INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT
Int. J. Network Mgmt 2011; **21:** 147–167
Published online 25 August 2010 in Wiley Online Library (wileyonlinelibrary.com) DOI: 10.1002/nem.755

FQL-RED: an adaptive scalable schema for active queue management

Seyed Saeid Masoumzadeh[1,*], Kourosh Meshgi[2], Saeid Shiry Ghidari[2] and Gelareh Taghizadeh[1]

[1]*Department of Computer and Information Technology, Qazvin Azad University, Qazvin, Islamic Republic of Iran*
[2]*Department of Computer Engineering, Amirkabir University of Technology, Tehran, Islamic Republic of Iran*

SUMMARY

In a potentially congested network, random early detection (RED) active queue management (AQM) proved effective in improving throughput and average queuing delay. The main disadvantage of RED is its sensitive parameters that are impossible to estimate perfectly and adjust manually because of the dynamic nature of the network. For this reason, RED performs differently during different phases of a scenario and there is no guarantee that it will have optimal performance. Giving adaptability to RED has been the subject of broad research studies ever since RED was proposed. After a substantial study of AQM schemes and presenting a novel categorization for so-called modern approaches utilizing artificial intelligence tools to improve AQM, this paper proposes an algorithm enhancing RED as an add-on patch that makes minimal changes to the original RED. Being built on the basis of AQM schemes like ARED and Fuzzy-RED, this algorithm inherits adaptability and is able to adjust RED inaccurate parameters regarding network traffic status, trying to optimize throughput and average queuing delay in a scenario. This algorithm is a Q-learning method enhanced with a fuzzy inference system to provide RED with self-adaptation and improved performance as a result. Given the name of FQL-RED, this algorithm outperformed RED, ARED, and Fuzzy-RED, as the OPNET simulations show. Copyright © 2010 John Wiley & Sons, Ltd.

Received 9 April 2009; Revised 12 February 2010; Accepted 18 February 2010

1. INTRODUCTION

It is generally accepted that the problem of network congestion control remains a critical issue and a high-priority one, especially given the growing size, demand, and speed (bandwidth) of the increasingly integrated service networks. Despite the research efforts spanning a few decades and the large number of different schemes proposed, there are no universally acceptable control solutions. Current solutions in existing networks are becoming ineffective, and it is generally accepted that these solutions cannot easily scale up—even with various proposed 'fixes'. In this paper we propose a fuzzy-based congestion control approach, with traffic adaptability to address the congestion control problem. The performance of the proposed controlled system is evaluated via simulation.

Since our proposed scheme—fuzzy Q-learning random early detection (FQL-RED)—is fully compatible with RED, we can easily upgrade or replace the existing RED implementations by FQL-RED through patches.

1.1. Classical active queue management (AQM)

The problem of active queue management was first introduced into congestion control literature by the RED algorithm [1]. The problem then received more attention and formed the classic congestion

*Correspondence to: Seyed Saeid Masoumzadeh, Qazvin Azad University, Computer and Information Technology, Qazvin, Islamic Republic of Iran.
E-mail: masoumzadeh@gmail.com

control schemes. Classic AQM algorithms are classified according to the criteria on which the decision to drop packets from the queue (when the link suffers from congestion) is made. Four different strategies for queue management can be identified: (i) based on average queue length; (ii) based on packet loss and link utilization; (iii) class-based QM; and (iv) based on control theory. Alternatively, algorithms can be classified as being either reactive or proactive. A reactive AQM algorithm focuses on congestion avoidance, i.e. active early detection of and reaction to congestion. Congestion can occur in this case, but it will be detected early. Decisions on the actions to be taken are based on current congestion. A proactive AQM algorithm focuses on congestion prevention, i.e. intelligent and proactive dropping of packets, resulting in prevention of congestion from ever occurring and ensuring a higher degree of fairness between flows. Decisions on the actions to be taken are based on expected congestion [2].

In the first class of reactive AQM algorithms the dropping decision is based on the observed average queue length. The algorithms in this class can be divided further based on attention paid to fairness as it applies to the distribution of available bandwidth over the active data flows. RED is the basis for all algorithms in this class. It has been widely used in combination with TCP, and is known to have several drawbacks, which have led to the design of many other, improved algorithms. ARED [3], FRED [4], SRED [5], CHOKe [6], QVARED [7], and PUNSI [8] are typical algorithms of this class.

The main idea behind the second class of algorithms is to perform AQM based on packet loss and link utilization rather than on the instantaneous or average queue lengths. Algorithms in this class maintain a single probability p, which is used to drop packets when they are queued. If the queue is continually dropping packets due to buffer overflow, p is incremented, thus increasing the rate at which congestion notifications are sent back. Conversely, if the queue becomes empty or if the link is idle, p is decremented. Note that, in contrast to the first class of AQM algorithms, no queue occupancy information is used. BLUE and SFB [9], MRED [10], SFED [11], FABA [12], YELLOW [13], LUBA [14] and RAQM [15] are members of this class.

For the algorithms belonging to the third class, the treatment of an incoming packet (under congestion circumstances) depends on the class this packet belongs to. Theoretically, there are many possible class definitions, but in practice it is most common to categorize incoming packets based on the transport protocol (i.e., TCP or UDP) that has been used to send the packet. With this type of algorithms, for every non-TCP class a threshold is defined, setting a limit to the maximum amount of packets that a certain class can achieve in the queue. CBT [16] and DCBT [17] characterized this class of AQM algorithms as well as SHRED [18].

The fourth category of algorithms has been developed using classical control theory techniques. The queue length at the router is regulated to agree with a desired value by eliminating the 'error', that is, the difference between the queue length and this desired value. As a good example of this class, PI-controller [19], DRED [20], AVQ [21], SAVQ [22], EAVQ [23] and PRC [24] can be mentioned.

There are some hybrid algorithms which make use of the benefits of each class to compensate the cons of another group. REM [25], SVQ [26], and RaQ [27] are examples of this group.

One important drawback of the reactive AQM algorithms is that their congestion detection and control functions depend only on either the current queue status or the history of the queue status (e.g. the average queue length). Hence the congestion detection and control in these algorithms are reactive to current or past congestion, but do not respond proactively to incipient congestion. For example, the congestion detection method in RED can detect and respond to long-term traffic patterns using exponentially weighted moving average (EWMA) queue lengths. However, it is unable to detect incipient congestion caused by short-term traffic load changes. In this case, the implicit congestion notification sent back to the end hosts by a packet drop may be an inappropriate control signal, and can possibly make the congestion situation worse. The idea behind proactive AQM schemes is that the packet drop probability at time t is not only a function of the current system parameters (and thus the parameter values at time t) and/or previous values (at time $t-1$, $t-2$, etc.), but also on (estimations of) future parameter values (e.g. at time $t+1$). GREEN [28] and PAQM [29] are two proposed algorithms of this class. A detailed study on different classic active queue management algorithms can be found elsewhere [2,30,31].

DOI: 10.1002/nem

1.2. Developments to AQM

Sally Floyd, who developed RED, proposed that this algorithm can perform better when it is adaptive regarding the traffic of the network [32]. Thus Floyd extended the algorithm to achieve this adaptability but with no means from artificial intelligence (AI). Various extensions of the RED algorithm can be found [33–37] but still no AI method is used.

The classic approaches of AQM suffer from inaccurate parameters and therefore need human experts to adjust these parameters based on their knowledge, the application and traffic history. On the other hand, incipient congestions in the queue of routers can hardly be detected, which in turn result in decreased performance because of dealing with congestion side effects instead of just avoiding them. On the other hand, misbehaved sources are a treat to fairness of algorithms but usually no effort is spent in finding and extinguishing them. Also the packet drop policy usually follows a linear approach and lacks the flexibility needed to have maximum link utilization and throughput with lower packet drop rate. Machine intelligence tools are introduced in the AQM literature to address these drawbacks of classic methods and form a new generation of so-called modern schemes. Again four different categories of AI strategies for performance boost are introduced and have embedded in the AQM literature: (i) parameter tuning of classic approaches; (ii) prediction or detection of incipient congestion; (iii) classify sources based on their behavior and define service policy for them; and (iv) intelligent packet drop rate adjustment.

The first and most popular type of enhancement of AQM schemes is tuning the parameters of classic schemes. Having free parameters, an AQM scheme suffers from degradation of functionality in some circumstances, so its parameters need to be adjusted carefully regarding the scenario of use. These parameters can be time constants, thresholds, weighting coefficients, or more complex ones that play a critical role in the behavior of the system. Handling traffic turbulences, maximizing the utilization of links, minimizing queuing delay etc. are the results of fine tuning of the parameters in a scenario. As an example, the classic ARED algorithm is shown to do better than RED across congested links through multiple experiments, yet it still needs to be modified to adapt to the changes of network load more effectively. Fuzzy DS RED [38] proposed to use fuzzy rules for RED, believing that fuzzy logic improves the packet drop in a network. Also being facilitated with a fuzzy rule base, AFRED [39] benefits from adaptability, which enables it to perform well in dynamic environments such as networks with dynamic traffic types. FCRED [40], another scheme, uses a fuzzy controller to adjust maximum drop probability. Neuron-RED [41] and Neuron PID [42] use a neural controller to perform the same task. These three algorithms try to stabilize the average queue length around the target queue length. To guarantee the queuing delay in a RED-based queue, a method has been devised [43] which uses stochastic learning automata and thus makes the RED adaptive. PSO-PID [44] extends the PID method with the help of the particle swarm optimization algorithm, using it to adjust PID controller parameters. RBF-PID [45] is another example of adjusting these parameters adaptively, in this case using an RBF neural network. In this approach, the RBF neural network is used to adjust PID parameters regarding link capacity, traffic load and transmission delay. In Yanfei *et al.* [46] the fuzzy decision process is used to make the packet drop mechanism intelligent. GA-based PID [47] offers a PID controller based on a genetic algorithm and states that this extended algorithm increases throughput and decreases packet loss in regard to the classical PID controller scheme and also keeps the queue length steady. A genetic algorithm has been proposed [48] and evaluated for the optimal tuning of fuzzy PI controller parameters. The main objectives of the controller design method are fast response to high load variations and disturbance rejection in steady-state behavior. These design goals are encoded in a performance index and the genetic algorithm optimally tunes the fuzzy controller parameters. Adaptive and classic control theory serves this category of AQM enhancement as well. Methods have been proposed [49–51] that were designed to stabilize RED by this means.

There are plenty of fixed-strategy self-configuring schemes that use heuristics to adapt parameters. Self-configuring RED [52] is a good example of these. The idea behind this algorithm is to infer whether RED should become more or less aggressive by examining the variations in average queue length. Based on the observed queue length dynamics, the algorithm adjusts the value of parameters by scaling them by constant factors, depending on which threshold it crosses. Although

DOI: 10.1002/nem

performing well, such algorithms do not use AI and do not meet our criteria for categorization as a modern approach.

A second way of providing intelligence to AQM approaches is predicting incipient congestion or detecting congestion in the early stages. Enabling a router to predict congestion by learning from previously experienced similar conditions or by embedding knowledge into its decision-making mechanism will improve the throughput and prevent drastic packet loss. Distinguishing bursty traffic from probable congestion is another goal which AI promises to help AQM towards. A congestion detector using fuzzy logic has been devised [53] that not only inherits good behavior of classic AQM schemes but also the membership functions are automatically defined using PSO algorithm. Another example is a method [54] that uses neuro-fuzzy prediction to capture traffic bursts and correctly predicts incoming steady congestion. NN-RED [55] uses an artificial neural network to predict an early congestion in RED.

Misbehaving sources challenge the fairness of a scheme and lower the quality of service (QoS) of the router to other sources. Classification of sources and crediting them based on their history is a favor that AI and accounting schemes can do for AQM algorithms. While accounting methods follow a crisp strategy to detect and interact with flows, intelligent approaches bring more flexibility and maintain high link utilization, yet are capable of handling congestion more effectively. There is no significant research in this area to date, but it is possible with current knowledge.

Early AQM algorithms propose a linear packet drop rate which relates to queue length or packet loss rate. Nonlinear approaches (e.g. GRED [56] and Exponential RED [35]) worked well to improve the performance of AQM approaches, but adjustable curves of packet drop rate against queue delay are preferred as they were shown to be effective in absorbing bursts and maintaining a non-full queue with high throughput. NLRED [57] is the same as the original RED except that the linear packet-dropping probability function is replaced by a nonlinear quadratic function. While inheriting the simplicity of RED, NLRED was shown to outperform RED, is less sensitive to parameter settings, has a more predictable average queue size, and can achieve a higher throughput. Another minimal adjustment to the RED algorithm was proposed by Hyperbola RED (HRED), which uses the hyperbola as the drop probability curve [58]. The control law of HRED can regulate the queue size, which can be set by the user. As the reference queue size is set by the user, HRED is no longer sensitive to the level of network load and it can achieve higher network utilization and result in predictable average queuing delays. It retains the ability to control short congestion by absorbing bursts, because it still keeps the moving average queue size algorithm and maintain a non-full queue. The next development used a fuzzy logic controller for RED [59]. This approach is based on the idea that linguistic labels can perform well in implementing nonlinear probabilistic drop functions because they have a better understanding of the environment, so this approach can provide better QoS for different types of traffic.

It gets even more interesting when AI tends to solve network congestion problems without having a classic AQM scheme as its basis. A good example is the novel approach proposed in Hadjadj Aoul *et al.* [60], which is based on self-adaptation, i.e. the AQM scheme adjusts its own parameters in the operation to achieve the highest performance level. This method does not follow any of the classic approaches but uses a fuzzy logic-based AQM and optimizes the equilibrium of throughput against queuing delay.

As is clear from the literature, enhancing RED as the most popular AQM used today and adjusting its parameters as the favorite type of enhancement over RED forms the largest portion of modern AQM literature. Most of these approaches try to solve the drawbacks of this algorithm (to be discussed in Section 2.2) by addressing one issue directly, which will also improve others implicitly. In this paper we follow this popular pathway and adjust RED parameters dynamically using AI, stated technically as a Q-learning algorithm enhanced by fuzzy inference system.

2. BASIC SCHEME OF CONGESTION CONTROL

2.1. Random early detection algorithm

RED, proposed in 1993, was the first AQM algorithm developed [1]. It has been widely used with TCP and has been recommended by the Internet Engineering Task Force. RED has served as the basis for

DOI: 10.1002/nem

```
For each packet arrival:
    Calculate the average queue size q_avg
If min_th ≤ q_avg < max_th
    Calculate probability p_a
    With probability p_a: mark/drop the arriving packet
Else if max_th ≤ q_avg
    Mark/drop the arriving packet
Else
    Do not mark/drop packet
```

Figure 1. Pseudo-code of RED algorithm.

many other algorithms. The algorithm detects incipient congestion by computing the average queue size at the router. When this average queue size exceeds a certain preset threshold, the router drops or marks each arriving packet with a certain probability p_a, which is a linear function of the average queue size. Connections are notified of congestion either by dropping packets arriving at the router, or rather by setting a bit in packet headers (which is referred to as 'marking a packet'). RED can be specified in pseudo-code as shown in Figure 1, where $\min_{th}$ and $\max_{th}$ are minimum and maximum thresholds which can be set by the algorithm user, and are both smaller than the maximum queue size allowed by the router. Whenever a packet arrival is detected by a router implementing RED, the average queue size q_{avg} is calculated using a low-pass filter to reduce the potentially significant influence that the short-term traffic bursts could have on the average queue size.

Calculation of q_{avg} is carried out using

$$q_{avg} = (1 - w_q) \times q_{avg} + w_q \times q \tag{1}$$

where q is the instantaneous queue length as observed at the router and w_q is the weight applied by the low-pass filter to the 'old' average queue size. By increasing this value w_q, the influence of the current queue size on q_{avg} is also increased. This will raise the influence of traffic bursts on the average length of queue. After q_{avg} has been calculated, it is compared to two threshold values: $\min_{th}$ and $\max_{th}$. When q_{avg} is smaller than $\min_{th}$ no packets are marked, and thus each arriving packet is appended at the end of the queue. When q_{avg} lies between $\min_{th}$ and $\max_{th}$, each arriving packet is marked with probability p_a. When q_{avg} is greater than $\max_{th}$, all incoming packets are marked with probability 1. The marking probability p_a is calculated as follows:

$$P_a = P_{max} \frac{q_{avg} - \min_{th}}{\max_{th} - \min_{th}} \tag{2}$$

In the original algorithm, p_a is increased even further as a function of the number of packets that have arrived since the last packet was marked, resulting in a new marking probability p_b. Figure 2 shows a graphical representation of the possible values for the marking probability p_a.

Here we summarize some of the design goals and guidelines for RED gateways. The main goal of this algorithm is to provide congestion avoidance by controlling the average queue size. Additional goals include the avoidance of global synchronization and of a bias against bursty traffic and the ability to maintain an upper bound on the average queue size even in the absence of cooperation from transport layer protocols. The first job of a congestion avoidance mechanism at the gateway is to detect incipient congestion. A congestion avoidance scheme maintains the network in a region of low delay and high throughput. The average queue size should be kept low, while fluctuations in the actual queue size should be allowed to accommodate bursty traffic and transient congestion. The gateway monitors the size of the queue over time, so it is the appropriate agent to detect incipient congestion. Having a unified view of the various sources contributing to this congestion, the gateway is also the appropriate agent to decide which sources to notify of this congestion. As a result, in a network with connections with a range of different requirements for roundtrip times, throughput, and delay sensitivities, the gateway is the most appropriate agent to determine the size and duration of short-lived bursts in the queue size to be accommodated by the gateway. The RED gateway can do this by controlling the time constants used by the low-pass filter for computing the average queue size.

DOI: 10.1002/nem

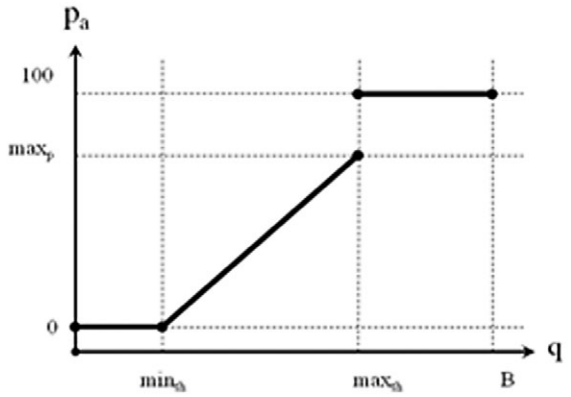

Figure 2. RED algorithm mark probability.

Our proposed scheme gives the gateway an intelligent tool to optimize this control, alongside controlling other parts of the algorithm autonomously. The goal of the gateway is to detect incipient congestion that has persisted for a 'long time' (several roundtrip times). The second job of a congestion avoidance gateway is to decide which connections to notify of congestion at the gateway. One goal is to avoid a bias against bursty traffic. Networks contain connections with different burstiness, and gateways such as Drop Tail and Random Drop gateways have a bias against bursty traffic. With Drop Tail gateways, if the incoming traffic from a particular connection is burstier, the gateway queue is more likely to overflow when packets from that connection arrive at the gateway. Another goal in deciding which connections to notify about the congestion is to avoid the global synchronization that results from notifying all connections to reduce their windows at the same time. Global synchronization has been studied in networks with Drop Tail gateways where this phenomena results in loss of throughput in the network. In order to avoid problems such as biases against bursty traffic and global synchronization, congestion avoidance gateways can use distinct algorithms for congestion detection and for deciding which connections to notify of this congestion. The RED gateway uses randomization in choosing which arriving packets to mark, as all its successors do. With this approach, the probability of marking a packet from a particular connection is roughly proportional to that connection's share of the bandwidth through the gateway. This method can be efficiently implemented without maintaining per-connection state at the gateway. One goal for a congestion avoidance gateway is the ability to control the average queue size even in the absence of cooperating sources. This can be done if the gateway drops arriving packets when the average queue size exceeds some maximum threshold (rather than setting a bit in the packet header). This method could be used to control the average queue size even if most connections last less than a roundtrip time (as could occur with modified transport protocols in high-speed networks), and even if connections fail to reduce their throughput in response to marked or dropped packets. This maximum threshold is also efficiently adjusted in our proposed method autonomously, as will be mentioned soon.

2.2. Disadvantages of RED

Despite having been widely used in combination with TCP for several years, RED has not found acceptance in the Internet research community [2]. The reason comes from the drawbacks of this algorithm, which can be summarized as follows:

- The packet loss fraction is equal for all flows, regardless of the bandwidth used by each flow.
- No restriction exists against aggressive, non-adaptive flows, which has a negative impact on well-behaving, adaptive flows.
- It is difficult to parameterize RED queues to give good performance under different congestion scenarios.
- The equilibrium queue length strongly depends on the number of active TCP connections.

DOI: 10.1002/nem

- RED performance is sensitive to the packet size [57].
- With RED, wild queue oscillation is observed when the traffic load changes [5].

These drawbacks have been the main reason for the development of a vast collection of improved AQM algorithms, of which the most significant are presented earlier in this paper. Although RED suffers from these defects, it is implemented in the vast majority of routers, including all well-known manufactured products. Designing add-ons for the current RED algorithm and applying them in the form of software or hardware patches to the existing routers, instead of changing them to totally new ones, is the only economically acceptable solution for this problem. In this paper, we introduce an add-on complementary algorithm which is able to solve most of the drawbacks of RED.

To address these issues, AQM algorithms have tried to collect additional information from the network to be able to react more intelligently to different conditions. Tools such as accounting mechanisms (e.g. SFB), active connection number estimators (e.g. FPQ [61]), zombie list (e.g. SRED [5]), and early congestion predictors (e.g. NN-RED [55]) are good examples of this claim. Although they provide good knowledge of the network, they are expensive in terms of computational power and/or memory for the routers they are operating on. On the other hand, most of these schemes are able to focus on limited parameters of the network but ignore other parameters. Furthermore, some of the proposed methods lose modularity in their design process and are hard to extend. As is obvious from the literature, adjusting efficient parameters for RED or its derivations is the most popular scheme, while some algorithms improve fairness too, and some other schemes prove independence to the number of TCP connections. Except for proactive schemes, all algorithms use traffic history directly or indirectly to detect incipient congestion, make decisions about dropping a packet, punish misbehaving connections, and other congestion-related operations. In order to maintain the traffic history, different means are utilized, from a single parameter that is updated with an EWMA [1] to neural networks [41] and complex data structures.

In summary, the main defect of the RED algorithm is its parameters, because it is difficult to adjust them well enough to perform satisfactorily in different scenarios. This issue is usually not fully solved even with the help of experts, due to factors such as non-real time network response, human mistakes, lack of full-time monitoring, and financial issues. Designing a mechanism to adjust the parameters automatically is the best choice that comes to mind. Briefly, this suggests a solution for 'adjusting parameters of RED algorithm to improve its performance in different congestion scenarios'. Each scenario is a network with different kinds of applications such as multimedia, file transfer, real-time applications, and low fault-tolerable data communication. The parameters of the RED algorithm are $\min_{\text{th}}$, $\max_{\text{th}}$, $\max_p$, and w_q. To improve performance, these parameters should change according to current network traffic, and the mission of AQM algorithms is to best manage the condition automatically. Our method uses machine learning to serve this need for automation.

2.3. Sensitivity of RED parameters

Network traffic can hardly be modeled in many networks. Thus the process of parameter adjustment in AQM algorithms is different and, if done manually, loses accuracy as traffic changes. RED gateways have additional parameters that determine the upper bound on the average queue size, the time interval over which the average queue size is computed, and the maximum rate for marking packets. The congestion avoidance mechanism should have low parameter sensitivity, and the parameters should be applicable to the networks with widely varying bandwidths. A network designer has to decide between three RED gateway parameters, w_q, $\min_{\text{th}}$, and $\max_{\text{th}}$, to make decisions about the average queue size and allowable queue burst. The parameter $\max_p$ can be chosen from a fairly wide range, because it is an upper bound on the actual marking probability p_a.

To have efficient parameter values for the basic RED algorithm some rules have been introduced that give adequate performance to the RED gateway under a wide range of traffic conditions [1]. Also there exist a few rules of thumb to improve performance of the RED gateway [61]:

1. To ensure adequate calculation of the average queue size, set $w_q \leq 0.001$. The average queue size at the gateway is limited by $\max_{\text{th}}$, as long as the calculated average queue size q_{avg} is a

DOI: 10.1002/nem

fairly accurate reflection of the actual average queue size. The weight w_q should not be set too low, so that the calculated average queue length does not delay too long in reflecting increases in the actual queue length. The calculation of the average queue size can be implemented particularly efficiently when w_q is a (negative) power of two. Equation (3) describes the upper bound on w_q required to allow the queue to accommodate bursts of L packets without marking packets. Given a minimum threshold $\min_{th}$, and given that we wish to allow bursts of L packets arriving at the gateway, then w_q should be chosen to satisfy the following equation for $\text{avg}_L < \min_{th}$:

$$L + 1 + \frac{(1 - w_q)^{L+1} - 1}{w_q} < \min_{th} \tag{3}$$

2. Set $\min_{th}$ sufficiently high to maximize network power. The thresholds $\min_{th}$ and $\max_{th}$ should be set sufficiently high to maximize network power. Because network traffic is often bursty, the actual queue size can also be quite bursty; if the average queue size is kept too low, then the output link will be underutilized.
3. Make $\max_{th} - \min_{th}$ sufficiently large to avoid global synchronization. Make $\max_{th} - \min_{th}$ larger than the typical increase in the average queue size during a roundtrip time, to avoid the global synchronization that results when the gateway marks many packets at one time. One rule of thumb would be to set $\max_{th}$ to at least twice $\min_{th}$. If $\max_{th} - \min_{th}$ is too small, then the computed average queue size can regularly oscillate up to $\max_{th}$; this behavior is similar to the oscillations of the queue up to the maximum queue size with Drop Tail gateways.
4. RED is highly dependent on the number of active TCP connections (N). As shown in Morris [61], the length of queue is related to $\max_{th}$, $\max_p$ and N as shown in equation (4):

$$q = 0.91 \times \sqrt[3]{N^2 \times \frac{\max_{th}}{\max_p}} \tag{4}$$

An RED router works well if q never exceeds $\max_{th}$, so that the router is never forced to drop 100% of incoming packets. Equation (4) implies that the parameter values required to achieve this goal depend on the number of active connections. This is a formalization of an idea used by ARED and SRED, which keep q low by varying $\max_p$ as a function of N. The observation that one could alternatively change $\max_{th}$ and keep $\max_p$ fixed is one way of looking at the FPQ algorithm presented in Morris [61]. In order to compensate for the influence of N or even to take advantage of it, the ratio of $\max_{th}/\max_p$ should be adjusted cautiously.

5. Set $\max_{th}$ high enough to avoid retransmissions. Due to the fact that drop probability in RED jumps from $\max_p$ to 100% when queue length exceeds $\max_{th}$, the volume of the retransmissions raises immediately for TCP packets, acting as another critical traffic burst.

2.4. Dynamics of RED

The RED algorithm deals with some problems such as global synchronization, bounded delay requirement, priority of some packets over others, transient network states, jitters, connection starvation, bursty traffic with characteristics of incipient congestion and, most important of all, four free parameters: $\max_{th}$, $\max_{th}$, $\max_p$, and w_q.

A queue is assumed to be lightly loaded when average queue size is less than $\min_{th}$, so more packets could pass the gateway without being marked and getting dropped. Setting $\min_{th}$ to an appropriate value could help the gateway to maintain good link utilization. Relatively small values of $\min_{th}$ ($\min_{th} << \max_{th}$) tend to keep the queue size low. Approaching $\min_{th}$ to $\max_{th}$ ($\min_{th} \rightarrow \max_{th}$) leaves the gateway vulnerable to bursts and in the case of equal values ($\min_{th} = \max_{th}$) RED is reduced to the TCP Tail Drop [62] mechanism.

Adjusting the maximum threshold of RED is the subject of much of the reviewed literature. This is due to the extensive role of this parameter in handling congestion. (i) Algorithms like ARED [36] and

DOI: 10.1002/nem

DSRED [63] try to converge average queue length to a predefined length L. In modern approaches this length is dynamically changed and max_{th} controls it directly by dropping all packets when average queue length exceeds this threshold. Therefore, in order to maintain a suitable queue length, max_{th} should be set properly. (ii) This parameter is the marginal value between the normal situation and a congested one, so in order to detect congestion the critical value would be max_{th} to determine congestion. (iii) As stated in equation (4), this parameter can neutralize the effect of active connection count. On the other hand, a quick change of queue length causes jitter and is not desired (though much control theoretic-based AQM tries to stabilize queue length as well as other schemes). Setting max_{th} to small values ($max_{th} \rightarrow min_{th}$) results in not using the full capacity of the link and again the case of equal values ($min_{th} = max_{th}$) results in the Tail Drop [62] mechanism. Increasing this threshold ($max_{th} \rightarrow B$) can increase the probability of full buffer and connection starvation.

In the transient congestion window, when the queue length is between two thresholds ($max_{th} < q \leq max_{th}$) the drop probability linearly correlates with average queue length, which would be max_p when this average reaches max_{th}. This parameter defines the way RED deals with transient traffic. Like max_{th}, this parameter can neutralize the effect of an active connection count. A high value of max_p ($max_p \rightarrow 100$) easily allows the queue length to grow to buffer size, and low values of this parameter ($max_p \rightarrow 0$) make a pessimistic assumption about traffic and interpret every burst as severe congestion. When this maximum probability becomes zero ($max_p = 0$), RED is again reduced to Tail Drop.

w_q defines the inertia of RED against the change in average queue length. Too high a value of w_q cannot filter transient congestion and expose average queue length to high turbulence. On the other hand, too small a value of w_q responds slowly to queue length changes and the gateway cannot detect early congestion.

3. PROPOSED SOLUTION

3.1. Automation and prediction

Earlier in this paper we justified that most AQM schemes take advantage of network traffic history to predict its future behavior. This history can affect several aspects in which an AQM scheme is dealing with the network. As is common in the real world, the traffic of networks is almost the same for long periods of time, maintaining mostly constant or periodic behavior. The reviewed AQM mechanisms counted on this fact and used the history as an immediate response to all conditions, even if it is not exactly the same. The main problem of AQM schemes is their inaccurate parameters, as was mentioned before. Setting these parameters automatically is of great importance to improve a scheme. Additionally, several individual input states can lead to a state of the network in which the history of the network may have the ability to predict the next probable state transition.

Regarding these two needs—automation and prediction—one of the best solutions is to give the scheme learning ability.

One of the missions of an AQM scheme is accommodating bursty traffic in a buffer. To accept L bursty packets, the RED algorithm needs to adjust min_{th} such that equation (3) remains true. This is a constraint on the values of min_{th} and w_q. On the other hand, these schemes should handle transient congestions and distinguish them from steady ones. The average queue length is an indicator for congestion. Its change over time is determined by means of an EWMA using w_q as the weight. Thus manipulating this weight will define the inertia of the scheme to change state between transient and steady congestion. The intelligence to adjust this parameter lies in using traffic history to optimally set this weight. Additionally, the parameter max_{th} cooperates in the process of congestion detection and deciding when immediate full drop is needed. The max_p parameter value really matters in dealing with transient congestion and tries to keep the queue as responsive as possible. The way traffic reacted in the past against packet drops in this situation is the key of improving this parameter. Nevertheless, max_p and max_{th} are engaged in another constraint dictated by equation (4) to compensate for the effect of the number of TCP connections. Estimation of this number, N, costs some computational power [61], which is not always feasible for all kinds of routers, and implicit compensation for it seems a better solution. The process of using history by means of some statistical methods or estimators is

DOI: 10.1002/nem

vulnerable to changes in the network, including changes in topology, application and faults. Thus, as a primary objective, the scheme should keep responding to new situations appropriately, and secondarily, should maintain the history to respond to cases similar to the stored ones.

Machine-learning tools are appropriate for maintaining history. They offer a wide range of methods, each of which is suitable for various types of problems. Reaction to traffic conditions of the network is a subject that should be learned in this case. Thus some features must be extracted in a way that represents a condition of traffic with appropriate details. On the other hand, traffic history contains many repeated situations. Appropriate actions to deal with these situations which change RED parameters are to be learnt along with traffic transition, although the learning agent will be more effective in the case of periodic traffic.

Reinforcement learning is the problem faced by an agent that has to learn behavior through trial and error interactions with a dynamic environment, Further, there is a focus on online performance, which involves finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). Thus reinforcement learning is particularly well suited to problems which include a long-term versus short-term reward trade-off. It has been applied successfully to various problems, including robot control, elevator scheduling, telecommunications, backgammon and chess [64]. One of the most effective algorithms of this type is Q-learning. Q-learning is a reinforcement learning technique that works by learning an action-value function that gives the expected utility of taking a given action in a given state and following a fixed policy thereafter. Q-learning is able to compare the expected utility of the available actions without requiring a model of the environment. The core of the algorithm is a simple value iteration update. For each state s from the state set S, and for each action a from the action set A, we can calculate an update to its expected discounted reward using the following expression:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(s_t, a_t) \times [r_{t+1} + \gamma \times \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{5}$$

where r_t is an observed real reward at time t, $a_t(s,a)$ are the learning rates such that $0 \leq a_t(s,a) \leq 1$, and γ is the discount factor such that $0\ \gamma < 1$.

The problem presented in this paper will be modeled in the form of a Q-learning optimization problem. Each Q-learning problem consists of four design parts: input states, actions, goal (or goals), and reinforce signal. Input states and actions will be described in full detail in the implementation section.

As stated, Q-learning uses a table to save the state–action values. This table is subjected to grow in size with the growth of states or actions. In this problem, each state is a combination of values of each input, each of which usually has a wide range. Thus there will be a large input space and the table seems impossible to fit into the available memory of routers. This issue, along with other issues such as similar actions for similar states and dependency of states count on the buffer size of each individual router, drives us to obtain assistance from fuzzy logic.

3.2. Using a fuzzy inference system

Modeling states of the reinforcement learning algorithm by using a fuzzy rule base facilitates this algorithm to deal with large or continuous spaces, e.g. state and/or action space. Thus it is a mechanism to handle high dimensionality for a table-driven algorithm like Q-learning. This ability for generalization provides flexibility to Q-learning so that it can have variable input state size in different cases. This model also enables knowledge encapsulation into the learning table, i.e. to merge a priori or expert knowledge into the problem. Taking advantage of fuzzy logic, the model is also able to generate continuous actions. Since Q-learning and temporal difference (TD) learning are typically slow, i.e. the agent needs more trials and episodes to learn the desired behavior, the need for a little boost emerged. Fuzzy logic, with its fast nature, speeds up the learning process of these algorithms [65], which makes fuzzy Q-learning (FQL) a fast and powerful extension to classic Q-learning and makes it a sound choice to serve as the machine learning basis of our algorithm.

Fuzzy reinforcement learning is based principally on fuzzy inference systems (FIS). FIS are universal approximators and they can learn by example. The most important feature of FIS is that they can incorporate human a priori knowledge into their parameters and that these parameters have clear

DOI: 10.1002/nem

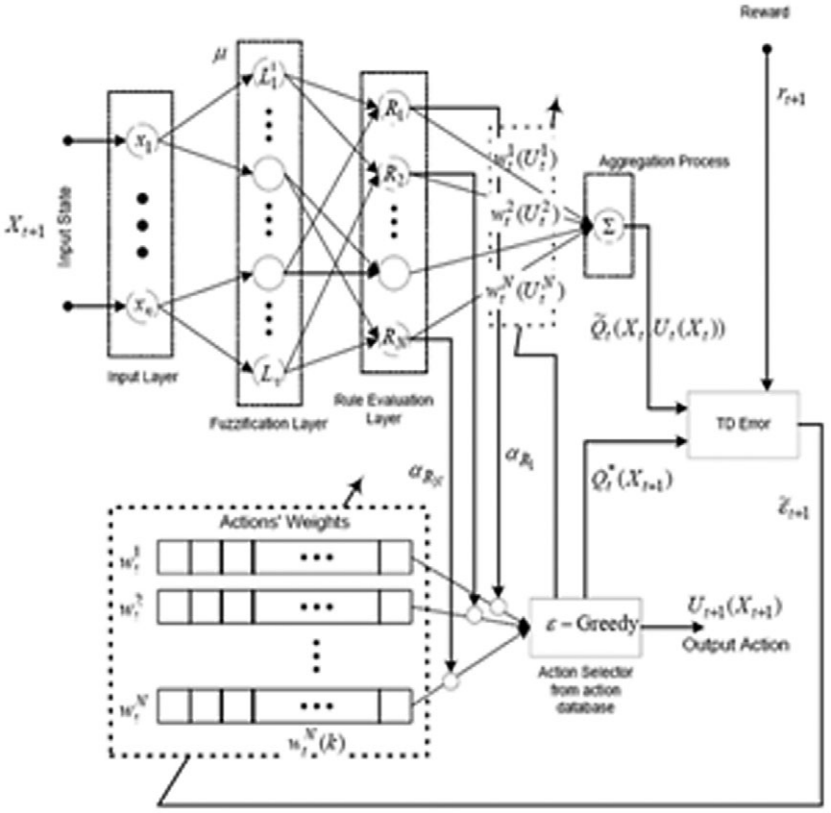

Figure 3. FQL Structure [66].

physical meaning which will actually speed up learning. An FIS is based on a rule base in which each rule received an antecedent part and its corresponding consequent. Usually the antecedent combination does not matter in tuning FIS, while the issue is the choice of different linguistic terms for each fuzzy variable and the conclusion of each rule. With the method used here it is possible to tune the conclusion part of each rule in a TS-FIS and this would be done over all possible rules in the rule base. FQL is the fuzzy extension of Q-learning, which is an online model-free optimization of a control policy. Briefly speaking, FQL use of TS-FIS estimates the Q-value function for the current state–action pair. This Q-value function, along with the optimal Q-value of the state, calculated in the same way, will be used to compute TD error. Later on, based on the TD error and update rule of TD learning, the action weights will be updated towards gaining more reinforcement, as is the case in Q-learning. The agent then chooses actions based on the quality values (weights) of different actions available in the action set of each rule, along with an exploration/exploitation mechanism named double e-Greedy [66]. As shown in the schematic diagram in Figure 3, everything is based on the actions' quality values and fuzzy inference. Our proposed method to improve RED algorithm was named FQL-RED after this learning mechanism and the whole process of learning is abstracted in the add-on module which is added to RED, just changing the main parameters of RED externally, leaving the main function of this algorithm (which is possibly hard-wired in the router dropping unit and the only access to that being router commands which configure RED parameters) unchanged.

3.3. Implementation

Here the whole process of FQL-RED is explained step by step.

1. *Initialization*. The very first configuration of RED, which is performed by the network designer and contains his expert knowledge. Initializing the learning parameters $\gamma$and exploration rate θ is done here too.

DOI: 10.1002/nem

2. *Data gathering*. Construct an input vector containing packet enqueue, packet drop, average queue size, minimum threshold, maximum threshold, which defines the current state of the learning agent.
3. *Fuzzification*. We apply a triangular membership functions to fuzzify each input vector into *low*, *medium* and *high*.
4. *Generation of reinforcement signal*. The reinforcement signal is a linear combination of throughput of the router and instantaneous queuing delay. These factors are inversely correlated, and thus the learning mechanism tends to balance their trade-off in dynamic traffic conditions in the network.
5. *Estimation of the optimal Q-value*. The current input value fires a set of rules. The activated rules have a truth value which is calculated by rule evaluation. The optimal Q-value is the maximum value resulting from the truth value of each rule multiplied by its best action value in the whole rule base:

$$Q^*(X_t) = \frac{\sum_{R_i \in A(X_t)} \alpha_{R_i}(X_t) \times \left[\max_{a \in U^i} w_t^i(a)\right]}{\sum_{R_i \in A(X_t)} \alpha_{R_i}(X_t)} \tag{6}$$

   In this equation R_i represents a rule from the rule base, αR_i is truth value of this rule, $A(X_t)$ is the set of activated rules with input X_t, and w_t is the table of Q-learning in time t. This table stores values of state–action pairs corresponding to state i and actions listed in the U_i set.
6. *TD error calculation*. In order to calculate the transfer probability of each state–action to a specific state, TD(0) error is calculated as follows:

$$\tilde{\varepsilon}_{t+1} = r_{t+1} + \gamma \times Q_t^*(X_{t+1}) - \tilde{Q}_t(X, U_t(X_t)) \tag{7}$$

   where r_{t+1} is the generated reinforce signal and $\gamma\,(0 < \gamma \leq 1)$ is the discount factor.
7. *Exploration/exploitation*. One of the most important issues of reinforcement learning is to balance exploration versus exploitation to find the best action while keeping good performance. We use the following formula for action selection:

$$\mathrm{EE}(a) = w_t^i(a) + \frac{\theta}{e^{n_t(a)}} \tag{8}$$

   in which θ is a positive coefficient for the direct exploration part, $w_t^i(a)$ is the action's weight, and $n_t(a)$ is the total number of times that the action has been used until time step t. EE becomes maximal for the actions that have low weights but are also scarce. Each action should satisfy some constraints to be applicable such as range checks and distance between thresholds. The control mechanism embedded in our algorithm marks inapplicable actions with a setting so that they can never be chosen.
8. *Local e-Greedy*. Local action selection in each rule will be done by a kind of e-Greedy strategy, selects all rules' candidates according to their EE values. Equation (9) formulizes this selection:

$$U_t^i = U^i(k) \left| \mathrm{EE}\left(U^i(k)\right) = \max_{a \in U^i} \mathrm{EE}(a) \right. \tag{9}$$

9. *Updating FIS*. This task is taken over by tuning action qualities, as equation (10) shows:

$$w_{t+1}^i\left(U_t^i\right) = w_t^i\left(U_t^i\right) + \tilde{\varepsilon}_{t+1} \times \alpha_{R_i}, \forall R_i \in A(X_t) \tag{10}$$

10. *Double e-Greedy action selection*. After having selected all rule candidates, in the next higher competition layer the total double e-Greedy or maximum e-Greedy action will be determined by a pure greedy approach based on the rules' truth values and the nominated actions' EE from previous steps. This procedure is described in equation (11):

$$U_t(X_t) = U_t^{i*} \left| \mathrm{EE}\left(U_t^{i*}\right) \times \alpha_{R_i}(X_t) = \max_{R_i \in A(X_t)} \left(\mathrm{EE}\left(U_t^i\right) \times \alpha_{R_i}(X_t)\right) \right. \tag{11}$$

DOI: 10.1002/nem

For each packet arrival:
Gather information from environment
Fuzzify the inputs and create rule base
Explore actions to find potential best actions
Find best action due to learnt traffic model
Update Q-learning tables
Commit chosen action
Calculate the average queue size q_{avg}
If $min_h \le q_{avg} < max_{th}$
Calculate probability p_a
With probability p_a: mark/drop the arriving packet
Else if $max_{th} \le q_{avg}$
Mark/drop the arriving packet

Figure 4. Pseudo-code of FQL-RED algorithm.

11. *Estimation of current Q-value*. Finally, this phase tends to computing and memorizing the Q-value of the current state–action pair based on the new Q-value function after tuning action weight parameter values through equation (12):

$$\tilde{Q}_t(X, U_t(X_t)) = \frac{\sum_{R_i \in A(X_t)} \alpha_{R_i}(X_t) \times w_t^i(U_t^i)}{\sum_{R_i \in A(X_t)} \alpha_{R_i}(X_t)} \tag{12}$$

12. *Actuation*. The FQL algorithm functionality is independent of the chosen actions, so it is up to network administrators to choose these actions. These actions perform a slight modification to all parameters: changing min_{th}, max_{th}, max_p and *exponential weight factor*. As advised in Xu *et al.* [36] the actions concerning setting max_p are using AIMD instead of MIMD strategy. Also the changes in max_{th} and min_{th} should be small enough to avoid oscillation in queue size. Note that *exponential weight factor* is a substitute for w_q in this implementation, as suggested by OPNET documentation [67], and is used to calculate w_q as in equation (13) to provide more separation between values of w_q, giving it meaningful changes:

$$w_q = 2^{-\frac{1}{\exp^w}} \tag{13}$$

In summary, the FQL-RED algorithm added some phases to the RED algorithm as mentioned, but did not change its normal procedure, as marked in bold in Figure 4.

4. SIMULATION AND RESULTS

In order to evaluate the performance of FQL-RED a simulation by OPNET is arranged and two different scenarios are employed. In both scenarios a common network configuration, often called Dumbbell topology [31], is used to evaluate the performance of AQM schemes, as shown in Figure 5. This topology imitates a subnet facing congestion, with two routers and a link connecting them that plays the role of the traffic bottleneck. Router 1 tries to transfer the traffic via this congested link to Router 2, so the output interface of Router 1 is where the AQM scheme should be deployed. In the first scenario, FQL-RED is compared against RED for a time-invariant HTTP traffic, and in the second scenario the performance of FQL-RED challenges ARED and FC-RED with a time-varying mixed traffic.

4.1. Scenario 1: time-invariant traffic

The goal of this experiment is to observe the behavior of FQL-RED in a time-invariant traffic and compare it with RED results. The proposed algorithm tries to achieve optimal policy, maximizing throughput while minimizing average delay, using its well-defined actions. With time-invariant traffic,

DOI: 10.1002/nem

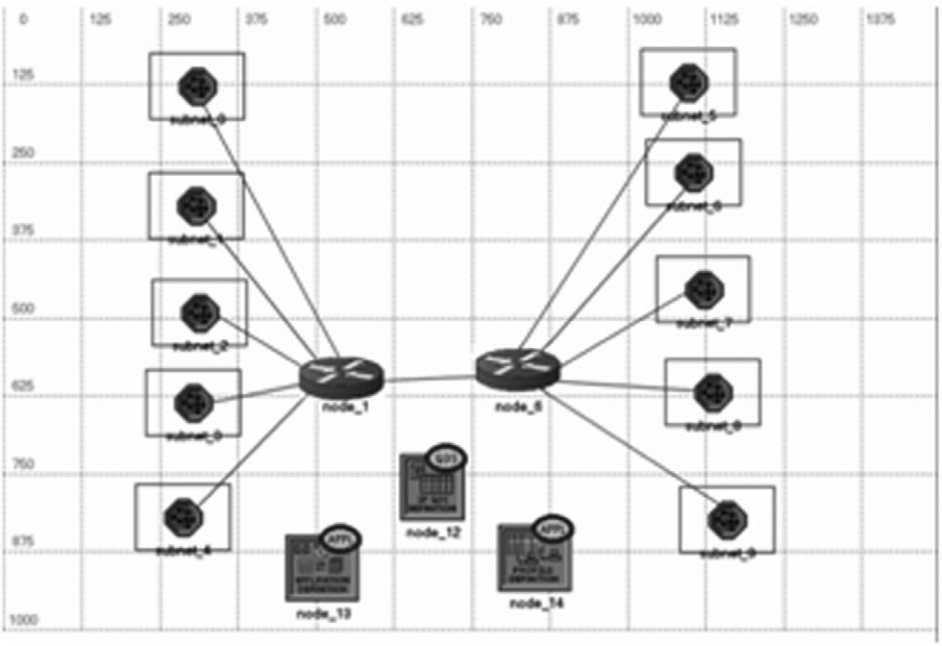

Figure 5. OPNET network model: bottleneck configuration known as dumbbell topology.

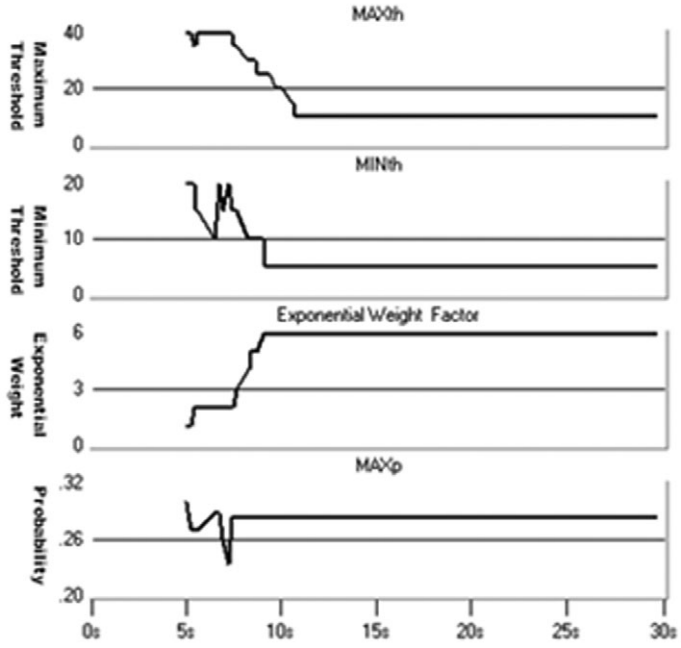

Figure 6. Change of parameters during simulation of Scenario 1 by FQL-RED.

FQL-RED is expected to set RED parameters in a way that fully improves throughput and delay using a trial-and-error approach and embedded exploration/exploitation mechanism. Figure 6 demonstrates the procedure of parameter adjustment by FQL-RED. As is clear in this graph, all parameters converge to a constant value after a few episodes. This occurs because when the parameter reaches a suitable value, the received reward from the environment increases and the estimated Q-value of the 'no-operation' action dominates other values. Table 1 shows the initial values of RED and FQL-RED parameters.

DOI: 10.1002/nem

Table 1. RED and FQL-RED parameters: initial configurations on R1 in Scenario 1.

Parameter	Value
Minimum threshold (min_{th})	20
Maximum threshold (max_{th})	40
Maximum value for Pb (max_p)	0.3
Exponential weight factor	1.0
Buffer size	100

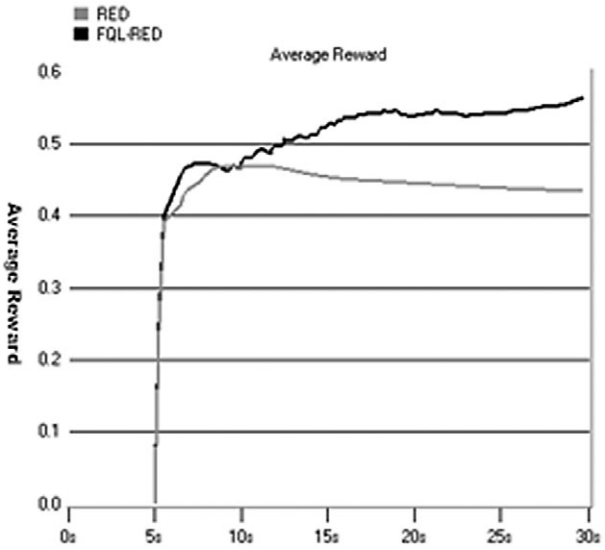

Figure 7. Comparison of average reward of RED and FQL-RED in Scenario 1.

The learning process is expected to increase the average of the reward signal. In the next experiment, the reinforcement signal used in FQL-RED is calculated for RED but no learning process is employed. This will help us to monitor the effect of the learning process and the reason for superiority of FQL-RED over RED. Figure 7 depicts this comparison.

Figure 8 demonstrates the average of queuing delay (Figure 8(a)) and throughput (Figure 8(b)) of FQL-RED and RED, while Figure 8(c) shows the average queue length during this scenario. In this implementation with regard to the definition of reward signal in the previous section, reinforcement is considered as summation of throughput and queuing delay with equal coefficients. The choice of coefficient indicates the importance of each factor and can be biased towards delay or throughput regarding the quality of service that the network plans to provide.

As Figure 8(c) shows, adjusting the parameters of RED using FQL stabilizes average queue size, leading to improved delay and throughput. The resulting adjustment may not as familiar for a domain expert but, as simulations show, it performs well.

4.2. Scenario 2: time-varying mixed traffic

This simulation uses a time-varying mixed traffic and is intended to study the learning ability of FQL-RED that enables it to react quickly and appropriately to sudden changes in traffic status. Although the proposed algorithm adjusts parameters based on trial and error for unseen or unfamiliar cases, in the case of long-term traffic there exist plenty of repeated or partly similar states. Thus the slightly slow convergence of the algorithm in the early stages is compensated by rapid optimal action selection in the later stages.

DOI: 10.1002/nem

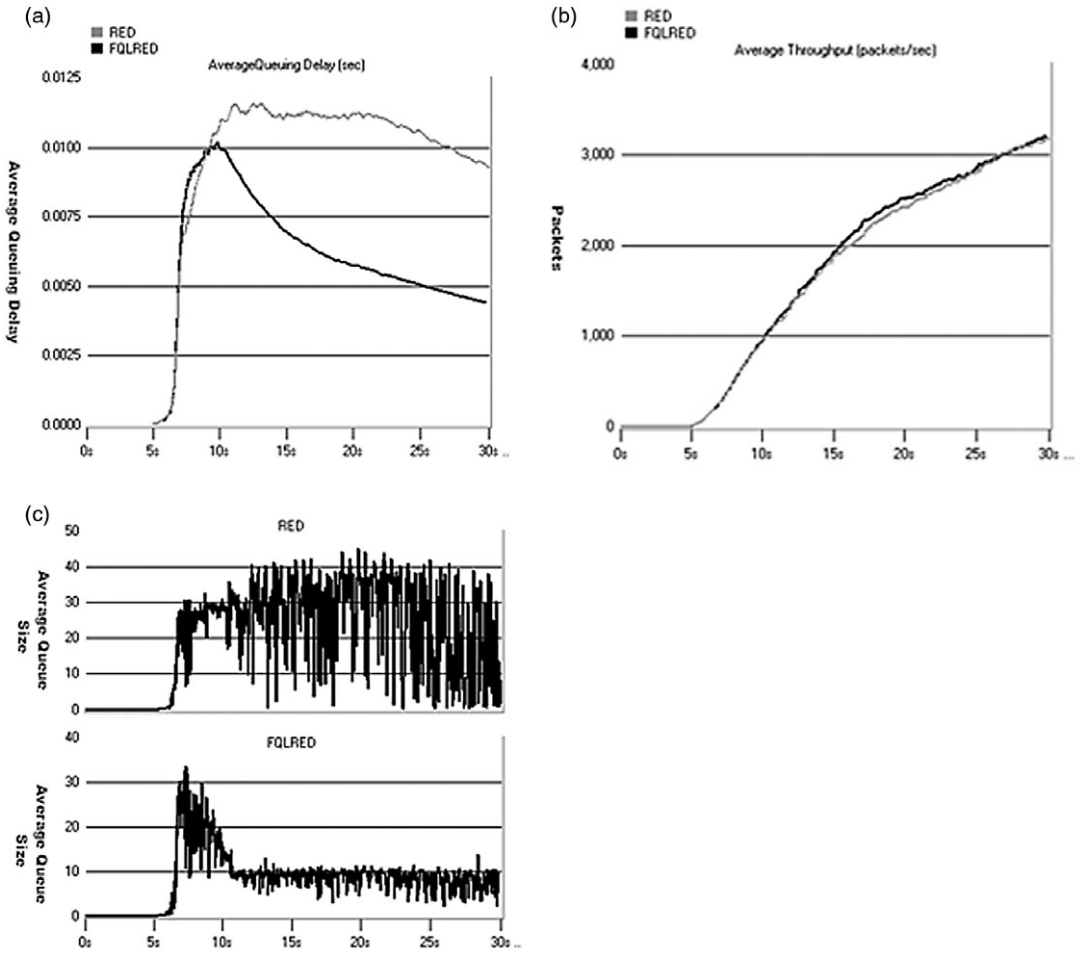

Figure 8. (a) Average queuing delay; (b) average throughput; and (c) average queue size of FQLRED and RED during simulation of Scenario 1.

In this scenario the FQL-RED is tested against ARED and FC-RED. The latter algorithms only adjust $\max_p$, but the former aims to adjust all RED parameters, because an AQM scheme can handle a situation by slightly changing a specific parameter rather than massively changing another one, as stated before. For example, a small change in $\max_{th}$ can have the same effect as large changes in $\max_p$ on the throughput and delay results. Not being limited to a single choice not only gives the algorithm more flexibility but also increases its speed in handling various traffic situations. We claim that our algorithm reacts faster and more accurately than RED, ARED, FC-RED, and theoretically other AQM schemes without the capability of learning. Table 2 shows initial values of RED, FQL-RED, ARED and FC-RED parameters.

Figure 9 describes the parameter tuning during simulation of the scenario, and this continuous change justifies why FQL-RED does not converge to a fixed value for parameters in this time period.

Figure 10 illustrate the superiority of FQL-RED against the other three algorithms and it is evident that it can improve delay (Figure 10(a)) and throughput (Figure 10(b)) better, maintaining a fairly suitable queue size under different conditions, as Figure 10(c) shows.

The goal of FQL-RED in these scenarios is to achieve optimal policy. The optimal policy is a sequence of actions leading to increased throughput while simultaneously lowering queuing delay. A so-called network traffic is a set of traffic conditions (i.e. states in the FQL algorithm) which possibly occur in a network, independent of the application that caused it. The main reason for using machine learning in this problem is to obtain a prediction, so that FQL-RED algorithm will be able to predict

DOI: 10.1002/nem

Table 2. RED, FQL-RED, ARED and FC-RED parameters: initial configurations on R1 in Scenario 2.

Parameter	Value
Minimum threshold (min_{th})	20
Maximum threshold (max_{th})	60
Maximum value for P_b (max_p)	0.1
Exponential weight factor	1.0
α (in ARED algorithm)	0.01
β (in ARED algorithm)	0.9
Q_T (in FC-RED algorithm)	40
Buffer size	100

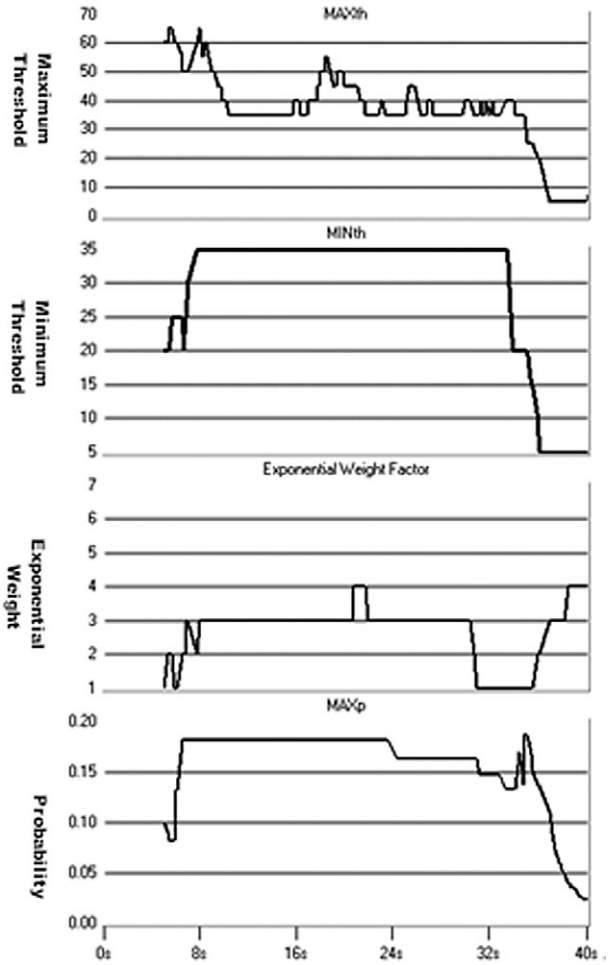

Figure 9. Change of parameters during simulation of Scenario 2 by FQL-RED.

the next traffic state while observing previously seen traffic conditions and provide the router with the optimal action to catch up with the goal of the system: higher throughput with lower queuing delay. Application independence, periodic and monolithic characteristics of traffic in the real world, and input signal fuzzification assure a finite set of traffic states encapsulated in a fuzzy rule base, which in turn assures convergence of the system to the optimal solution. The observed difference in queue length of RED and FQL-RED algorithms is due to actions chosen regarding traffic conditions by FQL-RED. The

DOI: 10.1002/nem

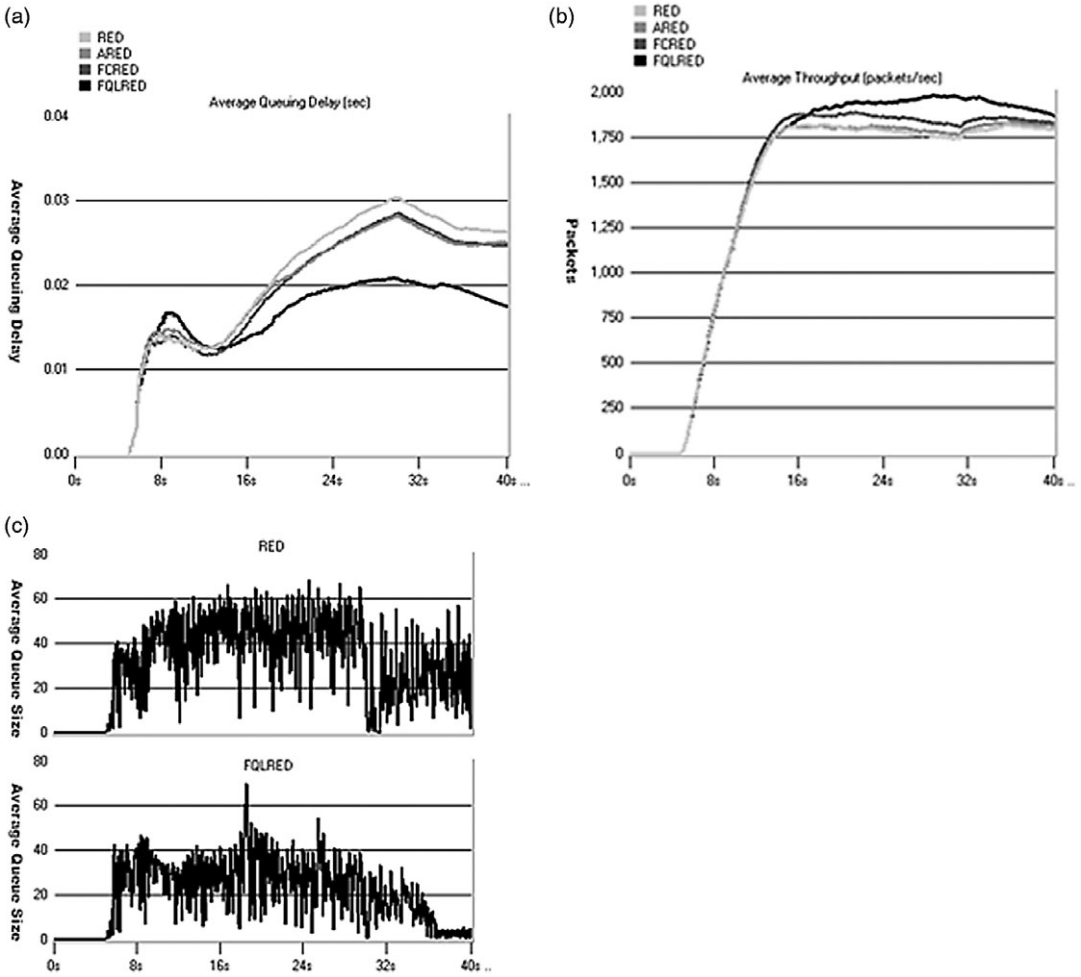

Figure 10. (a) Average queuing delay; (b) throughput; and (c) average queue length of FQL-RED and RED during simulation of Scenario 2.

key advantage of FQL-RED is improving RED performance without predefined parameters or any prior knowledge of traffic shape. Regarding this issue, performance is defined in terms of higher throughput and lower delay simultaneously.

5. CONCLUSION

This paper proposed an add-on to the original RED, granting adaptation by means of fuzzy Q-learning to address the main issue of this algorithm, which is parameter sensitivity. The resulting FQL-RED improves the performance of RED and adjusts these parameters in long-term traffic to optimally balance two opposing factors of a gateway: throughput and queuing delay. This algorithm is categorized as a self-adaptive one, inheriting this property from an artificial intelligence technique. Reinforcement learning and specifically Q-learning is a good choice because of the dynamic and episodic nature of network traffic. On the other hand, we enhance our learning mechanism with a Takagi–Sugeno fuzzy inference system to provide our solution with two major benefits: in the first place, to map a large state space to super-states with a reasonable number that covers a particular situation of the network traffic; secondly to embed expert knowledge into the scheme. The former means that the search space of reinforcement is small enough to ease convergence and the latter assures

DOI: 10.1002/nem

that the network starts from a good initial setting and uses the experience of a network expert (e.g. administrator) in decision making about packets. Note that in the worst case FQL-RED performs like a RED gateway with imperfectly adjusted parameters. Also FQL-RED learns traffic patterns which lead to congestion, and adapts parameters to effectively avoid congestion. This mechanism provides the algorithm with several other benefits such as independence to the number of active TCP connections, active adaptation to change in network topology or quality of service required, stable handling of bursty traffic, and so on. Furthermore, with the assistance of FIS, the system is provided with scalability. Finally, it is shown that FQL-RED outperforms RED, ARED, and FC-RED with regard to design goals (higher throughput with less delay).

Using more inputs, expanding the fuzzy inference system with advanced expert knowledge and other fuzzification mappings, improving the machine learning component, integrating a total solution of all RED defects into the learning process, e.g. fairness and misbehaving connections, by expanding the reinforcement signal, changing the combination form of factors in this signal, and generating standard congestion scenario benchmarks to compare this algorithm with other schemes are some guidelines to enlighten future studies in this field.

REFERENCES

1. Floyd S, Jacobson V. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking* 1993; **1**: 397–413.
2. Dijkstra S. *Modeling active queue management algorithms using stochastic Petri nets*. Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 2004; 79.
3. Su L, Hou JC. *An active queue management scheme for Internet congestion control and its application to differentiated services*. Ohio State University, Columbus, OH, January 2000.
4. Lin D, Morris R. Dynamics of random early detection. In *Computer Communication Review: Proceedings of ACM SIGCOMM 1997*, 1997; 127–137.
5. Teunis JO, Lakshman TV, Wong LH. SRED: stabilized RED. In *Proceedings of IEEE INFOCOM 1999*, March 1999; 1346–1355.
6. Pan R, Prabhakar B, Psounis K. CHOKe: a stateless active queue management scheme for approximating fair bandwidth allocation. In *Proceedings of IEEE INFOCOM 2000*, April 2000; 942–951.
7. Seol JH, Lee KY, Hong YS. Performance improvement of adaptive AQM using the variation of queue length. In *IEEE Region 10 Conference TENCON 2006*, 2006; 1–4.
8. Yamaguchi T, Takahashi Y. A queue management algorithm for fair bandwidth allocation. *Computer Communications* 2007; **30**: 2048–2059.
9. Feng W, Shin KG, Kandlur DD, Saha D. The BLUE active queue management algorithms. *IEEE/ACM Transactions on Networking* 2002; **10**: 513–528.
10. Koo J, Song B, Chung K, Lee H, Kahng H. MRED: a new approach to random early detection. In *15th International Conference on Information Networking*, 2001; 347–352.
11. Kamra A, Kapila S, Khurana V, Yadav V, Shorey R, Saran H, Juneja S. SFED: a rate control based active queue management discipline. *IBM India Research Laboratory Research Report*, 2000.
12. Kamra A, Saran H, Sen S, Shorey R. Fair adaptive bandwidth allocation: a rate control based active queue management discipline. In *Computer Networks* 2004; **44**: 135–152.
13. Long C, Zhao B, Guan X, Yang J. The yellow active queue management algorithm. *Computer Networks* 2005; **47**: 525–550.
14. Agarwal MK, Gupta R, Kargaonkar V. Link utilization based AQM and its performance. In *IEEE Global Telecommunications* Conference, Vol. 2, 2004.
15. Wang C, Li B, Hou YT, Sohraby K, Long K. A stable rate-based algorithm for active queue management. *Computer Communications* 2005; **28**: 1731–1740.
16. Parris M, Jeffay K, Smith FD. Lightweight active router-queue management for multimedia networking. in *Multimedia Computing and Networking: SPIE Proceedings Series*, January 1999.
17. Chung J, Claypool M. Dynamic-CBT and ChIPS: router support for improved multimedia performance on the Internet. In *Proceedings of ACM Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 2000; 239–248.
18. Claypool M, Kinicki R, Hartling M. Active queue management for web traffic. In *IEEE International Conference on Performance, Computing and Communication*, 2004; 531–538.
19. Hollot CV, Misra V, Towsley D, Gong WB. On designing improved controllers for AQM routers supporting TCP flows. In *Proceedings of IEEE INFOCOM 2001*, April 2001; 1726–1734.
20. Aweya J, Ouellette M, Montuno DY. A control theoretic approach to active queue management. *Computer Networks* 2001; **36**: 203–235.
21. Kunniyur SS, Srikant R. An adaptive virtual queue (AVQ) algorithm for active queue management. *IEEE/ACM Transactions on Networking* 2004; **12**: 286–299.
22. Long CN, Zhao B, Guan XP. SAVQ: stabilized adaptive virtual queue management algorithm. *IEEE Communications Letters* 2005; **9**: 78–80.

DOI: 10.1002/nem

23. Yanping Q, Qi L, Xiangze L, Wei J. A stable enhanced adaptive virtual queue management algorithm for TCP networks. In *IEEE International Conference on Control and Automation*, 2007; 360–365.
24. Chen CL, Yu JC. A proportional rate-based control scheme for active queue management. In *IEEE International Conference on Electro Information Technology*, 2005; 6.
25. Lapsley D, Low S. Random early marking for Internet congestion control. *IEEE Transactions on Networks* 1999; **3**: 1747–1752.
26. Deng X, Yi S, Kesidis G, Das CR. Stabilized virtual buffer (SVB): an active queue management scheme for Internet quality-of-service. In *IEEE Global Telecommunications Conference*, Vol. 2, 2002.
27. Sun J, Zukerman M. RaQ: a robust active queue management scheme based on rate and queue length. *Computer Communications* 2007; **30**: 1731–1741.
28. Feng W, Kapadia A, Thulasidasan S. GREEN: proactive queue management over a best-effort network. In *IEEE GLOBECOM*, November 2002.
29. Ryu S, Rump C, Qiao C. Advances in active queue management (AQM) based TCP congestion control. *Telecommunication Systems* 2004; **25**: 317–351.
30. Thiruchelvi G, Raja J. A survey on active queue management mechanisms. *International Journal of Computer Science and Network Security* 2008; **8**: 130–145.
31. Bitorika A, Robin M, Huggard M, McGoldrick C. A comparative study of active queue management schemes. In *Proceedings of IEEE ICC 2004*, Vol. 201, 2004; 6.
32. Floyd S, Gummadi R, Shenker S. Adaptive RED: an algorithm for increasing the robustness of RED's active queue management, http://www.icir.org/floyd/papers.html [August 2001].
33. Iannaccone G, May M, Diot C. Aggregate traffic performance with active queue management and drop from tail. *ACM SIGCOMM Computer Communication Review* 2001; **31**: 4–13.
34. Wang C, Li B, Hou YT, Sohraby K, Lin Y. LRED: a robust active queue management scheme based on packet loss ratio. In *IEEE INFOCOM*, 2004; 1–12.
35. Liu S, Basar T, Srikant R. Exponential-RED: a stabilizing AQM scheme for low-and high-speed TCP protocols. *IEEE/ACM Transactions on Networking* 2005; **13**: 1068–1081.
36. Xu YD, Wang ZY, Wang H. ARED: a novel adaptive congestion controller. In *International Conference on Machine Learning and Cybernetics*, Vol. 2, 2005.
37. Joo C, Bahk S, Lumetta SS. A hybrid active queue management for stability and fast adaptation. *Journal of Communication and Networks* 2006; **8**: 93–105.
38. Subasree S. Fuzzy DS RED: an intelligent active queue management scheme for TCP/IP Diff-Serv. In *International Conference on Computational Intelligence*, Istanbul, Turkey, 2004.
39. Wang C, Li B, Sohraby K, Peng Y. AFRED: an adaptive fuzzy-based control algorithm for active queue management. In *28th Annual IEEE International Conference on Local Computer Networks*, 2003; 12–20.
40. Sun J, Zukerman M, Palaniswami M. Stabilizing RED using a fuzzy controller. In *IEEE International Conference on Communications*, 2007; 266–271.
41. Sun J, Zukerman M. Improving RED by a neuron controller. In *Managing Traffic Performance in Converged Networks*, Vol. 4516. Springer: Berlin, 2007; 434.
42. Sun J, Chan S, Ko KT, Chen G, Zukerman M. Neuron PID: a robust AQM scheme. In *Proceedings of ATNAC*, 2006; 259–262.
43. Jahanshahi M, Meybodi MR. An adaptive congestion control method for guaranteeing queuing delay in RED-based queue using learning automata. In *International Conference on Mechatronics and Automation*, 2007; 3360–3365.
44. Wang X, Wang Y, Zhou H, Huai X. PSO-PID: a novel controller for AQM routers. In *International Conference on Wireless and Optical Communications Networks*, 2006; 5.
45. Jun-song W, Zhi-wei G, Yan-tai S. RBF-PID based adaptive active queue management algorithm for TCP network. In *IEEE International Conference on Control and Automation*, 2007; 171–176.
46. Yanfei F, Fengyuan R, Chuang L. Design of an active queue management algorithm based fuzzy logic decision. In *International Conference on Communication Technology Proceedings*, 2003; 286–289.
47. Chen CK, Kuo HH, Yan JJ, Liao TL. GA-based PID active queue management control design for a class of TCP communication networks. *Expert Systems with Applications* 2007; **36**: 1903–1913.
48. Di Fatta G, Hoffmann F, Lo Re G, Urso A. A genetic algorithm for the design of a fuzzy controller for active queue management. *IEEE Transactions on Systems, Man and Cybernetics* 2003; **33**: 313–324.
49. Fan Y, Jiang Z, Panwar S. An adaptive control scheme for stabilizing TCP. In *5th World Congress on Intelligent Control and Automation*, Hangzhou, China, 2004.
50. Qiang C, Yang OWW. A ST-PI-PP controller for AQM router. In *IEEE International Conference on Communications*, 2004; 2277–2281.
51. Wang H, Meng B, Jing Y, Liu X. An adaptive fuzzy sliding mode control for AQM systems. In *American Control Conference*, 2008; 4445–4450.
52. Feng WC, Kandlur DD, Saha D, Shin KG. A self-configuring RED gateway. In *Proceedings of IEEE INFOCOM'99*, 1999.
53. Nyirenda CN, Dawoud DS. Multi-objective particle swarm optimization for fuzzy logic based active queue management. In *IEEE International Conference on Fuzzy Systems* 2006; 2231–2238.
54. Zhani MF, Elbiaze H, Kamoun F. SNFAQM: an active queue management mechanism using neurofuzzy prediction. In *12th IEEE Symposium on Computers and Communications*, 2007; 381–386.
55. Hariri B, Sadati N. NN-RED: an AQM mechanism based on neural networks. *Electronics Letters* 2007; **43**: 1053–1055.

DOI: 10.1002/nem

56. Floyd S. Recommendations on the use of the gentle variant of RED (2000). http://www.icir.org/floyd/red/gentle.html [12 June 2010].
57. Zhou K, Yeung KL, Li VOK. Nonlinear RED: a simple yet efficient active queue management scheme. *Computer Networks* 2006; **50**: 3784–3794.
58. Hu L, Kshemkalyani AD. HRED: a simple and efficient active queue management algorithm. In *13th International Conference on Computer Communications and Networking ICCCN 2004*, 2004; 387–393.
59. Chrysostomou C, Pitsillides A, Rossides L, Sekercioglu A. Fuzzy logic controlled RED: congestion control in TCP/IP differentiated services networks. *Control Engineering Practice* 2003; **8**: 79–92.
60. Hadjadj Aoul Y, Mehaoua A, Skianis C. A fuzzy logic-based AQM for real-time traffic over internet. *Computer Networks* 2007; **51**: 4617–4633.
61. Morris R. Scalable TCP congestion control. In *IEEE INFOCOM 2000*, Vol. 3, Tel Aviv, Israel, 2000; 1176–1183.
62. Hashem E. Analysis of random drop for gateway congestion control. *Report LCS TR-465*, Laboratory for Computer Science, MIT, Cambridge, MA, 1989.
63. Zheng B, Atiquzzaman M. *DSRED: An Active Queue Management Scheme for Next Generation Networks*. IEEE Computer Society: Washington, DC, 2000; 242.
64. Sutton RS, Barto AG. *Reinforcement learning: An Introduction*. MIT Press: Cambridge, MA, 1998.
65. Berenji HR. Refinement of approximate reasoning-based controllers by reinforcement learning. In *Proceedings of the 8th International Workshop Machine Learning*, 1990; 475–479.
66. Naeeni AF. *Advanced multi-agent fuzzy reinforcement learning*. Master's thesis, Dalarna University, 2004.
67. OPNET Technologies (2004). http://www.opnet.com [12 June 2010].

AUTHORS' BIOGRAPHIES

Seyed Saeid Masoumzadeh was born in Qazvin, Iran. He received B.Sc. degree in computer hardware from Meybod Azad University and M.Sc. degree in artificial intelligence from Qazvin Azad University in 2004 and 2009, respectively. His research interests are intelligent and self-adaptive control, reinforcement learning, quality of service management, and image mining.

Kourosh Meshgi, born in Tehran, is a graduate student in robotics research lab in Amirkabir University of Technology, Tehran, Iran under supervision of Dr. Shiry. He also attended in this university before and earned his Bachelor's degree (2008) in hardware engineering working on intelligent network hardware. His research interests include computer vision, machine learning, robotics and cognitive neuroscience and he's currently working on reinforcement learning and brain derived vision.

Saeed Shiry Ghidary was born in Zanjan, Iran. He received his B.Sc. degree in Electronic engineering and M.Sc. in computer architecture from Amirkabir University of Technology in 1990 and 1994 respectively. He studied Robotics and artificial intelligent systems in Kobe University and got his PhD in 2002. He is an assistant Prof. at Amirkabir University of Technology since 2004. His research interests include intelligent robotics, machine learning, machine vision, cognitive science and brain modeling.

Gelareh Taghizadeh was born in Qazvin, Iran. She received her B.Sc. degree in computer hardware and M.Sc. degree in artificial intelligence from Qazvin Azad University in 2006 and 2010 respectively. She is currently a lecturer at University of Applied Science and Technology. Her research interests are intelligent optimization, natural inspired algorithms and artificial immune systems.

DOI: 10.1002/nem