REINFORCEMENT LEARNING BASED ACTIVE QUEUE MANAGEMENT WITH PROBABILISTIC ACTIONS

by

Caner Orkun Gölbaşı

B.S., Electronics and Communication Engineering, Istanbul Technical University, 2021

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in M.S., Computer Engineering

Boğaziçi University

2025

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my supervisor Hüseyin Birkan Yılmaz for his invaluable guidance, continuous support, and patience throughout this research process. His contributions have played a crucial role in shaping this thesis.

I would also like to express my sincere gratitude to Prof. Cem Ersoy, who made significant contributions to the quality of this study with his valuable support and suggestions throughout my research. Additionally, I would like to thank the current and former members of the Network Laboratory (NETLAB) for their unwavering support and assistance throughout my thesis.

Finally, I express my deepest gratitude to my beloved family who have believed in me throughout my life, stood by me in all circumstances, and made it possible for me to reach this point.

ABSTRACT

REINFORCEMENT LEARNING BASED ACTIVE QUEUE MANAGEMENT WITH PROBABILISTIC ACTIONS

With the increasing internet traffic density, queue management problems in routers have become more important. Classical Active Queue Management (AQM) algorithms are inadequate to adapt to variable internet traffic conditions due to fixed threshold values and parameter adjustment requirements. In this thesis, a Q-learning based AQM algorithm named Continuous Adaptive Learning Active Queue Management (CoAL-AQM) is developed to improve the performance of AQM algorithms.

CoAL-AQM analyzes the network status and makes decisions not only according to the queue occupancy but also according to the protocol type of the incoming packet and the packet drop rate of the flows. CoAL-AQM uses probabilistic actions labeled according to action probabilities ranging from 0.0 to 1.0 when making pass, mark, and drop decisions for packets. In addition, CoAL-AQM adapts better to changing traffic conditions with its continuous learning approach instead of being trained for a certain period of time and then deployed. Experimental studies are carried out in the OMNeT++ simulation environment, and CoAL-AQM showed better performance than classical algorithms. CoAL-AQM maintained its performance superiority in traffic scenarios with varying densities.

This thesis study shows that better queue management approaches can be provided with Reinforcement Learning based AQM approaches with different state, action, and reward mechanisms to queue management problems. CoAL-AQM offers an AQM solution that can adapt itself to different conditions, without the need for parameter tuning, and can balance delay, throughput, and packet loss ratio.

ÖZET

OLASILIKSAL EYLEMLER KULLANAN PEKİŞTİRMELİ ÖĞRENME TABANLI AKTİF KUYRUK YÖNETİMİ

İnternet trafiği yoğunluğunun artmasıyla birlikte, yönlendiricilerdeki kuyruk yönetimi sorunları daha da önemli hale gelmiştir. Klasik Aktif Kuyruk Yönetimi (AQM) algoritmaları, sabit eşik değerleri ve parametre ayarlama gereksinimleri nedeniyle değişken internet trafiği koşullarına uyum sağlamakta yetersiz kalmaktadır. Bu tezde, AQM algoritmalarının performansını iyileştirmek için Sürekli Uyarlamalı Öğrenme Aktif Kuyruk Yönetimi (CoAL-AQM) adlı Q-öğrenme tabanlı bir AQM algoritması geliştirilmiştir.

CoAL-AQM, ağ durumunu analiz ederek kararlarını sadece kuyruk doluluğuna göre değil, gelen paketin protokol türüne ve akışların paket bırakma oranına göre de verir. CoAL-AQM, paketler için geçme, işaretleme ve bırakma kararları verirken 0,0 ile 1,0 arasında değişen eylem olasılıklarına göre etiketlenen olasılıksal eylemleri kullanır. Ayrıca CoAL-AQM, belirli bir süre eğitilip daha sonra devreye alınmak yerine, sürekli öğrenme yaklaşımıyla değişen trafik koşullarına daha iyi uyum sağlar. OMNeT++ simülasyon ortamında deneysel çalışmalar yürütülmüş ve CoAL-AQM'nin klasik algoritmalara göre daha iyi performans gösterdiği görülmüştür. CoAL-AQM, değişen yoğunluklardaki trafik senaryolarında performans üstünlüğünü korumuştur.

Bu tez çalışması, kuyruk yönetimi sorunlarına farklı durum, eylem ve ödül mekanizmalarıyla Güçlendirme Öğrenmesi tabanlı AQM yaklaşımlarıyla daha iyi kuyruk yönetimi yaklaşımlarının sağlanabileceğini göstermektedir. CoAL-AQM, parametre ayarlaması gerektirmeden farklı koşullara uyum sağlayabilen ve gecikme, verim ve paket kaybı oranını dengeleyebilen bir AQM çözümü sunmaktadır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZET v
LIST OF FIGURES ix
LIST OF TABLES xii
LIST OF SYMBOLS xiii
LIST OF ACRONYMS/ABBREVIATIONS xv
1. INTRODUCTION 1
1.1. Problem Definition and Motivation 1
1.1.1. Queue Management Issues Arising from Increased Network Traffic 1
1.1.2. Limitations of Traditional AQM Methods 2
1.1.3. Potential of Reinforcement Learning-based Approaches 2
1.2. Thesis Purpose and Contributions 3
1.3. Thesis Scope and Organization 4
2. LITERATURE REVIEW 5
2.1. Active Queue Management (AQM) Methods 5
2.2. Reinforcement Learning and Network Applications 7
2.3. Evaluation of Related Studies 8
3. BACKGROUND INFORMATION 9
3.1. Queueing Theory Fundamentals 9
3.2. Active Queue Management (AQM) 11
3.3. TCP-IP Basics 15
3.3.1. IP Layer and Packet Routing 17
3.3.2. TCP Layer 17
3.4. Basic Principles of Reinforcement Learning 19
3.5. RL Based AQM Model 23
4. PROPOSED METHOD / MODEL 26
4.1. System Model and Network Topology 26

4.2. Details the Proposed CoAL-AQM Algorithm 29
4.2.1. Dynamic Threshold Value Calculation 29
4.2.2. State Vector s_t 30
4.2.3. Action Space A and Action Value a_t 31
4.2.4. Reward r_t 35
4.2.4.1. Queue Length Cost $f_1(q_t^{inst})$ 36
4.2.4.2. Drop/Pass Reward Cost f_2(drop/mark) 36
4.2.4.3. ratioBit Penalty f_3(ratioBit) 37
4.2.5. Q-table 38
4.2.6. Exploration-Exploitation Strategy 40
4.2.7. Gamma Factor 41
4.3. Implementation Steps 41
4.3.1. Packet Arrival and State 41
4.3.2. Selecting the State Corresponding Action 42
4.3.3. Making Decision for Package Based on Action Value 42
4.3.4. Reward Calculation 43
4.3.5. Updating Q-table 44
4.3.6. Output Signals and Statistics Collection 44
4.4. Test Scenarios 45
4.4.1. Constant Density Traffic Scenario 46
4.4.1.1. High Traffic 46
4.4.1.2. Medium Traffic 47
4.4.2. Dynamic Traffic Scenario 47
5. EXPERIMENTAL STUDIES AND RESULTS 49
5.1. Comparison Criteria and Measurement Methodology 49
5.1.1. System Parameters 49
5.1.1.1. Queue Delay 49
5.1.1.2. Packet Loss Rate 49
5.1.1.3. Throughput 49
5.1.1.4. Sender Based Loss Rate 50
5.1.2. Measurement Methodology 50

5.2. CoAL-AQM / RED /ARED Results Comparison 53
5.2.1. Scenario A. Constant Density Traffic Scenario 53
5.2.1.1. High Traffic 53
5.2.1.2. Medium Traffic 58
5.2.2. Scenario B-Traffic Scenario with Changing Density 63
6. CONCLUSIONS AND FUTURE WORK 70
6.1. Results 70
6.2. Summary of Contributions 71
6.3. Future Directions 72
REFERENCES 73

LIST OF FIGURES

Figure 1.1. Bufferbloat: latency bursting parallel to increasing queue length. . 1

Figure 1.2. High-level block diagram of the proposed CoAL-AQM (Continuous Adaptive Learning Active Queue Management) architecture. . . . 3

Figure 3.1. M/M/1 Queue system. 9

Figure 3.2. Decision-making flow: classical threshold-based AQM versus learning-based CoAL-AQM. 12

Figure 3.3. Components of the CoAL-AQM state vector at time step t. 14

Figure 3.4. TCP/IP layers and problems that we focused. 16

Figure 3.5. Probabilistic action selection in CoAL-AQM: when the queue is highly occupied and the best action is 0.2, the system probabilistically chooses between the actions of pass 20%, mark 44% and drop 36% (The trend value for the ECN incompatible packet is also assumed to be 0). 24

Figure 4.1. Network topology. 27

Figure 4.2. Topology diagram (created using OMNeT++ simulation environment under academic public license for research purposes). 45

Figure 5.1. Scenario A high traffic performance results. (a) Queue delay performance. (b) Network throughput. (c) Packet loss rate. 54

Figure 5.2. Scenario A high traffic PLR comparison. 55

Figure 5.3. Scenario A high traffic ratioBit performance results. (a) Queue delay performance. (b) Network throughput. (c) Packet Loss Rate. 56

Figure 5.4. Scenario A high traffic ratioBit flow awareness results. (a) Flow based PLR comparion. (b) Deviation of flow PLR from the mean. 57

Figure 5.5. Scenario A medium traffic performance results. (a) Queue Delay Performance. (b) Network Throughput. (c) Packet Loss Rate. . . 59

Figure 5.6. Scenario A mid traffic PLR comparison. 60

Figure 5.7. Scenario A medium traffic ratioBit performance results. (a) Queue Delay Performance. (b) Network Throughput. (c) Packet Loss Rate. 61

Figure 5.8. Scenario A medium traffic ratioBit flow awareness results. (a) Flow Based PLR Comparion. (b) Deviation of Flow PLR from the Mean. 62

Figure 5.9. Scenario B traffic performance results. (a) Queue Delay Performance. (b) Network Throughput. (c) Packet Loss Rate. 63

Figure 5.10. Scenario B PLR comparison. 65

Figure 5.11. Scenario B traffic ratioBit performance results. (a) Queue Delay Performance. (b) Network Throughput. (c) Packet Loss Rate. . . 66

Figure 5.12. Scenario B traffic ratioBit flow awareness results. (a) Flow Based PLR Comparion. (b) Deviation of Flow PLR from the Mean. . . . 67

Figure 5.13. Scenario B q-table adaptation frobenius analysis. (a)Frobenius Norm Over Time Intervals (0-1000). (b)Frobenius Norm with 5-Point Moving Average. (c)Average Frobenius Norm by 5-Point Groups. 68

LIST OF TABLES

Table 4.1. Simulation parameters. 28

Table 4.2. CoAL-AQM state vector components. 31

Table 4.3. Reward/penalty values according to q_t^{inst}. 36

Table 4.4. CoAL-AQM performance results with different alpha values. 40

Table 5.1. Network and algorithm parameters. 51

Table 5.2. Traffic scenario parameters. 52

Table 5.3. Dynamic traffic scenario phases (Scenario B). 53

LIST OF SYMBOLS

A	Action space
a_t	Action value
A'	Best action in the next state S' (in Bellman equation)
C	Link capacity
c	Number of servers in queueing system
f_1	Component of the reward function related to queue length cost
f_2	Component of the reward function related to drop/pass reward cost
f_3	Component of the reward function related to $ratioBit$ penalty
K	System capacity (buffer size)
L	Average queue length; Average packet size
max_p	Maximum drop probability parameter for RED/ARED algorithms
max_{th}	Maximum queue length threshold parameter for RED/ARED algorithms
min_{th}	Minimum queue length threshold parameter for RED/ARED algorithms
N	Customer population
n	Number of packet arrivals
P	State transition probability function in MDP
$N(t)$	Number of arrivals in time interval t
P_{drop}	Drop probability
P_{drop_ratio}	Drop ratio
P_{drop}^{base}	Base drop probability
P_{drop}^{trend}	Trend-based drop probability
$P_{drop}^{critical}$	Critical drop probability
$P_{leftover}$	Leftover probability (1 - P_{pass})
P_{mark}	Mark probability

P_{pass}	Pass probability
q_t^{avg}	Instantaneous queue-length/threshold ratio
q_t^{inst}	Instantaneous queue-length/threshold ratio
$Q(s, a)$	Q-value for state-action pair
Q_t^{avg}	Average queue length at time t
Q_t^{inst}	Instantaneous queue length at time t
R	Reward function
r_t	Reward at time t
r_t^{avg}	Scaled average occupancy ($Q_{\text{avg}}/3$ threshold)
r_t^{inst}	Scaled instantaneous occupancy ($Q/3$ threshold)
S	State space
s	State index
s_t	State vector at time t
t	Time
u	Random number for decision making
W	Average system residence time
$weightedNextQ$	Weighted average of next state Q-values used for Q-table update in CoAL-AQM
w_q	Queue weight parameter
α	Learning rate
α_{EWMA}	Average queue EWMA weight factor
α_{ARED}	ARED algorithm parameter
β_{ARED}	ARED algorithm parameter
γ	Discount factor
ϵ	Exploration rate
λ	Arrival rate
μ	Service rate
ρ	Utilization factor

LIST OF ACRONYMS/ABBREVIATIONS

ACK	Acknowledgment
AIMD	Additive Increase, Multiplicative Decrease
AQM	Active Queue Management
ARED	Adaptive Random Early Detection
ARP	Address Resolution Protocol
BGP	Border Gateway Protocol
bps	bits per second
CoAL-AQM	Continuous Adaptive Learning Active Queue Management
CoDel	Controlled Delay
cwnd	Congestion Window
DARPA	Defense Advanced Research Projects Agency
DQN	Deep Q-Networks
DRAM	Dynamic Random Access Memory
ECE	ECN-Echo
ECN	Explicit Congestion Notification
EWMA	Exponentially Weighted Moving Average
FIFO	First In First Out
FIN	Finish
FTP	File Transfer Protocol
GB	Gigabyte
HTTP	HyperText Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
INET	Internet Network Simulation Framework
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
KB	Kilobyte
LTE	Long Term Evolution

MB	Megabyte
Mbps	Megabits per second
MDP	Markov Decision Process
ms	millisecond
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
OMNeT++	Objective Modular Network Testbed in C++
OSPF	Open Shortest Path First
OSI	Open Systems Interconnection
PI	Proportional-Integral
PIE	Proportional Integral Enhanced
PLR	Packet Loss Rate
PPP	Point-to-Point Protocol
RED	Random Early Detection
RFC	Request for Comments
RL	Reinforcement Learning
RLAQM	Reinforcement Learning AQM
RR	Round Robin
RTT	Round-Trip Time
s	second
SYN	Synchronize
TCP	Transmission Control Protocol
TTL	Time to Live
UDP	User Datagram Protocol
VoIP	Voice over Internet Protocol
Wi-Fi	Wireless Fidelity

1. INTRODUCTION

1.1. Problem Definition and Motivation

1.1.1. Queue Management Issues Arising from Increased Network Traffic

Today's internet infrastructure is experiencing a rapid scaling. According to Sandvine's The Global Internet Phenomena Report, daily internet traffic worldwide has reached approximately 33 Exabytes, and data consumption per user is 4.2 GB per day. [1] Just eight major companies account for %65 of all internet traffic [1]. Network traffic is increasing day by day. With this increase, queue management problems are becoming more important. As the queue length increases, packet delays also increase and packet losses occur. With the increase in the transmitted data volume, especially during periods of intense internet use, the number of packets waiting in router queues increases and packet losses occur. Although this density may seem like mere capacity consumption at first glance, it also causes a problem called bufferbloat.

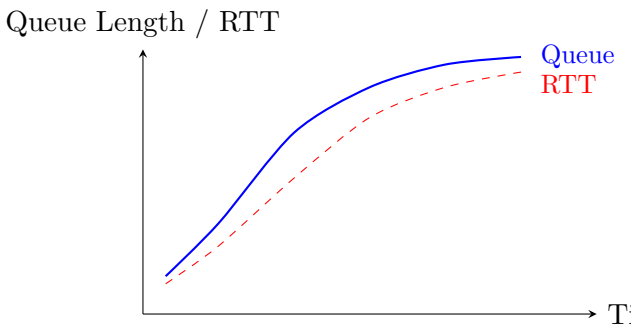


Figure 1.1: Bufferbloat: latency bursting parallel to increasing queue length.

Very large buffers increase the Round-trip time (RTT) values and cause packet loss. End users experience these situations as lag in games, freezing in videos, or slower access to the internet. Latency bursts cause problems such as deterioration in service quality as well as imbalance between flows.

1.1.2. Limitations of Traditional AQM Methods

Traditional queue management algorithms such as Random Early Detection (RED), Adaptive RED (ARED) and their different variants have been developed for queue management. These algorithms try to manage the queue by using the information about the number of packets in the queue and performing operations such as packet dropping and packet marking. Parameters such as threshold values of traditional algorithms need to be adjusted according to the traffic load, and when set incorrectly, delay and packet loss increases. This situation forces network administrators to adjust parameters for different network topologies, different working hours or different traffic types.

The static decision structure of traditional algorithms is also a problem, fixed threshold values cannot fully adapt to rapid load fluctuations [2]. Newer and parameterless algorithms such as Controlled Delay (CoDel) also make deterministic drops and may unnecessarily drop real-time traffic such as User Datagram Protocol (UDP) since they do not look at flows or flow protocol types [3]. In addition, in classical Active Queue Management (AQM) algorithms, the algorithm looks at the individual packet during another incomplete drop. No distinction is made according to flows coming from the same Internet Protocol (IP) or port. In terms of traditional queue management algorithms, parameter sensitivity, static threshold value structure, lack of flow and traffic type information, and lack of TCP-UDP traffic distinction are shortcomings.

1.1.3. Potential of Reinforcement Learning-based Approaches

Unlike traditional active queue management methods, reinforcement learning-based queue management can instantly determine the action to be taken for an incoming packet according to the learning process. The fact that different RL-based AQMs can be created using many inputs such as queue length, delay, packet loss, flow-based traffic, and traffic type also promises hope in developing AQM algorithms that can respond to different needs. The need to adjust the threshold value for adaptation to

different traffic types is a disadvantage for traditional AQMs. RL-based approaches provide advantages in this respect. RL-based algorithms can train themselves and optimize themselves to make the best decisions for network traffic, and adopt to the changes.

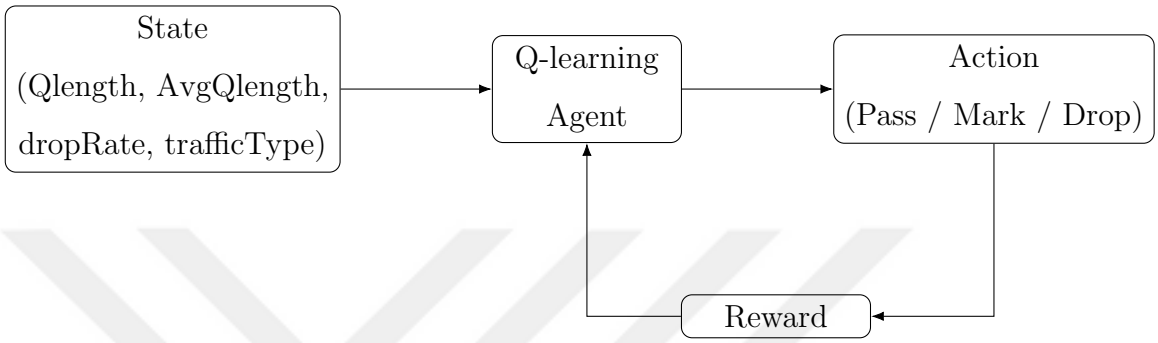


Figure 1.2: High-level block diagram of the proposed CoAL-AQM (Continuous Adaptive Learning Active Queue Management) architecture.

1.2. Thesis Purpose and Contributions

The aim of this thesis is to propose an innovative AQM based on Q-learning to overcome issues such as parameter sensitivity of classical AQM algorithms and adaptation problems to different traffic conditions. It is aimed to obtain an AQM that prevents queue overflows, reduces packet losses, decreases delay, increases throughput, makes decisions about packets by distinguishing traffic types and has flow awareness.

This thesis includes traffic type awareness and flow awareness in the designed RL-based AQM. It increases the sensitivity at these points by adding parameters such as packet type, flow-based drop rate in addition to the number of packets in the queue for pass, mark, and drop decisions for incoming packets. Another innovation is the design of a hybrid action space with a multi-objective reward function. The objectives of reducing delay, increasing throughput, increasing fairness between flows and limiting the total drop rate are combined in a single algorithm. (In the algorithm design, the criteria of reducing delay, increasing throughput and controlling the total drop rate were taken into consideration.) The algorithm is provided to adapt to changing traffics

with a continuously learning approach. Thus, this thesis introduces the concept of flow and traffic type sensitive multi-objective CoAL-AQM (Continuous Adaptive Learning Active Queue Management) to the literature and presents an AQM design that provides delay-drop probability-throughput balance.

1.3. Thesis Scope and Organization

This thesis deals with the design and evaluation of a new AQM algorithm based on Q-learning. The scope of the work is limited to the detailed examination and simulation of CoAL-AQM, which includes innovations that take into account buffer overflows, delay increase and flow-based drop rates in network traffic.

Overview of thesis chapters: Section 1 describes the problem definition, motivation, and objectives. Section 2 describes the literature review of approaches in the field. Section 3 describes the proposed method. Section 4 describes the implementation of the developed method. Section 5 describes the experimental results. Section 6 describes the future work and thesis conclusion.

2. LITERATURE REVIEW

2.1. Active Queue Management (AQM) Methods

Active Queue Management (AQM) is a method developed for congestion control in routers in the network. Instead of making a decision to drop a packet when the queue is full like the traditional DropTail mechanism, they make decisions to drop or mark a packet by looking at different queue parameters before the queue is completely full. In this section, the working principles, advantages and disadvantages of the classical AQM algorithms that are prominent in the literature will be examined.

Random Early Detection (RED) is one of the first and most effective AQM algorithms proposed by Floyd and Jacobsen in 1993 [4]. There are predetermined min_{th} and max_{th} threshold values in the RED algorithm. The average queue length is calculated and compared with these two threshold values. If the average queue length is below min_{th}, the packet is accepted to the queue. If the average queue length is between min_{th} and max_{th}, the packet is dropped with a linearly increasing probability value or marked with ECN. If the average queue length is above max_{th}, the packet is dropped, if it is ECN compatible, the packet is marked instead of dropping. The main advantage of RED is that it prevents global synchronization by preventing synchronization in TCP flows. However, the biggest disadvantage of the algorithm is that four critical parameters (min_{th}, max_{th}, max_p, w_q) must be adjusted according to the network conditions. Wrong parameter selection can lead to excessive packet dropping or high queue waiting time. In addition, RED cannot adapt to dynamic traffic conditions with its fixed parameters. It cannot distinguish between different traffic types such as TCP and UDP.

Adaptive RED (ARED) was proposed by Floyd et al. in 1999. ARED aims to solve the parameter tuning problem of RED [5]. ARED adjusts the maximum drop probability (max_p) according to the queue changes. If the average queue length is above

the target range, max_p is increased, if it is below, max_p is decreased. This approach allows ARED to adapt better to changing traffic conditions. The advantage of ARED is that the number of parameters that network administrators have to intervene in is reduced by automatically adjusting the max_p parameter. Better performance is provided by trying to keep the queue length within the target range. The remaining parameters in RED must be adjusted manually here as well. ARED cannot adapt to sudden traffic changes as fast as RED. It lacks flow and protocol awareness.

Controlled Delay (CoDel) was proposed by Nichols and Jacobsen in 2012. CoDel is a delay-based AQM algorithm. It does this by measuring the sojourn time of the packet instead of the queue length [6]. CoDel can operate in the target delay range of 5ms to 100ms. If the minimum queue delay remains above the target queue delay, CoDel starts dropping packets.

The biggest advantage of CoDel over other algorithms is that it is presented without parameters. Its delay-centered approach is successful in solving the bufferbloat problem and is not affected by changing packet sizes. In addition, the algorithm's time keeping for each packet increases the computational load. In addition, it does not consider which flow to drop packets when making drop decisions, which may cause packets to be dropped disproportionately from some flows. CoDel, like other AQMs, does not consider the protocol type and may unnecessarily drop real-time traffic without making a distinction between TCP/UDP.

Proportional Integral Enhanced (PIE) was proposed by Pan et al. in 2013. PIE uses a PI (Proportional-Integral) controller to keep the queue delay at a target value [7]. PIE dynamically adjusts the packet drop probability by looking at the delay trend and the current queue delay. PIE has lower computational complexity and uses less memory than CoDel. However, the PIE algorithm also has limitations such as setting parameters such as target update interval and optimizing according to traffic. PIE is also not flow aware and cannot distinguish packets according to protocol types.

2.2. Reinforcement Learning and Network Applications

Reinforcement Learning (RL) is a machine learning technique in which an agent learns optimal behavior through trial and error by interacting with its environment. In dynamic network environments, the application of RL to network management problems has attracted attention in recent years.

RL problems are modeled by Markov Decision Processes (MDP). An MDP is defined by (S, A, P, R, γ). S is the state space, A is the action space, P is the transition probabilities, R is the reward function and γ is the discount factor [8]. One of the most popular model-free RL algorithms is Q-learning [9]. Q-learning learns the optimal policy without knowing the model of the environment and has a relatively simple implementation.

RL applications have become more frequently used in network management in recent years. Mao et al. developed a system that uses RL to select adaptive bit rate in video streaming applications called Pensie [10]. Jay et al. proposed a method that uses RL to perform congestion control [11].

There are limited studies on Q-learning applications to AQM problems. Su et al. replaced the fixed decision structure of the classical RED algorithm with a Q-learning agent in their method called QRED. QRED takes the queue length information used in RED and adjusts the packet drop probability with the Q-learnimg agent instead of managing it with min_{th}, max_{th} and max_p parameters [12]. This study was one of the first examples of the use of RL agents in AQMs. It adjusted the max_p values adaptively according to the state of the network and looked at the average queue length in the state. Jiang et al. managed the fixed max_p value of the RED algorithm with Q-learning in their method called RLAQM (Reinforcement Learning AQM). In order to solve the parameter sensitivity problem of RED, they added the change in the average queue length to the state along with the average queue length. The system calculated the reward values using the function that maximizes throughput and minimizes delay [13].

DQN (Deep Q-Networks) is an approach that uses deep neural networks to process high-dimensional state spaces. Kim et al. proposed an algorithm for AQM using DQN [14]. There are significant disadvantages to implementing DQN-based approaches on routers. First, deep neural networks require libraries such as Tensorflow or PyTorch. Running these libraries on resource-constrained routers can lead to performance issues. In addition, the DQN approach usually involves deploying offline trained models, which makes it difficult to adapt to dynamically changing traffic conditions.

2.3. Evaluation of Related Studies

Traditional AQM algorithms have shortcomings such as parameter sensitivity, static structure, flow awareness and protocol discrimination. RED and similar ones require continuous parameter tuning because the optimal parameter values depend on the traffic conditions. Newer algorithms such as CoDel and PIE reduce the number of external parameters but are still based on fixed assumptions. They may have difficulty adapting to variable traffic conditions.

The shortcomings of existing RL-based AQM studies are as follows: First, most of them do offline training and have limited dynamic adaptation capabilities. Second, it is not practical to implement complex architectures on routers. Third, existing studies generally do not take protocol types into account and are not flow aware. Q-learning-based studies are developed based on the features of RED and are not innovative enough.

This thesis presents an innovative AQM algorithm to address the deficiencies in the literature. The proposed approach can be practically implemented in router environments using Q-learning with low computational complexity. Flow and protocol awareness are added to the algorithm by adding protocol separation and ratioBit mechanism. Thanks to the continuous learning mechanism, the algorithm can continuously adapt dynamically to variable traffic conditions.

3. BACKGROUND INFORMATION

3.1. Queueing Theory Fundamentals

Queuing theory is the mathematical analysis of waiting lines in systems with limited resources. Many systems in computer networks, such as packet switching nodes and router models, are modeled using queuing theory.

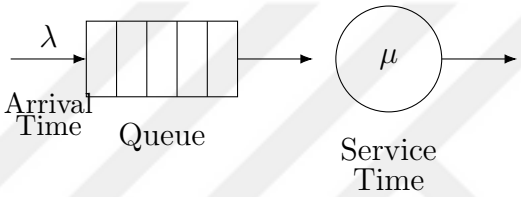


Figure 3.1: M/M/1 Queue system.

A queue consists of three components: arrival time, service time, and queue discipline shown in Figure 3.1. Arrival time defines the arrival times and arrival rates of packets to the system. One of the most common arrival process models is the Poisson process, where the interarrival time has an exponential distribution. In Poisson processes, the probability of n customers arriving in a unit time is expressed by the formula

$$\Pr\{N(t) = n\} = e^{-\lambda t}\frac{(\lambda t)^n}{n!}, \tag{3.1}$$

where λ is the average arrival rate [15].

The time required to service, process, and forward each packet is called service time. It usually depends on the packet size and the connection speed. Service time can be deterministic or stochastic. In network routers, service time is usually directly proportional to packet size.

Queue discipline also determines the order in which the customers will be served. The most common queue discipline is First In First Out (FIFO). The first packet to

arrive is the first to exit the queue. Another important queue discipline is Priority Queue. Packets are ranked according to their priority, and packets with higher priority are processed first. Round Robin (RR) queue discipline is a queue discipline in which each customer is served in order for a certain amount of time.

Kendall notation was proposed by D.G. Kendall for a standard description of queuing systems [16]. It is written in the format A/S/c/K/N/D where A is the arrival process distribution, S is the service time distribution, c is the number of servers, K is the system capacity (number of buffers), N is the customer population, and D is the queuing discipline These queueing systems are studied to optimize the system performance.

Network congestion is a queuing problem that needs to be handled by the queueing system. Congestion occurs when packets accumulate in router queues when the service time is lower than the arrival rate. The basis of this phenomenon was demonstrated by Kleinrock's Queuing Systems study, which showed that the waiting states at network nodes are subject to stochastic principles similar to standard counter-customer queues. Kleinrock's formulation emphasizes that even if the service rate and traffic rate are constant, as the variance of incoming traffic increases, the queuing distribution will be higher and the delay will increase dramatically. This is one of the reasons for the delay explosions referred to today as bufferbloat. For example, $M/M/1$ queue is the most studied queue model. $M/M/1$ queue means exponential arrivals, exponential service times and single server. There are analytical solutions for steady-state case. For steady-state case, average queue length is calculated as

$$L = \frac{\rho}{1-\rho}, \tag{3.2}$$

average waiting time is calculated as

$$W = \frac{1}{\mu - \lambda} \tag{3.3}$$

[17].

Little's Law is one of the most fundamental and powerful results of queuing theory. Proven by John Little in 1961, this law shows that for any steady-state queuing system, the relationship

$$L = \lambda W \tag{3.4}$$

holds between the average queue length (L), the arrival rate (λ), and the average system residence time (W) [18]. This law is valid regardless of the arrival and service distributions and is used in the analysis of queuing systems. In a stable system, the average queue length (L) is equal to the product of the arrival rate (λ) and the average system residence time (W). In other words, in order to reduce the delay in a router, it is necessary to reduce the packet arrival rate or redesign the queue management policy. Since millions of packets arrive at routers in current internet networks, the arrival rate (λ) cannot be fixed, so research has shifted to providing control on the waiting time (W) and queue length (L).

Classical AQMs provide this control by continuously monitoring the queue length and randomly dropping packets or marking them with Explicit Congestion Notification (ECN) when predetermined thresholds are exceeded.

3.2. Active Queue Management (AQM)

Network congestion is a waiting line problem in terms of queuing theory. Packets accumulate in router queues when the service rate is lower than the current demand. Even if the service rate and average traffic rate are constant, the queue length distribution becomes even more bloated as the variance of incoming traffic increases. This is one of the main reasons for the excessive delay problem, which is today called bufferbloat [19].

The first buffer management implementations were passive. In the queue management approach known as droptail, packets are accumulated until the FIFO queue

is full, and when this is reached, newly arrived packets are dropped. This approach causes a lot of packet loss and low link utilization.

AQM was developed to solve these problems. The basic idea of AQM is to send a congestion signal to the sender by *dropping* or *marking* packets before the buffer is completely full. Random Early Detection (RED), proposed by Floyd and Jacobson in 1993, is one of the first and effective AQM algorithms [4].

Therefore, queueing theory reveals why AQM is necessary and draws attention to the relationship between the number of packets waiting in the queue and the delay. This thesis presents a new AQM policy based on Reinforcement Learning at this point.

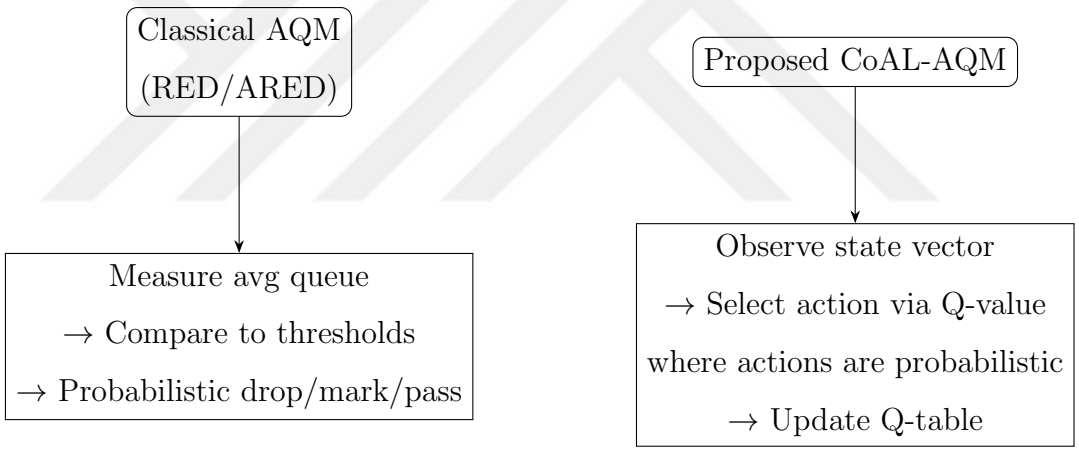


Figure 3.2: Decision-making flow: classical threshold-based AQM versus learning-based CoAL-AQM.

Each router in the network can be viewed as a queueing system that operates between an *arrival process* and a *service process*. The arrival process describes how packets arrive at the router over time, while the service process determines how quickly the router can process and forward these packets.

Although traffic in real networks exhibits self-similar properties (meaning traffic patterns repeat at different time scales), many analytical studies adopt *Poisson* ar-

rivals for mathematical tractability. A Poisson process is a mathematical model that describes events occurring randomly over time with a constant average rate.

For a Poisson arrival process with average rate λ (packets per second), the probability of observing exactly n packet arrivals in a time interval of length t is given by:

$$\Pr\{N(t) = n\} = e^{-\lambda t}\frac{(\lambda t)^n}{n!} \tag{3.5}$$

where $\Pr\{N(t) = n\}$ denotes the probability that $N(t)$ (the number of arrivals in time interval t) equals n. This formula reflects the memoryless nature of the Poisson process, where discrete packet arrival events are governed solely by the mean arrival rate λ, and future arrivals are independent of past arrival history.

Three indicators stand out when measuring the success of AQM algorithms: queue delay, packet drop rate, and throughput. Throughput is used to show how close the physical capacity of the network is, and is usually reported in the literature in bits per second (bps). Drop rate not only gives information about the rate of dropped packets, but also about how aggressive the early drop policies are and how much TCP is triggered by packet drop attempts. Similarly queue delay is a critical parameter of user experience. In terms of flow level fairness, classical AQMs look at individual packets and make decisions about the packet. However, in real networks, flow level fairness is also required to evaluate resource sharing. One of the algorithms developed within the scope of this thesis is based on Jain's famous fairness criterion approach, the ratioBit mechanism, where the drop rate of each resource is compared with the average drop rate of all packets that have arrived so far [20].

While the majority of real-time services use UDP or UDP-based algorithms, volume-intensive transfers rely on TCP. The failure of traditional AQMs to make this distinction results in aggressive UDP losses or TCP backpressure. CoAL-AQM learns to adjust the mark-and-drop operation in the action space according to the protocol type by adding the UDP-TCP flag to the state. The benefit of a similar service class

awareness approach has also been highlighted in the fairness-delay analysis of Confucius Queue Management [21]. These metrics are used as comparison axes throughout the thesis. Detailed graphs and tables will be presented in Chapter 5.

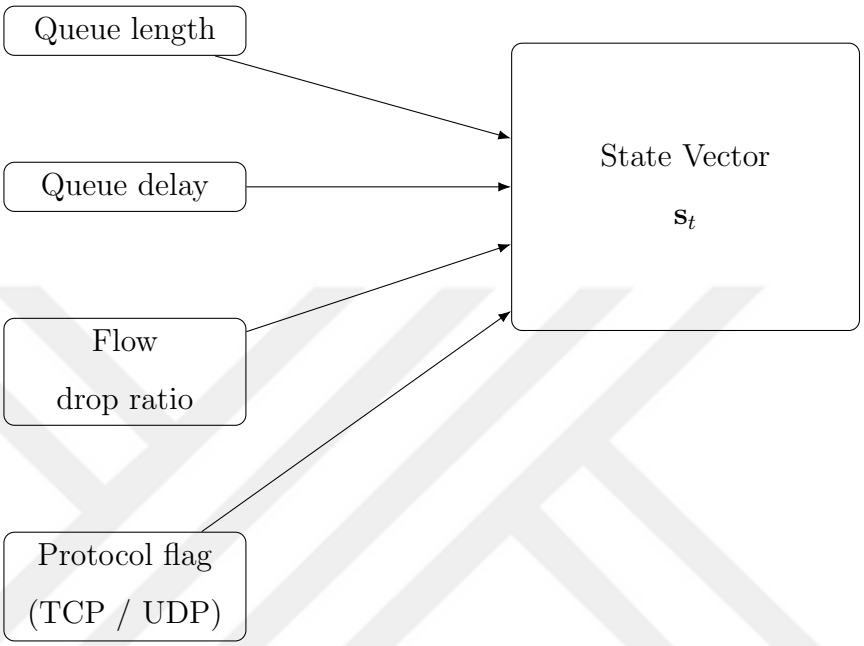


Figure 3.3: Components of the CoAL-AQM state vector at time step t.

The first buffer policies in packet-switched networks were passive, packets were accumulated until the FIFO queue was full, and when the capacity was exceeded, we felt the increased traffic with delay increases or consecutive heavy drops. In 1993, Floyd & Jacobson's Random Early Detection (RED) study added the average queue length as an active layer to this approach.

When the average queue length falls between two predetermined threshold values, the packet is randomly dropped according to a probability calculation or marked with Explicit Congestion Notification (ECN), thus reducing the windows of TCP flows and diluting the traffic before RTT growth begins. If it falls below the threshold values, the packet is added to the queue, if it falls above the threshold values, the packet is dropped. If the sender supports ECN, the packet is marked with ECN instead of drop and added to the queue, while the sender gets notified about congestion.

The basic parameters of the RED algorithm are: *minth* (minimum threshold), *maxth* (maximum threshold), maxp (maximum dropping probability), and wq (queue weight parameter). Floyd proposed the formula

$$minth = 0.5 \times C \times delay/packet_size \tag{3.6}$$

to determine the *minth* value, where C is the link capacity and delay is the targeted maximum queue *delay*. For example, in a simulation environment, the RED parameters can be set as follows: $minth = 25$, $maxth = 75$, $maxp = 0.02$, $wq = 0.002$, and $packetCapacity = 150$. Here, the value $wq = 0.002$ is the standard value suggested in the original paper and is used to filter out sudden fluctuations in the calculation of the average queue length; the smaller its value, the slower the average is updated. The value $maxp = 0.02$ is also the standard recommended value, which indicates that packets will be dropped with a 2% probability when the average queue length reaches *maxth*; increasing this value means more aggressive dropping.

In practice, the biggest problem was parameter setting. The min and max threshold values determined for different network traffics could be too loose or too aggressive. In particular, it is critical to determine the minth and maxth values correctly; very low values cause unnecessary packet loss, while very high values cause increased delay. For this reason, algorithms such as Adaptive RED (ARED) that dynamically adjust some parameters according to queue oscillation have been proposed. However, algorithms that can still achieve better results in sudden traffic changes have continued to emerge.

3.3. TCP-IP Basics

The TCP/IP protocol, which forms the backbone of the internet, has a layered architecture developed by DARPA (Defense Advanced Research Projects Agency) in the 1970s and has become the basic building block of today's internet infrastructure. This architecture, byte streams produced in the application layer are encapsulated into TCP or UDP segments in the transport layer, then into IP datagrams in the internet layer, and then into frames in the link layer and then transferred to the physical

cable. Data streams produced in the application layer are converted into TCP or UDP segments in the transport layer, then into IP datagrams in the internet layer, and then into frames in the link layer and transferred to the physical media. The encapsulation process between each layer is based on the principle that the layers add their own header information and forward the packet to the lower layer [22]. While the TCP/IP model consists of four layers (Application, Transport, Internet, and Network Access), it is often compared with the seven-layer OSI (Open Systems Interconnection) reference model, as shown in Figure 3.4. The TCP/IP model combines the functionality of the OSI's upper three layers (Application, Presentation, and Session) into a single Application layer, and merges the lower two layers (Data Link and Physical) into the Network Access layer. This layered architecture provides modularity, allowing each layer to operate independently of each other, making it easier to develop and maintain protocols.

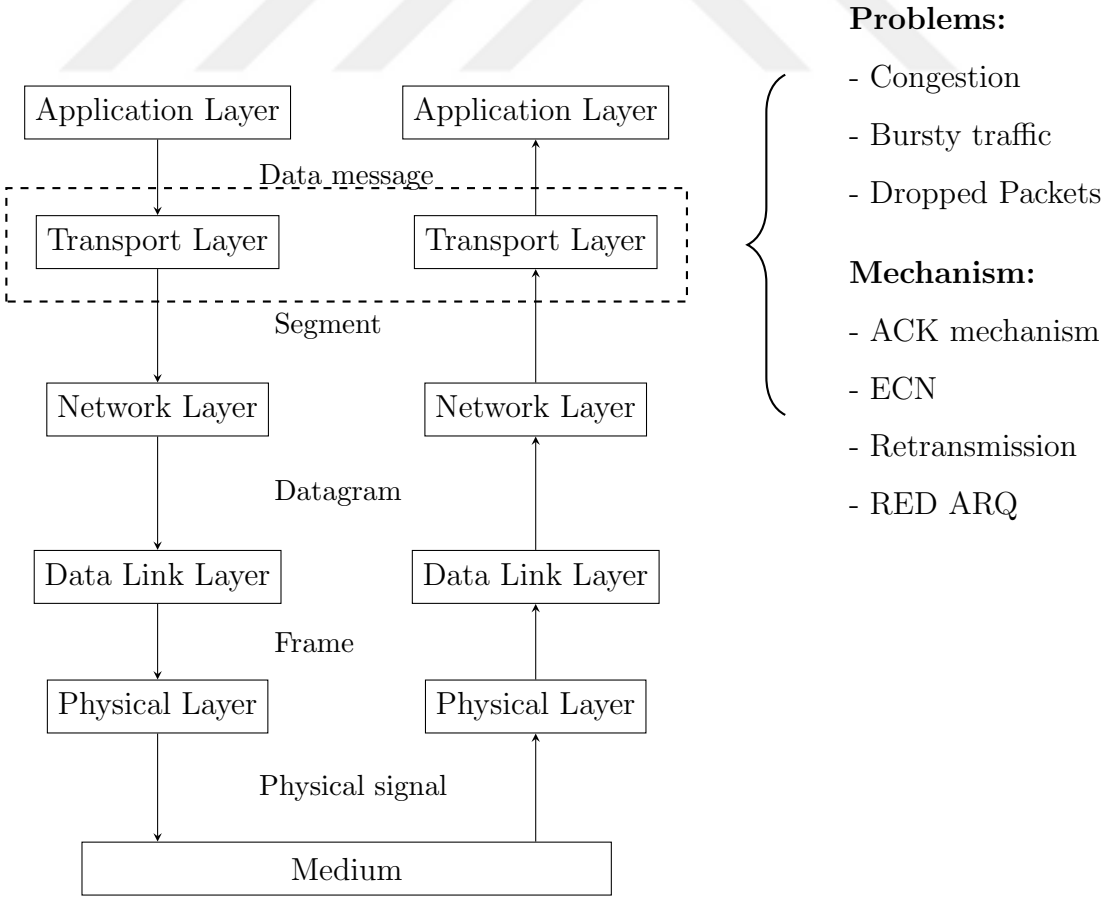


Figure 3.4: TCP/IP layers and problems that we focused.

3.3.1. IP Layer and Packet Routing

The IP (Internet Protocol) layer provides a basic datagram service for communication between networks. Each IP datagram contains fields such as source IP address, destination IP address, protocol number, Type of Service, Time to Live - TTL and fragmentation information. Routers perform the necessary operations according to this information [18].

The IP protocol implements what is known as best-effort delivery. This means that it will do its best to deliver a packet, but it does not guarantee it. During packet transmission, if the datagram size exceeds the MTU (Maximum Transmission Unit) of the underlying network, the IP layer performs fragmentation. The large datagram is broken into smaller pieces and reassembled at the destination.

3.3.2. TCP Layer

TCP (Transmission Control Protocol) is a connection-oriented and reliable protocol that runs on top of IP. TCP divides the data coming from the application layer into segments. It assigns a sequence number to the segments and the receiver reassembles these segments in the correct order. A three-way handshake mechanism consisting of SYN, SYN+ACK and ACK packets is applied to establish the connection. A four-stage handshake mechanism consisting of FIN and ACK packets is applied to terminate the connection [19].

Each transmitted segment is acknowledged with an ACK number, and if no ACK is received, the segment is sent again. TCP also provides flow control using a sliding window mechanism. This prevents buffer overflows on the receiver side by ensuring that data is sent without exceeding the receiver's processing capacity [23].

One of the most critical mechanisms of TCP is the congestion control mechanism. According to network capacity and load cwnd (Congestion Window) dynamically ad-

justs itself. TCP gradually increases the cwnd value when there is no congestion in the network. When it detects congestion, it reacts quickly by halving the cwnd size. This AIMD (Additive Increase, Multiplicative Decrease) algorithm aims to use network resources efficiently and fairly. Congestion detection and response mechanisms have been improved with developments in various variants of TCP such as Tahoe, Reno, New Reno and CUBIC.

There are two TCP congestion signals that routers can use: drop and mark. When a packet is dropped, TCP times out due to the lack of an expected ACK packet and detects congestion. Packet marking is done through the ECN (Explicit Congestion Notification) mechanism. If the ECN feature is available in the network, the router sets the ECN bit in the IP header when it detects congestion. The receiver that receives these marked packets informs the sender with the congestion notification bit ECN-Echo(ECE). Thus, the sender decreases the cwnd value without experiencing packet loss [24].

When TCP was first designed, buffers in routers were in kilobytes. With the development of technology and the decrease in hardware costs, buffer sizes increased. The increase in internet traffic and the constant fullness of buffers increased the latency per stream, causing bufferbloat [25]. Large delay fluctuations had negative effects especially on game and video traffic. TCP cannot detect these high delays with its feedback mechanism based on lost packets without AQM. Recall that TCP protocol increases cwnd when there is no congestion in the network and decreases cwnd rapidly when it detects congestion. Long-term connections benefit from this AIMD (Additive Increase, Multiplicative Decrease) rule because since the connections last long, they have the chance to increase the bandwidth again even if it decreases after congestion. On the other hand, Short-term flows are at a disadvantage in this case. In other words, the TCP-based system inherently favor long-term connections. Another important issue is that UDP traffic is insensitive to loss; since traditional AQMs do not distinguish between UDP and TCP, hence real-time flows may suffer unnecessary loss.

AQM mechanisms are designed to manage network congestion proactively. Traditional AQM algorithms make packet drop decisions based on the queue occupancy rate. However, these approaches may not effectively adapt to dynamic and variable network conditions because they are based on fixed threshold values.

There are two basic signaling situations for a router in a TCP-based system. These are packet dropping and marking. When a packet is dropped, the TCP timer detects congestion because no ACK packet has arrived. When a packet is marked, the ECE bit is set to 1, and the network end systems reduce the cwnd without experiencing any loss. While dropping a packet directly affects the throughput, the mark reduces the RTT oscillation and prevents drop increases. The selection of mark and drop margin in routers should be made wisely. Our proposed CoAL-AQM also dynamically evaluates the pass-mark-drop probabilities for the incoming packet. The agent learns to mark ECE in TCP packets and to optimize the loss in UDP packets. Section 5, which includes simulation results, shows that it reduces the total drop rate compared to RED and also reduces the average delay.

3.4. Basic Principles of Reinforcement Learning

Reinforcement Learning (RL) is a sub-branch of machine learning. It focuses on learning the optimum behavior according to reward mechanism in an environment. Unlike traditional machine learning methods, RL learns the optimum behavior through trial and error and tries to adapt to the best behavior according to the results. This method is suitable for AQMs that try to make a decision in constantly changing conditions due to internet traffic.

RL consists of a system, agent, environment, state, action, reward and policy components. Environment is the external world in which the agent is located. Agent is a component that makes decisions and performs actions according to the states in the environment it is located in. The interaction between the environment and the agent is formulated with MDP (Markov Decision Process).

At each step, the agent in the environment is represented with a state (s_t). The agent chooses an action (a_t) based on this state. As a result of this action, the environment produces a response with a new state (s_{t+1}), the next state is formed and a reward is obtained. These processes continue during the learning process. The agent implements a policy to maximize the total reward by learning in the long term [26].

An MDP can be represented by (S, A, P, R, γ). S is the state space containing all possible states. A is the action space containing all possible actions. P is the transition probability function. R is the reward function. $\gamma \in [0, 1]$ is the discount factor that determines the effect of future rewards on the current state.

The state space is the set of all possible states that an agent can have when interacting with the environment. These states contain the information necessary for the agent to make a decision. For this thesis, the state space consists of the ratio of the current queue length to the threshold, the ratio of the average queue length to the threshold, and the UDP-TCP flag information. The correct design of the state space has a direct impact on the performance of the algorithm. Different and more detailed state spaces can be created, but a very complex set of states can slow down or adversely affect the learning process.

The action space is the set of all possible actions that the agent can perform in any given state. The action space describes how the agent can interact with its environment. The action space defined can be discrete or continuous depending on whether it contains a certain or infinite number of actions. Action space has an important place in formulating the problem in RL.

Reward is the numerical response of the feedback signal received from the environment as a result of an action taken by the agent in RL. This value shows how good or bad the agent's action is for a particular situation. The agent tries to maximize the total reward for each state in the long run. It develops and learns policies in this direc-

tion. The reward function designed specifically for the problem is one of the important components that affect the agent's behavior [27].

For AQMs, the reward function can be designed to balance the goals of high throughput, low delay, and low packet drop. The discount factor γ takes into account the current value of future rewards. This maximizes the return for long-term optimization. The discount factor γ is a value between 0 and 1. The value of γ determines how much the agent values long-term rewards over short-term rewards. In network management applications such as AQM, a high value of γ is chosen to prevent short-term network changes from negatively affecting long-term network performance.

Q-learning is one of the widely used RL-based methods. The values corresponding to the state-action pairs in RL are kept in a Q-table. The reduced reward estimates of the action value function ,$Q(s, a)$, are recorded in the Q-table as a result of the agent's learning process. These values represent the expected long-term reward of the action selection of that state. The Q-value $Q(s, a)$ is the expected total reward if action a is chosen in state s. The Q-learning learning algorithm updates these values. A learning rule based on the Bellman equation is applied in the update process [8].

The Q-learning algorithm update $Q(S, A)$ using following equation

$$Q(S, A) \leftarrow Q(S, A) + \alpha\big(R + \gamma\, Q(S', A') - Q(S, A)\big), \tag{3.7}$$

where $Q(S, A)$ represents the current value of the state-action pair; R represents the reward value, S' is the next state, and A' is the best action that can be taken in that state. α is the learning rate, γ is the discount coefficient that determines how much future rewards are taken into account. With this update method, the agent improves the policy to reach the highest cumulative reward.

The learning rate α determines how effectively new information updates the Q value in the Q-table. High α gives more importance to new information, while low α slows down the change of old values [28].

As the algorithm progresses, the Q-table is updated and over time the values in the Q-table converge to the optimal policy. In order for the values to converge correctly, a learning scenario must be realized in which each state-action pair will be visited sufficiently. In the basic steps of Q-learning, the Q-table is initialized with zero values or random values. The current state from the environment is observed, the action is selected according to the action selection policy. The selected action value determines which action will be decided and the reward is calculated. The relevant state-action pair in the Q-table is updated with the Bellman Equation (3.7).

One of the most fundamental issues in RL is the exploration-exploitation dilemma. An agent has two different paths during learning: applying what it knows (exploitation) or trying what it does not know (exploration). If the agent only applies what it knows, it may get stuck with the solution it is used to, and if it constantly explores, it cannot benefit from well-known strategies. The ϵ-greedy rule balances these options with a simple dice roll method: it tries a random action with a small probability at each move, and with the remaining probability it chooses the best reward and corresponding action for its state from the Q-table. In Q-learning applications, ϵ is initially kept high, slowly lowered over time, and the learned policy is applied more frequently [29]. However, this method will not always yield good results for variable network traffic, and this issue is one of our contributions and will be discussed in the next section.

One of the important topics in RL is the exploration-exploitation dilemma. An agent has two different options during learning: applying what it knows (exploitation) or trying what it does not know (exploration). If the agent only applies what it knows, it will be stuck with the solution it is used to. If it constantly explores, it will not be able to benefit from well-known strategies. However, in computer networks with variable traffic scenarios, an RL algorithm that updates the learning rate coefficient at certain intervals and constantly adapts to new traffic will add a different perspective to this issue. The aim is to design an algorithm that will give good results not for certain scenarios but for all scenarios. The ϵ-greedy rule balances exploitation and exploration with a simple dice roll method. A random action is tried with a small probability at

each move, and the action with the best Q value from the Q-table is selected with the remaining large probability. In Q-learning applications, the exploration rate ϵ is kept high at the beginning, slowly decreased over time, and the learned algorithm is applied more frequently [29].

3.5. RL Based AQM Model

Classical AQMs rely on deterministic or semi-deterministic thresholds, the problem of manually setting parameters, and many deficiencies such as protocol differences and flow awareness occur in practice. The design idea in the Reinforcement Learning-based AQM approach is simple. Metrics such as the ratio of the number of packets in the queue to the threshold value, the change of this value over time, and tracking the drop rate of flows can be thought of as state. Then, an AQM can be created that overcomes the problems in classical algorithms by allowing the Reinforcement Learning algorithm to decide how to slow down the traffic without overflowing the queue. Thus, the need to manually adjust parameters such as minth, maxth, and pb for different traffic situations can be eliminated.

RL-based AQMs can learn queue statistics by signaling that the queue theory related data is mapped to the state vector, TCP feedback to the reward function, flow balance and traffic type to flags. This information teaches the agent to make decisions without overflowing the buffer, throttling throughput, increasing drop rate and increasing queue delay.

The state vector we define consists of four parts: the range in which the current queue length lies with respect to a threshold value, the range in which the exponentially weighted average queue length lies with respect to a threshold value, ratioBit, and the protocol flag of the packet. The current and average queue data captures the delay and buffer occupancy trend, the ratioBit allows comparisons between flows regarding drop, and the packet protocol flag separates UDP and TCP traffic. Different state vectors can be created with different combinations of these four parameters, but it should not be

forgotten that in order to present a significant new AQM against classical AQMs, the state vector must be chosen very well and the algorithm can be reached to a balanced point after many simulations.

The action space is a set where the agent will choose drop, mark, or pass. Here, instead of directly giving a deterministic action to be taken for the incoming packet, it is designed as a probabilistic action consisting of 0.0, 0.1, ..., 1.0 values. That shows the pass probability. When the action value is close to 1.0, most packets are passed; if it is close to 0.0, the probability of drop is higher. In addition, a variable decision is made according to the trend of the queue's fullness in the decision to drop or mark the packet. In this way, we avoid strict and hard threshold values and produce softer and adaptable responses according to different situations.

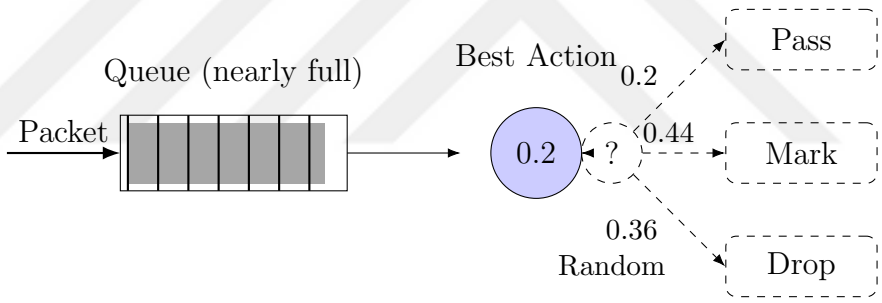


Figure 3.5: Probabilistic action selection in CoAL-AQM: when the queue is highly occupied and the best action is 0.2, the system probabilistically chooses between the actions of pass 20%, mark 44% and drop 36% (The trend value for the ECN incompatible packet is also assumed to be 0).

The reward function is calculated after the decision about the incoming packet is determined. The reward function here can be designed to have a multi-purpose structure. A reward function is created based on different information such as the ratio of the number of packets in the queue to the threshold value, unnecessary drops, the amount of packet bytes passed, and the comparison of the drop rate from a flow to the overall drop rate.

If we consider the training and testing in RL, first, the agent fills the Q-table during training and an ideal Q-table is created. Here, the agent is trained with different training scenarios. During this training period, the epsilon discovery rate is the parameter that the agent updates with a new value depending on the old reward value for the relevant field for that state and action instead of using the value in the table. The epsilon value continues to decrease during training and after a certain period of time, the change in the Q-table decreases and the training is terminated. Network traffic is in a structure where constantly changing and unpredictable different situations will be experienced. In the RL AQM approach in this thesis, training never ends, the epsilon value, which is the discovery parameter is reset at certain intervals. In this way, an algorithm that can better adapt to different traffic types and instantaneous traffic increases or decreases is proposed. For situations where the traffic is always similar, the Q-table can be recorded outside and the algorithm can be run with a discovery rate of 0.

The proposed RL AQM algorithm also has a low computational load. With the Qtable, which is formed as a result of 400 states and 11 different action values, and the comparison of 11 different numbers when each packet arrives, and the reward calculations that the algorithm will make for the Qtable, a RL AQM algorithm is created that is not very complex and independent of different libraries. Different RL applications such as DQN require more computational power and dependency on deep learning libraries such as TensorFlow and PyTorch compared to Q-learning. This creates additional software dependencies and makes the application difficult on routers.

The next section will make this abstract framework concrete: the parameter files, code flow, and experimental scenarios in OMNeT++ will be explained step by step, so that we can see in detail how the ideas described here are implemented and measured.

4. PROPOSED METHOD / MODEL

4.1. System Model and Network Topology

In this section, the OMNeT++ simulation environment is introduced in detail for the evaluation of the proposed CoAL-AQM algorithm. Our aim is to try out the queue dynamics seen in a real router under simulation conditions in a realistic way as much as possible. The Floyd–Jacobson study (1993) including RED's original verification scenarios was taken as a guide in the network design for the algorithm testing [4].

OMNet++ is an open source, modular, and discrete event simulation tool developed for network simulations. It is a C++-based simulation platform and is widely used in both academic and industrial environments. INET Framework is an open source communication network simulation package built on OMNeT++. INET provides models for internet protocols such as TCP, UDP, IPv4, IPv6, OSPF, BGP, technologies such as Ethernet, PPP, IEEE 802.11 (Wi-Fi), LTE, and routers, base stations, wireless access points, and mobile devices. OMNeT++ and INET Framework have become one of the widely used simulation tools in network research today. OMNeT++ is used in many studies published in leading academic conferences and journals [30].

In the topology, StandardHost, Router, and DatarateChannel classes provided by INET-Framework 4.5 were used directly, and the CoAL-AQM algorithm written specifically for the queue blocks. In order to make comparisons with RED and ARED, the queue selection was changed, and the network parameters remained constant.

The topology shown in Figure 4.1 shows the topology with two backbone routers (routerCore1, routerCore2). The four senders are connected to routerCore1 with 100 Mbps data rate and 1 ms delay channels. The four receivers are connected to routerCore2 with 100 Mbps data rate and 1 ms delay channels. The two routers are connected to each other with a channel of 10 Mbps data rate and 1 ms delay. Thus, the

link between the routers becomes a bottleneck that can carry much less than who total throughput capacity. This single bottleneck connection concept allows AQM responses to be measurable by modeling a queue. Since INET's configurator module is used, all IP addresses, routing tables and ARP entries are generated automatically at the beginning of the simulation.

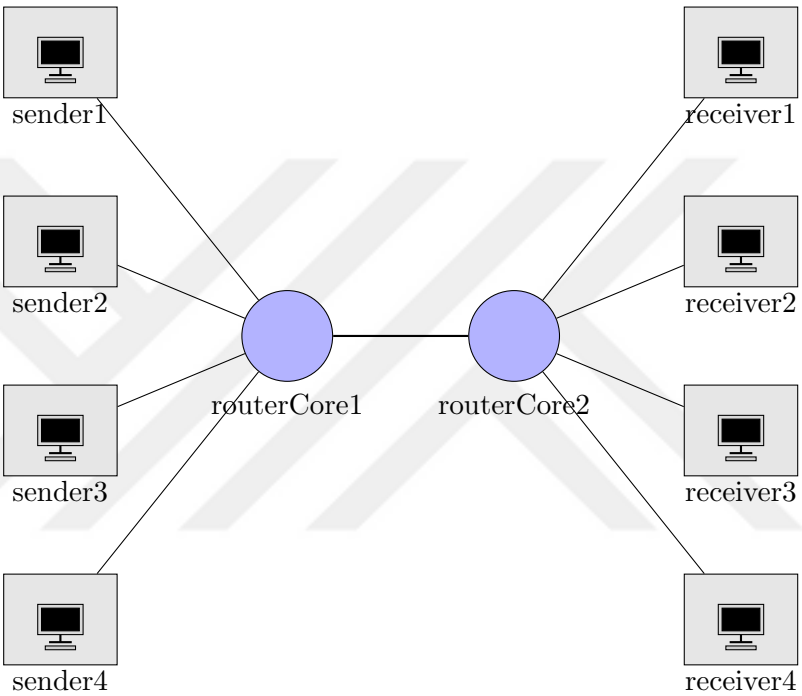


Figure 4.1: Network topology.

In the topology, user intervention is made in the queue section of the routers. The queue block to be tested is selected for the queue component in the Ethernet interfaces of the routers. The AQM algorithms are changed from the ini file. Thus, only the algorithm is changed without changing the simulation architecture and parameters. The only variable is the queue algorithm, so the observed significant performance differences can be attributed to the algorithm.

While the backbone connection is kept as 10 Mbps, the end connections are selected as 100 Mbps, thus ensuring that the total throughput capacity is much higher than the bottleneck link capacity. This ratio corresponds to oversubscription in modern networks. TCP and UDP flows are used to model realistic network traffic in the simu-

lations. TCP flows represent web traffic (HTTP-like request-response pattern) and file transfer scenarios, while UDP flows represent VoIP and video streaming traffic. Traffic density is tested at different levels as medium and high. In addition, the behavior of the algorithms against dynamic traffic scenarios and sudden traffic changes is observed. ECN support is activated in TCP connections. Considering the bandwidths and delays, the queue capacity is selected as 150 packets, thus both delay-sensitive approaches such as RED/ARED and the proposed CoAL-AQM algorithm will have difficulty in the face of delay bursts, and noticeable performance differences will emerge. All processes involving randomness are repeated with the seed-set feature of OMNeT++; thus, scientific reproducibility is ensured. Detailed test scenarios and parameter values will be explained in Section 4.4. Table 4.1 summarizes the key simulation parameters

Table 4.1: Simulation parameters.

Parameter	**Value**
Backbone link capacity	10 Mbps
End connection capacity	100 Mbps
Queue capacity	150 packets
Traffic Types	
Transport protocols	TCP, UDP
TCP traffic	Web (HTTP-like), File transfer
UDP traffic	VoIP, Video streaming
ECN support	Enabled (TCP only)
Traffic Patterns	
Traffic density	Medium, High
Traffic scenarios	Uniform, Dynamic (with sudden changes)
Reproducibility	
Random seed	Set via OMNeT++ seed-set feature

4.2. Details the Proposed CoAL-AQM Algorithm

This section summarizes the basic principles that distinguish the developed CoAL-AQM algorithm from classical threshold-based AQM algorithms. The aim is to present the mathematical skeleton and design logic of the module without going into the code details. Thus, it becomes clear under which abstract assumptions the experimental results to be examined in the following sections are produced.

4.2.1. Dynamic Threshold Value Calculation

In the developed CoAL-AQM algorithm, there is one threshold value and it is calculated dynamically. Floyd et al. proposed automatic parameter adjustment depending on the link bandwidth in their ARED studies [5]. The algorithm calculates the minimum threshold using the output speed (C) to which the queue is connected and the average packet size (L) in the system. The formula is applied as:

$$\min_{th} = \max(5, C \times \text{target_delay}/L). \tag{4.1}$$

This formula is used for the threshold calculation also in proposed CoAL-AQM.

Here C is the connection capacity (packets/second), target_delay is the maximum targeted queue delay (seconds), and L is the average packet size (bytes). The target delay value is determined by the user. In CoAL-AQM, the default value is 0.02 seconds (20 ms) when the expected delay is not given. This value was chosen to meet the delay requirements of modern Internet applications and is consistent with other AQM algorithms such as PIE (Proportional Integral controller Enhanced) [7].

The average packet size is updated every time a new packet arrives in the system. Since the average packet size changes, the minth value is also dynamically recalculated, thus obtaining a threshold that adapts to changing traffic conditions during traffic flow. The average packet size is reset every 100 seconds to prevent the packet average value from being affected by very old traffic and becoming outdated.

4.2.2. State Vector s_t

Each packet arrival resamples the queue in a Markov Decision Process manner. CoAL-AQM uses a four-element state vector

$$s_t = [q_t^{\text{inst}}, q_t^{\text{avg}}, \text{ratioBit}_t, \text{protoFlag}_t]. \tag{4.2}$$

In the Equation 4.2, q_t^{inst} is the ratio of the current queue length to the threshold. The queue length threshold ratio at the time the packet arrives is mapped to discrete values from 0 to 9 according to the following boundaries {0.5, 1.0, 1.4, 1.8, 2.2, 2.6, 3.5, 4.0}

These bounds are designed according to the RED algorithm bounds, which operate generally between $\min_{th}$ and $3 \times \min_{th}$. The bounds represent the ratio of the queue length to the threshold. The choice provides finer granularity in queue management between the critical operational range of 0.5 and 2.6, while also working for extreme conditions such as 3 and 4 times the threshold.

q_t^{avg} is the ratio of the average queue length to the threshold. The average queue length shown in equation

$$Qlength_{avg} = (1 - \alpha_{\text{EWMA}}) \cdot Qlength_{avg} + \alpha_{\text{EWMA}} \cdot Qlength_{current} \tag{4.3}$$

is updated with the Exponentially Weighted Moving Average (EWMA) method each time a packet arrives. This method provides an average that is more robust to sudden changes by limiting the effect of new incoming values. The ratio of the average queue length to the threshold is similarly mapped to discrete values from 0 to 9.

we scanned of 0.01 to 0.10 for the alpha EWMA value and evaluated performance metrics. In the range of 0.01, the response was slow and the amount of delay increased, while unnecessary drops increased towards the values of 0.10. The alpha average value was determined as 0.02 for the highest stability.

Another state component ratioBit, is 1 if the dropped packet ratio for the current packet's flow is higher than the average dropped packet ratio for all flows, otherwise

it is 0. Thus, the agent detects the high loss flow and can apply additional reward or penalty to this source in the reward function. With this approach, the agent is aware of flows and selects actions accordingly.

Another state component protoFlag, indicates the protocol marking of the packet. For UDP, protoflag is 0, for TCP it is 1. The response of UDP and traffic to congestion signals is fundamentally different. UDP is insensitive to loss, while TCP reduces cwnd as a result of packet loss or marking. Since TCP has a congestion control mechanism and UDP does not, UDP traffic can act aggressively and suppress TCP traffic when no distinction is made. With this protocol distinction, different protocol-specific optimizations can be learned. Queue management is performed according to the needs of different protocols.

Table 4.2 summarizes the state vector components and their possible values.

Table 4.2: CoAL-AQM state vector components.

State Component	**Possible Values**
q_t^{inst} (Instantaneous queue ratio)	0, 1, 2, ..., 9
q_t^{avg} (Average queue ratio)	0, 1, 2, ..., 9
ratioBit (Flow drop rate indicator)	0, 1
protoFlag (Protocol type)	0 (UDP), 1 (TCP)
Total state space size	$\mathbf{10 \times 10 \times 2 \times 2 = 400}$

4.2.3. Action Space A and Action Value a_t

In the CoAL-AQM algorithm, the action space consists of 11 values $A = \{0.0, 0.1, \ldots, 1.0\}$. The action value a_t is the probability of passing the value and $a_t \in A$. The agent chooses the a_t value corresponding to the highest reward value of the state in the Q-table. The remaining $1 - a_t$ part is shared between mark and drop

with a parameterized function depending on the current queue occupancy and trend. Thus, instead of the classical three separate options such as pass, mark, drop, a one-dimensional action space with 11 probabilities of probabilistic actions are used. Instead of taking the full drop action as soon as the threshold value is passed, the agent can gradually change the probability of passing. This approach transforms the sharp decision boundaries of classical threshold-based AQMs into a continuous decision space. This provides the advantages of both better learning, smoother Q-function and more stable behavior. Moreover, probabilistic actions may handle noise and delay variance better than deterministic ones, by avoiding sensitive thresholds.

To explain the action selection, the action value selected from the Q-table, a_t, is the pass probability, the remaining $1 - a_t$ is the probability of the packet being marked and dropped. The equations for probability of pass and probability of leftover are

$$\begin{aligned} P_{pass} &= a_t, \\ P_{leftover} &= 1.0 - P_{pass}. \end{aligned} \tag{4.4}$$

Before the action selection, the ratios of both instantaneous and average queue lengths to three times the threshold are calculated. Let Q_t^{inst} denote the instantaneous queue length and Q_t^{avg} denote the average queue length. These ratios are capped at 1.0 when the queue length exceeds $3 \times \min_{th}$. Specifically:

$$r_t^{\text{inst}} = \min\left(1.0, \frac{Q_t^{\text{inst}}}{3 \times \text{threshold}}\right), \tag{4.5}$$

$$r_t^{\text{avg}} = \min\left(1.0, \frac{Q_t^{\text{avg}}}{3 \times \text{threshold}}\right). \tag{4.6}$$

A ratio value of 1.0 indicates that the queue has reached or exceeded three times the threshold, signifying a severely congested state. The queue trend is then calculated as

$$\text{trend} = r_t^{\text{inst}} - r_t^{\text{avg}}. \tag{4.7}$$

This trend value provides information about the queue dynamics: positive values indicate queue growth, while negative values indicate queue reduction.

The base value of the probability of a packet being dropped is calculated as shown in equation

$$P_{drop}^{base} = 0.01 + 0.24 \cdot \left(r_t^{inst}\right)^2 . \tag{4.8}$$

The drop rate should increase as the queue fills. Instead of the linear climbing method in the RED algorithm, we used a smoother and convex function that has potential to better suppress the bufferbloat problem and obtained a basic drop curve with a lower bound of 1 % and an upper bound of 25 %.

the value of 0.01 is created based on the maxp value suggested in the RED algorithm article to protect against self-clocking mechanism. It is used in packet mark decision and is important for managing the congestion window size. The quadratic term here opens a corridor at low occupancy; at high occupancy, it increases faster than RED's linear ramp and suppresses bufferbloat. The value of 0.24 is chosen to target a P_{drop} of 0.25 when the queue is full. The quadratic term is chosen due to the soft increase in linear increases until it reaches 0.5 occupancy and then the hard increase.

In addition to the base drop probability, the calculation in equation

$$P_{drop}^{trend} = \max(0, 0.10 \times \text{trend}) \tag{4.9}$$

is used for the drop probability depending on the trend value. For example, a coefficient of 0.10 brings an additional drop of 2 percent in the case of a trend value of 0.2 (a sudden increase of 30 packets) in a queue with a capacity of 150. This drop probability also includes the effect of the trend in the calculation. Negative trends that will occur as a result of the queue starting to empty are not accepted and extra packets are not dropped. Extra drop is not triggered when the queue size starts to decrease.

In addition, when r_t^{inst} is above 0.75, p critical drop in equation

$$P_{drop}^{critical} = \begin{cases} 0.2\left[4\left(r_t^{inst} - 0.75\right)\right]^2 & \text{if } r_t^{inst} > 0.75 \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

is added to P_drop for critical levels. When 0.75 is subtracted from r_t^{inst}, a number between 0-0.25 is obtained. This value is multiplied by 5 and pulled to the range of 0-1. Thus, when this value is squared, its value is again less than or equal to 1. This reinforcement value is multiplied by 0.2, and in fact, it is limited to a ceiling value of 0.2. This value was selected by hyper parameter search considering the balance of the drop rate and average latency performance in different tests. When this value decreases, an increase in latency and a decrease in drops were observed in our experimental scenarios. When it increases, the opposite behavior was observed.

The drop ratio is calculated by summing three components:

$$P_{drop_ratio} = P_{drop}^{base} + P_{drop}^{trend} + P_{drop}^{critical}. \tag{4.11}$$

This value is found by adding the P_{drop}^{base} (4.8), the P_{drop}^{trend} (4.9) and $P_{drop}^{critical}$ (4.10).

To prevent the problem of TCP windows closing frequently, the upper limit of the P_{drop_ratio} is set to 50 %. For the 0.01 minimum value warning in the P_{drop}^{base} section, the lower limit of the P_drop is set to 1 %.

During the experiments, it was observed that especially in long-running ECN supported TCP flows, instead of dropping the packet, the sender was notified with a mark with ECE bit, thus the cwnd window was reduced in accordance with the sender and much fewer packets were dropped. Therefore, for ECN supported packets, the following formula was used:

$$\begin{aligned} P_{\text{drop}} &= P_{\text{leftover}} \times P_{\text{drop_ratio}} \times ecnDropScale \\ P_{\text{mark}} &= P_{\text{leftover}} - P_{\text{drop}} \end{aligned} \tag{4.12}$$

where the ECN drop scale factor is calculated as

$$ecnDropScale = (r_t^{\text{inst}})^2 \tag{4.13}$$

The logic behind this approach is that since the ECN mechanism already provides effective feedback for congestion control, the need for packet drops is reduced. The ecnDropScale factor is calculated by squaring r_t^{inst}. For example, when r_t^{inst} is 0.5,

ecnDropScale becomes 0.25, reducing the drop probability to one fourth. When r_t^{inst} is 0.8, *ecnDropScale* becomes 0.64. This quadratic scaling ensures that ECN signaling is prioritized at low and medium r_t^{inst} levels, while still dropping some packets at high levels.

For packets that are not ECN supported (UDP or ECN-incapable TCP), the drop probability is calculated as:

$$P_{drop} = P_{leftover} \times P_{drop_ratio}. \tag{4.14}$$

Since marking cannot be done in this case, congestion control is provided entirely through packet dropping. The remaining probability

$$P_{mark} = P_{leftover} \times (1.0 - P_{drop_ratio}) \tag{4.15}$$

is ineffective for these packets and is added to the pass probability in practice.

With this adaptive approach, flows with ECN support experience less packet loss, while flows without ECN support are subject to traditional drop-based congestion control, thus achieving optimized behavior for both protocol types.

Then, a random number between 0 and 1 is selected and the action to be taken for the package is selected according to the threshold values in

$$\text{Decision} = \begin{cases} \text{PASS} & \text{if } u < P_{pass} \\ \text{MARK} & \text{else if } u < P_{pass} + P_{mark} \\ \text{DROP} & \text{else} \end{cases} \tag{4.16}$$

. Although this method is deterministic, it provides a trend-sensitive action selection.

4.2.4. Reward r_t

In our RL-based AQM, multiple goals are considered for optimization: reducing latency, reducing unnecessary packet loss, preserving throughput, paying attention to

balanced drop probability among different flows, and managing packets according to packet type. Therefore, a multi-term reward function equation

$$r_t = f_1(q_t^{inst}) + f_2(\text{drop/mark}) + f_3(\text{ratioBit}) \tag{4.17}$$

is designed for CoAL-AQM.

4.2.4.1. Queue Length Cost $f_1(q_t^{inst})$. The $f_1()$ function maps the ratio of the queue length to the threshold from 10 different regions to the relevant intervals. The queue is encouraged to be at occupancy levels between threshold and 3 × threshold, and reward values are given accordingly. Since leaving the queue empty reduces latency but decreases throughput, negative rewards are given even to some regions where the queue is empty. Fine tuning is done for reward/punishment values in regions.

Table 4.3: Reward/penalty values according to q_t^{inst}.

q_t^{inst}	$[0, 0.5)$	$[0.5, 1)$	$[1, 1.4)$	$[1.4, 1.8)$	$[1.8, 2.2)$	$[2.2, 2.6)$	$[2.6, 3)$	$[3, 3.5)$	$[3.5, 4)$	$[4, \infty)$
Reward	-2	-1	$+1$	$+3$	$+5$	$+3$	$+1$	-3	-5	-7

4.2.4.2. Drop/Pass Reward Cost f_2(drop/mark). A dynamic penalty system is implemented for dropped packets. The basic drop penalty is determined by the queue status. When the queue length is below threshold, dropPenalty is 5, when the queue length is above 3 × threshold, drop penalty is 1, and in other cases, drop penalty is 3. In this approach, packet drops when the queue is almost empty are penalized with the highest penalty of 5 points, and when the queue approaches the critical occupancy level, a lower penalty of 1 point is imposed. The logic behind this is that packet drops are inevitable and necessary at high queue occupancy levels.

Here, a distinction is also made according to the protocol type:

- reward −= dropPenalty (TCP packets are dropped with a full penalty)
- reward −= $0.20 \times dropPenalty$ (UDP packets are dropped with a penalty reduced to 20%).

The reason for this distinction is that TCP is more sensitive to packet loss and the throughput drops dramatically with the congestion window mechanism. UDP traffic is more tolerant to packet loss and, since it is usually used in real-time applications, a moderate amount of packet dropping is acceptable to avoid excessive queueing [31].

Additionally, a 2 point bonus is given when no packets are dropped and the queue length is below 3 × threshold. This bonus encourages the agent to pass packets at low queue levels and prevents unnecessary packet drops. By updating the values in the Q-table according to this reward system, the agent learns the optimal drop/pass decisions in different queue states.

4.2.4.3. ratioBit Penalty f_3(ratioBit). To increase drop awareness among flows, ratioBit mechanism is used. If a flow's packet drop rate is higher than the system-wide average drop rate (ratioBit is 1) and packets are passed, it is rewarded with +4 reward. If a flow's packet drop rate is higher than the system-wide average drop rate (ratioBit is 1) and packets are dropped, it is penalized with -4 reward. This encourages the agenr to be more tolerent to the flows with drop rates higher than the average drop rate to compansate the inbalance. On the other hand, if a packet from a flow with an already high drop rate is dropped again, it is penalized with -4 points. This mechanism prevents the system from continuously penalizing certain flows and ensures that drop rates are balanced among all flows.

The thresholds were determined by experiments and literature review. This three-component design positions itself as a multi-objective controller for CoAL-AQM, which can simultaneously optimize delay–drop rate–throuhput.

In the CoAL-AQM algorithm, the reward calculation is designed to take into account the stochasticity resulting from the continuity of the action space. Since the selected action value is a probability value, instead of using only the reward of the actual outcome, the expected value approach is applied. First, the reward of the current decision (drop/mark/pass) is calculated (reward_1). Then, the reward of the

alternative decision is also calculated (reward_2), and the weighted average is taken with the formula final reward $= a_t \times \text{reward}_1 + (1 - a_t) \times \text{reward}_2$. For example, when $a_t = 0.7$ is selected and the PASS decision is made, the reward calculation includes the PASS reward with a weight of 70% and the reward of the alternative decision (mark or drop) with a weight of 30%.

In the classical approach, when only the reward of the performed action is used, there may be updates with high variance for the same state and action value. Thanks to the expected value calculation, the nature of the stochastic policy is properly reflected in the Q-learning process. This mechanism allows the Q-values to be more stable and the system to adapt more robustly to variable traffic conditions.

4.2.5. Q-table

Q-table is a two dimensional array $Q(s, a)$ that stores the expected reward for each state-action pair. The state (s) in Q-table consists of four independent components. The instantaneous queue length/threshold (q_t^{inst}) ratio consists of the numbers 0–9 and can take 10 values. The average queue length/threshold (q_t^{avg}) ratio consists of the numbers 0–9 and can take 10 values. ratioBit can take 2 values, 0 or 1. ProtoBit can take 2 values, 0 or 1. This definition corresponds to $10 \times 10 \times 2 \times 2 = 400$ discrete states for the state part of Q-table. Action (a) has 11 possible scalar values $\{0.0, 0.1, \ldots, 1.0\}$, theredore Q-table contains a total of 4400 cells. Since each cell is stored as a double (8B), the memory requirement is 35.2 kB. This is much less than the DRAM capacity of most routers in use. The indexing scheme is calculated as

$$s = q_t^{\text{inst}} + (q_t^{\text{avg}} \times 10) + (\text{ratioBit} \times 10 \times 10) + (\text{protoBit} \times 10 \times 10 \times 2) \tag{4.18}$$

so that the table can be kept as a one-dimensional vector.

In the configuration, all cells are initialized with random values between -0.01 and 0.01, which accelerates the initial RL discovery phase. After each packet pass, mark or drop decision, the corresponding cell in the table (s_t, a_t) is updated with the

formula based on the Bellman equation (3.7). However, since the protocol type of the incoming packet is unknown and the source from which the incoming packet will come is unknown, the next state is practically unknown. There are four different state values for the possibility that the protocol type is TCP or UDP and the *ratioBit* value is 0 or 1. The maximum value in the table row is selected among these four state values. A *weightedNextQ* value is obtained by dividing the sum of the values received as a result of four different state values by four. This value is used instead of the next state Q value. The relevant cell $Q(s_t, a_t)$ in the Q-table is updated with Equation (3.7).

During the development phase, statistics regarding the table changes were collected at 5-second intervals and the changes in alpha and epsilon values throughout the simulation were examined. By analyzing these statistics, inferences were made for the alpha learning rate and epsilon value.

Normally, two different scenarios are made in Qlearning applications as training and testing. Q-table filled in the training phase ideally produces the ideal result in the test scenario. Since internet traffic is variable and to minimize the need for different parameter settings for different application points of the algorithm, we decided to apply the alpha learning rate value calculated according to time as a piecewise function. Decreasing the learning coefficient of the algorithm over time, result in stopping to learn as time progresses for changing traffic. Reducing the learning rate with certain cycles was a more suitable method for variable internet traffic. The learning rate peak was tried between 0.1 and 0.3. When the value of 0.3 was selected, lower queueing time, lower throuhput and similar drop probability were obtained. In order for the throuhput to reach a higher value and to provide an advantage over RED, the learning rate peak was selected as 0.1. In this way, we implemented an RL algorithm that continuously trains itself according to variable traffic. For 1000-second simulations, a learning rate function that decreases linearly from 0.1 to 0.05 with 250-second intervals was used. The 250-second interval is an interval selected depending on the internet traffic change, but since its effect is a parameter that varies between 0.1 and 0.05, its

effect is less felt. Lower intervals can also be selected for places where traffic density changes are greater.

Table 4.4: CoAL-AQM performance results with different alpha values.

Algorithm	**Queueing Time (ms)**	**Throughput (Mbps)**	**PLR (%)**
CoAL-AQM $\alpha = 0.1$	7.380	9.600	0.112
CoAL-AQM $\alpha = 0.2$	7.379	9.590	0.095
CoAL-AQM $\alpha = 0.3$	7.122	9.587	0.092

From the moment the decision to be made about the incoming packet is known, the nextState value, which is the state in which this packet is added to the queue, is calculated. In this case, the instantaneous and average queue occupancy rate is known for the next state. However, since the protocol type of the incoming packet is not known for the next state, the protoBit to be determined accordingly is not known. Since it is not known from which source the incoming packet will come, the ratioBit value is not known. There are 4 different state values for the probability that the ratioBit value is 0 or 1, and the protoBit value is 0 or 1. The maximum value in the table row is selected among these 4 state values. A weightedNextQ value is obtained by dividing the sum of the incoming values by 4 as a result of the 4 different state values. This value is used when updating the Q-table.

4.2.6. Exploration-Exploitation Strategy

ϵ_{max} was tested in the range of 0.1-0.3. Similarly, a function that exhibits a linear decrease in time between 0.1 and 0.05 with 250-second intervals was used for the exploration-exploitation rate (epsilon). The risk of the policy locking into a local optimum was reduced in long simulations. With the 0.10 epsilon peak value, the values in the action space are tested, and the policy to be applied according to the Qtable values is also managed in a balanced manner. In trials where the epsilon value is

selected higher, the application of the policy learned with Qlearning decreases and the policy effect obtained in throuhput, queueing delay, and drop rate leaves itself to more randomness.

4.2.7. Gamma Factor

The gamma discount factor in which formula Equation (3.7) is $\gamma \in (0, 1)$. As the gamma value increases, the agent carries more future rewards to the present. Since the reward function in queue management includes not only instantaneous values but also factors such as drop rates between flows, it is not suitable for instantaneous evaluation. Different gamma values were tried in simulation repetitions. As the gamma value increased, a decrease in the average delay value – loss of throughput or the opposite situation, an increase in the average delay value – increase in throughput was observed. $\gamma = 0.9$ was chosen for the different traffic scenarios tried.

In this framework, a self-adaptive AQM that does not require parameter adjustments related to queue parameters has been constructed. The parameters that can be changed in RL-based AQM are the parameters that affect the sensitivity to traffic changes. In this way, the algorithm has a high-performance structure that is sensitive to the drop rate and traffic type between flows.

4.3. Implementation Steps

This section summarizes how the CoAL-AQM algorithm works in OMNeT++ through a step-by-step process flow. Section 4.2 explains in which order the theoretical framework presented is applied and which measurements are made.

4.3.1. Packet Arrival and State

While the simulation is running, each new packet comes to the `matchesPacket()` method. First, the instantaneous packet count is determined depending on the

`queueLength` variable. Epsilon and alpha values are updated depending on the simulation time. Every 250 seconds, the ϵ and α coefficients decrease linearly from 0.10 to 0.05; then increase again. The loop creates a periodic discovery window compatible with the traffic flow life. The threshold value is updated depending on the size of the new incoming packet. The `ratioBit` is determined by comparing the drop rate of the source with the general drop rate of the network; if there is a risky situation specific to the flow in terms of drop decisions between flows, the `ratioBit` is set to 1. The IPv4 headers of the packet are examined and protoBit = 0 is assigned if it is TCP, and protoBit = 1 if it is UDP. The current average queue occupancy value is also 0 at the beginning, and it is a known value because it is updated after the action to be taken regarding the packet is determined after each packet arrives. The values of q_t^{inst} (current queue length/threshold), q_t^{avg} (average queue length/threshold), `ratioBit` and `protoBit` are converted to a unique state index with the equation (4.18).

4.3.2. Selecting the State Corresponding Action

Then, the best action value in the table or a random action with an epsilon percent probability is returned with the Epsilon-Greedy method. The best action value is the action value in the column with the highest value in the row of the relevant status value in the table where action values are $\{0.0, 0.1, \ldots, 1.0\}$.

4.3.3. Making Decision for Package Based on Action Value

In the next stage, the action decision will be determined for the selected action value a_t. The trend value is determined according to the difference between r_t^{inst} (instant queue length/3 threshold) and r_t^{avg} (average queue length/3 threshold). The highest value that r_t^{inst} and r_t^{avg} values can take is 1. The action value is given to the P_{pass} variable for the package.

The drop ratio is calculated using the following formula:

$$P_{drop_ratio} = \left[0.01 + 0.24(r_t^{\text{inst}})^2 + 0.10 \max(0, r_t^{\text{inst}} - r_t^{\text{avg}})\right]. \tag{4.19}$$

If the r_t^{inst} ratio is above 0.75, the P_{drop_ratio} value is supplemented. The square of 4 times the difference from the instantaneous queue occupancy rate of 0.75 is multiplied by 0.2 and added to the

$$P_{drop_ratio} = P_{drop_ratio} + 0.2 \times (4 \times (r_t^{\text{inst}} - 0.75))^2 \tag{4.20}$$

rate as a supplement.

The calculated P_{drop_ratio} value is limited in the range $[0.01, 0.5]$. For ECN supported packets, the formulas $P_{\text{drop}} = P_{\text{leftover}} \times P_{drop_ratio} \times \text{ecnDropScale}$ and $P_{\text{mark}} = P_{\text{leftover}} - P_{\text{drop}}$ are used. Here $P_{\text{leftover}} = (1 - P_{\text{pass}})$ and $\text{ecnDropScale} = (r_t^{\text{inst}})^2$ is calculated. For packets without ECN support, the formulas $P_{\text{drop}} = P_{\text{leftover}} \times P_{drop_ratio}$ and $P_{\text{mark}} = P_{\text{leftover}} \times (1.0 - P_{drop_ratio})$ are applied. In ECN supported packets, the drop probability is reduced in proportion to the square of the queue occupancy level thanks to the `ecnDropScale` factor. Thus, marking is prioritized in ECN supported flows, while only the dropping mechanism is used in flows without ECN support.

Then a random value $r \in [0, 1]$ is selected. If $r < P_{\text{pass}}$, a pass decision is made, if $P_{\text{pass}} \leq r < (P_{\text{pass}} + P_{\text{mark}})$, a mark decision is made, and if $r \geq (P_{\text{pass}} + P_{\text{mark}})$, a drop decision is made.

4.3.4. Reward Calculation

After the decision is made for the package, a reward1 value is calculated based on the reward function

$$reward = (a_t \times reward1) + ((1 - a_t) \times reward2) \tag{4.21}$$

as mentioned in equation (4.17) equation in subsection 4.2.4. The reward2 value is calculated according to the opposite situation of whether the situation in the selected decision drops the packet or not. The reward value is obtained by multiplying the selected action value with the reward value, $1 - a_t$ with the reward2 value.

4.3.5. Updating Q-table

There are 4 different state values for the ratioBit value to be 0 or 1 for the next state and the protoBit value to be 0 or 1. The maximum value in the table row is selected from these 4 state values. A weightedNextQ value is obtained by dividing the sum of the values resulting from the 4 different state values by 4. The qtable is updated according to the formula below. The calculated weightedNextQ value is used for the Q value to be found for the next state in the formula

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)] \tag{4.22}$$

where S is current state, A is chosen action, R is reward, Q(S',A') is the value calculated and averaged for possible next states in our thesis (weightedNextQ), γ is discount factor, and α is learning rate.

Finally, the average queue length is updated using the Exponential Weighted Moving Average (EWMA) method with a weight factor of $\alpha_{\text{EWMA}} = 0.02$:

$$q_{avg} = (1 - \alpha_{\text{EWMA}}) \cdot q_{avg} + \alpha_{\text{EWMA}} \cdot q_{inst} \tag{4.23}$$

where q_{inst} is the current instantaneous queue length and q_{EWMA} is the updated average queue length.

4.3.6. Output Signals and Statistics Collection

Statistics such as Queueing Time, Avg Q Len, Throughput are collected from the queue by default in the INET Framework. Flow-based incoming and dropped packet numbers are also kept in two separate arrays of type long. When a packet arrives, an index corresponding to the sender's IP address is determined in the array, and the number is increased by one for the relevant index. After the decision is determined for the incoming packet, if a drop decision is made, the value in the relevant index in the array holding the drop values is increased. Statistics for incoming packets and dropped packets are kept in arrays. In this way, statistics such as the number of incoming packets, the number of dropped packets, and the drop rate per sender are obtained.

This sequential flow of *Package* → *State* → *Action* → *Reward* → *Table update* represents the classical Q-learning cycle, which is compatible with the discrete-event architecture of OMNeT++.

4.4. Test Scenarios

In order to evaluate the performance of the proposed algorithm in the thesis, two different test scenarios with fixed and variable density representing network traffic were designed. While the scenarios used the same network topology, only the traffic type and density were changed. In this way, the CoAL-AQM algorithm was compared with other algorithms under the same conditions.

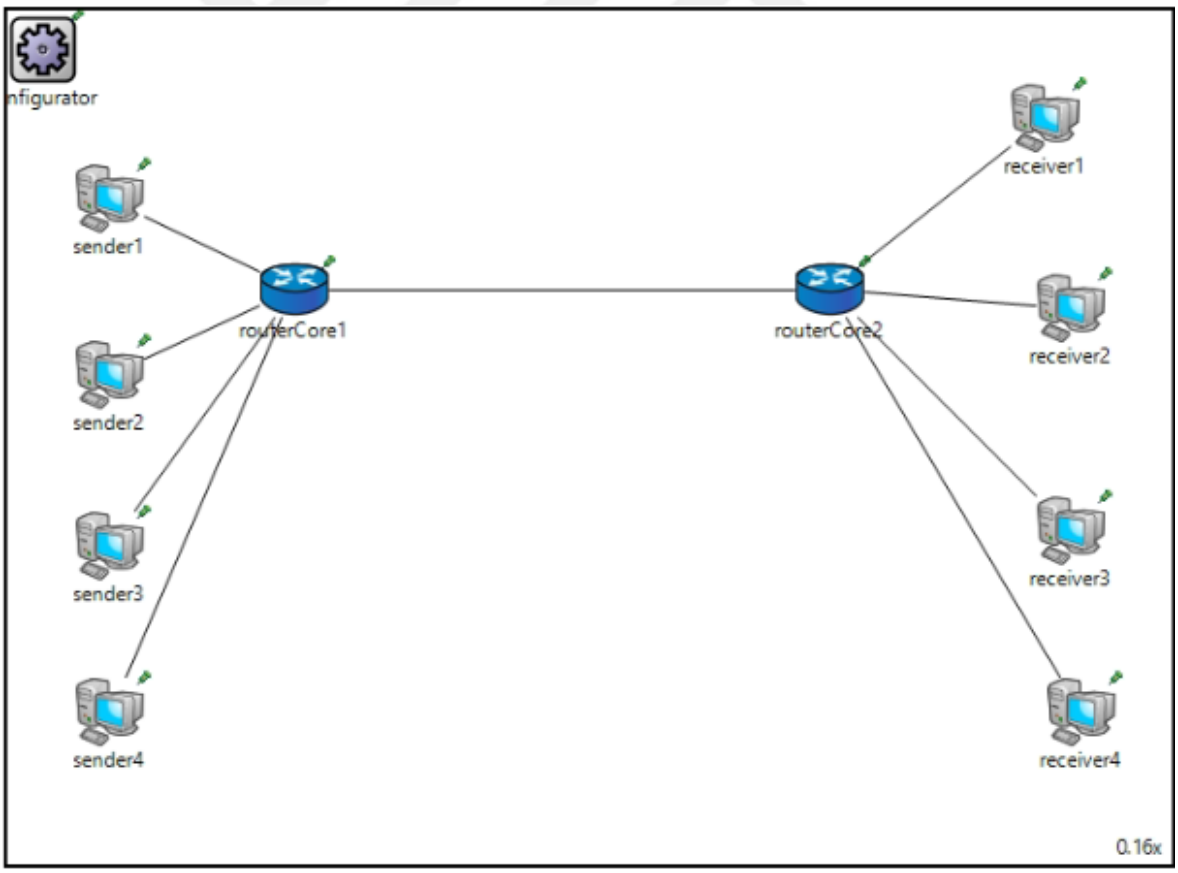


Figure 4.2: Topology diagram (created using OMNeT++ simulation environment under academic public license for research purposes).

In the tests, TCP Reno algorithm was used for TCP protocols, advertised window was 65535 bytes (64 KB), ECN was active, maximum segment size (MSS) was set to 1460 bytes.

4.4.1. Constant Density Traffic Scenario

In the first test scenario, two TCP and two UDP traffic generator sources are used to provide traffic diversity in modern networks. One TCP stream represents web traffic (HTTP), the other represents file transfer (FTP). The UDP streams represent voice (VoIP) and video streaming applications. This scenario is tested for two different traffic densities.

4.4.1.1. High Traffic. Traffic parameters have been increased to increase network traffic density. In web traffic, response sizes have been increased to 50KB-10MB, and the waiting time between data uploads has been increased to exponential (1s). Traffic has been generated such that the number of requests per session is between 50-150 and the waiting time between sessions is exponential (5s). For file transfer, 1GB data transfer has been sent at uniform (0.1s, 0.5s) intervals. VoIP traffic has been sent with 320-byte packets at 20ms intervals to represent multiple calls. For video streaming traffic, 1400-byte packets have been sent at 2.5ms intervals. Burst times have been set to exponential (2s) and gap times have been set to exponential (0.05s).

For this traffic situation, different $\min_{th}$-$\max_{th}$ parameters were tried for RED and ARED and the results were recorded. For busy lines, Drop Probability around 1% and average delay time less than 20 ms are reasonable values for a workable system in reality. Accordingly, $\min_{th} = 25$, $\max_{th} = 75$ were selected for RED, $\min_{th} = 20$, $\max_{th} = 60$ for ARED. Other parameters were kept the same as in the paper ($w_q = 0.002$, $\max_p = 0.02$, $\alpha_{\text{ARED}} = 0.01$, $\beta_{\text{ARED}} = 0.9$, aredUpdateInterval= 0.5) as stated in the previous sections.

4.4.1.2. Medium Traffic. For the simulation of web traffic, the request size was set to be uniformly distributed between 100-500 bytes, and the response size was set to vary between 20KB-6MB. In order to be similar to reality, the traffic was created with an exponential (3.5s) distribution for waiting times between data uploads, a request count of 20-70 per session, and an exponential (20s) idle time between sessions. For file transfer, 350MB data transfer was sent with uniform (0.2s, 0.5s) intervals. For VoIP traffic, 200-byte packets were sent with 20ms intervals. For video streaming traffic, 1400-byte packets were generated with 5ms intervals, exponential (1.2s) burst times, and exponential (0.12s) gap times.

For this traffic situation, different $\min_{th}$-$\max_{th}$ parameters were tried for RED and ARED and the results were recorded. For busy lines, Drop Probability around 1% and average delay time less than 20 ms are reasonable values for a workable system in reality. Accordingly, $\min_{th} = 5$, $\max_{th} = 15$ were selected for RED, $\min_{th} = 5$, $\max_{th} = 15$ for ARED. Other parameters were kept the same as in the paper ($w_q = 0.002$, $\max_p = 0.02$, $\alpha_{\text{ARED}} = 0.01$, $\beta_{\text{ARED}} = 0.9$, aredUpdateInterval= 0.5) as stated in the previous sections.

4.4.2. Dynamic Traffic Scenario

The second test scenario is designed to measure the adaptation ability of AQM algorithms to time-varying traffic density. Traffic load is changed to high, medium, high and medium traffic at 250-second intervals. During the 1000-second simulation period, traffic density changes in four different phases:

Phase 1 (0-250s): High traffic is modeled with 50KB-10MB response sizes, exponential(1s) think time and 50-150 requests per session for web traffic. FTP transfer is sent with 250MB data at uniform(0.1s, 0.5s) intervals, VoIP single call (160B/20ms), and video streaming is sent with 1400 byte packets at 10ms intervals. Burst times are set as exponential(1s) and gap times are set as exponential(0.5s).

Phase 2 (250-500s): Medium traffic is modeled with 50KB-10MB response sizes, exponential (1s) think time and 50-150 requests per session for web traffic. FTP transfer is sent with 250MB data at uniform (0.2s, 0.5s) intervals, VoIP single call (200B/20ms), and video streaming is sent with 1400 byte packets at 5ms intervals. Burst times are set to exponential (2s) and gap times are set to exponential (0.05s).

Phase 3 (500-750s): High traffic is modeled with 50KB-10MB response sizes, exponential (3.5s) think time and 50-150 requests per session for web traffic. FTP transfer is sent with 250MB data at uniform (0.1s, 0.5s) intervals, VoIP single call (320B/20ms), and video streaming is sent with 1400 byte packets at 0.8 ms intervals. Burst times are set to exponential (2s) and gap times are set to exponential (0.05s).

Phase 4 (750-1000s): Medium traffic is modeled with 100B-500B response sizes, exponential (3.5s) think time and 50-150 requests per session for web traffic. FTP transfer is sent with 1000MB data at uniform (0.2s, 0.5s) intervals, VoIP single call (200B/20ms), and video streaming is sent with 1400-byte packets at 5ms intervals. Burst times are set to exponential (1.2s) and gap times are set to exponential (0.12s).

For this traffic situation, different $\min_{th}$-$\max_{th}$ parameters were tried for RED and ARED and the results were recorded. For busy lines, Drop Probability around 1% and average delay time less than 20 ms are reasonable values for a workable system in reality. Accordingly, $\min_{th} = 30$, $\max_{th} = 90$ were selected for RED, $\min_{th} = 30$, $\max_{th} = 90$ for ARED. Other parameters were kept the same as in the paper ($w_q = 0.002$, $\max_p = 0.02$, $\alpha_{\text{ARED}} = 0.01$, $\beta_{\text{ARED}} = 0.9$, aredUpdateInterval= 0.5) as stated in the previous sections.

This dynamic scenario is important to measure the response of the algorithms to the bufferbloat problem, which occurs especially in sudden traffic changes. It aims to demonstrate the superiority of the adaptive learning ability of CoAL-AQM over classical AQM algorithms by creating traffic similar to the usage in real networks.

5. EXPERIMENTAL STUDIES AND RESULTS

5.1. Comparison Criteria and Measurement Methodology

Evaluating the performance of a new AQM technique is not limited to answering a one-dimensional question such as how much the throughput has increased. Queue is actually a multi-dimensional optimization problem in the axes of delay-loss-throughput [32].Therefore, four different metrics are selected while evaluating this multi-dimensional optimization problem in our experimental studies.

5.1.1. System Parameters

5.1.1.1. Queue Delay. Many of the applications that users use are interactive and are directly affected by RTT. RFC 7567, AQM design goals include the principle of keeping latency low [33]. We report the latency as the Queueing Time vector of INET. This metric objectively shows how successful CoAL-AQM's queue management strategy is.

5.1.1.2. Packet Loss Rate. Excessive packet dropping triggers TCP's slow-start/fast-retransmit mechanisms, which reduces throughput. In addition, insufficient packet dropping level increases the bufferbloat problem. Therefore, the Drop Probability value for all packets arriving at the queue is also used in the comparison.

5.1.1.3. Throughput. Throughput is a performance metric that expresses the amount of data that a network system can successfully transmit per unit of time. In queuing theory, it is a measure used to evaluate the performance of different algorithms. In this thesis, it is measured in bps (bits/sec) and reflects the data carrying capacity of the system.

5.1.1.4. Sender Based Loss Rate. One of the debated issues in AQM literature is the effect of high volume flows on other flows. Here, we will use the drop rate between flows as an evaluation criterion by determining if the loss percentage of a flow is above the overall loss percentage.

In some experimental scenarios, some senders generated UDP traffic and some TCP traffic. UDP is a protocol without loss feedback, excessive UDP traffic may negatively affect TCP traffic, while UDP-based applications can operate with few packet. We integrated protocol-based discrimination into the reward function in CoAL-AQM to differentiate and utile protocol based gains. Therefore, in the experiments, we will evaluate the question of whether the different penalty multiplier applied to UDP is active or not with sender-based drop rate.

5.1.2. Measurement Methodology

Each scenario is run with three different random seeds. The average of the outputs of the three seed values is used in the thesis. In the Performance and Results graphs, the error bars are calculated using a 2 sigma neighborhood, which represents a 90% confidence interval. As can be seen in the Results graphs, we did not repeat the tests for more seeds because the error bars were small enough. The deviations of the outputs from the seed mean were found and divided by the seed mean. In this way, we looked at the relative deviation rate of each output. Throuhput, queueing delay these values are below 1 percent. The relative deviation for PLR varies between 8% and 18% in the CoAL-AQM algorithm. However, the fact that the packet drop rates are 1 per thousand or lower in the CoAL-AQM algorithm indicates that the high relative deviation results are fluctation noise.

In addition, depending on the delay time requirement in the router, the CoAL-AQM algorithm can get the desired delay time from outside. If this value is not given, the default value of 20 ms is applied. The algorithm does not guarantee an absolute delay time with this parameter, it is a parameter used when determining the threshold.

Table 5.1: Network and algorithm parameters.

Parameter	**Value**
Network Configuration	
Backbone link capacity	10 Mbps
End connection capacity	100 Mbps
Queue capacity	150 packets
TCP variant	TCP Reno
TCP advertised window	65,535 bytes
Maximum segment size (MSS)	1,460 bytes
ECN support	Enabled (TCP only)
AQM Parameters of RED/ARED	
w_q (queue weight)	0.002
$\max_p$ (max drop probability)	0.02
α_{ARED}	0.01
β_{ARED}	0.9
ARED update interval	0.5 s
Reproducibility	
Random seed	Set via OMNeT++ seed-set feature

Reference algorithms RED and ARED algorithms are tested under the same queue capacity and same load scenarios. In the tests, RED and ARED are tested with different thresholds for different minth and maxth values based on information such as link capacity, average packet size, targeted delay. Among these, the threshold values that the algorithm shows the best performance are selected so that the drop rate remains at a reasonable level compared to normal internet traffic. In other thresholds, the values in the algorithm articles are adhered to. Simulation and traffic parameters are given in Table 5.1, Table 5.2 and Table 5.3.

Table 5.2: Traffic scenario parameters.

Traffic Type	**Scenario A - High**	**Scenario A - Medium**	**Scenario B - Dynamic**
Web Traffic (HTTP)			
Request size	intuniform(100-500 bytes)	intuniform(100-500 bytes)	Varies by phase
Response size	intuniform(50KB-10MB)	intuniform(20KB-6MB)	Varies by phase
Think time	Exponential(1s)	Exponential(3.5s)	Varies by phase
Requests/session	intuniform(50-150)	intuniform(20-70)	Varies by phase
Session idle time	Exponential(5s)	Exponential(20s)	Varies by phase
File Transfer (FTP)			
File size	1 GB	350 MB	250MB-1GB (varies)
Send interval	Uniform(0.1s, 0.5s)	Uniform(0.2s, 0.5s)	Varies by phase
VoIP Traffic			
Packet size	320 bytes	200 bytes	160-320 bytes (varies)
Packet interval	20 ms	20 ms	20 ms
Video Streaming			
Packet size	1,400 bytes	1,400 bytes	1,400 bytes
Packet interval	2.5 ms	5 ms	Varies by phase
Burst time	Exponential(2s)	Exponential(1.2s)	Varies by phase
Gap time	Exponential(0.05s)	Exponential(0.12s)	Varies by phase
AQM Threshold Parameters			
RED $\min_{th}$	25	5	30
RED $\max_{th}$	75	15	90
ARED $\min_{th}$	20	5	30
ARED $\max_{th}$	60	15	90

Parameter selections were configured to represent web browsing, file transfer, VoIP, and video traffic conditions. Tests were conducted using the parameters in Scenario A for different traffic densities. Scenario B generated varying traffic in different phases to evaluate the algorithms' adaptability to varying traffic densities. Scenario B, which includes variable packet sizes, variable sending frequencies, and variable connection frequencies, was created because real-world internet traffic behaves not in a fixed manner but also under varying densities. This approach will test the algorithms' performance under varying traffic conditions.

Table 5.3: Dynamic traffic scenario phases (Scenario B).

Parameter	**Phase 1 (0-250s)**	**Phase 2 (250-500s)**	**Phase 3 (500-750s)**	**Phase 4 (750-1000s)**
Traffic density	High	Medium	High	Medium
Web Traffic				
Request size	intuniform(100-500 bytes)	intuniform(100-500 bytes)	intuniform(100-500 bytes)	intuniform(100-500 bytes)
Response size	intuniform(50KB-10MB)	intuniform(20KB-6MB)	intuniform(50KB-10MB)	intuniform(20KB-6MB)
Think time	Exponential(1s)	Exponential(3.5s)	Exponential(1s)	Exponential(3.5s)
Requests/session	intuniform(50-150)	intuniform(20-70)	intuniform(50-150)	intuniform(20-70)
Session idle times	Exponential(5s)	Exponential(20s)	Exponential(5s)	Exponential(20s)
FTP				
File size	250 MB	100 MB	250 MB	100 MB
Send interval	Uniform(0.1s, 0.5s)	Uniform(0.2s, 0.5s)	Uniform(0.1s, 0.5s)	Uniform(0.2s, 0.5s)
VoIP				
Packet size	320 bytes	200 bytes	320 bytes	200 bytes
Video				
Packet interval	2.5 ms	5 ms	2.5 ms	5 ms
Burst time	Exponential(2s)	Exponential(1.2s)	Exponential(2s)	Exponential(1.2s)
Gap time	Exponential(0.05s)	Exponential(0.12s)	Exponential(0.05s)	Exponential(0.12s)

5.2. CoAL-AQM / RED /ARED Results Comparison

5.2.1. Scenario A. Constant Density Traffic Scenario

In this section, the performance of the algorithms is evaluated using the metrics mentioned in section 5.1.Comparative analyses are based on simulation scenarios performed with the same network topology and traffic scenarios. Experimental results reveal the advantages of the RL-based approach over traditional AQMs.

5.2.1.1. High Traffic. In the high-density traffic scenario, 2 TCP and 2 UDP flows representing different traffic types are active and the conditions that would create congestion in the backbone connection are tested.

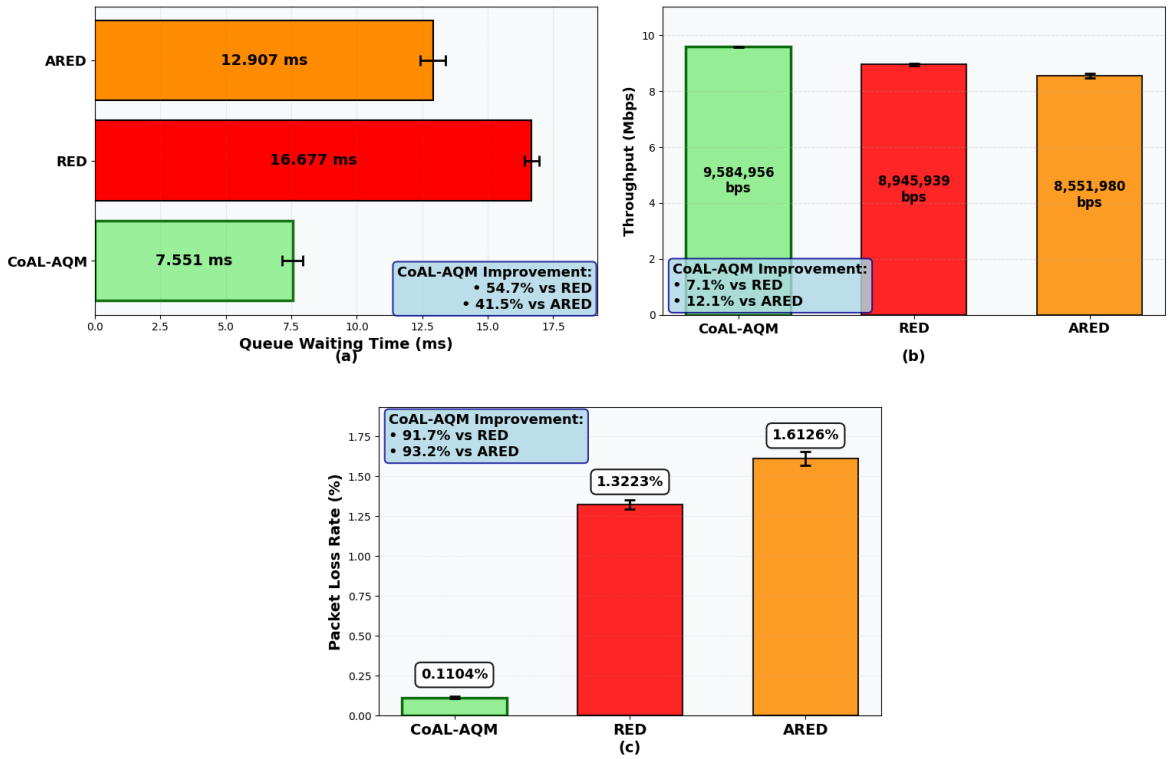


Figure 5.1: Scenario A high traffic performance results. (a) Queue delay performance. (b) Network throughput. (c) Packet loss rate.

As seen in Figure 5.1, the RED algorithm has an average latency of 16.677 ms and the ARED algorithm has an average latency of 12.907 ms. The CoAL-AQM algorithm has a remarkable performance with an average tail latency of 7.551 ms. CoAL-AQM provides a 54.7% improvement in latency compared to RED and 41.5% compared to ARED.

The throughput values presented in Figure 5.1 show that CoAL-AQM provides the highest data transmission speed with 9.584 Mbps. The capacity of the output link from the router where the measurements are made is 10 Mbps. This shows that 95.84% of the backbone capacity is used. The RED algorithm performed worse with an average throughput value of 8.945 Mbps (%89.45) and the ARED algorithm with an average throughput value of 8.551 Mbps (%85.51). CoAL-AQM provided a 7.1% increase in throughput compared to RED and a 12.1% increase compared to ARED. This proves that the algorithm maximizes link utilization by avoiding unnecessary packet drops.

When the packet loss rates in Figure 5.1 are examined, it is seen that CoAL-AQM has the lowest loss rate with 0.11%. RED has a loss rate of 1.32% and ARED has a loss rate of 1.61% and they dropped much more packets. The loss rate of CoAL-AQM is more than 10 times lower than ARED and RED. This shows that the smart dropping strategy of the algorithm is successful.

The results show that CoAL-AQM shows better results in all three performance metrics for the high density A traffic scenario. It shows that the algorithm provides both low latency, high throughput, low packet and low queue delay thanks to its dynamic threshold adjustment, protocol-based state and reward structure. Good use of ECN mechanism, adaptive drop/mark decisions provide a performance beyond the rigid structures of classical algorithms based on fixed threshold values.

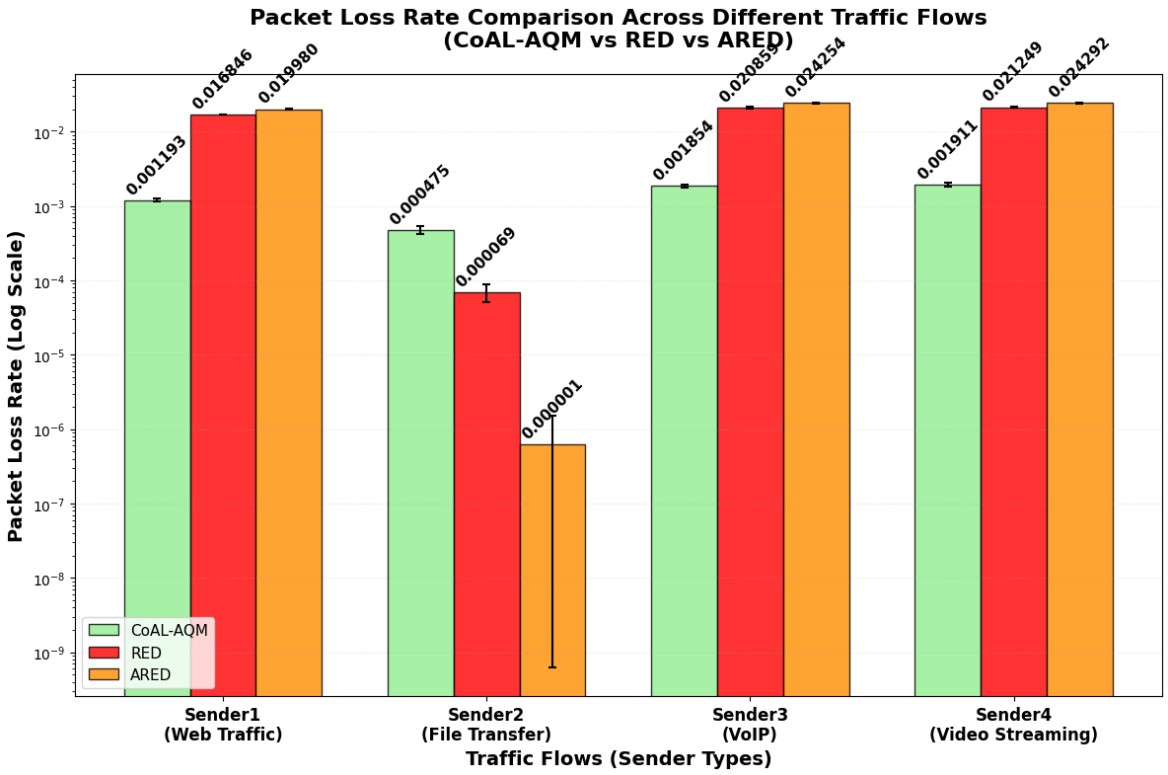


Figure 5.2: Scenario A high traffic PLR comparison.

Figure 5.2 shows the packet loss rates for different traffic types. The graph compares the performance of the algorithm for four different traffic types on a logarithmic scale. While CoAL-AQM drops more packets than other algorithms in file transfer

traffic, it exhibits the opposite behavior in delay-sensitive traffic types. Significant improvements in loss rates have been achieved, especially for real-time applications. The error bar range seen in Sender2 is in the high range because the ARED algorithm either does not drop any packets or drops a few packets, and this wide range is wide due to very low values. The number of packets generated with Sender1 is approximately 900 thousand, with Sender2 480 thousand, with Sender3 approximately 50 thousand, and with Sender4 approximately 180 thousand. These results show that CoAL-AQM manages packet loss rate values well for delay-sensitive traffic in different traffic environments.

Impact of Flow Awareness and RatioBit Mechanism: In order to evaluate the flow awareness feature of the CoAL-AQM algorithm, CoAL-AQM algorithm tests were conducted with and without the ratioBit mechanism. The results in Figure 5.3 show the effect of this mechanism on the algorithm performance.

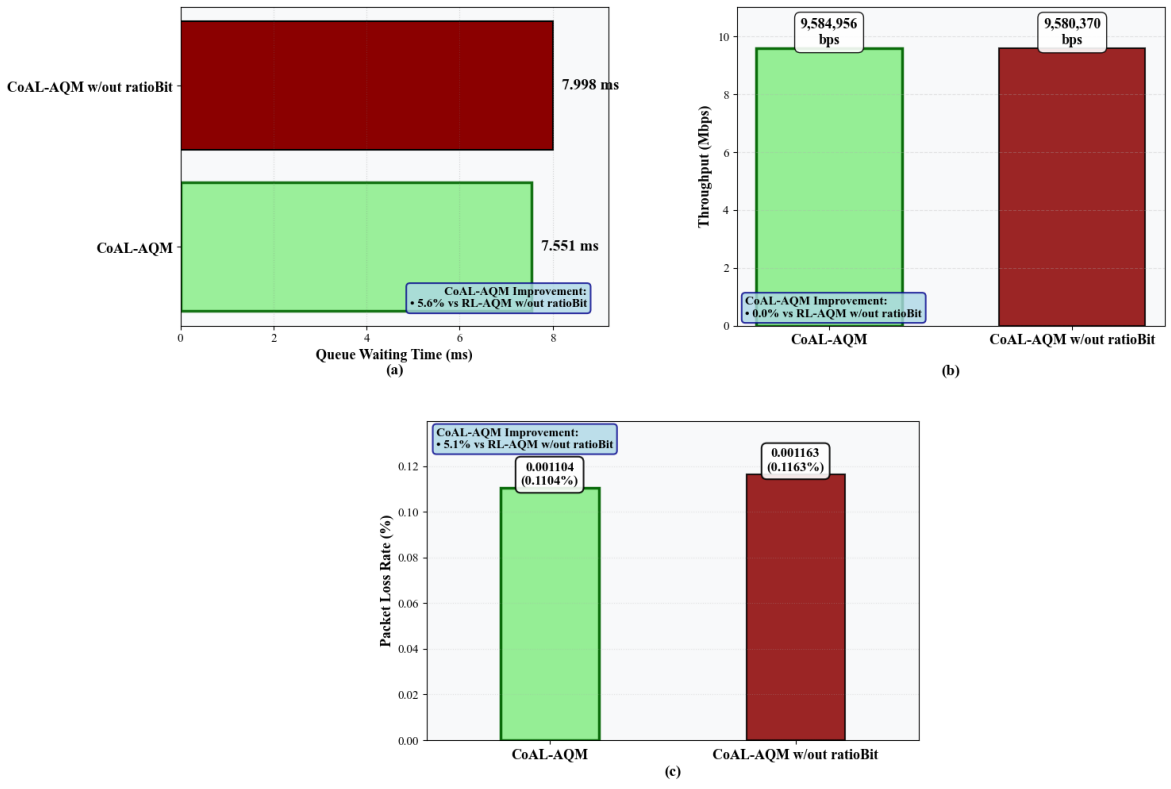


Figure 5.3: Scenario A high traffic ratioBit performance results. (a) Queue delay performance. (b) Network throughput. (c) Packet Loss Rate.

When the RatioBit mechanism is not applied, the basic network performance changes very little. When the RatioBit mechanism is activated, the queuing delay increases by 5.6% from 7.551 ms to 7.998 ms. The throughput is almost unchanged. The packet loss ratio increases by about 5% from 0.1163% to 0.1104%. These results show that the effect of RatioBit does not dramatically affect the overall system performance. The main effect is on the flow-based packet loss ratios.

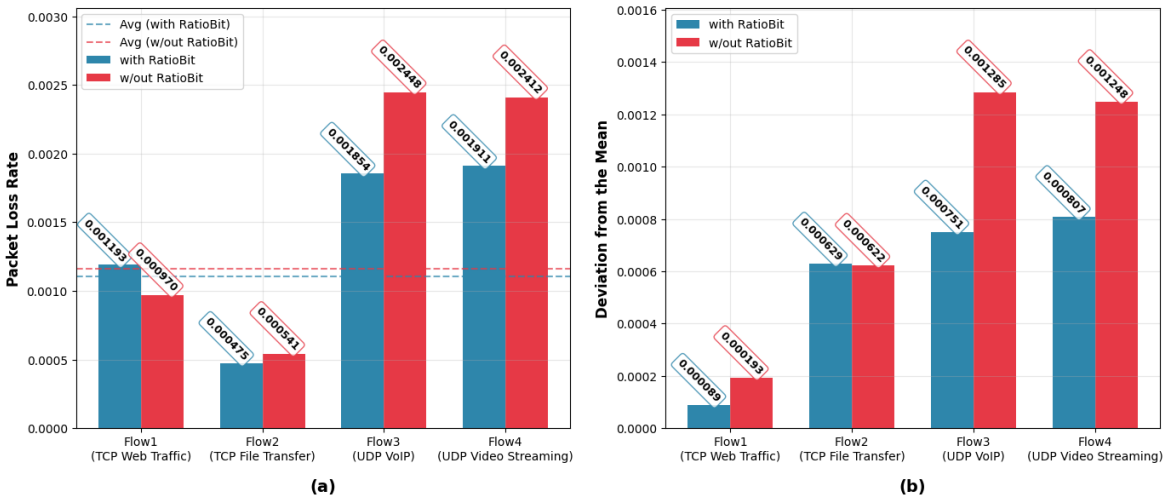


Figure 5.4: Scenario A high traffic ratioBit flow awareness results. (a) Flow based PLR comparion. (b) Deviation of flow PLR from the mean.

For CoAL-AQM, packet loss rates based on flow with and without ratioBit mechanism and the average loss rate of the system are shown in Figure 5.4 a. Here, flow refers to all flows between the sender and the router. For example, flow 1 shows the packet loss rate of all flows between sender1 and the router. Flow 1 has a packet drop rate of 0.001193 with ratioBit mechanism, 0.000970 without ratioBit mechanism, flow 2 has a packet drop rate of 0.000475 with ratioBit mechanism, 0.000543 without ratioBit mechanism, flow 3 has a packet drop rate of 0.001854 with ratioBit mechanism, 0.002448 without ratioBit mechanism, flow 4 has a packet drop rate of 0.001911 with ratioBit mechanism, and 0.002412 without ratioBit mechanism.

Especially in UDP type flow 3 and flow 4, when the ratioBit mechanism is active, it is seen that the PLR values are closer to the system average value of 0.001103. When

there is no ratioBit, UDP flows are more disproportionate and suffer from higher loss rates than TCP flows. The reason for this is that ECN is active in TCP flows in our tests and packet loss rates can be kept under control by increasing and decreasing *cwnd*.

The deviation from the mean graph in Figure 5.4 (b) clearly shows the effect of the ratioBit mechanism on flow awareness. When the ratioBit mechanism is active, the deviation of the PLR ratios from the mean decreases. The change is especially effective on the traffic types coming from sender 1, sender 3 and sender 4. Flow 2, which is formed by the traffic coming from sender 2, is the TCP flow that remains open for a long time. These flows tend to use the connection capacity well with the mark mechanism and to drop packets at a minimum level. Due to the effect of this and other sender traffic, no flow awareness effect was observed for flows coming from sender 2.

The RatioBit mechanism constitutes the CoAL-AQM flow aware characteristic. It has significantly improved the awareness between flows.

5.2.1.2. Medium Traffic. In the medium traffic scenario, the behavior of the algorithms will be examined in cases where the network density is reduced and there is less usage. The results in Figured 5.5 show the performance of the three algorithms in medium traffic.

The RED algorithm achieved an average delay of 3.198 ms, and the ARED algorithm achieved an average delay of 2.794 ms. In medium-density traffic conditions, the CoAL-AQM algorithm performed best with an average queue delay of 2.076 ms. This value is 35.1% lower than the result of RED and 25.7% lower than the result of ARED. The delay value here is directly related to the threshold value in the CoAL-AQM algorithm. While determining the threshold, we used the values of line capacity, average packet size, and expected delay. Currently, the algorithm works as a default

expected delay of 20 ms. As we said before, our algorithm does not make decisions for packets directly depending on this delay value.

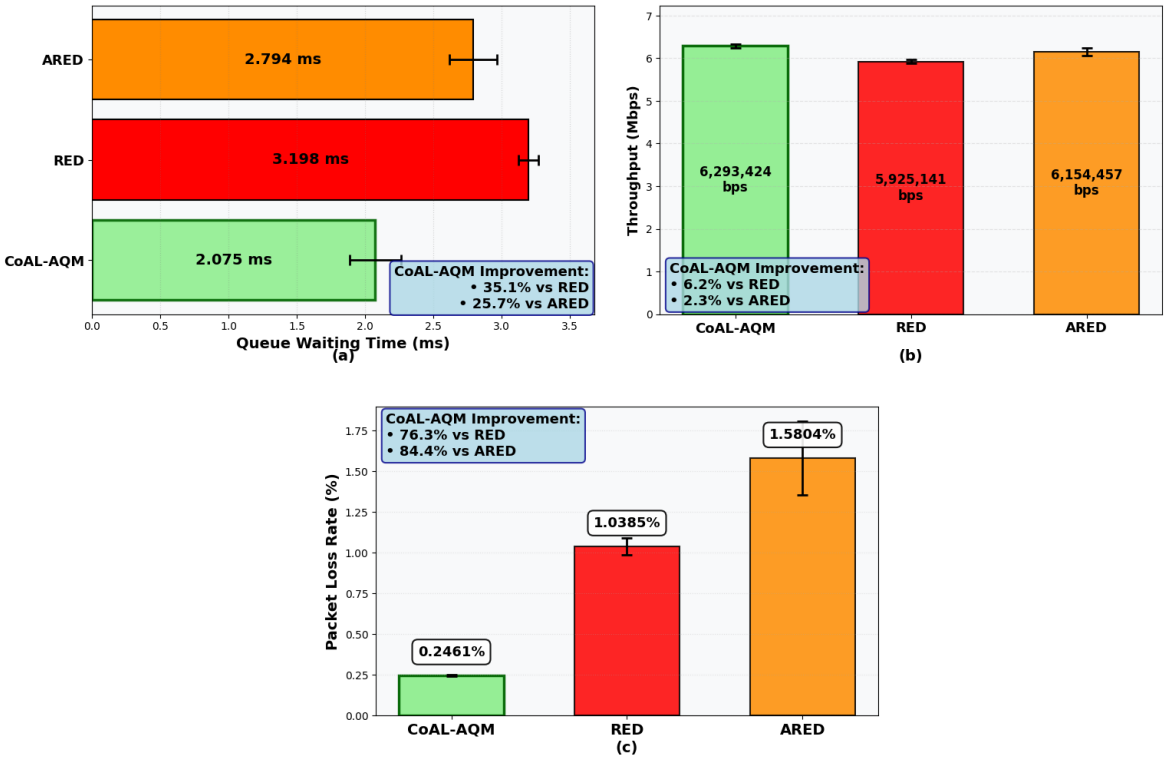


Figure 5.5: Scenario A medium traffic performance results. (a) Queue Delay Performance. (b) Network Throughput. (c) Packet Loss Rate.

Since there is a reward function according to the number of throughput and dropped packets, as seen here, it can keep the queue delay much lower than the expected delay in medium-density traffic.

When the throughput values are examined, RED reached 5.925 Mbps, ARED 6.154 Mbps. CoAL-AQM reached the highest data transmission speed with 6.293 Mbps.CoAL-AQM achieved a 6.2% increase in throughput compared to RED and 2.3% compared to ARED. It is expected that the throughput values in medium density will decrease. However, CoAL-AQM maintained its superiority in throughput and showed consistent performance under different load conditions.

When we look at Figure 5.5 c for packet loss rates, CoAL-AQM has achieved a very low packet loss rate of 0.0422%. RED has much higher packet loss rate values of 1.0385% and ARED has much higher packet loss rate values of 1.5804%. This significant difference shows that CoAL-AQM gives better results than other algorithms in medium density conditions. This result shows that fixed threshold values lead to suboptimal results even if threshold values are adjusted specifically for network traffic.

CoAL-AQM does not fill the queue unnecessarily when the traffic density decreases and achieves lower delay values than other algorithms. It avoids unnecessary packet dropping at low densities and provides high throughput advantage over other algorithms. As a result, CoAL-AQM exhibits good performance in medium-density traffic conditions thanks to its learning-based structure.

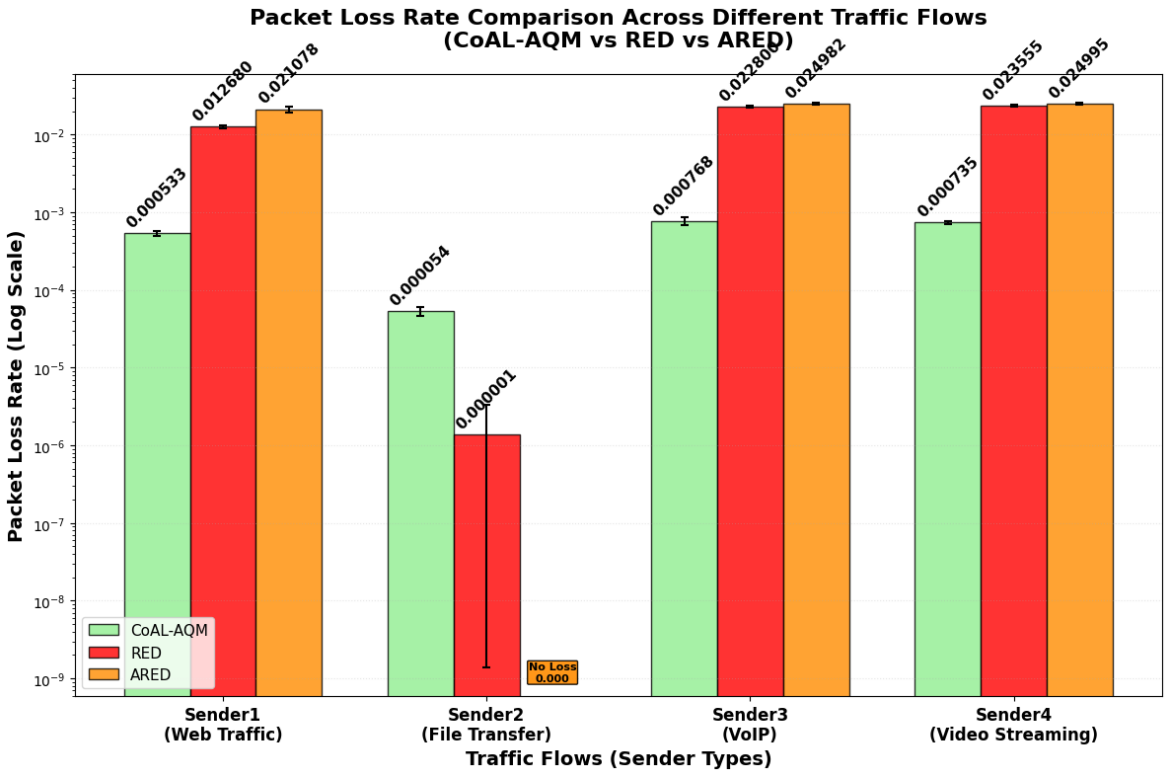


Figure 5.6: Scenario A mid traffic PLR comparison.

The packet loss rate results in Figure 5.6 show that CoAL-AQM can exhibit a more flow-aware approach to TCP and UDP traffic types. In file transfer type traffic, packet drop rate is kept low since it is increased and decreased in the most efficient

way with congestion window pass and drop signals for long TCP connections, therefore, 0 or 2 packets are dropped from different seed attempts out of approximately 480 thousand traffic packets sent. Therefore, the error bar interval width in the RED algorithm is ignored for sender2. The number of packets generated with sender1 is approximately 900 thousand, with sender3 approximately 50 thousand, and with sender4 approximately 180 thousand. Although CoAL-AQM prefers to drop more packets in long TCP flows, this situation is reversed in delay-sensitive traffic. The results show that CoAL-AQM exhibits packet dropping and protection behavior according to the traffic type in medium-density traffic and shows that the algorithm can meet QoS requirements more effectively.

Impact of Flow Awareness and Ratiobit Mechanism:Tests were conducted to evaluate the effects of the ratioBit mechanism for average traffic density.

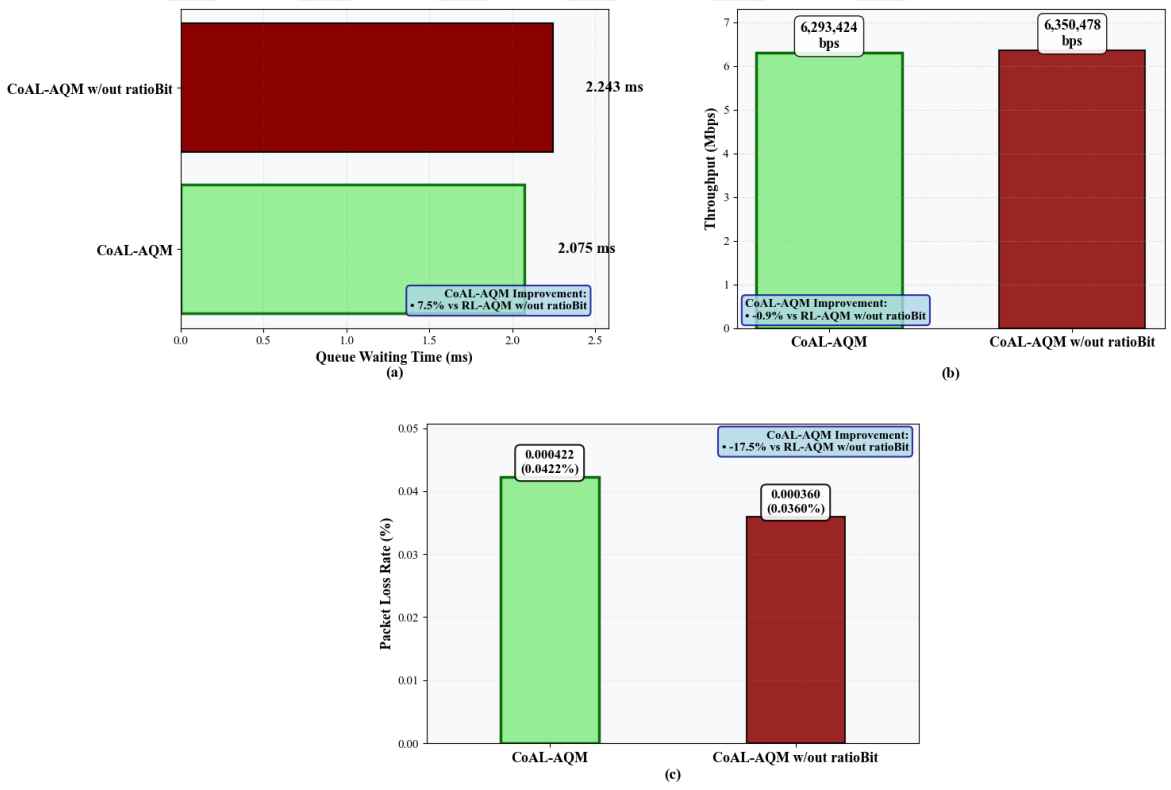


Figure 5.7: Scenario A medium traffic ratioBit performance results. (a) Queue Delay Performance. (b) Network Throughput. (c) Packet Loss Rate.

There is little change in the performance metrics when ratioBit mechanism is used and when it is not used. Throughput is 6.293 Mbps when ratioBit is used and 6.350 Mbps when ratioBit is not used. Throuhput is almost unchanged. Average queuing delay increased from 2.075 ms to 2.243 ms when ratioBit is used. Packet loss rate decreased from 0.0422% to 0.0360%. In medium traffic, lower latency but higher packet loss rate is observed when ratioBit mechanism is applied.

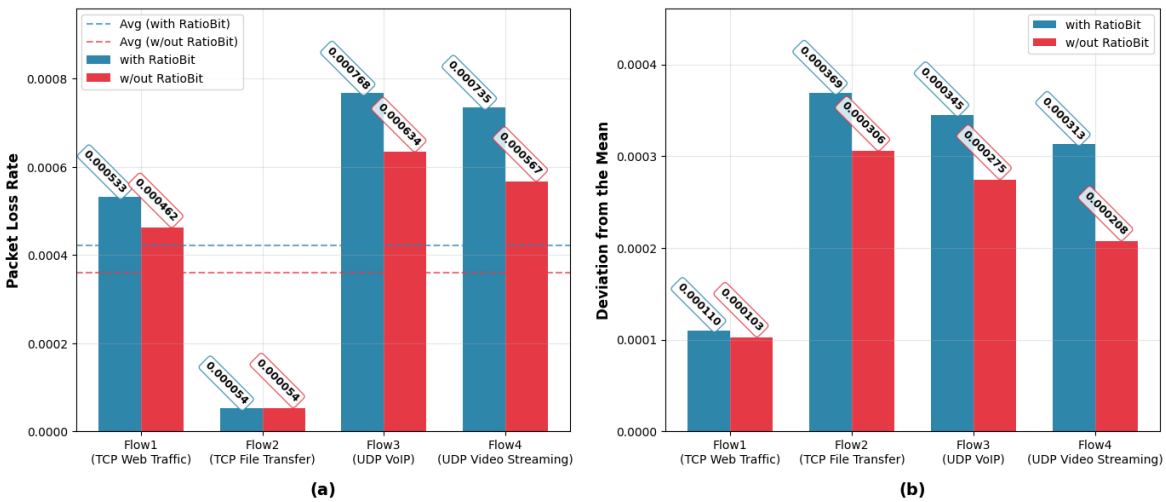


Figure 5.8: Scenario A medium traffic ratioBit flow awareness results. (a) Flow Based PLR Comparion. (b) Deviation of Flow PLR from the Mean.

When the PLR rates in Figure 5.8 (a) are examined, it is seen that the effect of the ratioBit mechanism is limited in medium density. The packet drop rate in all flows except Flow 2 decreases slightly. Unlike the high density scenario, no balancing mechanism effect is observed between flows in medium density. In the deviation graph in Figure 5.8 (b), the deviation value for both cases varies between 0.000110 and 0.000369 if ratioBit is active, and between 0.000103 and 0.000306 if ratioBit is disabled. These deviation values show that the deviation from the mean of the flow drop rates in medium density is much smaller than the flow drop rates in high density.

The ratioBit mechanism increased the packet loss rate while decreasing the average delay. It shows that this mechanism tries to prevent potential congestion while interfering with the reward values. The results of the medium density scenario con-

firm that the ratioBit mechanism is effective in high-load flow dominances because in medium density, slightly more than half of the router output 10 Mbps link value was used. The fact that the ratioBit mechanism has minimal effect in low-density conditions shows that the algorithm is not overly intrusive and adapts to changing conditions.

5.2.2. Scenario B-Traffic Scenario with Changing Density

The traffic scenario, where the traffic density changes at certain intervals, also tests the algorithm's adaptation ability during the 1000-second simulation period. Traffic continues in the form of high traffic and medium traffic at 250-second intervals. This traffic scenario simulates traffic fluctuations in real networks.

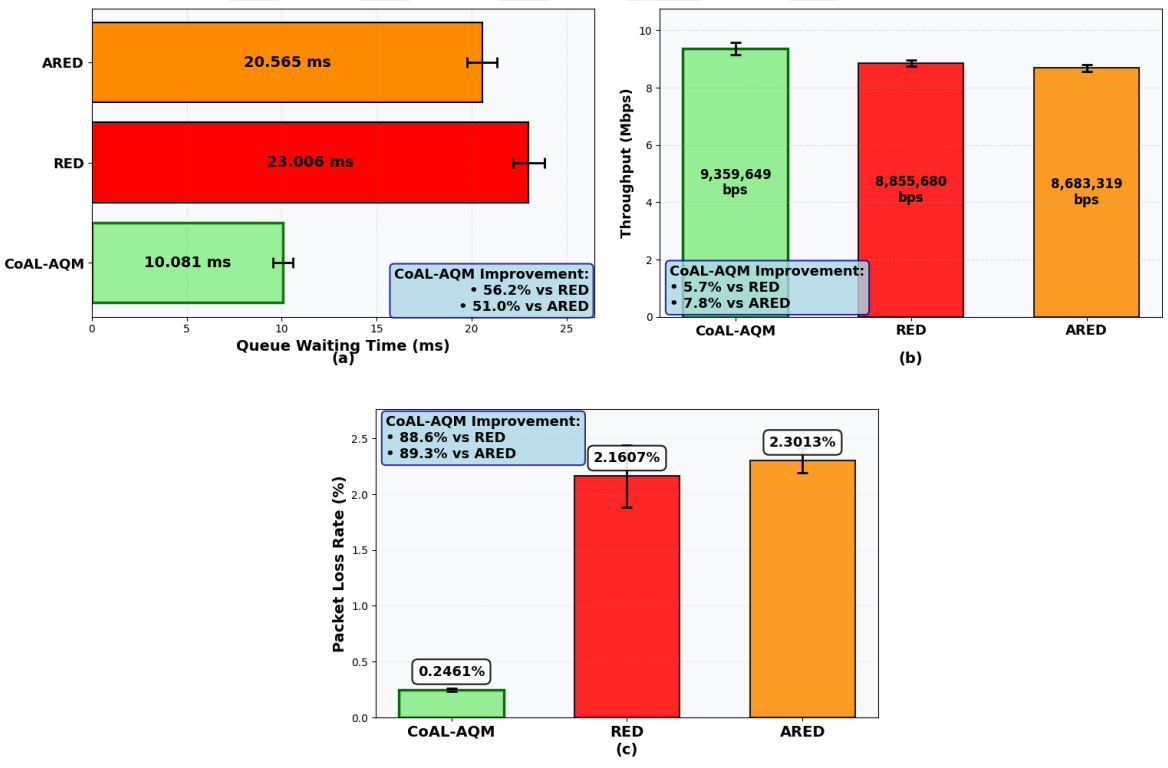


Figure 5.9: Scenario B traffic performance results. (a) Queue Delay Performance. (b) Network Throughput. (c) Packet Loss Rate.

In this traffic scenario, the RED algorithm has an average delay of 23.006 ms and ARED has an average delay of 20.565 ms. As can be seen in Figure 5.9, CoAL-

AQM has achieved a delay reduction of 56.2% compared to RED and 51% compared to ARED with an average queue delay of 10.081 ms.

When the throughput values are examined in Figure 5.9 (b), CoAL-AQM has reached a higher data transmission speed with 9.359 Mbps than the other algorithms. Here, CoAL-AQM has a 5.7% advantage over RED and 7.8% advantage over ARED, but the capacity of the link speed is 10 Mbps. This difference may increase in higher link speeds and denser traffic scenarios. In variable traffic conditions, CoAL-AQM's high performance in throughput in addition to delay shows that it is successful in transitioning between different traffic densities.

When we look at the packet loss rates in Figure 5.9 (c), CoAL-AQM has the lowest loss rate with 0.246%. RED exhibited much higher loss with 2.160% and ARED with 2.301% packet loss rates. CoAL-AQM has 88.6% lower loss rate compared to RED and 89.3% lower loss rate compared to ARED.

CoAL-AQM adapts to changes in traffic density by using the learned information in the Q-table and continuing to update some information. The learning process that continues throughout the simulation enables the algorithm to adapt to different traffic densities and patterns and to make correct decisions.

CoAL-AQM provides better user experience by minimizing the delay under different traffic densities. It maintains high throughput value under different traffic densities and selects the right strategy for the incoming packet with non-aggressive drop policy. Dynamic threshold setting strategy also provides different traffic management threshold selection for different traffics. As a result, CoAL-AQM exhibits superior performance under variable traffic densities. Learning-based infrastructure of the algorithm is a suitable solution for the dynamic and unpredictable nature of real networks.

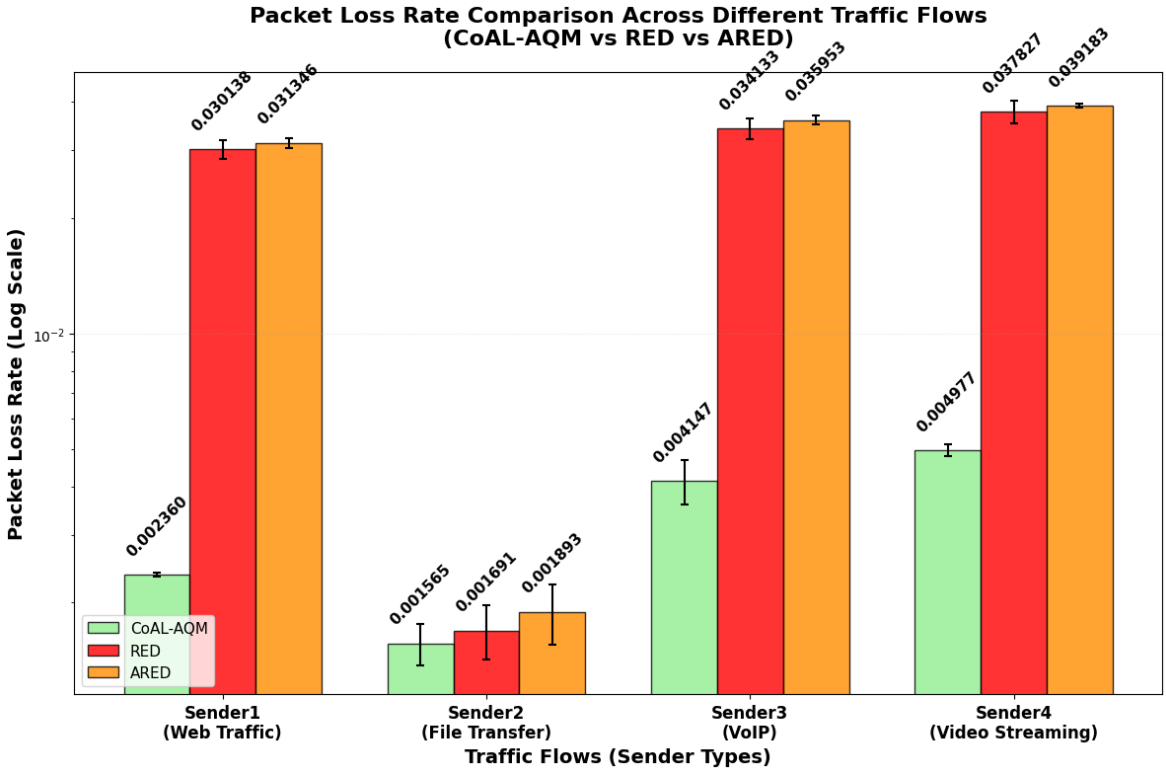


Figure 5.10: Scenario B PLR comparison.

Figure 5.10 shows the effect of traffic density fluctuations on the algorithm performance for a variable traffic scenario. Packet drop rates for different traffic flows are closer in CoAL-AQM algorithm than other algorithms. In delay-sensitive traffic and short-term TCP traffic, CoAL-AQM has a performance superiority over other algorithms. CoAL-AQM has adapted better to a traffic flow with a traffic density that changes every 250 seconds and has shown much better performance in packet loss rate ratios. CoAL-AQM's superiority continues in delay-sensitive traffic as well. These results emphasize the adaptation ability of the algorithm in variable traffic densities and show that good results are obtained with the Q-table update mechanism and dynamic threshold adjustment strategy.

Impact of Flow Awareness and Ratiobit Mechanism:In this section, the impact of the ratioBit mechanism on performance for variable traffic conditions will be examined.

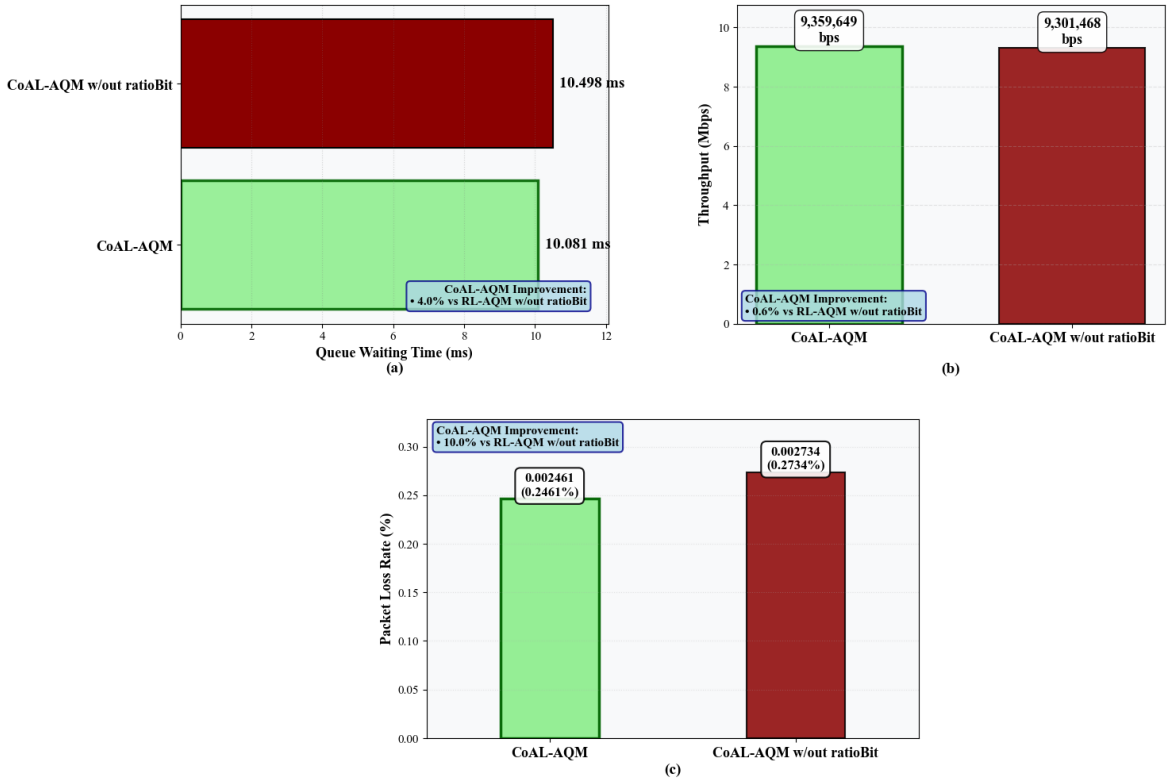


Figure 5.11: Scenario B traffic ratioBit performance results. (a) Queue Delay Performance. (b) Network Throughput. (c) Packet Loss Rate.

Referring to Figure 5.11 (a) and (b), when ratioBit mechanism is disabled, the average queue delay is 10.498 ms and the packet loss rate is 0.273%. When ratioBit mechanism is enabled, the delay is reduced to 10.081 ms and the packet loss rate is reduced to 0.246%. Throughput is 9.359 Mbps when ratioBit is enabled and 9.301 Mbps when it is disabled. These results show that ratioBit mechanism has a very small positive effect on the overall system performance. The main effect is on the flow-based packet loss rates.

For CoAL-AQM, packet loss rates based on flow with and without ratioBit mechanism and the average loss rate of the system are shown in Figure 5.12 (a). Here, flow refers to all flows between the sender and the router. For example, flow 1 shows the packet loss rate of all flows between sender1 and the router. Flow 1 has a packet drop rate of 0.002360 with ratioBit mechanism, 0.002010 without ratioBit mechanism, flow 2 has a packet drop rate of 0.001565 with ratioBit mechanism, 0.001622 without ra-

tioBit mechanism, flow 3 has a packet drop rate of 0.004147 with ratioBit mechanism, 0.006900 without ratioBit mechanism, flow 4 has a packet drop rate of 0.004977 with ratioBit mechanism, and 0.007601 without ratioBit mechanism.

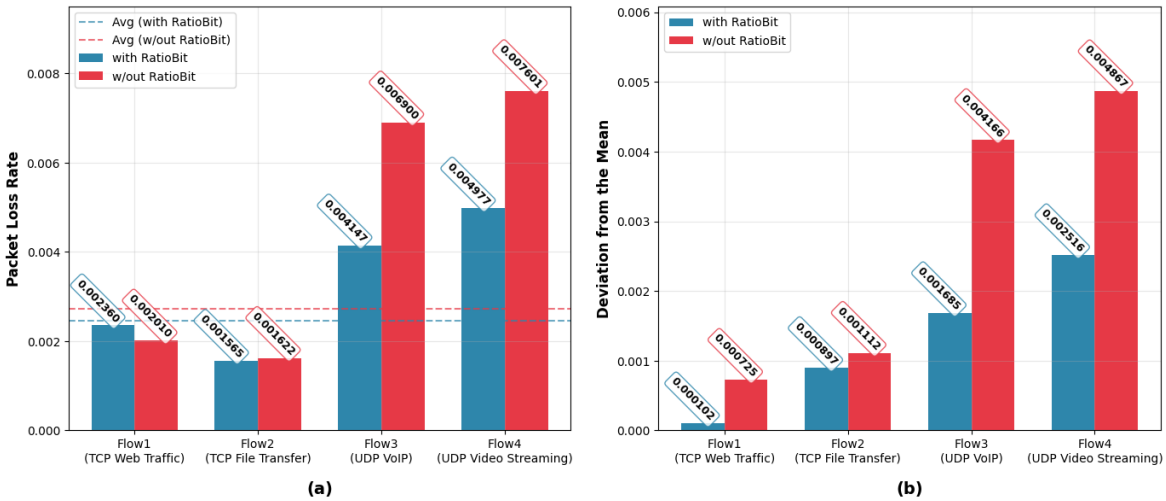


Figure 5.12: Scenario B traffic ratioBit flow awareness results. (a) Flow Based PLR Comparion. (b) Deviation of Flow PLR from the Mean.

Especially in UDP flows 3 and 4, when ratioBit is not active, high increases in PLR are observed. When ratioBit is active, flows 1, 3 and 4 approach the system average of 0.002461 PLR. When ratioBit is not active, UDP flows are more disproportionate and subject to higher loss rates than TCP flows. The reason for this is that ECN is active in TCP flows in our tests and packet loss rates can be kept under control by increasing and decreasing cwnd.

The average deviation graph in Figure 5.12 (b) shows the balancing effect of the ratioBit mechanism on flow awareness. When the ratioBit mechanism is active, the deviation of the PLR values of the flows from the system average of 0.002461 decreases. In the absence of the ratioBit mechanism, the average deviation values, especially for UDP flows, increased by 2-3 times. This also shows that the mechanism provides awareness, especially between different protocols.

In cases where traffic density changes, ratioBit mechanism shows balancing effects on packet loss rate between flows. Awareness is observed here for flows coming from different sources and packets in different protocols. Here, not only ratioBit mechanism but also UDP and TCP packet separation has an effect. As a result, flow-aware feature of ratioBit mechanism increases flow-aware performance of CoAL-AQM in network traffic with variable density.

Q-Table Adaptation Dynamics for Changing Density Traffic:

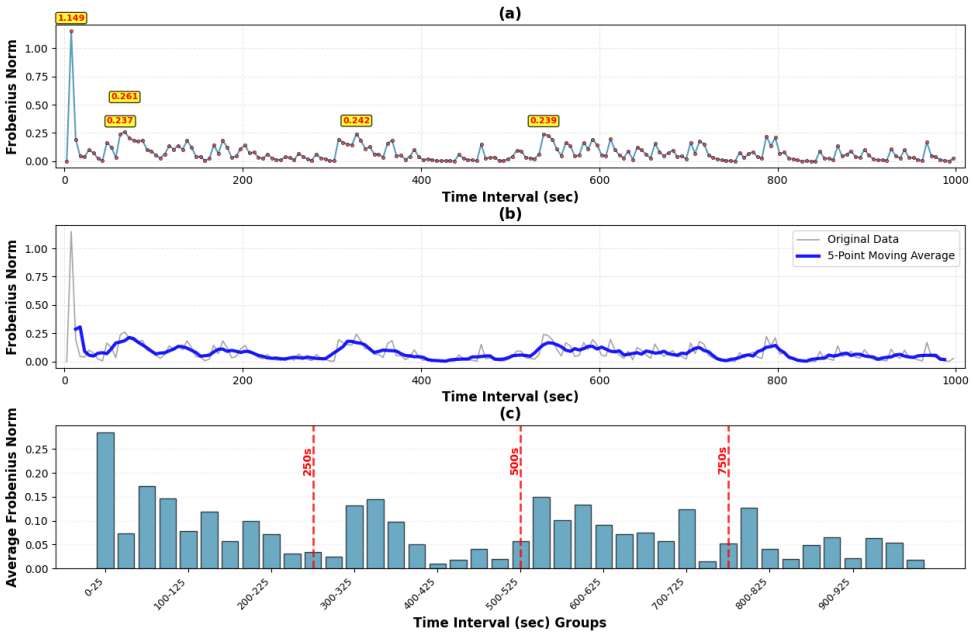


Figure 5.13: Scenario B q-table adaptation frobenius analysis. (a)Frobenius Norm Over Time Intervals (0-1000). (b)Frobenius Norm with 5-Point Moving Average. (c)Average Frobenius Norm by 5-Point Groups.

Figure 5.13 shows the Frobenius norm analysis of the Q-table adaptation behavior of the CoAL-AQM algorithm in a variable traffic scenario. Frobenius norm is a metric calculated as the square root of the sum of the squares of all elements of a matrix and measures the total change magnitude in the matrix. High Frobenius values are seen at the beginning of the simulation in Figure 5.13 (a). This shows that significant updates are made to the Q-table during the first learning phase. After the first 50

seconds, the value drops rapidly, indicating the adaptation of the algorithm to the traffic conditions. In Figure 5.13 (b), the 5-point moving average shows the changes in the Frobenius norm values at the moments of traffic density change. Figure 5.13 (c) shows the group averages for the first graph for 5 points. It shows the adaptation stages more clearly. While the Q-table stabilizes in approximately 100 seconds when switching from high traffic to medium traffic, the adaptation period is observed to be approximately 150 seconds when switching from medium traffic to high traffic. This different adaptation behavior also suggests that the algorithm can be improved in terms of adaptation in the transition to high-density traffic conditions. As a result, the Frobenius norm analysis shows once again that CoAL-AQM dynamically adapts to variable traffic conditions. The adaptation time of the algorithm of 100-150 seconds demonstrates its ability to adapt to real internet traffic fluctuations.

6. CONCLUSIONS AND FUTURE WORK

6.1. Results

In this thesis, an AQM algorithm based on RL Q-learning is developed as a solution to the parameter sensitivity and adaptation problem of classical AQM algorithms. The proposed algorithm was tested in different traffic scenarios and compared with classical AQM algorithms such as RED and ARED.

In the test results, the CoAL-AQM algorithm showed superior performance. The average queuing delay of CoAL-AQM decreased by 54.7% compared to RED and 41.5% compared to ARED in high-density traffic scenarios. The throughput value increased by 7.1% compared to RED and 12.1% compared to ARED. The packet loss rate is 12 times lower than RED and 14 times lower than ARED.

Experimental results show that CoAL-AQM outperforms RED and ARED algorithms. The average queuing delay of the algorithm is reduced by 54.7% compared to RED and 41.5% compared to ARED under high-density traffic conditions. At the same time, throughput values are increased by 7.1% compared to RED and 12.1% compared to ARED. The loss rate of CoAL-AQM is measured as 12 times lower than RED and 14.6 times lower than ARED, and the highest improvement is observed in packet loss rate.

Similar success is achieved in the medium-density test scenario. CoAL-AQM achieved an average tail delay value of 2.076 ms. This value shows a 35.1% improvement over RED and 25.7% improvement over ARED. The significant difference in packet loss rate continued here as well, with CoAL-AQM dropping much less packets than RED and ARED with a PLR of 0.0422%. CoAL-AQM is also ahead of RED and ARED in the drop value. The improvement observed in all three delay-throuhput-packet loss

rates under different load conditions reveals that the algorithm performs well under different load conditions.

The performance of the algorithms under variable test conditions for variable traffic density is measured. In this scenario where the traffic density changes with 250-second periods, CoAL-AQM provided 56.2% improvement over RED and 51% improvement over ARED by keeping the average queuing delay at 10.081 ms. CoAL-AQM, which gave 5.7% better results than RED and 7.8% better results than ARED in terms of throw value, also showed much lower rates in packet loss rate than RED and ARED. With its continuous learning-based structure, CoAL-AQM quickly adapted to traffic changes.

6.2. Summary of Contributions

This thesis study has brought important innovations to the AQM literature. The adaptive structure of the algorithm has eliminated the parameter tuning needs of classical AQMs. The algorithm, which can adapt to different traffics with its dynamically calculated threshold and continuous learning mechanism, has adapted well to different network conditions. This is a self-tuning AQM solution suitable for the dynamic structure of modern networks.

The multi-purpose reward function optimizes the trade-offs between queueing time-throughput-packet loss rate in a balanced manner. It improves the overall system performance. The algorithm achieves high throughput and low packet loss while maintaining the low latency target.

Another important contribution is the use of protocol type in the state space and reward function. Considering different protocol types such as TCP and UDP, our algorithm is designed to develop optimized queue management strategies for different protocol types. Another innovation is the flow awareness feature, which controls the excessive penalization of each flow with the ratioBit mechanism.

6.3. Future Directions

The results of this thesis study have shown the potential of the RL-based AQM approach. It is critical to test the algorithm in real-world applications. Software Defined Networking environments can be an ideal platform for implementing CoAL-AQM. The performance of the algorithm can be evaluated on real network hardware using the OpenFlow protocol.

The application of more advanced RL architectures to AQM problems should be investigated. Deep learning-based approaches such as Deep Q-Networks (DQN) can improve performance, but their applicability to real-life routers should be investigated.

Energy efficiency is of critical importance in modern networks. Energy-aware versions of the CoAL-AQM algorithm should be developed to explore strategies that will minimize system energy consumption while maintaining performance targets.

REFERENCES

1. Sandvine Incorporated ULC, "Global Internet Phenomena Report", Report, 2024, `https://www.sandvine.com/phenomena`, accessed on May 13, 2025.

2. Feng, W.-C., D. Kandlur, D. Saha and K. Shin, "A Self-configuring RED Gateway", *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, Vol. 3, pp. 1320–1328, New York, NY, USA, 1999.

3. Komolafe, T. A., O. A. Oladimeji, O. O. Adiamo, O. O. Oyebode and R. O. A. Oyeniyi, "A Simulation-based Comparative Study of Controlled Delay with Random Early Detection for Network Performance Evaluation", *Information Technology*, Vol. 6, No. 1, pp. 1–10, 2023.

4. Floyd, S. and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, pp. 397–413, 1993.

5. Floyd, S., R. Gummadi and S. Shenker, *Adaptive RED: An Algorithm for Increasing the Robustness of Random Early Detection's Active Queue Management*, Technical report, AT&T Center for Internet Research at ICSI, Berkeley, CA, 2001.

6. Nichols, K. and V. Jacobson, "Controlling Queue Delay", *Communications of the ACM*, Vol. 55, No. 7, pp. 42–50, 2012.

7. Pan, R., P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker and B. VerSteeg, "Proportional Integral controller Enhanced: A Lightweight Control Scheme to Address the Bufferbloat Problem", *2013 IEEE 14th International Conference on High Performance Switching and Routing*, pp. 148–155, Taipei, Taiwan,

2013.

8. Sutton, R. S. and A. G. Barto, *Reinforcement Learning: An Introduction*, A Bradford Book, Cambridge, MA, USA, Second Edition, 2018.

9. Watkins, C. J. C. H. and P. Dayan, "Q-learning", *Machine Learning*, Vol. 8, No. 3-4, pp. 279–292, 1992.

10. Mao, H., R. Netravali and M. Alizadeh, "Neural Adaptive Video Streaming with Pensieve", *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, p. 197–210, Association for Computing Machinery, New York, NY, USA, 2017.

11. Jay, N., N. Rotman, B. Godfrey, M. Schapira and A. Tamar, "A Deep Reinforcement Learning Perspective on Internet Congestion Control", K. Chaudhuri and R. Salakhutdinov (Editors), *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97 of *Proceedings of Machine Learning Research*, pp. 3050–3059, PMLR, Long Beach, California, USA, 2019.

12. Su, Y., L. Huang and C. Feng, "A Q-learning-based Active Queue Management Scheme", *Journal of Internet Technology*, Vol. 19, No. 4, pp. 1169–1178, 2018.

13. Jiang, F., C. Feng, C. Zhu and Y. Sun, "Performance Analysis of Active Queue Management Algorithm based on Reinforcement Learning", *Journal Européen des Systèmes Automatisés*, Vol. 53, No. 5, 2020.

14. Kim, M., M. Jaseemuddin and A. Anpalagan, "Deep Reinforcement Learning Based Active Queue Management for IoT Networks", *Journal of Network and Systems Management*, Vol. 29, No. 3, 2021.

15. Kingman, J. F. C., *Poisson Processes*, Clarendon Press, Oxford, 1992.

16. Kendall, D. G., "Stochastic Processes Occurring in the Theory of Queues and

Their Analysis by the Method of the Imbedded Markov Chain", *The Annals of Mathematical Statistics*, Vol. 24, No. 3, pp. 338–354, 1953.

17. Thomas, M. U., "Queueing Systems. Volume 1: Theory (Leonard Kleinrock)", *SIAM Review*, Vol. 18, No. 3, pp. 512–514, 1976.

18. "Internet Protocol", Request for Comments, No. 791, 1981, `https://www.rfc-editor.org/info/rfc791`, accessed on May 13, 2025.

19. Eddy, W., "Transmission Control Protocol", Request for Comments, No. 9293, 2022, `https://www.rfc-editor.org/info/rfc9293`, accessed on May 13, 2025.

20. Jain, R., D. Chiu and W. Hawe, "A Quantitative Measure of Fairness and Discrimination", *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, Vol. 21, No. 1, pp. 2022–2023, 1998.

21. Meng, Z., N. Atre, M. Xu, J. Sherry and M. Apostolaki, "Confucius: Achieving Consistent Low Latency with Practical Queue Management for Real-time Communications", arXiv preprint arXiv:2310.18030, 2024.

22. Tanenbaum, A. S. and D. J. Wetherall, *Computer Networks*, Prentice Hall Press, Upper Saddle River, NJ, Fifth Edition, 2010.

23. Jacobson, V., "Congestion Avoidance and Control", *SIGCOMM Computer Communication Review*, Vol. 18, No. 4, pp. 314–329, 1988.

24. Floyd, S., D. K. K. Ramakrishnan and D. L. Black, "The Addition of Explicit Congestion Notification to IP", Request for Comments, No. 3168, 2001, `https://www.rfc-editor.org/info/rfc3168`, accessed on May 13, 2025.

25. Gettys, J. and K. Nichols, "Bufferbloat: Dark Buffers in the Internet: Networks Without Effective AQM May Again Be Vulnerable to Congestion Collapse", *Queue*, Vol. 9, No. 11, pp. 40–54, 2011.

26. Kaelbling, L. P., M. L. Littman and A. W. Moore, "Reinforcement Learning: A Survey", *Journal of Artificial Intelligence Research*, Vol. 4, No. 1, pp. 237–285, 1996.

27. Ng, A. Y., D. Harada and S. J. Russell, "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping", *Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 278–287, San Francisco, CA, USA, 1999.

28. Even-Dar, E. and Y. Mansour, "Learning Rates for Q-learning", *Journal of Machine Learning Research*, Vol. 5, pp. 1–25, 2003.

29. Tokic, M., "Adaptive ε-greedy Exploration in Reinforcement Learning Based on Value Differences", *Proceedings of the 33rd Annual German Conference on Advances in Artificial Intelligence*, KI'10, p. 203–210, Springer-Verlag, Berlin, Heidelberg, 2010.

30. Varga, A. and R. Hornig, "An Overview of the OMNeT++ Simulation Environment", *Proceedings of International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, pp. 1–10, Marseille, France, 2008.

31. Kurose, J. F. and K. W. Ross, *Computer Networking: A Top-Down Approach*, Pearson, USA, Sixth Edition, 2012.

32. Adams, R., "Active Queue Management: A Survey", *IEEE Communications Surveys & Tutorials*, Vol. 15, No. 3, pp. 1425–1476, 2013.

33. Baker, F. and G. Fairhurst, "IETF Recommendations Regarding Active Queue Management", Request for Comments, No. 7567, 2015, `https://www.rfc-editor.org/info/rfc7567`, accessed on May 13, 2025.