International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Journal Européen des Systèmes Automatisés
Vol. 53, No. 5, October, 2020, pp. 637-644
Journal homepage: http://iieta.org/journals/jesa

Performance Analysis of Active Queue Management Algorithm Based on Reinforcement Learning

Fuchun Jiang, Chenwei Feng*, Chen Zhu, Yu Sun

Fujian Key Laboratory of Communication Network and Information Processing, Xiamen University of Technology, Xiamen 361024, China

Corresponding Author Email: cwfeng@xmut.edu.cn

https://doi.org/10.18280/jesa.530506

Received: 17 May 2020
Accepted: 2 September 2020

Keywords:
congestion control, active queue management (AQM), random early detection (RED), reinforcement learning AQM (RLAQM)

ABSTRACT

In the information society, data explosion has led to more congestion in the core network, dampening the network performance. Random early detection (RED) is currently the standard algorithm for active queue management (AQM) recommended by the Internet Engineering Task Force (IETF). However, RED is particularly sensitive to both service load and algorithm parameters. The algorithm cannot fully utilize the bandwidth at a low service load, and might suffer a long delay at a high service load. This paper designs the reinforcement learning AQM (RLAQM), a simple and practical variant of RED, which controls the average queue length to the predictable value under various network loads, such that the queue size is no longer sensitive to the level of congestion. Q-learning was adopted to adjust the maximum discarding probability, and derive the optimal control strategy. Simulation results indicate that RLAQM can effectively overcome the deficiency of RED and achieve better congestion control; RLAQM improves the network stability and performance in complex environment; it is very easy to migrate from RED to RLAQM on Internet routers: the only operation is to adjust the discarding probability.

1. INTRODUCTION

The data volumes are exploding due to the expansion of network scale and the growing number of users. Congestion will occur in the network, when the volume of data being transmitted approaches the maximum processing capacity of the network [1]. In the event of a congestion, the traditional congestion control strategy, transmission control protocol (TCP), can no longer satisfy the quality of service (QoS) requirements. To avoid congestion, the packets in the router buffer can be marked or discarded by a specific queue management mechanism configured on the router, or resume the normal network state as soon as possible.

There are two kinds of queue management schemes: passive queue management (PQM) and active queue management (AQM). The most representative PQM algorithm is Drop-Tail, which only discards packets when the router queue is saturated. Despite its simplicity, Drop-Tail is prone to problems like deadlock, queue saturation, and global synchronization. As a result, AQM [2] has become the popular queue management scheme in practice. The core idea of AQM is to control the queue length of the router cache by collecting and predicting the network state prior to congestion.

Over the years, AQM has been continuously improved by experts and scholars. The improved AQM algorithms fall into three categories: heuristic algorithms, optimization algorithms, and control algorithms [3]. Depending on intuition, the heuristic algorithm includes RED, gentle RED (GRED) [4], three-section RED (TRED) [5], fair weighted multi-level RED (FWMRED) [6], AQM with random dropping (AQMRD) [7], and new modified dropping function (NMDF) [8]. Specifically, GRED increases network stability by replacing the discontinuous change of discarding probability from P_{max} to 1 should with a gentle slope. To manage congestion level, FWMRED redefines the discarding probability in multi-level RED (MRED), and produces dynamic weighted traffic to enhance the stability of parameters [9]. AQMRD incorporates the change rate of average queue size as a parameter to capture the time variation of average queue size. The heurism algorithms significantly outperform the Drop-Tail. However, many heuristic algorithms need to configure their parameters as per the specific network conditions. The influence of their parameters is not fully known. If the parameters are not configured properly, the heuristic algorithms will quickly enter the unstable state, failing to respond timely to the dynamic changes of the network. This will result in reduced network utilization and deteriorated network performance.

Aiming to maximize network utilization, the optimization algorithms are essentially solvers of the gradient optimization problem. Typical optimization algorithms are random exponential marking (REM) [10], adaptive virtual queue (AVQ) [11], delay utilization knee (DUK) [12], artificial neural network-based AQM (ANB-AQM) [13], and deterministic perceptron-based AQM (DPB) [14]. Among them, DUK relies on measured runtime of the network, a natural threshold, and the knee on delay-utilization curve, rather than preset or pre-tuned parameters. To control congestion and ensure QoS, ANB-AQM tunes the parameters through self-learning to adapt to network nonlinearity. On the upside, the optimization algorithms perform well in analysis; on the downside, these algorithms are too complex to implement, and require the network parameters (e.g. the number of streams, and round-trip time) to be known in advance. But these parameters often change frequently,

making it difficult to design such algorithms.

Many AQM control algorithms have been developed under different control strategies, namely, proportional-integral (PI) controller [15], proportional–derivative (PD) controller [16], feedforward AQM (FF-AQM) [17], self-tuning compensated proportional-integral-derivative (ST-CPID) controller [18], and stable AQM (SAQM) [19]. Hollot et al. [15] designed a stabilizing PI controller that uses instantaneous samples of the queue size, and successfully overcame the instability and low-frequency oscillations of the low-pass filter design of RED in the regulated output. Wang et al. [17] proposed the FFAQM under feedforward model predictive control (MPC), which stabilizes the queue length at a target value as quickly as possible and smooths out the burst traffic. Kahe and Jahangir [18] put forward the ST-CPID to address the time-variation of network conditions induced by parameter changes and unresponsive connections. The control algorithms based on classical control theory can effectively control queue length, but their designs depend on approximate linear model or specific network model. The network performance will deteriorate, if the network environment changes or the model does not fit. In addition, the control algorithms are sensitive to the setting of system parameters.

In recent years, reinforcement learning (RL) has been introduced to network congestion control. Being a learning process through trial and error, RL attaches great importance to the interaction between the agents and the environment, and continuously adjusts the action selection strategy based on the feedback from the environment. The main advantage of RL is the elimination of the need for prior knowledge of the network or precise mathematical model of the control object. Through RL, the controller can be designed robustly, regardless of the model accuracy. Therefore, RL is highly suitable for networks with significant time variations and emergent properties.

In general, the optimization and control algorithms have better AQM performance than heuristic algorithms. Nevertheless, these algorithms are too complex to be implemented easily. Besides, any new implementation of AQM will greatly affect the design of router structure and software flow [20]. At present, AQM algorithms on Internet routers are mostly under the standard of RED. Therefore, it is of great significance to simplify and improve the practicality of RED-based AQM algorithms.

This paper presents the a simple and practical AQM algorithm called RLAQM based on RED algorithm. In this algorithm, Q-learning is employed to adaptively adjust the maximum discarding probability, making the algorithm less sensitive to parameter setting. Moreover, the network performance was optimized through learning under the dynamic network environment, thereby preventing congestion. Simulation results show that the RLAQM achieved stable and excellent performance under complex network environment.

The remainder of this paper is organized as follows: Section 2 summarizes the problems of RED algorithm; Section 3 introduces the RLAMQ algorithm; Section 4 simulates the performance of the proposed algorithm; Section 5 concludes the entire research.

2. PROBLEMS OF RED ALGORITHM

The RED algorithm detects congestion by monitoring the average queue length (avg) in the router cache. After detecting any sign of congestion, the algorithm will discard or mark individual groups at a certain probability, and inform the source to avoid congestion. To prevent global synchronization and bias against burst flows, the RED algorithm adopts a random strategy in the discarding or marking process. In addition, a reasonable queue length is maintained by imposing an upper bound on the average queue length. Any incoming packets longer than the upper bound will be discarded. The average queue length is calculated by exponentially weighted moving average (EWMA):

$$avg_n = (1-\omega_q)avg_{n-1} + \omega_q q \tag{1}$$

where, q is the instantaneous queue length; $\omega_q \in [0, 1]$ is the sensitivity of average queue length to q. The recommended value of ω_q is 0.002.

The discarding or marking probability must be a function of average queue length, which reflects the congestion situation. In the packet loss mode, the discarding probability can be calculated by:

$$p_b = \begin{cases} 0 & avg \in [0, min_{th}] \\ max_p \times \left(\dfrac{avg - min_{th}}{max_{th} - min_{th}}\right) & avg \in [min_{th}, max_{th}) \\ 1 & avg \in [max_{th}, +\infty) \end{cases} \tag{2}$$

where, max_p is the maximum discarding probability; min_{th} and max_{th} are the lower and upper bounds of average queue length, respectively. Figure 1 shows the relationship between average queue length and the discarding probability p_b.

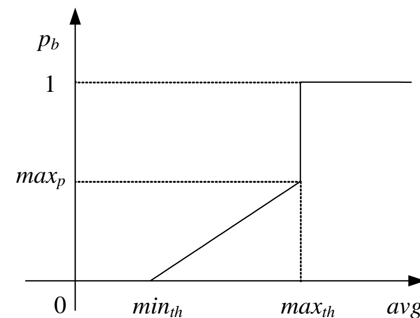


Figure 1. The curve of discarding probability in RED algorithm

To disperse the discarded groups, the discarded probability p_b needs to be modified as:

$$p_a = \frac{p_b}{1 - count \times p_b} \tag{3}$$

where, $count$ is the number of packets entering the router cache queue since the previous packet loss.

RED algorithm can solve most problems of the Drop-Tail, namely, deadlocks, full queues, and global synchronization. However, low link utilization or forced packet loss may arise from improper setting of algorithm parameters. In severe cases, the algorithm might suffer queue oscillation, throughput degradation, and delay jitter deterioration [5].

To overcome the above defects, this paper proposes the RLAQM algorithm, which stabilizes the overall throughput and delay and optimizes the data transmission in the network by adaptively adjusting the maximum discarding probability under different network loads.

3. ALGORITHM DESIGN

3.1 Q-learning

Inspired by Markov decision process (MDP) [21], RL emphasizes the interaction between agents and the environment. The action execution is determined as per the current state of the environment, and the performance is evaluated based on the return value of the environment on the action, and the migration from the current state to the next state. The optimal decision is learned through the accumulation of experience. The standard RL model involves the following factors:

(1) S={s_1, s_2, … s_m}: the set of environmental states;
(2) A={a_1, a_2, …a_n}: the set of agent actions;
(3) r: the return value of the environment on the action;
(4) π: S→A: the strategy executed by the agent.

Figure 2 illustrates the relationship between these factors [22]:

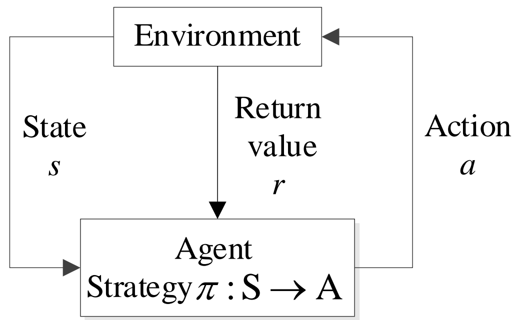


Figure 2. The relationship between factors of the standard RL model

The learning process is completed iteratively. In each iteration, the agent collects the current environmental state s_t∈S, selects an action a∈A according to the strategy π, and executes the action in the environment. Then, the environment state changes to s'∈S, while a return value r is generated at time t, and fed back to the agent. Based on the return value r and the environment state s', the agent updates its strategy π, kicking off the next iteration. Through continuous iterations, the agent searches for the optimal strategy $\pi^*(s)$ ∈A under each environmental state s∈S, thus maximizing the cumulative expectation of the return value:

$$V^{\pi}(s) = E\left\{\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) | s_0 = s\right\} \tag{4}$$

where, γ∈[0, 1) is a discount factor, reflecting the importance of future return value to the current state. The discount factor can be solved by the Bellman criterion. The maximum of formula (4) can be obtained by:

$$V^*(s) = V^{\pi^*}(s) = \max_{a\in A}\left[R(s,a) + \gamma \sum_{s'\in S} P_{s,s'}(a) V^*(s')\right] \tag{5}$$

where, $R(s, a)$ is the expectation of $r(s_t, a_t)$; $P_{s,s'}(a)$ is the state transition probability, reflecting the probability for the environmental state s to reach the next state s' under the action a.

Q-learning, a model-free RL algorithm extended from SARSA (state-action-reward-state-action), represents the experience of the agent with the state-action pair function $Q(s, a)$. The algorithm can find the optimal strategy π^* without knowing the specific values of $R(s, a)$ and $P_{s,s'}(a)$. The $Q(s, a)$ in strategy π can be expressed as $Q^{\pi}(s, a)$:

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s'\in S} P_{s,s'}(a) V^{\pi}(s') \tag{6}$$

Under the optimal decision condition:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'\in S} P_{s,s'}(a) V^*(s') \tag{7}$$

From formulas (5) and (6):

$$V^*(s) = \max_{a\in A} Q^*(s,a) \tag{8}$$

$$\pi^*(s) = arg\max_{a\in A} Q^*(s,a) \tag{9}$$

Q-learning iteratively updates the Q value at each moment:

$$Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha(r_t + \gamma \max_{a'\in A} Q_t(s',a')) \tag{10}$$

where, α∈[0, 1) is learning rate. When t→∞, if α decreases to zero, $Q_t(s, a)$ will converge to the optimal value of $Q^*(s, a)$. Following formula (4), the agent can obtain the optimal strategy π^* [23].

3.2 RLAQM

Besides being applicable to various loads, a reasonable AQM strategy must avoid forced packet loss and low link utilization. In RED algorithm, the discarding probability is linearly corelated with the average queue length. If the maximum discarding probability max_p is too large, the congestion level will be overestimated, and many useful packets will be discarded; if the probability is too small, the congestion level will be underestimated, and some redundant packets will be retained. It is difficult to adapt to various network loads with a fixed maximum discarding probability.

To adaptively adjust maximum discarding probability, this paper presents the RLAQM algorithm based on Q-learning, which is highly practical and low in complexity. Specifically, a Q-learning controller was added to the original RED to adjust maximum discarding probability as per network congestion level, making the algorithm less sensitive to parameter setting and more adaptive to networks with different loads.

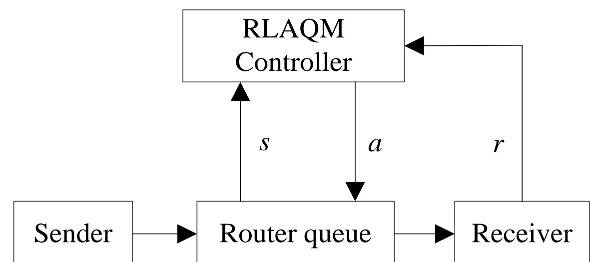


Figure 3. The block diagram of RLAQM controller

As shown in Figure 3, the learning process of RLAQM controller is defined by {S, A, r}, where S is the set of network states s (avg, Δavg) (avg is the average queue length of the buffer; Δavg is the change of the average queue length), A is the set of maximum discarding probabilities max_p, and r is the return value.

The RLAQM controller is embedded with the Q value table of network state-action pairs. Taking the current network state $s(avg, \Delta avg)$ as the input, the learning unit of the controller outputs the corresponding maximum discarding probability max_p for the action.

In each round of learning, the learning unit collects the current network state s and selects action a, i.e. the maximum discarding probability max_p of RED algorithm, according to the Q value table and action selection strategy. Then, the network state changes under the adjusted discarding probability from $s(avg, \Delta avg) \rightarrow s'(avg', \Delta avg')$, and calculates the return value r on action $a(max_p)$. After that, the Q value of the current state-action pair (s, a) is updated, completing this round of learning. This process is repeated until the Q value table is optimized.

The running speed of the algorithm is partly dependent on the number of network states. Hence, this number should be controlled within a reasonable range. To reduce algorithm complexity, the state $s(avg, \Delta avg)$ in the Q-learning controller was quantified to 14×10 levels. That is, the state set of learning units can be expressed as $S=\{s_{i,j}\}=\{avg_i, \Delta avg_j\}$, i=1,2,3…14, j=1,2,3…10. Similarly, the action $a(max_p)$ was divided into 14 levels, creating an action set of learning units $A=\{a_n\}$, n=1,2,3…14max_{p0}, where max_{p0} is the maximum discarding probability of RED algorithm.

The objective of the RLAQM controller is to find the optimal maximum discarding probability max_p that maximizes the network throughput while minimizing network delay. The objective function can be defined as a utility function [24]:

$$Utility = \delta \log T_{ave} - \eta \log D_{ave} \tag{11}$$

where, δ and η are the weights of throughput and delay, respectively; T_{ave} is the average network throughput in the learning cycle; D_{ave} is the average system delay in the learning cycle.

During the learning, each learning unit of RLAQM controller measures the algorithm effect by the return value r of state-action pair (s, a). The return value can be defined as the difference between successive utility values:

$$r = Utility_k - Utility_{k-1} = \delta \log \frac{T_{ave\ k}}{T_{ave\ k-1}} + \eta \log \frac{D_{ave\ k-1}}{D_{ave\ k}} \tag{12}$$

As shown in formula (12), the return value increases with the utility value. The function of the return value aims to maximize network throughput and minimize network delay. In this function, Q-learning strategy is adopted to iteratively converge to high throughput and low delay.

In each round, the learning starts once a new data packet arrives in the network. The specific flow of the RLAQM algorithm is as follows:

Step 1. Parameter initialization

The Q value table was initialized as zero; the discount factor γ, initial learning rate α_0, and initial exploration probability ε_0 were configured. The exploration probability is mainly used for strategy update, that is, the iterative optimization of the strategy. This requires the algorithm to traverse the strategy space in an efficient manner. Hence, an epsilon greedy strategy was adopted for action selection. In each iteration, the agent executes the action with the highest Q value in the current state at the probability of (1-ε), and selects one of the remaining actions at the probability of ε.

Step 2. Obtaining the current network state

The current state $s(avg, \Delta avg)$ was calculated after the arrival of a new packet.

Step 3. Selecting actions for execution

By the epsilon greedy strategy, an action was selected for execution, according to the $Q_t(s, a)$ that corresponds to each action of $s(avg, \Delta avg)$ in the current state. In other words, the maximum discarding probability max_p was selected under the current network state, and the executed actions were cached for further use when the Q value is updated.

Step 4. Obtaining the return value r

The router discards packets according to the new discarding probability function, and derives the return value r of the state-action pair (s, a) by formula (11).

Step 5. Predicting the next network state and updating the Q value

According to the current state and the selected action, the optimal $\max_{a \in A} Q^*(s, a)$ of the next network state s' was obtained, and $Q_t(s, a)$ was updated by formula (10).

Step 6. Parameter update

The learning rate α and the exploration probability ε were updated after each iteration to ensure the convergence of the algorithm. The updating rule was gradually reduced to zero, following the negative exponential law.

Step 7. Termination

The convergence or nonconvergence of Q value was judged. If the value does not converge, the Q value table was taken as the initial value, and the next iteration was executed from Step 2. If the value converges, the learning process was terminated, and the current Q value table was used to select the actions according to states.

4. SIMULATION ANALYSIS

The performance of the proposed RLAQM algorithm was verified through simulations on NS2 simulator. The network topology for simulation is described in Figure 4, where S1~Sn are n senders, D1~Dn are n receivers, and R1~R2 are two routers [25]. The bandwidth and delay between each sender and R1, and between each receiver and R2, were set to 20 Mbps and 20 ms, respectively

For comparison, RED algorithm and RLAQM algorithm were simulated under single load and variable load, and their queue lengths, throughputs, delays, and discarding probabilities were measured. During the simulation, the common parameters of the two algorithms were configured as follows:

The lower bound of average queue length min_{th} is 24 packets, the upper bound of average queue length max_{th} is 72 packets, the weight ω_q is 0.002, the buffer size is 120 packets the maximum discarding probability max_p of RED algorithm is 0.1, the initial maximum discarding probability max_{p0} of RLAQM algorithm is 0.1, and the initial values of α, γ, and ε are 0.01, 0.8, and 0.5, respectively. Both algorithms were executed on the links between the two routers, while the Drop-Tail was executed on the other links [26]. The application layer is based on TCP and file transfer protocol (FTP).

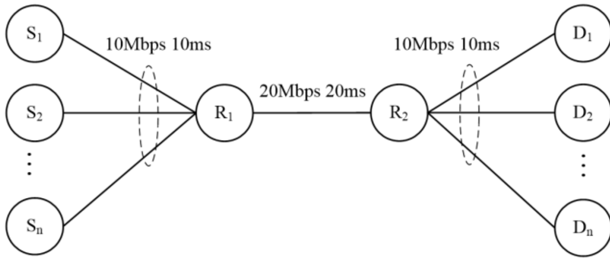


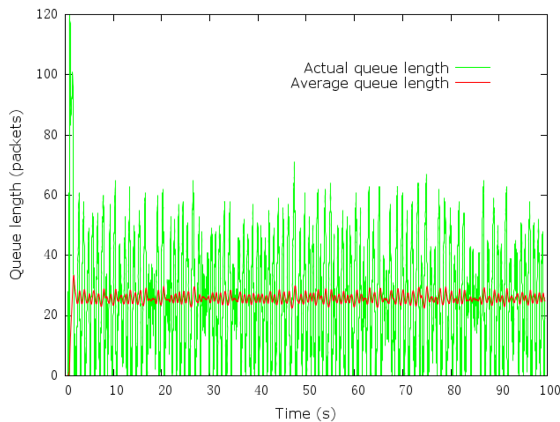
Figure 4. The topology of the simulation network

4.1 Single load simulations

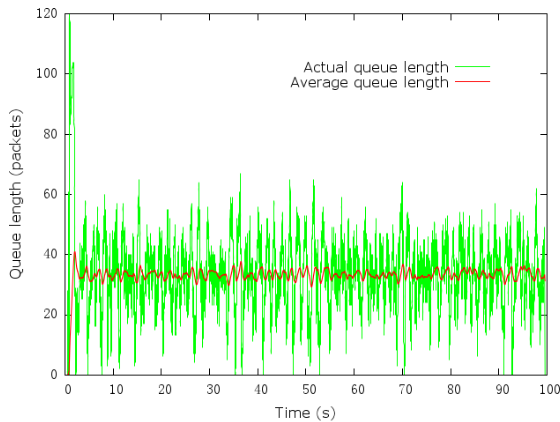
4.1.1 Light load

During single load simulations, RED algorithm and RLAQM algorithm were firstly simulated under light load with the number of network connections fixed at 16 and the simulation time at 100s.

Figure 5 compares the queue lengths of the two algorithms: the average queue length of RED algorithm fell between 22 and 30, while that of RLAQM algorithm fell between 30 and 38. RLAQM had a longer average queue length than RED. This is because RED has a short average queue length under light network load, and faces light network congestion; RLAQM increases the average queue length and improves network throughput, by reducing the maximum discarding probability and thus the discarding probability.



(a) RED algorithm



(b) RLAQM algorithm

Figure 5. The comparison of queue length under light load

Table 1 compares the throughputs, delays, and packet loss rate of RED algorithm and RLAQM algorithm. Obviously, RLAQM achieved higher average throughput than RED at the cost of a slightly longer delay, which meets the theoretical design. Overall, RLAQM increased network throughput by 0.20%, extended the delay by 2.6%, and reduced the packet loss rate by 8.6% from the levels of RED.

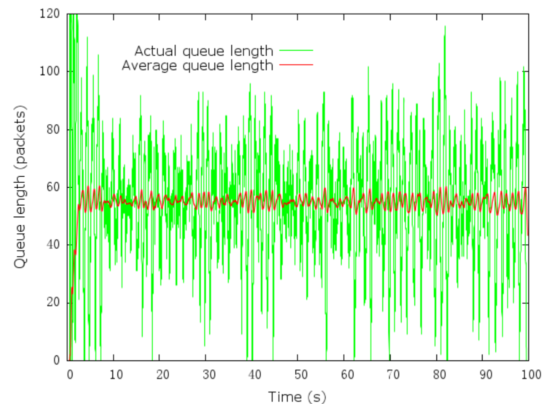
Table 1. The performance comparison under light load

Performance / Algorithm	Throughput (Kbps)	Delay (ms)	Packet loss rate (%)
RED	19,900.76	54.05	0.35
RLAQM	19,941.08	55.48	0.32

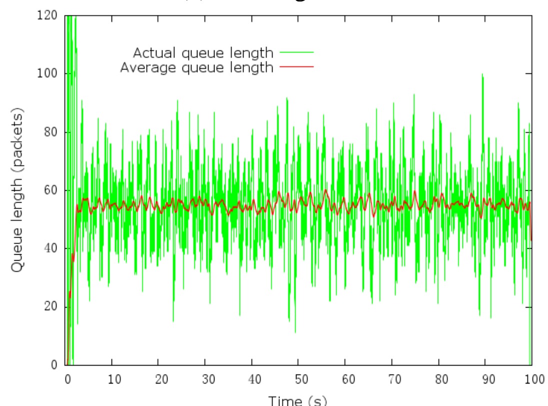
4.1.2 Moderate load

Next, RED algorithm and RLAQM algorithm were simulated under moderate load with the number of network connections fixed at 64 and the simulation time at 100 s.

Figure 6 compares the queue lengths of the two algorithms: the average queue lengths of both algorithms ranged between 50 and 60. The average queue length of RLAQM was not significantly different from that of RED. The main reason is that, under moderate network load, the congestion remains on the moderate level. In this case, RLAQM does not need to adjust the discarding probability from the level of RED. Thus, the maximum discarding probabilities of the two algorithms are similar, which greatly limits their difference in network performance.



(a) RED algorithm



(b) RLAQM algorithm

Figure 6. The comparison of queue length under moderate load

Table 2 compares the throughputs, delays, and packet loss rates of RED algorithm and RLAQM algorithm. It can be seen that, RLAQM achieved basically the same performance as RED, which meets the theoretical design. Overall, RLAQM

increased network throughput by 0.0009%, shortened the delay by 0.23%, and elevated the packet loss rate by 1.1% from the levels of RED.

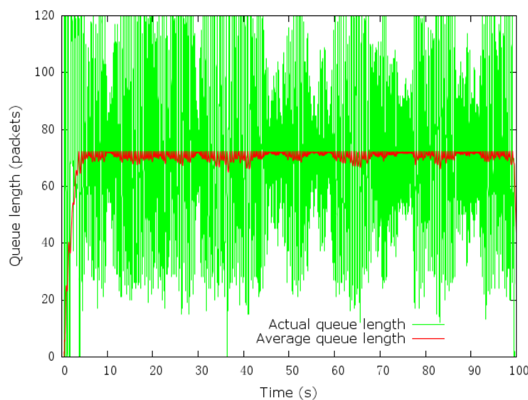
Table 2. The performance comparison under moderate load

Performance / Algorithm	Throughput (Kbps)	Delay (ms)	Packet loss rate (%)
RED	19,961.14	64.96	4.64
RLAQM	19960.96	64.81	4.69

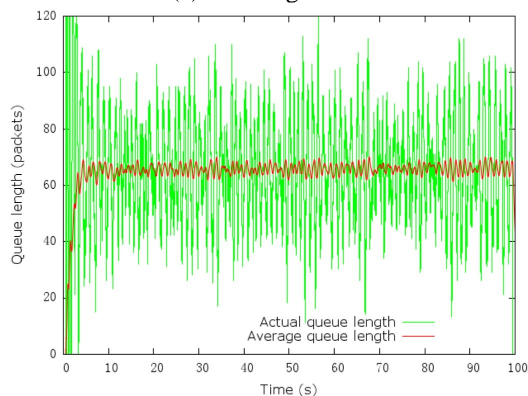
4.1.3 Heavy load

Further, RED algorithm and RLAQM algorithm were simulated under heavy load with the number of network connections fixed at 128 and the simulation time at 100 s.

Figure 7 compares the queue lengths of the two algorithms: the average queue length of RED algorithm changed from 68 to 72, while that of RLAQM algorithm varied from 62 to 68. RLAQM had a shorter average queue length than RED. This is attributable to the following facts: Under a heavy network load, RED has a long average queue length and the network is severely congested; RLAQM packet loss rate by increasing the maximum value, thereby reducing the average queue length and network delay.



(a) RED algorithm



(b) RLAQM algorithm

Figure 7. The comparison of queue length under heavy load

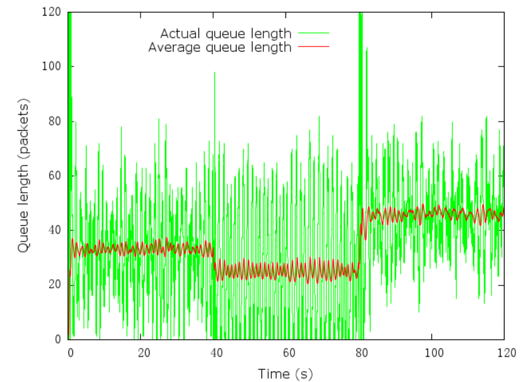
Table 3 compares the throughputs, delays, and packet loss rates of RED algorithm and RLAQM algorithm. It can be seen that, RLAQM achieved slightly higher average throughput and shorter network delay than RED, which meets the theoretical design. Overall, RLAQM increased network throughput by 0.005%, shortened the delay by 1.6%, and reduced the packet loss rate by 12% from the levels of RED.

Table 3. The performance comparison under heavy load

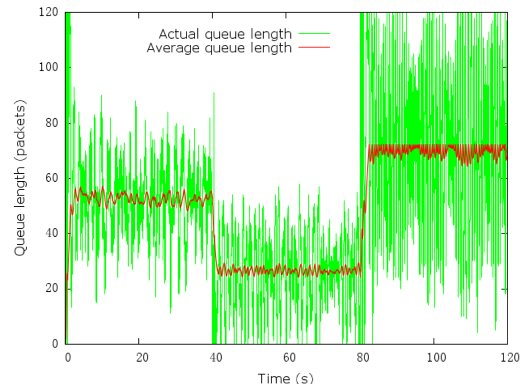
Performance / Algorithm	Throughput (Kbps)	Delay (ms)	Packet loss rate (%)
RED	19,972.69	72.02	13.07
RLAQM	19,973.71	70.84	11.54

4.2 Variable load simulations

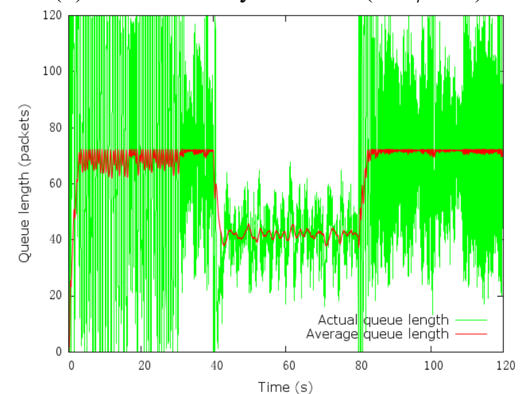
RED algorithm and RLAQM algorithm were also simulated under variable load, with the number of connections changing between 64, 16, and 128 every 40 s, and the simulation time of 120 s.



(a) Radical early detection (max_p=0.3)



(b) Moderate early detection (max_p=0.1)



(c) Conservative early detection (max_p=0.01)

Figure 8. The variation of queue length in RED algorithm

The stability of queue length is an important indicator of network performance. To disclose the impact of maximum discarding probability on queue length under variable load, the queue lengths of RED under different early detection performance (i.e. maximum discarding probabilities) were captured and plotted into Figure 8.

Under radical early detection (max_p=0.3) (Figure 8(a)), the queue length of RED was relatively stable in the first 40s under 64 connections and the last 40 s under 128 connections. However, the queue was almost empty in the 16 s~40 s under 16 connections. Hence, RED has a very low efficiency under radical early detection.

Under conservative early detection (max_p=0.01) (Figure 8(c)), the queue length of RED was stable only under 16 connections, and almost saturated under 64 and 128 connections. In the latter two scenarios, RED queue oscillated continuously between continuous packet loss and subsequent low utilization.

Under moderate early detection (max_p=0.1) (Figure 8(b)), the queue length of RED was stable under 64 connection, but empty or almost saturated under 16 and 128 connections. In the latter two scenarios, RED queue continued to oscillate.

Figure 9 presents the variation of queue length of RLAQM under the same simulation environment. It can be seemed that the queue of RLAQM was adjusted adaptively according to the changing network load, and the overall queue length remained relatively stable.

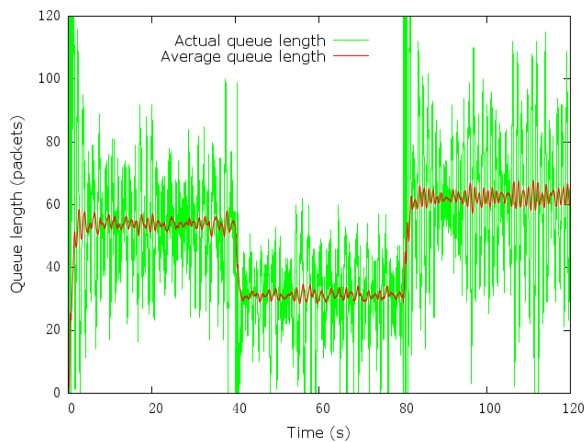


Figure 9. The variation of queue length in RLAQM algorithm

In addition, the throughputs, delays, and packet loss rates of RED and RLAQM algorithms were tested under the initial maximum discarding probability of 0.1. As shown in Table 4, RLAQM outshined RED under each load in the dynamic network environment.

Table 4. The comparison of dynamic network performance

Performance / Algorithm	Throughput (Kbps)	Delay (ms)	Packet loss rate (%)
RED	19,852.68	64.75	6.34
RLAQM	19,883.24	63.46	5.79

5. CONCLUSIONS

To reduce the sensitivity of RED to parameter setting and improve network performance, this paper improves RED into RLAQM, an algorithm that adaptively updates its parameters as per network conditions. The control strategy of network performance was optimized through iterative learning: the maximum discarding probability is adjusted adaptively to avoid network congestion and improve network performance. Simulation results show that RLAQM can maintain the queue stable in dynamic network environment, reduce network delay, and improve network throughput. The future research will introduce explicit congestion notification (ECN) to RLAQM, and further improve the action adjustment accuracy of RLAQM through fuzzy Q-learning.

ACKNOWLEDGMENT

This work was supported by High-level Talent Project of Xiamen University of Technology (Grant No.: YKJ17021R); Scientific Research Climbing Project of Xiamen University of Technology (Grant No.: XPDKT19006).

REFERENCES

[1] Molnár, S., Vágó, L. (2020). Networking in the absence of congestion control. Stochastic Models, 36(3): 401-427. https://doi.org/10.1080/15326349.2020.1742160

[2] Zheng, W., Li, Y., Jing, X., Liu, S. (2020). Adaptive Finite-Time Congestion Control for Uncertain TCP/AQM Network with Unknown Hysteresis. Complexity, 4138390. https://doi.org/10.1155/2020/4138390

[3] Xu, Q., Ma, G., Ding, K., Xu, B. (2020). An Adaptive Active Queue Management Based on Model Predictive Control. IEEE Access, 8: 174489-174494. https://doi.org/10.1109/ACCESS.2020.3025377

[4] Floyd, S., Fall, K. (2002). Promoting the use of End-to-End network troubleshooting in the Internet. IEEE/ACM Transaction Networking, 4: 458-472. https://doi.org/10.1109/90.793002

[5] Feng, C.W., Huang, L.F., Xu, C., Chang, Y.C. (2015). Congestion control scheme performance analysis based on nonlinear RED. IEEE Systems Journal, 11(4): 2247-2254. https://doi.org/10.1109/JSYST.2014.2375314

[6] Alkharasani, A.M., Othman, M., Abdullah, A., Lun, K.Y. (2017). An improved quality-of-service performance using RED's active queue management flow control in classifying networks. IEEE Access, 5: 24467-24478. https://doi.org/10.1109/ACCESS.2017.2767071

[7] Patel, S., Bhatnagar, S. (2017). Adaptive mean queue size and its rate of change: queue management with random dropping. Telecommunication Systems, 65(2): 281-295. https://doi.org/10.1007/S11235-016-0229-4

[8] Patel, S. (2019). A new modified dropping function for congested AQM networks. Wireless Personal Communications, 104(1): 37-55. https://doi.org/10.1007/S11277-018-6007-8

[9] Koo, J., Song, B., Chung, K., Lee, H., Kahng, H. (2001). MRED: a new approach to random early detection. In Proceedings 15th International Conference on Information Networking, pp. 347-352. https://doi.org/10.1109/ICOIN.2001.905450

[10] Athuraliya, S., Li, V.H., Low, S.H., Yin, Q. (2001). REM: Active queue management. In Teletraffic Science and Engineering, 4: 817-828. https://doi.org/10.1016/S1388-3437(01)80172-4

[11] Kunniyur, S.S., Srikant, R. (2004). An adaptive virtual queue (AVQ) algorithm for active queue management. IEEE/ACM Transactions on Networking, 12(2): 286-299. https://doi.org/10.1109/TNET.2004.826291

[12] Novak, J.H., Kasera, S.K. (2017). Auto-tuning active queue management. In 2017 9th International Conference on Communication Systems and Networks (COMSNETS), pp. 136-143. https://doi.org/10.1109/COMSNETS.2017.7945369

[13] Bisoy, S.K., Pandey, P.K., Pati, B. (2017). Design of an active queue management technique based on neural networks for congestion control. In 2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), pp. 1-6. https://doi.org/10.1109/ANTS.2017.8384104

[14] Chrost, L., Chydzinski, A. (2016). On the deterministic approach to active queue management. Telecommunication Systems, 63(1): 27-44. https://doi.org/10.1007/S11235-015-9969-9

[15] Hollot, C.V., Misra, V., Towsley, D., Gong, W. (2002). Analysis and design of controllers for AQM routers supporting TCP flows. IEEE Transactions on Automatic Control, 47(6): 945-959. https://doi.org/10.1109/TAC.2002.1008360

[16] Sun, J., Ko, K.T., Chen, G., Chan, S., Zukerman, M. (2003). PD-RED: to improve the performance of RED. IEEE Communications Letters, 7(8): 406-408. https://doi.org/10.1109/LCOMM.2003.815653

[17] Wang, P., Zhu, C., Yang, X. (2018). A novel AQM algorithm based on feedforward model predictive control. International Journal of Communication Systems, 31(12): e3711. https://doi.org/10.1002/DAC.3711

[18] Kahe, G., Jahangir, A.H. (2019). A self-tuning controller for queuing delay regulation in TCP/AQM networks. Telecommunication Systems, 71(2): 215-229. https://doi.org/10.1007/S11235-018-0526-1

[19] Bisoy, S.K., Pattnaik, P.K., Pati, B., Panigrahi, C.R. (2018). Design and analysis of a stable AQM controller for network congestion control. International Journal of Communication Networks and Distributed Systems, 20(2): 143-167. https://doi.org/10.1504/IJCNDS.2018.10010375

[20] Adams, R. (2012). Active queue management: A survey. IEEE Communications Surveys & Tutorials, 15(3): 1425-1476. https://doi.org/10.1109/SURV.2012.082212.00018.

[21] Su, Y., Huang, L., Feng, C. (2018). QRED: A Q-learning-based active queue management scheme. Journal of Internet Technology, 19(4): 1169-1178. https://doi.org/10.3966/160792642018081904019

[22] Li, X., Serlin, Z., Yang, G., Belta, C. (2019). A formal methods approach to interpretable reinforcement learning for robotic planning. Science Robotics, 4(37). https://doi.org/10.1126/SCIROBOTICS.AAY6276

[23] Wang, Y., Cao, S., Ren, H., Li, J., Ye, K., Xu, C., Chen, X. (2020). Towards cost-effective service migration in mobile edge: A Q-learning approach. Journal of Parallel and Distributed Computing, 146: 175-188. https://doi.org/10.1016/J.JPDC.2020.08.008

[24] Li, W., Zhou, F., Chowdhury, K.R., Meleis, W. (2018). QTCP: Adaptive congestion control with reinforcement learning. IEEE Transactions on Network Science and Engineering, 6(3): 445-458. https://doi.org/10.1109/TNSE.2018.2835758

[25] Patel, S. (2020). Nonlinear performance evaluation model for throughput of AQM scheme using full factorial design approach. International Journal of Communication Systems, 33(8): e4357. https://doi.org/10.1002/DAC.4357

[26] Kumhar, D., Kewat, A. (2020). QRED: An enhancement approach for congestion control in network communications. International Journal of Information Technology, 1-7. https://doi.org/10.1007/S41870-020-00538-1