

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL

PREDICTIVE ROUTING FOR MOBILE NETWORKS



ARİF BURAK DİKMEN

A THESIS OF MASTER OF SCIENCE
DEPARTMENT OF COMPUTER ENGINEERING

ADVISOR: PROF. DR. HASARI ÇELEBI

JANUARY 2025

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL

PREDICTIVE ROUTING FOR MOBILE NETWORKS



ARİF BURAK DİKMEN

A THESIS OF MASTER OF SCIENCE
DEPARTMENT OF COMPUTER ENGINEERING

ADVISOR: PROF. DR. HASARI ÇELEBI

JANUARY 2025

T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

GEZGİN AĞLAR İÇİN TAHMİNE DAYALI
YÖNLENDİRME

ARİF BURAK DİKMEN

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMAN: PROF. DR. HASARİ ÇELEBİ

OCAK 2025

MASTER of SCIENCE JURY APPROVAL FORM

A thesis submitted by Arif Burak DİKMEN, defended on 02/01/2025 before the jury formed with the 30/12/2024 date and 2024/68 numbered decision of the GTU Graduate Administration Board, has been accepted as a MASTER of SCIENCE thesis in the Department of Computer Engineering.

JURY

MEMBER

(THESIS ADVISOR) : Prof. Dr. Hasari ÇELEBİ

MEMBER

: Prof. Dr. Mehmet Kemal ÖZDEMİR

MEMBER

: Prof. Dr. Didem GÖZÜPEK

APPROVAL

Gebze Technical University Graduate Administration Board
...../...../..... date and/..... numbered decision.

SIGNATURE/SEAL



*To my
mother*

ABSTRACT

Routing in computer networks is a crucial problem. Wrong routing decisions will lead to more congestion and high congestion will cause more delays and packet loss. Computer networks are getting more complex, and networks are getting more crowded. The algorithms for the routing problem were designed decades ago. So, the design of new routing algorithms is needed. In this thesis, a deep reinforcement learning based algorithm is proposed for the routing problem. With the ability of learning agent, more intelligent routing decisions can be made. The aim of the thesis is to surpass Dijkstra shortest path algorithm-based routing in terms of number of packets delivered and have lower average delay. The proposed routing algorithm is based on Deep Double Q-Network (DDQN). To test the proposed method, a mobile network simulation environment with Python library called Simply is written. Before the tests, different types of research have been conducted to find the best neural network architecture. After finding the best neural network architecture, different types of reward functions have been tried before the test phase. After all the tests, proposed methods with two different reward functions and same architecture have been tested against Dijkstra shortest path-based routing algorithm. The test environment is mobile network topology which has 10 hosts where hosts are not moving and each host generates a packet in each timestep, 10 routers and 10 base stations. 95 GB, 45 GB and 22 GB are buffer sizes for the routers in each test setup. The advantages of proposed methods are shown in the results. Dijkstra based routing showed close results in each test but even between the proposed methods differ in terms of performance depending on the setup because of their reward function design. The proposed methods have given promising results that can be extended for usage in bigger networks. The next step of the work will be network with moving hosts and demonstration of the algorithm in the real-life network setup.

Keywords: Mobile Networks, Reinforcement Learning, Routing

ÖZET

Bilgisayar ağlarında yönlendirme çok önemli bir problemdir. Yanlış yönlendirme kararları daha fazla tıkanıklığa yol açacak ve yüksek tıkanıklık daha fazla gecikmeye ve paket kaybına neden olacaktır. Bilgisayar ağları giderek daha karmaşık hale gelmekte ve ağlar giderek daha kalabalıklaşmaktadır. Yönlendirme problemi için algoritmalar onlarca yıl önce tasarlanmıştır. Bu nedenle, yeni yönlendirme algoritmalarının tasarlanması gerekmektedir. Bu tezde, yönlendirme problemi için derin takviyeli öğrenme tabanlı bir algoritma önerilmiştir. Öğrenen ajan yeteneği ile daha akıllı yönlendirme kararları alınabilir. Tezin amacı, teslim edilen paket sayısı açısından Dijkstra en kısa yol algoritması tabanlı yönlendirmeyi aşmak ve daha düşük ortalama gecikmeye sahip olmaktır. Önerilen yönlendirme algoritması Derin Çift Q-Ağna dayanmaktadır. Sonuçları test etmek için Simply adlı Python kütüphanesi ile bir mobil ağ simülasyon ortamı yazılmıştır. Test sonuçlarından önce, en iyi sinir ağı mimarisini bulmak için farklı türde araştırmalar yapılmıştır. En iyi sinir ağı mimarisi bulunduğundan sonra, test aşamasından önce farklı türde ödül fonksiyonları denenmiştir. Tüm testlerden sonra, iki farklı ödül fonksiyonuna ve aynı mimariye sahip önerilen yöntemler Dijkstra en kısa yol tabanlı yönlendirme algoritmasına karşı test edilmiştir. Test ortamı, cihazların hareket etmediği ve cihazın her zaman adımında bir paket ürettiği 10 cihaz, 10 yönlendirici ve 10 baz istasyonundan oluşan mobil ağ topolojisidir. 95 GB, 45 GB ve 22 GB her bir test kurulumundaki yönlendiriciler için tampon boyutlarıdır. Önerilen yöntemlerin avantajları sonuçlarda gösterilmiştir. Dijkstra tabanlı yönlendirme her testte yakın sonuçlar göstermiştir, ancak önerilen yöntemler kendi aralarında bile ödül fonksiyonu tasarımları nedeniyle kurulumla bağlı olarak performans açısından farklılık göstermektedir. Önerilen yöntemler, daha büyük ağlarda kullanılmak üzere genişletilebilecek umut verici sonuçlar vermiştir. Çalışmanın bir sonraki adımı hareketli cihazların olduğu ağ ve algoritmanın gerçek hayattaki ağ kurulumunda gösterilmesi olacaktır.

Anahtar Kelimeler: Gezgin Ağlar, Pekiştirmeli Öğrenme, Yönlendirme

ACKNOWLEDGEMENTS

First, I would like to thank my thesis supervisor Prof. Dr. Hasari ÇELEBI. It was an honor to work with you, professor. I would like to thank my mother and father for bringing me to this day. Every moment I spent with you returned +1 as a reward.



TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT	vi
ÖZET	vii
ACKNOWLEDGEMENTS	viii
TABLE OF CONTENTS	ix
LIST OF SYMBOLS AND ABBREVIATIONS	x
LIST OF FIGURES	xi
LIST OF TABLES	xii
1. INTRODUCTION	1
1.1. Literature Review	1
1.2. Reinforcement Learning	5
1.2.1. Q-Learning	7
1.2.2. Exploration and Exploitation	7
1.3. Deep Learning	8
1.4. Deep Reinforcement Learning	8
1.4.1. Deep Q-Network	8
1.4.2. Double Deep Q-Network	9
1.5. Mobile Networks	10
2. PURPOSE OF THE STUDY	12
3. METHODOLOGY	13
3.1. Double DQN Architecture and Parameters	13
3.2. Exploration Strategy	14
3.3. State Representation	14
3.4. Reward Function	15
4. RESULTS AND DISCUSSION	17
4.1. Environment	17
4.2. Experiments	18
4.3. Discussion	32
5. CONCLUSION	33
REFERENCES	34
BIOGRAPHY	38
PUBLICATIONS AND PRESENTATIONS FROM THE THESIS	39

LIST OF SYMBOLS AND ABBREVIATIONS

r	: Reward
γ	: Discount Factor
a	: Action
π	: Policy
s	: State
s'	: Next State
t	: Timestep
R_t	: Return
G_t	: Return
ϵ	: Epsilon
T	: Final Timestep
e_t	: Experience Tuple
DQN	: Deep Q-Network
DDQN	: Double Deep Q-Network
OSPF	: Open Shortest Path First
QoS	: Quality of Service
NAR-NN	: Nonlinear Autoregressive Neural Network
MILP	: Mixed Integer Linear Programming
RIP	: Routing Information Protocol
IS-IS	: Intermediate System to Intermediate System
GB	: Gigabyte
M	: Meter
KM	: Kilometer
Mbps	: Megabit per second
Gbps	: Gigabit per second

LIST OF FIGURES

	<u>Page</u>
Figure 1.1: Wireless network.	10
Figure 2.1: General flow of reinforcement learning.	12
Figure 4.1: Wireless network topology.	17
Figure 4.2: Average collected rewards for every 100 episodes with four different neural network architectures.	18
Figure 4.3: Average number of packets delivered every 100 episodes for four different neural network architectures.	19
Figure 4.4: Collected rewards for two neural networks with different hidden layer numbers.	20
Figure 4.5: Number of packets delivered for two neural networks with different hidden layer numbers.	20
Figure 4.6: Number of packets delivered for six different neural networks architectures.	21
Figure 4.7: Number of packets delivered for different reward functions for the neural network with 96-128-96-128 neurons in hidden layers.	22
Figure 4.8: Number of packets delivered for reward function 1 and reward function 3.	23
Figure 4.9: Average number of packets delivered on 95 GB buffer.	24
Figure 4.10: Average packet delay on 95 GB buffer.	25
Figure 4.11: Average path length on 95 GB buffer.	25
Figure 4.12: Average packets delivered on 45 GB buffer.	26
Figure 4.13: Average packet delay on 45 GB buffer.	27
Figure 4.14: Average path length on 45 GB buffer.	27
Figure 4.15: Average packets delivered on 22 GB buffer.	28
Figure 4.16: Average packet delay on 22 GB buffer.	29
Figure 4.17: Average path length on 22 GB buffer.	29

LIST OF TABLES

	<u>Page</u>
Table 1.1: Characteristics of wireless networks.	11
Table 3.1: Neural network architectures.	13
Table 3.2: Parameters of ϵ -greedy exploration strategy.	14
Table 3.3: Reward function 1.	15
Table 3.4: Reward function 2.	16
Table 3.5: Reward function 3.	16
Table 4.1: Results for different neural network architectures.	21
Table 4.2: Results for different reward functions.	22
Table 4.3: Results for reward function 1 and reward function 3.	23
Table 4.4: Experiment results on 95 GB buffer.	30
Table 4.5: Experiment results on 45 GB buffer.	30
Table 4.6: Experiment results on 22 GB buffer.	31

1. INTRODUCTION

As of July 2024, 5.52 billion people are Internet users (Petrosyan, 2024) and this means that around 67% of the world population are Internet users (United Nations Population Fund, 2024). Various electronic devices like computers, smartphones, smartwatches, cars, and home appliances can connect to the Internet. The latest information about the average number of devices that can connect to the Internet per capita was from 2018. Globally, the number of devices connected to the Internet per capita was 2.4 back in 2018, and the prediction for 2023 was 3.6. The numbers are much higher in North America and Europe (Taylor, 2023).

Open Shortest Path First (OSPF) routing protocol uses Dijkstra shortest path algorithm, and it was designed in the late 1980s (Moy, 1989). In cases where particular routers get heavily congested rather than the rest of the network, traditional routing protocols fall short in solving the congestion because they never learn from past experiences. This will lead to higher congestion and Quality of Service (QoS) problems like losing packets and higher delay (Tang et al., 2017).

Internet traffic rises quickly due to the massive amount of Internet users and new concepts like the Internet of Things and augmented reality. As a result, new ways of network routing must be found (Yao et al., 2019). Machine learning-based routing could solve the routing problem dynamically by learning from previous experience. From the previous Internet traffic states, it can decide which routing action can be taken (Valadarsky et al., 2017). It is called predictive routing.

1.1. Literature Review

Various research has been done on machine learning based routing protocols in different types of networks. To solve the routing problem with machine learning based algorithms, 5 different subtopics exist. These techniques are supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning and evolutionary computing algorithms. Supervised learning can be applied when labelled data exists which is a costly process. Unsupervised learning does not need any labelled data. Unsupervised learning algorithms extract hidden patterns by itself, but the

success rate is lower than supervised learning. Semi-supervised learning is a mixture of the first two machine learning techniques. It uses both labelled and unlabelled data for higher accuracy. Reinforcement learning doesn't need any dataset. The agent learns by taking actions with the environment and tries to maximize the reward in the learning process. Evolutionary computing algorithms are used for NP-hard problems. It uses discovery-based methodology like Genetic Algorithms (Samadi and Seitz, 2022).

Valadarsky et al. (2017) proposed two different machine learning based routing algorithms. One of them is supervised learning routing and the other one is reinforcement learning based routing. They modelled the traffic by $n \times n$ matrix named demand matrix where i and j means source and destination respectively. In their supervised learning approach, their aim is to predict the next demand matrix by previous demand matrices. After this prediction, optimal routing for predicted demand matrix. Their learning architecture is based on deep neural networks. Three different deep neural networks have been used. In the learning process, 5, 10 and 20 past demand matrices have been given as input respectively and a single demand matrix which is the prediction taken as output. The architectures they used are fully connected network with 3 layers, convolutional neural network (LeCun et al., 1998) with 4 layers and lastly, Nonlinear Autoregressive Neural Network (NAR-NN) (Chow and Leung, 1996) with 4 layers have been used. It's been examined that only NAR-NN showed a good performance in the matrices that have been generated by deterministic gravity model (Roughan et al., 2002) but all three architectures were unable to predict the next traffic matrix from deterministic bi-modal model (Medina et al., 2002) generated traffic matrices. Proposed deep reinforcement learning based routing algorithm also showed promising results for the routing problem (Valadarsky et al., 2017).

Reis et al. (2019) proposed supervised deep learning-based model to solve the routing problem. In order to collect the dataset, researchers simulated a network, and the routing problem has been solved by Mixed Integer Linear Programming (MILP). But the problem has been demonstrated to be too difficult to resolve in a sensible length of time (Fortz and Thorup, 2000; Kodialam and Lakshman, 2000). That is why this method has been used in offline training part of the deep learning model.

Two main goals for network routing have been defined. One of them is maximum link utilization and another congestion measure metric defined in (Fortz and Thorup, 2000). The routing method has also employed a post processing algorithm to make the output

a valid path. The deep learning model takes flow features as input and gives the output as binary vectors where these vectors represent whether if a flow routed via a link. The neural network model is constructed by dense layers and all the hidden layers have ReLu as activation function. To gather a valid routing path, a heuristic algorithm has been suggested. This post processing algorithm starts from the source node to the destination node with choosing the highest output node. If the cycle happens, the Dijkstra algorithm finds a valid path.

The proposed algorithm has been evaluated in two different topologies and datasets. One of them is GEANT and the other one is Abilene network (Ruotian, 2024). Both include datasets and topologies. Results showed that the proposed deep learning-based routing method outperformed shortest path significantly and gives nearly as good as results as MILP and even surpasses it in some cases because deep learning-based method might have the ability of predicting the next flows (Reis et al., 2019).

Tang et al. (2017) proposed convolutional neural network-based algorithm for routing problems. The proposed algorithm has the ability of learn online from data coming in real-time. This ability leads to lesser congestion. The proposed routing algorithm can give intelligent routing decisions unlike classical routing algorithms like OSPF and prevent the congestion. The convolutional neural network architecture consists of two convolution and two pooling layers for feature extraction and finally, the model employs fully connected layer for the to the classification part. To give an input to model, a three-dimensional matrix is constructed. Each channel of this matrix represents network features like the remaining buffer size. The columns of the channels represent routers and rows represent time intervals. Output is the congestion represented as a two-dimensional vector. And the final output is N-dimensional vector.

Before the routing algorithm starts, the proposed method has an initial phase. In the initial phase, all routers in the network find all the possible paths to all the destinations. After these calculations, all the paths are stored in a minimum priority queue. When the initial phase is done, the running phase starts. Average delay, loss rate for each packet, packet generation rate, waiting queue sizes and whether congestion happened information are recorded. This process is made for training part of the deep learning model. For all the routing strategies recorded before, a neural network decides if congestion will happen or not. If congestion happens, the model continues assessing other routing strategies until the model finds a routing strategy that will not cause

congestion. To make neural networks more accurate, online training part exists. In certain time intervals, neural networks got trained by data collected from the network. The proposed routing algorithm evaluated against OSPF, Routing Information Protocol (RIP) and Intermediate System to Intermediate System (IS-IS) routing protocols in 3x3 wireless network and it has lesser average delay and packet loss rate than the other routing protocols (Tang et al., 2017).

Manfredi et al. (2021) proposed a deep reinforcement learning based routing algorithm in wireless networks. The aim of the proposed routing algorithm is to work on various types of network conditions. The input of the neural network is the state features. State features consist of packet features, local device features and aggregated neighbor features. Packet features are a packet's time to live value and packet's position in router's queue. For the local device features, estimated distance between current router and destination router, queue length of the router, queue length of the router just for the packets going to a certain destination and degree of router. And lastly, aggregated neighbor features formed by mean, maximum and minimum values of local device features of all the neighbor routers of current router.

Action features are constructed by local device features of the next possible hop for the router. The reward function has three different values. If the packet is delivered, 0 is given as a reward. If a transition happens, -1 is given as a reward and if a packet got dropped $1/(1-\gamma)$ is given as reward and γ has a value between 0 and 1. Researchers proposed packet-centric approach in routing where every packet decides its path instead of routers when the packet is queued at the top of the router's buffer. The deep neural network (Bengio, 2009) model consists of 4 layers. In the first layer, action and state features are given as input. $F = |f_s(\cdot)| + |f_a(\cdot)|$ where $f_s(\cdot)$ defines state features and $f_a(\cdot)$ defines action features. The second layer has $10F$ neurons and the third layer have $F/2$ neurons, meanwhile the last layer is the output layer. Q-value for the action and state features is predicted by the neural network as the output. To find the next hop router for each packet, action and state features are given into the neural network. The list of the Q-values can be used for the action selection.

Two different network topologies have been used for the evaluation. One of them is square grid lattice and the other one is geometric random graph. The proposed routing algorithm has been evaluated against shortest path (Bellman, 1958) and backpressure routing (Tassiulas and Ephremides 1990). Deep reinforcement learning based routing

agents trained in network where device size is 64 on static lattice and low traffic, static random graph and high traffic, static lattice and high traffic, dynamic lattice and high traffic, delay tolerant lattice and high traffic and delay tolerant random graph and high traffic. In the evaluation part, the network size varies between 9 and 100.

This work has shown that the proposed method worked as good as the shortest path in static lattice and low traffic in terms of packet delivery. In the static lattice and high traffic scenario, proposed method outperformed shortest path in terms of packet delivery rate and outperformed backpressure routing in terms of delay. When lattice becomes dynamic, and the traffic is high, proposed routing method outperforms both other routing methods. In the delay tolerant lattice where the traffic is high, backpressure routing outperforms both the proposed method and shortest path in terms of packet delivery because backpressure routing has the ability of choosing which packet to send in the router's buffer instead of sending the packet at the top of the queue. For the random geometric graph, the first evaluation made in static random graph where traffic is high. The agents in proposed algorithm delivered all packets in the network like shortest path did when network sizes are larger but when network sizes are smaller deep reinforcement learning agents except the agent trained in delay tolerant lattice and where the traffic is high outperformed the other routing methods. When the network is delay tolerant and traffic is high, specifically trained on delay tolerant lattice and random graph agents were able to deliver all the packets but when network sizes get smaller, compared to the backpressure routing and shortest path routing, it performed worse than both (Manfredi et al., 2021).

1.2. Reinforcement Learning

Reinforcement learning is a subfield of machine learning. The reinforcement learning agent is the learner of reinforcement learning system. A reinforcement learning agent learns with trial and error while trying to collect maximum reward. From now on, reinforcement learning agent will be called just agent. The learning process of the agent happens in few steps. An agent observes the current environment, and this observation is called state s . After the observation, the agent takes an action a and it gets a reward value r as feedback of the action (Morales and Zaragoza, 2012).

To fully grasp reinforcement learning, some concepts need to be defined. Reinforcement learning concepts are policy, environment, state, action and transition models. A policy is necessary for the agent to choose which action to take to maximize the total reward in state s . The environment is outside of the agent that the agent interacts. States represent information about the environment. A state value might be a numerical value or alphabetical value. A special type of state that means the end of the episode is called terminal state. Agent interacts with the environment by taking actions. Actions define what an agent can do while interacting with the environment. When an agent takes an action a in state s , a transition happens into next state s' and gets a reward value R . The transition function has a , s and s' as elements and as an output it gives the probability of this transition (Naeem et al., 2020).

The selection of the action a is made by a policy π . With the reinforcement learning, a policy changes over the time by agent interacting with the environment to collect the maximum reward. In every time step t , agent interacts with the environment and gets a reward single as a single number. This feedback is immediate, but the agent's main goal is to maximize the total rewards. Maximizing the total reward should achieve the main goal that why designing a suitable reward function for the goal is crucial. Agent needs to learn how to maximize all the collected rewards in the long run. Summation of all rewards collected is called return R_t where t is timestep and T is the final timestep that defined in equation (1.1) (Sutton and Barto, 2018).

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \dots + r_T \quad (1.1)$$

As a formal definition, the final state T is called terminal state. The sequence from the first-time step t to terminal state is called episodes. Not all the reinforcement learning tasks end by terminal states. Some tasks are infinite, and these tasks are called continuing tasks. The equation (1.1) does not fit for continuing tasks. Because since there is no definitive terminal state, the maximum reward can go to infinity. To define how important future rewards are, discount rate γ is introduced where it can take any value between 0 and 1. The definition of discounted return is defined in equation (1.2) (Sutton and Barto, 2018).

$$R_t = r_{t+1} + \gamma * r_{t+2} + \gamma^2 * r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k * r_{t+k+1} \quad (1.2)$$

If the γ is 0, only immediate reward is considered. When γ is getting closer to 1, the importance of future rewards become more important (Sutton and Barto, 2018).

As stated above, the policy selects an action for the agent. When an agent takes action a when in state s and policy determines the remaining actions, the Q-function is the discounted return of this process given in equation (1.3) where G_t denotes return (Morales, 2020).

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t \mid S_t = s, A_t = a] \quad (1.3)$$

1.2.1. Q-Learning

Q-Learning first introduced by Watkins (1989). It is a very effective reinforcement learning algorithm to achieve optimal policy without strictly following a certain policy. To find the optimal policy, better estimations for Q-function needs to be made. equation (1.4) represents Q-function update rule where α is learning rate (Sutton and Barto, 2018).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \quad (1.4)$$

In equation (1.4) the part $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ represents the value of the current reward plus the discounted return that is maximized started from next state. This part is called target value. The previous Q-function is subtracted from the target value to find the error. If there is no previous Q-function value, it is initiated as zero. The error is multiplied with the α . The total value is added to the previous Q-function value and the new Q-function is gathered. This process continues until the terminal state is reached (Morales, 2020).

1.2.2. Exploration and Exploitation

Agent needs to sufficiently explore which action to take in the environment to take the best possible actions. Otherwise, the agent will always pick actions with higher Q-values and will never know whether there is better action in its current state. To find a balance between exploration and exploitation ϵ -greedy is used. This exploration strategy uses ϵ value to decide when to explore and when to exploit. The computer

generates a random value and if the generated value is higher than ϵ , agent makes exploitation. Otherwise, it makes exploration and takes a random action (Szepesvári, 2022).

1.3. Deep Learning

Traditional machine learning algorithms require significant knowledge about the domain to be used in feature extraction tasks because they have restricted ability of using raw data as input. With deep learning, it is possible to extract representations without expert information (LeCun et al., 2015).

Deep learning architectures are formed by neural networks containing high number of layers. The neural network takes an input and processes the input in its hidden layers. Each layer consists of n number of neurons and these neurons have activation functions inside them. The neurons in a layer connected with other neurons residing in neighbor layers. When a value is processed in a neural network, it is multiplied by another value called weight while the input value is going between a node from another node in the next layer. All the inputs coming to a certain node from different nodes are summed and the result is given into activation function (Shrestha and Mahmood, 2019).

1.4. Deep Reinforcement Learning

When a deep neural network is used instead of approximating Q-function with vanilla reinforcement learning methods is called deep reinforcement learning. As an input of deep neural network, state of the environment is given, and approximation of Q-function is taken as output (Li, 2023).

1.4.1. Deep Q-Network

Mnih et al. (2015) suggested a reinforcement learning method that predicts Q-function by using deep convolutional neural network. But it is accepted that using a neural network can cause stability issues when predicting the Q-function (Tsitsiklis and Van Roy, 1996). This stability issue happens because of just minor changes in Q-function might have a big impact on the policy and consequently, distribution of the data. To overcome this problem, researchers suggested some methods. Changes in distribution of data are adjusted via experience replay (McClelland et al., 1995; O'Neill et al.,

2010; Lin, 1992). The solution for stability issues has been solved via Q-iteration (Riedmiller, 2005) but it is unable to work efficiently if neural network is large. The proposed method's experience replay part forms a list containing $e_t = (s_t, a_t, r_t, s_{t+1})$ where e_t is experience of an agent, s_t is current state, a_t is action and r_t is reward in a time step t and s_{t+1} is next state. To predict the Q-function, deep convolutional neural network has been employed. The recorded experiences are selected randomly in the training part and the loss function is given in equation (1.5) (Mnih et al., 2015).

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; Q_i^-) - Q(s, a; Q_i) \right)^2 \right] \quad (1.5)$$

The update equation for the predicted Q-function of the Deep Q-Network is given in equation (1.6) In the equation Q-function is predicted by a neural network θ_t^- instead of using the equation (1.4) unlike vanilla Q-Learning (Van Hasselt et al., 2016).

$$Y_t^{DQN} \equiv R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a; \theta_t^-) \quad (1.6)$$

1.4.2. Double Deep Q-Network

It has been shown in Thrun and Schwartz (2014), Q-function predictions when using an approximator will be noisy. Noise will prevent more precise predictions, and the values will be overestimated. Because of the overestimation problem, achieving optimal policy might not be possible. Double Q-Network (Hasselt, 2010) proposed that distinguishing deciding which action to take and assessing the action. Deep learning-based method called Double Deep Q-Network (DDQN) has been proposed by Van Hasselt et al. (2016).

DDQN employs two different neural networks, but the architecture of these networks is the same. One of them is used for evaluation and the other one is selection. Unlike Deep Q-Network (DQN), two different neural networks have been used to overcome overestimation problems. The weights of network used for action selection are copied to other network in every n number of episodes. Update equation of DDQN is given in equation (1.7). Unlike DQN's update equation given in equation (1.6), action selection part of equation (1.7) returns which action to take instead of the maximum Q value of it. This prediction is made by neural network θ_t and the prediction of the Q-

function is made by another neural network θ_t^- . This is a more stable approach than DQN (Van Hasselt et al., 2016).

$$Y_t^{Double\ DQN} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}} Q(S_{t+1}, a; \theta_t), \theta_t^-) \quad (1.7)$$

1.5. Mobile Networks

Early demonstrations for wireless transmission have been made in the late 1800s. In 1901, Guglielmo Marconi showed a transmission between continents without wired connection. This technology advancements lead to technologies like television and cellular networks (Miao et al., 2016).

Wireless networks consist of some components. These components are wireless hosts, wireless links and base stations. Wireless hosts are the devices that are connected to the Internet and run application layer software. These devices can be a laptop, gaming console or a smart phone. Unlike wired systems, these devices might not be stationary. Wireless links mean the links between different wireless hosts or wireless hosts and base station. The links can have different characteristics, varying transmission rates and ranges. And lastly, base stations are in the middle of the data transmission between wireless hosts (Kurose and Ross, 2021).

An example topology for a wireless network is given in **Figure 1.1** and **Table 1.1** shows the features of different wireless network types.

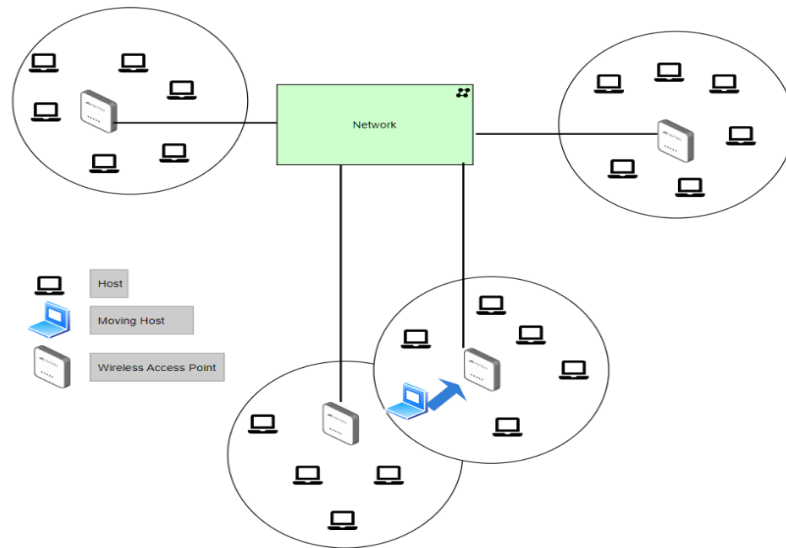


Figure 1.1: Wireless network (Kurose and Ross, 2021).

Table 1.1: Characteristics of wireless networks (Kurose and Rose, 2021).

Wireless Network	Range	Speed
Bluetooth	10 M-30 M	2 Mbps
802.11b	10 M-30 M <	11 Mbps
802.11g	10 M-30 M <	54 Mbps
801.11 af, ah	200 M-4 KM	600 Mbps
802.11n	50 M-200 M	600 Mbps
802.11ac	50 M-200 M	3.5 Gbps
802.11ax	50 M-200 M	14 Gbps
4G LTE	4 KM-15 KM	600 Mbps
5G	200 M-4 KM	10 Gbps

2. PURPOSE OF THE STUDY

In this study, the aim is to replace traditional heuristic routing algorithms with more intelligent deep reinforcement learning based routing algorithms. Unlike traditional networks, current networks have devices that are connected to internet wirelessly. To maintain the health of the network and avoid congestion, more dynamic ways of routing instead of traditional heuristics are needed. Machine learning techniques can capture the characteristics of the network and can predict the future states without explicitly told. These techniques can be used for different types of objectives like energy saving and avoiding network congestion. Finding datasets for the environment that researchers work on is another struggle. Reinforcement learning methods require no dataset, and it can be used as routing algorithm logic depending on the network (Mammeri, 2019). The general flow of reinforcement learning is given in **Figure 2.1**.

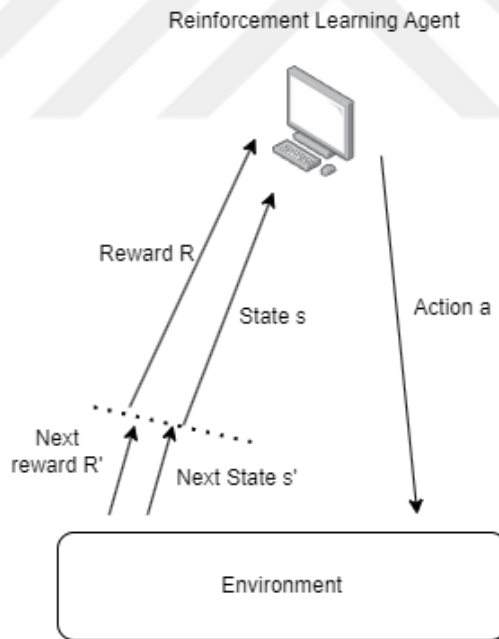


Figure 2.1: General flow of reinforcement learning (Mammeri, 2019).

The main objective of this study is to implement routing logic based on DDQN for mobile networks to avoid congestion, deliver more packets and have lesser average delay per packet compared to Dijkstra algorithm.

3. METHODOLOGY

In this section, the general architecture of DDQN, the exploration strategy, reward function and state representation have been explained.

3.1. Double DQN Architecture and Parameters

In DDQN, multiple neural network architectures are experimented with varying sizes of neuron and hidden layer numbers. The architecture of neural networks has been given in **Table 3.1**.

Table 3.1: Neural network architectures.

Neurons in Input Layer	Neurons in Output Layer	Number of Hidden Layers	Number of neurons in hidden layers
12	10	3	16-32-16
12	10	3	32-64-32
12	10	3	64-96-64
12	10	3	96-128-96
12	10	2	96-96
12	10	4	96-128-96-128

The number of neurons in input and output layers correspond to state size and action size. In every neuron, ReLU (Glorot et al., 2011) activation function has been used except the output layer which uses linear activation function. Every neural network architecture's loss function is Huber loss (Huber, 1992) and optimizer is RMSProp (Tieleman and Hinton, 2017). For the training part, experience replay (Lin, 1992) mechanism has been used. In every time step t , state s , action a , reward r , next state

s' and done flag is recorded. And these experiences are collected in agent's memory which can record 20000 experiences. Discount rate γ is 0,95 in every experiment and batch size 128. In every 10 episodes, parameters of the main neural network are copied to other network.

3.2. Exploration Strategy

The exploration strategy is ϵ -greedy (Sutton and Barto, 2018) where agent will start exploring first and in the latter episodes, the agent start exploiting. The each parameter of exploration strategy for both training and test is given in **Table 3.2**. In the training phase, starting ϵ value is 1. In order to make sure that agent starts exploitation after enough exploration, there is ϵ -decay value. The new ϵ is calculated every time in the end of each episode. New ϵ value is calculated as multiplying ϵ with ϵ -decay. But ϵ value can not be lower than ϵ -minimum value. In the test phase, ϵ value is declared zero because full exploitation is wanted whether to check if agent is learned.

Table 3.2: Parameters of ϵ -greedy exploration strategy.

Parameter	Value (Training Phase)	Value (Test Phase)
ϵ	1	0
ϵ -decay	0.995	0.995
ϵ -minimum	0.01	0.01

3.3. State Representation

State of the environment is also the input for the neural network. When designing the state representation, the main goal was capturing all of the important features that can generalize in different scenarios meanwhile excluding unnecessary features. All of the remaining router buffer sizes starting from router ID 1 to 10 written in state representation with min-max normalization. If the current router has no connection to any router from 1 to 10, value of that index is zero. The aim of the normalization is

agent to choose the least congested path whether the specific value of remaining buffer size. Rest of the values written in state representation are IDs of current and destination routers.

3.4. Reward Function

Designing the right reward function is crucial to teaching the agent what to learn. In this study, the main goal is to deliver more packets with lesser delay and avoid congested paths. To ensure the packets are delivered to their destination, a positive reward is given when packets arrive at their destination. While packets are traversing in the network, they might stick at infinite loops to maximize the total reward. Visited routers for each packet are recorded to check if a router has already been visited or not. If the router is visited for that packet, a negative reward for that action is given. Packets getting dropped is not a wanted situation. Again, a negative reward is given if packets get dropped when the time-to-live value is exceeded, or the agent chooses a router with not enough remaining buffer. To avoid more congested nodes, a negative reward value for each action is given inversely proportional.

In the experiments, three different reward function designs have been used. The reward function 1 given in **Table 3.3** is the main reward function of the reinforcement learning system.

Table 3.3: Reward function 1.

Case	Reward Value
The packet arrived at the destination router.	+10
Time to live value is exceeded for packet.	-10
If chosen router has not enough remaining buffer.	-10
If router is visited for the packet.	-2
Agent's reward for action if packet hasn't dropped or packet hasn't already visited the router.	$1 / \text{-(remaining buffer size in Kilobyte)}$

The reward function 2 and reward function 3 given in **Table 3.4** and **Table 3.5** are also explored to see whether better results will be gathered. In reward function 2, a less penalizing reward for visiting already visited routers is applied to check if already visited routers will be more helpful or not in some network traffic cases. Another proposal made in reward function 3 is to observe how the number of packets delivered will change by giving lesser positive rewards for packets reaching the destination.

Table 3.4: Reward function 2.

Case	Reward Value
The packet arrived at the destination router.	+10
Time to live value is exceeded for packet.	-10
If chosen router has not enough remaining buffer.	-10
If router is visited for the packet.	-1
Agent's reward for action if packet hasn't dropped or packet hasn't already visited the router.	$1 / \text{-(remaining buffer size in Kilobyte)}$

Table 3.5: Reward function 3.

Case	Reward Value
The packet arrived at the destination router.	+5
Time to live value is exceeded for packet.	-10
If chosen router has not enough remaining buffer.	-10
If router is visited for the packet.	-2
Agent's reward for action if packet hasn't dropped or packet hasn't already visited the router.	$1 / \text{-(remaining buffer size in Kilobyte)}$

4. RESULTS AND DISCUSSION

In this section, experiments have been shown for each setup in the simulation environment. Python is used for all the code for routing logic and simulation. The neural networks and whole deep learning part written with Keras. For the simulation part SimPy (Scherfke et al., 2020) has been used. Multiple experiments have been done with different neural network architectures and reward functions. After finding the best architecture, different reward functions have been applied and the best two reward functions have been used in deep reinforcement learning based routing methods.

The proposed method could not be evaluated against other proposed machine learning based routing methods in the literature used for routing problem because the proposed algorithms are suitable for their own network environment and the performance metrics of each work are varying. To compare the performance of the proposed algorithm, it has been decided to make a comparison with Dijkstra algorithm (Dijkstra, 1959).

4.1. Environment

The network is a wireless network, and the backbone architecture has 10 routers. The backbone network topology is embraced from (Guck et al., 2017). The wireless network has 10 host devices and 10 base stations. All hosts in the network considered as stationary in all experiments. When simulation started, all the hosts generate one packet in every time step t . Each packet size is fixed which is 64 Kilobyte. The wireless network topology is given in **Figure 4.1**.

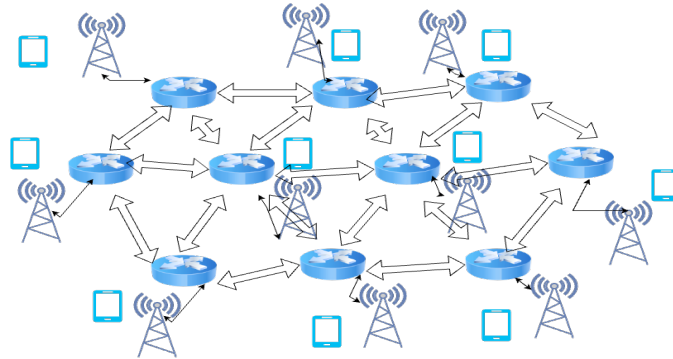


Figure 4.1: Wireless network topology.

4.2. Experiments

Experiments with different neural network architectures shown in **Table 3.1** have been conducted. Average total rewards and packets collected are given as bar charts. Each episode takes 200 timesteps and the network continues to flow after the episodes end. Router buffer size is taken 95 Gigabyte (GB) in all training experiments and reward function 1 that is given in **Table 3.3** is used for neural network experiments unless it is stated.

In **Figure 4.2**, it is natural to expect lesser rewards because it is mostly exploring the environment with random decisions in earlier episodes. The collected rewards for every 100 episodes got better because the agent has already explored the environment and learned the right routing decisions. In latter episodes, it has been mostly exploitation where the true test for the agent begins. The network with 16-32-16 neurons failed to achieve as good performance as the other neural networks. Meanwhile, neural networks with 32-64-32 and 64-96-64 neurons managed to surpass the network with the highest neurons in the latest episodes.

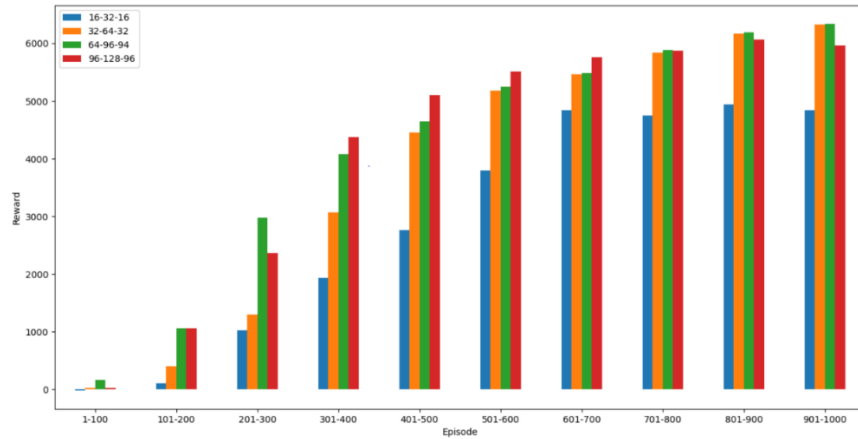


Figure 4.2: Average collected rewards for every 100 episodes with four different neural network architectures.

It has been seen in **Figure 4.3** that the number of packets delivered is correlated with total collected rewards. This shows that proposed reward function design is the right function for the routing problem.

Each model learned how to deliver packets, but both **Figure 4.2** and **Figure 4.3** show that more neurons can learn the complexity of the routing problem better than fewer

neurons, collecting more rewards and delivering more packets. Even if the performance in first 100 episodes is mostly even in all neural networks more complex neural networks outperformed where agents start exploiting. The network with 96-128-96 neurons slightly gave better results than the network with 64-96-64 neurons. It has been observed that when the number of neurons increases, the performance also gets better.

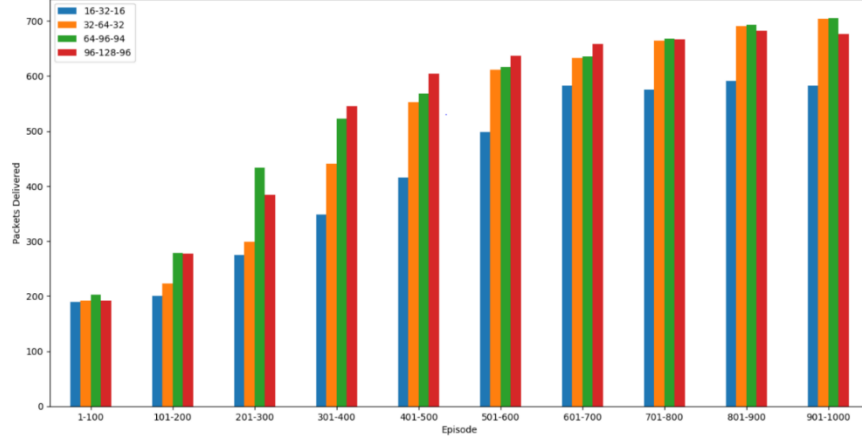


Figure 4.3: Average number of packets delivered every 100 episodes for four different neural network architectures.

The upcoming experiments will be focused on the results with varying hidden layer numbers.

The results shown in **Figure 4.4** points the same conclusion to the experiments done with varying neural network complexity. The most complex neural network of this work collected more rewards than the network with the least number of hidden layers. Especially in the second half of the simulation where the mobile network gets more crowded with packets, the most complex network managed to collect way more rewards than the other neural network.

In **Figure 4.5**, delivered packets have been shown for each neural network model. It is very correlated with the collected rewards shown in **Figure 4.4**. In the first 600 episodes, both models showed improving results, but the less complex model had decreasing result in the episodes between 701 and 800. Meanwhile the more complex model got better in every 100 episodes. Again, it has been seen that more complex networks are more suitable for the routing problem.

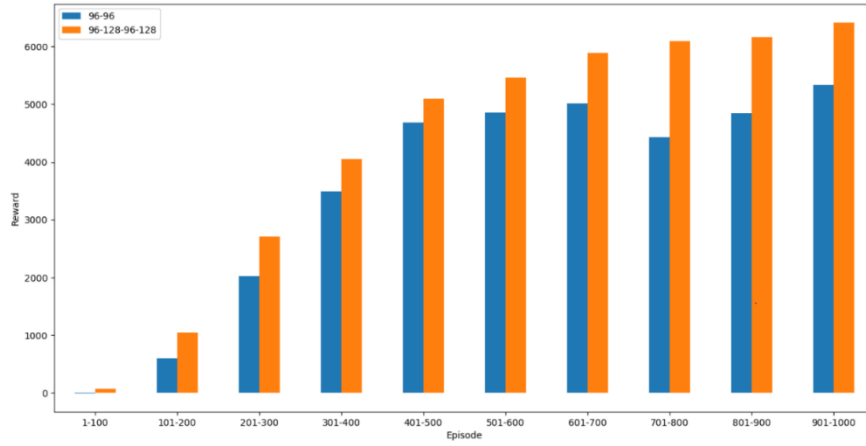


Figure 4.4: Collected rewards for two neural networks with different hidden layer numbers.

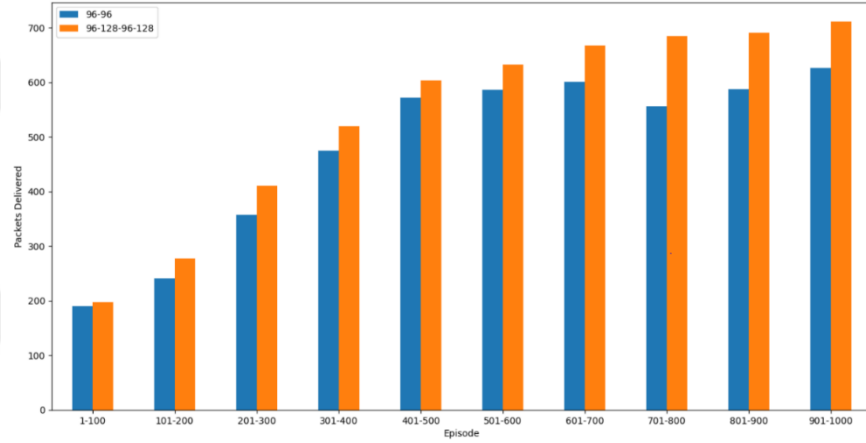


Figure 4.5: Number of packets delivered for two neural networks with different hidden layer numbers.

Figure 4.6 compared all the six models with how many packets they delivered in every 100 episodes and the exact results are shown in **Table 4.1**.

It has been decided that the network with most of the parameters (96-128-96-128) captures patterns of the environment better than the other networks. Adding more neurons and more hidden layers worked well. But it has been examined that the results have not changed so much with adding more complexity in some experiments. It has been decided that there is no need to add more neurons or hidden layers.

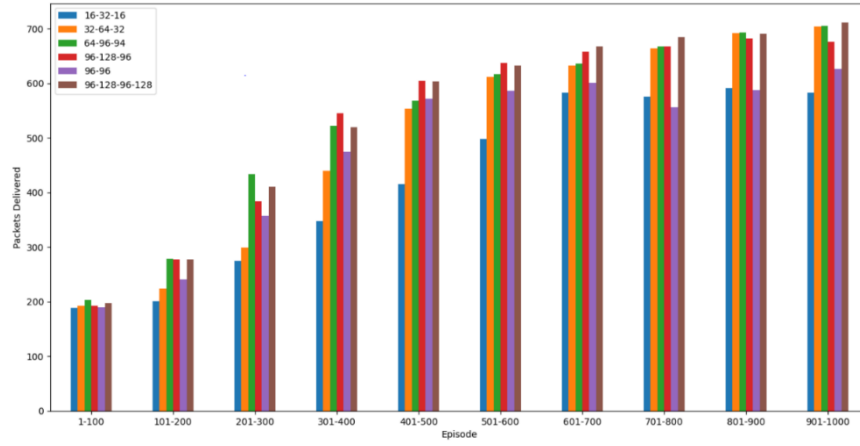


Figure 4.6: Number of packets delivered for six different neural networks architectures.

Table 4.1: Results for different neural network architectures.

Neural Network Architecture	Total Packets Delivered After 1000 Episodes	Average Rewards Collected
16-32-16	425776	2896.23
32-64-32	500929	3820.77
64-96-94	532213	4206.19
96-128-96	532226	4207.36
96-96	478942	3527.43
96-128-96-128	539269	4298.98

Now, different reward functions will be explored to see if there will be performance changes. The best performed model has been used for the upcoming tests. The results for the different reward functions are given in **Figure 4.7** and **Table 4.2**.

Figure 4.7 shows the delivered packets for every 100 episodes for each reward function. This time, direct comparison by the collected rewards is not possible due to reward function 3's lesser reward for the successful packet delivery. Also reward function 2 has more forgiving design for visiting already visited routers. Only successful packet deliveries are given.

It has been seen in **Table 4.2** that reward function 3 outperformed the other reward functions by a small margin. The performance of reward function 1 and reward

function 3 is identical in the latter episodes. It has been decided to train both reward functions for 2000 episodes to see the performance of them in the long run. **Figure 4.8** shows the comparison of reward function 1 and reward function 3 as bar chart and **Table 4.3** shows the exact results.

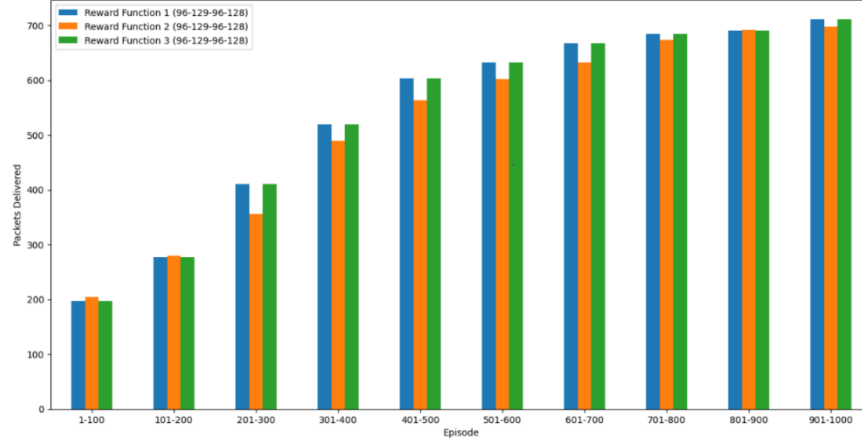


Figure 4.7: Number of packets delivered for different reward functions for the neural network with 96-128-96-128 neurons in hidden layers.

Table 4.2: Results for different reward functions.

Reward Function	Total Packets Delivered After 1000 Episodes
1	539269
2	518865
3	548207

As seen in both **Figure 4.8**, reward function 1 outperformed reward function 3 in the latter episodes. The reason behind this result is that at reward function 1, a higher reward is given for each successful packet delivery. The agent in reward function 1 is more focused on delivering packets so it achieved better overall results than the reward function 3. In this experiment, the only performance metric was the delivered packets. To see the changes in average packet delivery and average delay, both reward functions will be examined against the Dijkstra algorithm. Three experiments for two of the proposed methods and the Dijkstra algorithm. Each experiment has varying buffer sizes. All of the training was done on a 95 GB buffer, but the main goal was to

design an algorithm that can work on multiple routers that have different hardware. So, the proposed methods are also tested in routers with 45 GB and 22 GB buffers too. Average number of packets delivered, average packet delay in terms of timesteps and the average path length in each 200 episodes are measured for each experiment. This time, there is no need to add reward function because Dijkstra has no reward function and the other proposed methods' total collected rewards are not comparable. For the exploration strategy, the parameters for the test phase given **Table 3.2** are used. There will not be any exploration. The network will only exploit in all of the tests.

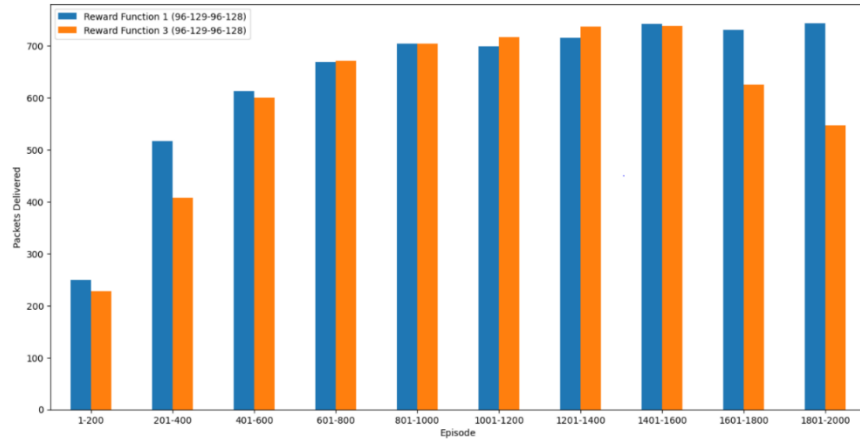


Figure 4.8: Number of packets delivered for reward function 1 and reward function 3.

Table 4.3: Results for reward function 1 and reward function 3.

Reward Function	Total Packets Delivered After 2000 Episodes
1	1275621
3	1194791

Figure 4.9 shows the average number of packets delivered for the router that has the largest buffer size. Even if network gets more crowded in last episodes, the routers will still have enough buffer. The proposed methods managed to deliver more packets than shortest path. Each value for the shortest path looks identical because shortest path is not a learning system. Meanwhile the proposed methods delivered more packets than shortest path in every column. Whether the network is crowded or not, they were able to manage network traffic effectively. The method with reward function 3 also

delivered more packets than the other proposed method most of the time. It shows that, giving lesser reward for the successful packet delivery will produce more brave paths than just achieving the goal.

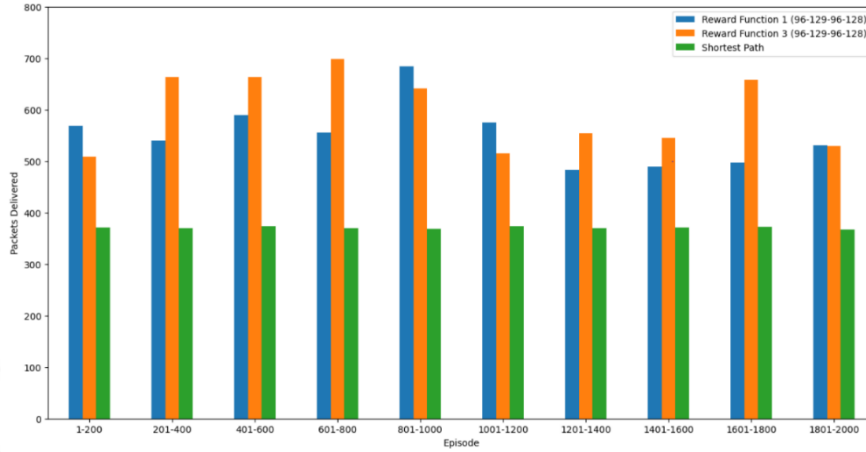


Figure 4.9: Average number of packets delivered on 95 GB buffer.

In **Figure 4.10**, the results of a QoS metric is given. When network is less crowded in the earlier episodes, shortest path had lesser delay than the proposed methods but when routers are filled with more packets in their queues, proposed methods showed better traffic management and they managed to deliver more packets with lesser delay. It shows the effectiveness of routing method that learns dynamically against static method. The proposed method with reward function 3 have higher delay in first 600 episodes but it has lower delay than the proposed method with reward function 1 between the episode 601 and 1800. In last 200 episodes, proposed method with reward function 3 has higher delay than the other proposed method but it managed to have lesser delay in most of the episodes so proposed method with reward function 3 is the best performance in this metric.

In **Figure 4.11**, shows the average path length for each method. This metric is calculated to measure the trade off between average path length and average delay. The produced path might be longer but longer path might result lesser delay. Shortest path always chooses the next hop router as closest one to the destination router. This is the reason that it has the lowest average path length. Both proposed methods took longer paths but they managed to deliver more packets and have lesser delay. Proposed methods know how to manage the traffic by past experiences. Some routers get more

crowded in the shortest path but it still chooses the same routers even if they will result higher delay because there is no learning system.

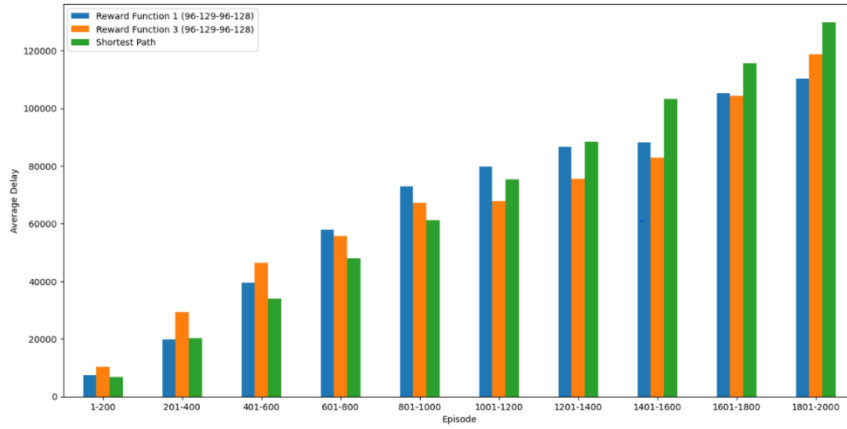


Figure 4.10: Average packet delay on 95 GB buffer.

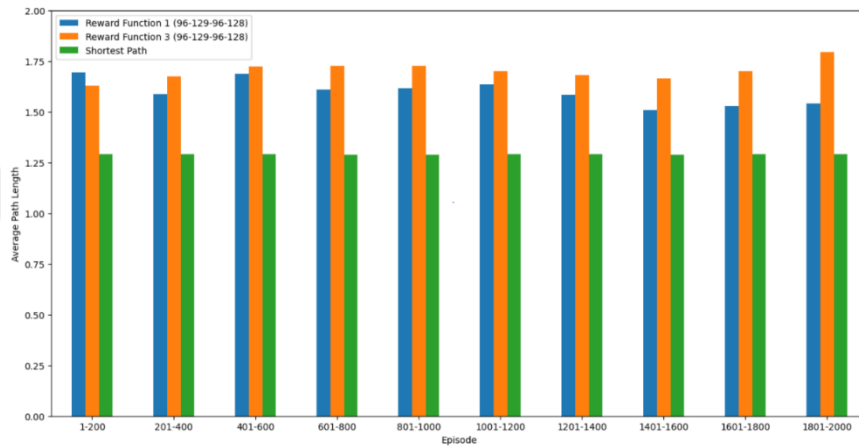


Figure 4.11: Average path length on 95 GB buffer.

In **Figure 4.12**, the routers have lesser buffer than the previous experiment. This time, proposed network with reward function 1 showed the best results in the most of the episodes. In the earlier episodes, it managed the traffic better than the other methods and the latter episodes it probably faced with more seen data than the other proposed method. Shortest path still has lower packet delivery compared to proposed methods.

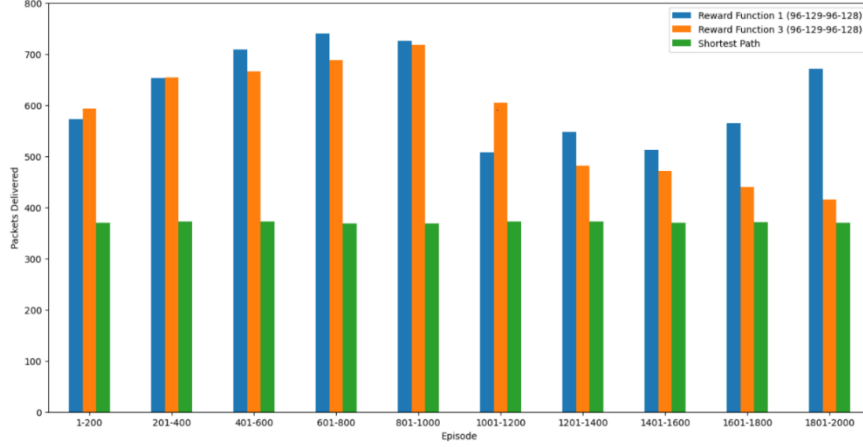


Figure 4.12: Average packets delivered on 45 GB buffer.

Even if proposed method with reward function 1 delivered more packets, the average delay of the proposed method with reward function 3 is lowest compared to other two methods in the latest episodes according to results shown in **Figure 4.13**. Shortest path managed to have the best average delay but when network gets more crowded the delay of shortest path raised significantly. This experiments show that giving more reward for the successful packet delivery will not guarentee both more packet delivery and lesser delay at the same time because of unpredictable network traffic. It is more important to observe that whether the reward function is, proposed methods outperformed existing algorithm.

Again, the average path lengths for proposed methods are longer than shortest path on 45 GB buffer shown in **Figure 4.14**. In the intervals where proposed method with reward function 1 outperformed the other proposed method in terms of packet delivery shown in **Figure 4.13**, average path length for reward function 1 is usually longer than the other proposed method. Proposed method with reward function 3 found a balance between packet delivery and delay with adjusting the path length.

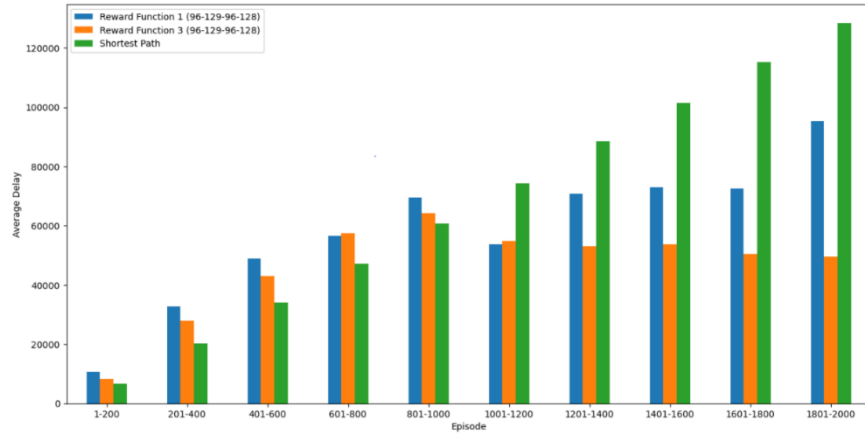


Figure 4.13: Average packet delay on 45 GB buffer.

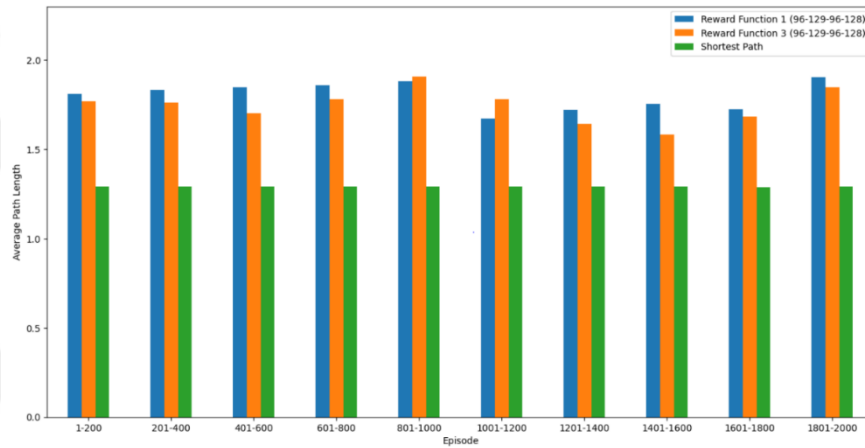


Figure 4.14: Average path length on 45 GB buffer.

The latest experiments will be on 22 GB buffer. These experiments will be on the smallest buffer size among all experiments.

In **Figure 4.15**, it shows the average packets delivered on the least amount of buffer size. This experiment shows the most challenging network environment. Because in every episode, 2000 packet is generated and each packet is 64 Kilobyte. 22 GB buffer will get nearly full in the latest episodes.

Both of the proposed methods already outperformed the Dijkstra but when routers get more crowded and each routing decision becomes critical due to limited amount of remaning buffer size in the latest episodes, proposed method with reward function 3 outperformed both. This shows that giving lesser reward for the packet reaching the destination will lead to more complex routes but it handles the most difficult network

condition better than the others. The rest of the metrics will be examine to give the final verdict.

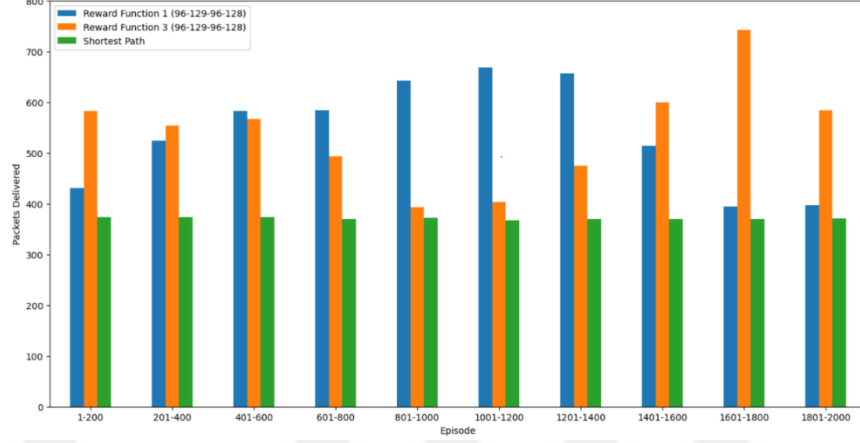


Figure 4.15: Average packets delivered on 22 GB buffer.

Average delay for each method is given in **Figure 4.16**. This time, proposed method with reward function 1 has the highest delay among other methods. This is caused because in earlier episodes, it could not find a balance between delivering packets and minimizing the delay. The method is more focused on delivering packets. And when the network gets more crowded, proposed method with reward function 3 outperformed both method significantly. It also delivered more packets because the focus is not just delivering packets.

The design of the reward function 3 is more suitable for the most challenging scenarios. In the crowded network traffics, minimizing delay is also helpful fore delivering packets. So the balanced approach worked even better than the rest. In this experiment, proposed method with reward function 1 behaved closer to shortest path than a learning system.

As for the last **Figure 4.17**, it shows the average path length of all methods. In the latest two hundred episodes, proposed method with reward function 3 preferred significantly longer paths than the other methods. This shows that the proposed method with reward function 3 learned from past mistakes and in order to prevent packet loss and higher delay, it took longer paths. This ability differs from the shortest path approach.

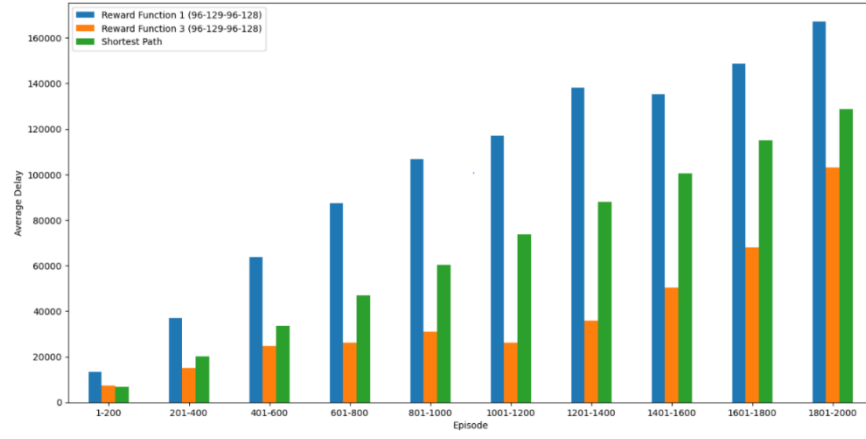


Figure 4.16: Average packet delay on 22 GB buffer.

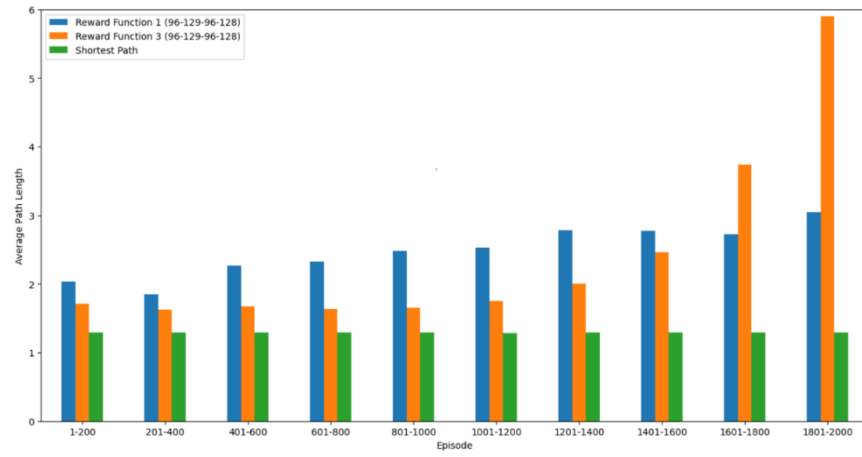


Figure 4.17: Average path length on 22 GB buffer.

Now the overall results of these experiments with exact values are given in **Table 4.4**, **Table 4.5**, and **Table 4.6**.

Table 4.4: Experiment results on 95 GB buffer.

Routing Method	Total Packets Delivered After 2000 Episodes	Average Path Length of 2000 Episodes (Nodes)	Average Delay of 2000 Episodes (Timesteps)
Proposed Method with Reward Function 1	1101904	1.59	66775.23
Proposed Method with Reward Function 3	1195236	1.7	65829.97
Shortest Path	741780	1.29	68298.78

Table 4.5: Experiment results on 45 GB buffer.

Routing Method	Total Packets Delivered After 2000 Episodes	Average Path Length of 2000 Episodes (Nodes)	Average Delay of 2000 Episodes (Timesteps)
Proposed Method with Reward Function 1	1239944	1.8	58335.74
Proposed Method with Reward Function 3	1146389	1.74	46258.77
Shortest Path	740589	1.29	67710.32

Table 4.6: Experiment results on 22 GB buffer.

Routing Method	Total Packets Delivered After 2000 Episodes	Average Path Length of 2000 Episodes (Nodes)	Average Delay of 2000 Episodes (Timesteps)
Proposed Method with Reward Function 1	1079064	2.48	101460.4
Proposed Method with Reward Function 3	1079247	2.41	38768.56
Shortest Path	742203	1.28	67351.64

In **Table 4.4** where buffer size is the largest, delay of each method is nearly the same. This shows that the difference between an intelligent method and classic heuristic will not be significant when routers have enough remaining buffer sizes. Proposed methods still managed to outperform the shortest path in the performance metrics. Especially packet delivery difference is significant.

When router buffer size is half the size, shortest path still showed same performance but the difference between proposed methods with different reward functions differs shown in **Table 4.5**. The proposed method with reward function 1 managed to deliver more packets but lacked behind proposed method with reward function 3 when it comes to average delay. This is because reward function 3 is giving less reward for packet delivery. This shows that it is a better approach to minimize delay when main when not forcing the agent to deliver packets more.

Lastly, **Table 4.6** shows the results of the most challenging part of the thesis. This is where routers are mostly filled with the packets than empty space in the latest episodes. Both proposed methods outperformed the shortest path with delivering more packets but reward function 3 has the lowest average delay than both. This is because the reward function 3 gives more freedom to the agent when selecting the paths. So, for the most critical states, agent acted more freely and delivered packets with lesser delay.

Meanwhile, the proposed method with reward function 1 acted closer to shortest path and it is failed to minimize delay. It still managed to deliver more packets but some of the packets delivered to destination with really long delays.

4.3. Discussion

Several types of experiments have been done to find the best neural network architecture. It has been seen that the neural network with the biggest number of parameters performed the best. This neural network has been used to evaluate the different reward functions. Two of the best performing reward functions have been moved to the test against Dijkstra algorithm. After 2000 episodes of tests, the proposed methods outperformed Dijkstra algorithm in terms of number of packets delivered in every experiment setup. Average delay is also lower than the experiments made on 95 GB and 45 GB but in the experiment on the 22 GB total buffer size, the average delay of proposed method with reward function 1 is higher. The possibility is some of the packets stuck in loops for a long time and caused higher delay and it failed achieve a balance between the packet delivery and delay.

It has been decided that reward function 3 is the best performing one in the most challenging scenarios. But also, both proposed algorithms showed promising results. Naturally, the Dijkstra algorithm has the shorter average path length but even if proposed methods took longer paths for routing, they still managed to make intelligent routing decisions and manage the traffic dynamically. The proposed methods can adapt to the traffic changes and ensure that QoS requirements are fulfilled.

5. CONCLUSION

In this thesis, the aim of this study was to learn from past routing decisions and make better routing decisions in the future network traffic states. It has been achieved by surpassing static heuristic algorithm. To our knowledge, the DDQN has never been used for this topology.

Two methods have been proposed for the routing problem. The difference is their reward functions. Both methods outperformed the classical routing approach in performance metrics on both 95 GB and 45 GB routers. But on the 22 GB router, the routing algorithm with the reward function 1 failed to minimize delay. It failed to find a characteristic between a dynamic approach and the shortest path. This shows us finding the balance between reaching the destination quicker and choosing the least congested routes, even if path lengths are longer, will give the best results in the challenging scenarios. This is why designing the right reward function is necessary for the routing problem, even if we use a dynamic routing approach.

The real life networks are much more complex than this environment. Even in an environment with ten hosts, classical routing methods fail to outperform a dynamic approach. The shortest path routing always produced the same result; meanwhile, dynamic routing methods proposed new routes that are learned from past examples. New routing methods like the proposed ones in this thesis need to be used in real life scenarios in order to fulfill the QoS requirements of the upcoming concepts.

To make the routing problem more complex, the hosts that are in motion and never stop will be researched in the future. Tracking each device's location and delivering packets to the moving hosts is a problem that was not present in this work. This kind of mobile network environment will make the routing problem significantly more complex due to the unpredictable trajectories of hosts. The current proposal is to research more advanced deep reinforcement learning-based algorithms that can generalize better. This future work will be closer to real-life mobile networks. After this research, real life demonstrations of the proposed algorithm will also be considered.

REFERENCES

- Bellman, R. (1958). On a routing problem. *Quarterly of applied mathematics*, 16(1), 87-90.
- Bengio, Y. (2009). Learning Deep Architectures for AI.
- Chow, T. W. S., & Leung, C. T. (1996). Nonlinear autoregressive integrated neural network model for short-term load forecasting. *IEE Proceedings-Generation, Transmission and Distribution*, 143(5), 500-506.
- Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1), 269–271. <https://doi.org/10.1007/bf01386390>
- Fortz, B., & Thorup, M. (2000, March). Internet traffic engineering by optimizing OSPF weights. In *Proceedings IEEE INFOCOM 2000. conference on computer communications. Nineteenth annual joint conference of the IEEE computer and communications societies (Cat. No. 00CH37064) (Vol. 2, pp. 519-528)*. IEEE.
- Glorot, X., Bordes, A., & Bengio, Y. (2011, June). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics (pp. 315-323)*. JMLR Workshop and Conference Proceedings.
- Guck, J. W., Van Bemten, A., Reisslein, M., & Kellerer, W. (2017). Unicast QoS routing algorithms for SDN: A comprehensive survey and performance evaluation. *IEEE Communications Surveys & Tutorials*, 20(1), 388-415.
- Hasselt, H. (2010). Double Q-learning. *Advances in neural information processing systems*, 23.
- Huber, P. J. (1992). Robust estimation of a location parameter. In *Breakthroughs in statistics: Methodology and distribution (pp. 492-518)*. New York, NY: Springer New York.
- Kodialam, M., & Lakshman, T. V. (2000, March). Minimum interference routing with applications to MPLS traffic engineering. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064) (Vol. 2, pp. 884-893)*. IEEE.
- Kurose, J. R., & Ross, K. (2021). *Computer Networking [Global Edition]*.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, 521(7553), 436-444.
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.

- Li, S. E. (2023). Deep reinforcement learning. In Reinforcement learning for sequential decision and optimal control (pp. 365-402). Singapore: Springer Nature Singapore.
- Lin, L. J. (1992). Reinforcement learning for robots using neural networks. Carnegie Mellon University.
- Mammeri, Z. (2019). Reinforcement learning based routing in networks: Review and classification of approaches. IEEE access, 7, 55916-55950.
- Manfredi, V., Wolfe, A. P., Wang, B., & Zhang, X. (2021, June). Relational deep reinforcement learning for routing in wireless networks. In 2021 IEEE 22nd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM) (pp. 159-168). IEEE.
- McClelland, J. L., McNaughton, B. L., & O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3), 419.
- Medina, A., Taft, N., Salamatian, K., Bhattacharyya, S., & Diot, C. (2002). Traffic matrix estimation: Existing techniques and new directions. ACM SIGCOMM Computer Communication Review, 32(4), 161-174.
- Miao, G., Zander, J., Sung, K. W., & Slimane, S. B. (2016). Fundamentals of mobile data networks. Cambridge University Press.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540), 529-533.
- Morales, E. F., & Zaragoza, J. H. (2012). An introduction to reinforcement learning. In Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions (pp. 63-80). IGI Global.
- Morales, M. (2020). Grokking deep reinforcement learning. Manning Publications.
- Moy, J. (1989). The OSPF specification. RFC-1131.
- Naeem, M., Rizvi, S. T. H., & Coronato, A. (2020). A gentle introduction to reinforcement learning and its application in different fields. IEEE access, 8, 209320-209344.
- Petrosyan, A. (2024, November 5). Worldwide digital population 2024. <https://www.statista.com/statistics/617136/digital-population-worldwide/>
- Reis, J., Rocha, M., Phan, T. K., Griffin, D., Le, F., & Rio, M. (2019, July). Deep neural networks for network routing. In 2019 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE.

Riedmiller, M. Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method. *Mach. Learn.: ECML*, 3720, 317–328 (Springer, 2005).

Roughan, M., Greenberg, A., Kalmanek, C., Rumsewicz, M., Yates, J., & Zhang, Y. (2002, November). Experience in measuring backbone traffic variability: Models, metrics, measurements and meaning. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement* (pp. 91-92).

Ruotian xie. (2024). Traffic datasets: Abilene, GEANT, TaxiBJ. IEEE Dataport. <https://dx.doi.org/10.21227/7x3c-5p06>

Samadi, R., & Seitz, J. (2022, November). Machine learning routing protocol in mobile IoT based on software-defined networking. In *2022 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)* (pp. 108-111). IEEE.

Scherfke, S., Lünsdorf, O., Grayson, P., LaFevers, E., Pinckney, T., Klein, C., ... & Grogan, M. (2020). SimPy. Discrete Event Simulation for Python. Date accessed, 25(04), 2023.

Shrestha, A., & Mahmood, A. (2019). Review of deep learning algorithms and architectures. *IEEE access*, 7, 53040-53065.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning, second edition: An Introduction. MIT Press.

Szepesvári, C. (2022). Algorithms for reinforcement learning. Springer nature.

Tang, F., Mao, B., Fadlullah, Z. M., Kato, N., Akashi, O., Inoue, T., & Mizutani, K. (2017). On removing routing protocol from future wireless networks: A real-time deep learning approach for intelligent traffic control. *IEEE Wireless Communications*, 25(1), 154-160.

Tassiulas, L., & Ephremides, A. (1990, December). Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. In *29th IEEE Conference on Decision and Control* (pp. 2130-2132). IEEE.

Taylor, P. (2023, January 19). Global number of devices and connections per capita 2018-2023. <https://www.statista.com/statistics/1190270/number-of-devices-and-connections-per-person-worldwide/>

Thrun, S., & Schwartz, A. (2014, March). Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 connectionist models summer school* (pp. 255-263). Psychology Press.

Tieleman, T., & Hinton, G. (2017). Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. Technical report.
Tsitsiklis, J., & Van Roy, B. (1996). Analysis of temporal-difference learning with function approximation. *Advances in neural information processing systems*, 9.

United Nations Population Fund. (2024). World Population Dashboard. Unfpa.org. <https://www.unfpa.org/data/world-population-dashboard>

Valadarsky, A., Schapira, M., Shahaf, D., & Tamar, A. (2017). A machine learning approach to routing. arXiv preprint arXiv:1708.03074.

Van Hasselt, H., Guez, A., & Silver, D. (2016, March). Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence (Vol. 30, No. 1).

Watkins, C. J. C. H. (1989). Learning from delayed rewards.

Yao, H., Mai, T., Jiang, C., Kuang, L., & Guo, S. (2019). AI routers & network mind: A hybrid machine learning paradigm for packet routing. IEEE computational intelligence magazine, 14(4), 21-30.



BIOGRAPHY

Arif Burak DİKMEN graduated from Fatih Sultan Mehmet Vakif University – Computer Engineering in 2020 with B.Sc. degree. Started his M.Sc. in Computer Engineering at Gebze Technical University in 2022. After working in the industry for 3 years, he decided to join Gebze Technical University as a research assistant in 2023.



PUBLICATIONS AND PRESENTATIONS FROM THE THESIS

Dikmen, A. B., & Çelebi, H. (2024). Deep Reinforcement Learning Routing in Mobile Networks. 2024 9th International Conference on Computer Science and Engineering (UBMK), 988–993. <https://doi.org/10.1109/ubmk63289.2024.10773524>

