DEEP REINFORCEMENT LEARNING BASED ACTIVE QUEUE MANAGEMENT FOR IOT NETWORKS

by

Minsu Kim

B.Eng., Yonsei University, the Republic of Korea, 2017

A Thesis

Presented to Ryerson University

in partial fulfilment of the

requirements for the degree of

Master of Applied Science

in the program of

Computer Networks

Toronto, Ontario, Canada, 2019

Author's Declaration

Abstract

Deep Reinforcement Learning based Active Queue Management for IoT Networks

©Minsu Kim, 2019

Master of Applied Science
Computer Networks
Ryerson University

Internet of Things (IoT) has pervaded most aspects of our life through the Fourth Industrial Revolution. It is expected that a typical family home could contain several hundreds of smart devices by 2022. Current network architecture has been moving to fog/edge architecture to have the capacity for IoT. However, in order to deal with the enormous amount of traffic generated by those devices and reduce queuing delay, novel self-learning network management algorithms are required on fog/edge nodes. For efficient network management, Active Queue Management (AQM) has been proposed which is the intelligent queuing discipline. In this paper, we propose a new AQM based on Deep Reinforcement Learning (DRL) to handle the latency as well as the trade-off between queuing delay and throughput. We choose Deep Q-Network (DQN) as a baseline of our scheme, and compare our approach with various AQM schemes by deploying them on the interface of fog/edge node in IoT infrastructure. We simulate the AQM schemes on the different bandwidth and round trip time (RTT) settings, and in the empirical results, our approach outperforms other AQM schemes in terms of delay and jitter maintaining above-average throughput and verifies that DRL applied AQM is an efficient network manager for congestion.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Alagan Anpalagan for giving me lots of opportunities and support, from leading me into Canada to completing my studies with meaningful results. Without his trust, help and guidance, I would not have completed my master's studies.

Also I would like to extend my gratitude to the Computer Networks Program and the School of Graduate Studies at Ryerson University which provided me with a chance to pursue my master's degree and all kinds of support for my studies.

I am grateful to Prof. Muhammad Jaseemuddin, Bobby Ma, and Javad Alirezaie for being members of my defense committee and giving me constructive advice for a more solid thesis.

It was my pleasure to be a member of WINCORE Lab and spend time with great colleagues and fellows, and I have felt their warm hearts and a friendly environment.

In addition, I would like to thank to my family for their trust and dedication during my master's studies, and my friends from all communities for their pray and encouragement.

Last but certainly not least, my special thanks go to my significant other, Buon, who is always a constant source of love and motivation for my life.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

6LoWPAN	IPv6 over Low-power wireless personal area network
APs	Access Points
AQM	Active Queue Management
CDF	Cumulative distribution function
CoDel	Controlled Delay
CSMA	Carrier-sense Multiple Access
DL	Deep Learning
DNN	Deep Neural Network
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
FAN	Fog access node
FIFO	First-In-First-Out
FQ-CoDel	FlowQueue-Controlled Delay
HTTPS	Hyper Text Transfer Protocol Secure
HVFT	High Volume Flexible Time
IETF	Internet Engineering Task Force
IoT	Internet of Things
MDP	Markov decision process
ML	Machine Learning
MLP	Multilayer Perceptron
P2P	peer-to-peer
P-FIFO	Priority-FIFO-fast
PIE	Proportional Integral controller Enhanced

QoS	Quality of Service
RED	Random Early Detection
RL	Reinforcement Learning
RTT	Round Trip Time
SDN	Software Defined Network
TCP	Transmission Control Protocol
TD	Temporal Difference
UDP	User Datagram Protocol

Chapter 1

Introduction

1.1 Background

The concept and definition of Internet of Things (IoT) is derived from a connectivity of every machine device on the accelerative convergence of many technologies such as wireless technologies, embedded systems, big data analysis, and real-time processing for smart automation toward *future Internet*. International Data Corporation (IDC) expects that worldwide IoT market will grow up to \$1.7 trillion in 2020 [1]. With the high demand for IoT devices such as smart hubs, cameras, and healthcare devices, Gartner predicts that a typical family home could contain more than 500 IoT devices by 2022 composing smart cities [2]. To deal with the data from IoT devices processed by cloud computing efficiently, fog/edge computing architecture has been considered these days.

1.1.1 Fog/Edge Computing Architecture for Internet of Things

Following the phenomenon of IoT generation, current cloud computing architecture is moving to fog/edge computing architecture to carry smart homes, and further smart cities. This architecture deploys edge devices to carry out the considerable amount of computation and storage locally by reducing the latency between IoT devices and servers, and improve the

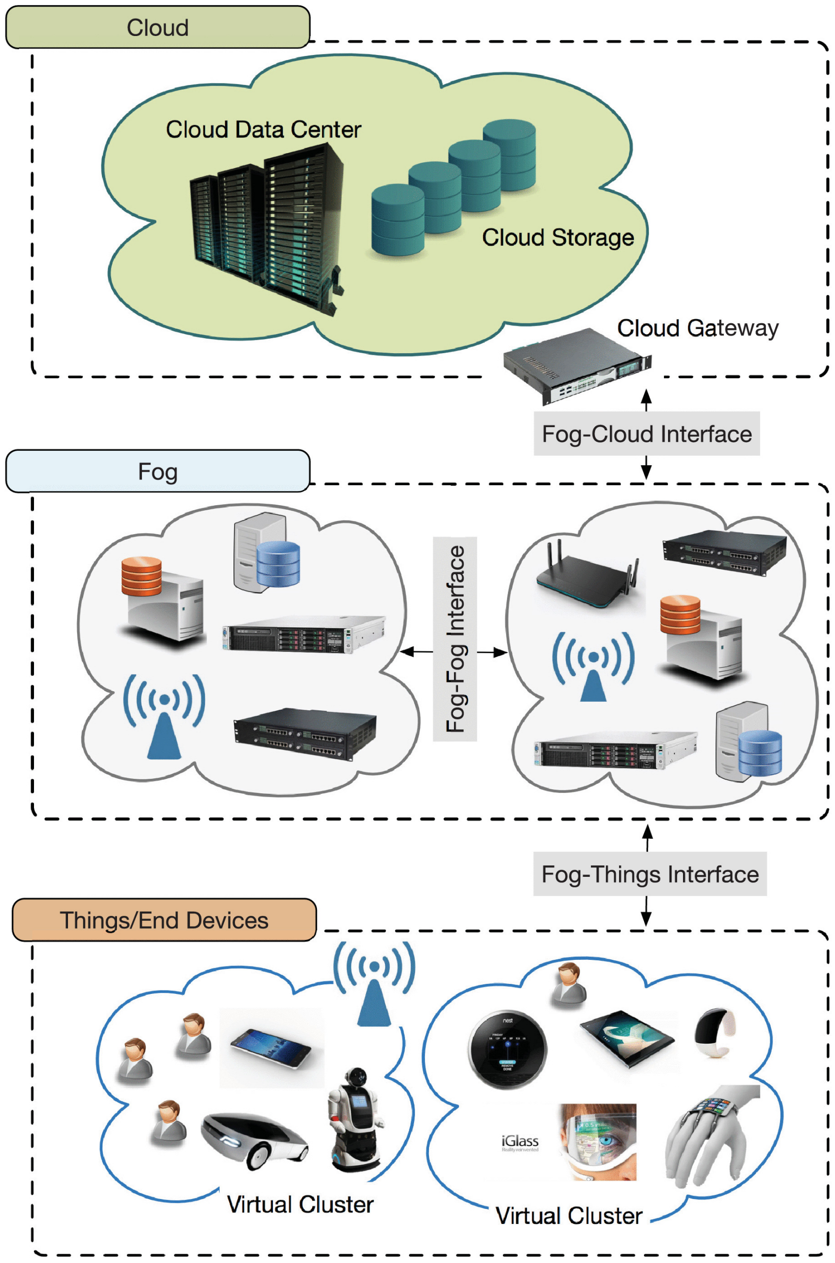


Figure 1.1: 3-tier fog computing architecture

user's experience and resilience of services in the long run [3]. Figure 1.1 describes the basic fog/edge computing architecture concisely, and it has three tiers as follows [4]:

- Tier-1 Things/End Devices : Devices of tier-1 contain IoT devices such as sensors and end user (EU)'s mobile devices including smart phones, tablets, and smart watch. These end devices are also known as Terminal Nodes (TNs).

- Tier-2 Fog : The fog/edge nodes of tier-2 consist of the network router, switch, gateway, and Access Points (APs). In fog/edge computing architecture, these devices have the function of computation and storage.
- Tier-3 Cloud : This tier includes usual cloud servers and data centers which are capable of providing sufficient storage and computing resources.

In the middle of the architecture, the fog/edge nodes are required to bear and accommodate traffic generated by both of the cloud servers and end devices through efficient network scheduling or resource management.

1.1.2 Active Queue Management for Internet of Things

Although IoT devices do not have to transmit entire packets to cloud servers through the edge gateway, edge devices suffer from traffic generated by numerous IoT devices. For efficient management when it comes to queuing networks, Active Queue Management (AQM) has emerged as the intelligent network managing mechanism to selectively transmit and receive packets [5]. Unlike passive queue such as First-In-First-Out (FIFO), AQM introduces the intelligent drop of network packets to reduce network congestion by tuning AQM's parameters such as packet-drop probability adapting to the state. Widely used and recommended AQM schemes by Internet Engineering Task Force (IETF) [6] include Random Early Detection (RED) and Controlled Delay (CoDel) that will be introduced in chapter 2.

1.1.3 Reinforcement Learning for Network Management

In recent years, Machine Learning (ML) has become an important technology and grown up in the industries and our quality of life. And especially, Deep Learning (DL) appears to be outperforming among various ML approaches in various fields such as efficient data coding and modeling objects (unsupervised learning) as well as normal classification and prediction jobs (supervised learning) [7]. DL also has been applied to Reinforcement Learning (RL), which is a category of ML aiming at how software agent learns optimal actions on certain states to take the maximum cumulative reward [8]. Deep Reinforcement Learning (DRL) is DL applied RL which means that software agent's function approximator deploys Deep Neural Network (DNN) as its baseline, and it has also demonstrated greater performance on RL based applications such as playing games and motion control. Recently, the domain or application of state-of-the-art DRL schemes has been extended to networking fields such as network resource management [9].

1.2 Motivation and Objective

Since the state transition of network environment is discrete, this discrete nature is suitable for being represented as a finite Markov Decision Process (MDP) which is introduced in chapter 3 in detail. Because the finite MDP is the main principle of reinforcement learning, we are motivated by this point, in addition to the fact that DRL has shown remarkable performance on highly stochastic environments, and IoT infrastructure consists of numerous various types of end devices causing stochastic state transition. Our objective is to use deep learning to AQM for the first time as well as designing and implementing a fine state-aware AQM scheme on the interface of fog/edge nodes against the transilient IoT environment. Since fog/edge nodes have enough computing power, training the environment through DNN is not an issue.

1.3 Thesis Contributions

To enhance active queue management, we propose a deep reinforcement learning applied AQM scheme for efficient network management and study the trade-off between queuing delay and throughput, while maintaining Quality of Service (QoS) in terms of low jitter. Our system is designed based on Deep Q-Network (DQN) including main Q-network and target network trained using experience replay. It selects a packet drop or non-drop action at the packet departure stage depending on the current state consisting of current queue length, dequeue rate, and queuing delay. After an action is selected, a reward is calculated based on the several elements that will be clarified in chapter 3 in detail. All experiments are conducted on IoT based topology.

The main contributions of this thesis can be summarized as follows:

- Introduction of DRL applied AQM scheme for efficient network management to study the trade-off between queuing delay and throughput with maintained QoS such as jitter
- Implementation of IoT testbed which contains 21 different real IoT devices' characteristics in seven different IoT device categories
- Analysis of the experimental results, verification of our proposed scheme' s performance, and providing of design guidelines for DRL based AQM

1.4 Thesis Organization

The rest of this thesis is organized as follows. In **Chapter** 2, related works including widely used AQM schemes and ML applied AQM algorithms are described. The overview and detail design of our AQM scheme are presented in **Chapter** 3. Next, our IoT testbed including topology, scenarios and IoT characteristics is explained in **Chapter** 4. Finally, we evaluate and analyze our experiment results in **Chapter** 5 and we conclude the thesis with summary and future work in **Chapter** 6.

Chapter 2

Literature Review of Active Queue Management

Active Queue Management (AQM) is one of the intelligent network management methods deployed in the network interface of a router or switch [5]. AQM has parameters such as packet-drop probability tunable by both of users and queue itself adjusting to the environment. Unlike the traditional queue management methods such as drop-tail and FIFO, AQM drops packets before the queue is full to avoid the queue overflow by bursty flows as known as *bufferbloat*. Consequently, it causes low throughput and high latency and jitter, which is the variation in the latency or queuing delay on packet flows and that is why AQM has been developed and used widely. However, while coping with this problem, early AQM schemes had a drawback which is the parameter tuning since the performance of AQM highly depends on the base parameters setup. There have been attempts for either minimum tuning of parameters or self parameter tuning. In this chapter, we introduce the widely-used AQM and research about ML applied AQM schemes as well as the network scheduling algorithms. We also review AQM performance studies on IoT testbed.

2.1 Active Queue Management Schemes

Traditional queue management cannot deal with rapidly changing queue's state such as oversized buffers which is *bufferbloat*. Recent IETF has expressed interest in up-to-date AQM schemes to deal with this problem [6]. Here, we briefly introduce and summarize three recognized AQM schemes: RED [10], CoDel [11], and PIE [12] recommended by IETF.

2.1.1 Random Early Detection (RED)

RED is one of the early classic AQM schemes for congestion avoidance, and basically it has three parameters to adapt to the state: two thresholds th_{min} and th_{max}, and packet marking probability p. If the average queue size L_{avg} is less than th_{min}, packet is accepted to enqueue. If queue length is in between th_{min} and th_{max}, RED calculates p to mark the packet by dropping it or by modifying the packet header depending on the transport protocol. If L_{avg} is greater than th_{max}, RED drops the packet rather than modifying the packet header to control L_{avg}. Variants of RED have been developed including Adaptive RED (ARED) and Weighted RED (WRED) for better QoS [13].

2.1.2 Controlled Delay (CoDel)

CoDel is packet-sojourn time based AQM, and tracks the (local) minimum queuing delay experienced by packets. The minimum packet-sojourn time can be decreased only when a packet is dequeued; and to maintain the updated minimum value, CoDel measures the value within a certain interval T_{int}, set to 100ms by default. CoDel assumes a target delay T_{target} and when the queuing delay exceeds T_{target} during T_{int}, a packet is dropped and the next drop time is set by a control law defined as follows:

$$t_{new} = t_{curr} + \frac{T_{int}}{\sqrt{N_{drops}}}, \tag{2.1}$$

where t_{new} is new next drop time, t_{curr} is current next drop time and N_{drops} is the number of drops since the dropping state was entered. When the queue delay is below T_{target}, the controller stops dropping the packets. A distinct feature of CoDel is that it drops packets at the packet departure stage. We deal with FlowQueue-CoDel (FQ-CoDel) [14] as well, which classifies flows into one of 1024 sub-queues by hashing the 5-tuple of source and destination IP addresses, source and destination port numbers, and protocol number. Each sub-queue is implemented based on CoDel, and FQ-CoDel deploys a byte-based deficit round-robin (DRR) mechanism to serve sub-queues. FQ-CoDel has a distinct parameter *quantum* which is the number of bytes each queue keeps to decide to dequeue on each round of the scheduling algorithm. The default size of *quantum* is set to 1514 bytes, which corresponds to the Ethernet maximum transmission unit (MTU) plus the hardware header length of 14 bytes.

2.1.3 Proportional Integral controller Enhanced (PIE)

PIE is a burst-tolerance AQM scheme. Similar to CoDel, PIE drops the packet depending on the queuing delay, but it is more delicate with additional parameters. For every T_{update} interval, PIE calculates packet-drop probability p based on queuing delay, α, and β where α and β are static configured parameters, set to 0.125 and 1.25 by default which determine the final balance between latency offset and latency jitter. When it comes to PIE, queuing delay is estimated by dequeue rate as follows:

$$cur\_del = \frac{qlen}{avg\_drate}, \tag{2.2}$$

where *cur_del* is the current queuing delay, *qlen* is current queue length in bytes, and *avg_drate* is the average dequeue rate. To overcome rate fluctuation which is a common issue in wireless networking, PIE measures the dequeue rate periodically (refer to Appendix A for details). PIE handles bursts as well by deploying a maximum bursting allowance parameter B_{max} to allow bursts bypassing the random drop process.

By using AQM schemes, the *bufferbloat* and network congestion have been alleviated comparing with using traditional passive queues, but it is still possible to improve the AQM combining different technologies such as Machine Learning (ML) dealt in the next section.

2.2 Machine Learning Applied AQM and Network Traffic Scheduling Algorithms

Machine learning (ML) is a study of data analysis that approximates an algorithm or mathematical model to improve performance of a certain task gradually. As one of ML methods, reinforcement learning (RL) is defined as training an agent to make it select an optimal action for a certain task by giving a reward. In networking field, the agent's "certain task" can be expressed as a network scheduling or resource management. There have been several attempts for applying ML and RL techniques to not only in AQM but also in network traffic and resource control.

In [15], Bouacida et al. implemented *LearnQueue* AQM algorithm based on reinforcement learning for wireless networking. By manipulating a buffer size dynamically using Q-Learning in a certain interval, they update the Q-table and optimize the policy of Q-function, but they tested their algorithm with only two and three nodes deployed scenarios. Bisoy et al. of [16] suggested the AQM scheme based on a shallow neural network of one hidden layer consisting of three neurons to deal with non-linearity of networking system and to reduce queuing delay, but their work did not deal with trade-off between throughput and delay performance. In [17], Vucevic et al. proposed Reinforcement Learning-Queuing Delay Limitation (RL-QDL) AQM algorithm. The RL agents receive topology information from bandwidth broker that manages the resource handling and QoS provisioning depending on the achievement of QoS expectations in egress routers (ER). It assumes class based queuing (CBQ) by supporting three different classes: expedited forwarding (EF), assured forwarding (AF), and best effort (BE) traffic in order to provide end-to-end QoS to users with different types of services.

In terms of the network scheduling algorithms, Chen et al. proposed DRL based optimized computation offloading policy by deploying a double DQN on the edge node [18]. Comparing with baseline algorithms, their approach indicated the optimal trade-off between the task delay and drops. In [19], Chinchali et al. investigated real data of cellular network traffic and introduced DRL based scheduler focusing on High Volume Flexible Time (HVFT) traffic including software and data updates to mobile IoT devices, large data transfer from IoT sensors to cloud servers, and pre-fetched ultra-high quality and bit-rate video. As a baseline of DRL model, authors used Deep Deterministic Policy Gradient (DDPG) algorithm to train actor and critic networks, and the RL agent resides at the cell tower for HVFT control. In [20], Xu et al. applied DRL to network traffic engineering by deploying actor-critic method with prioritized experience replay. Authors compared their algorithm with the widely used baseline solutions such as Shortest Path (SP), Load Balance (LB), and Network Utility Maximization (NUM), and verified their model performs better than given baseline solutions.

Although there have been meaningful works, current ML based AQM works do not consider stochastic environments such as IoT networks, or introduced network schedulers work on the task or job processing level, not packet processing level. In the next section, we review the research about traffic management for IoT applications as well.

2.3 Traffic Management for IoT Applications

Along with the growth of IoT markets and technologies, traffic scheduling and management for IoT applications have been flourished recently.

Kua et al. analyzed performances of several AQM schemes such as FlowQueue CoDel (FQ-CoDel) and PIE assuming the IoT enabled smart home testbed including low and high-rate IoT applications, video call traffic, bulk file transfers, and multimedia streaming server [21]. As the indicators of AQM schemes' efficiency, authors described throughput and round trip time (RTT). There were efforts to reduce IoT service delay for edge computation offloading policy as well [22], [23]. In [22], Zhang et al. consider G/M/1 queue as a task buffer at the edge computing server in First Come First Served (FCFS) and non-preemptive manner. The authors used Locality-First policy and probability-based policy by setting a probability parameter to make an offloading decision as the offloading policies. Yousefpour et al. used load sharing approach between edge nodes to reduce the IoT service delay as an edge offloading policy [23]. They considered the different request types from IoT devices as well as the queue length which is the task buffer for load sharing.

In [24], Zhao et al. introduced multi-tier in fog/edge layer for desired throughput-delay trade-off per user consisting of fog access node (FAN) which caches a subset of popular contents and fog control node (FCN) connected to FAN through the backhaul connections having higher computing and storage capacity than FAN. Zheng et al. investigated multiple service frequency constraints in wireless channels [25]. The authors scheduled packets arrived at links using round-robin and maximum-weight scheduling. The links are categorized into different stages indicating the priority level, and on the same stage, maximum-weight scheduling gives a priority to a link with longer queue size to be served.

In [26], Bhandari et al. applied priority queues to the cluster head to prioritize and aggregate the incoming packets. The proposed mechanism guaranteed QoS requirements in terms of latency and reliability. Al-Kashoash et al. proposed optimization-based hybrid congestion alleviation (OHCA) which combines both traffic and resource controls in the IPv6 over Low-power wireless personal area network (6LoWPAN) to utilize the network resources effectively [27]. Zhao et al. approached in a different way from above works which is laying cloudlets down on the optimal places to minimize the access delay using Software Defined Network (SDN) [28]. The control plane on SDN controllers assigns cloudlets to APs based on the average cloudlet access delay among all placement cases.

2.4 Summary

In this chapter, we introduced widely used AQM schemes including RED, PIE, and CoDel and reviewed related works for network traffic and resource management based on ML techniques. Unlike the research about traffic management for IoT applications dealt in section 2.3 which is task scheduling or management, our approach deals with network flow of packets itself at the physical queue level in the network device's interface. To the best of our knowledge, this is the first work to design deep learning based AQM as well as DRL. Table 2.1 tabulates the summary of related works, and in the next chapter, we introduce the design of our system model in detail.

Table 2.1: Summary of related works.

Publication	**Title**	**Scope**				
		Network management		**Machine Learning**		**IoT Application**
		AQM	**Other approaches**	**Deep Learning**	**Other ML methods**	
[15]	Practical and Dynamic Buffer Sizing using LearnQueue	✓			✓	
[16]	Design of an active queue management technique based on neural networks for congestion control	✓			✓	
[17]	Reinforcement Learning for Active Queue Management in Mobile All-IP Networks	✓			✓	
[18]	Optimized Computation Offloading Performance in Virtual Edge Computing Systems via Deep Reinforcement Learning		✓	✓		✓
[19]	Cellular Network Traffic Scheduling With Deep Reinforcement Learning		✓	✓		✓
[20]	Experience-driven Networking: A Deep Reinforcement Learning based Approach		✓	✓		
[21]	Using Active Queue Management to Assist IoT Application Flows in Home Broadband Networks	✓				✓
[22]	Theoretical Analysis on Edge Computation Offloading Policies for IoT Devices		✓			✓
[23]	On Reducing IoT Service Delay via Fog Offloading		✓			✓
[24]	FEMOS: Fog-Enabled Multitier Operations Scheduling in Dynamic Wireless Networks		✓			✓
[25]	Scheduling Flows With Multiple Service Frequency Constraints		✓			✓
[26]	Latency Minimization in Wireless IoT Using Prioritized Channel Access and Data Aggregation		✓			✓
[27]	Optimization-Based Hybrid Congestion Alleviation for 6LoWPAN Networks		✓			✓
[28]	Optimal Placement of Cloudlets for Access Delay Minimization in SDN-Based Internet of Things Networks		✓			✓
This work	**Design of Deep Reinforcement Learning based Active Queue Management**	✓		✓		✓

Chapter 3

System Model

In this chapter, we first give an explanation of Deep Q-Network (DQN) which is the baseline of our system. Then we describe the design of our system in terms of the state, action, and reward, and the algorithm is followed by focusing on how to give a reward to the agent in detail.

3.1 Deep Q-Network

In reinforcement learning, the software agent interacts with the environment ε in a sequence of discrete time step $t = 1, 2, 3, ...$, and generally ε is stochastic. At each time step t, the agent selects an action $a_t \in A(s)$ from the set of action space on the given state $s_t \in S$. After taking an action a_t, the agent receives a reward $r_t \in R \subset \mathbb{R}$ and observes a new state s_{t+1}. By iterating this interaction, each observation in the sequence or trajectory is suitable for being expressed as a finite Markov Decision Process (MDP) that begins like this:

$$s_1, a_1, r_1, s_2, a_2, r_2, s_3, a_3, r_3, ...$$

The transition of MDP satisfies Markov property that refers to the memoryless property of a stochastic process. In the stochastic process, future states of the process only depend on the present state, not on the whole trajectory of events. Figure 3.1 shows the finite transition of MDP [8]. On the network queue, as packets come into the queue, the agent can observe the current queue state such as queue length and take an action whether to drop the packet or not. The queuing delay is affected by the action, and we can derive a reward as well as observe a next state. Therefore, it can be considered as MDPs, and we can apply RL to the sequence representation at each time step t.

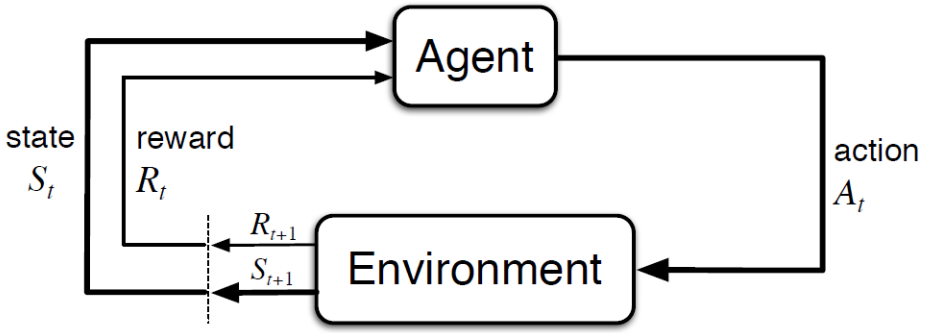


Figure 3.1: Markov Decision Process.

The two most significant features of RL are trial-and-error search and delayed reward which means that the next state's immediate reward is also considered on the current state [8]. Ultimately with these features, the agent selects an optimal action on any given state by maximizing the total reward. In order to evaluate the value of actions, Q-learning has been used as a common method of Temporal Difference (TD) learning that updates the model at the end of each time step [29]. Q-learning finds an optimal policy for expected cumulative future reward on the current state. Action-value function known as Q-function, gives a value for every action of each state, and is updated by the following equation:

$$Q(s_t, a_t) \leftarrow (1-\alpha) * Q(s_t, a_t) + (r_t + \gamma * \max_{a'} Q(s_{t+1}, a')), \tag{3.1}$$

where r_t is a reward after taking an action a_t on state s_t, a' is an expected action to maximize the Q-value, α is learning rate, and γ is discount factor for future reward. Since we design discrete actions of packet drop/non-drop actions, our policy $\pi = p(a|s)$ is deterministic that maps each state s to an action a. The goal of the agent is to select an action to maximize the cumulative future reward through Q-function, so the optimal Q-function can be expressed as follows:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}_{s_t \sim \varepsilon}\big[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... \, |s_t = s, a_t = a, \pi\big], \tag{3.2}$$

which is the maximum expected cumulative reward acquired by the policy π after taking an action a on the state s. Since we focus on deep neural network (DNN) as a function approximator, it outputs the predicted Q-value $Q(s, a|\theta)$ where θs are weights of neurons of DNN. In order to minimize the difference between predicted Q-value and optimal Q-value, loss is defined as:

$$L(\theta) = \min_{\theta} \sum_{t=1}^{T} \left[\Big(Q(s_t, a_t|\theta) - (r_t + \gamma * \max_{a'} Q(s_{t+1}, a'|\theta)) \Big)^2 \right], \tag{3.3}$$

However, when reinforcement learning uses a neural network as a function approximator, it has a disadvantage that is unstable or diverges to train Q-function. One reason of the instability is that the neural network receives highly correlated data as sequential inputs in the observation, so it causes over-fitting or falling into local minimum. Another reason of divergence is due to non-stationary target which means that both of the predicted Q-value and optimal Q-value depend on weights θ so when the neural network is trained, target value is also changed and it is difficult to converge. To overcome these issues, DQN has been developed, which is the baseline model of our system [30]. There are two main features of DQN: experience replay and network separation.

In order to reduce correlation between observations, the concept of experience replay is used. Instead of training the neural network in order of sequence, we store the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time step t into the replay memory $D_t = \{e_1, e_2, ..., e_t\}$. To update the neural network, mini-batches of experiences $(s, a, r, s') \sim U(D)$ are selected randomly in uniform distribution from the replay memory. The way of training the neural network at iteration i uses a similar loss function to the equation (3.3):

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(Q(s,a|\theta_i) - (r + \gamma * \max_{a'} \hat{Q}(s', a'|\theta_i^-))\right)^2\right] \tag{3.4}$$

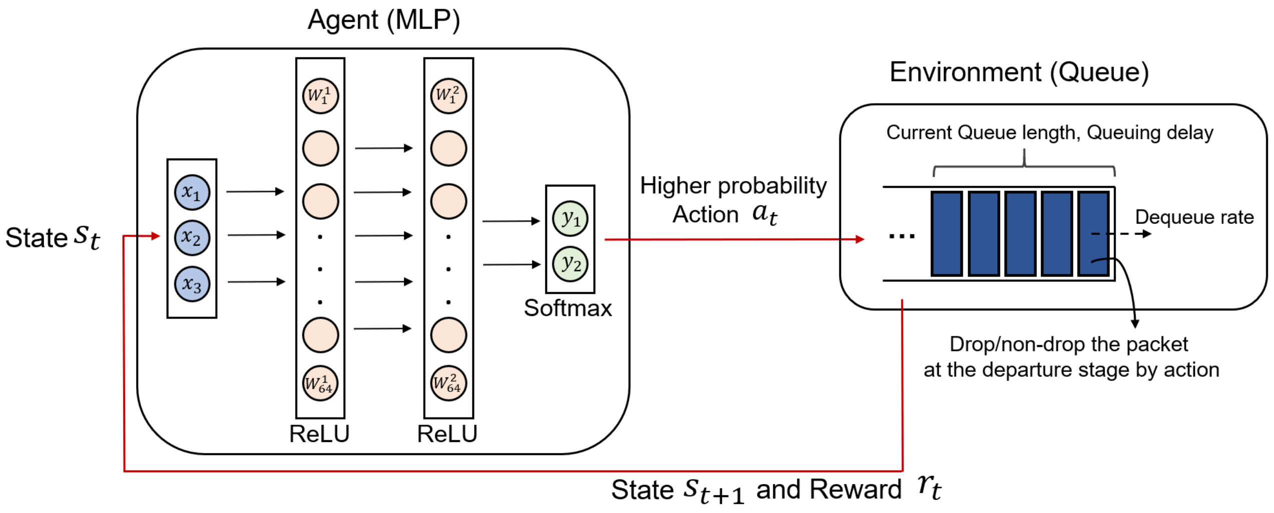
By deploying two networks (main Q-network Q and target network $\hat{Q}$), we can set the stationary target. The target network parameters θ_i^- are updated by Q-Network parameters θ_i periodically, not in every step. With these features, DQN has shown outstanding performances comparing with all previous Q-learning based algorithms as will be seen in chapter 5.

3.2 DQN based AQM System Design

In this section, we introduce the design of state, action, and reward function for our DQN based AQM algorithm.

3.2.1 Process of Selecting Action

With respect to the state of RL, we consider three elements: current queue length in packets L, dequeue rate R_{deq}, and queuing delay d and we use PIE's dequeue rate calculation method [12]. At each time step t, state s_t is defined as $s_t = \{L_t,\ R_{deq,t},\ d_t\}$ which is an input of multilayer perceptron (MLP) consisting of two hidden layers of 64 neurons for each layer. For selecting an action, main Q-network is used and it returns two probabilities as an output (drop/non-drop probability). Following the higher probability, it decides a packet drop/non-drop action a_t, and the action occurs on the packet departure stage of queue as if CoDel's dropping action [11]. In order to find a better action on a certain state, we use explore/exploit strategy which means that the agent takes an action based on its own selection (exploit), or sometimes takes a random action uniformly based on a certain probability (explore). For the explore/exploit strategy, we use ϵ-greedy algorithm starting from a highly random action probability. At the first episode of the network simulation, the exploring probability is set to 90% based on the round of episode and it dwindles down to 0% through the episodes. The agent takes an action in periodic interval T_{int} only when there is queuing delay in the queue. Figure 3.2 describes the process of selecting an action.



Agent (MLP)
State s_t
x_1
x_2
x_3
W_1^1
W_{64}^1
W_1^2
W_{64}^2
ReLU
ReLU
y_1
y_2
Softmax
Higher probability Action a_t
Environment (Queue)
Current Queue length, Queuing delay
Dequeue rate
Drop/non-drop the packet at the departure stage by action
State s_{t+1} and Reward r_t

Figure 3.2: Process of selecting an action.

3.2.2 Reward Engineering

After taking an action, the RL agent waits for next state s_{t+1} during the interval T_{int}. The selected action is evaluated by a reward function. The most important point of designing the reward function is to optimize the trade-off between queuing delay and drop-rate as well as to avoid infinite packet drop state or non-drop state. We refer *learnQueue*'s reward function as a baseline [15]. There are two main components for a reward: *delay_reward* and *enq_reward* for queuing delay and packet drop-rate respectively. The *delay_reward* is defined as:

$$delay\_reward = \delta * (delay\_ref - curr\_delay), \tag{3.5}$$

where *delay_ref* is the desired queuing delay, *curr_delay* is the current queuing delay defined in the equation 2.2, and δ is a scaling factor between *delay_reward* and *enq_reward*. The *enq_reward* is defined as:

$$enq\_reward = (1 - \delta) * (min\_delay - delay\_ref) * enq\_rate, \tag{3.6}$$

where *enq_rate* is enqueue rate during T_{int}, and it is calculated as:

$$enq\_rate = \frac{N_{enq}}{N_{enq} + N_{drops}}, \tag{3.7}$$

where N_{enq} is the number of enqueued packets and N_{drops} is the number of dropped packets. When N_{enq} is 0, *enq_rate* is set to 0. Depending on the scaling factor δ, we can balance the importance between the queuing delay and packet drop-rate by the agent. Unlike *learn-Queue*'s reward function that defines *max_delay* which is the maximum time required to drain the queue depending on the AP's physical data rate, we define *min_delay* which is delay when the link between the edge device and cloud gateway is fully utilized as follows:

$$min\_delay = \frac{L_{byte}}{R_{phy}}, \tag{3.8}$$

where L_{byte} is the current queue length of the edge device in bytes, and R_{phy} is the physical bandwidth (data rate) of peer-to-peer (P2P) link connected to the edge device. Finally, the reward is calculated as the sum of both components:

$$reward = \text{clip}(delay\_reward + enq\_reward, -1, +1). \tag{3.9}$$

To avoid the divergence of a reward and regularize it, we clip the reward to the constant minimum and maximum values corresponding to -1 and +1.

3.2.3 Training Process

Since we use DQN as our model, the agent stores an experience $e_t = (s_t, a_t, r_t, s_{t+1})$ in tuple format to the replay memory at each time step t. Once the number of experiences in replay memory is over mini-batch size, the agent randomly picks samples of the experiences from the memory in uniform manner. Following equation (3.4), the agent minimizes the temporal difference error (TD-error) using mean square error (MSE) loss function. We apply the rectified linear unit (ReLU) activation function defined as $f(x) = \max(0, x)$ at each hidden layer, and softmax function at the output layer to convert outputs into action probabilities. At the first episode, we initialize the weights of MLPs using *Xavier initializer* which is assigning the weights of the layers from a Gaussian distribution [31], and optimize the loss using *Adam optimizer* [32] to train the model. Figure 3.3 shows the flowchart of DQN based AQM training process. In the flowchart, t_{curr} is current time step t, and C is target update step to update the target network periodically. **Algorithm 1** describes the pseudocode of DQN based AQM training process. We follow the principle of DQN introduced in [30], but since there is no concept of "game over" in network simulation, the discounted future reward is always added to the immediate reward. To avoid infinite packet drop/non-drop action on certain states, we control the reward function. After giving the reward, we initialize the dequeue rate depending on the queuing delay following the PIE's dequeue rate calculation scheme [12]. In the pseudocode, we assume that in time step t, if the selected action is dropping the packet, $a_t = 1$, otherwise, $a_t = 0$. The interval of each time step is T_{int}.

3.3 Summary

In this chapter, we presented our system model in terms of the RL state and action setup, and reward function. Also we introduced which ML techniques we use to our MLPs including the activation functions, loss optimizer, and MLPs' weight initializer. In the next chapter, we introduce how we set up the IoT testbed, assumptions for the experiment, and parameters setup for the topology and AQM schemes.

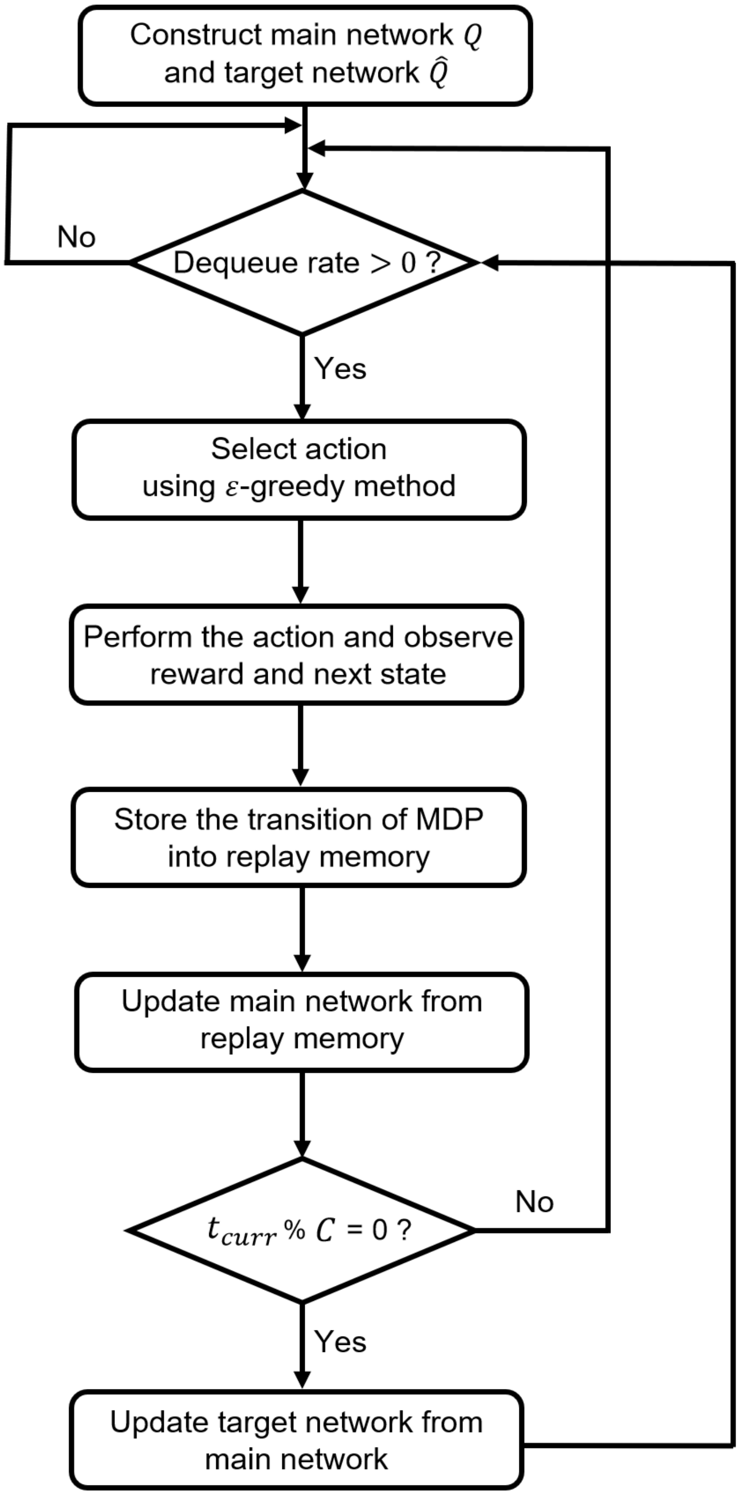

Construct main network Q and target network $\hat{Q}$
No
Dequeue rate > 0 ?
Yes
Select action using ε-greedy method
Perform the action and observe reward and next state
Store the transition of MDP into replay memory
Update main network from replay memory
No
t_{curr} % C = 0 ?
Yes
Update target network from main network

Figure 3.3: Flowchart of DQN based AQM training

Algorithm 1 DQN based AQM training algorithm pseudocode

```
1: for $i = 1 \rightarrow N_{episode}$ do
2:     Initialize main network $Q$ and target network $\hat{Q}$ with normalized weights $\theta$ and $\theta^{-}$
3:     Calculate $R_{deq}$ and $d$ when the packet is dequeued
4:     Call every $T_{int}$ periodically
5:     for $t = 1 \rightarrow T$ do
6:         if $R_{deq} > 0$ then
7:             Observe current state $s_t \leftarrow [L_t,\ R_{deq,t},\ d_t]$
8:             Take a random action in the training mode with random variable $\epsilon$
9:             $\epsilon \leftarrow random()$
10:            if $\epsilon < 1/((i/5) + 1)$ then
11:                Select action $a_t \leftarrow random()$
12:            else
13:                Select action $a_t \leftarrow \mathrm{argmax}_a \mathrm{Q}(\mathrm{s}_t, a; \theta)$
14:            end if
15:            Take action $a_t$ and observe next state $s_{t+1}$ after $T_{int}$
16:            Calculate $enq\_reward$ and $delay\_reward$
17:            if $a_t = 0$ and $min\_delay < delay\_ref$ then
18:                $enq\_reward \leftarrow 0$
19:            end if
20:            if $L_t = 0$ and $a_t = 1$ then
21:                reward $r_t \leftarrow -1$
22:            else
23:                if $L_{t+1} = 0$ and $a_t = 0$ and $enq\_rate = 0$ then
24:                    $r_t \leftarrow -1$
25:                else
26:                    $r_t \leftarrow enq\_reward + delay\_reward$
27:                end if
28:            end if
29:            $r_t \leftarrow \mathrm{clip}(r_t, -1, 1)$
30:            Store experience $e_t \leftarrow s_t, a_t, r_t, s_{t+1}$ in replay memory $D$
31:            Sample random mini-batch of $e_j$ from $D$ uniformly
32:            output $y_j \leftarrow r_j + \gamma * \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^{-})$
33:            $L(\theta) \leftarrow \Big(y_j - Q(s_j, a_j; \theta)\Big)^2$
34:            Perform a gradient descent step on $L(\theta)$ to minimize it w.r.t parameter $\theta$
35:            Update $\hat{Q} \leftarrow Q$ for every $C$ step
36:            if $d_{t+1} < 0.5 * delay\_ref$ and $d_t < 0.5 * delay\_ref$ and $a_t = 0$ and $R_{deq,t+1} > 0$
    then
37:                $R_{deq,t+1} \leftarrow 0$
38:            end if
39:        end if
40:    end for
41: end for
```

Chapter 4

Experiment Setup

In this chapter, we briefly explain the characteristics of IoT, and deal with our testbed, assumptions for the experiment, and how we set up the parameters.

4.1 Characteristics of IoT devices

In order to understand real behaviours of IoT devices, we survey the materials of analysis on smart home/campus network traffic. Authors in [33] measured the IoT network traffic for over three weeks, and clustered their characteristics applying K-means clustering algorithm with over 95% for IoT devices identification. After classification, they investigated the patterns of IoT as well. Their analysis shows that IoT devices remain idle for most of the time in general which means that no packet is exchanged during this interval. Also considering only IoT devices, daily average load is 66kbps which is higly low rate. Another observation is that most of the traffic by IoT is based on TCP, rather than UDP, and more than 50% of TCP port is 443 which is Hyper Text Transfer Protocol Secure (HTTPS) protocol. In [34], authors investigated traffic from smart home consisting of sensors, hubs, plugs, and electronics for 22 days, and it shows similar pattern with [33]'s result.

4.2 IoT Testbed

In order to construct the IoT testbed as close as possible to the real IoT infrastructure, we use real IoT characteristics clustered by [33]. From the clustered attributes, we use five characteristics to implement IoT testbed: Sleep time, Active time, Average packet size, Mean data rate, and Peak/Mean rate. Table 4.1 shows clustering of real IoT attributes, and Table 4.2 describes the measure of attribute clusters. In the topology, all IoT devices are connected to the edge device wirelessly, and the edge device is connected to the cloud gateway on P2P link. The gateway transmits the received packets to destination cloud servers through carrier-sense multiple access (CSMA) channel. For the camera category of IoT attributes, we set User Datagram Protocol (UDP) assuming real-time monitoring traffic, and for other categories, we set Transmission Control Protocol (TCP) assuming widely used HTTPS protocol for IoT to generate traffic. In terms of TCP congestion control algorithm, TCP CUBIC [35] and Proportional Rate Reduction (PRR) recovery algorithm [36] are deployed which are implemented and used by default in Linux kernels. Figure 4.1 depicts our topology design.

4.3 Assumptions and Parameter Setting

Due to the traffic characteristics of IoT such as periodic packet transmission and small size of packets, the queuing delay or packet loss rarely happens in the normal operation of mean data rate. We assume that IoT devices burst with sending packets in peak transmission rate to occur queuing delay and packet loss, and the distance from each IoT device to the edge device is equal and constant. We regulate the round-trip time (RTT) by controlling the baseline delay of P2P link and set the low link capacity between the edge device and cloud gateway to measure the performance of AQM schemes from various perspectives assuming the narrow bandwidth for IoT devices.

All tested AQM schemes are deployed on the interface of edge node connected to the

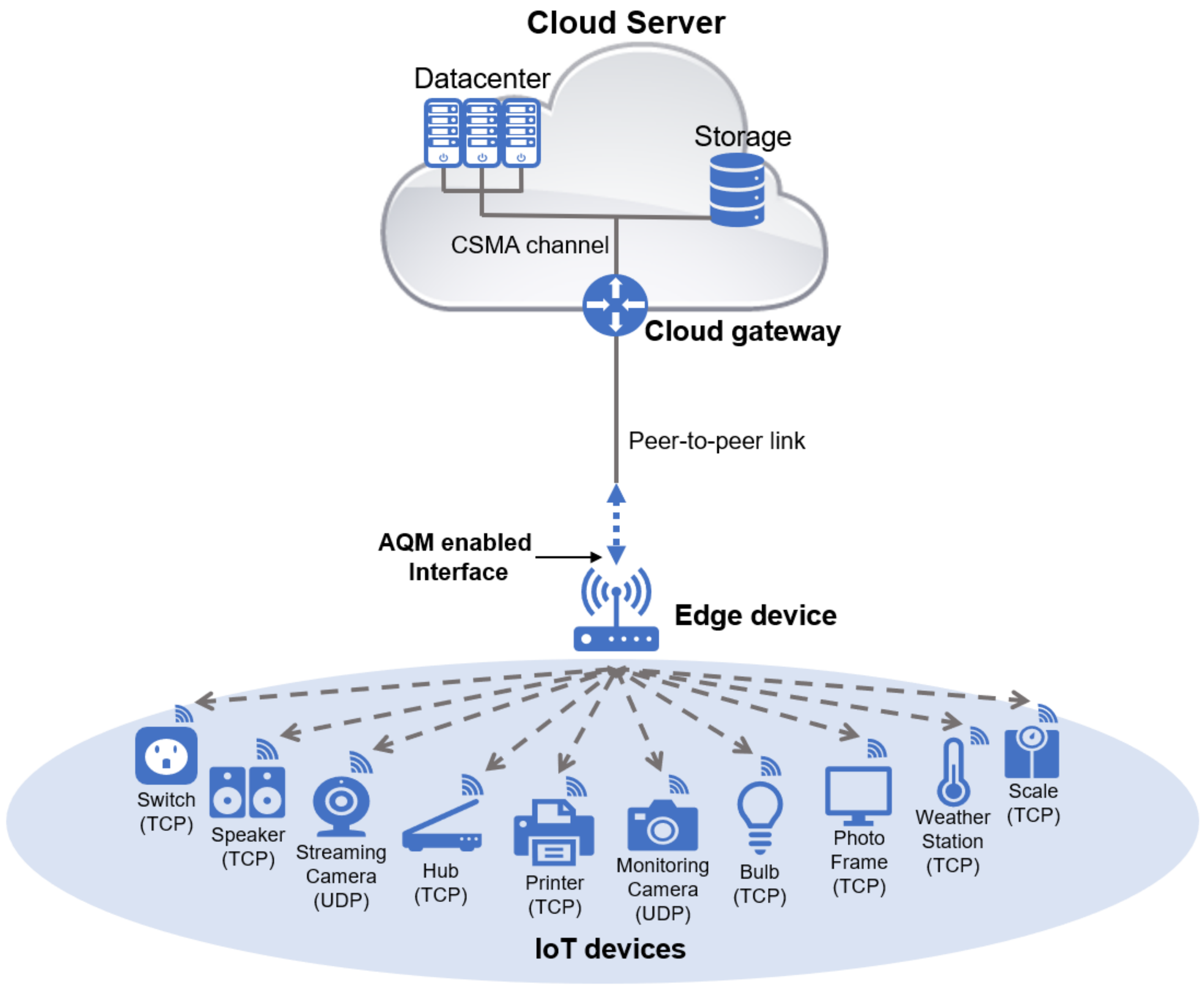


Figure 4.1: Overview of the IoT testbed

cloud gateway since the P2P link between them is set as a bottleneck link. Priority-FIFO-fast queuing discipline (P-FIFO), which has three priority bands based on the packet priority and is the default priority queue on Linux systems, is deployed on all other interfaces. To conduct the experiments, we set different RTTs, P2P link capacity, reward scaling factor δ, and the number of IoT devices connected to the edge device. The basic number of IoT devices that will be tested on the experiment is 21, which is the number of clustered real IoT devices. Also we test on three times the basic number of IoT devices which is 63, to observe the performance of each AQM on more extreme congestion. The testbed is implemented on *ns-3* network simulator [37], and we apply *tiny-dnn* which is a C++ based open source deep learning framework to *ns-3* [38]. Tables 4.3 and 4.4 describe the environmental setup parameters and comparison AQM setup parameters respectively. Mainly we set default parameters for the AQM schemes, but we only adjust PIE's mean packet size and dequeue threshold parameters adapting to the IoT testbed. In Table 4.4, $maxSize$ is maximum queue size, $meanPktSize$ is mean packet size set by user to determine each AQM's behaviours, and $delay\_ref$ is the desired queuing delay in common. In RED, the parameter $LInterm$ is the maximum probability of dropping a packet. In PIE, *dq_threshold* is the dequeue threshold that indicates if there are sufficient data in queue to calculate dequeue rate. Because PIE's authors recommend *dq_threshold* to be set 10 times of higher size than $meanPktSize$, we set *dq_threshold* to 3500 [12]. In CoDel, $minByte$ is the minimum bytes in queue to allow a packet drop. In FQ-CoDel, the variable $Flows$ is the number of sub-queues for classifying flows. Table 4.5 indicates the DQN based AQM algorithm setup parameters.

Table 4.1: Clustering of IoT attributes.

IoT Device Category	**Hubs**		**Cameras**						**Switches & triggers**				**Air quality sensors**		**Healthcare devices**			**Light Bulbs**	**Electronics**		
IoT Device	**Smart Things**	**Amazon Echo**	**Netatmo Welcom**	**TP-Link Day Night Cloud camera**	**Samsung SmartCam**	**Dropcam**	**Insteon Camera**	**Withings Smart Baby Monitor**	**Belkin Wemo switch**	**TP-Link Smart plug**	**iHome**	**Belkin wemo motion sensor**	**NEST Protect smoke alarm**	**Netatmo weather station**	**Withings Smart scale**	**Blipcare Blood Pressure meter**	**Withings Aura smart sleep sensor**	**LiFX Smart Bulb**	**Triby Speaker**	**PIX-STAR Photo-frame**	**HP Printer**
Sleep time	1	1	1	2	1	1	2	1	1	3	2	1	1	1	1	5	1	1	1	1	2
Active time	1	1	1	1	1	1	1	1	1	1	2	1	1	1	2	3	1	1	1	1	1
Avg. Pkt Size	1	2	4	1	4	2	2	1	4	2	1	3	3	3	3	2	2	1	2	3	1
Mean data rate	1	1	2	1	2	1	1	1	3	1	1	3	2	1	1	1	1	1	1	1	1
Peak / Mean rate	1	1	2	1	1	2	1	1	1	1	1	1	1	1	1	1	2	1	2	1	1

4.4 Summary

In this chapter, we introduced our IoT testbed by explaining which inherent characteristics of IoT devices we bring in for the experiments, and parameters we set for the network simulation. In the following chapter, we analyze the empirical results of the simulation, and verify the outstanding performance of our DQN based AQM scheme.

Table 4.2: Measure of attribute clusters.

Clusters	**Sleep time (Sec)**	**Active time (Sec)**	**Avg. Pkt Size (B)**	**Mean rate (Bps)**	**Peak/ Mean rate**
1	4	1	94	462	11
2	66	2	144	2,461	66
3	241	8	234	11,388	229
4	7,985	25	327	42,493	474
5	24,832	34	699	516,540	1,253

Table 4.3: Environmental setup parameters.

Parameter	**Value**
Number of servers in cloud	2
CSMA channel data rate in cloud	100Mbps
P2P link data rate	0.5, 1Mbps
Base RTT on P2P link	0, 10, 20, 50, 100ms
Number of IoT devices	21, 63
Wi-Fi standard	IEEE 802.11n 5GHz
Distance between IoT devices and edge device	30m
Simulation time	100s

Table 4.4: Comparison AQM setup parameters.

AQM	**Parameters**
P-FIFO	$maxSize = 1000$
RED	$maxSize = 25p$ $meanPktSize = 500$ $th_{min} = 5$ $th_{max} = 25$ $LInterm = 50$
PIE	$maxSize = 25p$ $meanPktSize = 350$ $dq\_threshold = 3500$ $\alpha = 0.125$ $\beta = 1.25$ $T_{update} = 0.03$ $delay\_ref = 20ms$ $B_{max} = 0.1$
CoDel	$maxSize = 50000bytes$ $delay\_ref = 5ms$ $T_{int} = 100ms$ $minByte = 50$
FQ-CoDel	$maxSize = 10240p$ $Flows = 1024$

Table 4.5: DQN based AQM algorithm setup parameters.

Parameter	**Value**
Number of hidden layers	2
Number of neurons in each hidden layer	64
Learning rate α	0.01
Mini-bacth size	32
Reward discount factor γ	0.99
Periodic target network update step C	5
Maximum queue size	50 packets
Action selection interval T_{int}	1ms
Desired queuing delay	30ms
Reward scaling factor δ	0.4, 0.5, 0.6

Chapter 5

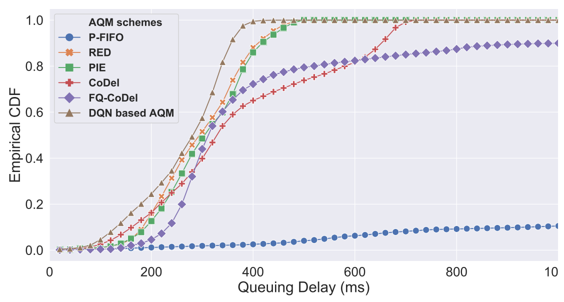
Results and Analysis

In this chapter, we analyze and discuss the results of experiments described in chapter 4. We compare our proposed AQM algorithm with P-FIFO and various AQM schemes including RED, PIE, CoDel, and FQ-CoDel on different environments by adjusting RTT and bandwidth of P2P link between the edge node and cloud gateway.

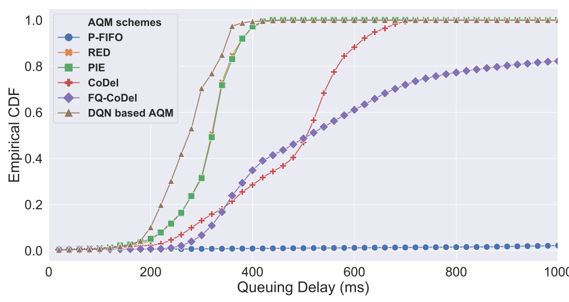
5.1 Comparison on Queuing Delay and Occupancy

In this section, we study the queuing delay and queue occupancy of each AQM scheme. Figure 5.1 shows the empirical cumulative distribution function (CDF) of delay which is induced RTT that the IoT devices' flows suffer during the simulation time, and comparison of each AQM scheme. We can observe that P-FIFO mostly experiences high delay of over one second from both cases of the 21 and 63 IoT devices scenarios because it just classifies packets, but does not work actively on the fluctuating congestion. In both cases, DQN based AQM performs better than other AQM schemes. RED and PIE adapt their queue for delay smaller or slightly larger than 500ms. Comparing with them, DQN based AQM has higher probability of queuing delay under 500ms than other AQM schemes' probabilities as well as maintaining less than 500ms in total. If queuing delay of 500ms is required under 1Mbps bandwidth, P-FIFO, RED, PIE, CoDel, FQ-CoDel and DQN based AQM provide such guarantee with 4.11%, 99.83%, 99.99%, 73.87%, 79.48%, and 100% respectively for 21 IoT devices, and 1.09%, 100%, 100%, 47.27%, 48.79%, and 100% respectively for 63 IoT devices. Alternatively, in 99% of probability, maximum delays increased for 21 IoT devices are 3332ms, 472ms, 478ms, 696ms, 2301ms, and 388ms respectively for P-FIFO, RED, PIE, CoDel, FQ-CoDel, and DQN based AQM schemes.

Figure 5.2 depicts the queue occupancy of each AQM schemes during the simulation time of 100s. We can observe that DQN based AQM does not exceed its maximum queue size of 50 packets as an optimal behaviour for maximum future cumulative reward.



(a) 21 IoT devices

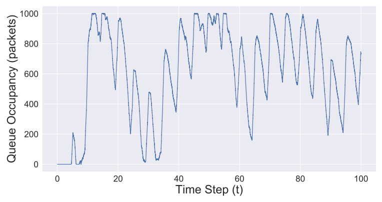


(b) 63 IoT devices

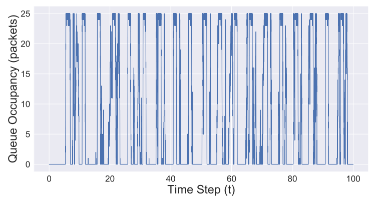
Figure 5.1: Empirical CDF of queuing delay for flows of IoT devices; P2P link data rate = 1Mbps; Baseline RTT = 0ms; Reward scaling factor $\delta = 0.5$

5.2 Comparison on Different Bandwidth

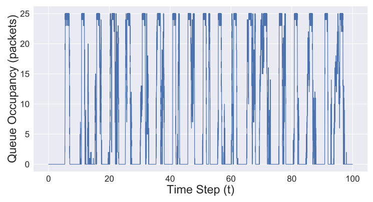
Here, we analyze the performance of AQM schemes on the topology consisting of 21 IoT devices using 1Mbps and 0.5Mbps P2P link capacity between the edge device and cloud gateway. Figure 5.3 is a boxplot of the throughput for each flow. Since the IoT devices deployed in our testbed vary and transmit packets stochastically, a large number of outlier points are observed comparing with related works and the boxplot could be inconspicuous, but this graphical measurement is relatively efficient. In Figure 5.3, medians of throughput for each AQM schemes are relatively similar. FQ-CoDel shows higher median throughput on 0.5Mbps bandwidth, but also undergoes the second highest delay depicted in Figure 5.4



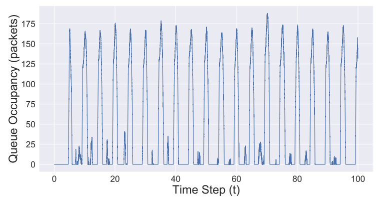
(a) P-FIFO



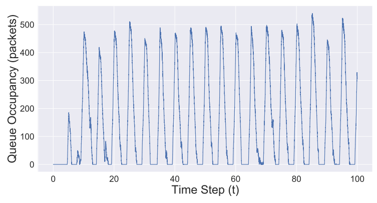
(b) RED



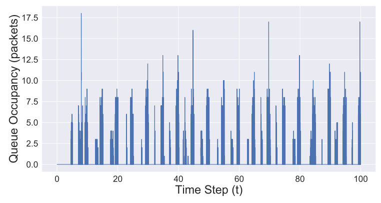
(c) PIE



(d) CoDel



(e) FQ-CoDel



(f) DQN based AQM

Figure 5.2: Comparison of queue occupancy of AQM schemes; P2P link data rate = 1Mbps; Baseline RTT = 0ms; Number of IoT devices = 21; Reward scaling factor $\delta = 0.5$

although the shape of all other delay boxplots AQM schemes' delay is too flat due to the P-FIFO. DQN based AQM shows the lowest median delay maintaining usual throughput comparing with other AQM schemes on 1Mbps bandwidth, and tenable performance of both throughput and delay on 0.5Mbps bandwidth.

5.3 Comparison on Different Baseline RTT

In this section, we compare the AQM schemes on different baseline RTTs: 0, 10, 20, 50, 100ms. Also, we set 63 IoT devices consisting of three duplicated devices from 21 distinct devices. Figure 5.5 shows each of boxplot of throughput for different baseline RTTs. We can observe that the inter-quartile range of boxplot is decreasing as the baseline RTTs become larger. The intriguing point is that the median of FQ-CoDel maintains higher throughput on every tested case. However, as a trade-off, FQ-CoDel shows high delay on every case as shown in Figure 5.6. We also investigate the maximum delay time and most frequent delay time that each flow suffers to clarify the degree of delay in Figures 5.7 and 5.8. We can see that DQN based AQM experiences low delays w.r.t. frequency and maximum values,

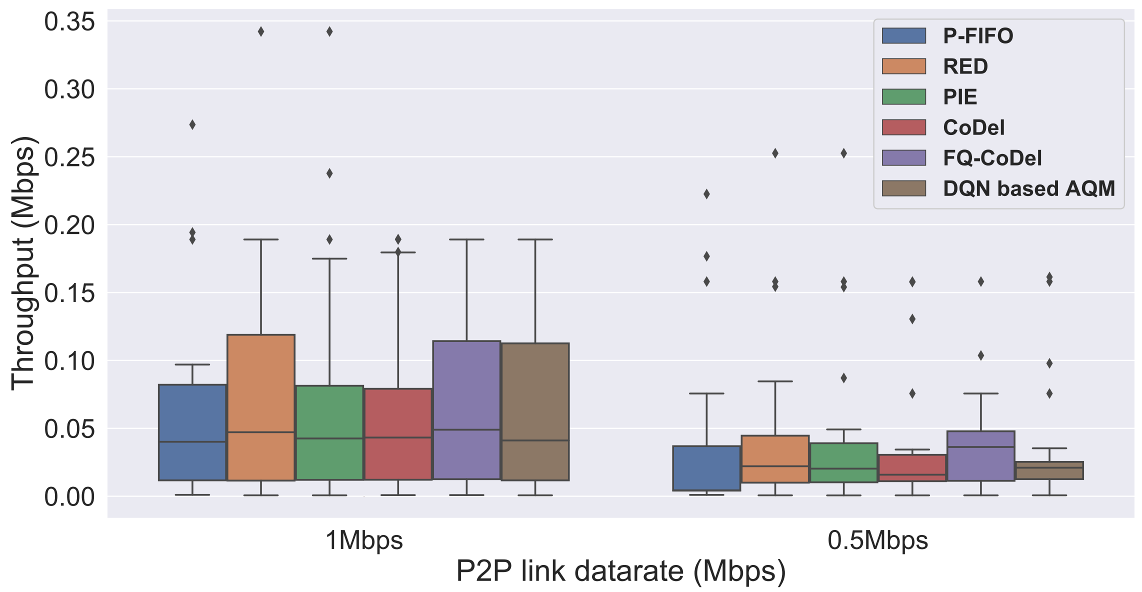


Figure 5.3: Throughput per flows of 21 IoT devices; Baseline RTT = 0ms; Reward scaling factor $\delta = 0.5$

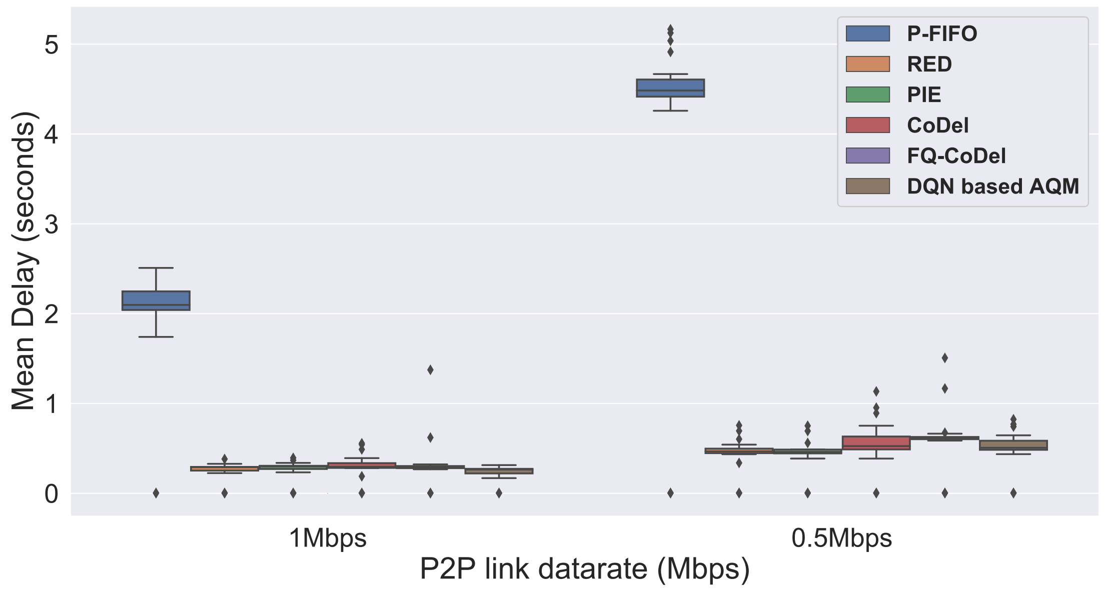


Figure 5.4: Mean delay per flows of 21 IoT devices; Baseline RTT = 0ms; Reward scaling factor $\delta = 0.5$

and outperforms other AQM schemes in terms of queuing delay on all given cases except for 50ms of RTT in that our approach is delay-aware learning algorithm. And generally, queuing delay of AQM schemes are not affected by different RTTs a lot on each delay-related figures.

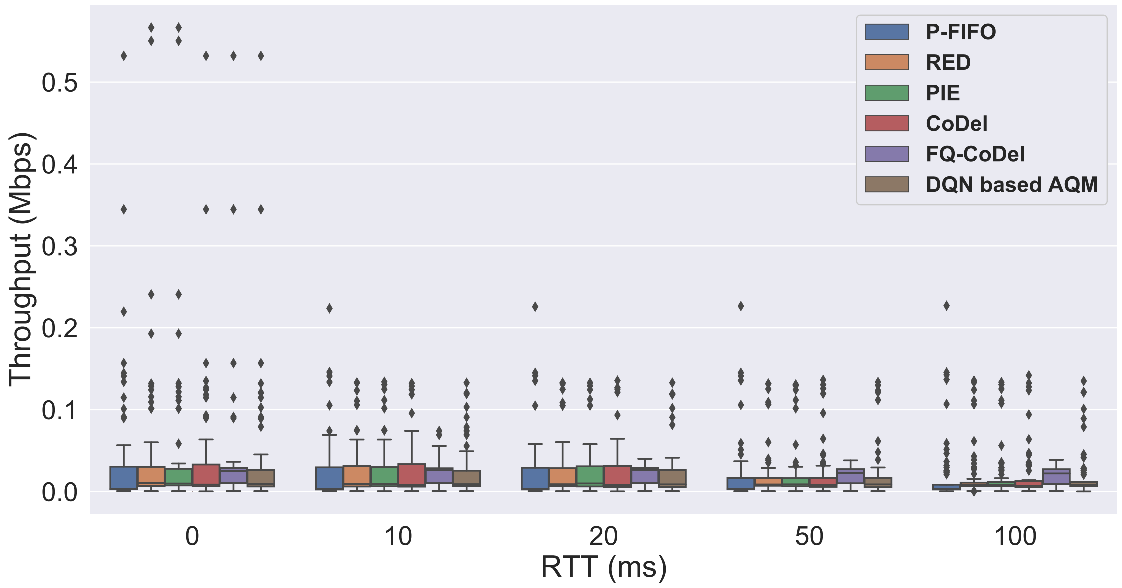


Figure 5.5: Throughput per flows of 63 IoT devices; P2P link data rate = 1Mbps; Reward scaling factor $\delta = 0.5$

5.4 Analysis on Different IoT Categories

Depending on the IoT categories, we set different transport layer protocols. For smart cameras, we set UDP protocol assuming the real-time monitoring and surveillance and store streaming data into cloud servers. For other categories consisting of Hubs, Switches & Triggers, Air quality sensors, Healthcare devices, Light bulbs, and Electronics, we set TCP protocol assuming HTTPS protocol which is mostly used for IoT devices in [33] and the reliable communication.

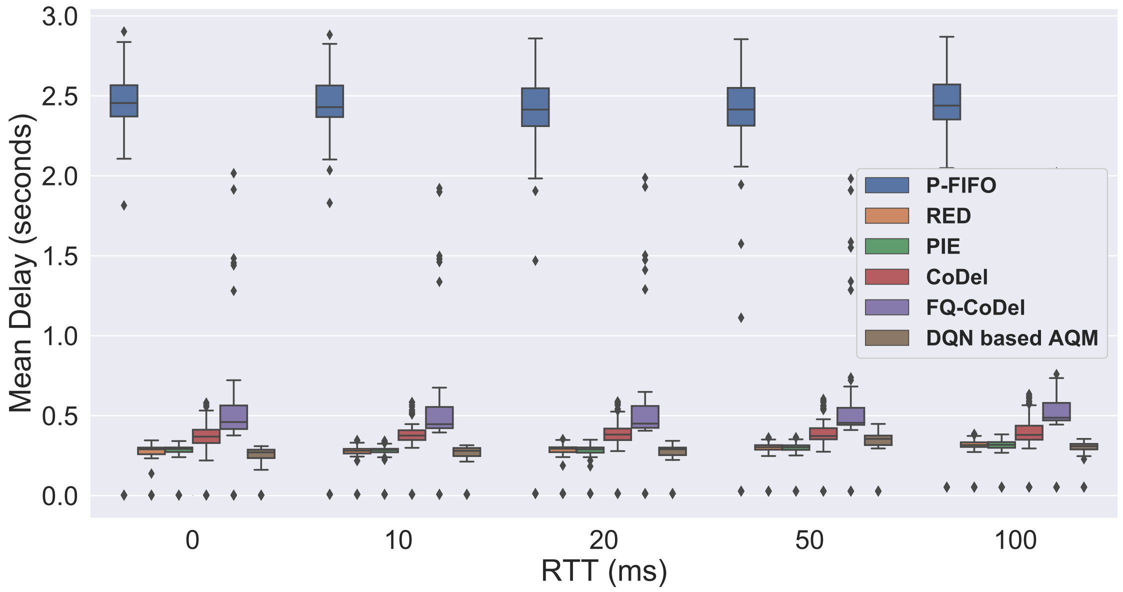


Figure 5.6: Mean delay per flows of 63 IoT devices; P2P link data rate = 1Mbps; Reward scaling factor $\delta = 0.5$

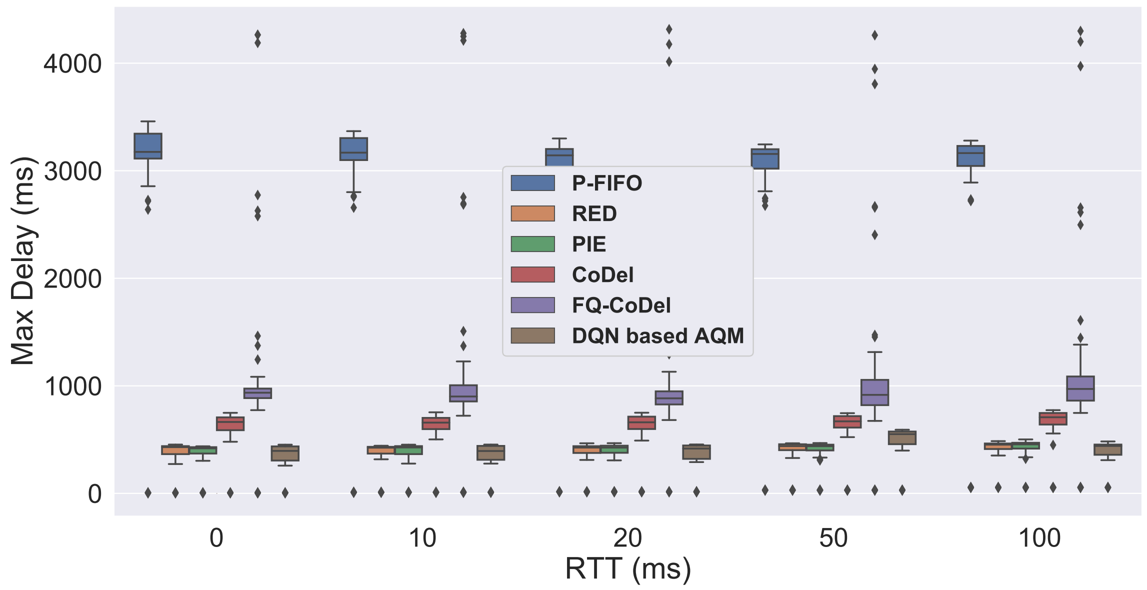


Figure 5.7: Maximum delay per flows of 63 IoT devices; P2P link data rate = 1Mbps; Reward scaling factor $\delta = 0.5$

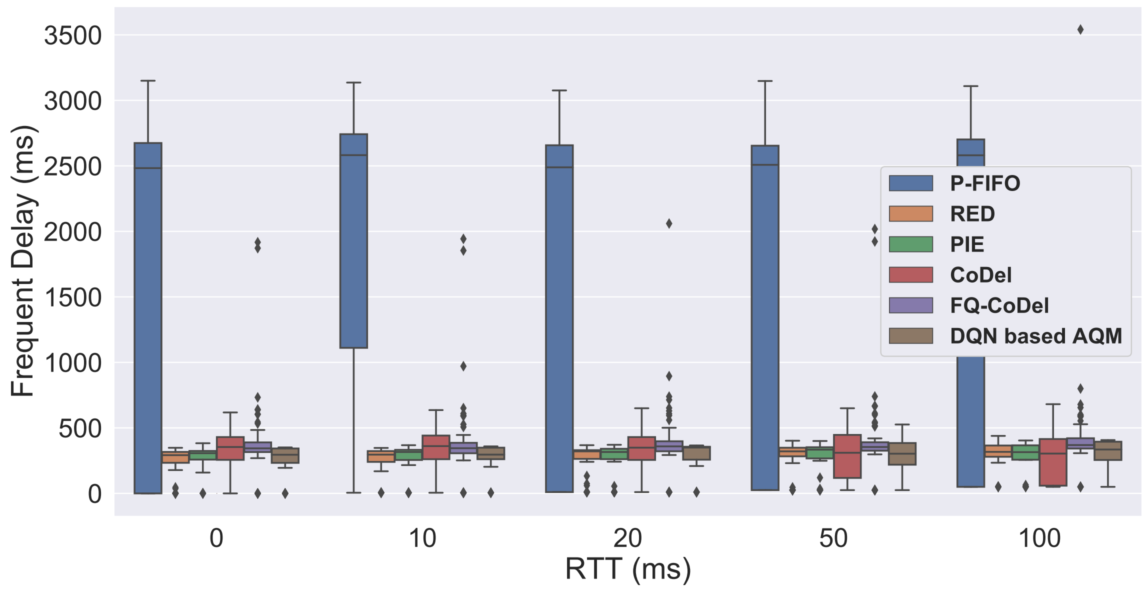


Figure 5.8: Most frequent delay per flows of 63 IoT devices; P2P link data rate = 1Mbps; Reward scaling factor $\delta = 0.5$

5.4.1 UDP based IoT flows

We classify UDP based IoT flows generated by smart cameras from the experiment of section 5.3, and Figures 5.9, 5.10, and 5.11 are boxplots of throughput, mean delay, and mean jitter per UDP flows respectively. In previous experimental results, FQ-CoDel has shown fine performance in terms of throughput in boxplots, however considering only UDP flows, FQ-CoDel shows deficient performances on all testbeds and metrics. In order to measure QoS to end users in networking, jitter is also an useful metrics for UDP based real-time communication such as video streaming and Voice over IP (VoIP). In Figure 5.11, we can observe that DQN based AQM sustains the low jitter regardless of RTTs.

5.4.2 TCP based IoT flows

TCP based packet transmission is one of the main reasons that causes traffic congestion due to the three-way handshake referred to ”SYN-SYN-ACK”. On this testbed, Figures

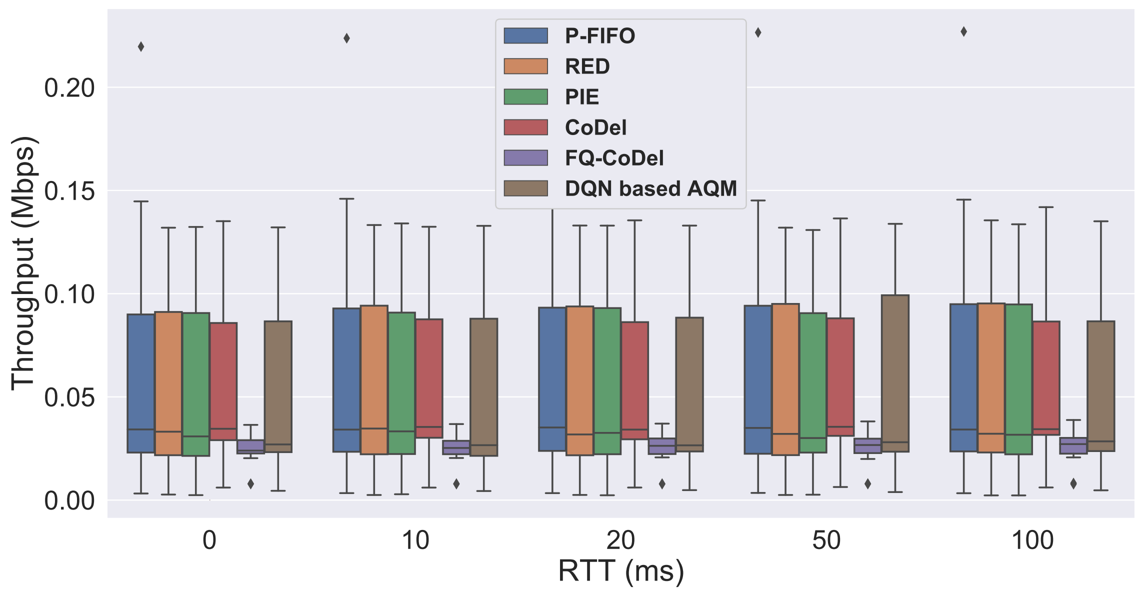


Figure 5.9: Throughput per UDP flows of 18 IoT devices; P2P link data rate = 1Mbps; Reward scaling factor $\delta = 0.5$

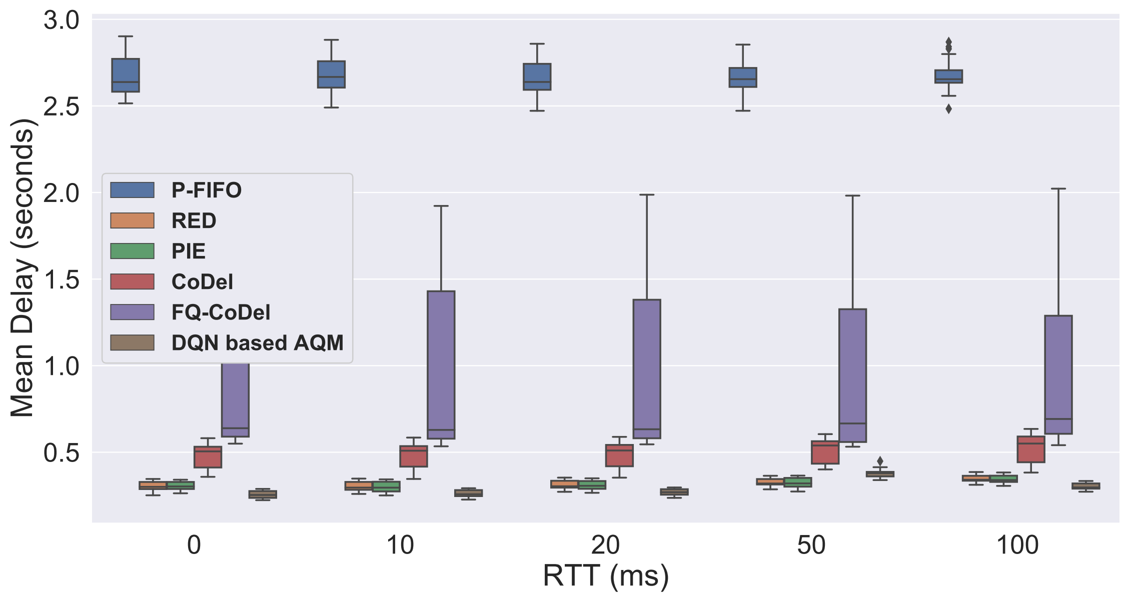


Figure 5.10: Mean Delay per UDP flows of 18 IoT devices; P2P link data rate = 1Mbps; Reward scaling factor $\delta = 0.5$

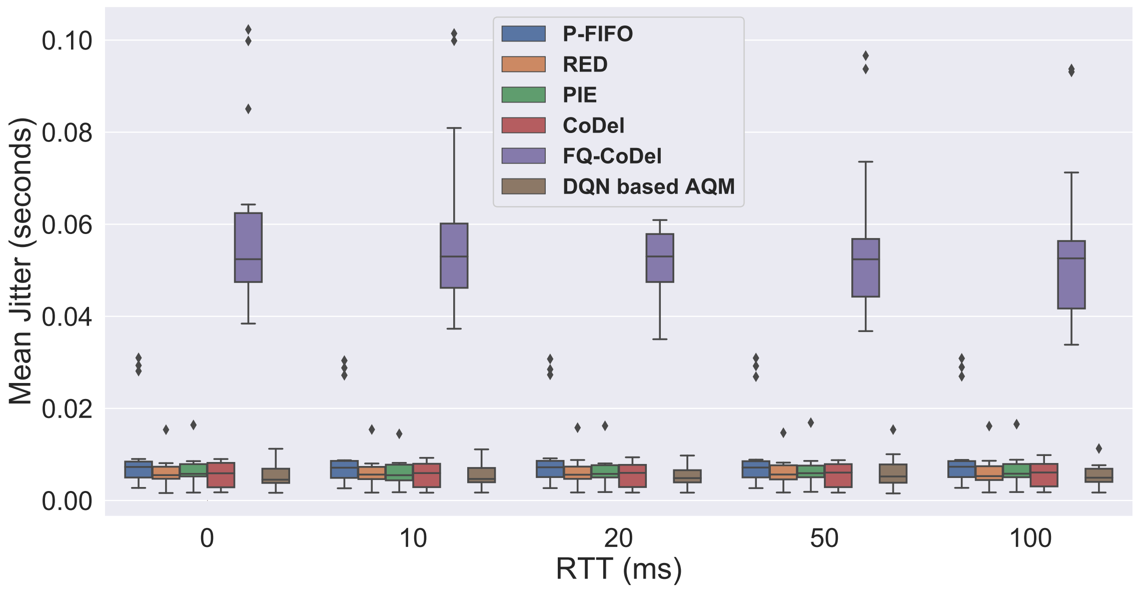


Figure 5.11: Mean Jitter per UDP flows of 18 IoT devices; P2P link data rate = 1Mbps; Reward scaling factor $\delta = 0.5$

5.12 and 5.13 show the throughput and delay on 48 TCP flows of IoT devices. Unlike UDP flows, FQ-CoDel shows better throughput and DQN based AQM's throughput is analogous to other AQM schemes. But with regard to the delay, DQN based AQM shows notable performance as compared with targeted AQM schemes.

5.5 Analysis on Reward Trajectory for Model Optimization

In this section, we focus on our reward function and how it effects to the results. Figure 5.14 shows the comparison of rewards on non-trained/trained network model, and we can see that cumulative reward of the trained model is much higher than that of the non-trained model. However, the non-trained model's cumulative reward increases linearly as well, since it is being trained in parallel through the simulation time step. In Figure 5.14(b), non-

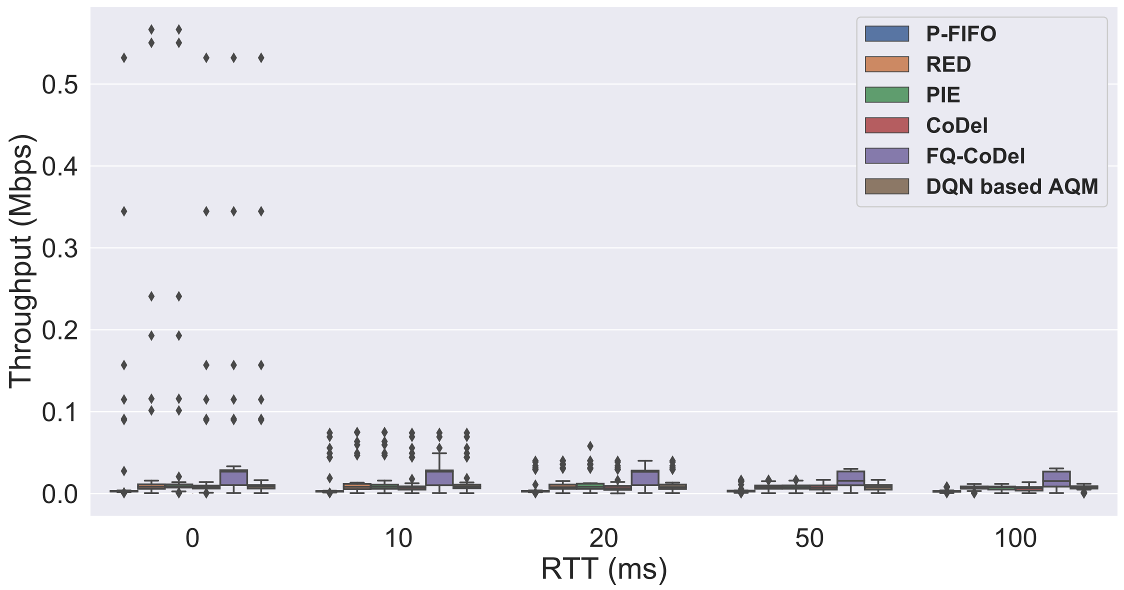


Figure 5.12: Throughput per TCP flows of 45 IoT devices; P2P link data rate = 1Mbps; Reward scaling factor $\delta = 0.5$

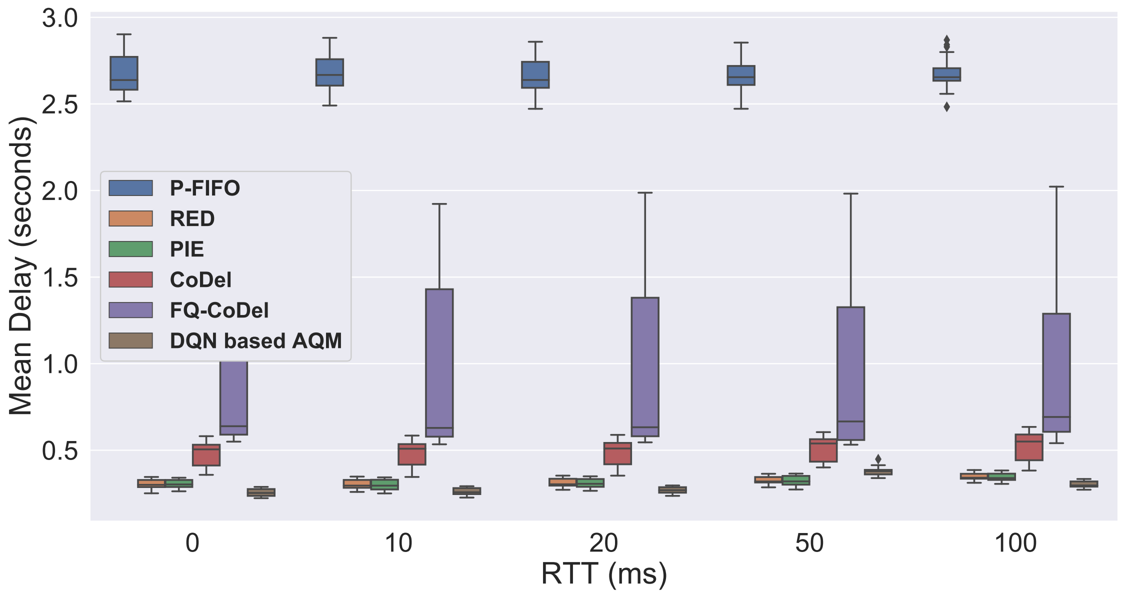
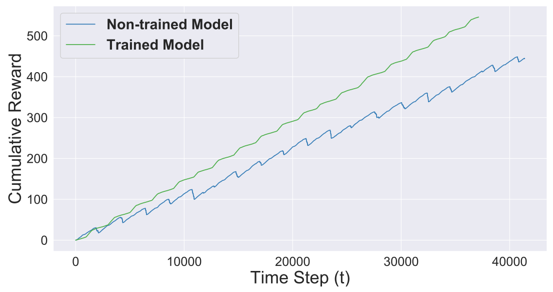
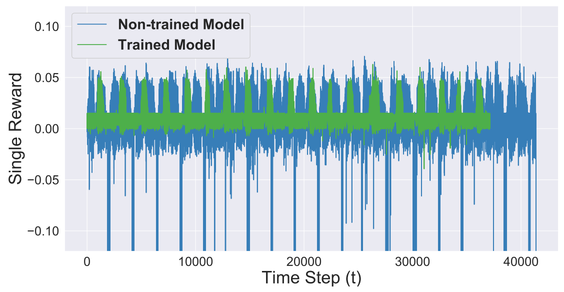


Figure 5.13: Mean Delay per TCP flows of 45 IoT devices; P2P link data rate = 1Mbps; Reward scaling factor $\delta = 0.5$



(a) Cumulative reward



(b) Single reward

Figure 5.14: Comparison of reward on non-trained/trained model; P2P link data rate = 1Mbps; Baseline RTT = 0ms; Number of IoT devices = 63; Reward scaling factor $\delta = 0.5$

trained model shows the unstable behaviour because its action has randomness. Comparing with the non-trained model, trained model performs 22.362ms and 0.57841ms shorter in delay and jitter respectively. Table 5.1 tabulates the sensitivity depending on the reward scaling factor δ. Higher scaling factor stands for that the reward is more weighted towards *delay_reward* than *enqueue_reward* by the agent. Since we use drop-tail function when the queue is filled with packets, packets are dropped by either exceeded queue size or packet-dropping action, and that is why the packet drop-rate is not in inversely proportion of the mean delay. However, we can also observe that depending on the higher scaling factor, mean delay is decreased.

Table 5.1: Reward scaling factor sensitivity; P2P link data rate = 1Mbps; Baseline RTT = 0ms; Number of IoT devices = 63

Scaling factor	**Throughput**	**Packet drop-rate**	**Mean delay**	**Mean jitter**
0.4	0.954097Mbps	24.23%	0.267s	0.008s
0.5	0.953510Mbps	24.17%	0.197s	0.007s
0.6	0.954050Mbps	23.1%	0.195s	0.0068s

5.6 Summary

In this chapter, we analyzed the experimental results on different testbeds. In general, P-FIFO performed inferior since the P-FIFO enabled queue works passively, and FQ-CoDel showed fine throughput performance, but suffers from the trade-off with high delay. The proposed DQN based AQM scheme verified low queuing delay on most cases preserving above the average throughput. In summary, Appendix B tabulates average throughput, packet drop-rate, delay, and jitter for each AQM schemes in various experimental environments, and DQN based AQM performs competitively trading off between mean delay and throughput, as well as low jitter. In the next chapter, we conclude this thesis by providing overall summary with future work to enhance efficient network management in IoT based networks.

Chapter 6

Conclusion

6.1 Conclusion and Summary

In this thesis, we proposed the design of Deep Reinforcement Learning based Active Queue Management. As a baseline model of the design, we selected Deep Q-Network since the state transition in networking is discrete and it is able to be expressed as a finite Markov Decision Process which is the fundamental principle of reinforcement learning. From the state setup to the reward function design, we implemented them on *ns-3* network simulator by applying deep learning framework *tiny-dnn*. Along with fog/edge computing architecture, we also constructed the IoT testbed by reflecting real IoT devices' characteristics including periodic operating cycle, actual transmission rate, packet size, and peak rate in wireless communication environment. We deployed the AQM scheme at the interface of the fog/edge device connected to the cloud gateway, and our proposed scheme achieved substantial performances such as low queuing delay and jitter on the stochastic IoT environment simultaneously maintaining good throughput comparing with widely used AQM schemes.

6.2 Future Work

6.2.1 Energy Efficient AQM Design

Although the fog/edge device has sufficient computation capability, energy efficient design plays a significant role for ideal *future Internet* infrastructure due to the fact that the number of devices of tier-1 is increasing exponentially in our life. In terms of energy efficient AQM, we expect that it can be achieved by increasing the interval of packet-drop probability calculation and optimizing memory usage. For DL based AQM, the light neural network model such as Binarized Neural Network (BNN) [39] can be used for the energy efficient AQM design. And applying optimal model and tuning hyper-parameters are always huge challenges for deep learning applied work. Deep learning field is growing extremely fast, and performance of the algorithms are being improved day by day. Optimal model can be achieved using model compression techniques such as weight pruning. Hyper-parameter tuning of selected model is also capable of improving the performance as well as selecting a good model, and even it has been extended as a new field which is Automatic Machine Learning (AutoML).

6.2.2 Construction of IoT-realistic Simulation Environment

In our experiment, we used IEEE 802.11n 5GHz as a Wi-Fi standard, and TCP and UDP protocols for transmission. If we set lightweight and resource limited protocol such as Constrained Application Protocol (CoAP) and message queue telemetry transport (MQTT) on the 6LoWPAN which is designed for low power and narrow bandwidth for IoT specified by IEEE 802.15.4, we would conduct more ideal experiments and get more accurate results [3].

Appendix A

Dequeue Rate Calculation in PIE

1. Upon packet departure, decide to be in a measurement cycle if :

$$qlen > dq\_threshold;$$

2. If the above is true, update the departure count in bytes dq_count :

$$dq\_count = dq\_count + dq\_pktsize;$$

3. Update dequeue rate once $dq\_count > dq\_threshold$ with averaging parameter ϵ, set to 0.5 by default, and reset counters :

$$dq\_int = time_{now} - time_{start};$$

$$dq\_rate = \frac{dq\_count}{dq\_int};$$

$$avg\_drate = (1 - \epsilon) * avg\_drate + \epsilon * dq\_rate$$

$$time_{start} = time_{now}.$$

$$dq\_count = 0;$$

Appendix B

AQM Performance Comparison in Various Environments

Table B.1: Performance comparison for all flows of 21 IoT devices; P2P link data rate = 1Mbps; Baseline RTT = 0ms; Reward scaling factor $\delta = 0.6$.

AQM scheme	**Total throughput**	**Packet drop-rate**	**Mean delay**	**Mean jitter**
P-FIFO	0.9537Mbps	5.36%	1.736s	0.0192s
RED	0.9512Mbps	23%	0.222s	0.0073s
PIE	0.9513Mbps	23.79%	0.23s	0.0072s
CoDel	0.9539Mbps	18.81%	0.284s	0.0099s
FQ-CoDel	0.9538Mbps	18.72%	0.379s	0.0151s
DQN based AQM	**0.9541Mbps**	**23.1%**	**0.195s**	**0.0068s**

Table B.2: Performance comparison for all flows of 21 IoT devices; P2P link data rate = 0.5Mbps; Baseline RTT = 0ms; Reward scaling factor $\delta = 0.5$.

AQM scheme	**Total throughput**	**Packet drop-rate**	**Mean delay**	**Mean jitter**
P-FIFO	0.4766Mbps	17.24%	4.69s	0.0329s
RED	0.4755Mbps	43.3%	0.5326s	0.0174s
PIE	0.4755Mbps	43.1%	0.5231s	0.0179s
CoDel	0.4767Mbps	36.52%	0.7242s	0.0228s
FQ-CoDel	0.4769Mbps	44.55%	0.7997s	0.032s
DQN based AQM	**0.4767Mbps**	**39.94%**	**0.5919s**	**0.0185s**

Table B.3: Performance comparison for all flows of 63 IoT devices; P2P link data rate = 1Mbps; Baseline RTT = 0ms; Reward scaling factor $\delta = 0.5$.

AQM scheme	**Total throughput**	**Packet drop-rate**	**Mean delay**	**Mean jitter**
P-FIFO	0.9535Mbps	46.97%	2.1559s	0.0245s
RED	0.9524Mbps	53.72%	0.2944s	0.0092s
PIE	0.9524Mbps	54.24%	0.3044s	0.095s
CoDel	0.9535Mbps	50.5%	0.3712s	0.0103s
FQ-CoDel	0.9543Mbps	63%	0.5015s	0.0707s
DQN based AQM	**0.9538Mbps**	**54.04%**	**0.2709s**	**0.0076s**

Table B.4: Performance comparison for all flows of 63 IoT devices; P2P link data rate = 1Mbps; Baseline RTT = 10ms; Reward scaling factor $\delta = 0.5$.

AQM scheme	**Total throughput**	**Packet drop-rate**	**Mean delay**	**Mean jitter**
P-FIFO	0.9532Mbps	46.5%	2.616s	0.0238s
RED	0.9522Mbps	53.35%	0.3065s	0.0093s
PIE	0.9522Mbps	53.55%	0.3067s	0.0093s
CoDel	0.9532Mbps	50.41%	0.4754s	0.0104s
FQ-CoDel	0.954Mbps	63.03%	0.7458s	0.071s
DQN based AQM	**0.9534Mbps**	**53.99%**	**0.2773s**	**0.0076s**

Table B.5: Performance comparison for all flows of 63 IoT devices; P2P link data rate = 1Mbps; Baseline RTT = 20ms; Reward scaling factor $\delta = 0.5$.

AQM scheme	**Total throughput**	**Packet drop-rate**	**Mean delay**	**Mean jitter**
P-FIFO	0.9527Mbps	46.04%	2.601s	0.0238s
RED	0.9521Mbps	53.54%	0.3138s	0.0094s
PIE	0.9521Mbps	53.6%	0.3142s	0.0094s
CoDel	0.9519Mbps	50.22%	0.4784s	0.0102s
FQ-CoDel	0.9535Mbps	62.74%	0.7493s	0.07s
DQN based AQM	**0.953Mbps**	**54.14%**	**0.2831s**	**0.0078s**

Table B.6: Performance comparison for all flows of 63 IoT devices; P2P link data rate = 1Mbps; Baseline RTT = 50ms; Reward scaling factor $\delta = 0.5$.

AQM scheme	**Total throughput**	**Packet drop-rate**	**Mean delay**	**Mean jitter**
P-FIFO	0.9522Mbps	45.86%	2.604s	0.0236s
RED	0.9503Mbps	53.41%	0.3279s	0.0093s
PIE	0.9516Mbps	53.87%	0.3286s	0.01s
CoDel	0.9513Mbps	49.89%	0.4908s	0.0104s
FQ-CoDel	0.9531Mbps	62.6%	0.7702s	0.0702s
DQN based AQM	**0.9518Mbps**	**53.1%**	**0.3781s**	**0.0102s**

Table B.7: Performance comparison for all flows of 63 IoT devices; P2P link data rate = 1Mbps; Baseline RTT = 100ms; Reward scaling factor $\delta = 0.5$.

AQM scheme	**Total throughput**	**Packet drop-rate**	**Mean delay**	**Mean jitter**
P-FIFO	0.9518Mbps	45.67%	2.621s	0.0234s
RED	0.9492Mbps	53.11%	0.3487s	0.0098s
PIE	0.9495Mbps	53.49%	0.3473s	0.0101s
CoDel	0.9455Mbps	49.48%	0.509s	0.0107s
FQ-CoDel	0.9526Mbps	62.38%	0.7957s	0.0714s
DQN based AQM	**0.9516Mbps**	**54.06%**	**0.3155s**	**0.0086s**

Bibliography

[1] IDC, "Worldwide Internet of Things Forecast Update, 20172021," in *document US43304017, IDC*, February 2018. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=US43304017.

[2] Gartner, "Gartner says a typical family home could contain more than 500 smart devices by 2022," September 2014. [Online]. Available: https://www.gartner.com/newsroom/id/2839717.

[3] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," *IEEE Internet of Things Journal*, vol. 4, pp. 1125–1142, March 2017. [Online]. Available: https://doi.org/10.1109/JIOT.2017.2683200.

[4] M. Mukherjee, L. Shu, and D. Wang, "Survey of Fog Computing: Fundamental, Network Applications, and Research Challenges," *IEEE Communications Surveys Tutorials*, vol. 20, pp. 1826–1857, March 2018. [Online]. Available: https://doi.org/10.1109/COMST.2018.2814571.

[5] N. Kuhn, P. Natarajan, N. Khademi, and D. Ros, "Characterization Guidelines for Active Queue Management (AQM)," in *Internet Engineering Task Force (IETF), RFC 7928*, July 2016. [Online]. Available: https://tools.ietf.org/html/rfc7928.

[6] F. Baker and G. Fairhurst, "IETF Recommendations Regarding Active Queue Management," in *Internet Engineering Task Force (IETF), RFC 7567*, July 2015. [Online]. Available: https://tools.ietf.org/html/rfc7567.

[7] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep Learning for IoT Big Data and Streaming Analytics: A Survey," *IEEE Communications Surveys Tutorials*, June 2018. [Online early access]. Available: https://doi.org/10.1109/COMST.2018.2844341.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* MIT Press, second ed., 2018.

[9] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey," in *arXiv preprint arXiv:1810.07862*, October 2018. [Online]. Available: https://arxiv.org/abs/1810.07862.

[10] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993. [Online]. Available: https://doi.org/10.1109/90.251892.

[11] K. Nichols and V. Jacobson, "Controlling Queue Delay," *Communications of the ACM*, vol. 55, pp. 42–50, July 2012. [Online]. Available: https://doi.org/10.1145/2209249.2209264.

[12] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "PIE: A lightweight control scheme to address the bufferbloat problem," in *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, July 2013. [Online]. Available: https://doi.org/10.1109/HPSR.2013.6602305.

[13] A. Sungur, "Tcp – random early detection (red) mechanism for congestion control," Master's thesis, Rochester Institute of Technology, 2015. [Online]. Available: https://scholarworks.rit.edu/theses/8904.

[14] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm," in *Internet Engineering Task Force (IETF), RFC 8290*, January 2018. [Online]. Available: https://tools.ietf.org/html/rfc8290.

[15] N. Bouacida and B. Shihada, "Practical and Dynamic Buffer Sizing using LearnQueue," *IEEE Transactions on Mobile Computing*, September 2018. [Online early access]. Available: https://doi.org/10.1109/TMC.2018.2868670.

[16] S. K. Bisoy, P. K. Pandey, and B. Pati, "Design of an active queue management technique based on neural networks for congestion control," in *2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, December 2017. [Online]. Available: https://doi.org/10.1109/ANTS.2017.8384104.

[17] N. Vucevic, J. Perez-Romero, O. Sallent, and R. Agusti, "Reinforcement Learning for Active Queue Management in Mobile All-IP Networks," in *2007 IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications*, September 2007. [Online]. Available: https://doi.org/10.1109/PIMRC.2007.4394713.

[18] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized Computation Offloading Performance in Virtual Edge Computing Systems via Deep Reinforcement Learning," *IEEE Internet of Things Journal*, October 2018. [Online early access]. Available: https://doi.org/10.1109/JIOT.2018.2876279.

[19] S. Chinchali, P. Hu, T. Chu, M. Sharma, M. Bansal, R. Misra, M. Pavone, and S. Katti, "Cellular Network Traffic Scheduling With Deep Reinforcement Learning," in *Thirty-*

Second AAAI Conference on Artificial Intelligence, February 2018. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16638.

[20] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven Networking: A Deep Reinforcement Learning based Approach," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018. [Online]. Available: https://doi.org/10.1109/INFOCOM.2018.8485853.

[21] J. Kua, S. H. Nguyen, G. Armitage, and P. Branch, "Using Active Queue Management to Assist IoT Application Flows in Home Broadband Networks," *IEEE Internet of Things Journal*, vol. 4, pp. 1399–1407, July 2017. [Online]. Available: https://doi.org/10.1109/JIOT.2017.2722683.

[22] Y. Zhang, B. Feng, W. Quan, G. Li, H. Zhou, and H. Zhang, "Theoretical Analysis on Edge Computation Offloading Policies for IoT Devices," *IEEE Internet of Things Journal*, October 2018. [Online early access]. Available: https://doi.org/10.1109/JIOT.2018.2875599.

[23] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, "On Reducing IoT Service Delay via Fog Offloading," *IEEE Internet of Things Journal*, vol. 5, pp. 998–1010, January 2018. [Online]. Available: https://doi.org/10.1109/JIOT.2017.2788802.

[24] S. Zhao, Y. Yang, Z. Shao, X. Yang, H. Qian, and C.-X. Wang, "FEMOS: Fog-Enabled Multitier Operations Scheduling in Dynamic Wireless Networks," *IEEE Internet of Things Journal*, vol. 5, pp. 1169–1183, February 2018. [Online]. Available: https://doi.org/10.1109/JIOT.2018.2808280.

[25] X. Zheng, Z. Cai, J. Li, and H. Gao, "Scheduling Flows With Multiple Service Frequency Constraints," *IEEE Internet of Things Journal*, vol. 4, pp. 496–504, June 2016. [Online]. Available: https://doi.org/10.1109/JIOT.2016.2577630.

[26] S. Bhandari, S. K. Sharma, and X. Wang, “Latency Minimization in Wireless IoT Using Prioritized Channel Access and Data Aggregation,” in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, December 2017. [Online]. Available: https://doi.org/10.1109/GLOCOM.2017.8255038.

[27] H. A. A. Al-Kashoash, H. M. Amer, L. Mihaylova, and A. H. Kemp, “Optimization-Based Hybrid Congestion Alleviation for 6LoWPAN Networks,” *IEEE Internet of Things Journal*, vol. 4, pp. 2070–2081, September 2017. [Online]. Available: https://doi.org/10.1109/JIOT.2017.2754918.

[28] L. Zhao, W. Sun, Y. Shi, and J. Liu, “Optimal Placement of Cloudlets for Access Delay Minimization in SDN-Based Internet of Things Networks,” *IEEE Internet of Things Journal*, vol. 5, pp. 1334–1344, March 2018. [Online]. Available: https://doi.org/10.1109/JIOT.2018.2811808.

[29] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, pp. 279–292, May 1992. [Online]. Available: https://link.springer.com/article/10.1007/BF00992698.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, pp. 529–533, February 2015. [Online]. Available: https://www.nature.com/articles/nature14236.

[31] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9, pp. 249–256, March 2010. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf.

[32] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *arXiv preprint arXiv:1412.6980v9*, January 2017. [Online]. Available: https://arxiv.org/abs/1412.6980.

[33] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Characterizing and classifying IoT traffic in smart cities and campuses," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, May 2017. [Online]. Available: https://doi.org/10.1109/INFCOMW.2017.8116438.

[34] Y. Amar, H. Haddadi, R. Mortier, A. Brown, J. Colley, and A. Crabtree, "An Analysis of Home IoT Network Traffic and Behaviour," in *arXiv preprint arXiv:1803.05368*, March 2018. [Online]. Available: https://arxiv.org/abs/1803.05368.

[35] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, pp. 64–74, July 2008. [Online]. Available: https://doi.org/10.1145/1400097.1400105.

[36] M. Mathis, N. Dukkipati, and Y. Cheng, "Proportional Rate Reduction for TCP," in *Internet Engineering Task Force (IETF), RFC 6937*, May 2013. [Online]. Available: https://tools.ietf.org/html/rfc6937.

[37] "ns-3." https://www.nsnam.org/. Accessed: 2018-12-12.

[38] "tiny-dnn." https://github.com/tiny-dnn/tiny-dnn. Accessed: 2018-12-12.

[39] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks," in *30th Conference on Neural Information Processing Systems (NIPS 2016)*, December 2016. [Online]. Available: https://papers.nips.cc/paper/6573-binarized-neural-networks.pdf.