International Journal of Innovative
Computing, Information and Control
Volume **9**, Number **3**, March **2013**

ICIC International ©2013 ISSN 1349-4198
pp. 1113–1126

DYNAMIC STOCHASTIC EARLY DISCOVERY: A NEW CONGESTION CONTROL TECHNIQUE TO IMPROVE NETWORKS PERFORMANCE

Mahmoud Baklizi[1], Hussein Abdel-jaber[2], Mosleh M. Abu-Alhaj[1]
Nibras Abdullah[1], Sureswaran Ramadass[1] and Ammar ALmomani[1]

[1]National Advanced IPv6 Center of Excellence
Universiti Sains Malaysia
Penang 11800, Malaysia
{ mbaklizi; mosleh; abdullahfaqera; sures; ammarali }@nav6.org

[2]Department of Computer Information and Network Systems
The World Islamic Sciences and Education University
Amman, Jordan
hussein.abdeljaber@wise.edu.jo

Received January 2012; revised May 2012

Abstract. *The present paper proposes the Dynamic Gentle Random Early Detection (DGRED) algorithm for early stage congestion detection at the router buffer. Generally, the proposed DGRED algorithm depends on the stability of the average queue length at a specific level between allocated minimum and maximum threshold values, with the aim to improve the network performance. The DGRED algorithm is simulated and compared with the most known Active Queue Management Early Detection (RED) algorithm and two of its variants, namely, Gentle RED and Adaptive GRED. This comparison was conducted based on different performance measures, such as mean queue length, throughput, average queuing delay, packet loss, and dropping probability for packets. The comparison aimed to identify the algorithm that offers better performance measurement results under either non-congestion or congestion situation at the router buffers. The acquired results show that the proposed algorithm contributes in providing lesser queue length, delayed queuing, and packet loss probability compared with the existing algorithms when high packet arrival probability appears, that is,* (> 0.63). *Furthermore, DGRED generates adequate throughput when the packet arrival probability value is high.*

Keywords: Congestion control, Gentle random early detection (GRED), Adaptive GRED, Random early detection (RED), Performance evaluation, Simulation

1. **Introduction.** The performance of computer networks and Internet technologies are constantly being questioned because of their rapid growth. Congestion is one of the major problems that challenge network performance [1,2]. Congestion occurs when buffers of the network routers can no longer handle the incoming packets, as the amount of incoming packets exceeds the available network resources [3]. The drawbacks of congestions are as follows. Congestion plays a major role in worsening network performance by increasing the packet dropping probability (D_p) and increasing packet loss probability (P_L). In addition, congestion may lead to an increase in the mean queue length (mql) and the mean waiting time (D) of packets, which will finally degrade the amount of packets passing through the buffer of the routers, namely, the throughput (T) [4].

Congestion is associated with the status of the average queue length (aql), which in turn affects network performance. When aql value increases, T value likewise increases. At the same time, D and P_L increase, and the router buffer overflows. By contrast, when the aql

value relatively decreases, D and T likewise decrease. Network efficiency is decreased in both cases. Thus, congestion control is required to maintain a stable aql value, optimize the utilization of network resources, and enhance its performance.

Enormous congestion control algorithms, such as Random Early Detection (RED) [5], Gentle RED (GRED) [6], and Adaptive Gentle RED (AGRED) [7], have been proposed. However, these algorithms have failed to dynamically adjust to provide the best solution based on the status of the aql. Generally, the disadvantages of the existing congestion control algorithm can be summarized as follows. The existing algorithms use static probability for packet dropping, which leads to large drops in the number of packets when the probability value is high and bursty traffic is present. With bursty traffic, a heavy congestion signal is given out, which then leads to large packet drops. Conversely, network performance becomes degraded when the probability of packet dropping is set too low. Specifically, D_p, P_L, mql, and D will increase, and T will decrease. Consequently, a dynamic mechanism is required to implement packet dropping based on the congestion status. This paper aims to propose a new algorithm called the Dynamic GRED (DGRED) to address the aforementioned disadvantages and to improve network performance. Improving network performance involves alleviating P_L and obtaining more satisfactory performance measurement results with reference to mql and D when heavy congestion occurs at the router buffers of networks.

The rest of the paper is organized as follows. Section 2 presents previous related work. The proposed algorithm is discussed in Section 3. Section 4 presents the simulation information. The results of the developed simulation are discussed in Section 5. Section 6 presents the applications of the DGRED algorithm. Finally, conclusions are stated in Section 7.

2. **Related Works.** Several studies on controlling congestion and handling the aforementioned problems have been conducted in the past [8-13]. The Drop Tail (DT) method [14,15] aims to control congestion using a fixed router buffer size to optimize queuing delay. The DT method sets the size of the router buffers to a maximum while dropping all incoming packets when the router buffers overflow. The drawback of this method is the possibility of a rise in high packet queuing delay. DT may also depend on setting the router buffers to a minimum size, which decreases the throughput T. The DT method has several other drawbacks, such as, increase in packet loss rate, saturation of the queue router buffer [4], and global synchronization [5].

Active Queue Management (AQM) is a set of methods proposed to overcome the limitations of the DT method discussed earlier. Unlike the DT method, which starts dropping packets only after the router buffers overflow, AQM methods usually start dropping packets in the early stages. Consequently, early congestion control allows sources to decrease their transmission rates early, right before the router buffers are completely occupied. AQM controls the congestion in the router's buffer, improves the throughput, decreases packet queuing delay, decreases packet loss rate, and keeps the mql at a minimum. AQM emerges with an adaptable utilization buffer size. Packet droppings are initiated based on a calculated threshold value to prevent buffer overflow. AQM first calculates the value of aql then compares it with the given threshold. When the aql value is greater than the threshold value, all packets arriving at the buffer are dropped with the probability of preventing router buffer overflow [16-18].

Enormous algorithms for congestion control, such as RED [5], GRED [6], AGRED [7], VIATIME-DELAY AFFINE TAKAGIUGENO FUZZY MODELS [19] and other time-discrete queue analytical models [5-8], have been built based on AQM.

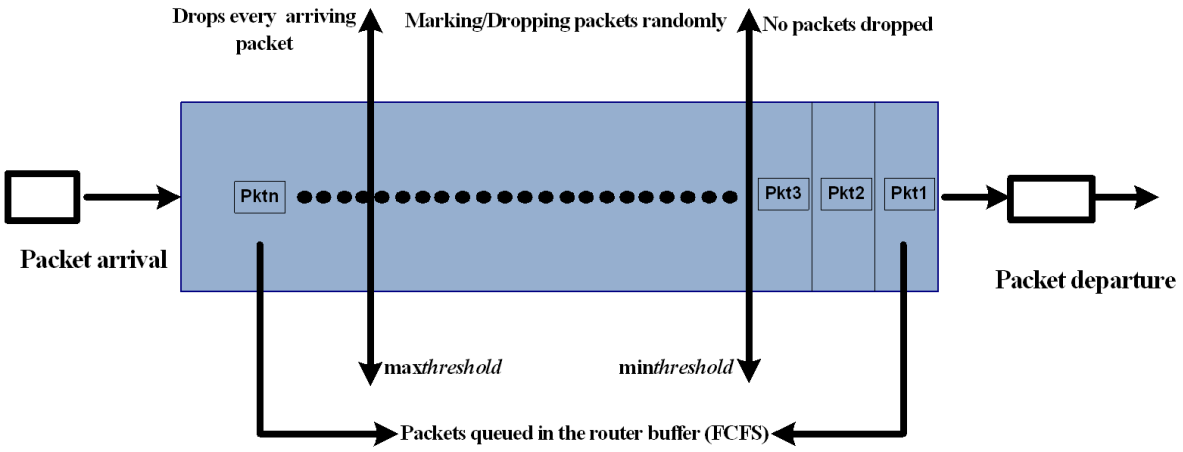

FIGURE 1. The single router buffer for RED

RED is one of the most significant algorithms for congestion control [5]. RED manages the congestion before the router buffer overflows using the computed *aql* and two calculated thresholds values, namely, *minthreshold* and *maxthreshold*. Generally, RED detects the congestion by initially computing the *aql* and comparing it with the *minthreshold* and *maxthreshold*. Congestion does not occur when *aql* is smaller than the *minthreshold*. The routers, therefore, do not drop any packet. If the *aql* is between the two thresholds, the arriving packet is dropped and the probability is calculated as D_p to alleviate congestion. Finally, when the *aql* is above the *maxthreshold*, all arriving packets are dropped to a D_p value equal to one (Figure 1).

Generally, RED's drawback is the varying *aql* computed according to the congestion status. Hence, if the congestion status is light, the *aql* value will be close to the *minthreshold*. If the congestion status is severe, the *aql* value will be close to the *maxthreshold*; thus, the packet D_p will increase, and the buffer will overflow. Another drawback is the reliance of the computed *aql* on the traffic load (number of connections). If the traffic load is high, the *aql* value may exceed the *maxthreshold*. In such a case, network performance in many aspects will worsen. Therefore, the router buffer will drop every arriving packet. Thus, the RED parameters must be set at particular values to ensure satisfactory performance. If the traffic load is low, the *aql* will normally be lesser than the *minthreshold*. Consequently, no arriving packet is dropped. Overall, RED cannot stabilize its *aql* value between the *minthreshold* and *maxthreshold* when the traffic load changes suddenly (i.e., bursty traffic) [20,21].

Floyd [6] proposed the GRED to overcome some of the limitations in RED [6,21,22]. Similar to RED, the GRED algorithm mainly aims to manage and control the congestion networks at the early stage. GRED implements its algorithm by stabilizing the aql at a certain level. GRED employs a similar approach used by RED in calculating the D_p. However, GRED utilizes three thresholds, namely, minimum, maximum, and doublemaximum. Generally, GRED reacts with the arriving packets based on one of the following scenarios (Figure 2):

1) When the *aql* at the router is below the *minthreshold*, no packets are dropped.
2) If the *aql* is between the *minthreshold* and *maxthreshold*, the router will drop the arriving packets randomly, similar to RED.
3) If the *aql* is between the *maxthreshold* and the *doublemaxthreshold*, the packets are dropped randomly with higher probability compared with the previous case.

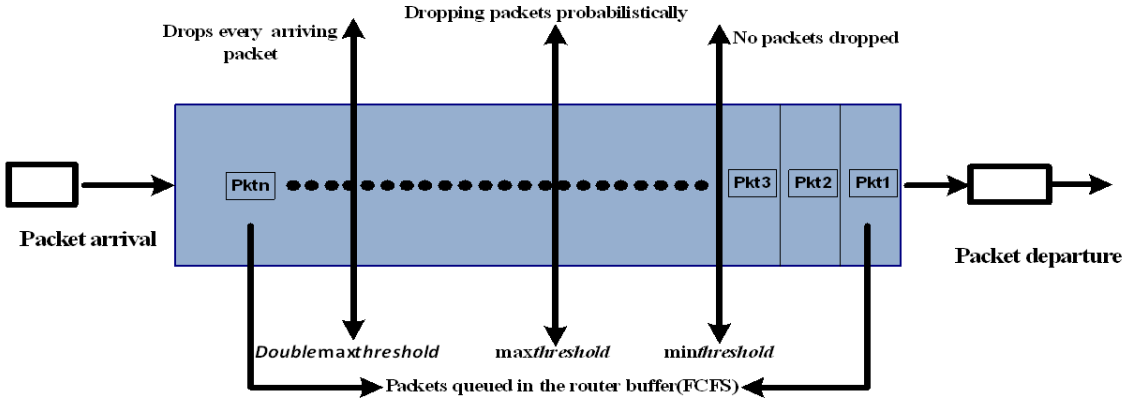

FIGURE 2. The single router buffer for GRED and AGRED

4) If the *aql* is equal or greater than the *doublemaxthreshold*, the GRED router drops the arriving packets with D_p equal to one (i.e., arriving packets are dropped).

Unfortunately, GRED has some limitations. First, GRED deals with several threshold values. Second, GRED must set its parameters to specific values to obtain satisfactory performance (i.e., parameterization). Third, when the *aql* is less than the *minthreshold* and heavy congestion occurs, the *aql* will take time to adjust, during which the router buffer will likely overflow. Thus, no packets are dropped despite the overflowing GRED router buffer.

The AGRED algorithm is proposed to improve the performance of GRED during router buffer congestion (i.e., deriving better quality results with reference to the *mql*, D, and P_L performance measures). In addition, the AGRED algorithm aims to enhance the parameter settings (e.g., the *maxthreshold* and the maximum value of D_{int}, which is the $D_{\max}$ of GRED). The calculation of the *aql* in AGRED is also similar to that in GRED. Therefore, AGRED decides on packet dropping in a manner similar to that in GRED [7] (Figure 2).

The main difference between the GRED and the AGRED lies in the calculation of the D_{init} value (the initial packet D_p). In AGRED, the D_{init} value varies between the $D_{\max}$ values to 0.5, as long as the *aql* value is between the *maxthreshold* and *doublemaxthreshold*. In GRED, when the *aql* value is between the *maxthreshold* and the *doublemaxthreshold*, the calculated D_{init} value of GRED varies from the $D_{\max}$ value to 1.0. In summary, all the aforementioned related methods fail to implement a congestion control that can deal with all congestion cases encountered by the network resources, the performance of which is consequently affected.

3. **The Proposed DGRED.** The proposed DGRED is an extension of GRED. DGRED employs a dynamic *maxthreshold* and *doublemaxthreshold* to control the congestion in the router buffer at the early stage before it overflows. The aim of the DGRED algorithm is to stabilize the *aql* using a new defined value called Target *aql*(T_{aql}). T_{aql} is calculated between the *minthreshold* and *maxthreshold* (Figure 3). Another aim for the proposed DGRED is providing better performance results than other AQM algorithms such as RED and two of its variants like GRED and AGRED. These better performance results are represented by the results of mean queue length, average queueing delay and packet loss probability when heavy congestion has occurred.

DGRED also updates the *maxthreshold* and *doublemaxthreshold* parameters at the router buffer to enhance network performance. DGRED uses the GRED algorithm's

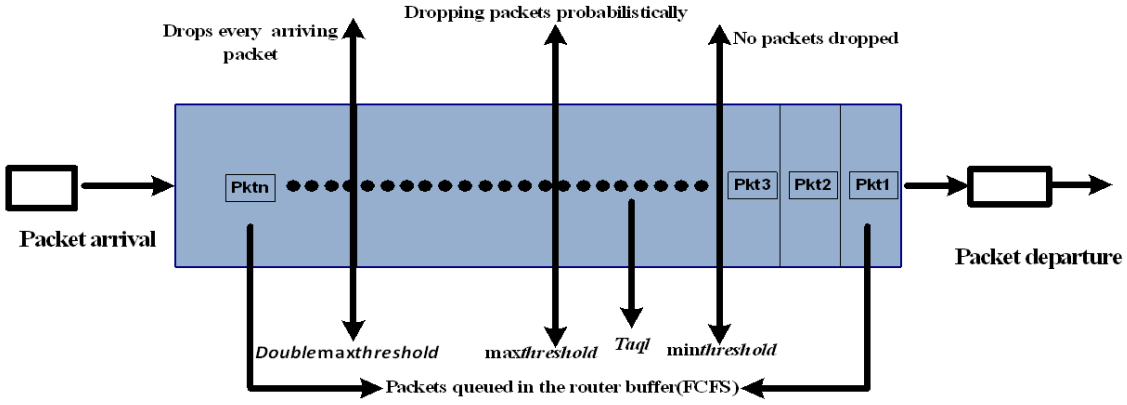

FIGURE 3. The single router buffer for proposed dynamic GRED

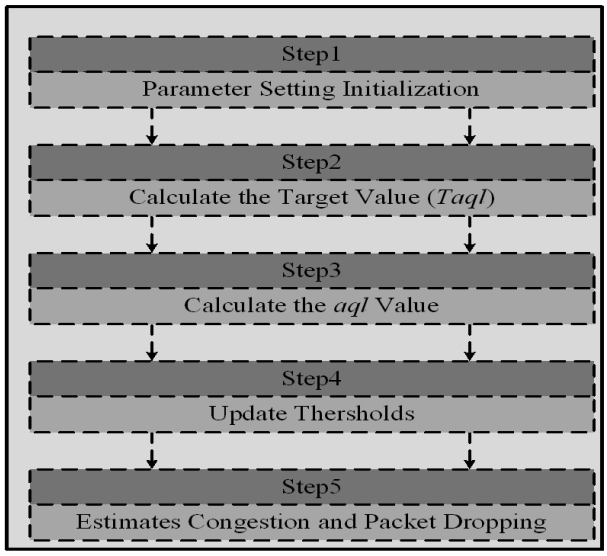

FIGURE 4. The proposed dynamic gentle random detection

policy in dropping packets with probability when the *aql* is between the *minthreshold* and *doublemaxthreshold*.

3.1. **DGRED process.** The DGRED's processing stages are illustrated in Figure 4. *The parameter setting initialization* step (Step 1) is triggered as the packet arrives at the router buffer. DGRED initiates the *minthreshold* and *maxthreshold* to the same values as those in the RED and GRED algorithms [5,6]. Furthermore, the *doublemaxthreshold* is set to the same value as that in GRED [6]. The *aql* is initialized to 0.0 and the counter (C) is set to -1. The parameter C represents the number of packets that have arrived at the router buffer thus far and have not been dropped since the last packet was dropped. The value of the *aql* is initialized in this stage as well.

The DGRED algorithm then *calculates the Target Value* (T_{aql}) (Step 2) using Equation (1). The proposed DGRED algorithm uses a calculated T_{aql} value that points to a specific position in the router buffer. The T_{aql} value is introduced to stabilize the *aql* between the *minthreshold* and *maxthreshold* fort the detection of congestion at the early stage. The indicated position by T_{aql} identifies the incipient congestion situation. Generally, T_{aql} is

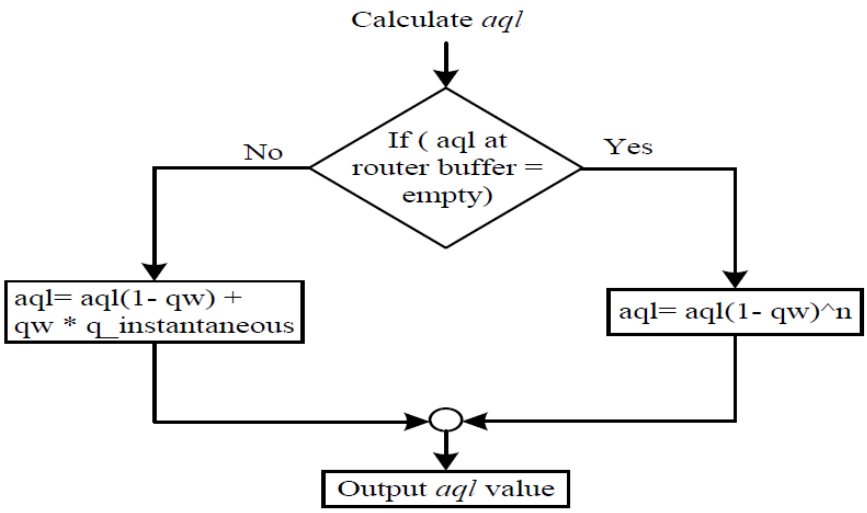

FIGURE 5. Calculation *aql*

calculated using Equation (1) as follows:

$$\frac{minthreshold + doublemaxthreshold}{\#threshold} \tag{1}$$

where # thresholds is the number of thresholds used by the algorithm (*minthreshold*, *maxthreshold*, *doublemaxthreshold*). Equation (1) is derived to obtain the T_{aql} between the *minthreshold* and *maxthreshold* position values. GRED recommends that the setting value of the *maxthreshold* is at least double that of the *minthreshold* [6]. Therefore, any setting value of the initialized *minthreshold* and *maxthreshold* always makes use of Equation (1) to provide a T_{aql} value between the *minthreshold* and *maxthreshold* values. This result helps in stabilizing the *aql* around the T_{aql} value between the *minthreshold* and *maxthreshold* positions. This method prevents build up in the router buffers, resulting in fewer dropped packets.

DGRED then (Step 3) examines the queue status and *calculates the value of aql* based on whether the router buffer is empty or not, as illustrated in Figure 5. Accordingly, if the queue is empty, the value of the *aql* is calculated based on the current_time-idle_time (n). The *aql* is calculated using Equation (2). On the other hand, if the queue at the router buffer is not empty, the *aql* is calculated using Equation (3).

$$aql = aql \times (1 - qw)^n \tag{2}$$

$$aql = aql \times (1 - qw) + qw \times q\_instantaneous \tag{3}$$

In the following step (Step 4), DGRED compares the calculated *aql* value with the T_{aql} value and subsequently *updates maxthreshold and doublemaxthreshold positions* to improve network performance (Figure 6). The values of both *maxthreshold* and *doublemaxthreshold* are updated with reference to the *aql* value. The increment and decrement in these thresholds are calculated using Equation (4). Notably, Equation (4) is performed only before the *maxthreshold* and *doublemaxthershold* values are updated.

$$(Doublemaxthreshold - minthreshold) \times \frac{1}{number\ of\ threshold} \tag{4}$$

The *maxthreshold* and *doublemaxthreshold* values are increased and decreased by Equations (5) to (7) to manage congestion at the router buffers. This management is done by increasing and decreasing *maxthreshold* and *doublemaxthreshold* values around the T_{aql}

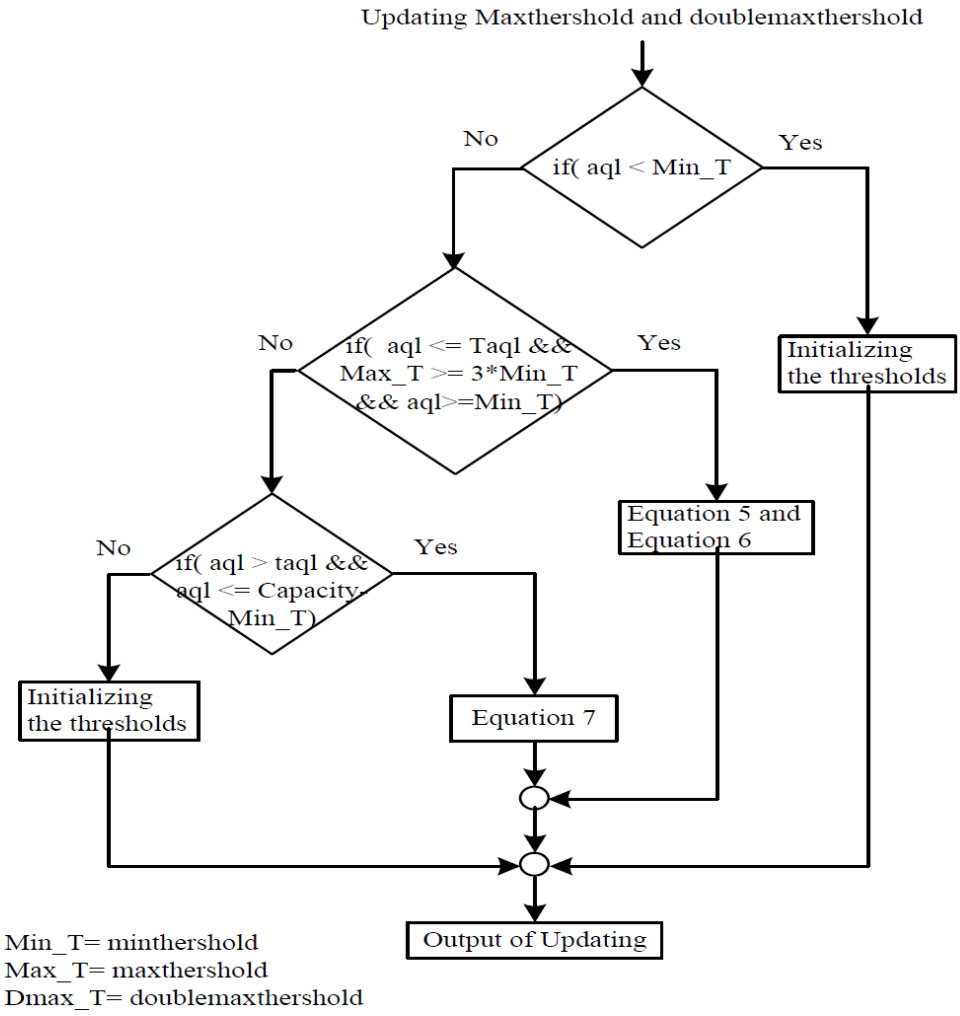

FIGURE 6. Maxthreshold and doublemaxthreshold updating process

level. Therefore, the *aql* value stabilizes at the T_{aql} level. This stabilization can prevent the filling up the router buffers. As a result, fewer packets are dropped. Furthermore, the calculated *maxthreshold* and *doublemaxthreshold* values provided by Equations (5) to (7) can increase and decrease the *aql* value around the T_{aql} level in a slow mode. Therefore, Equations (5) to (7) are derived.

As such, if the *aql* is smaller than the *minthreshold* value, no congestion occurs and the *minthreshold* and *maxthreshold* values will not change [6]. However, if the *aql* is larger than the *minthreshold* value and less than or equal to the T_{aql}, and the maxthreshold value is greater than or equal to three times that of the *minthreshold*, then the *maxthreshold* and *doublemaxthreshold* values are changed using Equations (5) and (6), respectively. As such, the *aql* value increases accordingly to be stabilized at the T_{aql}. On the other hand, if the *aql* is greater than the T_{aql} and less than or equal to the (capacity of buffer – *minthreshold*), the *maxthreshold* and *doublemaxthreshold* values are changed using Equation (7). Thus, they become equal and they prevent the *doublemaxthreshold* value to go over the buffer capacity. Subsequently, the *maxthreshold* and *doublemaxthreshold* values are increased to push the *aql* toward the T_{aql} and to alleviate the congestion at the router buffer by serving more packets.

$$maxthreshold - (doublemaxthreshold - minthreshold) \times \frac{1}{number\ of\ threshold} \quad (5)$$

$$doubleMaxmumthreshold - (doublemaxthreshold - minthreshold) \times \frac{1}{number\ of\ threshold} \quad (6)$$

$$maxithreshold + (doublemathreshold - minthreshold) \times \frac{1}{number\ of\ threshold} \quad (7)$$

Lastly, if none of these cases occurs, the *maxthreshold* is set to the same values as those in the RED algorithms [5] and the *doublemaxthreshold* is set to a similar value as that in GRED [6].

In the final stage (Step 5) of the DGRED algorithm, the *congestion is estimated and packet dropping is implemented*, as illustrated in Figure 7.

The congestion status is estimated based on the *aql* value. As such, if the *aql* value is smaller than the *minthreshold*, no packet is dropped because no congestion is presented at the DGRED router buffer. In addition, the D_p is set to 0.0 and the C value is set to -1. Hence, no packet is dropped. On the other hand, if the *aql* value is between the *minthreshold* and *maxthreshold* values, the DGRED router buffer drops packets in a way similar to that in GRED. Dropping packets is allocated with the increasing C value by

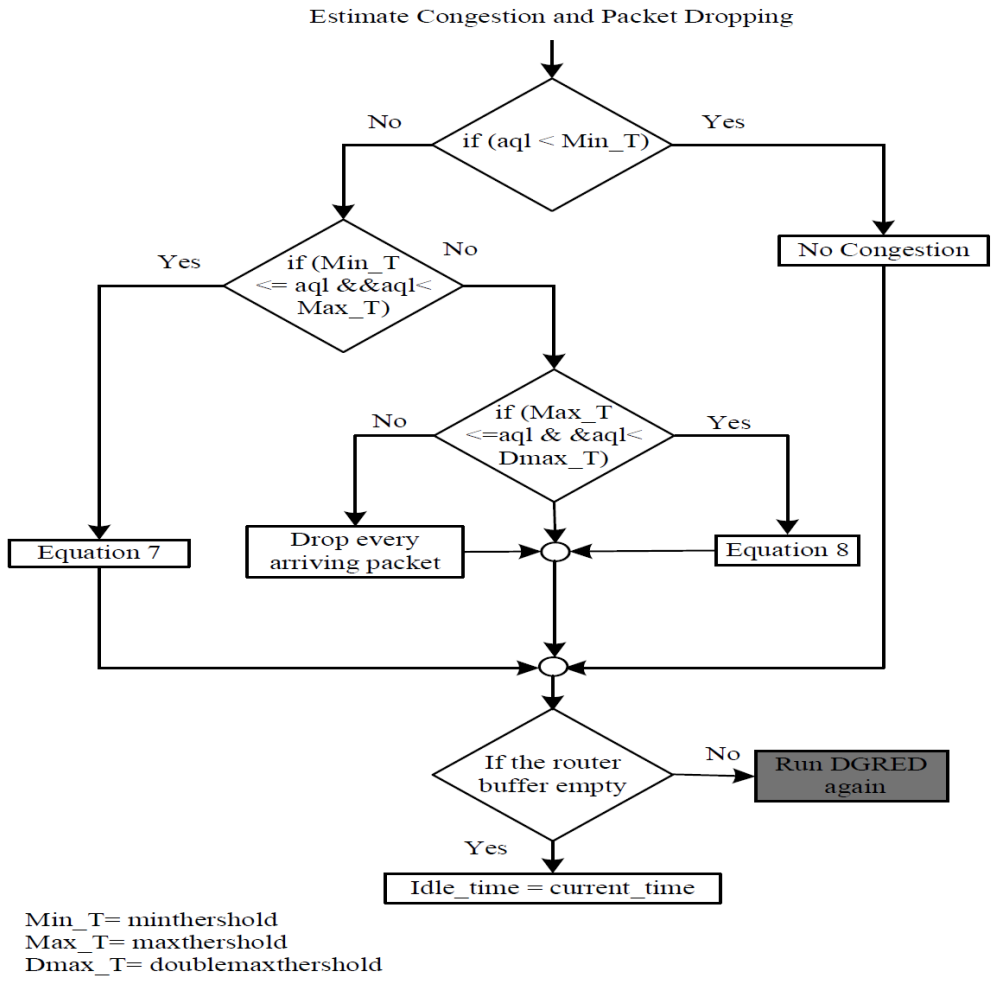

FIGURE 7. Estimate congestion and packet dropping

one and calculating the D_p for the arriving packet using Equation (8).

$$D_p = \frac{\frac{D_{\max} \times (aql - minthreshold)}{maxthreshold - niuthreshold}}{(1 - C \times D_{init})} \tag{8}$$

On the other hand, if the value of the *aql* is between the *maxthreshold* and *doublemaxthreshold*, the DGRED router buffer drops the packets in a way similar to that in GRED, which involves initializing the C value to 1 and calculating the D_p for the arrival packet using Equation (9). Finally, if the *aql* value is greater than or equal to the *doublemaxthreshold*, the DGRED router buffer drops/marks every arriving packet with $D_p = 1$ and sets C to zero. Subsequently, when the DGRED router buffer becomes empty, the idle_ time is set to current_ time.

$$D_p = \frac{D_{\max} + \frac{(1 - D_{\max}) \times (aql - maxthreshold)}{maxthreshold}}{(1 - C \times D_{init})} \tag{9}$$

4. **Simulation.** RED, GRED, AGRED, and the proposed DGRED are simulated based on a discrete-time queue that uses slot as a unit of time [10,23]. Each slot may involve packet arrival and/or departure. The compared algorithms are simulated by applying them in a network consisting of a single router buffer node. Notably, both packet arrival and departure are implemented in single mode. The scheduling mode is first-come-first-served. The RED, GRED, AGRED and DGRED simulations are implemented in Java on an i7 processor machine with 1.66GHz and 4GB RAM.

In the conducted simulation, the probability of the arriving packets at the router buffer in a fixed time unit called slot is denoted by α [23]. The probability of packet departure from the router buffer in a slot is denoted by β. Packet arrivals can be modeled using a Bernoulli process, whereas packet departures can be modeled using a geometrical distribution. Using geometrical distribution, packet inter-arrival times and service times are estimated to the values $1/\alpha$ and $1/\beta$, respectively.

5. **Evaluation Results.** The performance of the proposed DGRED algorithm is compared with those of GRED, AGRED, and RED. The performances of these algorithms are measured ten times in ten runs, each taking different seeds as input to the random number generator. This step removes possible bias in the output results and produces confidence intervals for the performance measures. The performances of all AQM methods are calculated after the system reaches a steady state.

For the parameter settings, RED, GRED, and AGRED are initiated using identical parameters at most. To create congestion and non-congestion scenarios at the buffer,

Table 1. Parameter settings for GRED, AGRED and RED algorithms

Parameter	*GRED, AGRED*	*RED*
Probability of packet arrival	*0.18-0.93*	*0.18-0.93*
Probability of packet departure	*0.5*	*0.5*
Router buffer capacity	*20*	*20*
Qw	*0.002*	*0.002*
$D_{\max}$	*0.1*	*0.1*
Number of slots	*2000000*	*2000000*
minthreshold	*3*	*3*
maxthreshold	*9*	*9*
doublemaxthreshold	*18*	– – – – –

the probability of packet arrival was set to several values; each value tends to create a congestion or non-congestion status. The buffer size room of 20 packets was used to detect congestion at small buffer sizes. The total number of slots used in the experiments was 2000000. This value allows the incorporation of accurate performance measures and encapsulates a sufficient warm-up period. The warm-up period is terminated when the system reaches a steady state. The *minthreshold*, *maxthreshold*, $D_{\max}$, and qw values are set to 3, 9, 0.1 and 0.002, respectively, as recommended in RED [5]. Finally, the *doublemaxthreshold* value is set to 18 as recommended in GRED [6]. Table 1 lists all the utilized parameters.

The simulation results are measured using several performance metrics (e.g., *mql*, T, D, P_L, and D_p), which are discussed in the following subsection.

5.1. **Mql, throughput, and delay.** Figures 8-10 illustrate the output performances of RED, GRED, AGRED, and DGRED using different probabilities of packet arrivals. Specifically, Figure 8 illustrates the *mql* versus the probability of packet arrival.

The *mql* for all algorithms and the proposed DGRED algorithm is identical up to certain value of the probability of packet arrival (e.g., 0.33). In such a low probability value, there is at most a light congestion state because the probability of packet arrival is lower than that of packet departure ($a < B$). In such case, all the compared algorithms sustain a good and stable *mql*. However, for a higher probability value, congestion is more likely to exist at the router buffers. Accordingly, the *mql* of the AQM algorithms increases exponentially. The proposed DGRED, on the other hand, performs better than the AQM

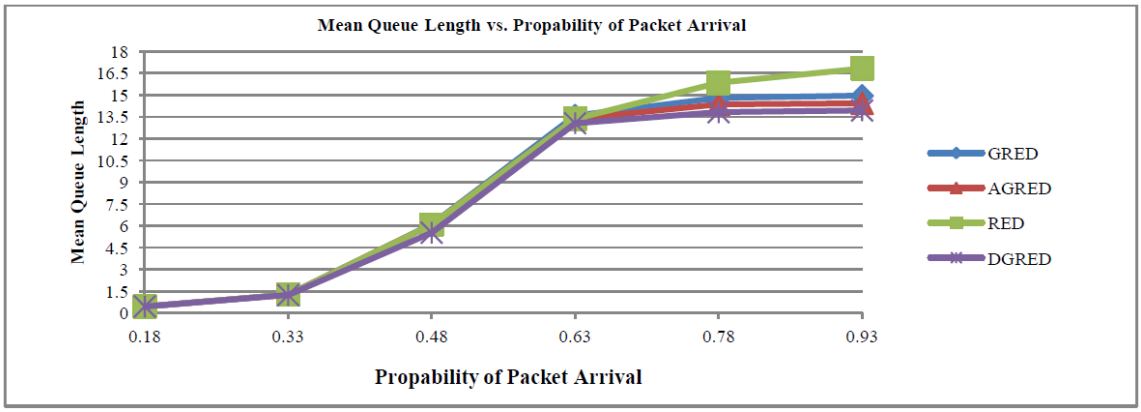

FIGURE 8. *mql* vs. probability of packet arrival

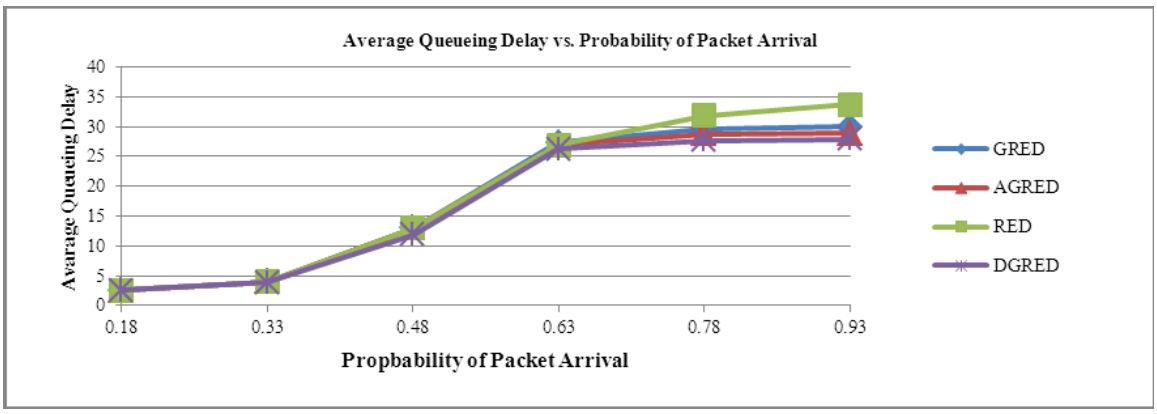

FIGURE 9. D vs. probability of packet arrival

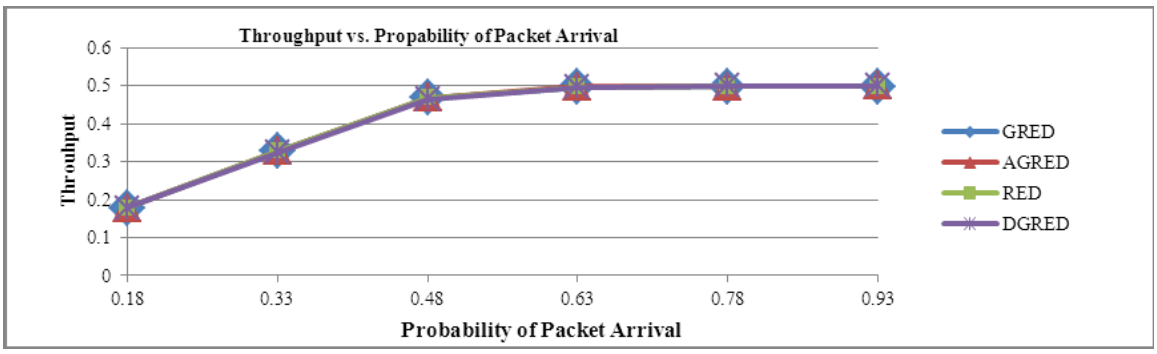

FIGURE 10. T vs. probability of packet arrival

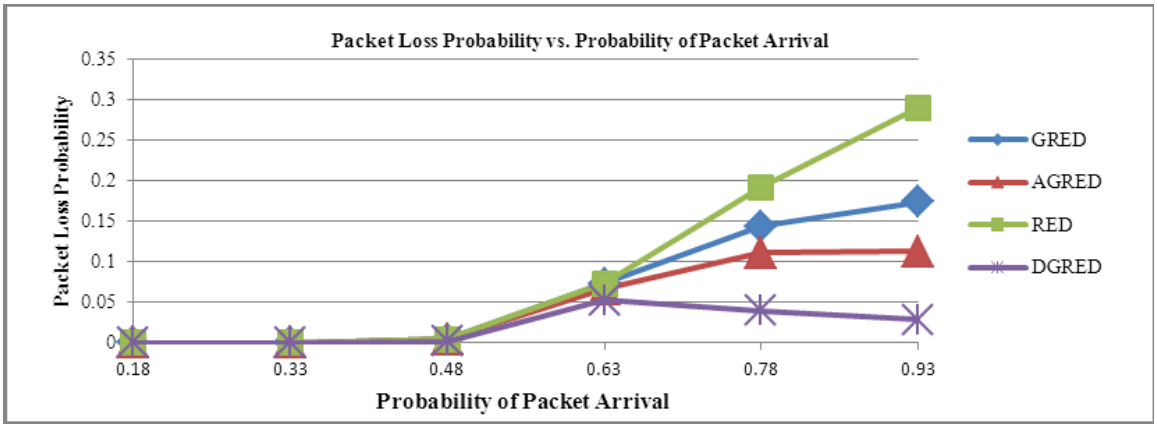

FIGURE 11. P_L vs. probability of packet arrival

algorithms in terms of *mql* at such high probability values. This phenomenon occurs mainly because DGRED drops fewer packets than RED, GRED, and AGRED. Figure 9 illustrates a comparison of the delays in all the algorithms.

Once again, the proposed DGRED performs better in terms of the average queuing delay. However, AGRED also shows good performance in terms of delay. This result is due to the fewer dropped packets in DGRED than those in RED, GRED, and AGRED.

Finally, Figure 10 illustrates the throughput under different packet arrival probabilities. As illustrated, the throughput of the proposed and compared algorithms give similar T results, whether the probability of packet arrival is set to a value lower or higher than the probability of packet departure value. Figure 10 indicates that a packet arrival probability equal to a value lower than the packet departure probability results in the increase of T for the compared algorithms, as long as the packet arrival probability increases. On the contrary, the T results for all compared algorithms are stabilized at the value of the packet departure probability when there is congestion at the router buffer of the algorithms.

5.2. **Packet loss and D_p.** The proposed DGRED algorithm is compared with the RED, GRED, and AGRED algorithms in terms of P_L and D_P in this subsection. The goal of the conducted comparison is to show the quantity of packets dropping at the router buffer in all compared algorithms. The performance measure results of P_L and D_P are computed after the system reaches a steady state. The results of P_L and D_P are obtained as before by running the algorithm simulations ten times with various random seeds, then taking the mean of the ten results. The performances of RED, GRED, ARED, and DGRED algorithms in terms of P_L and D_P are illustrated in Figures 11 and 12, respectively.

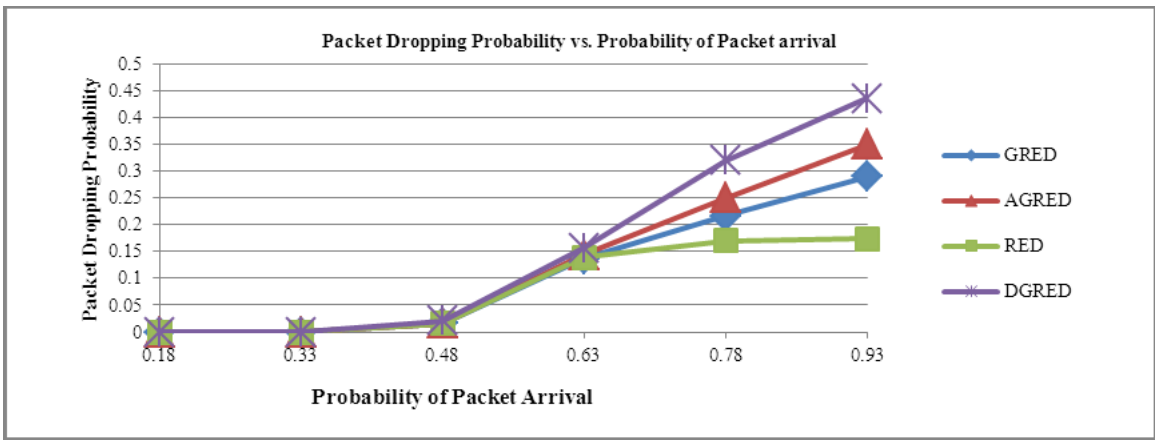

FIGURE 12. D_p vs. probability of packet arrival

In Figure 11, the proposed DGRED algorithm marginally produces the best and least P_L performance when the probability of packet arrival is larger than the probability of packet departure (existence of congestion). This performance is because the router buffer in the DGRED algorithm overflows at an earlier time compared with those in the RED, GRED, and AGRED algorithms. When the packet arrival probability is smaller than the packet departure probability, all algorithms provide similar P_L results under either a light congestion or no congestion situation.

Similarly, in Figure 12, the proposed DGRED algorithm evidently drops more packets at the router buffer than the RED, GRED, and AGRED algorithms when the probability of packet arrival is higher than the probability of packet departure. Similarly, the reason for this result is because the router buffer in the DGRED algorithm overflows at an earlier time compared with those in RED, GRED, and AGRED.

6. **Applications of the DGRED Algorithm.** The proposed DGRED algorithm can be applied as a congestion control technique for wired and wireless networks. Specifically, this algorithm can be applied at the router buffers of wired networks such as the Internet to alleviate overflowing packet loss probability of the router buffers (see Figure 11, which represents the results of the overflowing packet loss probability of the DGRED and other AQM algorithms). As a result, better network performance is accomplished (see Figures 8-11). In addition, applying the DGRED on wireless networks such as base stations of cellular networks can manage congestion by losing fewer packets. This process leads to enhanced network performance.

7. **Conclusions.** The current paper proposed a new AQM algorithm based on the GRED called the DGRED, which identifies congestion at router buffers at an early stage, right before the router buffer overflows. The proposed DGRED aims to stabilize the *aql* between the *minthreshold* and *maxthreshold* by updating the *maxthreshold* and *doublemaxthreshold* positions at the router buffer. This decrease or increase in the *aql* helps stabilize the *aql* at the T_{aql}.

DGRED uses an adaptive *maxthreshold* and *doublemaxthreshold* positions aimed to keep the *aql* between the *minthreshold* and *maxthreshold* at a particular level (T_{aql}), which may lead to fewer packet losses.

The DGRED technique is compared with the GRED, RED, and AGRED algorithms with regard to mql, T, D, P_L, and D_p, to identify which method offers better performance in terms of packet arrival probability parameters.

• The RED, GRED, AGRED, and DGRED algorithms provide similar performance measure results (mql, T, D, P_L and D_p) when the probability of packet arrival is set to a value lower than the probability of packet departure or in the event of light or no congestion.

• The DGRED algorithm marginally offers better mql and D results than the RED, GRED, and AGRED algorithms when the values of the probability of packet arrival are greater than the values of packet departure probability or in the event of heavy congestion. In addition, the RED, GRED, AGRED, and DGRED algorithms obtain similar T results with such values of packet arrival probability.

• The DGRED algorithm marginally outperforms the RED, GRED, and AGRED algorithms for P_L when the value of the probability of packet arrival is larger than the value of the probability of packet departure or in the event of heavy congestion. Moreover, RED, GRED, and AGRED drop fewer packets (Dp) at their router buffers than DGRED at such values of packet arrival probability.

REFERENCES

[1] G. Thiruchelvi and J. Raja, A survey on active queue management mechanisms, *IJCSNS International Journal of Computer Science and Network Security*, vol.8, 2008.

[2] M. Welzl, Network congestion control, *Proc. of Managing Internet Traffic*, Chichester, UK, 2005.

[3] A. S. Tanenbaum, *Computer Networks*, 4th Edition, Prentice Hall Ptr, 2002.

[4] D. Lin and R. Morris, Dynamics of random early detection, *Proc. of ACM SIGCOMM*, New York, NY, USA, pp.127-137, 1997.

[5] S. Floyd and V. Jacobson, Random early detection gateways for congestion avoidance, *IEEE/ACM Transactions on Networking*, pp.397-413, 1993.

[6] S. Floyd, *Recommendations on Using the Gentle Variant of RED*, http://www.aciri.org/floyd /red/gentle.html, 2000.

[7] M. Baklizi et al., Performance assessment of AGRED, RED and GRED congestion control algorithms, *Information Technology Journal*, vol.11, pp.255-261, 2012.

[8] J. Ababneh et al., Derivation of three queue nodes discrete-time analytical model based on DRED algorithm, *Proc. of the 7th International Conference on Information Technology: New Generations*, pp.885-890, 2010.

[9] H. Abdel-jaber et al., Traffic management for the gentle random early detection using discrete-time queueing, *Proc. of International Business Information Management Conference*, Marrakech, Morocco, pp.289-298, 2008.

[10] H. Abdel-Jaber et al., Performance evaluation for DRED discrete-time queueing network analytical model, *Journal of Network and Computer Applications*, vol.31, pp.750-770, 2008.

[11] H. Abdel-jaber et al., Modelling BLUE active queue management using discrete-time queue, *Proc. of 2007 International Conference of Information Security and Internet Engineering*, London, UK, pp.568-573, 2007.

[12] A. Moarefianpour and V. J. Majd, Input-to-state stability in congestion control problem of computer networks with nonlinear links, *International Journal of Innovative Computing, Information and Control*, vol.5, no.8, pp.2091-2106, 2009.

[13] X. Chen, T. Liu and J. Zhao, A logic-based switching control approach to active queue management for transmission control protocol, *International Journal of Innovative Computing, Information and Control*, vol.4, no.7, pp.1811-1820, 2008.

[14] C. Brandauer et al., Comparison of tail drop and active queue management performance for bulk-data and web-like Internet, *Proc. of IEEE ISCC 2001*, pp.122-129, 2001.

[15] R. Stanojevic et al., Adaptive tuning of drop-tail buffers for reducing queueing delays, *IEEE, Communications Letters*, vol.10, pp.570-572, 2006.

[16] A. Bitorika et al., A comparative study of active queue management schemes, *Proc. of IEEE ICC 2004, Congestion Control Under Dynamic Weather Condition*, Ireland, 2004.

[17] S. Ryu, *Active Queue Management (AQM) Based Internet Congestion Control*, Ph.D. Thesis, University at Buffalo Patent, 2002.

[18] J. H. Salim et al., Performance evaluation of explicit congestion notification (ECN) in IP networks, *RFC 2884*, 2000.

[19] H.-Y. Chu, K.-H. Tsai and W.-J. Chang, Fuzzy control of active queue management routers for transmission control protocol networks via time-delay affine Takagi-Sugeno fuzzy models, *International Journal of Innovative Computing, Information and Control*, vol.4, no.2, pp.291-312, 2008.

[20] F. Wu-chang et al., The blue active queue management algorithms, *IEEE/ACM Transactions on Networking*, vol.10, pp.513-528, 2002.

[21] S. Floyd et al., Adaptive RED: An algorithm for increasing the robustness of RED's active queue management, *AT&T Center for Internet Research*, 2001.

[22] J. Aweya et al., A control theoretic approach to active queue management, *Computer Networks*, vol.36, pp.203-235, 2001.

[23] M. E. Woodward, *Communication and Computer Networks: Modelling with Discrete-Time Queues*, IEEE Computer Society Press, Los Alamitos, CA, USA, 1994.