RED-I: A RED-Based Algorithm for Internet Routers

Samuel O. Hassan[1], Chigozirim Ajaegbu[2], Samson O. Ogunlere[3], Richmond U. Kanu[4], and Olusola S. Maitanmi[5]
[1]Department of Mathematical Sciences, Olabisi Onabanjo University, Ago-Iwoye, Nigeria
[2]Computer Science Department, Babcock University, Ilishan-Remo, Nigeria
[3]Information Technology Department, Babcock University, Ilishan-Remo, Nigeria
[4]Department of Basic Sciences, School of Science and Technology, Babcock University, Nigeria
[5]Software Engineering Department, Babcock University, Ilishan-Remo, Nigeria
Email: samuel.hassan@oouagoiwoye.edu.ng; ajaegbuc@babcock.edu.ng; ogunleres@babcock.edu.ng; kanur@babcock.edu.ng; maitanmio@babcock.edu.ng

Abstract**—**An essential goal of Active Queue Management (AQM) algorithm is to improve delay by keeping the average queue size low and maintain a high throughput. However, Random Early Detection (RED) and its various improved variants have not been able to achieve much in this regard. This paper is concerned with the problem of RED in terms of large delay resulting from RED's inability to keep the average queue size small. Therefore, we propose a new RED-based AQM algorithm named RED-Improved (RED-I) which utilizes two linear packet dropping functions. ns-3 simulation experiments to compare the performance of the proposed RED-I with RED confirmed that RED-I outperformed RED in terms of delay especially at both light and heavy traffic load scenarios. Replacing/upgrading the RED algorithm implementations in Internet routers (either software or hardware) requires minimal effort since only the packet dropping probability profile needs to be adjusted.

Index Terms**—**Packet dropping probability, active queue management, delay, QoS, Internet routers

I. Introduction

In the current Internet, congestion is a key concern as it has a deteriorating effect on Quality of Service (QoS) for end-users on the network. Network congestion occurs when the amount of transmitted data packets exceeds the buffer size of the network resource. In order to ensure an improved network performance, it is important to avoid congestion [1]. Internet routers queue management algorithms can be classified into two broad categories: Active Queue Management (AQM) and Passive Queue Management (PQM). The traditional Drop-Tail algorithm which was initially used as a default algorithm for queue management in the router employs the concept of FIFO (First-In, First-Out), which means that packets leaves the buffer in their order of arrival. The algorithm enqueues packet at the tail of the buffer and dequeues packet at the head of the buffer. When the buffer is full, all arriving incoming packet are dropped. Drop-Tail algorithm is also referred to as PQM. The weaknesses of Drop-Tail algorithm are: large delay in data delivery, buffer overflow, high packet loss rate, global synchronisation, and network deadlock [2]. Unlike Drop-Tail algorithm, AQM algorithms detects congestion at an early stage and sends notification to connections to back-off by dropping incoming packets before the queue is full.

The Random Early Detection (RED) AQM algorithm developed by [3] consists of two computational sections. The first section is used for computing the average queue size for detecting incipient congestion while the second part is used to compute the packet dropping probability. If the average queue size is less than a minimum threshold of the router buffer, the packet will be accepted. If the average queue size varies between a minimum threshold and a maximum threshold of the router buffer, then the packet will be dropped with a probability that increases linearly from zero to a maximum packet drop probability. However, if the average queue size exceeds the maximum threshold, then the packet is dropped. Packet dropping is meant to send a congestion signal to connections to back-off. Although RED algorithm clearly performed better than Drop-Tail by avoiding bursty traffic and global synchronization problems, it has some shortcomings namely low throughput, high packet loss rate, large delay and jitter. The RED algorithm has been recommended for implementation in routers by the IETF (Internet Engineering Task Force) [2].

Several RED-based AQM algorithms has been developed by researchers to improve its performance, such as DcRED (Delay-Controlled Random Early Detection) [4], MRED (MultiRED) [5], GRED (Gentle RED) [6], QRED (Q-Learning-based RED) [7], MRED [8], FXRED (Flexible RED) [1], WQDAQM (Weight Queue Dynamic AQM) [9], SARED (Self-Adaptive Random Early Detection) [10], QRTRED [11], DSRED (Double Slope RED) [12], MRED [13], DGRED (Dynamic GRED) [14], SDGRED (Stabilized DGRED) [15], DQRED (Dynamic Queue RED) [16], RED_E (RED-Exponential) [17], just to mention a few.

One major problem of RED algorithm is its inability to keep the average queue size low, which results in large delay. In this paper, we present a new improved RED algorithm called RED-I, which utilizes two linear packet dropping functions to distinguish between light and

Manuscript received October 21, 2021; revised March 15, 2022.
Corresponding author email: samuel.hassan@oouagoiwoye.edu.ng
doi:10.12720/jcm.17.4.260-266

heavy traffic loads instead of a single linear function used in RED.

The rest of the paper is organized as follows: In Section II, we present related works. Section III describes the proposed RED-I algorithm. Section IV presents the simulation results. Finally, Section V concludes the paper.

II. Related Works

A. Random Early Detection Algorithm

In 1993, Floyd and Jacobson in [3] proposed the famous Random Early Detection (RED) algorithm which addresses the shortcomings of the traditional Drop-Tail algorithm. For each packet arrival into the router, RED computes the average queue size (avg) which is used as a measure of congestion based on whether the router buffer is empty or not. If the queue is empty, the router first calculates the idle time parameter m, which is then used to compute the average queue size as follows:

$$m = f\left(time - q_{-idle\_time}\right) \quad (1)$$

where q_{-idle_time} is the beginning of the queue idle time

$$avg = (1 - W_q)^m \times avg' \quad (2)$$

where avg' is the calculated previous queue size; W_q is a pre-determined weight parameter to calculate avg.

However, if the packet arrives to a non-empty queue, the average queue size is calculated using a low-pass filter which is an EWMA (exponential weighted moving average):

$$avg = \left(\left(1 - W_q\right)^m \times avg'\right) + (W_q \times q) \quad (3)$$

where: q is the current queue size;

The average queue size is then compared with two pre-determined thresholds: minimum ($minTh$) and maximum ($maxTh$).

Therefore,

a) if avg is less than $minTh$ threshold, then the packet would be enqueued. That is,

$$P_b = 0 \quad (4)$$

b) if avg is higher than *maxTh* threshold, then the packet will be forced dropped. That is,

$$P_b = 1 \quad (5)$$

c) if avg ranges between *minTh* and *maxTh* thresholds, then the packet would be dropped linearly from zero to maximum dropping probability *maxP*. That is,

$$P_b = maxP\left(\frac{avg - minTh}{maxTh - minTh}\right) \quad (6)$$

Therefore, the initial dropping probability P_b function of RED is given as:

$$P_b = \begin{cases} 0 & avg < minTh \\ maxP\left(\frac{avg - minTh}{maxTh - minTh}\right) & minTh \le avg < maxTh \\ 1 & maxTh \le avg \end{cases} \quad (7)$$

Thus,

$$P_a = \frac{P_b}{(1 - count.P_b)} \quad (8)$$

where P_a is the final packet dropping probability and $count$ is the number of arrived packets since the last dropped.

B. Gentle RED Algorithm

[6] proposed GRED (Gentle RED) extended RED by adding another queue threshold, $2maxTh$ threshold in order to reduce the aggressiveness of RED algorithm. GRED mechanism accepts packets when avg is less than *minTh* threshold; if avg is between *minTh* and *maxTh* thresholds, the packet will be dropped with a probabilility that increases linearly from 0 to $maxP$; however, when avg varies between $maxTh$ and $2maxTh$ thresholds, then the packet will be dropped with a probability that increases linearly from $maxP$ to 1. GRED achieved an increased throughput.

C. MRED Algorithm

MRED was developed by [8] and operates similar to RED except that when avg varies between $minTh$ and $maxTh$ thresholds, MRED uses a stepwise function for computing the packet dropping probability instead of a linear function used in RED in order to achieve an improved throughput and delay.

D. Double Slope RED Algorithm

The DSRED (Double Slope RED) algorithm [12] which aimed at improving the throughput performance of RED algorithm divides the router's buffer into four sections by introducing a mid-point threshold between the $minTh$ and $maxTh$ thresholds and utilizes a combination of two linear packet dropping functions with different slopes.

E. Self-Adaptive Random Early Detection Algorithm

The SARED (Self-Adaptive Random Early Detection) algorithm proposed by [10] integrates a self-adaptive mechanism with RED algorithm, such that when avg varies between $minTh$ and $maxTh$ thresholds, packets are dropped with a nonlinear packet dropping function for a light and moderate traffic load conditions, however, when avg falls between $minTh$ and $maxTh$ thresholds, the algorithm switches to a linear mode for a high traffic load condition. At low and moderate traffic loads, SARED obtained an improved throughput performance.

F. QRTRED Algorithm

[11] developed the QRTRED algorithm which dynamically configures the $minTh$ and $maxTh$ thresholds of RED according to a metric called *QRT* which estimates the network condition from occupancy of the router's buffer in order to obtain an increased link utilization.

G. Delay-Controller RED Algorithm

Delay-Controller RED (DcRED) algorithm was proposed by [4]. The algorithm extended RED by

computing a delay parameter (according to the arrival rate, departure rate, and queue size) in order to determine the dropping probability. DcRED achieved an improved delay.

H. Dynamic GRED Algorithm

[14] proposed Dynamic GRED (DGRED) extended GRED by dynamically adjusting $maxTh$ and $2maxTh$ thresholds and stabilises the average queue size at a calculated target point, T_{aql} between $minTh$ and $maxTh$ thresholds. The algorithm obtained an improved packet loss rate.

I. Stabilized DGRED Algorithm

[15] developed the Stabilized DGRED (SDGRED) as an enhancement for DGRED. SDGRED eliminated the need to calculated target point and stabilizes the average queue size between $minTh$ and $2maxTh$ while dynamically adjusting the $maxTh$ and $2maxTh$ parameters based on the current average queue size. The algorithm achieved a reduced packet loss rate and queuing delay needed for real-time applications.

J. Weight Queue Dynamic AQM Algorithm

The Weight Queue Dynamic AQM (WQDAQM) was proposed by [9] as an improved version of SDGRED. The algorithm performs packet dropping by dynamically adjusting the queue weight and thresholds according to the traffic load in order to stabilize the queue weight between $minTh$ and $maxTh$ thresholds. The algorithm obtained an improved performance in terms of average queue size, delay and packet loss.

K. MultiRED Algorithm

The MultiRED (MRED) algorithm developed by [5] splits the router queue into two virtual queues: one for TCP traffic and the other for UDP traffic. Each of the queue uses the RED algorithm while using a policy named TP for marking packets with different code point according to the protocol type. MRED achieved a reduced packet loss rate of sensitive traffic flows.

L. Dynamic Queue RED Algorithm

[16] proposed Dynamic Queue RED (DQRED) which extended RED by splitting the router queue into three virtual queues thereby classifying incoming traffic into three classes according to their types, namely, UDP-based video traffic, UDP-based audio traffic, TCP-based traffic. Packets from these traffics are served in a dynamical approach. Each queue has RED algorithm for queue management. DQRED achieved a reduced delay and packet loss rate needed for real-time applications.

M. Flexible RED Algorithm

[1] developed FXRED (Flexible RED) algorithm which also integrates a self-adaptation mechanism with RED algorithm. The algorithm uses both average queue size and current traffic load condition as congestion indicators. When the a*vg* is between *minTh* threshold and a mid-point threshold, FXRED utilises a nonlinear quadratic drop function for low and moderate traffic loads in order to improve throughput and link utilization, however, when *avg* is between the mid-point threshold and *maxTh* threshold, FXRED utilizes a linear packet dropping function for high traffic load in order to improve delay.

N. Q-Learning-based RED Algorithm

An improved algorithm named QRED (Q-Learning-based RED) developed by [7] aimed at improving the throughput performance of RED algorithm by dynamically adjusting *maxP* through Q-Learning mechanism.

O. RED-Exponential Algorithm

The RED_E (RED-Exponential) algorithm proposed by [17] is an improvement over RED algorithm in the sense that when avg varies between $minTh$ and $maxTh$ thresholds, the packet dropping probability is increased exponentially from 0 to 1 thereby eliminating the need for $maxP$ in order to obtain an improved delay performance especially at heavy congestion.

P. MRED Algorithm

The MRED algorithm proposed by [13] is quite similar to GRED algorithm except that the linear packet dropping function used when avg varies between $minTh$ and $maxTh$ thresholds is replaced by a nonlinear (quadratic function). MRED was reported to achieve an improved throughput and packet loss rate performance.

III. The Proposed RED-I Algorithm

The proposed algorithm is called Random Early Detection - Improved (RED-I). RED-I subdivides the segment between $minTh$ and $maxTh$ threshold positions of RED algorithm into two segments so as to distinguish between light and heavy traffic loads. The packet dropping probability function for RED-I is depicted in Fig. 1.

For every packet that arrives the queue, RED-I computes the average queue size (avg) similar to RED using (1) - (3). Therefore,

a. If avg varies from 0 and $min1Th$, then the packet will be enqueued. That is,

$$P_b = 0 \tag{9}$$

where $min1Th$ is same as $minTh$ of RED.

b. If avg varies from $min1Th$ and $min2Th$, then the packet is dropped with probability:

$$P_b = 2maxP\left(\frac{avg - min1Th}{maxTh - 3min1Th}\right) \tag{10}$$

where

$$min2Th = \left(\frac{min1Th + maxTh}{2}\right) - min1Th \tag{11}$$

Here, RED-I increases the packet dropping probability from 0 to $maxP$ using a linear function.

c. If avg varies from $min2Th$ and $maxTh$, then the packet is dropped with probability:

$$P_b = maxP + 2(1 - maxP)\left(\frac{avg - min2Th}{maxTh + min1Th}\right) \quad (12)$$

Here, RED-I increases the packet dropping probability from $maxP$ to 1 using a linear function.

d. Lastly, if avg is greater than $maxTh$, then the packet will be dropped. That is,

$$P_b = 1 \quad (13)$$

Therefore, the initial dropping probability function of RED-I is given as:

$$P_b = \begin{cases} 0 & avg < min1Th \\ 2maxP\left(\frac{avg - min1Th}{maxTh - 3min1Th}\right) & min1Th \le avg < min2Th \\ maxP + 2(1 - maxP)\left(\frac{avg - min2Th}{maxTh + min1Th}\right) & min2Th \le avg < maxTh \\ 1 & maxTh \le avg \end{cases} \quad (14)$$

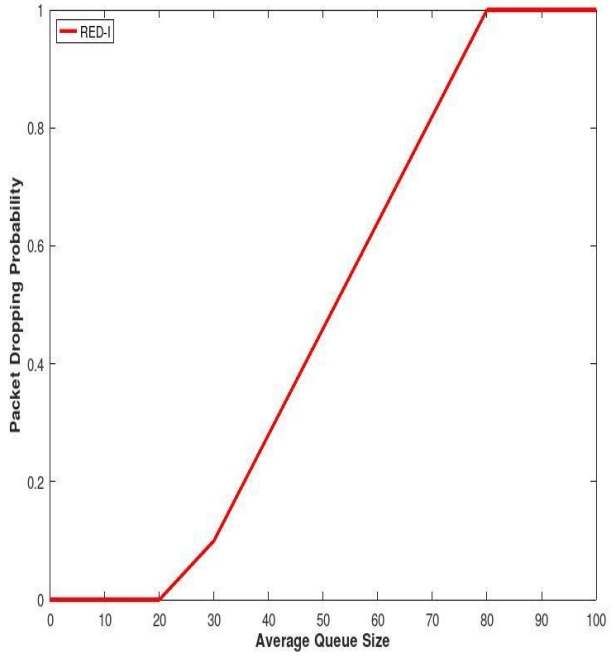

Fig. 1. RED-I's packet dropping probability

The pseudocode for RED-I algorithm is presented in Algorithm 1.

Algorithm 1 Pseudocode for RED-I Algorithm

Initialization:
$avg = 0$
$count = -1$
For each packet arrival,
Compute the average queue size avg
If the buffer of the router is non-empty **then**
$$avg = \left((1 - W_q)^m \times avg'\right) + (W_q \times q)$$
Else $m = f(time - q_{-idle\_time})$
$avg = (1 - W_q)^m \times avg'$
End if
If $avg < min1Th$ **then**
No packet drop
Set $count = -1$
Else if $min1Th \le avg < min2Th$ **then**
Set $count = count + 1$
Calculate the packet drop probability P_a
$$P_b = 2maxP\left(\frac{avg - min1Th}{maxTh - 3min1Th}\right)$$
$$P_a = P_b/(1 - count.P_b)$$
Mark/drop the arriving packet with probability P_a :
$count = 0$
Drop the packet
Else if $min2Th \le avg < maxTh$ **then**
Set $count = count + 1$
Calculate the packet drop probability Pa
$$P_b = maxP + 2(1 - maxP)\left(\frac{avg - min2Th}{maxTh + min1Th}\right)$$
$P_a = P_b/(1 - count.P_b)$
Mark/drop the arriving packet with probability P_a:
$count = 0$
Drop the packet
Else if $avg \ge maxTh$ **then**
Drop the arriving packet
Set $count = 0$
Else $count = -1$
When the buffer of the router becomes empty
Set $q_{-idle\_time} = time$
End if

IV. Simulation and Performance Evaluation

We evaluate RED-I AQM algorithm in light and heavy traffics and compare it with RED algorithm in ns-3 [18]. The network topology configuration is presented in Table I.

A. Simulation Scenario 1: Light TCP Traffic

The dumbbell topology used in this scenario consists of 5 TCP flows that started transmission all at the same time trafficking through a shared bottleneck link. This simulation scenario determines how the schemes (RED-I and RED) handles light traffic congestion.

TABLE I: NETWORK SETUP

Parameters	Value
Topology	Dumbbell
Buffer size	250 packets
Bottleneck bandwidth	10 Mbps
Bottleneck RTT	100 ms
Bottleneck queue	RED-I
Mean packet size	1000 bytes
Non-bottleneck bandwidth	100 Mbps
Non-bottleneck RTT	1 ms
Non-bottleneck queue	Drop-Tail
$minTh = min1Th$	20
$min2Th$	30
$maxTh = 4 \times min1Th$	80
$maxP$	0.1
W_q	0.002
Simulation time	100 s

Fig. 2 shows average queue size of RED-I and RED algorithms. It can be seen that the average queue size of RED-I is lower than RED. The initial peak in RED-I reaches to 52.202 while RED reaches to 53.6447. Both algorithms perform similar by bringing down the average queue size. The mean value of instantaneous average queue size for RED-I is 5.8148 while RED is 7.9862. This simply implies at light traffic load when the network is less congested, RED-I maintains the queue size better than RED. This is because, when the average queue size is higher than $min1Th$ the packet dropping probability of RED-I is higher than RED.

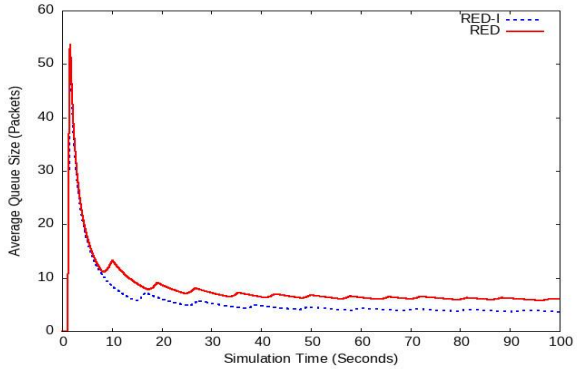

Fig. 2. Average queue size under light traffic condition

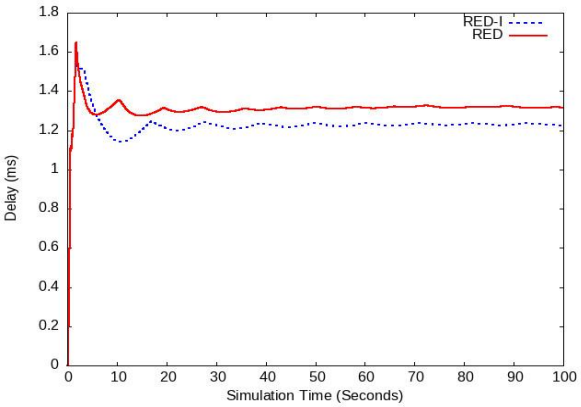

Fig. 3. Delay under light traffic condition

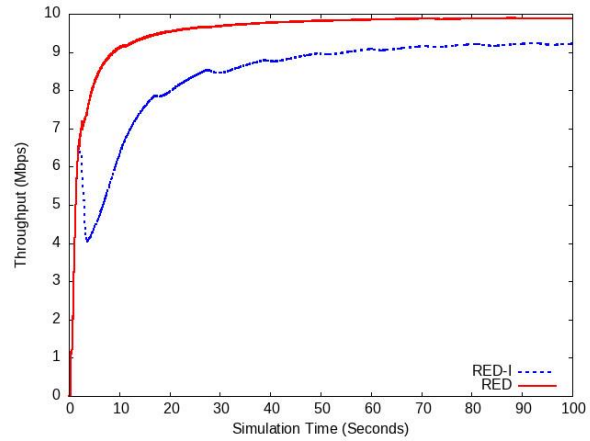

Fig. 4. Throughput under light traffic condition

Fig. 3 shows the delay of RED-I and RED algorithms. It can be seen that both algorithms performs similar in terms of delay. The initial peak in RED-I reaches to 1.6269 while RED reaches to 1.6478. The mean value of delay for RED-I is 1.2256 while RED is 1.3075. RED-I offers a lower delay at light traffic.

Fig. 4 shows the throughput of RED-I and RED algorithms. It can be seen that both algorithms performs similar in terms of throughput. It can be seen that the initial peak in RED-I reaches to 9.2428 while RED reaches to 9.8947. The mean value of throughput for RED-I is 8.3614 while RED is 9.5057. This implies that RED-I improves the delay at the expense of throughput in this scenario.

B. Simulation Scenario 2: Heavy TCP Traffic

This scenario uses a dumbbell topology which has 50 TCP flows that began transmission all at the same time passing through a shared bottleneck link. This simulation scenario determines how the schemes (RED-I and RED) handles heavy traffic load.

Fig. 5 shows average queue size of RED-I and RED algorithms. It can be seen that the average queue size of RED-I is lower than RED. The initial peak of RED-I RED algorithm reaches 48.262 while RED reaches to 87.8063. The mean value of instantaneous average queue size for RED-I is 15.2168 while RED is 36.5848. This is because, at heavy traffic load, RED-I linearly increases the packet dropping probability faster than RED.

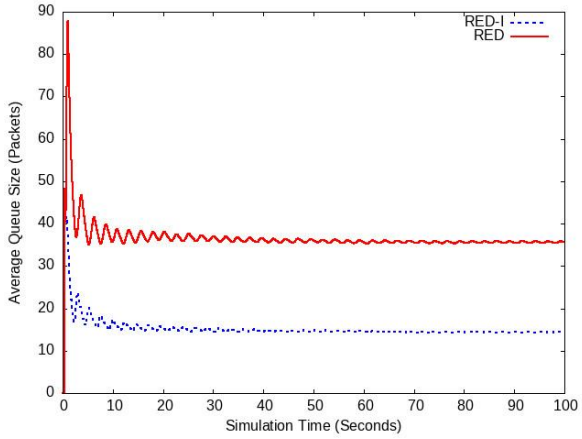

Fig. 5. Average queue size under heavy traffic condition

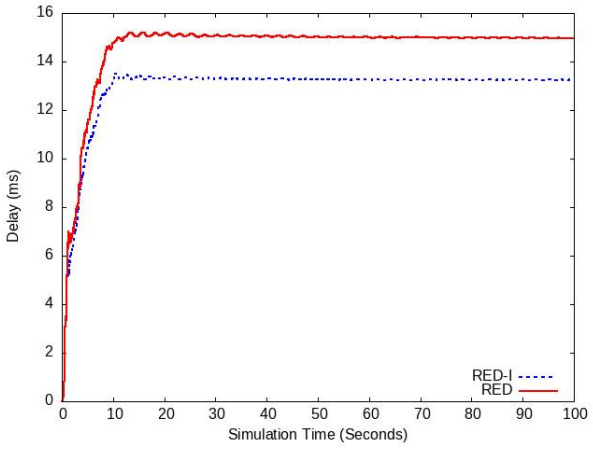

Fig. 6. Delay under heavy traffic condition

Fig. 6 shows the delay of RED-I and RED algorithms. It can be seen that both algorithms perform similar in terms of delay and that RED-I offers a better performance than RED. The initial peak in RED-I reaches to 13.5374 while RED reaches to 15.2101. The mean value of delay for RED-I is 12.8764 while RED is 14.5473.

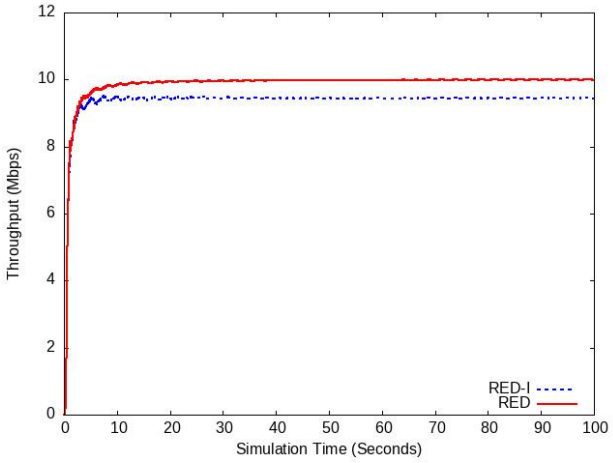

Fig. 7. Throughput under heavy traffic condition

Fig. 7 shows the throughput of RED-I and RED algorithms. It can be seen that both algorithms performs similar in terms of throughput. The initial peak in RED-I reaches to 9.5131 while RED reaches to 10.0022. The mean value of throughput for RED-I is 9.3565 while RED is 9.8561. This implies that RED-I improves the delay at the expense of throughput in this scenario.

V. Conclusion

This paper presents an improved RED-based active queue management algorithm called RED-I which utilizes a combination of two linear packet dropping functions to distinguish between light and heavy traffic loads. ns-3 simulation results showed that at both light and heavy traffic loads, RED-I clearly outperformed RED in terms of end-to-end delay as a result of achieving a lower average queue size at the expense of throughput. In future work, we intend to compare the performance of RED-I with other AQM algorithms, such as GRED, MRED, RED_E, and DSRED.

Conflict of Interest

The authors declare no conflict of interest

Author Contributions

All authors contributed equally and approved the final version.

References

[1] A. Adamu, V. Shorgin, S. Melnikov, and Y. Gaidamaka, "Flexible random early detection algorithm for queue management in routers," *LNCS*, vol. 12563, pp. 196-208, 2020.

[2] B. Braden, *et al*., "Recommendations on queue management and congestion avoidance in the internet," *RFC 2309*, 1998.

[3] S. Floyd and V. Jacobson, "Random early gateway for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397-413, 1993.

[4] A. A. Abu-Shareha, "Controlling delay at the router buffer using modified random early detection," *International Journal of Computer Networks and Communications*, vol. 11, no. 6, pp. 63-75, 2019.

[5] E. Fgee, A. Smeda, and K. AbooElgaseem, "MRED: An algorithm to insure high QoS IP networks," *Journal of Communications*, vol. 12, no. 4, pp. 200 – 206, 2017.

[6] S. Floyd. Recommendation on using the "gentle" variant of RED. [Online]. Available: http://www.icir.org/oyd/red/gentle.html

[7] Y. Su, L. Huang, and C. Feng, "QRED: A Q-Learning-based active queue management scheme," *Journal of Internet Technology*, vol. 19, no. 4, pp. 1169-1178, 2018.

[8] J. Koo, B. Song, K. Chung, H. Lee, and H. Kahng. "MRED: A new approach to random early detection," *IEEE*, pp. 347–352, 2001.

[9] M. Baklizi, "Weight queue dynamic active queue management algorithm," *Symmetry*, vol. 12, no. 12, pp. 1-16, 2020.

[10] A. Adamu, Y. Surajo, and M. T. Jafar, "SARED: Self-Adaptive active queue management scheme for improving quality of service in network systems," *Journal of Computer Science*, vol. 22, no. 2, pp. 253–267, 2021.

[11] S. Jamali, B. Alipasandi, and N. Alipasandi, "An improvement over random early detection algorithm: A self-tuning approach," *Journal of Electrical and Computer Engineering Innovations*, vol. 2, no. 2, pp. 57–61, 2014.

[12] B. Zheng and M. Atiquzzaman, "DSRED: An active queue management scheme for the next generation networks," in *Proc. 25th Annual IEEE Conference on Local Computer Networks, IEEE Computer Society*, 2000, pp. 242-251.

[13] Y. Zhang, J. Ma, Y. Wang, and C. Xu, "MRED: An improved nonlinear RED algorithm," *International Proceedings of Computer Science and Information Technology*, vol. 44, no. 2, pp. 6-11, 2012.

[14] M. Baklizi, H. Abdel-Jaber, M. Abu-Alhaj, N. Abdullah, S. Ramadass, and A. Almomani, "Dynamic stochastic early discovery: A new congestion control technique to improve networks performance," *International Journal of Innovative Computing, Information and Control*, vol. 9, no. 3, pp. 1113–1126, 2013.

[15] M. Baklizi, "Stabilizing average queue length in active queue management method," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 3, pp. 77–83, 2019.

[16] H. Mohammed, G. Attiya, and S. El-Dolil, "Active queue management for congestion control: Performance evaluation, new approach, and comparative study," *International Journal of Computing and Network Technology*, vol. 5, no. 2, pp. 37–49, 2017.

[17] H. Abdel-Jaber, "An exponential active queue management method based on random early detection," *Journal of*

Computer Networks and Communications, vol. 2020, pp. 1-11, 2020.
[18] The Network Simulator ns-3. [Online]. Available: https://www.nsnam.org

Samuel O. Hassan received a Ph.D degree in Computer Science from Obafemi Awolowo University, Ile-Ife, Nigeria. Currently, he is a Lecturer in the Department of Mathematical Sciences (Computer Science Unit), Olabisi Onabanjo University, Ago-Iwoye, Nigeria. He is a Certified Information Technology Practitioner (*C.itp*). Member of International Association of Engineers (IAENG), Computer Professionals (Registration Council) of Nigeria (CPN), Nigeria Computer Society (NCS). He is also a Member and Advisor of MathTech Thinking Foundation, India. His research interests include Computational Mathematics, Mathematical modeling and simulation, Internet congestion control, computer networks and communications, and numerical computation.

Ajaegbu Chigozirim is a Senior Lecturer in Babcock University, Ilishan-Remo, Nigeria. His research interest includes wireless networking, humanized computing and information technology. He is also an associate editor and reviewer of some high ranking journals.

Samson O. Ogunlere is an Associate Professor at Babcock University, Information Technology (IT) Department in IT and Computer related courses. He is a registered member of Nigeria Society of Engineers (MNSE), Council for Regulation of Engineering in Nigeria (COREN) and Nigeria Computer Society (NCS) with many years of working experiences in IT and Computer Industries. He is also a member of Computer Science and IT Department Research group in Babcock University, Ogun State, Nigeria.

Richmond U. Kanu is a Senior Lecturer in the Department of Basic Sciences, School of Science and Technology, Babcock University, Ogun State, Nigeria. He holds a B. Sc. (hons) degree in Mathematics from University of Uyo, Nigeria, an M. Sc. in Mathematics from University of Ibadan, Nigeria and a Ph.D. degree in Functional Analysis from University of Ilorin, Nigeria. He is also a member of Nigeria Mathematical Society.

Olusola S. Maitanmi is a Senior Lecturer in the Department of Software Engineering, Babcock University, Ilisan Remo, Ogun State, Nigeria. His area of interest is not limited to Cyber Physical System, Information Security, Quantitative & Qualitative Research, Grant writing, Research and proposal writing.