Exploiting a cross-layer design for network performance improvement through Deep Reinforcement Learning

JORIS KUIPER, University of Twente, The Netherlands

The increase in industrial IoT has brought many different connectivity requirements such as latency, packet loss and throughput. With this rise of connected devices, Quality of Service (QoS) has become more important to ensure these requirements are met by the network. However, initial QoS in Wi-Fi has only been managed by the MAC layer, limiting the application of QoS. More diverse QoS requirements must be met for Industrial Internet of Things (IIOT) networks currently not supported by QoS. Since QoS is also affected by parameters on the other layers of the OSI Stack, we deployed a cross-layer design to improve the QoS using Deep Reinforcement Learning (DRL). We achieved similar throughput, decreased latency by 9.47% and decreased packet loss by 24.90% compared to Minstrel using a DDQN DRL model.

Additional Key Words and Phrases: WiFi, SDN, QoS, Quality of Service, IoT, Internet of Things, ns-3, DRL, Deep Reinforcement Learning, cross-layer, minstrel

1 INTRODUCTION

In recent years, the Internet of Things has seen a massive increase in adoption. Networks have been rapidly expanding, and the expectations are that the number of connected devices will double from roughly 15 billion in 2023 to 30 billion in 2030 [26].

This rise of connected devices can also be found in Industrial IoT (IIoT) networks. From existing factories introducing more automation and monitoring to fully automated warehouses, IIoT networks have seen a diverse landscape of use cases, such as sensors, autonomous mobile robots, safety controls, and other applications. Some of these applications can be found in Figure 1 [8]. Each use case and its applications come with a diverse set of connectivity requirements, which the employed network access technology tries to satisfy through its Quality of Service (QoS) tools [22]. However, these QoS requirements are diverse and hard to meet [7].

TScIT 41, July 05, 2024, Enschede, NL

Although various communication methods (e.g. LoRaWan, WirelessHART) have been proposed and used in IIoT networks [24]. This research will focus on the technology of IIoT networks combined with Wi-Fi (also known as IEEE 802.11) as its access technology since Wi-Fi is more commonly used in IoT networks.

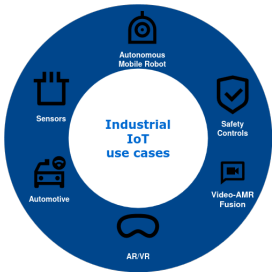

Fig. 1. Industrial IoT use-cases

WiFi employs multiple techniques at MAC and PHY layers to handle the increased traffic and QoS requirements. In the MAC layer, a contention-based scheme called Distributed Coordinated Function (DCF) exists. In DCF, the devices connected to an Access Point (AP) scan the air interface to determine the availability of the channel. If this channel is idle, the connected device starts transmitting. However, a collision occurs if the channel is busy or two devices start transmitting. To prevent this, Wi-Fi uses Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) [5].

To meet the QoS needs of users, WiFi uses an Enhanced Distributed Channel Access (EDCA) scheme. EDCA creates four Access Categories (AC): background, best effort, video, and voice. These ACs are then mapped to a transmit queue. Each AC has four parameters to control them, and these are the minimum contention window size, the maximum contention window size, the transmission opportunity limit, and the arbitration inter-frame spacing [30]. Using these four parameters, EDCA can prioritise one channel over the other so that the QoS of these categories can be met. However, if a high-priority AC constantly transmits, a lower-priority AC might not be able to access the channel and transmit.

However, this method of QoS is limited due to the diverse needs of modern IIoT networks. Other research in QoS improvements has mainly contained themselves to a single layer, and a necessity has been identified for a more cross-layer design to meet these requirements [25].

Metric	Unit	Description
Latency	ms	The time it takes for a data packet to travel from the source to the destination.
Jitter	ms	The variation in packet arrival time. It is a measure of the stability and predictability of the network.
Packet Loss	%	The percentage of packets that are lost during transmission.
Throughput	Mbps	The rate at which data is successfully transmitted over the network.
Availability	%	The percentage of time the network is available for use.

Table 1. Quality of Service (QoS) Metrics for WiFi

Some researchers proposed Network Slicing in Wi-Fi using SDN technology to improve QoS in IoT networks [3]. Network Slicing brings flexibility in QoS delivery as any number of slices can be created and managed in the network; however, most works focus on only MAC layer parameters like CW, AIFSN and Queue Quantum to control slice resources, which is not sufficient to create reliable slice differentiation. Since QoS is being managed at all layers of the OSI stack in any access technology, Cross-Layer Design targeting parameters of all layers possess significant potential to improve the QoS in the network.

Researchers have employed cross-layer design by targeting a combination of the MAC and Network layers and the MAC and PHY layers; however, their work remains limited to only a few parameters due to the complexity of the problem. Most previous work targeted one or two layers of the OSI model. However, with the advancements in AI and ML, algorithms can now handle much higher complexities; therefore, we aim to employ DRL to develop a cross-layer design-based QoS solution that targets parameters from transport, network, MAC and PHY layers together, thus combining various layers into this complex problem. Since QoS optimization will improve network throughput by managing network resources efficiently, we aim to analyze the effects of our DRL solution on the throughput in the network compared to other data rate managers such as minstrel [16].

2 RELATED WORK

Early research regarding a cross-layer design in Wi-Fi networks to improve QoS has proposed a cross-layer scheduler Liu et al. [14]. Each connection would be assigned a priority updated dynamically based on the channel quality, QoS satisfaction, and service priority. Then, the scheduler would always schedule the highest priority first. The scheduler provided diverse QoS guarantees using the transmission mode at the PHY layer based on their requirements at the MAC layer. However, further research was needed to evaluate the scheduler better because it negatively affects bandwidth and has performance degradation on lower-priority connections.

Another QoS management system with a cross-layer design was proposed in Chen et al. [6]. The authors created an algorithm using the Received Signal Strength Indicator (RSSI) value and channel sensing in the link layer and the QoS in the application layer to provide the QoS in the network. The QoS algorithm classifies the data flows and selects the best parameters to improve the QoS. However, in this algorithm, the quality of various classes is reduced when the network becomes busy.

Another way of improving QoS is by using software-defined networking (SDN). The Open Networking Foundation [9] states that "*In the SDN architecture, the control and data planes are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications*". This decoupling makes it easier to manage the network resources, efficiency, and quality of service while also being easy to program. SDNs have made it possible to define network flows, which has helped improve the QoS. [12]

SDNs are used in Amur et al. [3] and in Richart et al. [20] to build network slices in SDN to improve the QoS in IoT networks. With slicing, network flows that share similar requirements are grouped and ensure the QoS of the services in the slice regardless of the resources [20]. The slices have proved to be an effective technique to improve the QoS in a network with various requirements for QoS. However, these slices in WiFi are only controlled through the MAC layer. The potential to control them more reliably exists through a cross-layer design utilizing more than one or two layers.

Some research works employing AI and ML for Cross-Layer Design have also been proposed in literature [31]. Here, the authors investigated the possibilities of AI and ML for building reliable QoS in a cross-layer design for a dense IoT network. In Zia et al. [31], the authors have identified that Open System Interconnect (OSI) model stack parameters can be used to improve the QoS in IoT networks. However, the authors have proposed that further research is needed in optimizing QoS with a cross-layer design through AI & ML.

Deep Reinforcement Learning (DRL) has been used in Wang et al. [28] to build a more energy-efficient 5G wireless network while achieving a good QoS. Although this work focuses on 5G technology, similar work on Wi-Fi technology is missing in the literature and can bring significant improvements. Due to the technology difference between 5G and Wi-Fi, it is interesting to see if these results can be replicated in Wi-Fi networks. Pundir and Sandhu [18] discussed the current state of AI and QoS. The authors classified the QoS across the OSI layers and investigated future research directions. They recommend an ML-integrated approach to improve QoS further. This integrated approach was investigated in Abbasi et al. [1]; the authors used DRL to improve QoS at the MAC layer. They specifically investigated spectrum access, joint user association and adaptive data rate control. However, the authors only investigated DRL with the MAC layer and did not use a cross-layer design.

Many other works have investigated a cross-layer design. In Qu et al. [19], the authors investigated a cross-layer design for optimizing video traffic by proposing a new packet scheme and changing the video encoder to optimize the network. However, they did not modify any parameters in the channel to improve conditions. Furthermore, their solution needed specific conditions and requirements to be in place for it to work, which might not be the

case in every network. More works have been investigated; however, they remain limited due to the complexity of the problem [15]. Some research such as Nie et al. [17] has investigated singular layers of the OSI model regarding the OSI stack. The authors mainly used the TCP Initial Window with Reinforcement Learning. However, due to the nature of the OSI model, more parameters and different layers of the OSI stack can and will influence the stability of QoS.

3 NETWORK MODEL

The Industrial IoT network comprises various sensors, AGVs, Cameras, robotic arms, etc., deployed across different floors in a factory 4.0 environment. All devices are interconnected through different access technologies, including Wi-Fi. To simulate such a factory environment, we have employed a widely used discrete event-based Network Simulator called NS-3.

NS-3 is a network simulator built in C++ which aims to be a platform to test devices and implementations when moving from simulation to experiment [21]. As a successor to NS-2, it tries to be easier to debug and easier to use. The simulator models network nodes (e.g. IoT devices) that use network devices (e.g. WiFi APs, Base Stations) to connect to communications channels through communications protocols, protocol headers and network packets, thus simulating a network [21].

For our network model, we simulated n nodes and distributed them over 40 square meters around the AP. For simplicity, static nodes were considered in an industrial environment; however, mobility can be added to certain nodes to simulate Automated Guided Vehicles (AGV). In the nodes, we defined four applications they could use: best effort, control data for Automated Robotic Arms, real-time machine monitoring and CCTV. For each application we defined a data rate it would use, these can be found in Table 2. For the real-time monitoring application, we chose 25 Mbps and 4 Mbps for the CCTV application. This was chosen because it is important that most details can be seen for real-time monitoring. For CCTV, the sharpness of the image is less important, and some quality can be sacrificed. The control data would use small packets and need to arrive consistently; thus, a smaller data rate of 0.512 was selected. For all other use cases, we grouped them under the best-effort application and gave them a data rate of 0.256 Mbps. During the simulation, the nodes would only be receiving data from the AP in a manner similar to the UDP protocol.

To simulate the wireless conditions of an indoor factory environment, we used two propagation loss models: the Log-distance Path Loss Model and the Nakagami Propagation Loss Model. The Log-distance Path Loss Model is a model that is well suited for urban and industrial environments [2] and thus well-suited for our industrial setting. The Nakagami Propagation Loss Model makes it possible to simulate environments with no direct line of sight to the AP, which is the case in our industrial environment, where there might be walls or other machinery between the node and the AP. Due to these extra parameters and propagation loss models, we can define our industrial environment more fine-grained [23].

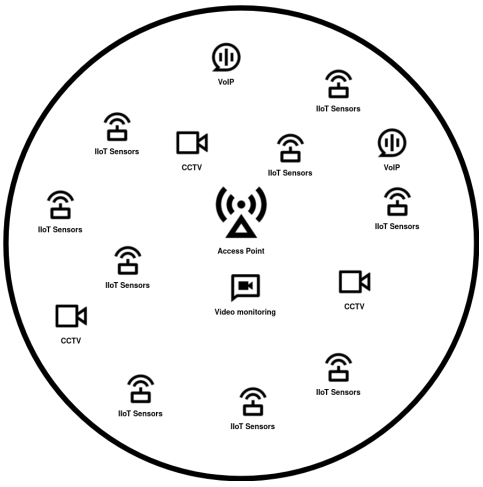

Fig. 2. Network model example

Description	Data Rate (Mbps)	Amount of users
Best effort	0.256	$n - n\_control - n\_monitoring - n\_cctv$
Control data	0.512	$n\_control$
Real-time monitoring	25	$n\_monitoring$
CCTV	4	n_cctv

Table 2. Data Rates and amount of users for Various Applications

4 REINFORCEMENT LEARNING MODEL

To optimise our network model, we decided to deploy Deep Reinforment Learning (DRL). DRL is well suited for environments where you have a relatively small set of features of a complex state where these choices can and will affect later states [4] and optimizing our network is an example of such an environment. DRL can be described as a Markov Decision Process (MDP). In MDP, there is a set of states, a set of actions, and a reward function that tries to find an optimal policy that will yield a maximum expected return from the states [4].

4.1 States

To describe our environment, our states would be modelled by using the following variables as input: average throughput, average latency, average packet loss ratio, average queue fill-up, average signal-to-noise ratio, and average collision for each timestep of the simulator. Average throughput is a key indication of the network's performance. A low throughput indicates that less data is coming through, thus increasing other variables. Thus, network throughput is a key metric for the performance of our network. Latency is the delay in the sending and receiving of data packets. A high latency indicates that the network is performing poorly and since we are working with applications that need real-time performance, latency is a good indicator next to throughput.

Packet loss tells whether the network is reliable or whether packets are dropping or colliding, which leads to re-transmissions and reduced network efficiency. Since we need a reliable network for

our applications, packet loss is a good metric to model our states with.

A high queue fill-up indicates that a lot of data is in the network buffer and suggests that network performance is decreasing. Therefore, it is a good indicator of congestion in the network, which we are trying to avoid.

Signal-to-noise (SNR) measures the signal quality compared to the background noise and is an important measurement regarding the reliability of data transmission. Without keeping track of the signal-to-noise ratio, it is harder to identify what is causing the higher packet-loss or the higher latency. Since SNR is used to select the proper Modulation Coding Scheme (MCS) and Transmit power which are both possible actions in our DRL model, it is important to use SNR as one of the variables in our states.

The average collision rate is the final variable we used to model the network's state. It measures the number of collisions and, if managed effectively, can lead to a smoother and more reliable network. Since the rate of collisions is used to identify a better Contention Window (CW), it is a key metric for our states.

Together, these variables are able to give a good overview of the environment of our network without cluttering the DRL model with too much information when it is trying to select an action to perform.

4.2 Actions

For actions to perform, we selected three variables to be modified by our DRL model. These are the Contention Window (CW), the Modulation and Coding Scheme (MCS), and the transmit power.

The CW controls the time nodes wait before attempting to access the network, influencing collision rates and overall throughput. Research has shown that the CW can be optimized effectively through the use of DRL [29] and its effects on collision rate, which in turn affects the re-transmissions and throughput, makes it a good action to include in our DRL model.

Selecting different MCS can be an important action for our DRL model. Each MCS has different properties (such as amplitude, frequency, and base) which are more effective in different conditions. If the network conditions are optimal, a different MCS would perform with a better throughput than when the conditions are poor. Since our DRL model will be able to react in real-time, changing the MCS will be a very important action it could take.

Another action we selected was changing the transmit power. The transmit power defines the amount of power used to transmit packets over the network. A strong signal might get a better SNR while creating more interference (or collisions). A low signal might result in fewer collisions but a lower SNR, which results in dropped packets needing retransmission. Thus, changing the transmit power is another important action when optimising the network.

For each of these actions, we gave the DRL model three choices: either increase the variable, decrease it or not change it at all. Since the choice can be independent for each variable, this resulted in an action space of 27 possible combinations from which the DRL can choose.

4.3 Reward

The reward function is one of the most important parts of Reinforcement Learning (and, by extension, DRL). The environment rewards the agent as the agent performs actions on the state and continues to the next state. The agent's goal is to take actions that result in receiving a bigger (or increasing) reward [4]. However, this process is trial and error; thus, every interaction makes the agent learn more about the environment if the reward function is properly defined.

Many different reward functions were tested for our DRL setup. We started with a complex reward function where throughput, latency, and packet loss were weighted and added to each other through a rather complex formula, see Equation 1. The next reward function was a simpler iteration of the first equation. In this reward function, if the difference were positive for each of the variables throughput, latency, and packet-loss, they would receive a +10. They would be scored with a −10 if negative. Then, these values would be weighted and added to each other. Unfortunately, we also did not see any learning with this reward function, leading to us making an even more simplified reward function. We used a dense reward function and dropped the latency and packet loss for our final reward function. The exact equation can be found in Equation 2. This reward, combined with the DRL model discussed below, showed learning and some improvements.

$$\text{reward} = \frac{w_{\text{th}}\left(\frac{\text{th}_{\text{diff}}}{\text{th}_{\text{avg}}}\right) - w_{\text{lat}}\left(\frac{\text{lat}_{\text{diff}}}{\text{lat}_{\text{avg}}}\right) - w_{\text{plr}}\left(\frac{\text{plr}_{\text{diff}}}{\text{plr}_{\text{avg}}}\right)}{\text{Tx}_{\text{pwr}}} \tag{1}$$

$$\text{reward} = \begin{cases} 20 & \text{if throughput\_diff} > 5 \\ 10 & \text{if } 2 < \text{throughput\_diff} \leq 5 \\ 0 & \text{if } 0 < \text{throughput\_diff} \leq 2 \\ -10 & \text{if } -2 < \text{throughput\_diff} \leq 0 \\ -20 & \text{if throughput\_diff} \leq -2 \end{cases} \tag{2}$$

4.4 DRL Model

Due to the network simulated in ns-3 being a rather noisy environment, we used the Double Deep Q Network (DDQN) algorithm. DDQN solves the overestimation problem in regular Deep Q Networks (DQN) and performs better than a DQN model in this environment. The DDQN algorithm uses 2 DQN agents, one to select an action and the other to evaluate it. During the DDQN model's training phase, the weights of the selection agent are periodically copied to the evaluation agent. This leads to less overestimation than in regular DQN networks while still keeping computational overhead low [27].

We implemented the Double Deep Q Network (DDQN) in ns-3 using ns-3 gym framework [10]. This extension of ns-3 combines it with the OpenAI gym so that ns-3 can be used more easily for RL research. Furthermore, ns-3 gym also comes with Python bindings for ns-3, so experiments can easily be made and run with Python instead of using C++ directly.

4.5 Neural Network Architecture

The implementation was made using Keras and TensorFlow in Python 3s. The parameters we used in the DDQN implementation can be found in Table 3 and were determined after an empirical search for a configuration that would result in a stable network. The neural architecture of the DDQN model was 2 Dense layers with 256 units and a ReLU activation function. These two layers would then be followed by a third dense layer with 27 units that map the results from the previous 2 layers into 27 possible actions. The optimizer we used was Adam [13], and our loss function was the Mean Square Error function.

Parameter	Value
Learning rate	0.005
Epsilon	0.99
Gamma	0.6
Batch size	128
Epsilon decrement	0.998
Replace target	25

Table 3. Parameters of the DDQN network

5 RESULTS

To evaluate the performance our DDQN model, we ran two scenarios in the ns-3 simulator. The first scenario utilized the DDQN model and configuration and ran the DDQN agent in the AP. The second scenario used the Minstrel-HT [16] rate selection algorithm in the AP. Both scenarios ran for 10.000 iterations with the same network model in ns-3 with 33 nodes. Each iteration would take 0.2 seconds, and we would only send data over the downlink from the AP to the nodes. To measure the performance of our simulations, we kept track of three metrics: average throughput, average latency, and average packet loss.

5.1 Average Throughput

Our first performance metric is that of the average throughput of the network. It is crucial since it measures the overall capacity our network is performing at. Furthermore, its results can immediately indicate what is happening in the network. If the throughput is high, then the network is performing optimally. However, if the throughput is low or decreasing, there is a problem somewhere else in the network. This can range from buffers overflowing or packets being dropped either due to collision or noise. Thus, throughput is an important metric for measuring performance.

The results from both scenarios were first averaged over time. In Figure 3, we plotted the network's average throughput over time. Initially, the DDQN agent performs significantly less than Minstrel. This is due to the trial and error phase of the DRL model. As the simulation continues, the throughput increases until it is stable at around 25 Mbps. The minstrel scenario initially starts stronger than the DDQN agent. However, it experiences a drop in performance until it just outperforms the DDQN agent by 0.61%.

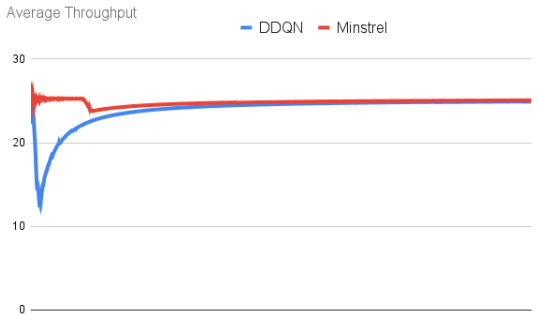

Fig. 3. Average Throughput

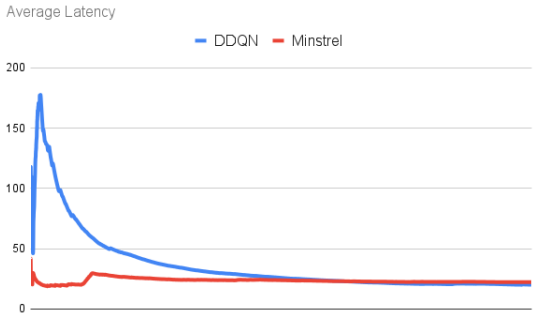

Fig. 4. Average Latency

5.2 Latency

Our next metric for measuring the scenario's performance is latency. Latency shows the time it took for a packet to be delivered at the node. High latency suggests that the packet is stuck in the queue, it is being dropped, or the throughput is low. Low latency indicates either high throughput, low packet-loss, or small queues in the buffer. Thus, high latency is another key metric we can use to evaluate the performance of our scenario.

In Figure 4, the average latency for both scenarios is plotted. The DDQN agent starts with a very high latency. However, this drops significantly to a low latency as the agent learns. For the minstrel scenario, the latency starts lower than compared to the DDQN agent but stays fairly stable as the scenario keeps running. At the end of the simulation, we can see that the DDQN agent has 9.47% lower latency than the Minstrel scenario. This result and a similar throughput might indicate that another metric is performing better, which results in this low latency.

5.3 Packet-Loss

Our third performance metric is packet-loss. Although latency can indicate if significantly more packets are being dropped, we need this third metric to confirm that. Packets being dropped can happen under various circumstances: the buffer is overflowing, there is a low SNR, or there is a high collision rate. Thus the average packet loss

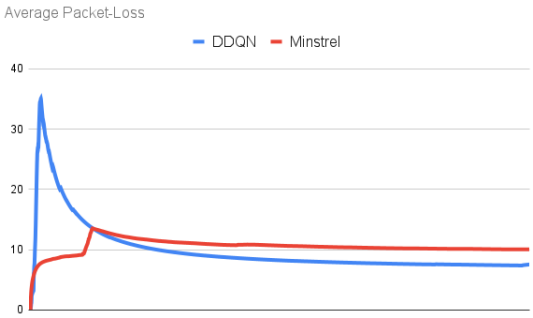

Fig. 5. Average Packet-Loss

can give a better understanding of the performance of our network together with latency and throughput.

For the packet loss ratio, we can see in Figure 5 that the DDQN agent starts with a significantly higher packet-loss. This decreases sharply as the scenario continues. the minstrel scenario starts with a lower packet- loss and has a small increase before becoming more stable. Ultimately, the DDQN agent outperforms the minstrel scenario with a 24.9%lower packet-loss ratio.

5.4 Analysis

To analyze our performance, we have identified three metrics. Together, these metrics can give us a clear and concise understanding of what is happening in our network. We can see that the DDQN agent performs similarly to the Minstrel data rate manager in terms of throughput. Furthermore, the DDQN agent performs better regarding average latency and packet loss. In Minstrel, the only variable it can change is the MCS, whereas our DDQN agent can change the CW, MCS, and Transmit power. We noticed that in our simulations, the DDQN agent was actively performing actions that would enable it to achieve the aforementioned results. This shows that the DDQN agent can have more fine-grained control over the network by reacting to the network conditions with the action that would most affect the performance metrics.

6 FUTURE WORK

Our novel approach to improving the QoS of a network through DRL has resulted in promising results. However, future work is still needed to improve and further optimise the network performance through DRL.

As mentioned before, the DDQN agent only changes variables in the AP to optimize the network. Future work can be done to run the DDQN agent on the nodes; this would require significantly more monitoring and increase the complexity of the problem. However, since our network only sent data from the AP to the nodes, this was out of the scope of our research.

Automated Guiding Vehicles (AGV) are increasingly used in industrial IoT environments. These AGVs are moving around, and thus, this is an important consideration for modelling the network. In our network topology, we only used a network consisting of static nodes due to the complexity the mobility in nodes brings to the network. Future work can be done in applying our model to a network topology with mobility added to the nodes. Initial attempts have shown that mobility will introduce significant difficulties in finding a stable DDQN agent capable of learning.

As the network is being used, small changes in earlier states can influence the state of the network further down the line. These long-term dependencies are hard to identify, and DDQN is unsuitable for these environments. DDQN assumes that a new state can only be influenced by the preceding state, thus missing the long-term dependencies [4].

To capture these dependencies across multiple states, we recommend investigating further the possibility of using the Long Short-Term-Memory (LSTM) model. The LSTM architecture makes use of memory cells and keep/forget gates. This allows the LSTM model to selectively retain or discard information it learns throughout training the model [11]. The LSTM model performs best in an environment where sequential decision-making and delayed rewards occur. This is also the case in our scenarios where the decisions are made sequentially and can impact performance.

In our work, we tried to implement LSTM in our setup. However, we were unable to configure a stable model that would learn. Thus, we encourage future work to explore this promising possibility.

Another component worth further investigation is the reward function. The reward equation is a piece-wise deep reward function that rewards relative to the difference in throughput and no other variables. We recommend investigating the possibility of including latency and packet-loss as part of the reward function. However, our initial attempts to include these performance metrics did not yield a stable network. Furthermore, we believe that through a more systematic investigation, finding a reward function using the metrics will be possible.

Currently, our DRL agent is optimizing the overall network performance using throughput, latency and packet-loss. However, future research can be done in applying a multi agent DRL model in the network and employing network slicing to optimize the QoS and network performance for each specific application and service running on the network.

7 CONCLUSION

With the rise of devices connected to IIoT networks, optimizing the network's performance has become a complex problem. Our novel approach employed a DDQN model to optimize the network performance through Deep Reinforcement Learning.

We simulated a static IoT network in an industrial environment using ns-3. In the network, we deployed a set of nodes around the AP to receive data from the AP in the downlink. We ran two scenarios, one using the Minstrel data rate manager and the other using a DDQN model in the Access Point.

Our DDQN agent has been shown to perform similarly to Minstrel regarding the network's overall throughput. However, at the end of

the scenario, the DDQN agent performed slightly better in latency and significantly better regarding packet loss.

Although these results have promising potential, future work is needed to explore the feasibility of a DDQN agent to improve the QoS of Industrial IoT networks. Furthermore, more parameters from the TCP layer (e.g., congestion window or selective acknowledgements) and MAC layer (e.g., frame aggregation or channel selection) can be included in the DDQN agent to improve the Cross-Layer Design.

8 ACKNOWLEDGEMENTS

I would like to thank my supervisor Kamran Zia MSc MRes for his support, patience, and kindness throughout this research.

REFERENCES

[1] Mahmoud Abbasi, Amin Shahraki, Md. Jalil Piran, and Amir Taherkordi. 2021. Deep Reinforcement Learning for QoS provisioning at the MAC layer: A Survey. *Engineering Applications of Artificial Intelligence* 102 (2021), 104234. https://doi.org/10.1016/j.engappai.2021.104234

[2] Robert Akl, Dinesh Tummala, and Xinrong Li. 2006. Indoor Propagation Modeling at 2.4 GHz for IEEE 802.11 Networks. In *wireless and optical communications*. 1–6.

[3] Sri Harsh Amur, Kamran Zia, Alessandro Chiumento, and Paul Havinga. 2023. Autonomous Network Slicing and Resource Management for Diverse QoS in IoT Networks. In *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. 160–165. https://doi.org/10.1109/PerComWorkshops56833.2023.10150306

[4] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine* 34, 6 (2017), 26–38. https://doi.org/10.1109/MSP.2017.2743240

[5] Sourangsu Banerji. 2013. On IEEE 802.11: Wireless LAN Technology. *CoRR* abs/1307.2661 (2013). arXiv:1307.2661 http://arxiv.org/abs/1307.2661

[6] Jiann-Liang Chen, Shih-Wei Liu, Szu-Lin Wu, and Ming-Chiao Chen. 2011. Cross-layer and cognitive QoS management system for next-generation networking. *International Journal of Communication Systems* 24, 9 (2011), 1150–1162. https://doi.org/10.1002/dac.1218 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/dac.1218

[7] Ren Duan, Xiaojiang Chen, and Tianzhang Xing. 2011. A QoS Architecture for IOT. In *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*. 717–720. https://doi.org/10.1109/iThings/CPSCom.2011.125

[8] WBA Wi-Fi 6/6E for IIOT Project team. 2022. Wi-Fi 6/6E for Industrial IOT Enabling Wi-Fi determinism in an IOT world. https://wballiance.com/wi-fi-6-6e-for-industrial-iot-whitepaper/

[9] Open Networking Foundation. 2012. *Software-Defined Networking: The New Norm for Networks*. Technical Report. Open Networking Foundation.

[10] Piotr Gawłowicz and Anatolij Zubow. 2019. ns-3 meets OpenAI Gym: The Playground for Machine Learning in Networking Research. In *ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)* (Miami Beach, USA). http://www.tkn.tu-berlin.de/fileadmin/fg112/Papers/2019/gawlowicz19_mswim.pdf

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[12] Murat Karakus and Arjan Durresi. 2017. Quality of Service (QoS) in Software Defined Networking (SDN): A survey. *Journal of Network and Computer Applications* 80 (2017), 200–218. https://doi.org/10.1016/j.jnca.2016.12.019

[13] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[14] Qingwen Liu, Xin Wang, and G.B. Giannakis. 2006. A cross-layer scheduling algorithm with QoS support in wireless networks. *IEEE Transactions on Vehicular Technology* 55, 3 (2006), 839–847. https://doi.org/10.1109/TVT.2006.873832

[15] Aqsa Malik, Junaid Qadir, Basharat Ahmad, Kok-Lim Alvin Yau, and Ubaid Ullah. 2015. QoS in IEEE 802.11-based wireless networks: A contemporary review. *Journal of Network and Computer Applications* 55 (2015), 24–46. https://doi.org/10.1016/j.jnca.2015.04.016

[16] Andrew Mcgregor and Derek Smithies. 2010. Rate adaptation for 802.11 wireless networks: Minstrel. *Submitted to ACM SIGCOMM* (2010).

[17] Xiaohui Nie, Youjian Zhao, Dan Pei, Guo Chen, Kaixin Sui, and Jiyang Zhang. 2018. Reducing Web Latency Through Dynamically Setting TCP Initial Window with Reinforcement Learning. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. 1–10. https://doi.org/10.1109/IWQoS.2018.8624175

[18] Meena Pundir and Jasminder Kaur Sandhu. 2021. A Systematic Review of Quality of Service in Wireless Sensor Networks using Machine Learning: Recent Trend and Future Vision. *Journal of Network and Computer Applications* 188 (2021), 103084. https://doi.org/10.1016/j.jnca.2021.103084

[19] Qi Qu, Yong Pei, James W Modestino, and Xusheng Tian. 2006. Source-adaptation-based wireless video transport: a cross-layer approach. *EURASIP Journal on Advances in Signal Processing* 2006, 1 (2006), 028919.

[20] Matías Richart, Javier Baliosian, Joan Serrat, Juan-Luis Gorricho, and Ramón Agüero. 2020. Slicing With Guaranteed Quality of Service in WiFi Networks. *IEEE Transactions on Network and Service Management* 17, 3 (2020), 1822–1837. https://doi.org/10.1109/TNSM.2020.3005594

[21] George F. Riley and Thomas R. Henderson. 2010. *The ns-3 Network Simulator*. Springer Berlin Heidelberg, Berlin, Heidelberg, 15–34. https://doi.org/10.1007/978-3-642-12331-3_2

[22] Manisha Singh and Gaurav Baranwal. 2018. Quality of Service (QoS) in Internet of Things. In *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*. 1–6. https://doi.org/10.1109/IoT-SIU.2018.8519862

[23] Pranav Kumar Singh. 2012. Influences of TwoRayGround and Nakagami propagation model for the performance of adhoc routing protocol in VANET. *International Journal of Computer Applications* 45, 22 (2012), 1–6.

[24] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. 2018. Industrial Internet of Things: Challenges, Opportunities, and Directions. *IEEE Transactions on Industrial Informatics* 14, 11 (2018), 4724–4734. https://doi.org/10.1109/TII.2018.2852491

[25] Giacomo Tanganelli, Carlo Vallati, and Enzo Mingozzi. 2018. *Ensuring Quality of Service in the Internet of Things*. Springer International Publishing, Cham, 139–163. https://doi.org/10.1007/978-3-319-58190-3_9

[26] Transforma Insights; Exploding Topics. 2023. Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2023, with forecasts from 2022 to 2030 (in billions) [Graph]. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[27] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 30, 1 (Mar. 2016). https://doi.org/10.1609/aaai.v30i1.10295

[28] Ying Wang, Xiangming Dai, Jason Min Wang, and Brahim Bensaou. 2019. A Reinforcement Learning Approach to Energy Efficiency and QoS in 5G Wireless Networks. *IEEE Journal on Selected Areas in Communications* 37, 6 (2019), 1413–1423. https://doi.org/10.1109/JSAC.2019.2904365

[29] Witold Wydmański and Szymon Szott. 2021. Contention Window Optimization in IEEE 802.11ax Networks with Deep Reinforcement Learning. In *2021 IEEE Wireless Communications and Networking Conference (WCNC)*. 1–6. https://doi.org/10.1109/WCNC49053.2021.9417575

[30] XiPeng Xiao. 2008. Chapter 13 - QoS in Wireless Networks. In *Technical, Commercial and Regulatory Challenges of QoS*, XiPeng Xiao (Ed.). Morgan Kaufmann, Boston, 225–246. https://doi.org/10.1016/B978-0-12-373693-2.00013-6

[31] Kamran Zia, Alessandro Chiumento, and Paul J. M. Havinga. 2022. AI-Enabled Reliable QoS in Multi-RAT Wireless IoT Networks: Prospects, Challenges, and Future Directions. *IEEE Open Journal of the Communications Society* 3 (2022), 1906–1929. https://doi.org/10.1109/OJCOMS.2022.3215731

A DECLARATION OF AI AND AI-ASSISTED TECHNOLOGIES IN THE WRITING PROCESS

During the preparation of this work, the author used OpenAI's ChatGPT in order to improve the author's understanding of small sections of the theory and format tables for LaTeX. After using this tool/service, the author reviewed and edited the content as needed and takes full responsibility for the content of the publication.

Furthermore, during the preparation of this work, the author used Grammarly in order to improve the author's grammar and spelling. After using this tool/service, the author reviewed and edited the content as needed and takes full responsibility for the content of the publication.

B TOOLS USED

- Overleaf
- Visual Studio Code
- Grammarly
- Google Sheets
- ChatGPT
- LibreOffice Calc
- Ubuntu 22.04