QRED: A Q-Learning-based Active Queue Management Scheme

Yuhan Su[1], Lianfen Huang[1], Chenwei Feng[2]

[1] Department of Communication Engineering, Xiamen University, China
[2] Department of Communication Engineering, Xiamen University of Technology, China
suyuhan066@foxmail.com, lfhuang@xmu.edu.cn, cwfeng@xmut.edu.cn

Abstract

The Active Queue Management (AQM) algorithm is one of most important research fields in network congestion control. To adjust the maximum dropping probability (*max*$_p$) according to the network situation the *max*$_p$ calculation based on the RED algorithm is improved using the Q-learning algorithm, and a new algorithm, known as QRED (Q-learning RED), is proposed. The self-adaptive adjustment for the *max*$_p$ is achieved using the QRED algorithm and the queue length stability in a dynamic network environment is realized. In addition, the QRED algorithm not only avoids the sensitivity of the RED algorithm parameters, but also adapts the packet loss rate according to the specific network service type. Results based on the NS2 simulation show that the QRED algorithm has better stability in complex network environments, and hence, are superior to the RED active queue management algorithm.

Keywords: Network congestion control, Active queue management, Q-learning, RED algorithm, 5G

1 Introduction

The increasing number of Internet users brings growing congestion to the Internet. In order to solve the network congestion problem [1-5], it is not enough to rely solely on the TCP congestion control mechanism [6] provided by the source node, so that the network itself is also involved in congestion control. Congestion control based on the intermediate node includes two parts: queue scheduling and queue management [7]. The former is used to solve the data network bandwidth distribution problem, focusing on network fairness. The latter aims at maintaining routing squadron stability by choosing a certain packet drop probability based on the route circumstance. Queue management algorithms can be divided into two categories: passive queue management (PQM) and active queue management (AQM). The traditional Drop-Tail algorithm is based on the PQM algorithm mechanism, in which the packet drop probability is controlled by setting a maximum value for the queue. The Drop-Tail, however, may cause several problems, such as network deadlock, full queue, global synchronization and delay due to a continuously full queue. In 1993 Floyd proposed the famous RED [8] congestion control mechanism which effectively improved the Drop-Tail. The RED algorithm evaluates the changes in network congestion by calculating the average queue. When the average queue length increases rapidly, it will increase the labeled packet drop probability to inform the sender to appropriately reduce the transmission rate, to ease network congestion. However, for the reason that RED is sensitive to the parameter settings, a sudden increase in the packet drop probability up to 1 may occur when the average queue length is greater than the queue length upper limit. Subsequently, the Floyd and Feng groups proposed the GentleRED [9] and ARED [10], [11] algorithms, respectively. The former improves the RED algorithm design patterns for packet drop probability calculation. When the average queue length is greater than the upper queue threshold, the packet drop probability linearly increases to 1 with the increase in queue length. The latter introduces increasing and decreasing coefficients, to adaptively adjust the packet drop probability according to the degree of congestion. Although GentleRED improves RED algorithm performance in some ways, the sensitivity to parameters defect still exists in GentleRED, and consequently the stability and robustness is poor when encountering sudden flow. With more complex parameter settings the ARED is more sensitive to the parameters.

1.1 Related Work

To our knowledge, relatively few researches have been conducted on congestion control based on the learning algorithm. In [12] a new algorithm, known as DEEP BLUE, is proposed to improve the conventional BLUE algorithm [13], in which the fuzzy theory and reinforcement learning theory are applied to the congestion control algorithm. Using the fuzzy Q-Learning algorithm to adaptively select the BLUE

[*]Corresponding Author: Chenwei Feng; E-mail: cwfeng@xmut.edu.cn
DOI: 10.3966/160792642018081904019

algorithm parameters according to the network circumstances, DEEP BLUE solves the lack of adaptability problem in the BLUE algorithm. It also improves the convergence speed and algorithm precision. Integrating the reinforcement learning idea and gradient descent method, the AQM algorithm is proposed in [14]. The AQM uses the link rate matching and queue length as the optimization objective. It was shown that through adaptive update step size adjustment and packet drop probability direction, the queue length can quickly converge to the target value with a smaller jitter. [15] combined with reinforcement learning and neural networks to solve the congestion problem of Broadband Integrated Services Digital Networks (BISDN). [16] proposed the AHC (Adaptive Heuristic Critic) algorithm based on reinforcement learning, using the idea of Temporal-Difference learning to learn the original experience for congestion control, without considering the dynamic model of the environment. [17] solved the dynamic high-speed network congestion problem using multi-agent reinforcement learning methods.

Many scholars have improved the AQM algorithm in recent years. An active queue management algorithm based on the fuzzy neural PID (FNPID) algorithm is presented in [18]. The fuzzy logic part is used to calculate the learning rate, while the neural network PID calculates the packet drop probability using the weighted momentum gradient learning algorithm. [19] presented an improved algorithm called FlowRED based on this type of protocol. The algorithm is based on the original RED algorithm for increased UDP packet drop probability to improve the algorithm fairness. In [20] an improved gCHOKe algorithm, sgCHOKe (Sampling based gCHOKe) is proposed, through analyzing the non-response flows hit low responsiveness, which samples several packets from the queue and compares them with the packet arrival and employs a new packet-drop mechanism.

The above methods improved the congestion control algorithms in some respects but did not obviously improve other network performances, such as throughput, delay, etc., and cannot adapt to different network service type transmission. Some algorithms increased the complexity. There are also some congestion control algorithms based on learning algorithms such as [15-17], did not consider the learning convergence problem.

The present paper proposes a new algorithm, known as QRED (Q-learning RED), based on the intelligent processing ability characteristics of the Q-learning algorithm. The QRED algorithm optimizes the maximum dropping probability (max_p) calculation method by designing a learning controller based on the Q-learning algorithm [21], [22] to adaptively select max_p, it partly eliminates the parameter sensitivity in the RED algorithm and improves the overall network performance. Moreover, the QRED algorithm can adaptive adjustment parameters according to the different types of network service transmission, so it can support the network slicing scene [23-26] in 5G network.

1.2 Our Contributions

In this paper, the problem of RED algorithm is deeply studied, especially the parameter sensitive problem. Aiming at this problem, this paper proposes A Q-Learning-based Active Queue Management Scheme: QRED. This algorithm adaptively learn RED parameter max_p in the current network scene through the Q-learning algorithm. And the RED algorithm parameters are selected according to the optimal value after learning iteration. In addition, we propose an actions selection strategy based on the RED algorithm: *G-Policy*. This policy can speed up the *Q-value* convergence and improve learning efficiency. In general, The QRED algorithm eliminates the parameter sensitive problem, improves the throughput of the system, and reduces the end-to-end transmission delay.

The paper is organized as follows. Section 1 covers the related work. In Section 2, we mainly introduce the RED Algorithm and analysis its existing problems. In addition, we propose an active queue management algorithm based on Q-learning. Performance evaluation has been conducted in Section 3. Finally, the paper is concluded in Section 4.

2 QRED Algorithm

2.1 RED Algorithm

The RED algorithm was proposed by Floyd Van and Jacobson Sally to realize router congestion control. A router using the RED algorithm will mark the data packets that arrive at the router. It will send packets that exceed its buffering zone to the sender so that the sending end can reduce the transmission window to avoid congestion. The algorithm marks the data packets with certain randomness to avoid congestion in the early stage. This approach is called the random early detection (RED) algorithm.

The RED scheme drops packets with a certain probability by computing the average queue length (*avg*) to notify traffic sources about the early stages of network congestion. The average queue length is calculated as the result of the exponentially weighted moving average (EWMA) [27], which really acts as a low-pass filter that smoothes out the burstiness of the instantaneous queue length [28] to provide a more stable measure. The degree of smoothing is determined by weighting factor w_q. In addition, the average queue length is expressed as：

$$avg = \begin{cases} (1-w_q)\times avg + w_q \times q_1 \times q, & if\ q<0 \\ (1-w_q)\times avg, & otherwies \end{cases}, \quad \textbf{(1)}$$

where q is the current queue length, $w_q \in[0,1]$ is the weight equivalent to the low pass filter time constant, m is the number estimated by some function which is dependent on idle time of the router. The value of w_q is very important. If it is set too large temporary congestion cannot be filtered out. On the contrary, if it is set too small the *avg* response to changes in the actual queue length will be too slow for the router to detect the initial congestion stages.

In addition to EWMA weight w_q, RED has three more parameters, i.e., minimum threshold min_{th}, maximum threshold max_{th}, and the maximum dropping probability max_p at max_{th}. If the average queue length is below min_{th}, RED drops no packets. However, if the average queue length increases above min_{th} but is below max_{th}, RED drops incoming packets with a probability proportional to the average queue length linearly. When the average queue length exceeds max_{th}, all the arriving packets are dropped. The dropping probability p_b can be calculated using the algorithm's internal data variables and the average queue obtained by reading the buffer, i.e.,

$$p_b=\begin{cases}0, & \mid avg\in[0,min_{th}]\\ \dfrac{max_p(avg-min_{th})}{max_{th}-min_{th}}, & \mid avg\in[min_{th},max_{th})\\ 1, & \mid avg\in[max_{th},+\infty]\end{cases}. \quad \textbf{(2)}$$

The packet number between the first and second packet drop probability settings should not be too large so that the packet drop probability should vary with the number of packets between the two packet drop probability settings,

$$p_a=\frac{p_b}{1-count\times p_b}, \quad \textbf{(3)}$$

where *count* is the number of unmarked packets between the two packet drop probability settings and p_a is the packet drop probability. The packet drop probability should be a function of *avg*, since *avg* can reflect the degree of congestion. Based on the above analysis one can plot the curve of p_b varying with *avg* as Figure 1.

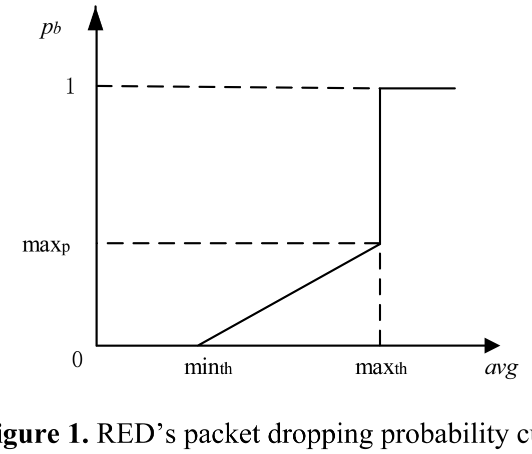


Figure 1. RED's packet dropping probability curve

The RED algorithm solved the global synchronization problem in the Drop-Tail algorithm, but there were still several problems remaining which include:

(1) Parameter sensitivity [29]: In order to achieve good packet loss rate, time delay and throughput performances under various network loads it is necessary to configure appropriate parameters.

(2) Fairness issue [30]: The RED algorithm did not effectively solve the fair competition problem for the network resources between the TCP and UDP flows in the transport layer.

2.2 Q-Learning Algorithm

As one of the main reinforcement learning algorithms [34], Q-learning is a model-free learning method which provides the intelligent system with the ability to select the optimal action according to the action sequences from experience in the Markov environment [32]. A key assumption of Q-learning is that the interaction between the agents and the environment can be treated as a Markov decision process (MDP), i.e., the current state and action of the agent will determine the state transfer probability distribution and the next state with an immediate reward. The goal of Q-learning is to find a policy that can maximize the reward.

The *Q-value* is an important parameter in Q-learning. It is defined as the sum of rewards for executing the current related actions and those to be performed subsequently in accordance with a certain strategy. A given state s and action a correspond to a given *Q-value Q*(s,a). *Q-value* is used in the learning process to select the action. If the subsequent actions are performed according to the optimal polices the corresponding *Q-value* is referred to as the optimal *Q* value *Q**,

$$Q^*(s,a)=r(s,a)+\gamma\sum T(s,a,s')maxQ^*(s',a'), \quad \textbf{(4)}$$

where $T(s, a, s')$ represents the transfer probability from state s to state s' via action a, $r(s,a)$ represents the reward for executing action a from state s, $\gamma\in(0,1)$ is the discount factor, which indicates the degree of farsightedness. If the γ value is small, the system pays attention to only the recent actions. If γ is large the actions during a relatively long period of time are involved. An agent learning process can be viewed as selecting an action from a random state using a strategy. The value of $Q(s,a)$ is updated according to

$$Q_{t+1}(s,a)=(1-a)+Q_t(s,a)+a[\gamma max(s',a')], \quad \textbf{(5)}$$

where $\alpha\in(0,1)$ is the learning factor used to control the speed of learning: the greater the value of α, the faster the convergence speed. After performing the selected action the agent observes the new state and the reward obtained, and then updates the *Q-value* of the state and action based on the maximum *Q-value* of the new state. In this way the agent continually updates the action according to the new state until it arrives at the terminal state with an optimal *Q-value* Q^*. A block diagram of the Q-learning algorithm is given in Figure 2.

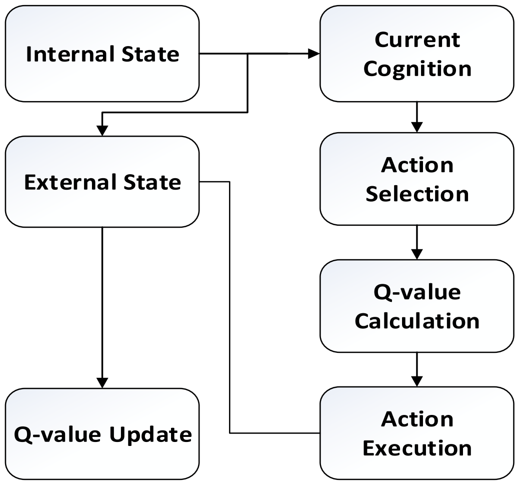


Figure 2. Q-learning process diagram

2.3 QRED Algorithm

In the RED algorithm the relationship between the packet drop probability and the average queue length is linear, so that a relatively high packet drop probability may occur near the minimum threshold when the network is not in a serious congestion state. However, when the network is in a serious congestion state near the maximum threshold and there is a need for higher packet drop probability to ease network congestion, the parameter max_p is the maximum drop probability before the packet dropping probability jumps to 1. In this stage too large a value for max_p may lead to a heavy congestion indication, serious grouped packet dropping, and hence decrease in the network throughput and buffer occupation. Conversely, too small a value for max_p will result in a light congestion indicator and buffer overflow, forming the Drop-Tail mode. For these reasons it is difficult to find an appropriate parameter max_p that can adapt to a variety of network environments and deal with a burst in traffic, so it is necessary to improve the packet drop probability function.

In this paper, Q-learning algorithm is used to solve the parameter sensitivity problem of RED algorithm, considering that Q-learning has model-independent characteristics, it can obtain the optimal system strategy under the condition that the environment transfer function and the return expectation cannot be get, conform the actual situation of network congestion. The basic idea of the QRED algorithm is: by introducing an offline Q-learning controller into the RED algorithm, the max_p is adjusted according to the network congestion, so as to achieve reducing the RED algorithm parameter sensitivity. The system model performance goal is to maximize throughput or minimize the delay. The system model can be established as follows:

$$Max\ C = \sum_{t=1}^{T} C_t, \tag{6}$$

$$Min\ d = \sum_{t=1}^{T} D_t, \tag{7}$$

where T is the total simulation time of the system. C_t represents the throughput at t. D_t is the transmission delay at t.

The learning process of Q-learning algorithm will affect the system delay, so we choose the offline learning, it means that the learning block is set out of the system. The flow chart of the QRED algorithm is shown in Figure 3.

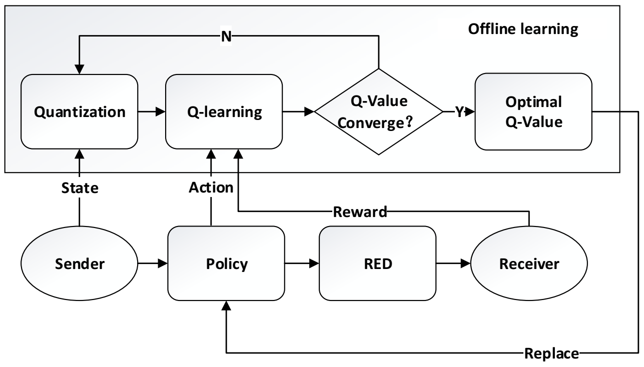


Figure 3. QRED algorithm

In the QRED algorithm, the learning process is defined by the triplet $\{s, a, R\}$, where s is the set of states, the set of average queue lengths (avg), a is the set of actions, that is, the set of max_p, R is the reward function. Q-learning controller to perceive the current network, its corresponding agent, states, actions, reward function, policy is defined as follows:

Agent. Take the Q-learning controller as an agent, the agent store a table for the *Q-value* based on the state-action pair of the network, as shown in Table 1.

Table 1. Q-value

Action / State	a_1	a_2	...	a_{13}	a_{14}
s_1	$Q(s_1,a_1)$	$Q(s_1,a_2)$	...	$Q(s_1,a_{13})$	$Q(s_1,a_{14})$
s_2	$Q(s_2,a_1)$	$Q(s_2,a_2)$	...	$Q(s_2,a_{13})$	$Q(s_2,a_{14})$
...	...	...	...	...	...
s_{13}	$Q(s_{13},a_1)$	$Q(s_{13},a_2)$	...	$Q(s_{13},a_{13})$	$Q(s_{13},a_{14})$
s_{14}	$Q(s_{14},a_1)$	$Q(s_{14},a_2)$	...	$Q(s_{14},a_{13})$	$Q(s_{14},a_{14})$

The input the controller is the current network state, i.e., the average queue length (avg). The output is the action adaptively adjusted according to the action selection strategy. The router then processes the data packets by discarding some of them according to the adjusted drop probability function. The max_p value reflects the degree of network congestion, i.e., the more serious the congestion the larger max_p value and vice versa. The curve for the packet drop probability vs the average queue length is shown in Figure 4.

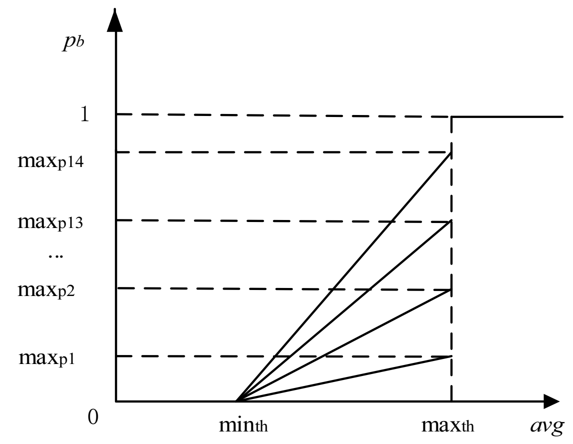


Figure 4. QRED's packet dropping probability curve

States. In order to reduce the system complexity, the state $s(avg)$ in the Q-learning controller is quantized into 14 classes, thus the learning unit state set is given by $s = \{s_i\}$, i = 1, 2, 3... 14. The state is set to 14 ×14 groups, as shown in Table 2.

Table 2. 14 States groups

No.	State	No.	State
s_1	$0.3max_{th} - 0.35max_{th}$	s_8	$0.65max_{th} - 0.7max_{th}$
s_2	$0.35max_{th} - 0.4max_{th}$	s_9	$0.7max_{th} - 0.75max_{th}$
s_3	$0.4max_{th} - 0.45max_{th}$	s_{10}	$0.75max_{th} - 0.8max_{th}$
s_4	$0.45max_{th} - 0.5max_{th}$	s_{11}	$0.8max_{th} - 0.85max_{th}$
s_5	$0.5max_{th} - 0.55max_{th}$	s_{12}	$0.85max_{th} - 0.9max_{th}$
s_6	$0.55max_{th} - 0.6max_{th}$	s_{13}	$0.9max_{th} - 0.95max_{th}$
s_7	$0.6max_{th} - 0.65max_{th}$	s_{14}	$0.95max_{th} - max_{th}$

Actions. Similarly, the action $a(max_p)$ is divided into 14 classes and the learning unit action set is $a=\{a_j\}, j$ = 1, 2, 3... 14. The action is also set to 14×14 groups, as shown in Table 3.

Table 3. 14 Actions groups

No.	Action	No.	Action
a_1	$0.1max_p$	a_8	$0.8max_p$
a_2	$0.2max_p$	a_9	$0.9max_p$
a_3	$0.3max_p$	a_{10}	max_p
a_4	$0.4max_p$	a_{11}	$1.5max_p$
a_5	$0.5max_p$	a_{12}	$2max_p$
a_6	$0.6max_p$	a_{13}	$2.5max_p$
a_7	$0.7max_p$	a_{14}	$3max_p$

Reward. The reward function can be defined by the following formula:

$$R = K_T \frac{C_{ave} - C_{min}}{C_{max} - C_{min}} + K_D \frac{D_{max} - D_{min}}{D_{max} - D_{ave}}, \qquad \textbf{(8)}$$

where C_{max} is the maximum value of system throughput during the learning cycle and C_{min} is the minimum value of system throughput during the learning cycle. C_{ave} is the average of the system throughput during the learning cycle. D_{max} is the maximum value of the system delay in the learning cycle, and D_{min} is the minimum value of the system delay in the learning cycle. D_{ave} is the average of the system delay during the learning cycle. K_T is the throughput weight, and K_D is the delay weight. The weight can be set according to the particular network service type. For example, if the uRLLC (Ultra-Reliable Low latency Communications) service [24] in the 5G network scene is transmitted, the K_D value can be set higher.

The reward function consider the overall network throughput performance and network delay factors, making the system throughput as large as possible, the delay as small as possible. In this reward function, the Q-learning strategy is chosen to iterate in the direction of high throughput and low delay.

Policy. In the agent, the choice of action in the state is actually a process of exploring the unknown. In this process, the agent cannot always choose the current action of maximum *Q-value*, to avoid falling into the local optimal, and cannot always choose the new action, ignoring the experience accumulated before. Therefore, Q-learning generally through some policy to ensure the balance between conservative and aggressive, the main methods include *ε-greedy* algorithm and Boltzmann algorithm [34].

Because we have already known the linear relationship between the *avg* and the max_p, the QRED algorithm can be set a new policy according to experience, *G-policy*, the *G-policy* can be expressed by the formula (9). Before the end of the study, the system selects the action with the largest *G-value* in the current state.

$$G(i,j) = \frac{1}{|i-j|+k}. \qquad \textbf{(9)}$$

Where i and j are, respectively, the ordinal numbers of the current state s group and the action under the current s, and $k \in (0,1)$ is the return coefficient. The value of k determines the state sensitivity to the action, i.e., the smaller the *k-value* the more sensitive the current state is to the current action. We set $m = |i-j|$, then:

$$G(i,j) = G(m) = \frac{1}{m+k}, \qquad \textbf{(10)}$$

$$G'(m) = \frac{1}{(m+k)^2} < 0, \qquad \textbf{(11)}$$

We can know from (10) (11) that $G(m)$ is a monotonically decreasing function with respect to m, so Equation (9) shows that the closer the values of i and j are, the larger the *G-value* is. As shown ($k = 0.1$) in Figure 5.

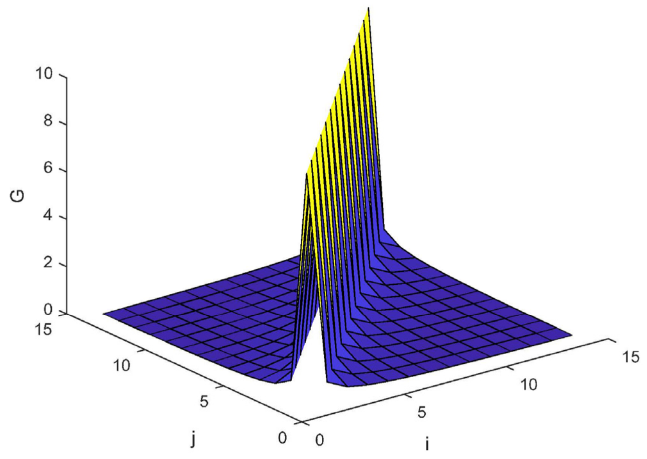


Figure 5. G-Policy's 3D curve

Because according to Table 2 and Table 3 we can see the mapping relationship of (i, j) and (avg, max_p), the *G-policy* indicates that, according to experience, when the average queue length is small, the current network congestion is low, and at this time we can choose a smaller max_p. when the average queue length is small, the current network congestion is low, then we can choose a smaller max_p. When the average queue length is large, the current network congestion is high, then we can choose a larger max_p. So this policy is in line with the empirical conclusion. The system selects the action by the *G-policy* according to the G-value Table before the *Q-value* is converged. The G-value Table is shown as in Table 4.

Table 4. G-value

i \ j	1	2	...	13	14
1	G(1,1)	G(1,2)	...	G(1,13)	G(1,14)
2	G(2,1)	G(2,2)	...	G(2,13)	G(2,14)
...	...	...	...	...	...
13	G(13,1)	G(13,2)	...	G(13,13)	G(13,14)
14	G(14,1)	G(14,2)	...	G(14,13)	G(14,14)

Simulation shows if use this policy, can not only improve the network congestion problem in the learning phase, and can greatly reduce the learning time.

The Q-learning controller process can be summarized as follows:

Algorithm. QRED Algorithm

```
1: Initialize the Q-value, state, action for the agent
2:  for time $t$ do
3:     for each $s$ do
4:       obtain corresponding information
         ($C_{\max}/C_{\min}/C_{ave}/D_{max}/D_{min}/D_{ave}$)
5:        calculate the state $s_t$ and reward $R$ as give in (8) (9)
          (10)
6:        update Q-value Table as given in (11)
7:         if the Q-value not converges then
8:            select action according to G-policy
9:         else
10:           choose $a_{t+1} = \arg\max Q(s_{t+1}, a_{t+1})$
11:        end if
12:    end for
13: t = t +1
14: end for
15: If the network topology changes, start learning again
```

3 Simulation Analysis

3.1 Simulation Scene and Parameter Setting

This section validates the validity and performance of the designed QRED algorithm by NS2 [33] simulation experiment, the simulation use the typical single-bottleneck network topology as shown in Figure 6. There are n senders (S_1~S_n), n receivers (D_1~D_n) and 2 routers (R_1, R_2) in the network. The bandwidth and delay between each sender and R_1 are 10Mbps and 10ms, the bandwidth and delay between each receiver and R_2 are also 10Mbps and 10ms. The link between R_1 and R_2 is a bottleneck link, the bandwidth and delay are 20Mbps and 20ms respectively. For comparison, we respectively analyzed respectively the queue length, throughput, delay and packet loss rate [31] of RED algorithm and QRED algorithm under low load, mid load, high load and changing load.

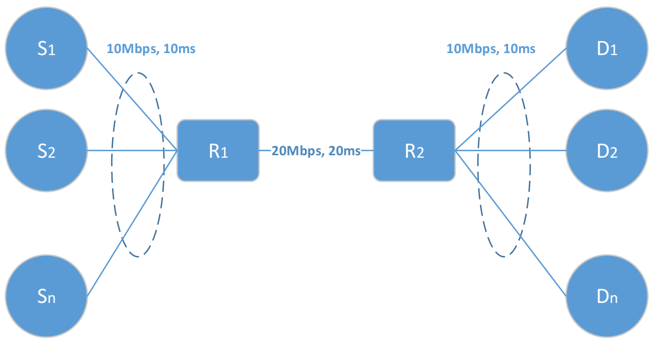
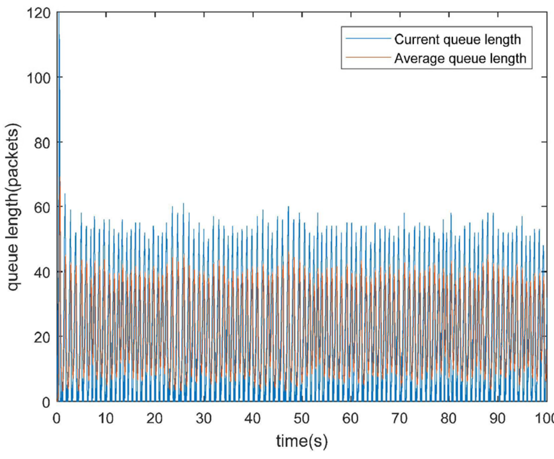


Figure 6. Simulation topology

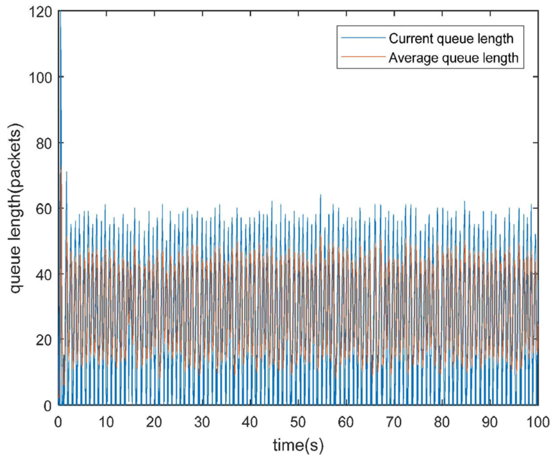
In the simulation, the common parameters of the two algorithms are set as follows: min_{th}=24, max_{th}=72, ω_q=0.002, the buffer size is 120 packets, the application layer uses TCP-based FTP services. In the learning module, the α value is set to 0.01 and γ is set to 0.8. K_T and K_D are set to 0.5, the learning cycle is set to 10s. RED algorithm and QRED algorithm in the implementation of the link between the two routers. The other links perform the Drop-Tail algorithm. In order to fully reflect the performance of QRED algorithm, this performance of the simulation after learning is completed.

3.2 Low Load

The performance of the algorithm in low load, that is, the number of network connections is fixed at 16, the simulation time is 100 seconds. Figure 7 shows the queue length of the RED algorithm and the QRED algorithm. As can be seen from the figure, the average queue length of the QRED algorithm is larger than that of the RED algorithm. This is because when the network is low loaded, the network is less congested, so the QRED algorithm reduces the drop probability by reducing max_p, and increasing the average queue length to improve network throughput.



(a) RED



(b) QRED

Figure 7. Queue length of the two algorithms in low load

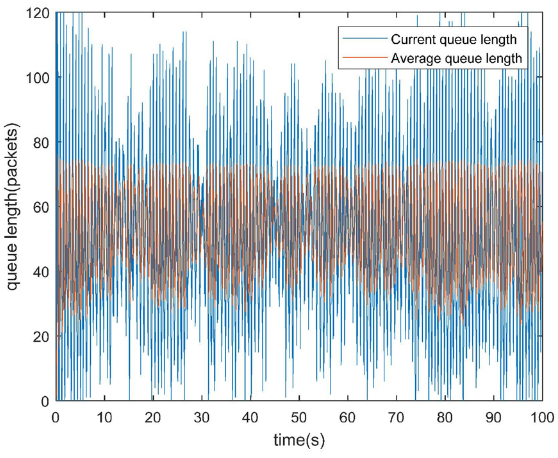
Table 5 shows the statistical results of the throughput, latency, and packet loss rate of the RED algorithm and QRED algorithm under the low load. As can be seen from Table 5, the average throughput of the QRED algorithm is larger than the RED algorithm, but the cost is to increase the delay.

Table 5. Network performance in low load

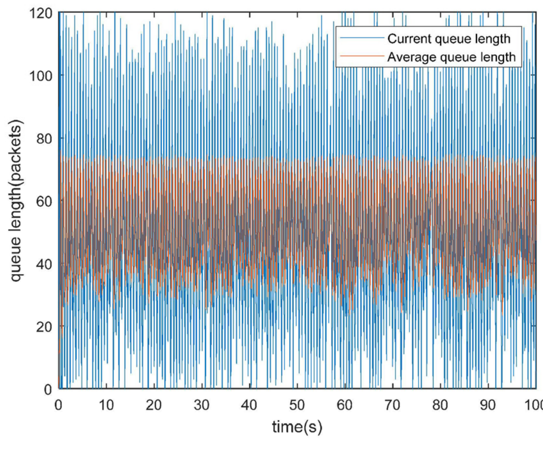
Performance / Algorithm	Throughput (Kbps)	Delay (ms)	Packet loss rate (%)
RED	19900.99	53.85	0.36
QRED	19940.93	55.78	0.33

3.3 Mid Load

The performance of the algorithm in mid load, that is, the number of network connections is fixed at 64, the simulation time is 100 seconds. Figure 8 shows the queue length of the RED algorithm and the QRED algorithm. It can be seen from the figure that the average queue length of the QRED algorithm is not much different from the RED algorithm. This is because when the network is mid loaded, the network congestion is moderate, the drop probability of the QRED algorithm does not need to be adjusted compared to the RED algorithm, so the performance of the network performance is not much difference.



(a) RED



(b) QRED

Figure 8. Queue length of the two algorithms in mid load

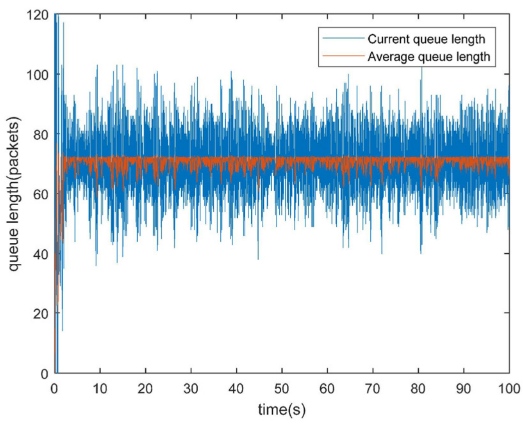
Table 6 shows the statistical results of the throughput, latency, and packet loss rate of the RED algorithm and QRED algorithm under the mid load. As can be seen from Table 6, the performance of the QRED algorithm is almost the same as that of RED.

Table 6. Network performance in mid load

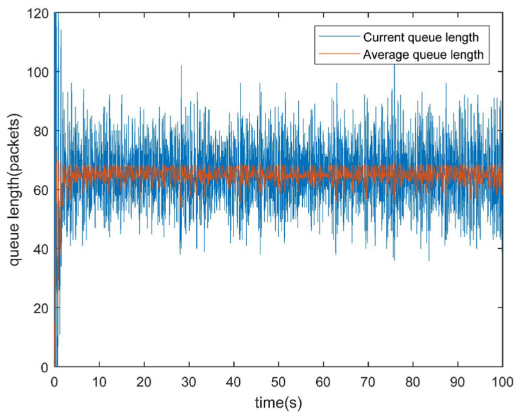
Performance / Algorithm	Throughput (Kbps)	Delay (ms)	Packet loss rate (%)
RED	19963.92	65.75	4.51
QRED	19969.71	65.70	4.47

3.4 High Load

The performance of the algorithm in high load, that is, the number of network connections is fixed at 128, the simulation time is 100 seconds. Figure 9 shows the queue length of the RED algorithm and the QRED algorithm. It can be seen from the figure that the average queue length of the QRED algorithm is smaller than that of the RED algorithm. This is because when the network is high loaded, network congestion is more serious. Therefore, the QRED algorithm increases the drop probability by increasing *max*$_p$, which reduce the average queue length and reduce the network delay.



(a) RED



(b) QRED

Figure 9. Queue length of the two algorithms in high load

Table 7 shows the statistical results of the throughput, latency, and packet loss rate of the RED algorithm and QRED algorithm under the high load. As can be seen from Table 7, the delay of the QRED algorithm is smaller than that of the RED algorithm, and the average throughput is slightly higher than the RED algorithm.

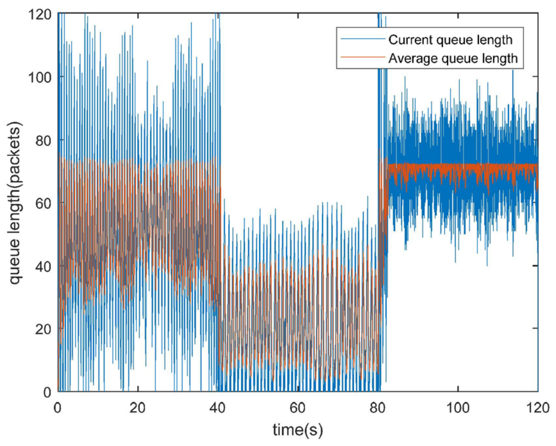
Table 7. Network performance in high load

Performance / Algorithm	Throughput (Kbps)	Delay (ms)	Packet loss rate (%)
RED	19970.88	71.72	13.15
QRED	19971.51	69.04	11.69

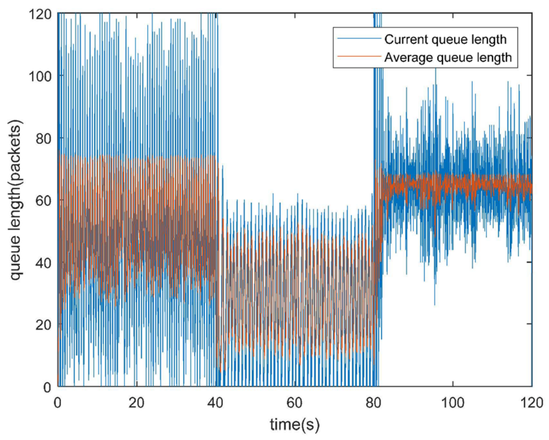
3.5 Changing Load

The performance in the case of changing load, that is, the number of connections is changed between 64, 16 and 128 every 40 seconds, the simulation time is 120 seconds.

Figure 10 shows the queue length of the QRED algorithm under the changing load. As can be seen from the figure, when the load changes, the queue of QRED algorithm can be adjusted adaptively, the overall queue length is more stable.



(a) RED



(b) QRED

Figure 10. Queue length of the two algorithms in changing load

The statistics of the throughput, delay, and packet loss rate of the RED algorithm and QRED algorithm are shown in Table 8. It can be found that in the dynamic network environment, the performance of the QRED algorithm is better than that of the RED algorithm.

Table 8. Network performance in changing load

Performance / Algorithm	Throughput (Kbps)	Delay (ms)	Packet loss rate (%)
RED	19854.53	67.76	6.26
QRED	19892.71	66.51	5.83

Through the aforementioned simulations and analysis, the performance of QRED algorithm include throughput, delay, and packet loss rate is consistent with the expected results. Although unilateral performance improvement is not too much, the overall performance in different network scenarios is better than the RED algorithm. Therefore, QRED algorithm can improve the sensitive parameters of RED algorithm to a certain extent, so that the QRED algorithm achieve better network performance and can select appropriate parameters adaptively according to different network scenarios.

4 Conclusion

In order to improve the RED algorithm parameter settings, make the algorithm achieve better network performance, the RED algorithm adaptively selects the appropriate parameters according to different network scenarios. This paper presents an improved RED algorithm, known as QRED. In this algorithm, the Q-learning algorithm is used to select the maximum packet drop probability parameter. The RED algorithm parameters are sensitive to defects and can predict dynamic changes in network systems. The optimal network performance control strategy is obtained by the learning unit. It can adaptively adjust the algorithm's maximum packet drop probability, achieving improved network performance by avoiding congestion. The simulation experiments verify the advantages of the QRED algorithm, which can be implemented in the network to maintain stability, reduce delay, improve the throughput and so on. The QRED algorithm overall network performance is better than that produced by the RED algorithm. In our future work, we will consider improving the accuracy of adjustment of actions, such as considering Fuzzy Q-learning. On the other hand, we will consider the fairness issue of the RED algorithm.

References

[1] V. Jacobson, Congestion Avoidance and Control, *Acm Sigcomm Computer Communication Review*, Vol. 18, No. 4, pp. 314-329, August, 1988.

[2] N.-F. Huang, G.-Y. Jai, H.-C. Chao, Y.-J. Tzang, H.-Y. Chang, Application Traffic Classification at the Early Stage by Characterizing Application Rounds, *Information Sciences*, Vol. 232, pp. 130-142, May, 2013.

[3] L. Zhou, H.-C. Chao, Multimedia Traffic Security Architecture for the Internet of Things, *IEEE Network*, Vol. 25, No. 3, pp. 35-40, May/June, 2011.

[4] L.-J. Zhang, D.-Y. Gao, W.-C. Zhao, H.-C. Chao, A Multilevel Information Fusion Approach for Road Congestion Detection in VANETs, *Mathematical and Computer Modelling*, Vol. 58, No. 5-6, pp. 1206-1221, September, 2013.

[5] Y.-C. Chang, Heterogeneous Wireless Sensor Network with EPC Network Architecture for U-life Environment, *Journal of Internet Technology*, Vol. 15, No. 4, pp. 647-655, July, 2014.

[6] Z.-P. An, D.-Y. Zhang, H.-M. Dang, H.-N. Ding, Enhanced Random Early Detection Algorithm, *Journal of Xian Jiaotong University*, Vol. 37, No. 8, pp. 829-832, August, 2003.

[7] T. Yamaguchi, Y. Takahashi, A Queue Management Algorithm for Fair Bandwidth Allocation, *Computer Communications*, Vol. 30, No. 9, pp. 2048-2059, June, 2007.

[8] S. Floyd, V. Jacobson, Random Early Detection Gateways for Congestion Avoidance, *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, pp. 397-413, August, 1993.

[9] S. Floyd, K. Fall, Promoting the Use of End-to-end Congestion Control in the Internet, *IEEE/ACM Transactions on Networking*, Vol. 7, No. 4, pp. 458-472, August, 1999.

[10] W.-C. Feng, D. D. Kandlur, D. Saha, K. G. Shin, A Self-configuring RED Gateway, *Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, New York, NY, 1999, pp. 1320-1328.

[11] S. Floyd, R. Gummadi, S. Shenker, *Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management*, http://www.icir.org/floyd/papers.html.

[12] S. S. Masoumzadeh, G. Taghizadeh, K. Meshgi, S. Shiry, Deep Blue: A Fuzzy Q-learning Enhanced Active Queue Management Scheme, *2009 International Conference on Adaptive and Intelligent Systems*, Klagenfurt, Austria, 2009, pp. 43-48.

[13] W.-C. Feng, K. G. Shin, D. D. Kandlur, D. Saha, The BLUE Active Queue Management Algorithms, *IEEE/ACM Transactions on Networking*, Vol. 10, No. 4, pp. 513-528, August, 2002.

[14] Y.-B. Zhang, D.-M. Hang, Z.-X. Ma, Z.-G. Cao, A Robust Active Queue Management Algorithm Based on Reinforcement Learning, *Journal of Software*, Vol. 15, No. 7, pp. 1090-1098, July, 2004.

[15] A. A. Tarraf, I. W. Habib, T. N. Saadawi, Reinforcement Learning-based Neural Network Congestion Controller for ATM Networks, *Military Communications Conference*, San Diego, CA, 1995, pp. 668-672.

[16] M.-C. Hsiao, K.-S. Hwang, S.-W. Tan, C.-S. Wu, Reinforcement Learning Congestion Controller for Multimedia Surveillance System, *2003 IEEE International Conference on Robotics*

and Automation, Taipei, Taiwan, 2003, pp. 4403-4407.

[17] K.-S. Hwang, S.-W. Tan, M.-C. Hsiao, C.-S. Wu, Cooperative Multiagent Congestion Control for High-speed Networks, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 35, No. 2, pp. 255-268, April, 2005.

[18] S.-J. Tang, J.-Q. Zhou, Z. Zhang, Research on Active Queue Management Based on Fuzzy Neural PID Controller, *Computer Technology and Development*, Vol. 8, pp. 99-102, August, 2015.

[19] C. Gao, Y.-J. Ou, Y.-C. Wang, An Improved Fairness Algorithm of RED by Distinguishing the Type of Protocol, *Science Technology and Engineering*, Vol. 15, No. 1, pp. 96-99, January, 2015.

[20] M. Zhang, J.-Q. Zhou, S.-J. Tang, Research on Improved gCHOKe Algorithm Based on Sampling, *Computer Technology and Development*, Vol. 25, No. 9, pp. 98-101, September, 2015.

[21] C. J. C. H. Watkins, *Learning from Delayed Rewards*, Ph.D. Thesis, University of Cambridge, Cambridge, UK, 1989.

[22] C. J. Watkins, P. Dayan, Technical Note: Q-Learning, *Machine Learning*, Vol. 8, No. 3-4, pp. 279-292, May, 1992.

[23] N. Nikaein, E. Schiller, R. Favraud, K. Katsalis, D. Stavropoulos, I. Alyafawi, Z. Zhao, T. Braun, T. Korakis, Network Store: Exploring Slicing in Future 5G Networks, *Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture*, Paris, France, 2015, pp. 8-13.

[24] H. Wei, Z.-F. Zhang, B. Fan, Network Slice Access Selection Scheme in 5G, *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference*, Chengdu, China, 2017, pp. 352-356.

[25] H.-J. Zhang, N. Liu, X.-L. Chu, K.-P. Long, A.-H. Aghvami, V. C. M. Leung, Network Slicing based 5G, Future Mobile Networks: Mobility, Resource Management, and Challenges, *IEEE Communications Magazine*, Vol. 55, No. 8, pp. 138-145, August, 2017.

[26] K. Katsalis, N. Nikaein, E. Schiller, A. Ksentini, T. Braun, Network Slices toward 5G Communications: Slicing the LTE Network, *IEEE Communications Magazine*, Vol. 55, No. 8, pp. 146-154, August, 2017.

[27] S.-B. Zhang, G. Li, J. Kang, Study of Congestion Control Algorithm based on Neural Network Supervised Control, *Application Research of Computers*, Vol. 27, No. 2, pp. 657-660, February, 2010.

[28] W. Wu, Y. Ren, X.-M. Shan, Stability Analysis on Active Queue Management Algorithms in Routers, *Proceedings of 9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Cincinnati, OH, 2001, pp. 125-132.

[29] M. May, J. Bolot, C. Diot, B. Lyles, Reasons not to Deploy RED, *1999 Seventh International Workshop on Quality of Service*, London, UK, 1999, pp. 260-262.

[30] G. Hasegawa, T. Matsuo, M. Murata, H. Miyahara, Comparisons of Packet Scheduling Algorithms for Fair Service Among Connections on the Internet, *Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, Tel Aviv, Israel, 2000, pp. 1253-1262.

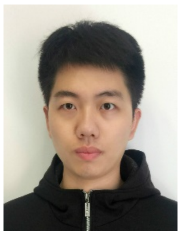
[31] L. Shen, X.-J. Mao, M.-G. Dong, The Construction of a Self-adaptive Multi-Agent System Based on Reinforcement Learning, *Computer Engineering & Science*, Vol. 33, No. 12, pp. 72-77, December, 2011.

[32] D.-H. Hu, An Introduction to Markov Process in Random Environment, *Acta Mathematica Scientia*, Vol. 30, No. 5, pp. 1210-1241, October, 2010.

[33] E. Altman, T. Jimenez, NS Simulator for Beginners, *Synthesis Lectures on Communication Networks*, Vol. 5, No. 1, pp. 1-184, January, 2012.

[34] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, Cambridge: MIT Press, 1998.

Biographies



Yuhan Su received his B.S. degree from Huaqiao University, Xiamen, China. He is currently pursuing the Ph.D. degree in communication and information system at Xiamen University, China. His research interests include wireless communication, congestion control, network coding, etc.

Lianfen Huang received the B.S. degree in radio physics and the Ph.D. degree in communication engineering from Xiamen University, Xiamen, China, in 1984 and 2008, respectively. She was a Visiting Scholar at Tsinghua University, Beijing, China, in 1997, and a Visiting Scholar at Chinese University of Hong Kong, Shatin, Hong Kong, in 2012. She is a Professor of Communication Engineering with Xiamen University. Her research interests include wireless communication, wireless network, and signal process.

Chenwei Feng received the B.S. degree in communication engineering from Fuzhou University, Fuzhou, China, in 2004. The M.S. degree and the Ph.D. degree in communication engineering from Xiamen University, Xiamen, China, in 2007 and 2017, respectively. He is an Associate Professor of Communication Engineering with Xiamen University of Technology, Xiamen, China. His research interests include wireless communication, wireless network and congestion control.