Enhanced Random Early Detection using Responsive Congestion Indicators

Ahmad Adel Abu-Shareha[1]
Information Technology and Computing Department
Arab Open University (AOU)
Riyadh, Saudi Arabia

***Abstract*—Random Early Detection (RED) is an Active Queue Management (AQM) method proposed in the early 1990s to reduce the effects of network congestion on the router buffer. Although various AQM methods have extended RED to enhance network performance, RED is still the most commonly utilized method; this is because RED provides stable performance under various network statuses. Indeed, RED maintains a manageable buffer queue length and avoids congestion resulting from an increase in traffic load; this is accomplished using an indicator that reflects the status of the buffer and a stochastic technique for packet dropping. Although RED predicts congestion, reduces packet loss and avoids unnecessary packet dropping, it reacts slowly to an increase in buffer queue length, making it inadequate to detect and react to sudden heavy congestion. Due to the aforementioned limitation, RED is found to be significantly influenced by the way in which the congestion indicator is calculated and used. In this paper, RED is modified to enhance its performance with various network statuses. RED technique is modified to overcome several disadvantages in the original method and enhance network performance. The results indicate that the proposed Enhanced Random Early Detection (EnRED) and Time-window Augmented RED (Windowed-RED) methods—compared to the original RED, ERED and BLUE methods—enhances network performance in terms of loss, dropping and packet delay.**

Keywords*—Congestion; random early detection; active queue management

I. Introduction

The evolution of the computer network and its broad usability for communication, remote controlling, organizational monitoring and information governing has resulted in the widespread utilization of its resources. Congestion is a phenomenon that occurs on a computer network when the traffic load exceeds the capabilities of these resources. The memory allocated by the network router is the most critical resource in the network that is susceptible to congestion, which can cause delays, packet loss and low network performance [1]. Congestion degrades the quality of services provided to the users and the applications. To predict congestion before it occurs, or before it starts to severely affect performance, various Active Queue Management (AQM) methods have been proposed. Although AQM methods were proposed to overcome the limitations of the first approach, i.e. Random Early Detection (RED)[2], RED is still the most commonly utilized method; this is because RED provides stable performance under various network statuses [3-6].

The AQM methods monitor the status of the router buffers, calculate the dropping probability (*Dp*) for each packet arrival, and implement packet dropping stochastically based on the calculated value. Accordingly, AQM methods take one of the two opposite decisions: packet accommodation and packet dropping. The role of the router is to accommodate the arrival packets to transfer them to the intended destination; However, to avoid congestion, packets are dropped when the buffer overflows and network performance degrading is expected [4]. Generally, the developed AQM methods consist of three main components: the congestion indicators used to monitor the buffer/network status, the function used to calculate *Dp* and the algorithm that determines when to use these equations and indicators [6]. For RED, *Dp* is calculated using a mathematical function with a reference to the average queue length (*aql*), a parameter that reflects the average length over time [7, 8].

The advantages of RED are summarized as follows: (1) RED predicts congestion before it affects network performance and reacts by dropping packets stochastically to avoid the effects of congestion; (2) RED, using random dropping, avoids global synchronization—a phenomenon that occurs when all senders reduce their transmission rates simultaneously for a period and then start increasing them again, also simultaneously; (3) RED uses *aql* to calculate the average queue length over time. Accordingly, when the arrival rate increases for a short while, *aql* does not increase rapidly. Thus, RED avoids dropping packets unnecessarily during short, heavy traffic [9]. Thus, the limitations can be summarized as follows: (1) RED exhibits low performance during sudden congestion, because it uses *aql*, which is insensitive to sudden changes in queue length; (2) RED causes packet delay in heavy traffic, because it does not monitor the router buffer delays [2, 9-11].

Using *aql* as a congestion indicator has its advantages and disadvantages. One advantage is that *aql* is able to avoid false, and thus deceptive, congestion indications. However, the disadvantages are the result of sudden high traffic and unexpected congestion. To overcome this disadvantage, it is recommended to use modified parameters that are equal to *aql* or to ease the dependency on *aql* to enhance network performance. However, RED is found to be badly influenced by the way in which the congestion indicator is calculated and used [4].

In this paper, RED is modified to enhance its performance with various network statuses. RED technique is modified to overcome several disadvantages in the original method and

enhance network performance. Enhanced Random Early Detection (EnRED) and Time-window Augmented RED (Windowed-RED) methods are proposed to address the problem of slow reaction time, and thus increase the number of queued packets in the buffer. The rest of the paper is organized as follows: Section II discusses the RED method in details and the disadvantages to be overcome are clarified. Section III presents the related work. The proposed Work is presented in Section IV; the simulation details are given in Section V and the related results is presented in Section VI. Finally, the conclusion is given in Section VII.

II. RED METHOD

RED is implemented, as given in Algorithm 1, in three stages; these are: (1) *aql* calculation, (2) *Dp* calculation and stochastic packet dropping, (3) buffer tracing.

Algorithm 1: RED

```
1   PARAMETER SETTING: w_q, Th_min, Th_max, D_max
2   VARIABLE INITIALIZATION: aql:= 0, count:= -1
3   FOR-EACH A
4     1) CALCULATE aql:
5      IF (q equal to 0)        → aql:=(1-w)^f(cTime- iTime) * aql
6      IF (q not equal to 0)    → aql:= (1-w)* aql + w_q *q
7     2) CALCULATE Dp & IMPLEMENT packet dropping,
8      IF (min_th ≤ aql < max_th)
9          → counter ++
10         → Dp'= D_max* (aql -Th_min)/( Th_max - Th_min)
11         → Dp = Dp'/ (1- counter * Dp')
12          IF (Drop(Dp) is TRUE)
13              → Drop-A, count := 0
14     ELSE IF (aql > max_th)
15         → Drop-A
16         → counter:= 0
17     ELSE
18          → counter:= -1
19     3) TRACE Buffer Idleness
20     IF (q equal to 0 && Idle equal to False)
21         → iTime=cTime, Idle = TRUE
22     ELSE Idle = False
```

Parameters:

w_q: queue weight
Th_{min}: minimum threshold
Th_{max}: maximum threshold
D_{max}: maximum probability value

Saved Variables:

A: current packet arrival
Dp: packet-dropping probability
Dp': initial packet-dropping probability
cTime: current time
q: current queue size
aql: average queue size
iTime: idle time
Idle: idle status true/false
count: # of packets in medium buffer length that are not dropped. Count is a balanced parameter for *Dp* calculation

In the first stage, *aql* is calculated using two different equations. The first is used if the buffer is currently idle (see line #5). This equation is a function of the previous *aql* and the period of idleness. The second equation is used if the buffer is not idle, and it is a function of the previous *aql* and the current queue size *q*. In the second stage, *Dp* is calculated and the dropping is implemented or skipped according to the three scenarios.

In the first case, as given in line #8, *Dp* is calculated as a function of *aql* and another variable, *counter*, as well as a set of parameters. Subsequently, the packet is dropped or accommodated as a function of the calculated *Dp*, while the *counter* variable is set to zero if the packet is dropped. The *counter* parameter counts the number of packets in the critical case, which were not dropped according to the stochastic decision. The counter is simply increased with each accommodated packet, and is reset when a packet is dropped. As the value of the *counter* increases, the *Dp* value increases significantly. This variable is required to avoid, as much as possible, dropping sequential packets to circumvent global synchronization. In the second case, the packet is dropped and the counter is set to zero [12]. In the third case, the packet is accommodated and the *counter* is set to the value of negative one. A negative *counter* value reduces the dropping probability when the status of the queue jumps from a low-queue state to a high-queue critical state. A *Dp* value that is calculated using a negative value for *counter* will be of a low value. Accordingly, with sudden heavy traffic, the *Dp* value will increase slowly. Finally, in the third stage, the buffer tracing process saves the period of idleness [2, 12, 13].

RED calculates the value of *aql* with each packet arrival. The calculated value is then compared to two pre-determined threshold values (the minimum and maximum thresholds), which divide the buffer into three parts, as illustrated in Fig. 1. The value of *aql* based on the threshold comparison determines the action to be taken, which can be one of the following: (1) If the calculated *aql* value is less than the minimum threshold, then the arrival packet is accommodated with a *Dp* value equal to zero; no dropping occurs when *aql* is below the minimum threshold [14]. (2) If the *aql* value is greater than the maximum threshold, then the arrival packet is dropped with *Dp* equal to one; The arrival packet is firmly dropped when *aql* is greater than the maximum threshold. (3) A stochastic dropping process is implemented when *aql* is between the minimum and the maximum thresholds. In this case, *Dp* is calculated using a mathematical function *Dp(aql)*, which depends on the value of *aql* that affects *Dp* proportionally. As such, *Dp* increases as *aql* increases, and vice versa [14-17].

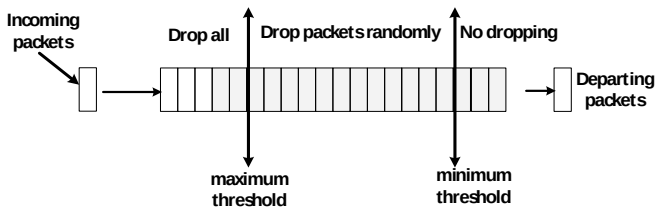

Fig. 1. Buffer Illustration and RED Actions and Parameters.

III. Related Work

Various extensions to the **RED** method have been proposed in the literature to overcome the limitations discussed above. For example, **Gentle RED (GRED)** [12] was proposed to resolve the delay problem in **RED**. **GRED** uses a third threshold parameter, called the double maximum threshold, in addition to the other parameters (as illustrated in Fig. 2). The goal of the introduced threshold is to stabilize *aql* at a specific value. Accordingly, the computed *aql* value is compared to three thresholds: minimum, maximum and double maximum. This creates four cases for the calculated *Dp* rather than three. This extension, however, increases the dependency on parameter settings. Moreover, **GRED** is more sensitive to sudden congestion, because it reduces packet dropping compared to **RED.** Moreover, **GRED** exhibits poor performance when heavy congestion occurs while *aql* is below the maximum threshold. Thus, **Adaptive RED (ARED)** [13] was proposed to resolve the delay problem in **RED** while preserving the RED mechanism as much as possible. **ARED** used *aql*, *minimum* and *maximum* thresholds as the main parameters, in addition to a new parameter that represents the optimal target of the queue length, called target *aql* (*Taql*). Accordingly, *aql* is compared with three values in **ARED**, and *Dp* is calculated the same as in **RED**. However, the value of the initial dropping, D_{max}, which is fixed in **RED**, is calculated adaptively in **ARED.** The D_{max} value increases or decreased based on the value of *aql* compared to *Taql*.

Other methods extend RED by marginally modifying the original algorithm. For example, **PI** [18] uses traffic load value and *aql* value to calculate *Dp*. Accordingly, dropping increases when traffic load increases when *aql* is low to reduce delay. **Dynamic RED (DRED)** [19] extends **RED** by comparing instance queue length to a single threshold value. When queue length is below the threshold, no dropping is implemented. Moreover, when queue length is above the threshold, *Dp* increases or decreases based on queue length and the previous *Dp* value. **Random Exponential Marking (REM)** [20] extends **RED** using instance queue length, instead of *aql*, combined with the estimated load rate. However, instance queue length causes unnecessary packet dropping and false congestion. **BLUE** method [21] uses an adaptive value of *Dp*, which increases or decreases based on the estimated congestion status; This is similar to **ARED** and **DRED** in regard to packet loss and the threshold value. Various other AQM methods have been proposed in the literature, with different modifications made to the original RED mechanism for different purposes, as summarized in Table I.

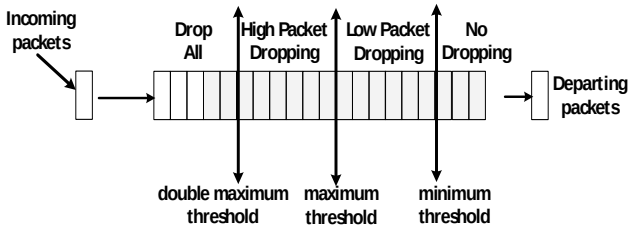

Fig. 2. Buffer Illustration and GRED Actions and Parameters.

TABLE I. Summary of RED's Extensions

Method	Modification	Objectives
RED [2]	Original	Avoid global synchronization, reduce packet loss and dropping
GRED [12]	Use one more threshold to stabilize *aql*	Reduce delay
ARED [13]	Adaptively increase the initial dropping value based on *aql*	Reduce delay
PI [22]	Use load parameter (besides *aql*) to calculate *Dp*	Reduce delay
DRED [19]	Adaptively increase the initial dropping value based on *ql*	Reduce delay
REM [20]	Adaptively increase the initial dropping value based on *ql and load rate*	Reduce loss and maximize resource utilization
BLUE [21]	Adaptively increase the initial dropping value based on *packet loss*	Reduce packet loss
SRED [23]	Adaptively increase *Dp* value based on *aql* and number of active flows	Fair resource allocation
ERED [24]	Use *q* (besides *aql*) to calculate *Dp*.	Reduce packet loss
MRED [25]	Use the heuristic method to calculate *Dp*	Reduce packet loss
AVQ [26]	Use *delay* to calculate *Dp*	Reduce round-trip delay
RaQ[27]	Add more equations that calculate *Dp* based on *aql*	Maximize resource utilization
Yellow [28]	Use *load* to calculate *Dp*	Maximize resource utilization
CRED [3]	Use cloud membership degree calculation to calculate *Dp*	Reduce packet dropping
Adaptive Threshold RED [16]	Use rules set to calculate *Dp* based on *aql* and *q*	Reduce packet dropping
Adaptive-AQMRD [29]	Adaptive parameter tuning	Reduce packet dropping and solve parameterization problem
FLRED [8]	Use fuzzy logic to calculate *Dp* based on *aql* and delay	Solve parameterization problem

Overall, existing AQM methods can be broadly classified into two groups : The first involves methods that preserve the RED mechanism and parameters for monitoring and reacting, such as GRED [12] and ARED [13]; The other group contains methods that implemented major modifications while preserving the overall concepts of stochastic packet dropping.

Accordingly, the similarities among the existing AQM methods are: (1) AQM depends on stochastically packet dropping to avoid global synchronization; (2) AQM dropping probability is calculated with reference to the buffer or the network status; (3) AQM buffer monitoring is tracked and the decision-making is activated with each packet arrival.

The differences between the AQM methods can be summarized as follows: (1) Different congestion indicators and

parameters (such as *aql*, *q*, *load*, *delay*, *loss*) are utilized by different AQM methods. (2) Different decision-making scenarios are used for different AQM methods; RED used three scenarios (firmly dropping, stochastic dropping and no dropping), while other methods keep the stochastic dropping and add or remove other scenarios. (3) Different AQM methods use different heuristic equations and methods to calculate the dropping probability. These differences are motivated by different objectives, such as to reduce packet loss, reduce packet dropping, reduce round-trip delay, maximize resource utilization, and fair allocation. The first four objectives are conflicted, as reducing delay will minimize resource utilization and reducing dropping will increase delay.

RED is a well-known, stable AQM technique that was adapted by the Internet Engineering Task Force (IETF) in RFC 2309. For a long time, RED was utilized in distributed routers all over the world. Extended methods provide different capabilities and cause different QoS. Accordingly, replacing RED with different methods will harm the stability of the existing systems. Thus, it is necessary to enhance RED while maintaining its characteristics and configuration.

IV. Proposed Work

As network technology evolves, the limitations of the RED have been discovered. RED can be viewed as a dual-mechanism method, in which the first mechanism is a multi-reaction process activated by the implemented algorithm and all the utilized parameters, except for *aql*. This mechanism reacts differently based on the status of the queue, as discussed above, and it is the reason behind the first and the second advantages of the RED method. The second mechanism is the RED congestion alert mechanism, which is implemented using *aql*. This mechanism is related to the third advantage and the first disadvantage. Accordingly, it is easier to point at the source of the limitation, which can be referred back to the utilization of *aql*. RED uses *aql* for two tasks: (1) As an indicator for monitoring to determine the reaction scenario; (2) As a parameter for dropping the probability calculation.

In fact, all of these tasks use *aql* in different ways. Since using *aql* has both positive and negative effects, replacing *aql* in all of these tasks is not a good approach; Instead, RED can use different bounds and parameters to gain an advantage and eliminate its disadvantages.

In the dropping calculation, using *aql* has proven to have the following advantages: (1) Avoid high dropping with limited increase in buffer length; (2) Avoid high delay with limited decrease in buffer length. These advantages refer to slow changes in *aql* value with the changes in the buffer length, which can be temporarily according to the busty nature of the traffic. To determine the reaction scenario, using three scenarios for packet dropping (based on a threshold comparison) is sufficient and has several advantages. However, using *aql* as a monitoring indicator has the disadvantage of providing an unreal indication of the queue status. This disadvantage refers to variations in *aql* as a result of sudden congestion. Because the slow variation has both advantages and disadvantages, the disadvantage of using *aql* is directly related to the monitoring of the queue rather than the dropping calculation. Accordingly, the decision between fully dropping, no dropping, and stochastic dropping should be made based on the status of the buffer. Thus, instead of using *aql* to make a suitable decision, other indicators should be used to determine buffer status. Accordingly, an enhanced random early detection (EnREd) method and Windowed RED (WRED) method based on simple, yet efficient, monitoring parameters are presented.

A. Indicator Calculation

Two indicators are used to replace *aql* for congestion monitoring: instance queue length (q) and average queue length on a limited time window (aql_w). At any time, the number of packets queued in the router buffer is referred to as *q*. The advantages of using this indicator are: (1) No extra calculation is required, as *q* is used in RED to calculate *aql*; (2) Overcome the disadvantages of using *aql* in the original RED. The indicator aql_w represents an equally averaged packet number on a limited time window. Unlike *aql*, which is calculated as a weighted average with low weights for current length, aql_w is calculated as the equal-weight average for a predetermined time window. Accordingly, aql_w intermediate *q* and *aql* in considering of past and recent queue length as illustrated in Fig. 3 [8]. The advantages of using this indicator are: (1) Can fully replace the existing *aql* for monitoring, decision making and probability calculation; (2) Overcome the disadvantages of using *aql* in the original RED, to some extent.

B. EnRED Mechanism

The proposed EnRED uses *q* to determine the reaction scenario for packet dropping as each packet arrives. This process requires a simple and direct modification to the original RED algorithm. As given in Algorithm 2, RED is modified at line #8 and line #14 by comparing *q* with the two thresholds, instead of using *aql* for comparison, as in the original RED. EnRED is implemented, as in Algorithm 2, in three stages. In the first stage, *aql* is calculated using two different equations. The first is used if the buffer is currently idle, as in line #5. The second is used if the buffer is not idle, and is a function of the previous *aql* and the current queue size *q*. In the second stage, *Dp* is calculated and the dropping is implemented or skipped according to three implemented scenarios. The first scenario, as given in line #8, is implemented when the queue length is between the minimum and maximum threshold. Queue length provides a true indication of buffer status, regardless of network status. Nevertheless, *Dp* is calculated as a function of the *aql* that reflects the status of the network. Subsequently, the packet is dropped or accommodated as a function of the calculated *Dp*, while the *counter* variable is set to zero if the packet is dropped. As the value of *counter* increases, the *Dp* value increases significantly, and vice versa. This variable is required to avoid, as much as possible, dropping sequential packets to circumvent global synchronization. The second case is implemented when the queue length exceeds the maximum threshold. In this case, the packet is dropped and the counter is set to zero. Finally, the third case is implemented when the queue length is below the minimum threshold, and results in packet accommodation and *counter* reset to the value of negative one. In the third stage, the buffer tracking process saves the period of idleness. Accordingly, packet monitoring in EnRED is the responsibility of the parameter *q*, while the dropping value is calculated based on *aql*.

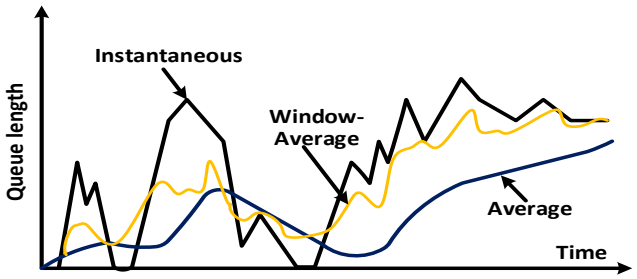

Fig. 3. The Shape of Different Indicators Overtime.

Algorithm 2: EnRED

```
1   PARAMETER SETTING: wq, Thmin, Thmax, Dmax
2   VARIABLE INITIALIZATION: aql:= 0, count:= -1
3   FOR-EACH A
4       1)  CALCULATE aql:
5        IF (q equal to 0)          → aql:=(1-w)^f(cTime- iTime) * aql
6        IF (q not equal to 0)      → aql:= (1-w)* aql + wq *q
7       2)  CALCULATE Dp & IMPLEMENT packet dropping,
8        IF (minth ≤ q < maxth)
9            → counter ++
10           → Dp'= Dmax* (aql -Thmin)/( Thmax - Thmin)
11           → Dp = Dp'/ (1- counter * Dp')
12           IF (Drop(Dp) is TRUE)
13               → Drop-A, count := 0
14       ELSE IF (q > maxth)
15           → Drop-A
16           → counter:= 0
17       ELSE
18           → counter:= -1
19      3)  TRACE Buffer Idleness
20       IF (q equal to 0 && Idle equal to False)
21           → iTime=cTime, Idle = TRUE
22       ELSE Idle = False
```

C. Time-Window Augmented RED (Windowed-RED)

The proposed Windowed-RED (also WRED) is implemented using aql_w to determine the reaction scenario and to calculate dropping probability (*Dp*). This process required replacing all instances of *aql* with aql_w in the original RED algorithm; aql_w is implemented based on two scenarios, as given in lines #5 and #6. In the initial stage, aql_w is set to a value equal to *q*. Subsequently, aql_w is calculated on a window size that is equal to the buffer size; the window-tailed boundary is the current queue length. The dropping calculation and the dropping scenarios will be calculated based on aql_w, as given in lines #8–14. The tracing empty buffer is eliminated, as the utilized aql_w does not require empty buffer tracing. As in Algorithm 3, Windowed-RED is implemented in two stages: aql_w calculation and *Dp* calculation and stochastic packet dropping. In the first stage, aql_w is calculated in two ways. The first is used when Window-RED starts running (see line #5). In this case, aql_w is set to a value equal to *q*. While the second is used throughout the execution of the Window-RED, and is calculated as window-average of *q*. In the second stage, *Dp* is calculated and the dropping is implemented or skipped according to three implemented scenarios. In the first case, as given in line #8, *Dp* is calculated as a function of aql_w and *counter*. Then, the packet is dropped or accommodated as a function of the calculated *Dp*, while the *counter* variable is set to zero if the packet is dropped. As the value of *counter* increases, the *Dp* value increases as well, and vice versa. This variable is maintained to avoid global synchronization. In the second case, the packet is dropped and the counter is set to zero if aql_w is greater than the maximum threshold. In the third case, the packet is accommodated and the *counter* is set to the value of negative one if aql_w is less than the minimum threshold. Accordingly, packet monitoring in Windowed-RED and dropping calculation is the responsibility of the parameter aql_w, which is more sensitive to queue changes than *aql*.

Algorithm 3: WRED

```
1   PARAMETER SETTING: wq, Thmin, Thmax, Dmax
2   VARIABLE INITIALIZATION: aql:= 0, count:= -1
3   FOR-EACH A
4       1)  CALCULATE aql:
5        IF (#ArrivedPackets < BufferSize)  → aqlw:=q
6        Else → aqlw:= A = Σ_{i=cTime}^{(cTime−BufferSize)+1} qi /BufferSize
7       2)  CALCULATE Dp & IMPLEMENT packet dropping,
8        IF (minth ≤ aqlw < maxth)
9            → counter ++
10           → Dp'= Dmax* (aqlw -Thmin)/( Thmax - Thmin)
11           → Dp = Dp'/ (1- counter * Dp')
12           IF (Drop(Dp) is TRUE)
13               → Drop-A, count := 0
14       ELSE IF (aqlw > maxth)
15           → Drop-A
16           → counter:= 0
17       ELSE
18           → counter:= -1
```

V. Simulation and Parameter Settings

In the simulation process, the router buffer is modelled as first-in-first-out (FIFO). The network as a whole is simulated using the discrete time queue model, which is commonly used in simulating and capturing the performance of AQM methods. Generally, the discrete time queue is represented by a sequence of time slots, each of which has a single departure process, a departure and arrival process, a single arrival or none of these. Accordingly, in each time slot, the arrival and departure process simulates the stochastic process based on a predefined arrival and departure rate. A discrete time queue implements the departure process, d_n, before arrival. Accordingly, d_n occurs at time slot *n-1*. Thus, the performance of the network can be captured accurately by counting and tracing the arrived, departure, lost, dropped, and other packets at each time slot [30, 31]. Eventually, these traced counters can be used to calculate the network performance accurately [4, 7, 32]. The simulation parameters are set to match the parameters recommended in the literature for RED; these parameters are listed in Table II. The arrival and departure rates are also set to create different traffic loads and congestion statuses.

TABLE II. SUMMARY OF RED'S EXTENSIONS

Parameter	Values
Probability of Packet Arrival	0.3–0.95
Probability of Packet Departure	0.3, 0.5
Total Number of Slots	2,000,000
Number of Slots for Warm-Up Period	800,000
Number of Slots for Results	1,200,000
Capacity of the Router Buffer	20
Queue Weight for *aql* Calculation	0.002
D_{max}	0.1
min_{th}	3
max_{th}	9

VI. RESULTS

The performance of the proposed EnRED and WRED is evaluated and compared to the original RED method, as well as to the BLUE and ERED methods. The performance is captured using the set of commonly utilized measures for network evaluation, which are delay, packet dropping, packet loss, and total packet missing at the router buffer. When the number of dropped packets using method *A* is more than the number of dropped packets using method *B*, and while loss is less in method *A*, then method *A* is considered to be more efficient than method *B*. Accordingly, to clearly illustrate this indistinct relationship between loss and drop using AQM methods, the total missing indicator is used in addition to the well-known measures.

The first experiment evaluates the proposed methods in comparison with the other methods under **extremely heavy traffic** load. Accordingly, the arrival probability, α, is set to a value of 0.9; the departure probability, β, is set to 0.3. According to the results depicted in Fig. 4, packet loss in EnRED and WRED is less than all other methods (Fig. 4(a)). Moreover, BLUE and RED lost less packets compared to ERED, which seems to perform badly under heavy traffic. In Fig. 4(b), the packet dropping rate of EnRED, WRED and RED is better than the rate of BLUE. ERED drops less packets compared to all other methods. Fig. 4(c) illustrates the total number of packets lost and dropped. All the methods are equal according to this measure, which means that the method with less packet loss is better. Accordingly, EnRED and WRED outperform all other methods; EnRED is slightly better than WRED, and RED is better than ERED and BLUE. Accordingly, the experiments showed that the proposed methods preserve the network performance and deal more efficiently with the queued packets compared to all other methods, as the utilized congestion indicators are more accurate than the indicators used in the other methods. Finally, in terms of packet delay, BLUE outperforms all other methods, while the proposed EnRED slightly outperforms RED and WRED, as illustrated in Fig. 4(d). According to the results presented in Fig. 4, packet dropping using the indicators selected in the proposed methods is much better than packet dropping in the original RED. This is because the utilized indicator is better at sensing congestion and avoiding packet loss. Dropping and delay of the proposed methods are not the best this is because the proposed methods avoid loss. Nevertheless, delay and dropping for the proposed method are shown to be comparable with the best methods, except for ERED, for which the loss rate is huge.

The second experiment evaluates the compared methods under a **heavy traffic** load. Accordingly, the arrival probability is set to a value of 0.9, while the departure probability is set to 0.5. The results of this experiment, as illustrated in Fig. 5, are almost similar to the obtained results in the first experiment. Packet loss, using EnRED and WRED, is less than in all other methods (Fig. 5(a)). Moreover, BLUE and RED drop less packets compared to ERED. Fig. 5(b) reveals that the packet dropping of EnRED, WRED and RED is better than all other methods. Moreover, ERED drops less packets compared to all other methods. Fig. 5(c) illustrates the total packets lost and dropped. Accordingly, EnRED and WRED outperform the other methods, while EnRED is lightly better than WRED, and RED is better than ERED and BLUE. Finally, in terms of packet delay, BLUE outperforms all other methods, while the proposed EnRED slightly outperforms RED and WRED, as illustrated in Fig. 5(d).

The third experiment evaluates the methods based on **moderate traffic** load. Accordingly, both the arrival probability and departure probability are set to a value of 0.5. The results of this experiment are illustrated in Fig. 6. Loss is avoided by all methods (Fig. 6(a)). Similarly, dropping is not necessary when the traffic load is moderate, and thus the evaluated methods did not implement packet dropping except for the BLUE method, which scarified unnecessary packets (Fig. 6(b)). Fig. 6(c) reveals that BLUE is the worst in such a case, because it drops unnecessary packets. Finally, Fig. 6(d) illustrates that delay is almost equal in all evaluated methods except for the BLUE method, which exhibits an improved delay due to the implemented dropping. The fourth experiment evaluates and compares the proposed methods based on **light traffic** load, as illustrated in Fig. 7. Accordingly, the arrival probability is set to a value of 0.3 and the departure probability is set to 0.5. Moreover, no dropping or packet loss occurred, and the results are almost identical for all the compared methods.

Fig. 8 presents and compares the results of the various methods using the average of several arrival probabilities with a departure probability of 0.5. The results indicate that RED, EnRED and WRED are the best in terms of packet dropping, while BLUE outperforms ERED. For dropping, ERED is the best, while RED, EnRED and WRED outperform the BLUE method. In terms of delay, BLUE is the best, while EnRED outperform RED and WRED, which outperforms ERED. Accordingly, the proposed EnRED method and Windowed-RED method enhance network performance, avoid congestion and preserve delay in extremely heavy, heavy and moderate traffic. As noted, replacing the congestion indicator with a firm indicator or tightening the loose indicator with a more compact indicator in the windowed-RED significantly enhances RED performance.

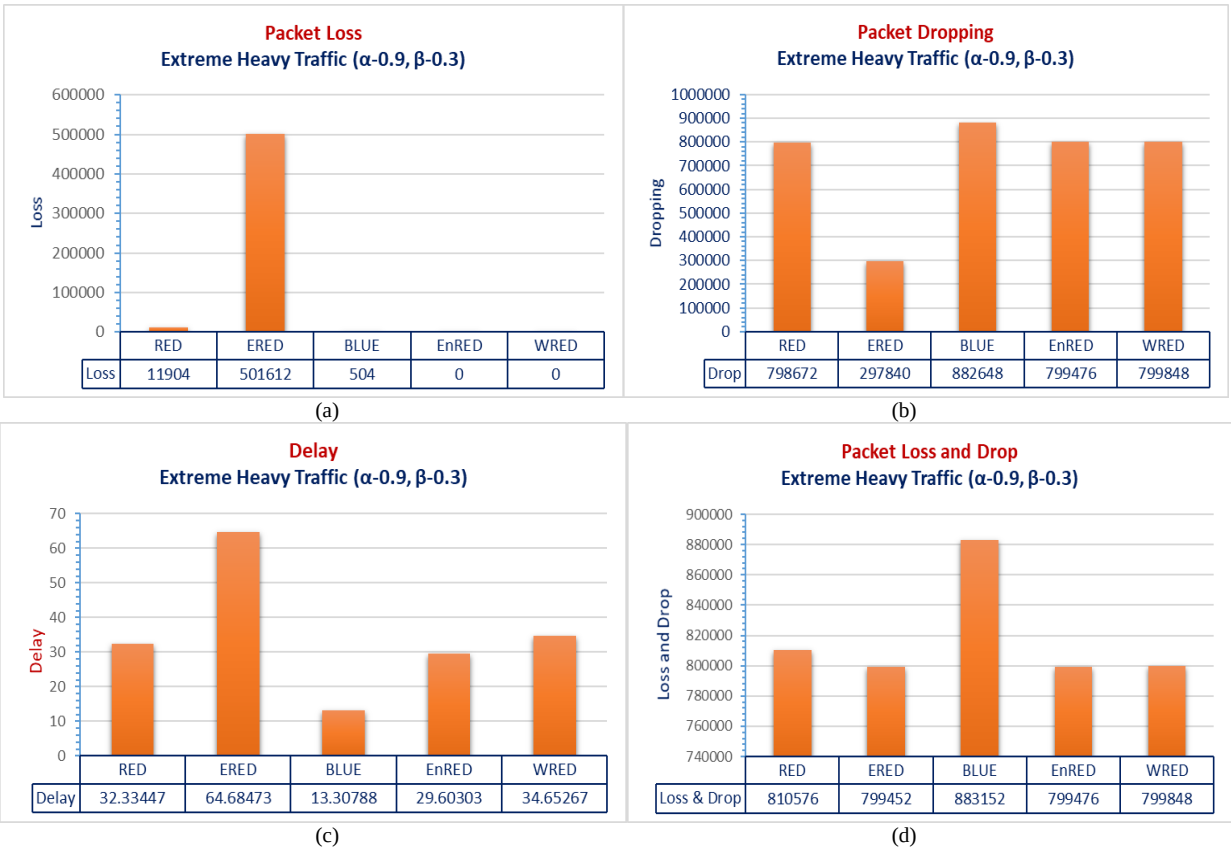

Fig. 4. Performance Results under Extremely Heavy Traffic.

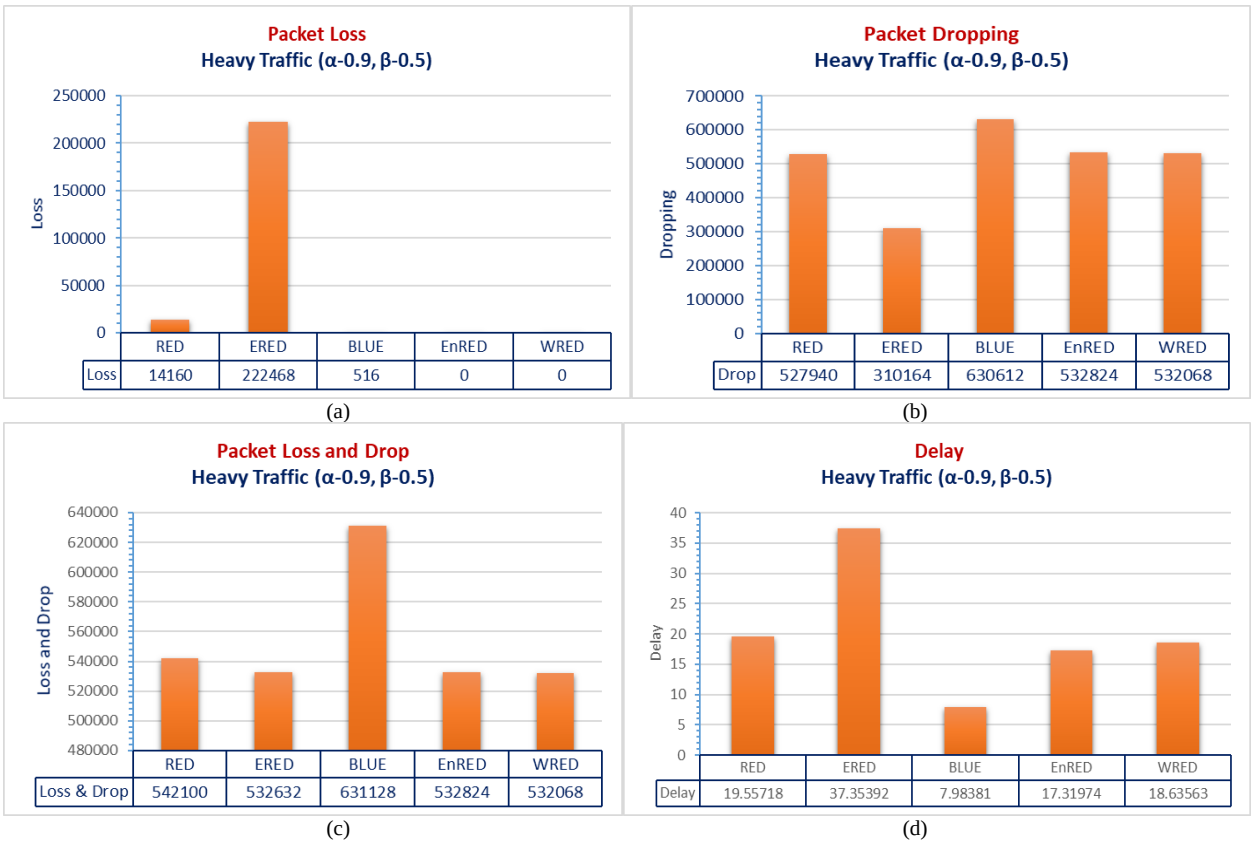

Fig. 5. Performance Results under Heavy Traffic.

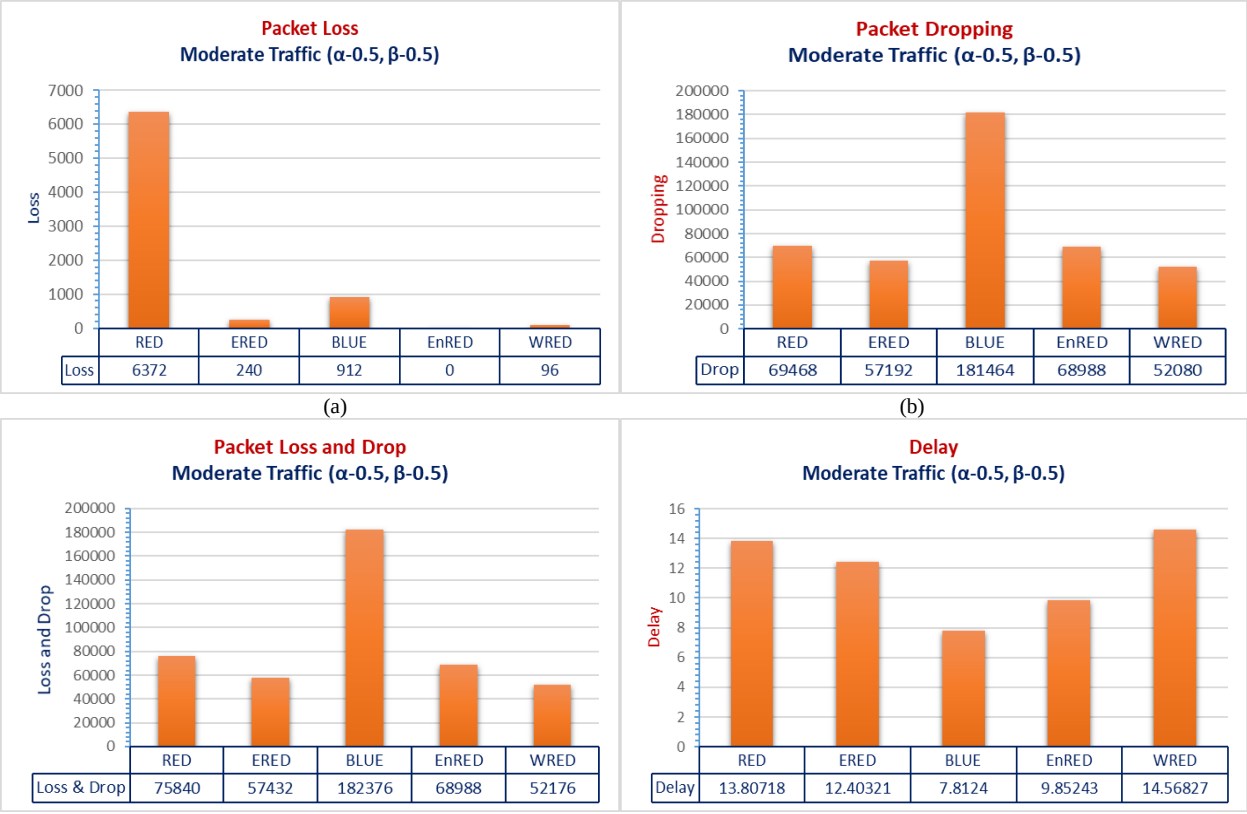

Fig. 6. Performance Results under Moderate Traffic.

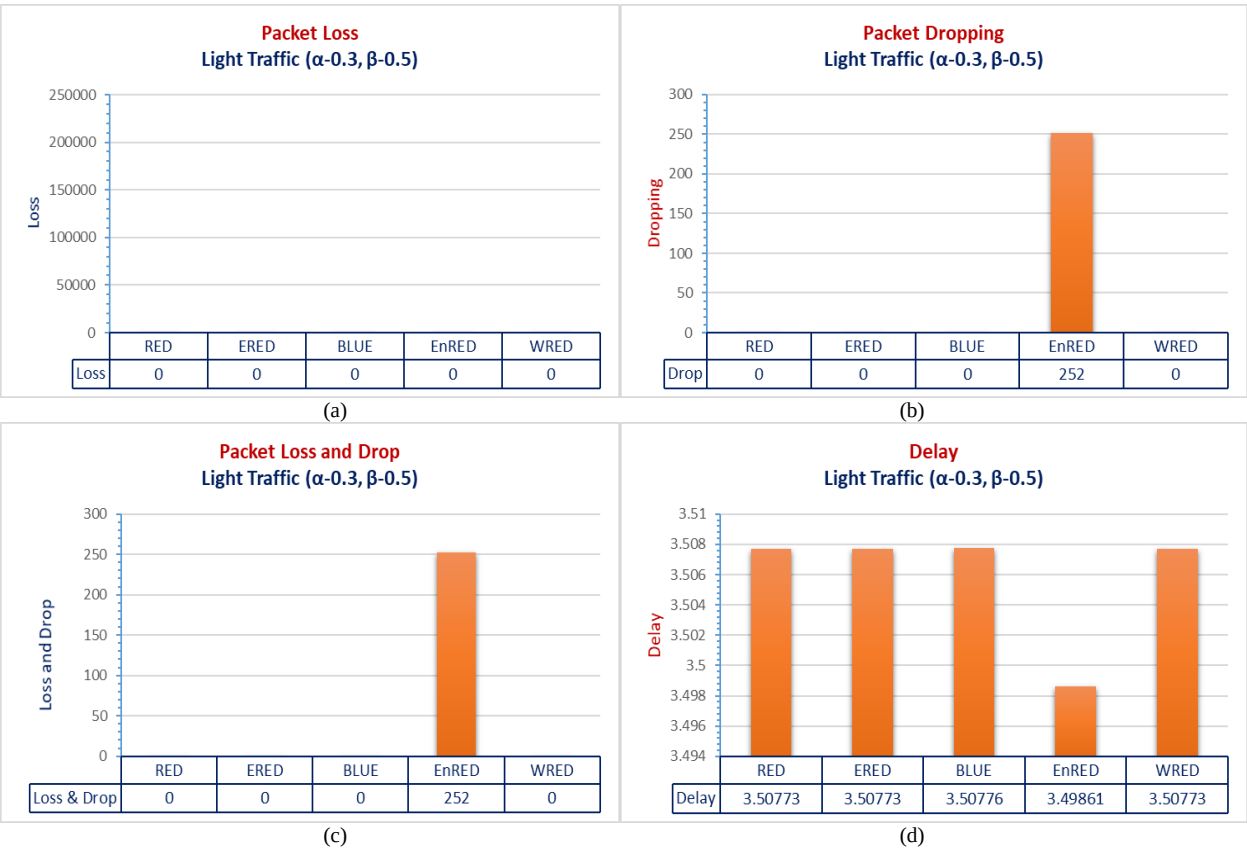

Fig. 7. Performance Results under Light Traffic.

Fig. 8. Performance Results for Various Arrival Rates.

VII. Conclusion

In this paper, the RED technique is analyzed, and the source of the shortcomings is identified. The loose congestion indicator in RED that leads to packet loss is fixed using two approaches: First, by preserving all the steps in the original RED and replacing the loose congestion indicator with a firmer one. Secondly, by replacing the loose congestion indicator and with a more confirmed one. Thus, both methods preserve the form of the original RED, and consequently, the stability of the existing systems that use RED. Although WRED is not as good as EnRED, it has the advantage of reducing the number of utilized parameters, and thus reducing the sensitivity of parameter initialization. The results also confirm that RED outperforms both ERED and BLUE. Future work should focus on using other indicators that enhance network performance and reduce overall delay.

References

[1] Sharma, N., et al., P-RED: Probability Based Random Early Detection Algorithm for Queue Management in MANET, in Advances in Computer and Computational Sciences. 2018, Springer. p. 637-643.

[2] Floyd, S. and V. Jacobson, Random early detection gateways for congestion avoidance. IEEE/ACM Trans. Netw., 1993. 1(4): p. 397-413.

[3] Zhao, Y., et al., An Improved Algorithm of Nonlinear RED Based on Membership Cloud Theory. Chinese Journal of Electronics, 2017. 26(3): p. 537-543.

[4] Yu-hong, Z., Z. Xue-feng, and T. Xu-yan, Research on the Improved Way of RED Algorithm S-RED. International Journal of u-and e-Service, Science and Technology, 2016. 9(2): p. 375-384.

[5] Briscoe, B., Insights from Curvy RED (Random Early Detection). 2015, Technical report TR-TUB8-2015-003, BT.

[6] Jamali, S., N. Alipasandi, and B. Alipasandi, An Improvement over Random Early Detection Algorithm: A Self-Tuning Approach. Journal of Electrical and Computer Engineering Innovations, 2014. 2(2): p. 57-61.

[7] Baklizi, M., et al., Fuzzy logic controller of gentle random early detection based on average queue length and delay rate. International Journal of Fuzzy Systems, 2014. 16(1): p. 9-19.

[8] Abualhaj, M.M., A.A. Abu-Shareha, and M.M. Al-Tahrawi, FLRED: an efficient fuzzy logic based network congestion control method. Neural Computing and Applications, 2018. 30(3): p. 925–935.

[9] Chen, W. and S.-H. Yang, The mechanism of adapting RED parameters to TCP traffic. Computer Communications, 2009. 32(13): p. 1525-1530.

[10] Seifaddini, O., A. Abdullah, and a.H. Vosough, RED, GRED, AGRED CONGESTION CONTROL ALGORITHMS IN HETEROGENEOUS TRAFFIC TYPES, in International Conference on Computing and Informatics. 2013.

[11] Alshimaa, I., et al., Enhanced Random Early Detection (ENRED). International Journal of Computer Applications, 2014. 92.

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

[13] Floyd, S., R. Gummadi, and S. Shenker, Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management. AT&T Center for Internet Research at ICSI, 2001.

[14] Baklizi, M., et al., Performance assessment of AGRED, RED and GRED congestion control algorithms. Information Technology Journal, 2012. 11(2): p. 255.

[15] Marin, A., et al. Performance evaluation of AQM techniques with heterogeneous traffic. in 2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC). 2016. IEEE.

[16] Patel, Z.M. Queue occupancy estimation technique for adaptive threshold based RED. in Circuits and Systems (ICCS), 2017 IEEE International Conference on. 2017. IEEE.

[17] Sun, J., M. Zukerman, and M. Palaniswami, Stabilizing RED using a fuzzy controller, in International Conference on Communications (ICC'07). 2007, IEEE. p. 266-271.

[18] Silva, G.J., A. Datta, and S.P. Bhattacharyya, PI stabilization of first-order systems with time delay. Automatica, 2001. 37(12): p. 2025-2031.

[19] Aweya, J., M. Ouellette, and D.Y. Montuno, A control theoretic approach to active queue management. Comput. Netw., 2001. 36(2-3): p. 203-235.

[20] Lapsley, D. and S. Low. Random early marking: an optimisation approach to Internet congestion control. in Networks, 1999. (ICON '99) Proceedings. IEEE International Conference on. 1999.

[21] Feng, W.-c., et al., BLUE: A New Class of Active Queue Management Algorithms. 1999, University of Michigan, Ann Arbor, MI, Technical Report.

[22] Hollot, C.V., et al. On designing improved controllers for AQM routers supporting TCP flows. in INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. 2001. IEEE.

[23] Ott, T.J., T.V. Lakshman, and L. Wong. SRED: stabilized RED. in INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. 1999.

[24] Abbasov, B. and S. Korukoglu, Effective RED: An algorithm to improve RED's performance by reducing packet loss rate. Journal of Network and Computer Applications, 2009. 32(3): p. 703-709.

[25] Koo, J., et al. MRED: a new approach to random early detection. in Information Networking, 2001. Proceedings. 15th International Conference on. 2001. IEEE.

[26] Kunniyur, S. and R. Srikant, End-to-end congestion control schemes: Utility functions, random losses and ECN marks. Networking, IEEE/ACM Transactions on, 2003. 11(5): p. 689-702.

[27] Sun, J. and M. Zukerman, RaQ: A robust active queue management scheme based on rate and queue length. Computer Communications, 2007. 30(8): p. 1731-1741.

[28] Long, C., et al., The Yellow active queue management algorithm. Computer Networks, 2005. 47(4): p. 525-550.

[29] Bhatnagar, S. and S. Patel, A stochastic approximation approach to active queue management. Telecommunication Systems, 2018. 68(1): p. 89-104.

[30] Khatari, M. and G. Samara, Congestion control approach based on effective random early detection and fuzzy logic. arXiv preprint arXiv:1712.04247, 2017.

[31] Guizani, M., et al., Network modeling and simulation: a practical perspective. 2010: John Wiley & Sons.

[32] Tsavlidis, L., P. Efraimidis, and R.-A. Koutsiamanis, Prince: an effective router mechanism for networks with selfish flows. Journal of Internet Engineering, 2016. 6(1).