Neural Comput & Applic
DOI 10.1007/s00521-016-2730-9

ORIGINAL ARTICLE

FLRED: an efficient fuzzy logic based network congestion control method

Mosleh M. Abualhaj[1] · Ahmad Adel Abu-Shareha[2] · Mayy M. Al-Tahrawi[1]

Received: 20 June 2016 / Accepted: 19 November 2016

Abstract The number of applications running over computer networks has been increasing tremendously, which increased the number of packets running over the network as well leading to resource contention, which ultimately results in congestion. Congestion increases both delay and packet loss while reducing bandwidth utilization and degrading network performance. Network congestion can be controlled by several methods, such as random early detection (RED), which is the most well-known and widely used method to alleviate problems caused by congestion. However, RED and its variants suffer from linearity and parametrization problems. In this paper, we proposed a new method called fuzzy logic RED (FLRED), which extends RED by integrating fuzzy logic to overcome these problems. The proposed FLRED method relies on the average queue length (***aql***) and the speculated delay ($\boldsymbol{D_{Spec}}$) to predict and avoid congestion at an early stage. A discrete-time queue model is used to simulate and evaluate FLRED. The results showed that FLRED outperformed both RED and effective RED (ERED) by decreasing both delay and packet loss under heavy congestion. Compared with ERED and RED, FLRED decreased the delay by up to 1.5 and 4.5% and reduced packet loss by up to 6 and 30%, respectively, under heavy congestion. These findings suggest that FLRED is a promising congestion method that can save network resources and improve overall performance.

Keywords Congestion control · Network performance · Active queue management · Fuzzy logic · RED

1 Introduction

Computer networks have propagated worldwide, starting from home networks to multi-branches organization networks. Huge amounts of data are exchanged among network users in the form of packets. In their journey from source to destination, packets travel over several network links and switches/routers [1]. When numerous sources transmit over the same intermediate link, packets will be queued in the routers' buffer and wait for their turn to be transmitted. However, in view of the limitations in network resources (link capacity and buffer size), new incoming packets will be dropped once the number of packets exceeds the resources capacity. Figure 1 shows a potentially congested router's buffer. A network wherein packet dropping occurs frequently is considered a congested network [1–3].

Tail-drop method is the default behavior of computer networks in managing congestion at the router's buffer. This method employs a first-in first-out (FIFO) approach, in which arriving packets are dropped once the buffer becomes completely full. Figure 2 illustrates a tail-drop buffer. When packets arrive regularly, tail drop is considered an efficient method of exploiting a buffer, because it ensures that all slots at the buffer are used [3, 4]. However, most computer network traffic arrives irregularly (i.e., bursty traffic), which involves the quick transmission of

✉ Mosleh M. Abualhaj
m.abualhaj@ammanu.edu.jo

Ahmad Adel Abu-Shareha
aabushareha@meu.edu.jo

Mayy M. Al-Tahrawi
mtahrawi@ammanu.edu.jo

[1] Faculty of Information Technology, Al-Ahliyya Amman University, Amman, Jordan

[2] Faculty of Information Technology, Middle East University, Amman, Jordan

Published online: 29 November 2016

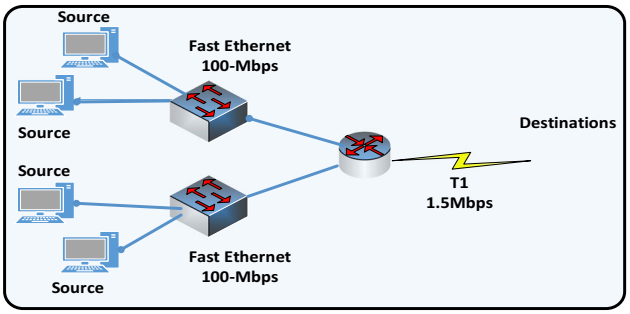

Fig. 1 Potential congested router's buffer

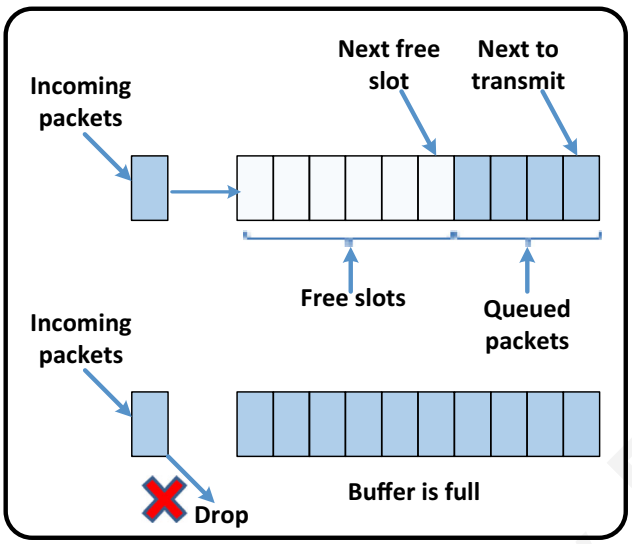

Fig. 2 Tail-drop buffer

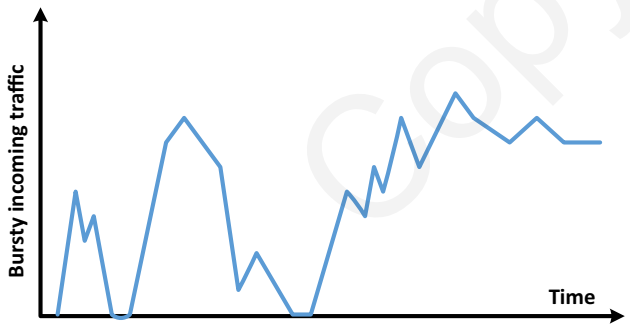

Fig. 3 Bursty traffic

data blocks followed by idle periods [3, 5, 6]. Figure 3 depicts the bursty traffic of a computer network.

Given the bursty traffic nature of computer networks, queues are rapidly filled and then emptied again. Bursty traffic causes several problems, particularly when a tail-drop buffer is used. These problems include: (1) increased packet loss resulting from the overwhelmed buffer under heavy load; (2) increased packet transmission delay because of the long queue at the router buffer under heavy load; (3) wasted link bandwidth caused by the repeated retransmission of dropped packets; (4) possible network collapse on prolonged periods of congestion; (5) increased chances of transmission control protocol (TCP) global synchronization when considerable amounts of packets are dropped at the same instant; consequently, hosts simultaneously decrease their sending rates while increasing their sending rates, leading to cyclical periods of heavy load followed by periods of poor utilization of a link capacity; and (6) TCP starvation problem caused by the TCP global synchronization problem; that is, when hosts reduce their TCP flow rates during congestion, non-TCP flows occupy the buffer, leaving no place for TCP flows [7–11].

Congestion control is the term used to describe the efforts to predict and avoid congestion. Unfortunately, tail drop causes several problems. Active queue management (AQM) methods are proposed as a robust alternative for controlling the congestion at the route's buffer. AQM methods generally predict congestion at an early stage by monitoring the queue at the buffer. When the queue at the buffer develops and congestion is about to occur, AQM methods randomly drop incoming packets on the calculated dropping probability ***Dp***. In response, hosts lessen their sending rates before the buffer is saturated. Consequently, AQM methods reduce the delay and packet loss resulting from the saturated buffer and alleviate the impact TCP global synchronization and TCP starvation by dropping packets randomly [3, 12–14]. Typical AQM methods include random early detection (RED) [5], gentle RED (GRED) [15], dynamic GRED (DGRED) [13], effective RED (ERED) [16]. Other AQM methods utilize fuzzy logic, such as fuzzy explicit marking (FEM) [17], RED fuzzy logic (REDFL) [18], and the intelligent rate (IntelRate) controller [12]. However, existing AQM methods cannot efficiently handle or predict the congestion early enough, which degrades network performance [2, 13, 14]. This paper proposes a new method that can predict congestion at an early stage to overcome problems resulting from the tail drop by improving network performance (e.g., delay and packet loss) and to alleviate the effects of TCP global synchronization and TCP starvation. The proposed method achieves this goal by integrating fuzzy logic with two congestion prediction factors, namely speculation delay and average queue length (***aql***). The detailed contributions of this method are discussed in Sect. 3.

The remainder of this paper is organized as follows. Section 2 clarifies the congestion problem, discusses the main methods for congestion control, and enumerates the key reasons for designing a new AQM method. Section 3 describes the proposed method and discusses how it predicts congestion at an early stage. Section 4 shows the environment in which the proposed method is implemented and evaluated. Section 5 presents the analyses of the

performance results of the proposed method. Finally, Sect. 6 concludes the paper.

2 Related works and problem definition

Congestion control is a major issue in computer networks investigated by many researchers since computer networks emerged. This section discusses the main and most widely known congestion control methods related to this work.

RED [5] is the first method established to manage congestion control issues. Floyd and Jacobson [6] proposed RED in 1993 to overcome the aforementioned problems arising from the traditional tail-drop method. Similar to other AQM methods, the core idea of RED is to predict the congestion before a buffer overflows by calculating the ***aql*** of the router buffer using the weighted queue (***Wq***). Packets are then dropped before the buffer overflows according to the predetermined ***Dp*** value. The ***aql*** is compared to two thresholds at the router's buffer (min and max), and the packets are then dropped according to the following three rules. (1) If ***aql*** $<$ min, then no packets are dropped. (2) If min $<$ ***aql*** $<$ max, then packets are dropped based on a certain equation. (3) If ***aql*** $>$ max, then all incoming packets are dropped. In this manner, RED drops packets proportionally to ***aql***. In addition, packets are dropped randomly, making RED a suitable solution for bursty traffic. These findings suggest that RED avoids the global synchronization and TCP starvation problems that resulted from the tail-drop method. In addition, compared with the tail drop, RED exhibits a higher throughput and a lower delay and packet loss. Figure 4 illustrates the RED buffer.

GRED [15] is another method proposed by Floyd in 2000 to address network congestion. Similar to RED, GRED predicts the congestion of the router's buffer at an early stage before the buffer overflows by calculating the ***aql***. Unlike RED, GRED uses three thresholds [min, max, and double max ($2 * \max$)]. Therefore, the rules for packet dropping are as follows. (1) If ***aql*** $<$ min, then no packets are dropped. (2) If min $<$ ***aql*** $<$ max, then packets are dropped based on a certain equation. (3) If max $<$ ***aql*** $<$ double max, then packets are dropped based on a certain equation but with a higher probability than rule two. (4) If ***aql*** $>$ double max, then all incoming packets are dropped. The simulation showed that GRED outperforms RED because of the additional threshold (double max). Baklizi et al. [13] extended GRED and developed DGRED in 2013. Both methods follow the same dropping rules and core strategy of using the same three thresholds in GRED. However, unlike GRED, DGRED uses dynamic max and double-max thresholds to keep the ***aql*** between the min and max thresholds at a specific calculated value called target ***aql*** (***Taql***). Performance analysis showed that DGRED outperforms both RED and GRED.

As shown, RED, GRED, and DGRED depend on the average size (***aql***), rather than the instantaneous size of the queue to calculate ***Dp***. Thus, these algorithms respond to the long-term status, instead of the current status of the buffer. On the one hand, when the incoming traffic suddenly decreases, the instantaneous queue length instantly declines as the ***aql*** gradually lessens. If the ***aql*** is high, then incoming packets are dropped even if the instantaneous queue length is small or the buffer is empty. On the other hand, when the incoming traffic suddenly increases, the instantaneous queue length instantly rises although the ***aql*** gradually increases. Consequently, the queue becomes overwhelmed, and no packet is dropped because the minimal ***aql*** (less than the min threshold) [5, 16, 19]. Figure 5 depicts the ***aql*** versus the instantaneous queue length.

To solve these issues, several algorithms have used the instantaneous queue (***q***) size to predict congestion. Ott et al. [20] established stabilized RED (SRED), which uses the

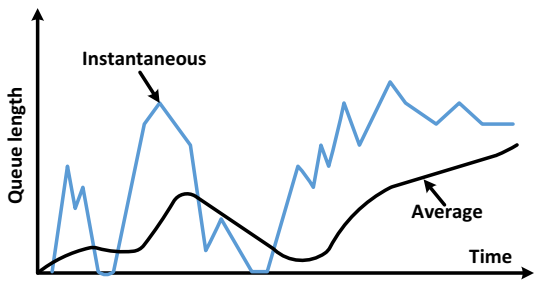

Fig. 5 ***aql*** versus ***q*** length

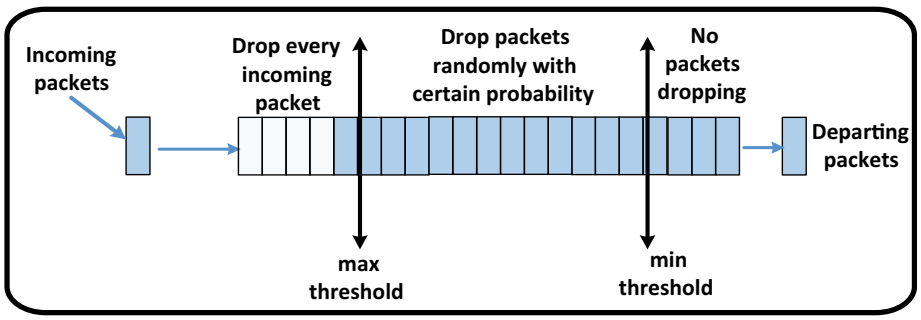

Fig. 4 RED method buffer

number of flows and q to detect and avoid congestion. If the q size exceeds a specific level, packets are dropped according to the probability calculated from the number of flows to avoid congestion. Thus, the packet dropping is directly proportional to the number of flows. Jamali et al. [21] proposed another q-based algorithm in 2014. The algorithm proposed is an improvement of RED, and it adaptively and proportionally updates RED's thresholds to the current q size. The remaining parts of the proposed algorithm are the same as the original RED. As shown, these algorithms depend on the q instead of the aql. Therefore, they respond to the current rather than the long-term status of the buffer, and they can successfully predict congestion and avoid the aforementioned problems. However, as mentioned, internet traffic is bursty by nature, causing the buffers to fill up quickly and then to become empty again. Therefore, if the buffer is empty most of the time and the aql is small, then the network is not congested and the hosts should not be informed to reduce their sending rates [3, 5, 16]. Accordingly, solving the aql problem by using q produces new problems. Thus, researchers proposed the new trend of combining q and aql to calculate the Dp. Abbasov and Korukoglu [16] proposed ERED in 2009. ERED uses both q and aql to predict the congestion at the router buffer. The dropping rules used by ERED are as follows. If min $<$ -aql $<$ max and q $>$ min, then packets are dropped based on the same equation as RED. If aql $<$ min and q $<$ 1.75 $\times$ max, then packets are dropped with a high probability. The remaining parts of ERED are the same as RED. Performance analysis showed that ERED is an outstanding method that outperforms most, if not all, of the prominent AQM methods, including RED [5], REM [22], Blue [23], SRED [20], and flow RED (FRED) [24].

In summary, existing AQM methods have successfully addressed the problems resulting from tail-drop method up to a certain extent. However, these methods, as well as many other AQM methods, assume the stable status of the Internet and the occurrence of a linear traffic arrival and departure. However, the status of the Internet constantly changes over time because of the bursty nature of the traffic (not linear) [5, 25, 26]. Another important issue is that these methods require substantial parameter settings. In RED and all its variants for example, at least four parameters should be initialized, namely min and max thresholds, Dp, and Wq [1, 17, 25]. The performance of AQM methods is highly sensitive to these parameters. As mentioned in [26, 27], even if the values of the proposed parameters suit a certain scenario, the applicability of the AQM method is limited only to that scenario and does not cover the dynamic changing conditions of actual scenarios. Accordingly, assuming the linear nature of Internet traffic and changing the parameter values are inefficient solutions making it difficult to obtain a stable and robust method [25, 26]. To avoid the linearity and parameterization problems, fuzzy logic is adopted for congestion control. Fuzzy logic is a methodology that deals with nonlinear systems, uncertain parameters, and imprecise measurement and modeling [1, 28]. Fuzzy logic was initially proposed for congestion with ATM networks and was subsequently used in IP networks to reduce packet loss rate and improve utilization [28].

Chrysostomou et al. [17] proposed FEM. FEM is one of the first methods that used fuzzy logic to overcome the limitations of RED and its variants. Evidently, FEM is an explicit congestion control method that works with differentiated services (DiffServ) to control congestion using fuzzy logic. The method utilizes linguistic knowledge to precisely identify network status and mark packets accordingly. FEM also addresses the limitations of RED and its variants by avoiding unnecessary parameterization. The results showed that FEM performs better in different scenarios without any parameterization.

Liu et al. [12] proposed the IntelRate controller. Similar to FEM, the IntelRate controller uses fuzzy logic to overcome the linearity and parametrization problems in RED and its variants. Similar to SRED, the IntelRate controller uses q to predict congestion and to calculate probability drop. However, IntelRate uses q with fuzzy logic to overcome the linearity and parametrization problems in SRED. The IntelRate controller is simulated using the well-known OPNET modeler. The results showed that the IntelRate controller is robust and effective, as it improves both throughput and link utilization as it reduces delay.

Woodward et al. [18] proposed the REDFL. This method extends RED by integrating fuzzy logic within the RED algorithm. In addition, REDFL added another congestion indicator $Packet\ loss$ to aql, the only indicator used by RED. Thus, REDFL uses aql and $Packet\ loss$ as input linguistic variables for a fuzzy logic system. This process reduces the extensive parameter dependency and settings in RED. The output of the used fuzzy logic system is the packet dropping rate used to alleviate congestion. REDFL is simulated using the discrete-time queue model and compared with the RED algorithm. The results showed that REDFL outperforms RED in terms of average queue length, throughput, packet loss rate, and packet dropping probability.

In addition to the above mentioned methods, several other congestion methods employ fuzzy logic with various input linguistic variables to improve performance. However, none of these methods efficiently managed or predicted congestion early [1, 2, 13]. The present paper extends RED using fuzzy logic with two new input linguistic variables. However, aql and speculated delay (D_{Spec}) are used as input linguistic variables instead of the aql and $Packet\ loss$ used in REDFL. Delay is an important measurement and indicator of network congestion. As shown in the following subsections, the use of speculated

delay (D_{Spec}) and ***aql*** as input linguistic variables of the proposed FLRED (fuzzy logic RED) method indicates remarkable congestion management, especially given heavy congestion. Furthermore, to avoid linearity and parameterization, the proposed FLRED method reduces both packet loss and delay, improves bandwidth utilization, and avoids both global synchronization and TCP starvation.

3 Proposed FLRED method

This section discusses the design of the proposed FLRED method. FLRED employs fuzzy logic to address the problems of congestion control methods. Fuzzy logic is particularly used to overcome the linearity and parametrization existing in most current congestion control methods. In addition, FLRED uses two congestion indicators, namely ***aql*** and D_{Spec}, to improve congestion control performance. ***aql*** and D_{Spec} are two effective indicators for predicting and avoiding congestion at an early stage, as will be discussed in the following two subsections. These two indicators (***aql*** and D_{Spec}) are used to calculate the ***Dp*** for incoming packets within a fuzzy inference process (FIP). Figure 6 shows the flowchart of using FIP to calculate the ***Dp*** and other values. Section 3.1 discusses the calculation of the ***aql***. Section 3.2 discusses the calculation of the D_{Spec}. Section 3.3 describes how FLRED utilizes fuzzy logic.

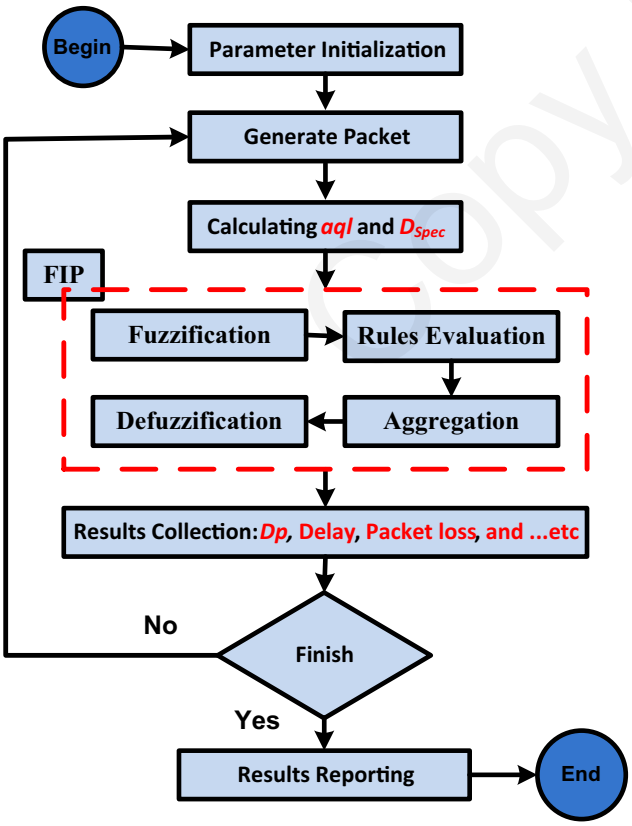

Fig. 6 Flowchart of the calculation of ***Dp*** and other values

3.1 Low-pass filter *aql*

aql is one of two variables aside from D_{Spec} used to calculate ***Dp*** in FLRED. This variable is the calculated average value of ***q*** using a low-pass filter (weighted queue ***Wq***), in which the effect of a new value to the average can be determined as needed [5]. When a low-pass filter is used, a sudden increase in the queue size caused by bursty traffic or temporary congestion will not cause a noticeable increase in ***aql***. Therefore, unnecessary packet dropping is avoided in temporary congestion cases, because ***aql*** is considered over a sufficient period time [1]. Equation 1 is used to calculate ***aql*** using a low-pass filter.

$$aql = (1 - Wq) * aql + W * q \quad (1)$$

Given minimal ***Wq***, the ***aql*** responses to ***q*** changes slowly. Therefore, a congestion method will not predict the congestion at early stages. When ***Wq*** is massive, the temporary congestion will not be filtered by the averaging procedure. As recommended by RED and based on several calculations, ***Wq*** is set to 0.002 in the proposed FLRED for all experiments [5].

3.2 Delay speculation

Delay refers to the time taken by a packet to travel from source to destination. Delay is one of the main factors affecting network performance quality. High network delay lessens network performance and vice versa. The main types of network delays are in processing, transmission, propagation, and queuing. This study focuses on queuing delay, which denotes the waiting time of a packet at the router's buffer queue before being transmitted; queueing delay depends on the length of the queue. Long queues cause high queuing delays and vice versa [1, 29]. Therefore, a high queuing delay indicates network congestion. In the proposed FLRED method, D_{Spec} is the other variable aside from ***aql*** used to predict congestion and calculate ***DP***. Little's law [30], which predicts the packet waiting time in the queue, is derived and used to calculate D_{Spec} according to arrival (***Arr***) and departure rates (***Dep***), as given in Eq. 2.

$$D_{Spec} = (Arr_{Spec} - Dep_{Spec}) * q \quad (2)$$

where Arr_{Spec} is the arrival rate speculation and Dep_{Spec} is the departure rate speculation. D_{Spec} increases and decreases with the arrival rate. In contrast, D_{Spec} decreases when the departure rate increases and increases when it decreases. The ArrSpec is a low-pass filter of the average

arrival rate over the previous and current packet arrival rates. Equation 3 is used to calculate Arr_{Spec}.

$$Arr_{Spec} = Arr_{t-1}(1 - W_{arr}) + Arr_t(W_{arr}) \quad (3)$$

where t is the time value, Arr_{t-1} is the previous value of the packet arrival rate, Arr_t is the current value of the packet arrival rate, and W_{arr} is the weight parameter. The Dep_{Spec} is a low-pass filter of the average departure rate over the previous and current packet departure rates. Equation 4 is used to calculate Dep_{Spec}.

$$Dep_{Spec} = Dep_{t-1}(1-W_{dep}) + Dep_t(W_{dep}) \quad (4)$$

where Dep_{t-1} and $Dept$ are the previous and current values of the packet departure rate, respectively. The used low-pass filter contributes more to the previous rates of both Arr_{Spec} and Dep_{Spec}. This is achieved by setting both W_{arr} and W_{dep} values below 0.5. Accordingly, the calculated outputs of both Arr_{Spec} and Dep_{Spec} change slightly when the rate changes over time. In the proposed FLRED method, several weight values (1.0, 0.9, 0.8, …, 0.1, 0.09, 0.08, …, 0.0.01) are tested. Finally, the weight value for both $W_{ar}r$ and W_{dep} is set to 0.2.

The optimum value of D_{Spec} is gained when Arr is equal to Dep. A large delay occurs when the value of Arr exceeds Dep and the buffer (q) is not empty. A tiny delay occurs when the value of Arr is less than Dep. As shown in Fig. 7, D_{Spec} is proportional to the term ($Arr_{Spec} - Dep_{Spec}$).

3.3 FLRED fuzzification

As an FIP-based method, FLRED is implemented in four sequential steps: fuzzification, rule evaluation, aggregation, and defuzzification. Figure 6 shows the four sequential steps of the FIP process.

3.3.1 Step 1: Fuzzification

The first step of the FIP process is **fuzzification**, in which linguistic terms are extracted from the input crisp of the variables, aql and D_{Spec}, and converted to fuzzy sets. This study uses three variables with three fuzzy sets as follows: aql: {null, trivial, normal, large}, D_{Spec}: {zero, trivial, normal, large}, and Dp: {zero, trivial, average, long}. The commonly used triangular is used to formulate the input and output linguistic variables. Figure 8 shows the membership function of the aql input linguistic variable. Figure 9 depicts the membership function of the D_{Spec} input linguistic variable. Figure 10 shows the membership function of the Dp output linguistic variable.

The membership functions for aql, D_{Spec}, and Dp have the following boundaries: aql: [{0, 0, 0.3, 0.4}, {0.3, 0.4, 0.4, 0.5}, {0.4, 0.5, 0.7, 0.8}, {0.7, 0.8, 1.0, 1.0}], D_{Spec} [{0, 0, 0.3, 0.4}, {0.3, 0.4, 0.4, 0.5}, {0.4, 0.5, 0.7, 0.8}, {0.7, 0.8, 1.0, 1.0}], and Dp: [{0, 0, 0.005, 0.01}, {0.005, 0.01, 0.01, 0.02}, {0.01, 0.02, 0.2, 0.4}, {0.2, 0.4, 1.0, 1.0}].

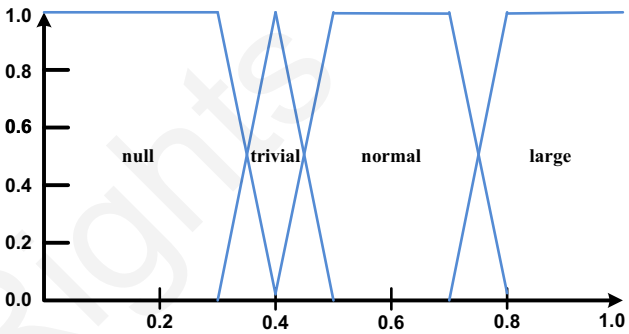

Fig. 8 Membership function of aql

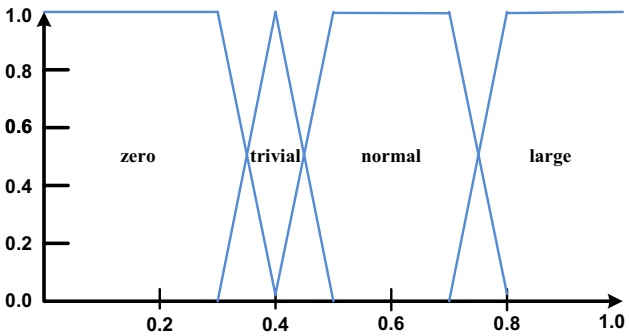

Fig. 9 Membership function of D_{Spec}

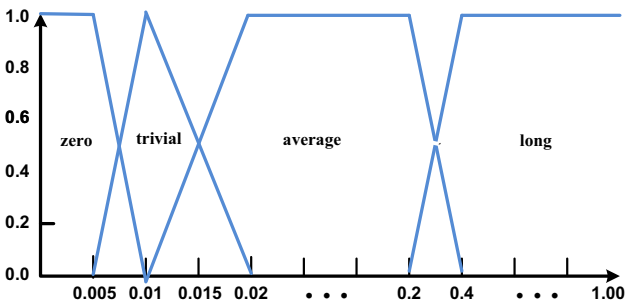

Fig. 10 Membership function of Dp

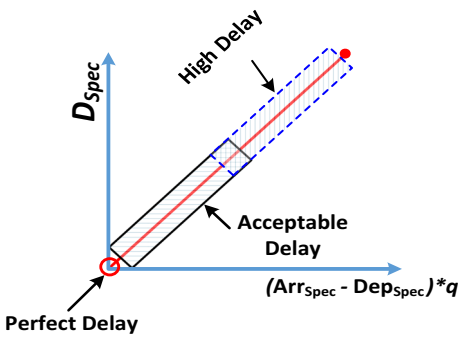

Fig. 7 D_{Spec} versus $(Arr_{Spec} - Dep_{Spec}) * q$

Table 1 Set of fuzzy logic rules

D_{Spec}	*aql*			
	Null	Trivial	Normal	Large
Zero	Zero	Zero	Zero	Long
Low	Zero	Zero	Normal	Long
Moderate	Zero	Normal	Long	Long
High	Zero	Normal	Long	Long

After creating the membership functions, the linguistic terms in the fuzzy set are obtained from the corresponding crisp values. Several linguistic terms with different probability values can be obtained from one input crisp value.

3.3.2 Step 2: Rule evaluation

The second step of the FIP process is **rule evaluation**, which defines and evaluates the rules employed to extract the linguistic terms from the previous step. Table 1 shows the rules employed in the proposed FLRED method. The input terms are shown in the header of the rows and columns of the table. The output terms are shown in Table 1.

3.3.3 Step 3: Aggregation

The third step of the FIP process is **aggregation**, during which the probabilities of the output linguistic terms obtained by the applied rules are calculated. Moreover, the repetitions of the same linguistic terms obtained by these rules, probably with different probabilities, are determined by producing a single probability value for each term. This step takes the probability values of the input linguistic terms obtained in the fuzzification step and produces a probability for the linguistic term of the output variable, as given in Eq. 5. As expected, the same linguistic term for the output variable may be obtained because the rules with the same output term are applied. These identical terms are aggregated in a single output with an underlying term and a probability value that is the maximum among the calculated probabilities for that term.

$$\mu^{A,B} = max\,(\mu^{A}(x),\ \mu^{B}(x)) \tag{5}$$

where $\mu^{A}(x)$ is the probability value of the first input variable, ***aql***, and $\mu^{B}(x)$ is the probability value of the second input variable, D_{spec}. The maximum value of similar output linguistic terms extracted from different rules is considered.

3.3.4 Step 4: Defuzzification

The fourth and final step of the FIP process is **defuzzification**, wherein the output linguistic variable values are produced based on the fuzzy set. Center of gravity (COG) method is employed for defuzzification [31]. COG locates the center point among all the output linguistic terms. Given that several output linguistic terms are expected to be extracted as several rules, Eq. 6 is used to calculate the final output crisp result.

$$COG = \frac{\sum_{x=a}^{b} \mu^{A}(x)x}{\sum_{x=a}^{b} \mu^{A}(x)} \tag{6}$$

where the numerator is the probability of each linguistic term for the output variable as calculated in the aggregation step multiplied by the values of the term in the fuzzy set. Meanwhile, the denominator is the number of values for that term in the fuzzy set. Figure 11 shows the algorithm of the proposed FLRED method.

The Proposed FLRED Algorithm

```
1. Initialize FuzzySets,
2. Initialize FuzzyRules
3. with packet arrival
4.     If q= =0 THEN aql:=(1-w)f(time- q_time) * aql
5.     If q <> 0 THEN aql:= (1-w)* aql + w *q
6.     DSpec = (ArrSpec - DepSpec) * q
7.     Fuzzyaql = Fuzzify (aql)
8.     FuzzyDSpec = Fuzzify (DSpec)
9.     GroupFuzzyDp = Appy Rules(Fuzzyaql, FuzzyDSpec)
10.    FuzzyDp = Aggregate
11.    Dp = De-fuzzify (FuzzyDp)
12.    Drop packet with Dp
13.    If QueueLenth ==0
14.        q_time=time
```

Fig. 11 Algorithm of the proposed FLRED method

4 Simulation

This section clarifies the simulation model in which the FLRED method was evaluated. FLRED was simulated using the widely used discrete-time queue model. The discrete-time queue model is an inexpensive method for testing whether a design of a proposed system works and evaluating the performance of that system. This model uses time slots (equal time periods) for a given system performance evaluation [1, 32]. A packet might arrive, depart, or both (arrive and depart) in a single time slot. The discrete-time queue model has been used by several AQM methods that calculate the performance factors at each time slot [1, 2, 18].

The simulation models of the proposed FLRED, RED, and ERED were implemented on a single router buffer with a capacity of 20 packets and a FIFO queue. As recommended, the min and max thresholds were set to 3 and 9 for the RED and ERED methods, respectively. A total of 2,000,000 slots of time were used to perform the experiments. The first 800,000 slots were used as warm-up and were not counted in the measurement of the evaluation

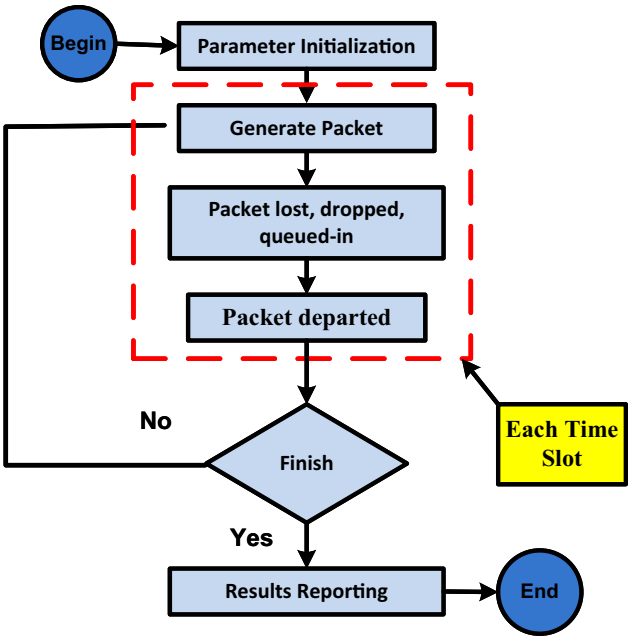

Fig. 12 Flowchart of the experiment steps

parameters. Consequently, the system achieved a stable state and therefore a more accurate performance. The remaining 1,200,000 slots were used to calculate the performance metrics. Three different arrival rates (0.93, 0.5, and 0.33) were utilized in the remaining 1,200,000 time slots to simulate different congestion scenarios, as will be demonstrated in the following section. Meanwhile, departure rate was fixed at 0.5. FLRED was implemented on Java using NetBeans Integrated Development Environment (IDE) 8.1 under 64 bits Windows 10, in Intel Core i3 2.10 GHz processor, and 6 GB RAM.

The experiment consisted of several steps to obtain the results. In step 1, the parameters were initialized to specific values. In step 2, a packet was generated with a certain probability and sent to the queue in a time slot. In step 3, the generated packet might be dropped, lost, or queued depending on the queue status. Meanwhile, a packet might be departed. Steps 1 and 2 were performed at each time slot. Finally, the results were collected and reported. These steps are depicted in Fig. 12.

5 Performance analysis

This section presents the performance evaluation of the proposed FLRED. To my knowledge, no fuzzy-based congestion method is available online for comparison. For example, a number of papers provide a figure showing the membership functions of the algorithm in general but not the exact boundary of the membership functions. Other papers provide either the membership functions or the rules set, whereas other studies give neither. In the present paper, the proposed FLRED method is described in detail, including the boundaries of the membership functions, the rules set, and the parameter values. Thus, the proposed method can be used as basis for comparison with any new proposed fuzzy-based congestion method. Furthermore, FLRED is compared with two of the most widely known and commonly used AQM methods (without fuzzy), namely RED and ERED.

FLRED was evaluated and compared with RED and ERED in four different scenarios in terms of packet loss, packet dropping, and delay. Packet loss occurs because of buffer overflow. Packet dropping was performed by a congestion method before a router buffer becomes full to avoid congestion [1, 5, 13]. The first and second scenarios assumed heavy congestion, in which the packet arrival rates were 0.98 and 0.93, respectively, whereas the departure rate was only 0.5 for both scenarios. Figure 13a, b shows the packet losses and packet dropping ratios, respectively, of FLRED, RED, and ERED for the first scenario. Figure 14a, b presents the packet losses and packet dropping ratios, respectively, of FLRED, RED, and ERED for the second scenario. Compared with RED and ERED, FLRED displays less packet loss. However, FLRED drops more packets than RED and ERED because FLRED predicts the congestion at an earlier stage than both methods under heavy congestion. When both packet loss and packet dropping are aggregated, the total packets missed are nearly the same. However, this does not mean

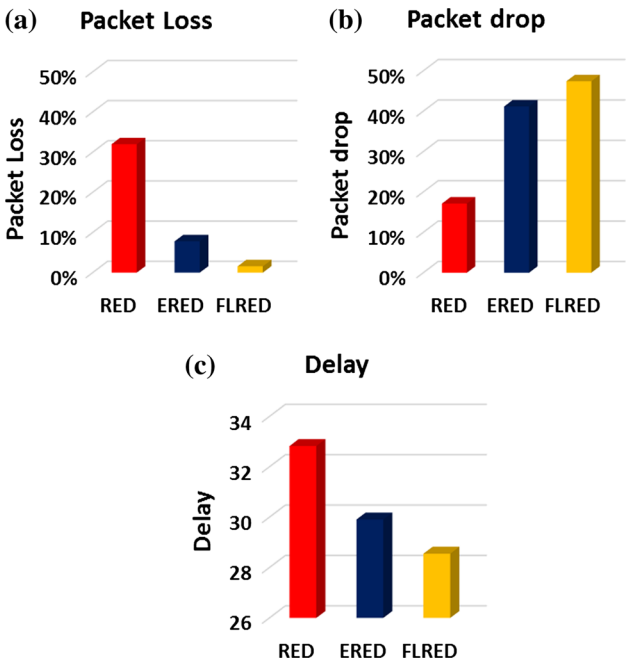

Fig. 13 Heavy congestion (first scenario): **a** packet loss ratio, **b** packet dropping ratio, **c** delay for heavy congestion (first scenario)

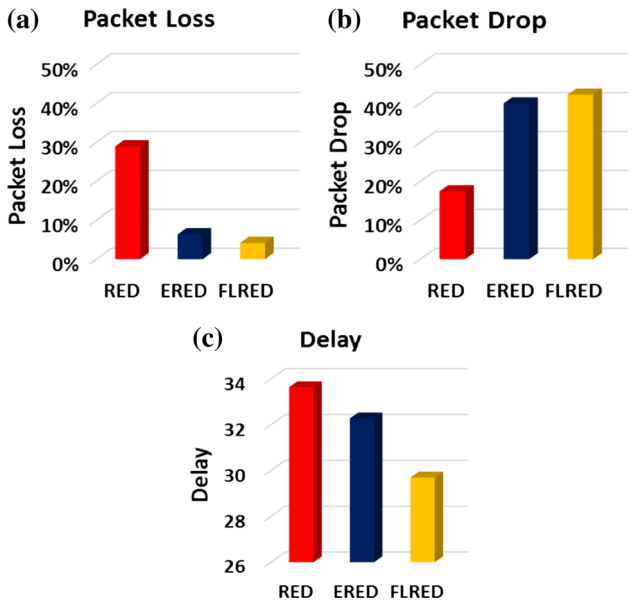

Fig. 14 Heavy congestion (second scenario): **a** packet loss ratio, **b** packet dropping ratio, **c** delay for heavy congestion (second scenario)

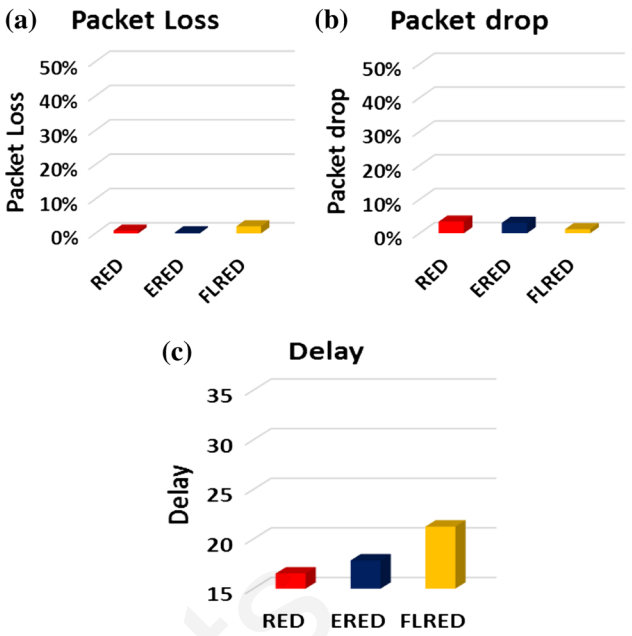

Fig. 15 Light congestion (third scenario). **a** packet loss ratio, **b** packet dropping ratio, **c** delay for light congestion (third scenario)

that the three methods exhibit the same performance. Packet dropping shows the ability of a congestion method to predict the congestion at early stage and to control the number of dropped packets and from which source to drop. Consequently, delay was reduced, bandwidth utilization was improved, and both global synchronization and TCP starvation problems were overcame. Packet loss means that every packet is thrown when the buffer is overwhelmed. Thus, a congestion method cannot predict the congestion early enough to control the number of dropped packets or even the source of the dropped packets. Consequently, delay was extended, bandwidth utilization becomes inefficient, and both global synchronization and TCP starvation problems worsened [7–11]. Figure 13c shows the delays of FLRED, RED, and ERED for the first scenario. Figure 14c indicates the delays of FLRED, RED, and ERED for the second scenario. As shown, FLRED shows less delay than RED and ERED, suggesting that FLRED outperforms both RED and ERED under heavy congestion, because it reduces delay and drops more packet than lost packets than RED and ERED.

The third scenario assumed a light congestion, in which packet arrival and departure were both 0.5. In theory, neither packet loss nor packet dropping should occur because the packet arrival and departure were equal. However, packet loss and packet dropping inevitably occur because of the bursty nature of the network traffic. Figure 15a, b shows the packet loss and packet dropping ratios, respectively, of FLRED, RED, and ERED for this scenario. FLRED shows a slightly higher packet loss and fewer packet drops than RED and ERED. Figure 15c shows the delays of FLRED, RED, and ERED for this scenario. As shown, FLRED shows a greater delay than RED and ERED. Thus, RED and ERED exhibited a slightly better performance than FLRED under light congestion.

The fourth scenario assumed no congestion at all, in which the packet arrival and departure rates were 0.33 and 0.5, respectively (the arrival rate was less than the departure rate). Figure 16a, b shows the packet loss and packet dropping ratios, respectively, of FLRED, RED, and ERED for this scenario. FLRED, RED, and ERED show the same packet dropping and packet loss probabilities that are equal to 0, because the arrival rate was less than the departure rate, which indicated the lack of congestion. Figure 16c shows the delays of FLRED, RED, and ERED for this scenario. As shown, FLRED was slightly less delay than RED and ERED. Accordingly, FLRED outperformed both RED and ERED in the no heavy congestion scenario.

A congestion method mainly aims to control the traffic flow, especially when heavy congestion occurs [1, 3, 13]. As shown, the proposed FLRED remarkably outperformed both RED and FLRED under heavy congestion scenarios (first and second scenarios), which were the key scenarios for evaluating a congestion method. This displayed approximately the same performance under light and no congestion, which were the secondary scenarios for evaluating a congestion method. FLRED reduced both packet loos and delay, improved bandwidth utilization, and avoided both global synchronization and TCP starvation.

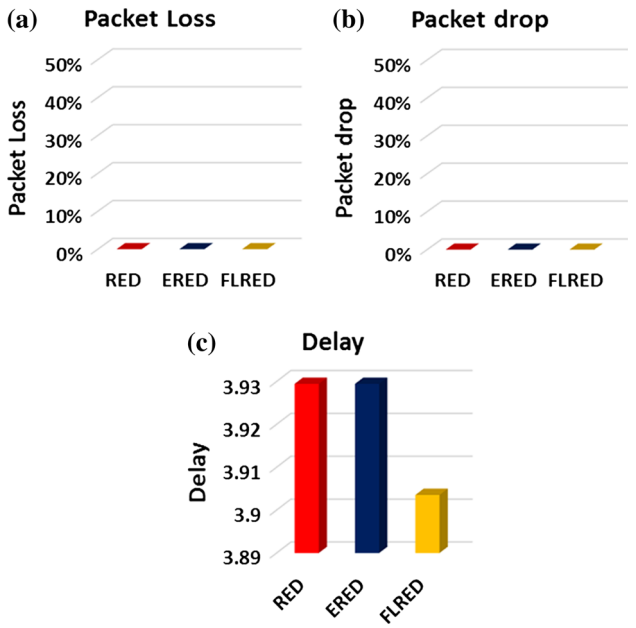

Fig. 16 No congestion (fourth scenario): **a** packet loss ratio, **b** packet dropping ratio, **c** delay for no congestion (fourth scenario)

Accordingly, FLRED is a promising congestion method that can be adopted to control the traffic flow and consequently save network resources and improve overall performance.

6 Conclusion

Congestion control is one of the major issues in computer networks that has been widely investigated from the emergence of computer networks. This paper proposed a new congestion control method called FLRED. FLRED extends RED, the most well-known and widely used congestion control method. FLRED uses fuzzy logic to avoid the linearity and parametrization problems in RED. In addition, FLRED uses two congestion indicators (***aql*** and $\boldsymbol{D_{Spec}}$) to predict and avoid congestion at an early stage. FLRED, RED, and ERED were simulated, evaluated, and compared using the widely used discrete-time queue model under Java environment. The simulation result showed that FLRED outperformed both RED and ERED by decreasing both delay and packet loss, particularly given heavy congestion. FLRED is a promising congestion method that can save network resources and improve overall performance. Future studies can explore a wide range of topics on FLRED. The value of ***aql*** is highly dependent on network characterization. Therefore, further investigation should be conducted on finding the optimum ***aql*** value. In addition, other congestion indictors aside from ***aql*** and $\boldsymbol{D_{Spec}}$ should be considered. Furthermore, a discrete-time queue model was used in this study to implement FLRED. Thus, a real environment implementation in the presence of a TCP protocol can be performed. Moreover, the conduct of FLRED under multi-hop networks and wireless networks would also be interesting to evaluate.

References

1. Khatari M, Samara G (2015) Congestion control approach based on effective random early detection and fuzzy logic. MAGNT Res Rep 3(8):180–193
2. Baklizi M et al (2014) Fuzzy logic controller of gentle random early detection based on average queue length and delay rate. Int J Fuzzy Syst 16(1):9–19
3. Peterson LL, Davie BS (2012) Computer networks a systems approach, 5th edn. Morgan Kaufmann, Burlington
4. Wurtzler M (2002) Analysis and simulation of weighted random early detection (WRED) queues. Diss., University of Kansas
5. Floyd S, Jacobson V (1993) Random early detection gateways for congestion avoidance. IEEE/ACM Trans Netw 1(4):397–413
6. Chen C-K, Liao T-L, Yan J-J (2009) Active queue management controller design for TCP communication networks: variable structure control approach. Chaos, Solitons Fractals 40(1):277–285
7. Zhan Z, Jie ZHU, Di XU (2012) Stability analysis in an AVQ model of Internet congestion control algorithm. J China Univ Posts Telecommun 19(4):22–28
8. Xiong N et al (2010) A novel self-tuning feedback controller for active queue management supporting TCP flows. Inf Sci 180(11):2249–2263
9. Liu J (2014) Network traffic control based on modern control techniques: fuzzy logic and network utility maximization. Diss., University of Ottawa
10. Lautenschlaeger W, Francini A (2016) Global synchronization protection for bandwidth sharing TCP flows in high-speed links. arXiv:1602.05333
11. Kim H-S et al (2015) A measurement study of TCP over RPL in low-power and lossy networks. J Commun Netw 17(6):647
12. Liu J, Yang OW (2013) A stable fuzzy logic controller using the least parameter for explicit traffic control. Int J Innov Comput Inf Control 9(7):2801–2819
13. Baklizi M et al (2013) Dynamic stochastic early discovery: a new congestion control technique to improve networks performance. ICIC Int 9(3):1113–1126
14. Zhu Y (2009) A new class-based traffic queue management algorithm in the internet. KSII Trans Internet Inf Syst (TIIS) 3(6):575–596
15. Floyd S (2000) Recommendations on using the gentle variant of RED. http://www.aciri.org/floyd/red/gentle.html
16. Abbasov B, Korukoglu S (2009) Effective RED: an algorithm to improve RED's performance by reducing packet loss rate. J Netw Comput Appl 32(3):703–709
17. Chrysostomou C et al (2003) Fuzzy explicit marking for congestion control in differentiated services networks. In: Proceedings of the eighth IEEE international symposium on computers and communication, 2003 (ISCC 2003). IEEE
18. Abdel-Jaber H et al (2008) Fuzzy logic controller of random early detection based on average queue length and packet loss rate. In: International symposium on performance evaluation of computer and telecommunication systems, 2008 (SPECTS 2008). IEEE
19. Zhou K, Yeung KL, Li VOK (2006) Nonlinear RED: a simple yet efficient active queue management scheme. Comput Netw 50(18):3784–3794

20. Ott TJ, Lakshman TV, Wong LH (1999) Sred: stabilized red. In: Proceedings of the eighteenth annual joint conference of the IEEE computer and communications societies (INFOCOM'99), vol 3. IEEE
21. Jamali S, Alipasandi N, Alipasandi B (2014) An improvement over random early detection algorithm: a self-tuning approach. J Electr Comput Eng Innov 2(2):57–61
22. Athuraliya S et al (2001) REM: active queue management. IEEE Netw 15(3):48–53
23. Feng W-c et al (1999) BLUE: a new class of active queue management algorithms. Ann Arbor 1001:48105
24. Lin D, Morris R (1997) Dynamics of random early detection. In: ACM SIGCOMM computer communication review, vol. 27, no. 4. ACM
25. Chrysostomou C, Pitsillides A, Ahmet Sekercioglu Y (2009) Fuzzy explicit marking: a unified congestion controller for Best-Effort and Diff-Serv networks. Comput Netw 53(5):650–667
26. Chrysostomou C (2006) Fuzzy logic based AQM congestion control in TCP/IP networks. Diss., University of Cyprus, 2006
27. Plasser E, Ziegler T (2004) A RED function design targeting link utilization and stable queue size behavior. Comput Netw 44(3):383–410
28. Liu J, Yang OW (2013) Using fuzzy logic control to provide intelligent traffic management service for high-speed networks. IEEE Trans Netw Serv Manag 10(2):148–161
29. James K, Keith R (2012) Computer networking, a top-down approach, 6th edn. Pearson, London
30. Allen AO (1990) Probability, statistics, and queueing theory. Academic Press, London
31. Michael N (2011) Artificial intelligence a guide to intelligent systems, 3rd edn. Pearson Education, Victoria
32. Guizani M et al (2010) Network modeling and simulation: a practical perspective, 1st edn. Wiley, New York