Q-AIMD: AIMD-based Window Flow Control with Reinforcement Learning

Yuta Takesada*, Han Nay Aung*, Ryo Nakamura†, Hiroyuki Ohsaki*
* *School of Engineering Kwansei Gakuin University*
Email: {gak49963,hannayaung,ohsaki}@kwansei.ac.jp
† *Faculty of Engineering Fukuoka University*
Email: r-nakamura@fukuoka-u.ac.jp

***Abstract*—Research on congestion control algorithms, which utilize machine learning techniques without prior knowledge of the network's internal state, has been actively performed. In recent years, various researchers have proposed Reinforcement Learning (RL) based congestion control algorithms. In the RL-based approach, the window size of the data sender is adjusted according to the reward value obtained from the environment. While recent RL-based algorithms (such as Owl, QTCP, and MVFST-RL) adjust the window size, they either rely solely on incremental changes or the additive increase or multiplicative decrease. The specific values for these adjustments are determined through empirical evaluation. We suggest that modeling the behavior of TCP/IP congestion control algorithms (such as the AIMD algorithm) as action sets and incorporating them into the Q-learning framework can lead to more effective outcomes. Furthermore, these algorithms assume that all network flows are greedy. In real-world networks, flows are not homogeneous and every flow requires different QoS (Quality of Service) requirements. The objective of this research is to propose a Q-learning-based AIMD window flow control (Q-AIMD), a congestion control algorithm that dynamically adjusts the window size using Q-learning in an AIMD manner and also supports heterogeneous flows with different QoS requirements.**

***Index Terms*—Congestion Control (CC), AIMD (Additive Increase Multiplicative Decrease), Reward design, Q-Learning, Reinforcement Learning**

I. Introduction

Addressing the challenge of the Internet congestion control is a complex endeavor influenced by various factors such as resource limitations, high traffic demands, complex architecture, and the dynamic nature of network traffic [1, 2]. Internet congestion occurs when the overall demand for network resources surpasses the available capacities, resulting in potential issues such as extended delays in data transmission, data packet losses, and in severe cases, a complete breakdown of the network [3, 4].

Numerous congestion control algorithms have been introduced by researchers. Notable ones among them are TCP Reno [5], TCP Cubic [6], and TCP BBR (Bottleneck Bandwidth and Round-trip propagation time) [7], all of which have gained popularity due to their efficiency. Each congestion control algorithm is proposed based on specific network scenarios. However, a congestion control algorithm designed for a specific network may not be directly applicable to other types of networks. Moreover, as networks become complex, human knowledge alone may not always accurately capture the intricate network characteristics [2].

With recent advancements in machine learning, researchers have proposed congestion control algorithms that utilize machine learning techniques [8-11]. For example, Sacco *et al.* [8] proposed a window-based flow control algorithm called Owl, which utilizes Deep Q-learning, a variant of Deep Reinforcement Learning (DRL). In Owl, each data-transmitting sender uses current network conditions, like the window size and round-trip time, to perform Deep Q-learning, adjusting the window sizes. Similarly, the authors in [11] presented MVFST-RL, which employs Reinforcement Learning (RL) to design congestion control algorithms. In [9], the author proposed a congestion control algorithm based on the multicast QUIC protocol using a data-driven reinforcement learning method. These works demonstrate that employing reinforcement learning can significantly improve performance in congestion control algorithms.

In the Q-Learning-based congestion control algorithms, the congestion window size of a data sender is periodically updated based on the learned Q-table. These algorithms either rely solely on additive changes [8] or the additive increase or multiplicative decrease approach [11]. The specific values for these increases and decreases are determined through empirical evaluation or are common in congestion control algorithms.

We believe that representing the window size behavior of TCP/IP congestion control algorithms (such as AIMD [12]) as action sets and utilizing these within the Q-learning framework can lead to more effective performances. Furthermore, existing studies assume that all network flows are greedy. However, in real-world networks, flows are not homogeneous and that every flow requires different QoS (Quality of Service) requirements.

We sought to answer the following research questions.

- How efficiently or inefficiently does the AIMD (Additive Increase, Multiplicative Decrease) algorithm, used in TCP/IP networks, when represented as an action set and utilized in reinforcement learning-based window flow control?
- To what extent does fairness achived in reinforcement learning-based AIMD window flow control when multiple flows that have similar QoS requirements compete for the bottleneck link?
- How can AIMD actions be adapted to accommodate heterogeneous networks, where multiple flows with differing QoS requirements compete for a bottleneck link?

In this paper, we propose a Q-learning-based AIMD window flow control (Q-AIMD), a congestion control algorithm that dynamically adjusts the window size using Q-learning in an AIMD manner and also supports heterogeneous flows with different QoS requirements.

The main contributions of this paper are summarized as follows.

- We propose an AIMD-based window flow control using Q-learning.
- Through simulations, we quantitatively reveal how the performance of Q-AIMD varies under different network environments, including network topology, the number of flows, link bandwidth, and propagation delay.
- We demonstrate that Q-AIMD achieves high throughput and fair bandwidth sharing among competing flows and that it can handle heterogeneous flows with different QoS requirements.

The structure of this paper is as follows. In Section II, we provide an overview of related work in machine-learning-based congestion control techniques. In Section III, we present the proposed Q-AIMD congestion control algorithm. In Section IV, we present our experimental design for investigating the effectiveness of our Q-AIMD. In Section V, we present the results of our simulations. Finally, Section VI summarizes our findings and discusses potential directions for future research.

II. Background and Related Work

This section provides a brief introduction to Reinforcement Learning (RL) and existing RL-based network congestion control algorithms.

A. Reinforcement Learning

RL, a sub-domain of machine learning serves as a tool in dealing with Markov Decision Processes (MDP), model decision-making problems in discrete-time, where outcomes are influenced by randomness and the control of a learning agent (decision-maker) [13, 14]. It has been applied to various research fields such as game theory, operations research, and network communications [15-17].

Among RL algorithms, the Q-learning [18] is efficient in obtaining an optimal policy when the state space (the set of all possible states a learning agent can encounter in the environment) and action space (the set of all possible actions that the agent can take) is relatively small.

In Q-learning, a learning agent initially observes its current state, takes an action, and receives an immediate reward and information about its new state from the environment. The agent then uses the acquired information to adjust its policy. This iterative process continues until the agent's policy converges towards the optimal policy. It has been applied in the domain of network congestion control algorithm [19], network routing [20-22] and mobile and wireless networking [23].

B. RL-based congestion control algorithm

Authors of [2] point out that traditional rule-based congestion control algorithms (e.g., TCP Reno, New-Reno, Compound TCP (CTCP), Vegas, TCP Cubic, and TCP BBR) have several limitations. For example, it cannot adapt to new networks and cannot learn from experience or prior information about link bandwidth, channel characteristics, or the number of concurrent flows to improve performance.

One of the earliest congestion control algorithm that utilized machine learning was Remy [24]. Remy's design process takes into account critical network parameters, assumptions, and traffic models to generate an algorithm to optimize throughput, queueing delay, or a combination of both. Remy employs a decentralized partially observable Markov decision process, creating congestion control algorithms aligned with specific objectives and network assumptions.

Adding to the advancements, the authors of [19] introduce Aurora, a DRL based scheme. It is reported in [19] that employs a fully-connected Deep Neural Network (DNN) to learn state-action pairs from historical data, using parameters like latency gradient, latency ratio, and sending ratio as state features. Aurora surpasses BBR and Remy in terms of performance.

In another work, the authors of [10] present QTCP, a congestion control algorithm utilizing RL. QTCP autonomously learns effective strategies by dynamically adjusting the congestion window, achieving high throughput and low latency in real time. The learning agent interacts with the network environment, exploring and refining optimal policies through sequential actions. Experiments demonstrate that QTCP outperforms traditional rule-based TCP, achieving a substantial increase in throughput while maintaining low latency.

In [11], the authors introduce MVFST-RL, a framework for congestion control in the QUIC transport protocol that utilizes asynchronous RL training with off-policy correction. MVFST-RL was then evaluated on emulated networks from the Pantheon platform.

In recent years, the authors in [10] introduced Owl, a transport protocol designed to address the limitations of current congestion control methods, especially in dynamic network environments like cellular and wireless networks. By using RL, specifically Deep Q-Learning, Owl dynamically adjusts the congestion window to maximize throughput, ensure fairness, or minimize packet loss and delay. Comprehensive evaluations demonstrate Owl's effectiveness across various network scenarios, making it a promising solution for improving network performance.

III. Q-learning based AIMD window flow control (Q-AIMD)

This section introduces the proposed congestion control algorithm that operates on the data-transmitting sender.

Q-AIMD is a Q-learning based window flow control algorithm that aims to dynamically adjust the number of sending packets, referred to as the window size (*cwnd*), that can be transmitted within a Round-Trip Time (*rtt*).

Let's consider a network environment comprising a set of learning agents (data-transmitting senders) $\mathrm{LA} = \{\mathrm{la}_1, \mathrm{la}_2, \ldots, \mathrm{la}_n\}$ and an environment (Fig. 1). Given n learning agents, each learning agent $\mathrm{la} \in \mathrm{LA}$ can first define its utility function (desired goals), such as high throughput, or specific minimum thresholds for throughput, denoted as U_{la}.

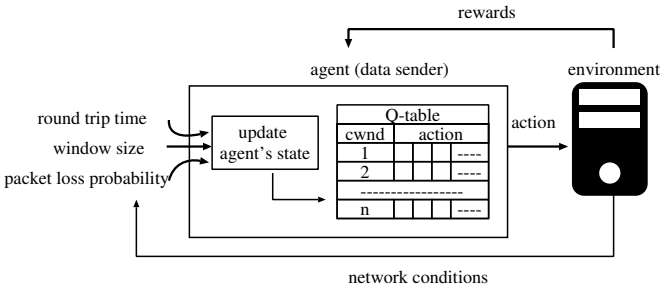

Fig. 1. Overview of Q-AIMD

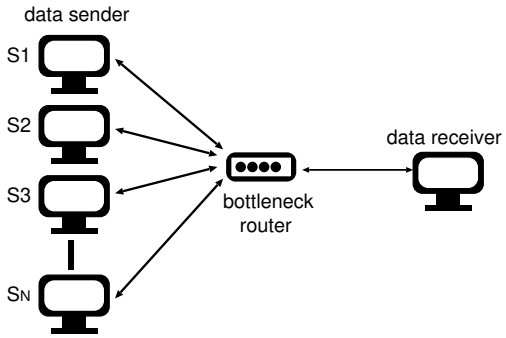

Fig. 2. Dumbbell network topology (multi-flows with single bottleneck)

The learning agent then interacts with the environment and continuously explores optimal action by taking sequential actions (e.g., varying the *cwnd*) based on feedback (current network conditions and rewards) from the environment.

Like typical RL-based congestion control algorithm, Q-AIMD consists of the following elements.

- State S : each learning agent retains a record of observable network state (i.e. *cwnd*).
- Actions A: an action set specifies how Q-AIMD should change its *cwnd* in response to variations in the network conditions. We set the action set of Q-AIMD to $\{cwnd + 0, cwnd + 1, cwnd/2\}$, reflecting the behavior of window size in the AIMD algorithm.
- Reward: the reward value is decided by the utility function. For maximizing throughput, the reward r is calculated by the following equation [8]:

$$r = \lambda - \delta\lambda(1-p)^{-1}, \tag{1}$$

where, λ and p are throughput and packet loss probability. δ is a parameter that adjusts the weighting of throughput and packet loss probability. We calculated λ using the following equation:

$$\lambda = \frac{cwnd}{rtt}. \tag{2}$$

Here, *cwnd* and *rtt* are the window size and round-trip time of a learning agent at the current time, respectively. *rtt* is determined by the sum of the round-trip propagation delay between a data sender and a receiver and queueing delay at an intermediate router.

After deciding on state, actions, and reward, each learning agent begins with the random initialization of $Q(s,a)$ for all states $s \in S$ and actions $a \in A$, stored in the Q-table. Each learning agent updates its Q-table at regular intervals Δ and simultaneously adjusts the next *cwnd* based on the current Q-table values.

The Q-table comprises of rows representing state S and columns representing actions A and each value within the table stores the Q-value, which denotes the expected long-term reward upon selecting a specific action within a given state. The $Q(s,a)$ estimates are updated using the Bellman equation.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max Q(s',a') - Q(s,a)] \tag{3}$$

Here, $Q(s,a)$ represents the estimated Q-value for the state-action pair (s,a). Our approach can be applied to other QoS attributes. r represents the reward obtained when taking action a in state s. Also, α is the learning rate, and γ is the discount factor, determining the weight between future and present states. The state s' refers to the state reached after taking action a in state s, and $\max Q(s',a')$ denotes the maximum Q-value in the new state, reflecting the estimated future reward.

In selecting an action, Q-AIMD uses an epsilon-greedy strategy: with a probability of $(1-\epsilon)$, it chooses the optimal action according to Q-table, and with a probability of ϵ, it selects a random action.

Various applications and services, often have distinct QoS specifications and demands. For instance, real-time communication applications, like video conferencing, may prioritize low latency and minimal packet loss, while bulk data transfers might prioritize maximum throughput over low latency. Recognizing and accommodating these diverse QoS requirements are essential for optimizing the network performance and ensuring that each receives the specific quality attributes it necessitates.

Here, we set utility functions for each learning agent. For the utility function that aims to maximize throughput $U_{\max}(\lambda)$, the reward is calculated using Eq. (1). In the case of the utility function that focused on achieving a minimum required throughput, $U_{\min}(c)$, where c represents the desired throughput. The reward r is calculated as follows.

$$r = U_{\min}(c) = \min(\lambda,\, c) - \delta\,\lambda\,(1-p)^{-1} \tag{4}$$

IV. Experimental Design

In this section, we conducted simulations to evaluate the performance of the Q-AIMD window flow congestion control algorithm. We present our simulation scenario, along with the action set and parameters for Q-AIMD.

We conducted simulations using the network simulator we developed. We used two different topologies: a dumbbell network topology (with N data senders, a router, and a data receiver) as shown in Fig. 2, and a parkinglot network topology (with N data senders, three routers, and a data receiver) depicted in Fig. 3.

In each network topology, data senders are connected to the router(s) to transmit packets, while the router(s) are linked to a single data receiver. We vary the bandwidth of all links uniformly between 10 [packets/ms] and 100 [packets/ms], while the round-trip propagation delay between a data sender

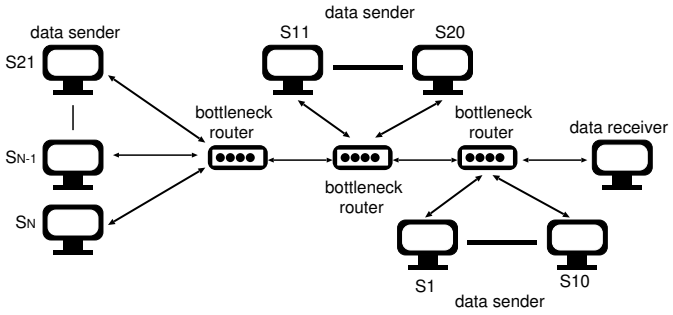

Fig. 3. Parkinglot network topology (multi-flows with multi-bottleneck)

and the router ranged from 1 [ms] to 10 [ms]. The router's maximum queue length was set to 20 [packets].

Each data sender utilized the proposed Q-AIMD algorithm to regulate the number of packets sent. We set the action set of Q-AIMD to $\{cwnd + 0, cwnd + 1, cwnd \times 1/2\}$, reflecting the window size behavior in the AIMD algorithm. The first action does nothing to the current *cwnd*, allowing it to remain unchanged. The second action increases the current *cwnd* by 1 [packet]. The last action reduces the size of *cwnd* by half, which helps to reduce congestion in the network flow. We set the control parameters of Q-AIMD as follows: the learning rate α is 0.1, the discount factor γ is also 0.1, the exploration rate ϵ is set to 0.05, and the scaling factor δ, which influences the impact of loss probability on throughput, is set to 0.7, and each learning agent updates its Q-table at intervals defined by Δ, which is dynamically adjusted based on the sum of the round-trip propagation delays for both the router and the receiver.

V. Performance Evaluation of Q-AIMD

This section discusses the performance of Q-AIMD. We evaluated the effectiveness of our AIMD-based reinforcement learning congestion control algorithm using evaluation metrics such as average throughput, Jain's fairness index, and packet loss probability. We also aimed to clarify how Q-AIMD can accommodate differences in QoS requirements for each flow when the utility functions of the data senders vary.

Additionally, we investigated how the performance of Q-AIMD changes depending on the action set to show the effectiveness of AIMD-based action set. We used the action types used in Owl and MVFST-RL. We define the action sets as follows: (i) Owl $\{-10, -3, -1, 0, +1, +3, +10\}$ as outlined in [8], and (ii) MVFST-RL $\{\times 1/2, -10, 0, +10, \times 2\}$ as specified in [11]. Throughout this paper, the term Q-owl will refer to Q-AIMD, with the action set being identical to that of Owl, while Q-mvfst will refer to Q-AIMD, with the action set identical to that of MVFST-RL.

A. Throughput and Packet Loss Probability

In this subsection, we present the average throughput and packet loss probability of a flow.

To evaluate the performance of the proposed algorithm, we conducted various scenarios in which we varied the number of flows N, link bandwidth B, and propagation delay between

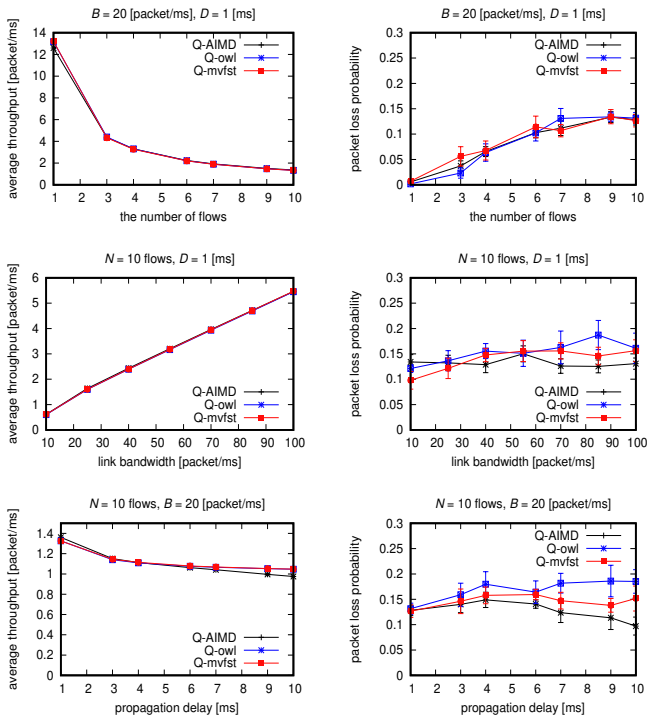

Fig. 4. Performance of Q-AIMD in dumbbell network topology

link D, while keeping the router's maximum queue length is fixed at 20 [packets]. Each simulation is conducted at least 5 times, setting a different random seed for each trial and obtained a 95% confidence interval.

Fig. 4 presents the average throughput and packet loss probability of Q-AIMD, Q-owl, and Q-mvfst in the dumbbell network topology. Fig. 5 presents the results in the parkinglot network topology. Network parameters (such as the total number of flows N, link bandwidth B, and propagation delay D) are specified in the titles of each subfigure.

These results indicate that Q-AIMD achieves an average throughput comparable to that of Q-Owl and Q-mvfst, while also exhibiting a similar packet loss probability, indicating comparable performance. These results also show that Q-AIMD adapts its throughput to closely match the bottleneck link's available bandwidth.

From the view of packet loss probability, the scaling factor δ plays a critical role in balancing throughput and packet loss. We used Eq. (1) to calculate the reward value in each network topology. When δ is set low, each flow focuses solely on maximizing throughput, resulting in significant packet losses. Conversely, a high δ makes each flow more sensitive to packet losses, leading to reduced throughput. Therefore, finding the right balance for δ is crucial.

Also, Q-AIMD operates with only three possible selectable actions in its action set, while Q-owl has seven and Q-mvfst has five. Despite having fewer selectable actions, Q-AIMD can achieve higher throughput and lower packet loss probability. This suggests that the AIMD algorithm's window size behavior, commonly used in TCP/IP networks, serves effectively as the action set in Q-learning.

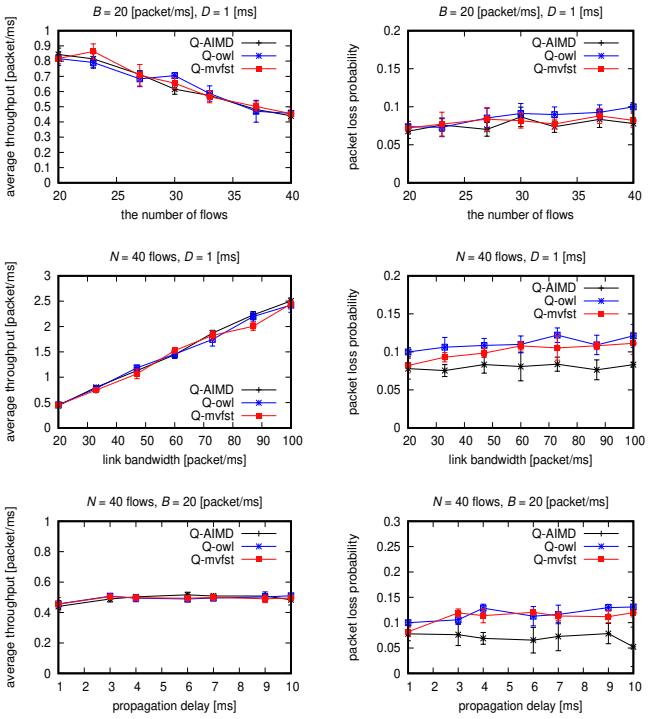

Fig. 5. Performance of Q-AIMD in parkinglot network topology

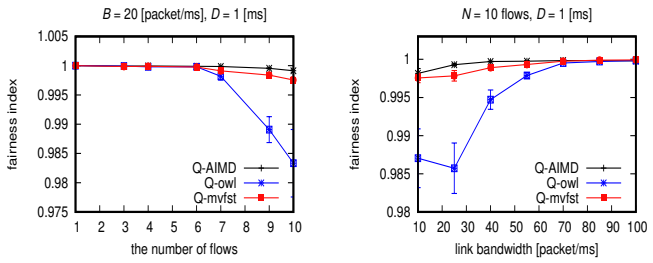

Fig. 6. Fairness in dumbbell network topology

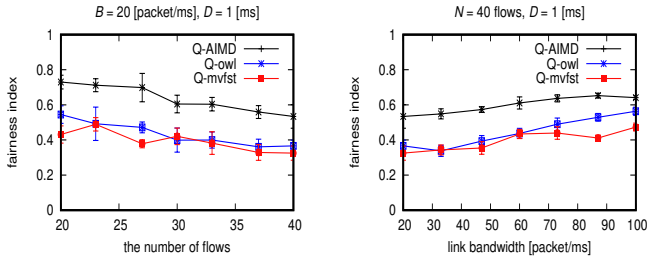

Fig. 7. Fairness in parkinglot network topology

B. Throughput Fairness

In this subsection, we assess the fairness of Q-AIMD among multiple flows that utilize the same congestion control algorithm and utility function while competing for network resources. To quantify the fairness among the flows, we utilized Jain's fairness index, which is calculated using the following equation:

$$J = \frac{\left(\sum_{i=1}^{N} \lambda_i\right)^2}{N \cdot \sum_{i=1}^{N} \lambda_i^2}. \tag{5}$$

Here, N represents the number of flows in a network, and λ_i is the amount of resources (throughput) allocated to the i-th flow. The ideal fairness value is 1.

Fig. 6 shows the fairness of Q-AIMD, Q-owl, and Q-mvfst as we vary the total number of flows (data senders) or the link bandwidth in the dumbbell network topology. Fig. 7 presents the results for the parkinglot network topology.

These results show that Q-AIMD can maintain a higher fairness index even as the total number of flows and the router's link bandwidth increase, indicating a more equitable distribution of network resources (throughput) among flows. In contrast, both Q-owl and Q-mvfst show a decrease in fairness as the number of flows increases.

C. Performance of Q-AIMD in handling diverse QoS requirements

In this subsection, we investigate how Q-AIMD can accommodate heterogeneous flows with different QoS requirements.

We used the same network topologies described in Section IV. Each flow is characterized by a distinct utility function U. In the simulation setup for both network topologies, we assume that half of the flows desire high throughput, and thus the utility function is defined as $U_{\max}(\lambda)$. The other half focuses on achieving a minimum throughput, and their utility function is defined as $U_{\min}(c)$.

Figures 8 and 9 show the result for the dumbbell and parkinglot network topologies. The results indicate that Q-AIMD performs relatively well in handling heterogeneous flows with varying QoS requirements.

In both dumbbell and parkinglot network topology, even changing the total number of flows, both $U_{\max}(\lambda)$ and $U_{\min}(c)$ exhibit relatively good performance. For instance, in the dumbbell network topology with a total of 53 flows, flows aiming for a minimum throughput of 0.5 [packets/ms] achieve an actual throughput of 0.4548 [packets/ms], while the other half seeking high throughput receive approximately 1.6177 [packets/ms]. In a homogeneous network where all flows aim for high throughput, each flow will receive around

$$\lambda = \frac{B}{N \cdot rtt} \approx \frac{100}{53 \cdot 2} \approx 1 \text{ [packets/ms]}. \tag{6}$$

VI. Conclusion and Future Work

In conclusion, we introduced Q-AIMD, a Q-learning-based AIMD window flow control algorithm. Q-AIMD is designed to intelligently adjust the congestion window size of a data sender within a network, while also accommodating heterogeneous flows with different QoS requirements.

From the perspective of the action set within the Q-learning framework, conventional reinforcement learning approaches rely on additive changes, such as those used by Owl and QTCP, or employ combinations of both additive and multiplicative adjustments, like MVFST-RL. In contrast, Q-AIMD

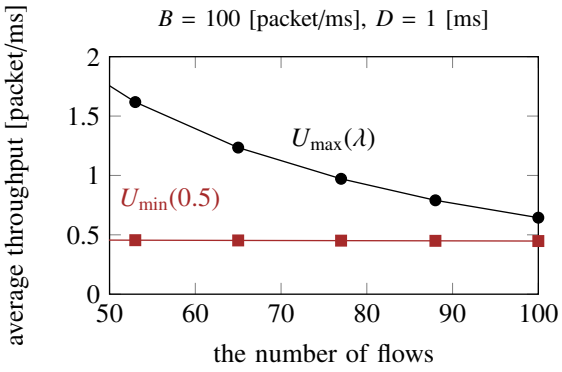

Fig. 8. Dumbbell network topology

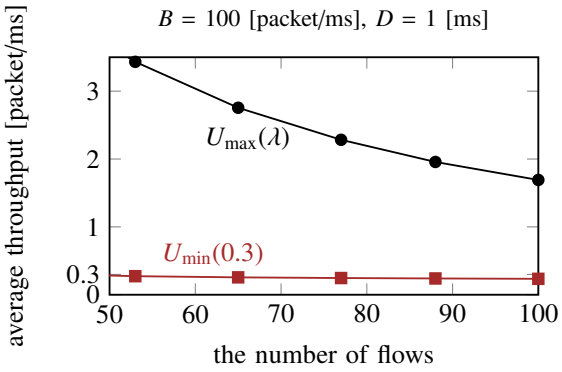

Fig. 9. Parkinglot network topology

utilizes the window size behavior in the AIMD algorithm, which is used in TCP/IP networks. Q-AIMD also supports heterogeneous flows with distinct QoS requirements, by implementing distinct reward functions tailored to these QoS specifications.

Our simulation results affirm the effectiveness of Q-AIMD and its adaptability in heterogeneous network environments, where each flow's QoS requirements differ. In future work, we plan to model the behavior of widely used TCP/IP congestion control algorithms as an action set. By examining various combinations of these action strategies in different environments, we aim to determine the optimal configuration values for action parameters tailored to network characteristics. Additionally, we plan to design an optimal reward function that aligns with the QoS requirements of the network flows.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number 24K02936.

References

[1] H. HUANG, X. ZHU, J. BI, W. CAO, and X. ZHANG3, "Machine Learning for Broad-Sensed Internet Congestion Control and Avoidance: A Comprehensive Survey," *IEEE Access*, vol. 9, pp. 31525–31545, Feb. 2021.

[2] T. Zhang and S. Mao, "Machine Learning for End-to-End Congestion Control," *IEEE Communications Magazine*, vol. 58, pp. 52–57, June. 2020.

[3] L. Zhang, Y. Cui, M. Wang, Z. Yang, and Y. Jiang, "Machine Learning for Internet Congestion Control: Techniques and Challenges," *IEEE Internet Computing*, vol. 23, pp. 59–64, Dec. 2019.

[4] R. Jain and K. Ramakrishnan, "Congestion Avoidance in Computer Networks with a Connectionless Network Layer, Part I: Concepts, Goals and Methodology," *Proceedings of the Computer Networking Symposium*, pp. 134–143, Apr. 1998.

[5] A. Gurtov, T. Henderson, S. Floyd, and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 6582, Internet Engineering Task Force (IETF), Apr. 2012.

[6] G. Vardoyan, C. Hollot, and D. Towsley, "Towards Stability Analysis of Data Transport Mechanisms: A Fluid Model and an Application," in *in Proccedings of the 37th Annual Joint Conference of the IEEE Computer and Communications (INFOCOM 2018)*, pp. 666–674, Apr. 2018.

[7] N. Cardwell, Y. Cheng, C. Gunn, S. Yeganeh, and V. Jacobson, "BBR: congestion-based congestion control," *Communications of the ACM (CACM)*, vol. 60, pp. 58–66, Jan. 2017.

[8] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "Partially Oblivious Congestion Control for the Internet via Reinforcement Learning," *IEEE Transactions on Network and Service Management*, vol. 20, pp. 1644–1659, June. 2023.

[9] Y. Chen, H. Shi, Q. Weng, and Z. Shi, "Congestion Control Design of Multicast QUIC Based on Reinforcement Learning," in *Proceedings of the International Conference on Ubiquitous Communication (Ucom 2023)*, pp. 232–236, July. 2023.

[10] W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis, "QTCP: Adaptive Congestion Control with Reinforcement Learning," *IEEE Transactions on Network Science and Engineering*, vol. 6, pp. 445–458, July. 2019.

[11] V. Sivakumar, T. Rocktäschel, A. H. Miller, and H. Küttler, "MVFST-RL: An asynchronous RL framework for congestion control with delayed actions," in *arXiv:1910.04054*, Oct. 2019.

[12] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Computer Communication Review*, vol. 25, pp. 157–187, Jan. 1995.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Bradford Books, Nov. 2018.

[14] X. Wang, S. Wang, X. Liang, D. Zhao, *et al.*, "Deep Reinforcement Learning: A Survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, pp. 5064–5078, Apr. 2024.

[15] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, *et al.*, "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 21, pp. 3133–3174, May. 2019.

[16] W. Jun and Z. Jinzhou, "Q-learning Based Radio Resources Allocation in Cognitive Satellite Communication," in *Proceedings of the International Symposium on Networks, Computers and Communications (ISNCC 2022)*, pp. 1–5, July 2022.

[17] C. Qiu, H. Yao, F. Yu, F. Xu, *et al.*, "Deep Q-Learning Aided Networking, Caching, and Computing Resources Allocation in Software-Defined Satellite-Terrestrial Networks," *IEEE Transactions on Vehicular Technology*, vol. 68, pp. 5871–5883, June 2019.

[18] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-Learning Algorithms: A Comprehensive Classification and Applications," *IEEE Access*, vol. 7, pp. 133653–133667, Sept. 2019.

[19] N. Jay, N. H. Rotman, P. B. Godfrey, M. Schapira, and A. Tamar, "A Deep Reinforcement Learning Perspective on Internet Congestion Control," in *Proceedings of the 36th International Conference on Machine Learning (ICML 2019)*, pp. 5390–5399, June. 2019.

[20] W. Kim, J. Min, Y. Son, and J. Paek, "A Recent Reinforcement Learning Trend for Vehicular Ad Hoc Networks Routing," in *Proceedings of the 14th International Conference on Information and Communication Technology Convergence (ICTC 2023)*, pp. 529–532, Oct. 2023.

[21] R. A. Nazib and S. Moh, "Reinforcement Learning-Based Routing Protocols for Vehicular Ad Hoc Networks: A Comparative Survey," *IEEE Access*, vol. 9, pp. 27552–27587, Feb. 2021.

[22] Z. A. E. Houda, D. Nabousli, and G. Kaddoum, "Cost-efficient Federated Reinforcement Learning-Based Network Routing for Wireless Networks," in *Proceedings of the IEEE Future Networks World Forum (FNWF 2022)*, pp. 243–248, Oct. 2022.

[23] C. Zhang, P. Patras, and H. Haddadi, "Deep Learning in Mobile and Wireless Networking: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 21, pp. 2224–2287, Mar. 2019.

[24] K. Winstein and H. Balakrishnan, "TCP ex Machina: Computer-Generated Congestion Control," *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM (SIGCOMM 2013)*, vol. 43, pp. 123–134, Aug. 2013.