu çalışmada, SDN kullanmakta olan WSN içerisindeki kaynak tüketiminin optimizasyonunu sağlamak adına yeni bir yöntem sunulmaktadır. Sunulan yöntem, WSN'nin ömrü bazında, RL (Takviyeli Öğrenme) kullanımı ile eğitilen bir nöral ağdan yararlanmaktadır. WSN'nin ömrünü uzatabilmek için, nöral ağın en ideal rotaların kullanılması gereken ağlardaki devrelerin güç tüketimini optimize etmesi gerekmektedir. Bu tez çalışmasında üç tür nöral ağ değerlendirilmketedir; İleri Beslemeli Nöral Ağ (Feed-Forward Neural Network – FF-NN), 2 Boyutlu Evrişimsel Nöral Ağ (2D- Convolutional Neural Network – 2D-CNN) ve 3D-CNN. Bu modellerin değerlendirmesi; gelişen en son teknoloji ile var olan 578122.2 Saniye ömür süresini %17 değerinde uzatarak, 678251.6 Saniye ortalama ömür süresi ile 3D-CNN'nin en iyi performansı elde ettiğini ortaya koymaktadır. Bu model dahilinde, bir paketin hedefine ulaşabilmesi için gerekli ortalama sıçrama değeri, ortalama %85.07'lik Paket Aktarım Oranı (Packet Delivery Rate - PDR) ile 9.81 sıçramadır. Ayrıca, 2D CNN modeli, paket başına ortalama 12.37 sıçrama ve %82.47'lik PDR ile 638169.2 Saniye ömür süresi elde etmiş, FF-NN modeli ise paket başına 8.31 sıçrama ve %83.37 PDR ile 578381.6 Saniye ömür süresine ulaşmıştır. Sonuçlar, 3D-CNN modelinin üstünlüğüne ek olarak, en kısa yolların kullanımının, yoğun trafiği kontrol altında tutan bölgelerde konumlandırılmış olan bazı devrelerin kaynaklarında tükenme meydana getirdiğini ve WSN'nin toplam ömrünü kısalttığını da göstermiştir. Yani, ömrün uzatılması, söz konusu tükenmeden kaçınabilmek adına alternatif, yani daha uzun, rotaları tercih etmeyi gerektirmektedir.

Anahtar Kelimeler*:* Yazılım Tanımlı Ağ; Nesnelerin İnterneti; Kablosuz Sensör Ağları; Takviyeli Öğrenme; Yapay Nöral Ağ.

TABLE OF CONTENTS

Page

LIST OF TABLES

Pages

LIST OF FIGURES

Pages

LIST OF ABBREVIATIONS

WSN : Wireless Sensor Network

IoT : internet of Things

SDN : Software-Defined Network

ML : Machine Learning

RL : Reinforcement Learning

ANN : Artificial Neural Network

CNN : Convolutional Neural Network

VANET : Vehicular ad hoc Network

GPS : Global Positioning System

PPG : Photo-Plethysmography

RFID : Radio-Frequency Identification

MSDS : Material Safety Data Sheets

API : Application Programming Interface

REST : Representational State Transfer

HTTP : Hyper-Text Transfer Protocol

FF-NN : Feed-Forward Neural Network

DQN : Deep Q-Networking

PDR : Packet Delivery Rate

1. INTRODUCTION

With the rapidly growing interest in collecting data of different types for different applications, smaller devices have been used to collect these data and execute remote commands. The smaller size is a mandatory feature of these devices, i.e. sensors, to provide them with the mobility required to satisfy the applications these devices are employed for [1]. Moreover, the use of wireless communications has also been able to improve the mobility and flexibility of the networks that these devices create and use to communicate. Such networks are known as Wireless Sensor Networks (WSNs) and use a special type of communication to establish connections among the sensors and the external world, so that, the collected information is delivered to their destinations [2-4].

The high availability of internet connections in different types has eased access to that network, which can be used to establish communications among devices in remote locations. This availability has brought attention to using the internet's infrastructure to connect several types of devices to the internet, in order to transfer the data that they collect and execute the commands sent to them. These devices may vary from as simple as a coffee maker to autonomous cars, which has presented the Internet of Things (IoT) [5, 6]. However, as the infrastructure of the internet, such as the communication protocols, are designed to be used by computers with larger resources and less mobility, compared to IoT devices, the IoT has imposed new challenges toward optimizing the operation of these devices [7, 8].

One of the main concerns about the operation of IoT devices is the amount of energy the devices consume to perform the tasks required from them. To maintain the mobility of these devices, they are usually energized with power sources of limited energy [9, 10]. Moreover, in larger WSNs, the devices act like nodes to deliver the traffic from one host to another, which can be another node in the network or the sink node that collects the data and deliver it to the internet, i.e. the gateway of the network. Hence, the power consumed by a certain node in the network does not only depend on the traffic initiated from, or directed to, that node but it also depends on the traffic that the node is required to deliver to other nodes to maintain the communications in the network [11, 12].

With the growing complexity of network topologies, routing the packets through the network is also requiring more complex techniques, so that, the overall performance of the network can be improved by improving the routing decisions. Thus, Software-Defined Networking has emerged as a solution to such topologies, to handle the dynamic nature of the network where static rules cannot be used to route the traffic [13]. In a Software-Defined Network (SDN), a controller is designated to make the appropriate decisions for each packet, depending on the state of the network at the time the packet is being transmitted. Hence, these controllers require more information about the network than only the addresses of the source and destination hosts. This information is used by the controller to choose the appropriate route for the packet to reach its destination [14].

Machine Learning (ML) techniques have been widely used in SDNs, according to the ability of ML techniques to dynamically interact with the inputs, unlike the use of static rules. These techniques extract knowledge about the domain they are required to interact with using sets of examples collected from that domain [15]. Depending on the approach used to collect these data, ML techniques are categorized into three main categories, unsupervised, supervised and reinforcement. In unsupervised learning, the data is provided to the ML technique as collected from the domain, without any additional information, where the relations among the inputs are detected and used to interact with the environment. In supervised learning, additional information is added to the collected data, so that, the ML technique investigates the relations between the characteristics of each input and the information added to it. This additional information normally represents an expert's knowledge, which is aimed to be learned by the ML technique [16].

Reinforcement Learning (RL) techniques do not require any examples from the domain they are interacting with, as such knowledge is extracted by directly interacting with the domain. This interaction is defined as an agent that executes actions in the environment, by collecting the state of the environment. The main aim of RL is to create a model that approximates the behavior of the environment, so that, the best actions can be selected by the agent to execute in the environment, depending on the state of the environment. This model is created based on the execution of random actions in different states, so that, the outcome of each action at a certain state can be predicted. These predictions can be used to select the optimal action at each state the agent may be in the environment [17, 18].

One of the widely used approached to approximate complex functions is Artificial Neural Networks (ANN). Such networks mathematically replicate the signals communicated among the biological neurons in human brains to reach an appropriate decision, based on the inputs collected from different senses. Thus, these networks are trained to predict the outcome, i.e. reward, of executing a certain action at a certain state by providing the states of the environment and the actual reward collected from the environment when a certain action is executed. During operation, the state of the environment is fed to the neural network, which predicts the reward expected for each possible action. Then, the action with the highest reward is executed by the agent, to optimize the operation of the environment by executing the best possible actions [19].

The neurons in a neural network are distributed in layers, where the input of a certain layer is collected from the outputs of the neurons in the previous layer. Depending on the way these inputs are collected, there exist several types of neural networks. Each of them has shown better performance when interacting with a certain type of inputs [20]. Convolutional Neural Networks (CNNs) use multi-dimensional filters that are convoluted throughout the input in order for the neuron to calculate its output. This type of neural networks has shown significantly better performance in detecting local, multi-dimensional, features in inputs, which is the reason that these networks have shown good performance when interacting with images [21, 22]. Moreover, this type of neural networks has also shown good performance in RL applications employed for the controllers of SDNs, according to their ability of detecting relations among adjacent nodes, so that, the nodes closer to the one forwarding the packet can be detected, regardless of the topology on the network [23, 24].

1.1. PROBLEM STATEMENT

According to the dynamic nature of IoT networks, several methods are proposed to manage routing the packets in the networks using RL [23-25]. However, these techniques do not consider the power consumption in the network, which can exhaust certain nodes depending on their positioning in the network. Such exhaustion can dramatically reduce the lifetime of the network, as the loss of these nodes disturbs the communications among the

nodes of the network, hence, delivering the data to the required host. The method proposed by Lin et al. [23] calculates the reward from the environment, which is the network, based on the packets delay and loss in addition to the throughput of the network. According to these measures, the highest reward predicted by the neural network is expected to represent the shortest path, regardless of balancing the loading on the nodes in the network. Moreover, Stampa et al. [25] also aim to shorten the path each packet travels through, to reduce the time required to deliver the packet, whereas Zhang et al. [24] propose a method to maximize the probability of delivering the packets in Vehicular ad hoc Networks (VANETs).

1.2. AIM OF THE THESIS

This study presents a novel method to manage routing in SDNs based on RL. The proposed method considers the lifetime of the network, so that, the SDN controller selects alternative, possibly longer, routes for the packets to travel through. Despite the possibly longer paths, the proposed method aims to balance loading among the nodes of the network, so that, exhaustion of certain nodes is avoided. The proposed method uses different types of ANNs to approximate the function that represents the operation of the network. Then, the state of the network is collected and provided to the neural network, alongside with the information of the packet being routed. The node that maximizes the reward is then selected as the next hop in the route, until the packet is delivered. The lifetime of the network is included in the computations of the reward, so that, a balanced performance can be achieved, between the speed of the communications and the lifetime of the network. Hence, the proposed method can justify the gap in the existing methods [23-25], which make use of RL but do not consider balancing the nodes' loading.

1.3. THESIS LAYOUT

The remainder of this thesis is organized as follows:

- Chapter Two reviews the literature related to RL and the methods proposed to employ it in SDNs controllers.
- Chapter Three describes the proposed method and the different types of neural networks that can be used to achieve the required task.
- Chapter Four presents the experiments conducted to evaluate the performance of the proposed method using the selected types of neural networks.
- Chapter Five discusses the results collected from the experiments and compares them to the state-of-the-art method in the literature for the same purpose.
- Chapter Six illustrates the conclusions of this thesis and the future work that is going to be conducted to improve the performance of the proposed method.

2. LITERATURE REVIEW

The high availability of internet connection has encouraged the use of these connections to communicate information among different devices, other than computers and smart devices usually used by internet users. These devices are used in different applications to provide different kinds of services to the users, where in most cases, the information being exchanged are automatically collected by these devices and require to user's interaction. This phenomenon has created the Internet of Things (IoT) and it has become mandatory to adopt these devices and handle their communications [26]. The use of IoT devices has grown rapidly in recent years, according to the features they provide, such as mobility and accuracy. Thus, the IoT devices have been widely used in different fields of applications, such as healthcare, manufacturing, electricity, security and vehicles. The use of IoT devices in different applications is illustrated in Figure 2.1, based on the percentage of devices used in each field [5].

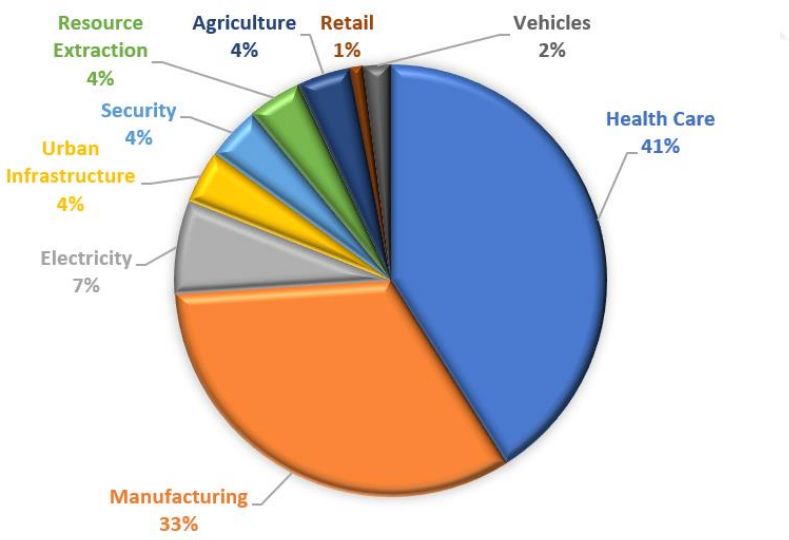

Figure 2.1: The use of IoT devices in different fields of applications [5].

2.1. EMPLOYMENT OF IOT DEVICES

A healthcare system based on IoT is proposed by Arijit et al. [27] that collects several biometric and environment measures to detect any anomaly in these measures in order to alarm the patient to seek for a medical care. The anomaly detection is based on data mining techniques, where measured variables are sent to a remote server to detect these anomalies. The patient's location, activity, movement and heart activity are monitored using the Global Positioning System (GPS), Accelerometer, Magnetometer, and Photo-Plethysmography (PPG) sensors, respectively. Moreover, an emergency health transmission system is proposed by Govindhan et al. [28] the relies on the IoT to monitor the parameters of a patient's body in order to assist providing better health care for that patient in case of emergency, where the recent vital measures can be analyzed to predict the required care in need. The existence of such systems imposes the need for rigid long-life monitoring using IoT devices, so that, reliable services are provided.

Although the IoT is not limited to healthcare, this field has the highest share among all other fields as shown in Figure 2.1. However, there are different other services that rely on IoT devices to improve their performance. A system is proposed by Kim et al. [29] manages the security of a chemical laboratory using IoT sensors. This system uses flame, gas and Radio-frequency identification (RFID) sensors to evaluate the environment in the laboratory using Material Safety Data Sheets (MSDS) to detect any hazards, so that, the administrator of the laboratory is immediately informed. The system uses an existing Application Programming Interface (API) to achieve communications among the different parts of the system, using Representational State Transfer (REST) architecture based on Hyper-Text Transfer Protocol (HTTP).

Moreover, the use of RL with the IoT has presented huge breakthroughs in modern applications, such as autonomous, i.e. self-driving, vehicles. Such application is gaining significant attention according to its benefits in reducing the risks of accidents and the comfort it provides to the drivers [30, 31]. These applications rely mainly on RL techniques, according to the enormous number of states and actions the autonomous driver is required to handle [32, 33]. In vehicles with larger power sources, such as cars, more

resources can be available for the computing device that is responsible for predicting the optimal action based on the state of the vehicle.

These applications illustrate the importance of the mobility of IoT devices, maintaining communications among them and the lifetime of the network. Exhausting the resources of a certain node in the network, according to the heavy traffic being forwarded through that node, can affect the performance of that node in its designated task. Thus, vital information that is being collected by the IoT device can be lost, especially when used in healthcare applications [27, 28]. Moreover, any disturbance in the communications that is caused by the loss of that node can cause the loss of an entire subregion of the network, especially in less-dense networks. Such loss can prevent urgent information from being delivered to its destination, in order to execute the appropriate procedure to handle an emergency event [29]. However, maintaining the speed of the network is also important, so that, faster decisions can be made by the nodes in the network, such as interchanging information among different autonomous vehicles [30, 31].

2.2. SOFTWARE-DEFINED NETWORKING (SDN)

The business requirements of modern networks can change quickly, which requires improvement in the methods used to control these networks to rapidly respond to these changes. SDN has been presented as a solution to such challenges, where the traffic of the network is controlled from a central console. This centralized console can communicate with the different parts of the network that are responsible for routing the traffic by interchanging information and command using the same network [34]. The architecture of an SDN can be represented using three main layers, the application, control and infrastructure layers[35].

Unlike traditional networks, in which a designated device is required for each application, the controller in an SDN can be used to run these applications, such as firewalls, load balancing and intrusion detection, in the application layer. Hence, software is required to run in the controller of the SDN, where this software represents the control layer in the SDN's architecture. This software runs on a server in the network, which collects information from the devices on the network alongside with the information of the packets

being communicated, which are interchanged using the hardware of the network, which represents the infrastructure layer [36]. Figure 2.2 shows the generic architecture of an SDN.

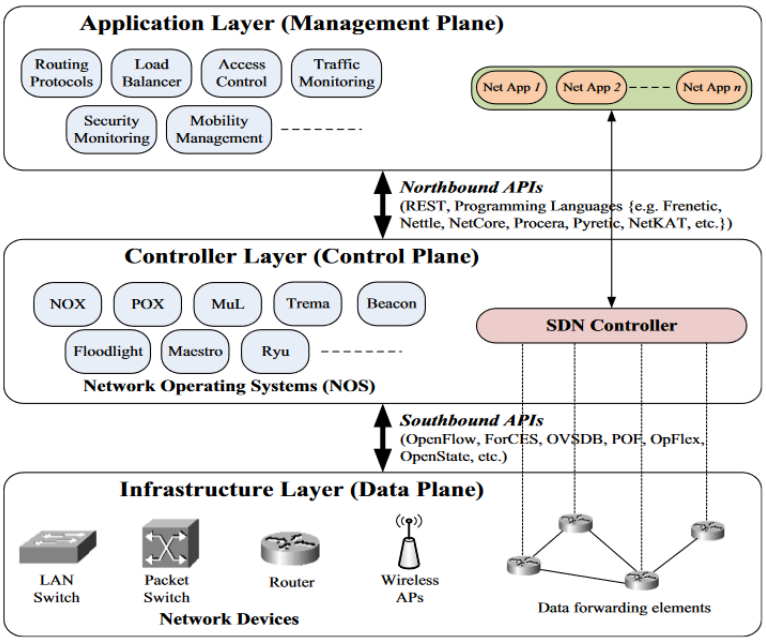

Figure 2.2: Generic architecture of an SDN [36].

The devices responsible for forwarding the packets in the network, in order to deliver them to their destinations, consult the SDN controller for the next device the packet is forwarded to. Using the information collected about the state of the network, represented by the state of the devices in that network, and the characteristics of the packet being forwarded, the controller can select the optimal operation and instruct the device, according to the configurations of the software. The controller can instruct the device to forward the packet to a certain host on the network, which can be the destination or another hop in the route recognized by the controller, or to drop that packet, if the destination is unreachable or the packet is considered an intrusion attempt, for example [37, 38].

2.3. REINFORCEMENT LEARNING

Reinforcement learning uses the concepts of agents, environments, states, actions and rewards [39-41]. As shown in Figure 2.3, the environment receives the actions selected by the agent and outputs the new state of the agent and the reward. Agents, on the other hand, collect the new state and the reward in order to select the next action, which in return produces new state and reward from the environment. However, the agent does not have a clue about the way the environment returns the next state and the rewards of a certain action. Thus, in reinforcement learning, the agent attempts to predict the action that maximizes the rewards received from the environment, by approximating the behavior of the environment and how it responds to the actions [42].

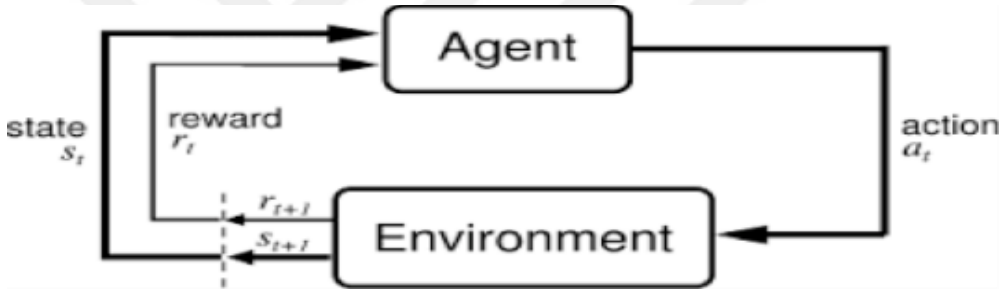

Figure 2.3: Illustration of the interaction between the Agent and the Environment in reinforcement learning.

The main components in RL applications are defined as follows:

- **Agent:** Is the component that is responsible of making the appropriate decision, depending on the state collected from the environment, to achieve the goal of the task assigned to it, such as making a delivery by a drone or navigating a car, safely, to the intended destination.
- **Action (A):** Defines the set of possible actions that an agent can take, so that, the agent can predict the reward it gets upon the execution of each action at a certain state. For an autonomous vehicle, the possible actions at any state are to accelerate, deaccelerate, go left, go right, go straight and do nothing. This set represents the simplest actions for the RL agent, where more actions can produce better performance but increase the complexity of the decision-making procedure, according to the larger possibilities.

- **Discount Factor:** To allow the agent to focus on maximizing the overall reward rather than emphasizing on the instant one, the maximum reward from the new state the agent becomes into when an action is executed is included in the computation of the current rewards. However, the reward value of the next state is reduced by multiplying it by the discount factor, so that, the effect of the instant reward and the overall reward is balanced. For instance, if an autonomous vehicle is rewarded based on the instant values only, deacceleration at risky situations is not considered by the agent, as it cannot result in the maximum instant reward. Including the final rewards in the computations increases the reward expected from avoiding accidents, which allow the agent to make the appropriate decisions in that manner. Moreover, relying only on the final reward can encourage the agent to take some unwanted actions, such as driving off roads, to maximize the final reward. Thus, the discount factor must be selected to balance all the scenarios and produce the optimal performance from the agent.
- **Environment:** The domain that the agent is interacting with, by executing the actions and collecting the rewards. In autonomous driving, the environment represents the street the car is being driven through and the traffic in those streets.
- **State (S):** The description of the current situation of the agent in the environment, which can be represented to the agent in different formats. For instance, an autonomous driver requires knowledge about the path it is following, its current position on that path, the nearest vehicle and obstacles ahead.
- **Reward (R):** Represents the feedback from the environment for the action selected by the agent. Higher rewards values indicate more appropriate actions for the current state, while lower values indicate that the correspondent actions are less appropriate for the current state. For instances, deaccelerating the vehicle may reduce the reward under certain circumstances, such as clear path and low speed, but such action can have higher rewards in states that describe an incoming vehicle, which can result in an accident.

- **Policy (π):** Is the approach employed by the agent to select the action appropriate for the current state to maximize the reward.
- **Value (V):** Under policy π, the long-term reward expected by the agent for the current state Vπ(s), considering the discount factor defined for the agent. This value allows the agent to avoid being in states that can dramatically reduce the long-term reward, even if it maximizes the instant reward. For instance, increasing the speed above the speed limit can increase the instant reward, as more distance is traveled faster, but considering the possibility of a fine or an accident allows the agent to make more reasonable decisions.
- **Q-Value (Q):** This value defines the overall reward for a certain action at a certain state, i.e. Q^{π}(s, a). The agents rely mainly on this value in making their decisions, so that, the action that returns the maximum overall reward.

Reinforcement is based on the Bellman equation, which is proposed by the American mathematician Richard Bellman. Using this equation, the reward per each action for a certain state can be calculated based on the instant reward and all the rewards collected until the end of the episode, which can be terminated as the agent reaches its goal or by performing a specified number of actions [18, 43]. This reward is calculated as shown in Equation 2.1.

$$Q^{\pi}(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma R_{t+3} + \cdots | s_t, a_t] \quad (2.1)$$

According to this equation, the highest Q value from a certain state, s_t, can be used to calculate the Q value for any action that ends up with the agent in that state, by simply multiplying it by the maximum Q value, as shown in Equation 2.2.

$$new\ Q(s, a) = Q(s, a) + \alpha[R(s, a) + \gamma max Q'(s', a') - Q(s, a)] \quad (2.2)$$

where the learning rate α is used to damp the variation in the Q value for the selected action in the current state and γ is the discount factor that controls the balance between the instant and long-term rewards. The new Q value is then used to update the function that is used to represent the environment, so that, the actual reward from executing the action is produced instead of an approximation. This value also assists the computation of the reward values expected in previous states, as this value provides the actual reward received from the environment.

2.4. ARTIFICIAL NEURAL NETWORKS

According to the complexity of the computations required to predict the reward of each action available for the agent at a certain state, deep neural networks are used to make the predictions, based on the inputs collected from the environment at that state. The action selected based on the predictions of the neural network aims to maximize the reward received from the environment by executing the action with the highest. However, these rewards may not be instant and depend on the results of a series of actions. Thus, it is important to wait to the end of the interaction with the environment to evaluate these actions. Moreover, in order to adjust the exploration and exploitation of the selected actions, a certain percentage of random actions in any time instance is allowed to be executed by the agent, especially before providing significant training for the neural network. This enables finding the optimal output, even after a certain solution is found [44].

Artificial neural networks consist of neurons distributed in layers, where the output of a neuron is weighted and connected to another neuron in a different layer, as input [45], as shown in Figure 2.4. The decisions made by these networks depend on the values of these weights, which are updated using backpropagation [46]. Backpropagation measures the difference between the output of the neural network, and the actual output required from it and update the weights among the neurons, based on the effect of each weight over the output. The effect of each weight is measured by calculating the rate of change of the output values, with respect to that weight. Thus, these computations require intensive processing and neural networks with more layers, known as deep neural networks, have significantly more weights, which increases the complexity of the computations. These computations are very exhausting for the IoT device, according to their limited resources, and require larger computers to achieve them [47]. However, these computations occur during the training phase of the neural network only, and no further updates are required during runtime, in most cases. The computations required to calculate the output of a neural network are relatively easier than those required to train it, and they can be handled by the IoT device itself, as the output of each neuron can be calculated by simply passing the weighted summation, of the outputs collected from the neurons connected to it, through an activation function [20, 48].

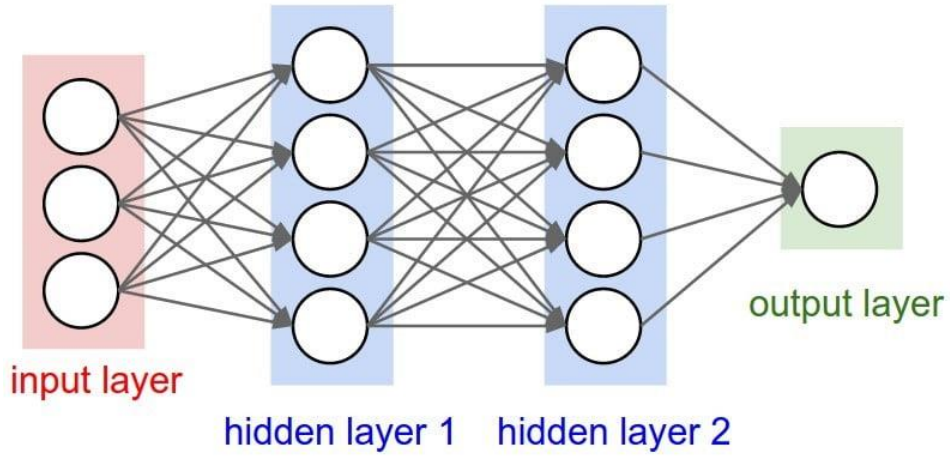

Figure 2.4: Hierarchy of a sample deep neural network.

Depending on the distribution of the inputs collected by each neuron in a layer, different types of artificial neural networks can be produced, for different tasks. The Feed-Forward Neural Network (FF-NN) shown in Figure 2.4 is the basic neural network that is used in different applications. However, when the neuron collects its inputs from two-dimensional windows, i.e. filters, that are convoluted through the two- or three-dimensional input, the neural network is known as Convolutional Neural Network (CNN). CNNs have shown significantly better performance than other types of neural networks when the inputs of the neural networks are images, which are normally represented in two- or three-dimensional arrays. Such better performance is the result of it CNN's ability of detecting and combining local features detected by the filters, regardless of their position in the input [49].

CNNs contain convolutional layers, which consists of two- or three-dimensional filters that are convoluted throughout the input of each neuron. Mathematically, the filter is actually the weight values of that neuron, which enable the neuron to detect local two-dimensional patterns in the input. The sizes of the filters in a convolutional layer is constant and patterns in the input can be detected within the size of the filter. However, by going deeper into the neural network, i.e. layers farther from the input layer, each filter detects patterns defined by the patterns detected by the previous layer's filters. This enables the CNN to combine the recognized patterns and detect more complex features. Although the output of a neuron in a convolutional layer can have different dimensions from its input, the number of

dimensions is similar to that in the input, i.e. a neuron processing a two-dimensional input outputs a two-dimensional array [50, 51].

During convolution, the number of values that the filter moves per each step is defined as the strides, which can have different values for the horizontal and vertical movements. All the values within the filter are multiplied with their corresponding weights and processed in the neuron, which arranges its outputs according to the arrangement received during the convolutions of its filters. Skipping more than one value per each convolution can cause the loss of detecting important patterns, which can negatively affect the performance of the CNN, despite the reduction in the size of the neuron's output, which can simplify the computations in following layers. To reduce the size of the output from a neuron without losing important information, pooling layers can be placed after a convolutional layer to subsample the values outputted from the convolutional layers [52], as shown in Figure 2.5.

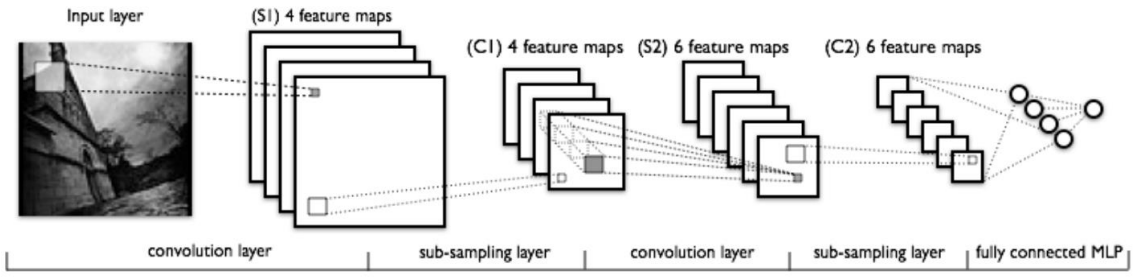

Figure 2.5: A sample convolutional neural network [53].

A pooling layer also consists of filters that are convoluted throughout its input, which is the output of the neuron. However, these filters have a different approach to process the input values, as they are not forwarded to a neuron and has no weights. Despite the existence of different types of pooling layers, Max-Pooling layer is one of the widely used pooling layers that are used to reduce the size of the processed data without losing important information. As shown in Figure 2.6, the filter in a max-pooling layer searches for the maximum value within its dimensions, and outputs that value to represent that region. By selecting the highest value, the most important feature in that region is selected, so that, it is less likely to lose important information as in increasing the strides of the filter in the convolutional layer [52].

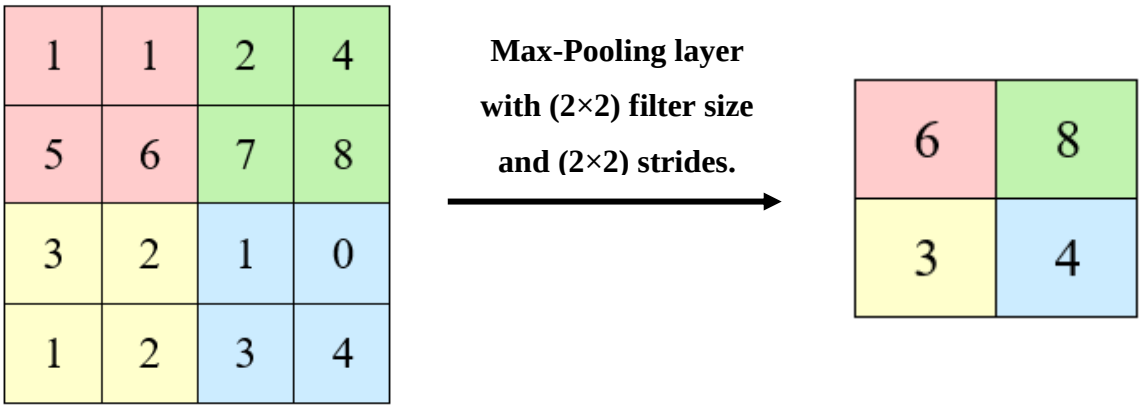

Figure 2.6: Sample input and output of a Max-Pooling layer.

As Figure 2.5 shows, when a max-pooling layer is used, the values less than the maximum values are neglected. Hence, only the feature the has maximum matching with the convolutional filter is selected within the window of the pooling layer. However, in some applications, it is still important to consider the other values in that window and maintain the size reduction produced using pooling layers. Thus, average-pooling layers are used in neural networks that are required for such applications, where the value outputted from each pooling filter is equal to the average of the values in that filter, as shown in Figure 2.7.

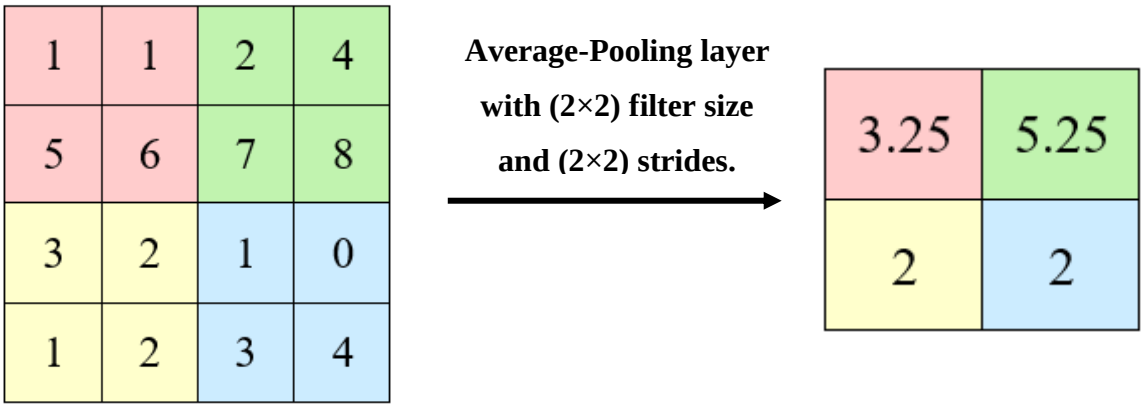

Figure 2.7: Sample inputs and outputs of an average-pooling layer.

2.5. DEEP Q-LEARNING

The use of artificial neural networks to approximate the function that defines the environment and predict the Q values per each action for a certain state, so that, the agent can select the most appropriate action is known as Q-Learning. The aim of this learning approach is to provide the neural network with the actual rewards collected from the environment, so that, it can predict these rewards in future operations [17]. However, as the neural network does not have any knowledge about the environment that the agent is interacting with, the training process relies on executing random actions at the beginning of the training [54]. As the neural network starts to gain more knowledge about the environment, the decisions of the agent can start to be less random and more dependent on the predictions of the neural network. To control such behavior, a value is defined to control the randomness in the decisions made by the agent. This value is denoted as the epsilon and it normally starts with a high value, i.e. more random actions, and reduced as the neural network gains more knowledge about the environment [55].

To select between the execution of a random action or based on the outputs of the neural network, the epsilon value is compared to a randomly generated value. If the random value is less than the epsilon, the action selected by the agent is the action that produces the highest reward, based on the predictions of the neural network. Otherwise, the action is selected randomly and executed against the environment [56]. In both cases, the reward collected from the environment upon the execution of the selected action at the current state is used with the maximum Q value predicted by the neural network for the new state the agent becomes in, to produce a new Q value that is used to train the neural network [57].

When the agent finishes an episode, the neural network is trained using the data collected by the agent during the episode, i.e. the states, actions and rewards, and the epsilon value is reduced by a predefined ration, known as the gamma value. This process is repeated until the defined number of training episodes is reached, in which the neural network is expected to have gained enough knowledge to produce accurate Q value that can assist the agent to select the optimal action per each state it faces [18, 58]. The ability of the neural networks to provide approximations for states that it has never been through, during the training, allows the employment of these networks in the Deep Q-Learning (DQN) approach, so that,

the agent still has approximate Q values to make the appropriate decision. Comparing this approach to the use of tables that contains the states and their corresponding Q values shows the benefits of the approximated computations, as Q values for states that are included in the Q table can be recognized by the agent [59, 60]. Thus, DQN has been widely used in approximating the functions of complex environments, such as those faced by autonomous vehicles drivers.

2.6. EMPLOYMENT OF DRL IN SOFTWARE-DEFINED NETWORKING

As illustrated earlier in this chapter, the software in the SDN controller is required to handle very complex and dynamic states and present the appropriate decisions to the devices responsible for forwarding the packets to their destinations. Hence, static rules can impose dramatic restrictions on the operation of the network, as such rules are not able to handle changes in the topology of the network. Moreover, according to the ability of DRL to interact with states that have not been included in the training i.e. the agent has not been through before, several methods are proposed in the literature as the backbone of the SDN controller. Zhang et al. [24] propose a DRL-based method for the SDN's controller to manage the traffic in Vehicular ad hoc Networks (VANETs). The main challenge in such networks is the continuous movement of the nodes in the network, i.e. the vehicles. Hence, this method emphasizes on finding the route that has the highest probability of delivering the packet to its destination, without being affected by the movement of the nodes in the network. To train the DRL technique for such a task, the Packet Delivery Rate (PDR) is used as the reward of the DQN. The results show that the best performance has been achieved by the proposed method when the CNN is used to predict the reward values for each network state and use it to select the best route. Similarly, the method proposed by Lin et al. [23] also considers the PDR in the computations of the reward value but also include the delay of packets before reaching their destinations and the overall throughput of the network. However, both methods do not consider the resources consumption of the nodes in the network, which can exhaust certain nodes depending on their positioning and the flow of the traffic in the network.

Stampa et al. [25] propose a method that aims to reduce the time each packet travel, in order to reach its destination. Hence, this method attempts to reduce the number of hops, i.e. nodes, that forward the packet and attempt to find the shortest route, so that, the highest reward is collected. However, focusing only on the number of hops neglects balancing the load over the nodes in the network. For instance, communicating packets between the red and blue nodes in the sample WSN shown in Figure 2.8 without considering balancing the load can exhaust the resources of the red node. To avoid such exhaustion, it is important to forward some of these packets through the green nodes despite the longer path. Considering such balance can significantly increase the lifetime of the network.

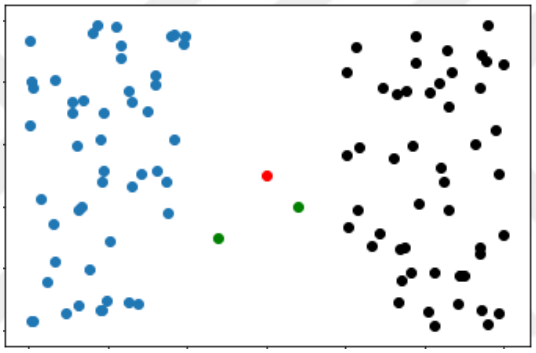

Figure 2.8: A sample wireless sensor network.

3. METHODOLOGY

The method proposed in this thesis relies on DQN to manage the traffic in a WSN, so that, the loading is balanced among the nodes to extend the lifetime of the network. This DQN predicts the reward value, which represents the lifetime of the network when the packet is forwarded to every possible node in that network. Hence, the node that maximizes the lifetime of the network when the packet is forwarded to it is selected as the packet's next hop. The use of the networks' lifetime guarantees balanced loading on the nodes in the network, as the use of longer paths or exhausting the power source of one of the nodes can both reduce the lifetime of the network. However, it is important to consider the constraints of communications in such networks, which are the limited transmission power of the nodes and the packets delivery rate.

Different types of neural networks are evaluated for the proposed method, where the information collected about the nodes in the network as well as the packet is delivered to the neural network in order to select the next hop. Each neural network is trained to predict the reward value using RL, where the lifetime of the network is used to represent the quality of the decisions. Moreover, decisions that are not in compliance with the constraints of the network are punished, i.e. negative reward values are assigned to them. The neural networks used for the DQN are a Feed-Forward Neural Network. 2D-CNN and 3d-CNN.

In order to route a packet through the network, it is important for the controller to recognize the state of the packet and the network, so that, the appropriate decision, i.e. action, is selected by the RL model in the controller. The main information required by the controller are:

- **Remaining energy:** To accomplish the required load balancing, it is important for the controller to monitor the energy remaining in each node. Accordingly, alternative routes can be selected to avoid exhausting certain nodes.
- **Positioning:** The positioning of the nodes is required to select the next hops that is capable of delivering the packet to its destination and within the range of the node that the packet is currently at. Hence, the node that is in the required direction to

accomplish the route and can receive the packet using the limited transmission power of the node.

- **Source and Destination:** The source and destination nodes must be known to the controller in order to define the appropriate path for the packet, based on the positioning of the nodes in the network and the remaining energy in each node.

3.1. USING THE FEED-FORWARD NEURAL NETWORK (FFNN)

As illustrated in Section 2.4, the input of an FFNN is a one-dimensional vector. Hence, the state of the network is summarized in such a vector, which represents the following features:

1. The energy remaining per each host.
2. The distance between each host and the source host, where the source host can be recognized as the host corresponding to the position of the values 0.
3. The distances between each host and the destination host, where the destination host has a distance of 0 in this set.
4. The distances between each host and the host the packet is currently at.
5. The hosts that the packet has been through, where a value of 1 is correspondent to the host that the packet has passed through.

Accordingly, the number of features in the vector is equal to $5 \times N$, where N is the number of nodes in the WSN. However, as the size of the input layer of the neural network must be equal to the number of features in the vector and according to the possibly varying number of nodes in a WSN, the value of N is set to 100, so that, the trained neural network can be used with WSNs of up to 100 nodes. The values correspondent to nodes that do not exist, i.e. WSNs with less than 100 nodes, are set to -1, so that, the neural network can recognize the absence of these nodes. Hence, the number of neurons in this neural network is set to 500, while the output layer contains 100 neurons. The value outputted from each neuron in the output layer represents the reward value predicted if the packet is forwarded to that node. Thus, forwarding the packet to the node corresponding to the neuron with the highest output is expected to maximize the lifetime of the network. The structure of the FFNN implemented for the proposed method is shown in Table 3.1.

Table 3.1: Structure of the FFNN implemented for the proposed method.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               256512
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_4 (Dense)              (None, 100)               12900
=================================================================
Total params: 433,636
Trainable params: 433,636
Non-trainable params: 0
_________________________________________________________________
```

3.2. USING THE 2D CONVOLUTIONAL NEURAL NETWORK (2D-CNN)

According to the limitation imposed using the FFNN, regarding the limited number of possible nodes in the WSN, convolutional neural networks are employed for the proposed method. According to the ability of CNNs to accept three-dimensional inputs, the values can be positioned according to the positioning of the nodes in the network. Hence, the input to the neural network is presented in a three-dimensional array of size 100×100×5. In other words, the input consists of five layers, each with 100×100 values. The values in each layer represent a certain feature, which are:

1. The remaining energy of each node.
2. A value of one corresponding to the position of the source host.
3. A value of one corresponding to the position of the destination host.
4. Value of ones positioned at the positions of the nodes that are within the range of the node that the packet is currently at.

5. Route description for the hosts that the packet has been through up to the current hop, where the source node is assigned with zero while the current host is assigned with one. Other hosts that the packet has passed through are assigned with value depending on the sequence of hosts in the route, lower values are assigned to the hosts the packet has passed through earlier. These values are calculated by using the algorithm shown in 3.1.

Input: Hops' list of a packet; Position of nodes. **Output:** Two-dimensional representation of the hops list.	
Step1:	H ← Read hops list. L ← Length(H) R ← Full(100×100, -1). //A 100×100 array the value -1 for the output.

Step2:	For i = 1 to L:
	p = H(i).position //Find the position the node of the current hop.
	R[p] ← i/L

Step3:	Return R

Figure 3.1: Hops representation algorithm.

The output of the CNN is a two-dimensional array, where the reward values are mapped on the position corresponding to each node, so that, the node closed to the highest reward value is selected to forward the packet. The structure of the 2D-CNN implemented for this method is shown in Table 3.2.

Table 3.2: Structure of the 2D-CNN implemented for the proposed method.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 100, 100, 32)	544
conv2d_2 (Conv2D)	(None, 100, 100, 16)	2064

```
conv2d_3 (Conv2D)            (None, 100, 100, 8)       1160
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 100, 100, 4)       804
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 100, 100, 1)       101
=================================================================
Total params: 4,673
Trainable params: 4,673
Non-trainable params: 0
_________________________________________________________________
```

3.3. USING THE 3D CONVOLUTIONAL NEURAL NETWORK (3D-CNN)

Unlike 2D-CNNs, the three-dimensional filters in the 3D-CNNs allow detecting features that have values distributed in several layers of the input array. However, as these values represent a single node and according to the need for a two-dimensional output, producing a single reward value per each node, an Average-Pooling layer is used to summarize the values per each node into a single value. Hence, a single value is produced in the hidden layers of the neural network that can be used to predict the reward value if that node is selected. The values in the layers are similar to those collected in 2D-CNN, but the filters in the Average-Pooling layer are set to summarize the features detected by the first hidden layer into a single value, i.e. the size of the filter is set to (1, 1, 5). According to the existence of weight values between the input and the first hidden layers, the effect of each feature value, e.g. the remaining power and route description, are adjusted in the value produced by the average-pooling filters, according to the requirements detected by the neural network during training. The structure of the implemented 3D-CNN is shown in Table 3.3

Table 3.3: Structure of the 3D-CNN implemented for the proposed method.

Layer (type)	Output Shape	Param #
conv3d_1 (Conv3D)	(None, 100, 100, 4, 32)	160
average_pooling3d_1 (Average	(None, 100, 100, 1, 32)	0
conv3d_2 (Conv3D)	(None, 100, 100, 1, 16)	8208
conv3d_3 (Conv3D)	(None, 100, 100, 1, 8)	4616
conv3d_4 (Conv3D)	(None, 100, 100, 1, 4)	3204
conv3d_5 (Conv3D)	(None, 100, 100, 1, 1)	401

Total params: 16,589
Trainable params: 16,589
Non-trainable params: 0

3.4. TRAINING THE DQN MODELS

Initially, the neural network has no knowledge about the rewards it can get for each action. Hence, the packets are forwarded in a random manner, so that, the reward returned by the network based on the selected action, i.e. next hop, is used to train the neural network. After a few iterations, the neural network starts to gain knowledge about the environment and how to deliver the packets from one node to another. However, this knowledge can be limited to the approaches recognized during the use of random actions. For example, the neural network may start to learn to deliver the packet to the destination node using the shortest path but still unable to extend the lifetime of the network. Thus, a fraction of the decisions is still required to be executed randomly in order to balance exploration and

exploitation. Thus, a variable with a value equal to one is set at the first iteration and compared to randomly generated numbers in the interval [0,1], so that, if the random number is greater than the value of the variable, the action is selected based on the output of the neural network. Otherwise, the action is selected randomly. This value of this variable is reduced by multiplying it to 0.99 after each iteration, so that, the number of actions selected based on the predictions of the neural network is increased as the knowledge of the neural network increases.

Per each iteration, the training of the neural network is continued until the energy of one of the network's nodes is drained. Then, the lifetime of the network is used to update the reward values of the neural network. However, as the delivery of each packet in the network is not related to other packets, the lifetime of the network is used to update the reward values of each packet solely, i.e. the packets deliveries are considered parallel operations rather than serial and the reward value is assigned for the last action or hop. This value is reduced using the discount factor (Gamma), which is set to 0.9, as higher values for the discount factor have shown better performance in [23].

The predictions of the neural network are updated using the formula shown in Equation 3.1, where Q is the predicted reward value for executing action a in state s. R is the actual reward value retrieved from the environment after executing the action, maxQ' is the maximum reward expected from the agent after being in the new state s', i.e. after forwarding the packet to the next hop.

$$New\ Q(s,a) = Q(s,a) + \propto (R(s,a) + \gamma maxQ'(s',a') - Q(s,a)) \quad (3.1)$$

As the value computed using this formula represents only the reward value of the node the packet is forwarded to, the reward values for the other nodes are maintained as predicted by the neural network. Using such an approach, any prior knowledge is maintained and the knowledge extraction can continue even when random actions are selected. However, to present the knowledge required by the neural network to avoid forwarding packets to positions that do not nodes in them, reward values of -1, i.e. punishments, are placed in the positions that have no nodes in them. This training procedure is conducted after the first node in the network is exhausted. However, instant training occurs when one of the following conditions occurs:

- The packet is forwarded to a node that is out of the transmission range of the current node.
- The packet is forwarded to a node that does not have sufficient power to receive or forward the packet unless it is the destination node.
- The packet is forwarded to a node that is in the list of hops that the packet has been through, to avoid infinite loops.
- The number of hops the packet passes through exceed 10 times the number of nodes in the network.

4. EXPERIMENTAL RESULTS

The proposed method is implemented using the different neural networks models employed for the DQN. To train these neural networks, a set of 100 randomly generated WSNs is used, with a number of nodes varying from 8 to 32 randomly distributed in a region with 1000×1000 square meters. However, to ensure the generation of the same networks for all the evaluated methods, random seeds are used with similar values, so that, the exact same random numbers are generated per each experiment. Each node is initiated with the energy of 1 joule, where the transmission or receival of a packet consumes 5×10^{-9} joule. The packets are set to be of 1024 bytes in size with 2Mbps data rate. The maximum distance a node can transmit a packet is set to 300 meters. Hence, a packet that is transmitted to a farther node is considered a failed transmission. Moreover, each node consumes 10^{-10} joule/sec when it is in idle mode, i.e. no packets are sent from or to that node. Figure 4.1 shows examples of the randomly generated WSNs with the randomly generated sample traffic.

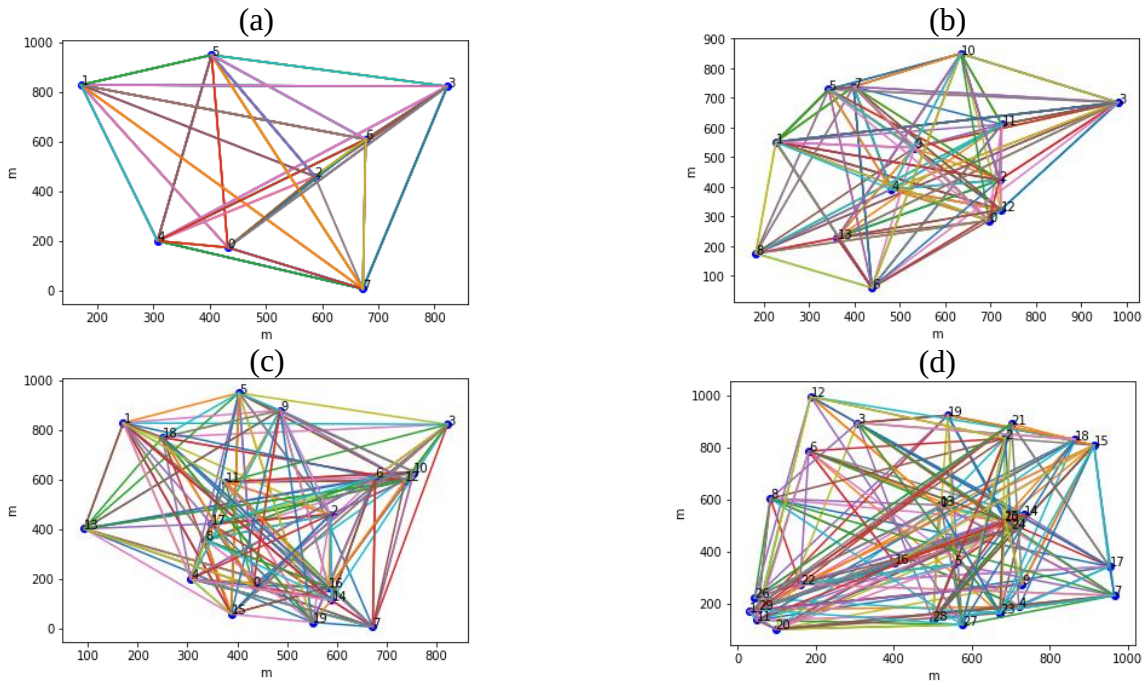

Figure 4.1: Samples of the randomly generated WSNs and traffic. (a): 8-node network; (b): 14-node network; 20-node network; 30-node network.

The experiments are implemented using Python programming language [61] using a computer with Intel® Core™ i7-7700 processor with 2.8GHz frequency and 16GB of random access memory. Python's Keras library [62] is used to implement the neural networks used for the DQN, using Google's Tensorflow machine learning library [63] backend. The Keras library produces the matrices and their operations to be conducted using Tensorflow, so that, the same efficiency acquired from Tensorflow can be implemented using simpler representation in Keras. Each DQN is trained using 100 randomly generated WSNs and evaluated using 10 other networks. The packets are generated randomly, i.e. random source and destination hosts, until one of the nodes in the WSN exhausts its power.

To reach its destination, a packet passes through one or more hops, depending on the position of the source and destination hosts in the WSN. However, in some situations, the packet may not be delivered to its destination. For instances, if the source or destination hosts are out of the range of remaining nodes or if the network controller fails to make the appropriate routing decisions. Accordingly, the ratio between the number of packets to the total number of packets initiated in the network, i.e. the lifetime of the WSN is calculated based on the total number of hops H the packets travel through, the size of these packets, which is 1024 bytes in this setup, and the bandwidth of the network, which is 2Mbps, as shown in Equation 4.1. The average prediction time required by the method to predict the next hop for the packet is also measured to illustrate the delay imposed by the network controller according to the complexity of the model. For a total of H hops each required E_h second to be calculated, the average prediction time is calculated as shown in Equation 4.2 The average number of hops that the network packets travel in order to reach their destinations is also calculated as a performance measure, as longer paths can produce more delay for a packet to be delivered according to the limited bandwidth in the network. For T total packets, each is delivered using P_t hops, the average number of hops can be calculated as shown in Equation 4.3. Finally, the PDR, is calculated as a performance measure of the routing method. Hence, for a WSN that has been able to deliver D packets to their destinations, out of T total packets, the PDR can be calculated shown in Equation 4.4.

$$lifetime = \frac{H \times 1024 \times 8}{2 \times 10^6} \tag{4.1}$$

$$Average\ Prediction\ Time = \frac{\sum_{h=1}^{H} E_h}{H} \tag{4.2}$$

$$Average\ hops = \frac{\sum_{t=1}^{T} P_t}{T} \tag{4.3}$$

$$PDR = \frac{D}{T} \times 100\% \tag{4.4}$$

4.1. PERFORMANCE OF THE FFNN MODEL

After training the implemented FFNN for the DQN using 100 WSNs, its performance is evaluated using the other 10 WSNs. The average lifetime of these networks is 593814.16 sec, with an average number of hops equal to 13.61 hops per packet and 83.37% PDR. Moreover, each prediction of the next hop has consumed an average of 278.82*uS*. The average minimum power remaining on the nodes is calculated for these networks as they operate and illustrated in Figure 4.2, which shows that when the minimum available power decreases, the FFNN attempt to avoid forwarding the packets through that node. This behavior is reflected by the decreasing slope of the average minimum value as time progresses.

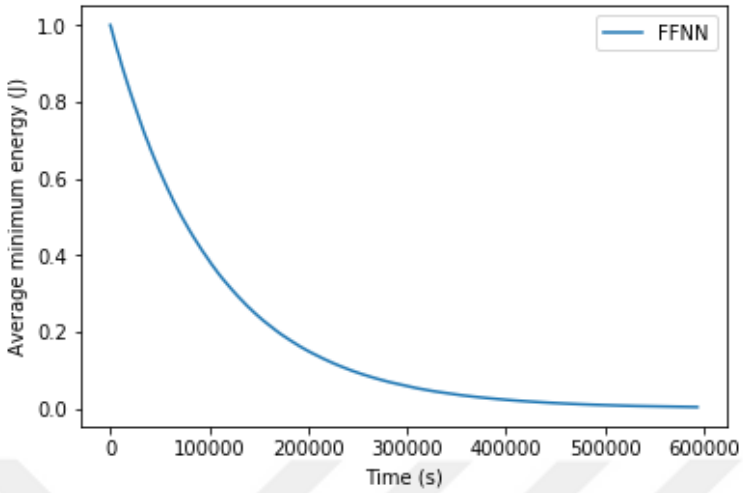

Figure 4.2: Average minimum energy in the nodes of the evaluation WSNs using the FFNN model.

Moreover, the average number of hops is also monitored during the operation of the WSNs, using the proposed method as the traffic flow manager. For better illustration, the values are summarized for every 1000 seconds, i.e. these average values are also averaged. These values are noticed to be increasing as time progresses, according to the use of longer paths to avoid consuming power from the nodes with less remaining energy, as shown in Figure 4.3.

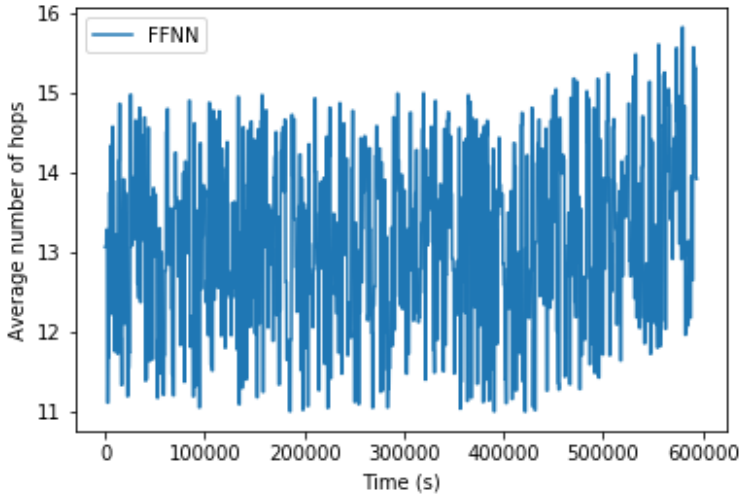

Figure 4.3: Average number of hops versus time using the FFNN model.

Additionally, the average PDR has also been monitored during the operation of the WSNs, which is also measured for every 1000 seconds to produce a better illustration. As shown in

Figure 4.4, the proposed method has been able to maintain high PDR despite the longer paths the packets travel when the energy of the nodes starts getting exhausted.

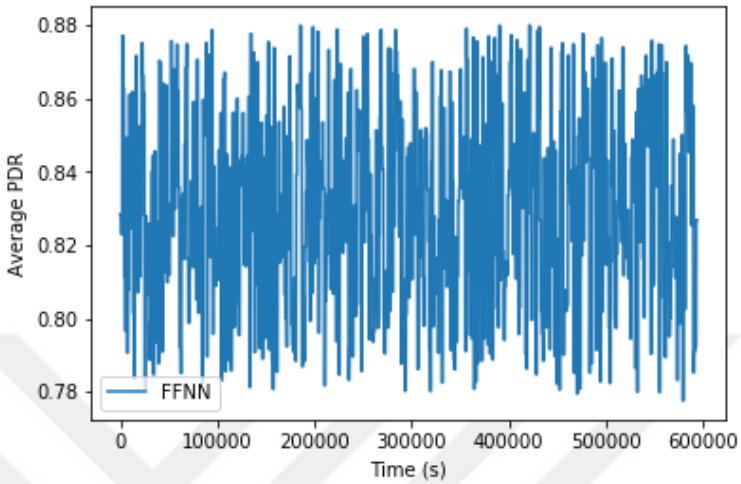

Figure 4.4: Average PDR versus time using the FFNN model.

4.2. PERFORMANCE OF THE 2D-CNN MODEL

Similar to the previous experiment, the 2D-CNN is trained using the same 100 WSNs, before being evaluated using another 10 networks. The average lifetime of these networks is 638169.21 sec, which is large than the time achieved by the FFNN. The average number of hops in this experiment is 12.37 hops per packet with 82.47% PDR and 316.03*uS* average prediction time. The average minimum power remaining in the nodes is calculated for these networks as they operate and illustrated in Figure 4.5, which shows that when the minimum available power decreases, the 2D-CNN also attempts to avoid forwarding the packets through that node. This behavior is reflected by the decreasing slope of the average minimum value as time progresses.

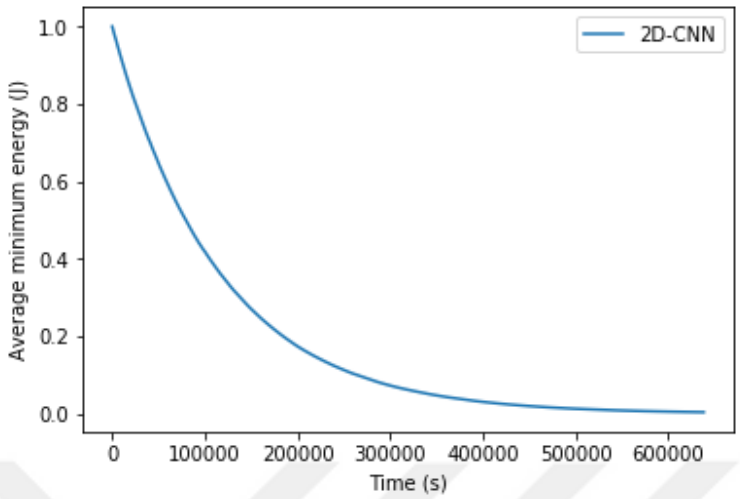

Figure 4.5: Average minimum energy in the nodes of the evaluation WSNs using 2D-CNN model.

Moreover, the average number of hops is also monitored during the operation of the WSNs, using the proposed method as the traffic flow manager. For better illustration, the values are summarized for every 1000 seconds, i.e. these average values are also averaged. It is noticeable that the average number of hops starts to increase after 400,000 seconds as the minimum power of the nodes starts becoming critical. However, by the end of the networks lifetime, this model starts to favor shorter paths, as the energy of most of the nodes becomes lower and avoiding them becomes more exhaustive to the overall network, as shown in Figure 4.6.

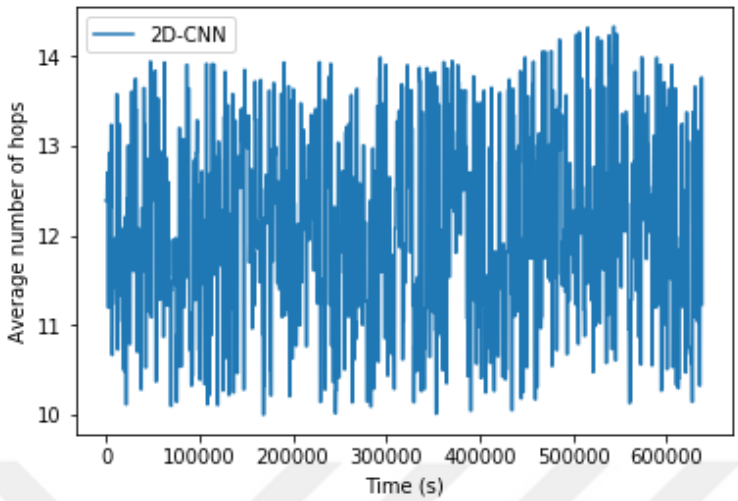

Figure 4.6: Average number of hops versus time using the 2D-CNN model.

Additionally, the average PDR has also been monitored during the operation of the WSNs, which is also measured for every 1000 seconds to produce a better illustration. As shown in Figure 4.7, the proposed method has been able to maintain high PDR despite the longer paths the packets travel when the energy of the nodes starts getting exhausted.

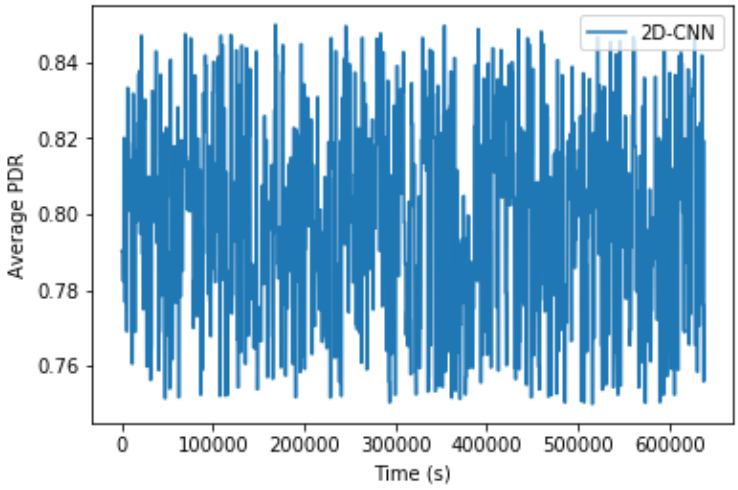

Figure 4.7: Average PDR versus time using 2D-CNN model.

4.3. PERFORMANCE OF THE 3D-CNN MODEL

After training the implemented 3D-CNN for the DQN using 100 WSNs, its performance is evaluated using the same 10 WSNs in the previous experiments. The average lifetime of these networks is 678251.62 sec, with an average number of hops of 9.81 hops per packet and 85.07% PDR, where each prediction has consumed an average of 351.71*uS*. The average minimum power remaining on the nodes is calculated for these networks as they operate and illustrated in Figure 4.8, which shows that this model has achieved less slope, compared to the previous experiments. Hence, this model has achieved a better balancing of energy consumption among the nodes in the network.

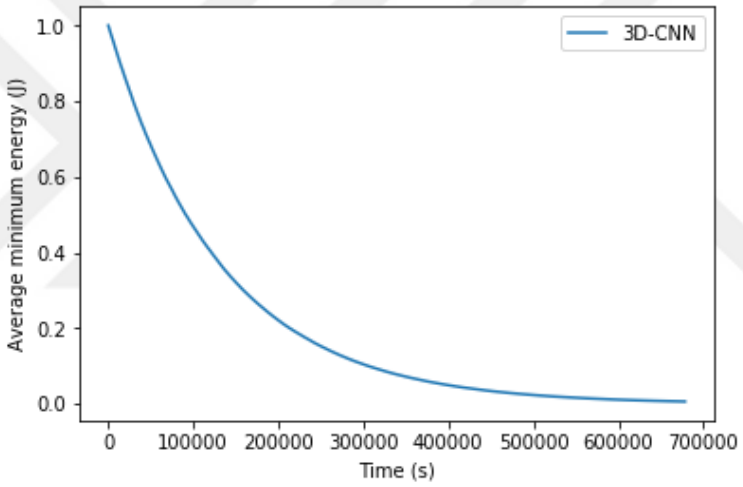

Figure 4.8: Average minimum energy in the nodes of the evaluation WSNs using 3D-CNN model.

Moreover, the average number of hops is also monitored during the operation of the WSNs, using the proposed method as the traffic flow manager. For better illustration, the values are summarized for every 1000 seconds, i.e. these average values are also averaged. A noticeable change in the average number of hops cannot be recognized at any range of time in the lifetime of the network. This behavior combined with the lower overall average of number of hops per packet, compared to the 2D-CNN, shows that the 3D-CNN model has attempted balancing the load earlier than the 2D-CNN, i.e. before the energy of the nodes starts to get exhausted.

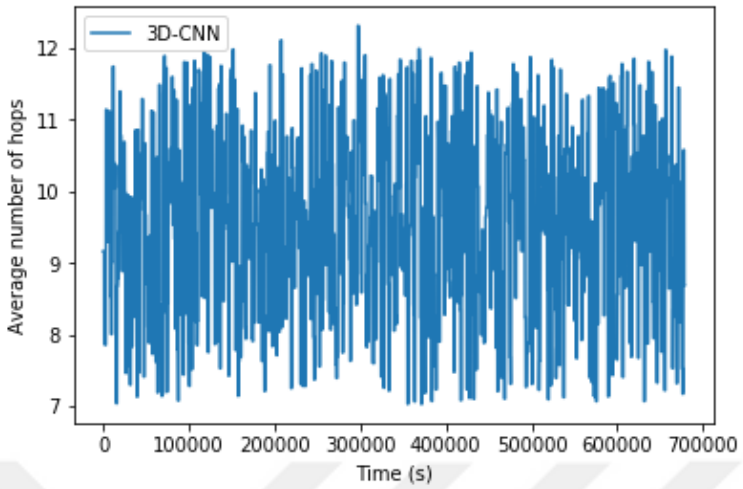

Figure 4.9: Average number of hops versus time using the 3D-CNN model.

Additionally, the average PDR has also been monitored during the operation of the WSNs, which is also measured for every 1000 seconds to produce a better illustration. As shown in Figure 4.10, the proposed method has been able to maintain high PDR until the end of the lifetime of the WSN.

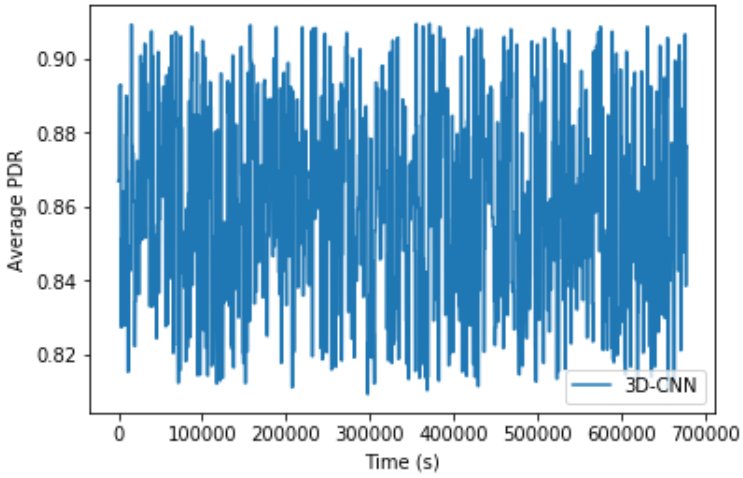

Figure 4.10: Average PDR versus time using 3D-CNN model.

5. RESULTS SUMMARY AND DISCUSSION

The results collected from the experiments conducted in this study are summarized in Table 5.1, which also summarizes the results of state-of-the-art studies that also use DRL for SDNs. This summary shows that the proposed method using the 3D-CNN DQN model has been able to achieve the highest lifetime of the network, whereas the FFNN has achieved the lowest, as also illustrated in Figure 5.1. However, despite the similar lifetime of the WSNs between the FFNN and the method proposed by Lin et al. [23], all DQN models have achieved longer lifetime, according to the use of this value as the reward for the DQN.

Table 5.1: Summary of the performance measure for the existing and proposed methods.

Method	Lifetime	Prediction Time(uS)	Average Hops	PDR(%)
FFNN	578382	278.8	8.31	83.37
2D-CNN	638169	316	12.37	82.47
3D-CNN	**678252**	351.7	9.81	**85.07**
Lin et al. [23]	578122	328.1	9.32	81.68
Zhang et al. [24]	541840	341.9	10.53	72.55
Stampa et al. [25]	520364	**283.1**	**8.76**	64.71

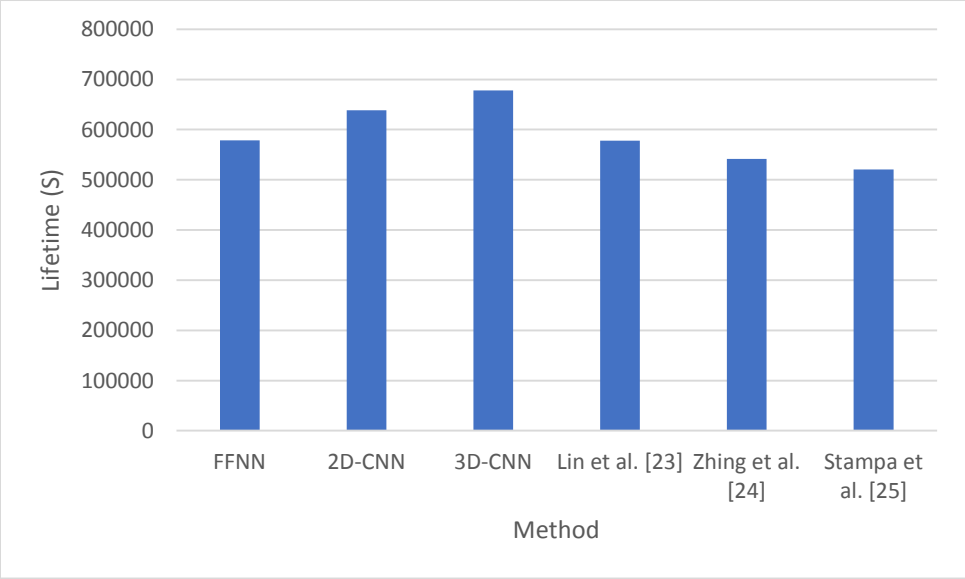

Figure 5.1: Summary of the average lifetime for the proposed and existing methods.

The 3D-CNN model has been able to extend the lifetime of the network by 17% compared to the highest lifetime using the existing method, which is achieved by Lin et al. [23]. Moreover, this model has also been able to improve the PDR of the network with a quite similar average number of hops, which proves the hypothesis of this study that the use of the networks lifetime as the reward value can balance the overall performance of the network. Additionally, the highest average number of hops, shown in Figure 4.2, using the method proposed in this study is expected behavior, as the SDN controller is required to use longer paths in order to avoid exhausting the resources of a certain node.

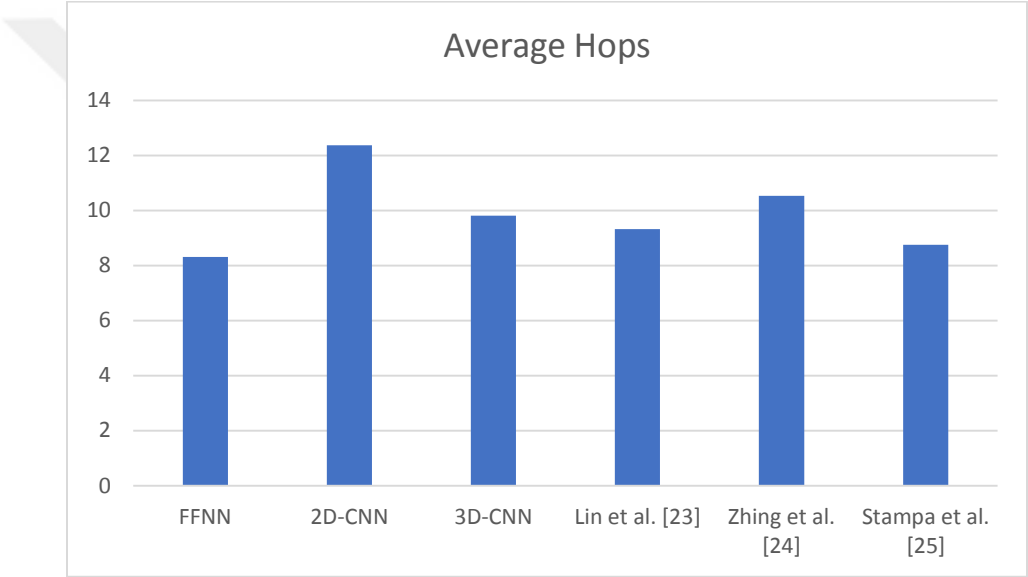

Figure 5.2: Average number of hops using the proposed and existing methods.

Although the FFNN has only been able to slightly extend the lifetime of the WSN, compared to the method proposed by Lin et al. [23], this model has been able to reduce the average number of hops required to deliver the packets an improve the PDR, as shown in Figure 5.2 and 5.3. However, the main drawback of using the FFNN is the limited number of possible nodes in the WSN, which must be less than or equal to the number of neurons in the output layer. Otherwise, the implementation and training of a new neural network are required to handle larger WSNs, which may increase the time required to predict the optimal node a packet must be sent to. In the 2D- and 3D-CNNs, the same neural network can be used regardless of the number of nodes in the network and the dimensions of the regions the network is placed in. The values are mapped into a fixed-size array, according to their actual position in that region. Hence, the prediction time is constant for these

methods. Figure 5.4 shows the average prediction time required by the proposed and existing methods.

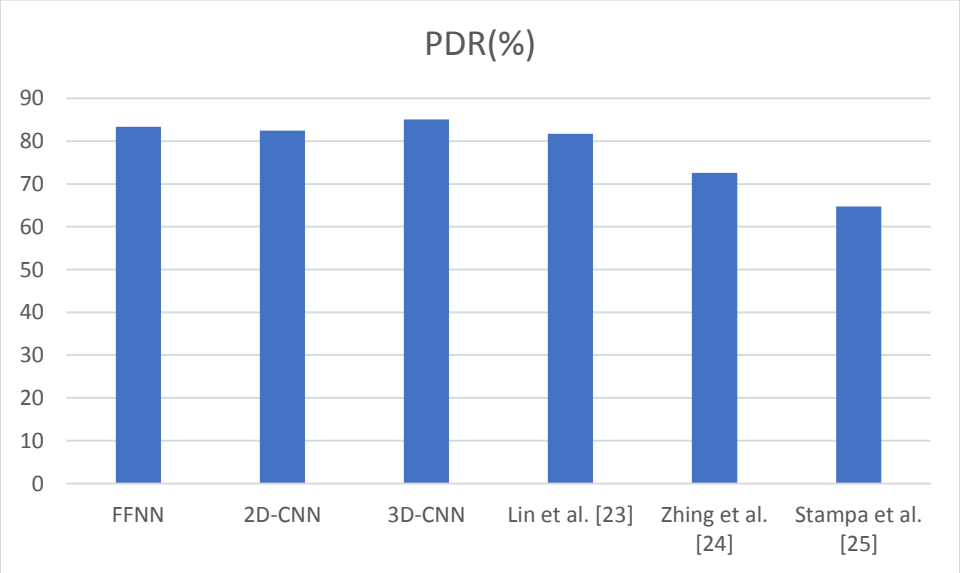

Figure 5.3: Summary of the PDR for the existing and proposed methods.

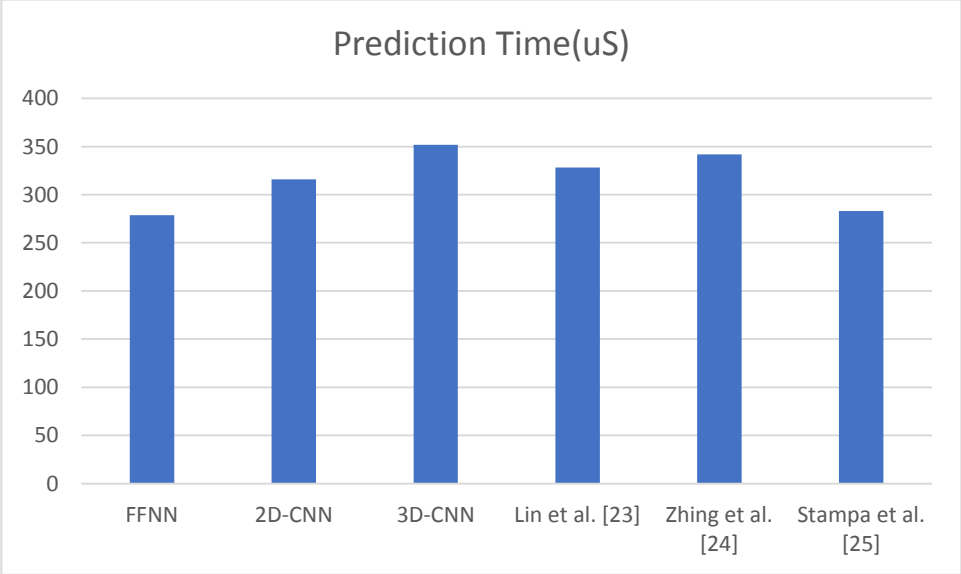

Figure 5.4: Average prediction time required by the existing and proposed methods to predict the optimal next hop.

6. CONCLUSIONS AND FUTURE WORK

The use of smaller highly-mobile devices to collect information and execute commands remotely has presented the era of the Internet of Things. Normally, these devices are of limited resources, such as processing power, memory and energy sources. However, to maintain the flexibility of networks used to exchange the information in these WSNs, the IoT devices are required to deliver traffic initiated by other nodes in the network. Thus, it is important to optimize the flow of the traffic in the network, so that, communications are established using minimum resources consumption. SDN has emerged as a new solution to such challenges, where a controller is designated for the network to control the flow of the packets and maintain communications.

In this study, a new method is proposed to be used in the controller of an SDN. The proposed method uses DQNs to predict the reward values for a packet is forwarded to each of the nodes in the network. Information about the packet as well as the nodes in the network is provided to the DQN, so that, the node that maximizes the lifetime of the network is selected to forward the packet to it. The use of the lifetime value as the reward to train the DQN can balance the overall performance of the WSN, as extending the lifetime requires optimizing the resources consumption from the nodes. Several DQN models are implemented using FFNN, 2D-CNN and 3D-CNN, where the state of the networks and the information of the packet being forwarded are summarized in a one-dimensional vector for the FFNN, whereas these values are summarized in a three-dimensional array for the CNN models.

The results of the experiments conducted to evaluate the performance of the proposed method show that the 3D-CNN has the best performance, as it has achieved the longest lifetime. This performance is according to the use of three-dimensional average pooling filters to summarize the features detected for each node into a single value that can be used to represent the overall state of that node. This model has been able to extend the lifetime of the WSNs by 17% to 678251.6S, compared to the longest lifetime of 578122.2S using the existing state-of-art-methods. However, to ensure the extension of the WSN's lifetime, i.e. balanced resources consumption, the average number of hops required to deliver the

packets has relatively higher value of 9.81 hops per packet, compared to 8.76 in the existing methods. However, the PDR of the network has significantly increased to 85.07%.

The FFNN has achieved the shortest lifetime of 578381.6S among the proposed models, according to the limited information can be represented in the one-dimensional vector, compared to the three-dimensional arrays in the CNN models. However, this method has been able to achieve the lowest number of hops per packet, i.e. shorter paths, with 83.37% PDR. The 2D-CNN model has shown a reasonable lifetime of 638169.2S with 82.47% PDR. However, this model has shown longer paths, with an average of 12.37 hops per packet, which is longest among the evaluated models. The results show that the 2D-CNN starts to use longer paths as the energy available in the nodes start to get exhausted, up to a certain level where most of the nodes start to have critical energy levels. At that point, this model falls back to choosing the shortest paths, as most of the nodes have low energy values and the selection of longer paths can increase the consumption of the already limited resources in the network.

In future work, the RL model is going to be provided with the ability to drop packets, so that, a packet that is sent to a node which is out of reach is dropped immediately without being forwarded to any other nodes. However, to implement such modes, it is important to include the PDR in the computations of the reward value. Otherwise, the DQN is going to learn that dropping the packets returns the highest reward, as it does not consume any additional resources. The use of such a model is expected to extend the networks lifetime furthermore, as packets can be dropped immediately instead of being forwarded to nodes that cannot deliver the packet to its destination.

REFERENCES

[1] M. Kocakulak and I. Butun, "An overview of Wireless Sensor Networks towards internet of things," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, 2017, pp. 1-6.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer networks*, vol. 38, pp. 393-422, 2002.

[3] J. N. Al-Karaki and A. E. Kamal, "Routing techniques in wireless sensor networks: a survey," *IEEE wireless communications*, vol. 11, pp. 6-28, 2004.

[4] B. Rashid and M. H. Rehmani, "Applications of wireless sensor networks for urban areas: A survey," *Journal of network and computer applications*, vol. 60, pp. 192-219, 2016.

[5] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE communications surveys & tutorials*, vol. 17, pp. 2347-2376, 2015.

[6] I. Lee and K. Lee, "The Internet of Things (IoT): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, pp. 431-440, 2015.

[7] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: a survey," *Future generation computer systems*, vol. 56, pp. 684-700, 2016.

[8] F. Wortmann and K. Flüchter, "Internet of things," *Business & Information Systems Engineering*, vol. 57, pp. 221-224, 2015.

[9] W. Ejaz, M. Naeem, A. Shahid, A. Anpalagan, and M. Jo, "Efficient energy management for the internet of things in smart cities," *IEEE Communications Magazine*, vol. 55, pp. 84-91, 2017.

[10] M. S. Mahmoud and A. A. Mohamad, "A study of efficient power consumption wireless communication techniques/modules for internet of things (IoT) applications," 2016.

[11] S. Bera, S. Misra, S. K. Roy, and M. S. Obaidat, "Soft-WSN: Software-defined WSN management system for IoT applications," *IEEE Systems Journal*, vol. 12, pp. 2074-2081, 2016.

[12] A. Akbas, H. U. Yildiz, B. Tavli, and S. Uludag, "Joint optimization of transmission power level and packet size for WSN lifetime maximization," *IEEE Sensors Journal*, vol. 16, pp. 5084-5094, 2016.

[13] G. Lee, "Software defined networking-based vehicular adhoc network with fog computing," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 1202-1207.

[14] H. Farhady, H. Lee, and A. Nakao, "Software-defined networking: A survey," *Computer Networks*, vol. 81, pp. 79-95, 2015.

[15] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, 2016, pp. 258-263.

[16] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, pp. 255-260, 2015.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, p. 529, 2015.

[18] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[19] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *nature*, vol. 529, p. 484, 2016.

[20] I. N. Da Silva, D. H. Spatti, R. A. Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves, "Artificial neural networks," *Cham: Springer International Publishing*, 2017.

[21] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, 2016, pp. 3844-3852.

[22] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, *et al.*, "Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, pp. 1285-1298, 2016.

[23] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "QoS-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach," in *2016 IEEE International Conference on Services Computing (SCC)*, 2016, pp. 25-33.

[24] D. Zhang, F. R. Yu, and R. Yang, "A Machine Learning Approach for Software-Defined Vehicular Ad Hoc Networks with Trust Management," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1-6.

[25] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *arXiv preprint arXiv:1709.07080,* 2017.

[26] Z. Yang, Y. Yue, Y. Yang, Y. Peng, X. Wang, and W. Liu, "Study and application on the architecture and key technologies for IOT," in *Multimedia Technology (ICMT), 2011 International Conference on*, 2011, pp. 747-751.

[27] A. Ukil, S. Bandyoapdhyay, C. Puri, and A. Pal, "IoT healthcare analytics: The importance of anomaly detection," in *Advanced Information Networking and Applications (AINA), 2016 IEEE 30th International Conference on*, 2016, pp. 994-997.

[28] P. Govindhan, G. V. Pratap, S. Balaji, M. Gurumoorthy, and H. Khudhrathulla, "Emergency Health Transmission System via Internet," *International Journal of Engineering Science*, vol. 16508, 2018.

[29] H. Kim, E. Lee, D. Kwon, and H. Ju, "Chemical laboratory safety management service using IoT sensors and open APIs," in *Information and Communications (ICIC), 2017 International Conference on*, 2017, pp. 262-263.

[30] T. Tettamanti, I. Varga, and Z. Szalay, "Impacts of autonomous cars from a traffic engineering perspective," *Periodica Polytechnica Transportation Engineering*, vol. 44, pp. 244-250, 2016.

[31] A. Lari, F. Douma, and I. Onyiah, "Self-driving vehicles and policy implications: current status of autonomous vehicle development and minnesota policy implications," *Minn. JL Sci. & Tech.*, vol. 16, p. 735, 2015.

[32] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295,* 2016.

[33] G. Hartman, Z. Shiller, and A. Azaria, "Deep Reinforcement Learning for Time Optimal Velocity Control using Prior Knowledge," *arXiv preprint arXiv:1811.11615*, 2018.

[34] T. Lin, J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "Enabling SDN applications on software-defined infrastructure," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1-7.

[35] M.-K. Shin, K.-H. Nam, and H.-J. Kim, "Software-defined networking (SDN): A reference architecture and open APIs," in *2012 International Conference on ICT Convergence (ICTC)*, 2012, pp. 360-361.

[36] I. Z. Bholebawa and U. D. Dalal, "Performance analysis of sdn/openflow controllers: Pox versus floodlight," *Wireless Personal Communications*, vol. 98, pp. 1679-1699, 2018.

[37] R. Kandoi and M. Antikainen, "Denial-of-service attacks in OpenFlow SDN networks," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 1322-1326.

[38] A. Bianco, P. Giaccone, S. Kelki, N. M. Campos, S. Traverso, and T. Zhang, "On-the-fly traffic classification and control with a stateful SDN approach," in *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 1-6.

[39] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine learning proceedings 1994*, ed: Elsevier, 1994, pp. 157-163.

[40] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330-337.

[41] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning,* vol. 8, pp. 279-292, 1992.

[42] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602,* 2013.

[43] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, *et al.*, "Deep q-learning from demonstrations," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[44] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971,* 2015.

[45] G. Ertaş, H. Ö. Gülçür, O. Osman, O. N. Uçan, M. Tunacı, and M. Dursun, "Breast MR segmentation and lesion detection with cellular neural networks and 3D template matching," *Computers in biology and medicine,* vol. 38, pp. 116-126, 2008.

[46] P. Gorgel, N. Kilic, B. Ucan, A. Kala, and O. N. Ucan, "A Backpropagation Neural Network Approach For Ottoman Character Recognition," *Intelligent Automation & Soft Computing,* vol. 15, pp. 451-462, 2009.

[47] O. Osman, A. M. Albora, and O. N. Ucan, "Forward modeling with forced neural networks for gravity anomaly profile," *Mathematical Geology,* vol. 39, p. 593, 2007.

[48] O. Osman, A. M. Albora, and O. N. Ucan, "A new approach for residual gravity anomaly profile interpretations: Forced Neural Network (FNN)," *Annals of Geophysics,* vol. 49, 2006.

[49] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097-1105.

[50] B. Kayalibay, G. Jensen, and P. van der Smagt, "CNN-based segmentation of medical imaging data," *arXiv preprint arXiv:1701.03056,* 2017.

[51] Y. Lu, S.-C. Zhu, and Y. N. Wu, "Learning frame models using cnn filters," *arXiv preprint arXiv:1509.08379*, 2015.

[52] G. Tolias, R. Sicre, and H. Jégou, "Particular object retrieval with integral max-pooling of CNN activations," *arXiv preprint arXiv:1511.05879*, 2015.

[53] K.-L. Hua, C.-H. Hsu, S. C. Hidayati, W.-H. Cheng, and Y.-J. Chen, "Computer-aided classification of lung nodules on computed tomography images via deep learning technique," *OncoTargets and therapy*, vol. 8, 2015.

[54] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh, "Action-conditional video prediction using deep networks in atari games," in *Advances in neural information processing systems*, 2015, pp. 2863-2871.

[55] Y. Chen and E. Kulla, "A Deep Q-Network with Experience Optimization (DQN-EO) for Atari's Space Invaders," in *Workshops of the International Conference on Advanced Information Networking and Applications*, 2019, pp. 351-361.

[56] S. Yoon and K.-J. Kim, "Deep Q networks for visual fighting game AI," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 2017, pp. 306-308.

[57] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in neural information processing systems*, 2007, pp. 153-160.

[58] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 2137-2145.

[59] A. Pritzel, B. Uria, S. Srinivasan, A. P. Badia, O. Vinyals, D. Hassabis, *et al.*, "Neural episodic control," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017, pp. 2827-2836.

[60] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Advances in neural information processing systems*, 2016, pp. 4026-4034.

[61] B. Rhodes, J. Goerzen, A. Beaulne, and P. Membrey, *Foundations of Python network programming*: Springer, 2014.

[62] F. Chollet. (2018). *Keras: The python deep learning library*. Available: https://keras.io/

[63] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265-283.