Journal of Basics and Applied Sciences Research (JOBASR)
ISSN print: 3026-9091
Volume 2(1) March 2024
DOI: https://doi.org/10.33003/jobasr-2024-v2i1-13

A Priority-Based Self-Adaptive Random Early Detection Algorithm in IoT Gateways

Abdulrahman Nasiru Sada*, Oyenike Mary Olanrewaju, Yusuf Surajo
Computer Science Department, Federal University Dutsin-ma, Katsina, Nigeria
*Corresponding Author Email: nasirabdulrahman182@gmail.com

ABSTRACT
Effective algorithms for queue management are crucial in place of guaranteeing maximum efficiency in gateway routers since network traffic continues to expand dramatically. An online researcher has suggested the Active Queue Management (AQM) strategy regarding the upcoming generation of gateway switches. The common active queue scheme remains (RED) Random Early Detection. Random early detection is susceptible to parameterization issues and lacks a self-adaptation mechanism. Several RED variants have been formed; nevertheless, variations in traffic load have an adverse effect on all of them. Due to the fact that each has a static drop pattern, to address the RED and its variation schemes, the SARED system, or the design of self-adaptive random early detection was created. But in order to prevent congestion, during the time when the queue length surpasses a present maximum threshold limit, SARED aggressively removes packets. This causes networks having a lot of traffic situations the average is expected to increase queue delay, so in those cases, SARED should be less aggressive. This paper develops a priority-based queuing congestion control method for IoT gateways to manage network congestion. Our method (priority-based algorithms) performs substantially better with regard to throughput, delay, and packet loss than the present methods of SARED. The outcomes of the conducted simulation experiments have shown that in scenarios with heavy traffic loads, priority-based self-adaptive random early detection (PSARED) has greatly decreased average queuing delay by 3%, minimized average throughput by 1%, and decreased the rate of packet loss by 10% in contrast to SARED.

Keywords:
Congestion,
AQM,
RED,
SARED,
PSARED.

INTRODUCTION

The increasing demand for fast data transfer and the quick growth of network traffic present formidable obstacles regarding gateway routers in terms of effectively are controlling their queues to guarantee dependable and seamless net functionality (Karmanje et al., 2023; Atzori et al., 2010; & Varghese et al., 2017). By regulating packet transmission and averting network congestion, queue management algorithms contribute significantly to the upkeep of ideal gateway router performance (Floyd, & Jacobson, 1993; Jain, 1990). In order to prevent congestion, gateway routers have historically employed the tail-drop queue management technique. When a tail-drop method is used to fill a queue that is full, some packets are discarded till there is sufficient room in queue for new traffic flow (Adamu et al., 2021). Nevertheless, Tail Drop is not able to fully regulate congestion because of the problems of overflow, worldwide synchronization, lockout, and bias against busty traffic (Karmeshu et al., 2017). The AQM active queue technique was introduced to address the difficulties with the tail-drop technique that were noted, enabling internet routers to appropriately regulate lengths of queues, reduce queuing latency, and avoid global synchronization (Karmeshu et al., 2017). Once the queue is filled, active queue management tries to make better use of the queue by giving the sources feedback to reduce the speed at which they are sending and the speed at which new packets are sent to the queue (Feng et al., 2014).

One popular active queue management method that is frequently used in modern routers is the RED Random Early Detection Algorithm (Floyd, and Jacobson, 1993) that prevent the gateway router's queue from filling up, RED handles congestion by probabilistically discarding every incoming packet. The greatest likelihood of a packet dropping (max_p), the lowest and highest thresholds (min_{th} and max_{th}), average queue length (avg), and weighted factor (w) are the fundamental factors employed by the RED algorithm. By using the exponential Weighted Moving Average (EWMA), the RED algorithm determines the average queue length (avg) approach toward the current waiting time. According to the AVG, packets are discarded in a probabilistic way. The packets will be returned with a probability of 0 when the intended average is less than the minimal threshold value. And once the average surpasses the maximum threshold value, the probability of dropping every incoming packet is 1. Nevertheless, if the calculated average is in the range of $\min_{th}$ to $\max_{th}$, linear deletion of packets will occur. From 0 to $\max_p$ (Floyd, and Jacobson, 1993; and Bonald et al., 2015). However, because it decreases at a fairly steady rate, the RED algorithm has several performance-related problems, including a high latency, limited throughput, and insensitivity to traffic demand. The RED algorithm's poor performance may be attributed to its linear drop task, which has a tendency also to be combative during periods of minimal traffic load and insufficiently combative during periods of high demand (Bonald, et al., 2015; Korolkova, et al., 2019).

Numerous enhanced RED variations, like Gentle RED, were created to alleviate RED's vulnerability (Floyd, 2000). Double Slope in RED (Zheng, 2006). Adaptive RED (Floyd, et al., 2001). and Nonlinear RED (Zhou, et al., 2006). Random dropping combined with adaptive queue management (Karmeshu et al., 2017). Change Trend Queue Management (Tang, & Tan, 2019). Autonomous RED (Ho, & Lin, 2008). Cautious Adaptive (Tahiliani, et al., 2011). Improved Nonlinear RED (Zhang et al., 2012). Three Section RED (Feng, et al., 2014). Quadratic RED (Kumhar, et al., 2021). Quadratic Exponential RED (Hassan, et al., 2023). etc. No matter how much better things get, variations in traffic volume will always have a detrimental effect on the effectiveness of congestion control.

A recent RED variation called Self-Adaptive Random Early Detection (SARED) shown in Adamu, et al. (2021) in an attempt toward tackle the self-adaptive issue with RED and its replacement plans. The proposed SARED contains several drop patterns (Adamu, et al., 2021). for a range of load scenarios. Unlike earlier RED-enhanced versions, SARED automatically modifies the maximum dropping probability and an appropriate drop pattern to provide the best performance while also taking the current load conditions into consideration. SARED functions effectively regardless of the load condition, according to outcomes from Adamu, et al. (2021). SARED works effectively in low, moderate, and high load scenarios; nevertheless, it will cause an increase in average queuing latency for networks that experience continuous high load circumstances.

In this work, a Priority-Based Self-Adaptive Random Early Detection (PSARED) algorithm is utilized, a distinctive technique that utilizes two queues to distinguish between higher and lower priority traffic and reduce the network traffic load of the IoT gateways. In the PSARED algorithm, each queue processes data packets using a SARED routing pattern. There are other factors, besides queue priority, that affect how long packets take to process. High-priority packets may always be processed first, leaving low-priority packets to languish in a queue unprocessed for a lengthy period, perhaps leading to starvation. To solve this problem, PSARED takes into account the time that has passed between the packets to establish their overall priority. A packet's elapsed time is the amount of time that has passed between when it was created on the IoT device and the present. Packets with high priorities are always processed before packets with low priorities, for the purpose of increasing throughput and reducing the delay of life-critical data in emergency situations.

Related Work

The Random Early Detection (RED) algorithm was recommended by Van Jacobson and Sally Floyd. (1993) is so far the most popularly known AQM algorithm for controlling network congestion.

The RED algorithm, which controls the average queue length, consists of two sub-algorithms. The average queue length must be computed in the initial segment of the algorithm in order to prevent bias against busty traffic. For every new packet that reaches at the gateway, the average queue length is considered by RED (i.e., the typical quantity of packets in the router buffer) by a little filter with Exponential Weighted Moving Average (EWMA) (equation 2), and after the queue is unfilled, *avg* is calculated by calculating the potential figure of little packets that were transmitted in that vacant period of time (equation 1) (Floyd and Jacobson, 1993).

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

$$avg = ((1 - w_q) \times avg') + (w_q \times q) \quad (2)$$

Where $q$the current queue size; avg'; is the earlier determined average queue length and w_q is a weight factor that has already been established.

The second part of the method begins marking packets at random once the average falls among the *min_{th}* and *max_{th}*, preventing global synchronization. If the intended average queue length is discovered to be inferior compared to the (min_{th}), at that time the packet is released through probability 0 (that is, acceptable into the router's buffer). If it turns out that the computed average queue size is more than the (max_{th}), then there is a probability of 1 dropping the packet. Though, if the average queues size is initiate to be a significance among the (min_{th}) and (max_{th}), then the packet is dropped linearly from 0 to max_p with possibility as shown in (equation (3)

$$p_b = max_p(\frac{avg - min_{th}}{max_{th} - min_{th}}) \quad (3)$$

Where p_b is the likelihood of the first packet dropping and max_p is the maximum drop probability. Thus,

$$p_a = max_p(\frac{p_b}{1 - count * p_b}) \quad (4)$$

Where p_a is the likelihood of a last packet falling as presented in (equation (4)) and amount is the sum of reached packets since the last released. Consequently, the dropping function $p_d(avg)$ of RED can be articulated in (equation (5)

$$p_d(avg) = \begin{cases} 0, & avg < \min_{th} \\ \dfrac{avg - \min_{th}}{\max_{th} - \min_{th}} \bullet \max_p, & \min_{th} \le avg < \max_{th} \\ 1, & \max_{th} \le avg \end{cases} \quad (5)$$

Even if evidence has demonstrated the RED algorithm performs noticeably superior to the tail drop technique, studies have disclosed that RED has several disadvantages, such as Problems of parameterization (i.e., RED parameters must be continuously adjusted to get better performance), limited throughput, large delays, and the absence of a self-adaptation strategy (Patel, 2013; Misra et al., 2000; Plasser et al., 2010; and Floyd, 2000).
According to a study by Floyd (2000). All incoming packets in RED are released when the computed average rate exceeds the maximum threshold. Since RED is thought to be overly aggressive in this situation, low throughput will result from RED's drop tendency. Floyd thus suggested Gentle Red (GRED) as a solution to this issue. A second queue threshold, 2maxth, is added to GRED after max$_{th}$. In the event that the measured average falls between max$_{th}$ and 2maxth, the likelihood of dropping packets rises linearly between maxp and 1. Consequently, it is more sympathetic than RED. While average exceeds max$_{th}$
However, Traffic-Adaptive Priority-based MAC (TAP-MAC) designed an adaptable superstructure to give the nodes the best channel access for both routine and emergency data. (Henna et al., 2017). There are two categories for the channel access period (CAP): CAP1 should only be used in cases of urgency, while CAP2 is available for both normal and emergency traffic. TAP-MAC enhances the possibility that emergency traffic will be able to access the channel since it would have lower argument space standards because low-priority traffic shares a portion of CAP. Even in situations where emergency traffic is present, nodes with lower-priority data may still be able to win the channel. Moreover, no technique has been put out to halt the transmission of the usual traffic.
Furthermore, Zheng, (2006). Suggested double-slope RED (DS-RED) and saw limited throughput as a significant RED restriction. To outperform RED, DS-RED employs two distinct drop probability distributions. The suggested DS-RED modifies the drop function's slope in response to the degree of network congestion. As the queue length rises over a certain point, DS-RED dumps packets far more forcefully than RED. However, because each of their linear drop functions is specified with distinct slopes, DS-RED functions very similarly to GRED (Floyd, 2000), Given that DS-RED relies on linear drop functions, parameterization remains a challenge; it inherited the aggressiveness of RED.
Additionally, a modified version of RED known as Nonlinear RED (NLRED) developed by (Zhou et al. 2006). This version retains all other characteristics of RED except for replacing the nonlinear quadratic drop function with the linear drop function shown in RED. According to their theory, NLRED is more aggressive under big traffic loads but kinder than RED under modest traffic loads because of its function for nonlinear packet loss. Consequently, when there is little traffic, instead of having the router operate in a single average queue size, NLRED encourages it to run in a range. In situations when there is a high volume of traffic and the normal queue size is close to the maximum threshold (maxth), NLRED permits more aggressive packet dropping to rapidly decrease. Nevertheless, when there are exceptionally high loads, NLRED may result in force loss and congestion.
An Improved Nonlinear RED, or INRED, was offered by Zhang et al. (2012). As a solution to the Nonlinear RED (NRED) problem. While the average is among the min$_{th}$ and max$_{th}$, INRED employs a nonlinear drop function, in contrast to nonlinear red (NRED). However, when the average is between the max$_{th}$ and 2max$_{th}$, like in GRED, the drop probability grows linearly to the maximum of 1. Even in situations of extreme congestion, Zhang et al. found that the INRED simulation results outperform GRED and NRED in terms of performance. In essence, INRED is viewed as a better NRED and GRED variant.
In research carried out by Akshatha and Vedananda (2018). Suggested a revised RED algorithm called Hybrid Modified Random Early Detection (HMRED) to overcome the problems of traditional RED, for example, poor throughput and excessive packet loss. HMDRED uses Modified Gaussian Function (MGF) drop probability to reduce packet drop. The simulation results have shown that HMDRED provides better throughput and packet drop compared to the RED algorithm. Nevertheless, the dropping approach that was first implemented in poor connection usage results from HMDRED's extreme aggression.
To overcome the problems of RED and enhance network performance, Abu-Shareha (2019). suggested Enhanced Random Early Detection (ERED) and Time-window Augmented RED (Windowed-RED). The results obtained revealed that the proposed algorithms improve the slow reaction time problem of the traditional RED algorithm. However, ERED and Window-RED inherit many problems of the original RED algorithm, and when implemented in IoT gateways, they will lead to long queuing delays and high packet loss rates.
The self-adaptive problem with RED-based AQM schemes is addressed by an improved RED scheme that was put out in Adamu et al. (2021). The self-adaptive approach in Adamu et al. (2021) adjusts an appropriate

drop pattern and maximum dropping chance based on the average queue length and the current traffic load in order to achieve the required presentation level. For the purpose of greatly enhancing the SARED technique's performance under high loads, this study proposes an exponential variation.

MATERIALS AND METHODS

Priority-Based Self-Adaptive Random Early Detection Algorithms

Several studies have shown that in network with continuous fluctuating traffic loads. Both the original RED and several of its enhanced versions fall short of providing the necessary performance; however, clearly, in networks the stream of traffic loads is unstable, and as such, SARED recommended and AQM algorithm that resulted in improved performance regardless of the traffic load fluctuations. The SARED algorithms uses the computed average queue length (avg) and network traffic loads congestion displays built on which packet dropping possibilities are clear, However, SARED uses only a single queue as such, it cannot differentiate between higher-priority traffic(e.g emergency or life-critical) and lower priority traffic(e.g normal data).

This study proposes a Priority-Based Self Adaptive Random Early Detection (PSARED) algorithm, a single scheme that employs two queues to differentiate between higher and lower priority traffic and reduce the network traffic load of the IoT gateways. Higher priority data packets must be transmitted as quickly as possible. In PSARED algorithm, as shown in Fig. 1, each queue processes data packets using the SARED routing strategy since it can lead to higher performance regardless of how changes in traffic load occur. Packet processing times are influenced by factors other than queue priority. If high-priority packets are always treated sooner, low-priority packets could be left in a queue and never processed by the IoT gateway, leading to starvation. To solve this problem, PSARED considers the packets' elapsed time in order to control the complete priority of the packets. A packet's elapsed time is the amount of time that has passed between when it was created in the IoT device and the present. Packets with high priorities are always processed before packets with low priority, help boost throughput and shorten the delay of vital information during emergencies.

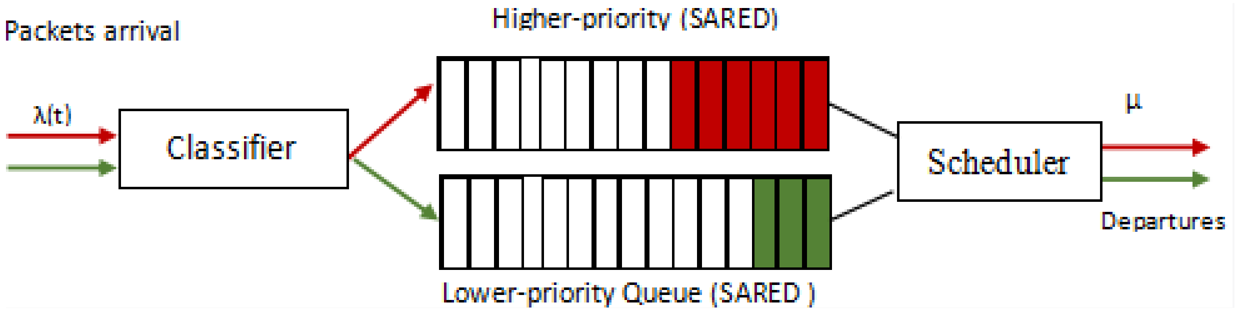

Figure 1: PSARED Queue Model

In PSARED, the lapsed time for a packet t_e is the time intermission among the creation of the packet from the sensor node t_c and the present time t as presented in equation 6.

$$t_e = t - t_c \quad (6)$$

Similarly, $P_r(t)$ shows the significance of the packet priority at time t and the overall priority of the packet $P_i(t)$ located at IoT gateway *i* is calculated using equation (7).

$$P_i(t) = (\beta_1 \times P_r(t)) + (\beta_2 \times t_e) \quad (7)$$

Where β_1 and β_2 reflect weighing considerations that have a direct impact on Making decisions Each weight component should have a value between zero and one, and the total of the weights assigned should not be greater than one. In other words, $\beta_1 + \beta_2 = 1, \beta_1, \beta_2 \in [0,1]$. In a queue, each packet has the same priority. Furthermore, Each queue uses the SARED algorithm to holder every packet. Consequently, A comparison of the front packets in each queue is required to ascertain the packets' total priority and transmission priority.

Similar to the SARED algorithm, in the proposed PSARED algorithm, two congestion indicators are used for active queue management, i.e. average queue length (*avg*) and the current network's traffic load state, based on these indicators, a drop pattern is adapted to ensure good and stable performance regardless of load state. To determine the traffic load status of the network, the flow rate from each active IoT device is actively monitored. Let $\lambda_n(t)$ be data flow rate from n^{th} source, *n* = 1, when N is the total number of IoT devices in use at a given moment (t). The following formula provides the overall data arrival rate at the IoT gateway at time t: (8).

$$\lambda(t) = \sum_{n=1}^{N} \lambda n\,(t) \quad (8)$$

If μ represents the bottleneck link's bandwidth, then the traffic load at time t may be found using equation (9).

$$\rho(t) = \frac{\lambda(t)}{\mu} \quad (9)$$

If $\rho(t) < 1$, then there is little traffic regarding the bottleneck connection, Consequently, packets won't gather in the row, If the situation is retained long period of time, then establish a link underutilization will follow. If $\rho(t) \approx 1$, thus, the volume of traffic is sensible regarding the bottleneck connection , henceforth presentation is ideal. But still, if thus, the volume of traffic is high regarding the bottleneck connection, as packets accumulation in the queue, they'll hold off on being transmitted, Long-term maintenance of this condition might probably result in congestion and overflow. PSARED familiarizes its drop level built on the detected$\rho(t)$, for high load, PSARED operates aggressively in a linear manner to prevent force drop and congestion, notwithstanding for light to moderate loads, PSARED works moderate mode, where its nonlinearity depend on the detected value of$\rho(t)$.

In PSARED maximum drop probability max_p is described as a result of the observed load ρ (i.e. high max_p for high load and low max_p for light load) and average queue length (*avg*) is calculated by the Exponential Weighted Moving Average (EWMA) (equation (2.1)). If $avg < min_{th}$, then no packet will be released and if $avg \geq max_{th}$ all the arriving packets will be dropped with probability 1. But still, if $min_{th} \leq avg < max_{th}$, then packet releasing possibility increases linearly or nonlinearly (based on the observed$\rho(t)$) from 0 to the present max_p. PSARED drop function is obtainable in equation (10).

$$p_{PSARED} = \begin{cases} 0, avg < min_{th} \\ \left(\frac{avg - min_{th}}{max_{th} - min_{th}}\right)^i max_p, min_{th} \leq avg < max_{th} \\ 1, avg \geq max_{th} \end{cases} \quad (10)$$

$$i = k^{\frac{1}{\rho(t)}}, k \geq 2 \quad (11)$$

Essentially, PSARED adjusts its max_p based on present weight (high max_p for high load, moderate max_p for moderate load, and low max_p for light load), furthermore in PSARED once *avg* drops in min_{th} and max_{th}, a function of drops whose exponent is also a function of load is defined and it increases linearly or nonlinearly from 0 to the present max_p. Then, the exponent of PSARED's drop gathering is a function of load, as such its amount of nonlinearity depend also going on current load, i.e. its degree of nonlinearity increases as load decreases (its slope decreases) and decreases as load increases. Subsequently, Based on load situation, PSARED can work in two modes, linear and nonlinear modes. It works in nonlinear (moderate) mode for light to pacify load and changes to linear. These attributes of PSARED make it self-adaptive so that a variety of loads may be handled with ease.

The pseudo code of the PSARED Queuing algorithm

Algorithm 1: PSARED QUEUING ALGORITHM

```
1. Input: $min_{th}$, $max_{th}$, $w$, $k$, $\mu$,
2. Output: count
3. Initialization:
4. $avg \leftarrow 0$
5. count ← -1
6. for all source nodes do
7. Return $\lambda(t)$ [Mbps] (equation 8)
8. $\rho(t) \leftarrow \lambda(t)/\mu$ (equation 9)
9. $i \leftarrow k^{1/\rho(t)}$ (equation 11)
10. $max_p \leftarrow 1- e^{-\rho(t)}$
11. end for
12. for each packet arrival do
13. Calculate the average queue size: avg
14. if the queue is nonempty then
15. $avg \leftarrow (1 - w) \times avg' + w \times q(t)$
16. else
17. $m \leftarrow f(t - t_{queue\_idle\_time})$
18. $avg \leftarrow ((1 - w)^m \times avg')$
19. end if
20. Determine packet discard
21. if $avg < min_{th}$ then
22. No packet drop
23. Set count ← -1
24. else if $min_{th} \leq avg < max_{th}$ then
25. Set count ← count + 1
```

```
26. Calculate the packet drop probability $P_b$
27. $P_b \leftarrow ((avg - min_{th}) / (max_{th} - min_{th}))^i \times max_p$
28. $P_a \leftarrow P_b / (1 - count \cdot P_b)$
29. Mark the arriving packet with probability $P_a$
30. Set $count \leftarrow 0$
31. Drop the packet
32. else if $max_{th} \le avg$ then
33. Drop the arriving packet
34. Set $count \leftarrow 0$
35. else $count \leftarrow -1$
36. When the router's queue becomes empty
37. Set $t_{queue\_idle\_time} \leftarrow t$
38. end if
39. end for
```

The pseudo code of the PSARED scheduling algorithm.

```
Algorithm 2: PSARED SCHEDULING ALGORITHM
    1. Input: $\beta_1, \beta_2, t_{c1}, t_{c2}$
    2. Output: $P_i(t)$
    3. Initialization:
    4. //Compute the overall priority of the front packet in the Highest Priority Queue ($Q_1$)
    5. Return $P_{r1}(t)$      //Priority value of the front packet of $Q_1$
    6. $t_{e1} = t - t_{c1}$
    7. $P_1(t) = (\beta_1 \times P_{r1}(t)) + (\beta_2 \times t_{e1})$
    8. //Compute the overall priority of the front packet in the Lowest Priority Queue ($Q_2$)
    9. Return $P_{r2}(t)$      //Priority value of the front packet of $Q_2$
    10. $t_{e2} = t - t_{c2}$
    11. $P_2(t) = (\beta_1 \times P_{r2}(t)) + (\beta_2 \times t_{e2})$
    12. If ( $P_1(t) > P_2(t)$ ) then
    13. return $P_1(t)$
    14. else
    15. return $P_2(t)$
```

Table 1: PSARED Algorithms Parameter

Saved Variable	Fixed Parameter	Other
avg present average queue length	*w*: queue weight	*t*: present time
avg': considered earlier average queue length	min_{th}: minimum threshold for the queue	*λ(t)*: current total incoming traffic load (Mbps)
$t_{queue\_idle\_time}$: start of the queue idle time	max_{th}: maximum threshold for the queue	*ρ*: present bottleneck traffic load
count: packets since last marked packets	Δ: queue's middle Threshold	*q*(*t*): current queue length
max_p: current maximum drop probability	*μ*: bottleneck connection size (Mbps)	*f*(*t*): a linear gathering of the time *t*
i: proponent of the nonlinear drop function		p_b: temporary probability used in the calculation
k: advocate for nonlinear drop functions		p_a: Present packet-marking likelihood

RESULTS AND DISCUSSION

Simulation studies were carried out to verify the efficacy of the suggested PSARED algorithm. The NS-2.35 simulator on Ubuntu 18.04 LTS was used for the simulation studies, which used the network topology shown in Figure 2.

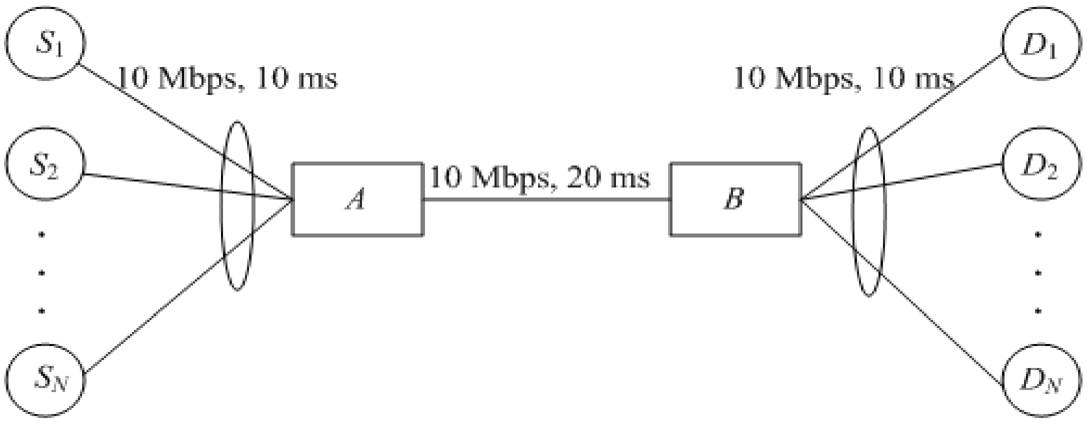

Figure 2: Topology for Simulation

The two IoT gateways make up the topology The bottleneck that connects A and B and owned by N TCP source nodes. The bottleneck link can support up to 10 Mbps and a broadcast delay of 20 ms. Through connections, the network's hosts are connected to the IoT gateways. (Each has the ability to support 10 Mbps), and their broadcast delays are 10 ms.

The simulation produced various load levels by altering the value of N, which ranged from 5 to 95 TCP source nodes. An active queue management algorithm is applied at IoT gateway A, whose capacity for a queue is 140 packets. The packet size generated by sources is 512 bytes. An updated Reno TCP setup is used. When a network has a specific quantity of TCP source nodes, a 200-s simulation is conducted, and the mean value of the network's parameter is obtained. For computation of the traffic load (ρ), the present figure amount arriving in line $\lambda(t)$ is acquired from the object that monitors queues. For the presentation study, the input factors that were employed were min_{th} = 20, max_{th} = 120, where w_q = 0.002, = 0.75, and k = 2. SARED and the suggested PSARED performance were compared.

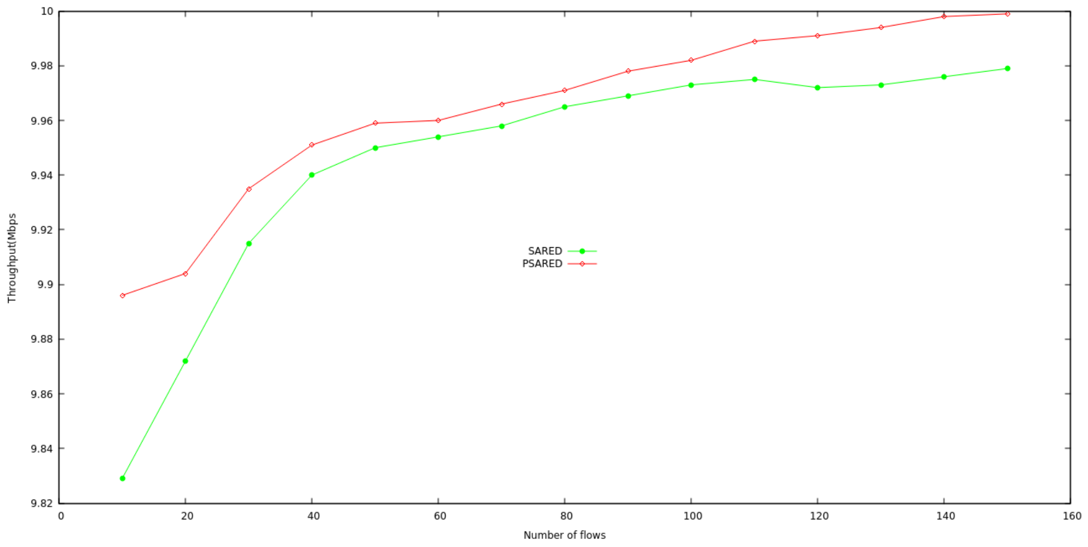

Figure 3: Throughput vs. number of node

Based on the simulation experiments conducted and the analysis of throughput as seen in Figure 3, Show that even with low loads, PSARED reduces throughput; nevertheless, in moderate and high load situations, the throughput of both PSARED and SARED reached the maximum limit. The results show that PSARED establishes a critical and non-critical discipline that enables the server to send high-priority packets at the head of the queue. In a similar vein, the discretion rule ensures that, during their non-critical intervals, high-priority packets do not obstruct the delivery of low-priority packets. Consequently, the sum of all of packets received increased massively by exploiting the critical and non-critical packet data. On the other hand, compared to the SARED, the PSARED performs better.

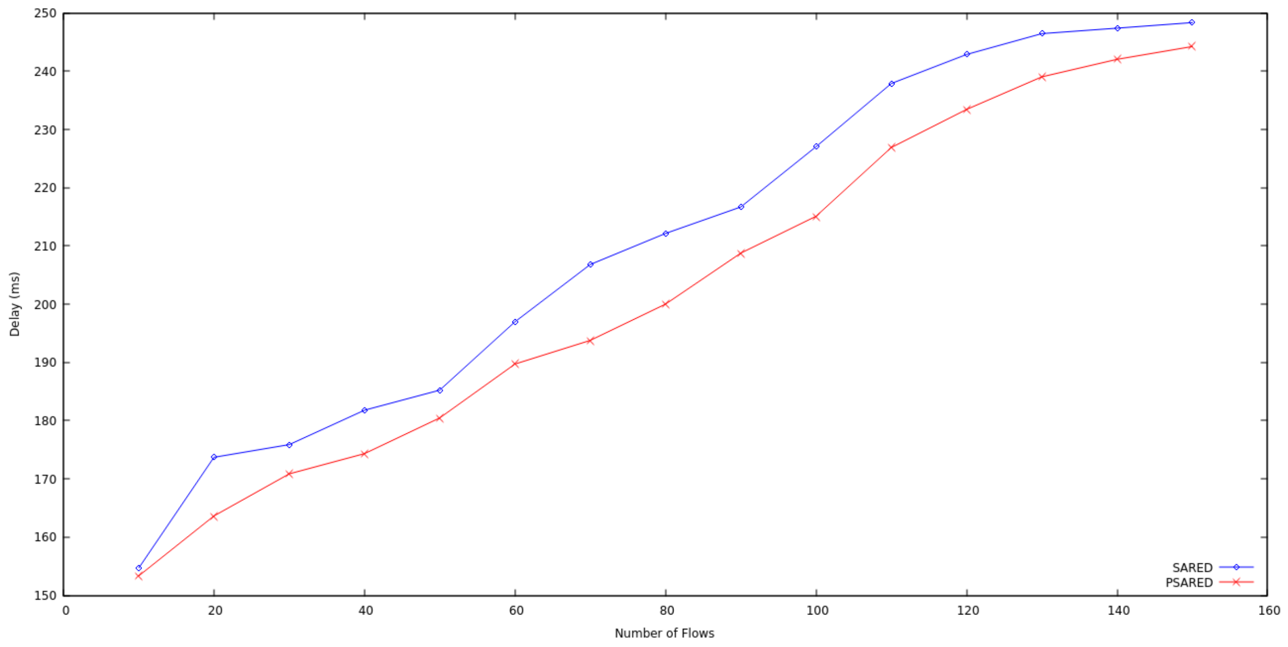

Figure 4: Delay vs. Numbers of flow

The graph in Figure 6 illustrates the delay of PSARED and SARED. The amount of time that passes between when an application in the leaf nodes generates a packet and when it reaches the IoT gateways is known as the average delay. For time-sensitive applications, it is more crucial to calculate the average delay of the high-priority packet in the PSARED. The average delays of the PSARED and SARED are shown in Figure 4. The PSARED has a lower average delay than the SARED. The explanation for this is that high-priority packets are sent directly to take the lead in the queue as they arrive at the link server in the PSARED and are instantly categorized based on that information. Additionally, high-priority packets are referred unavailable of the link server, while low-priority packets wait in the queue if the discretion rule is met.

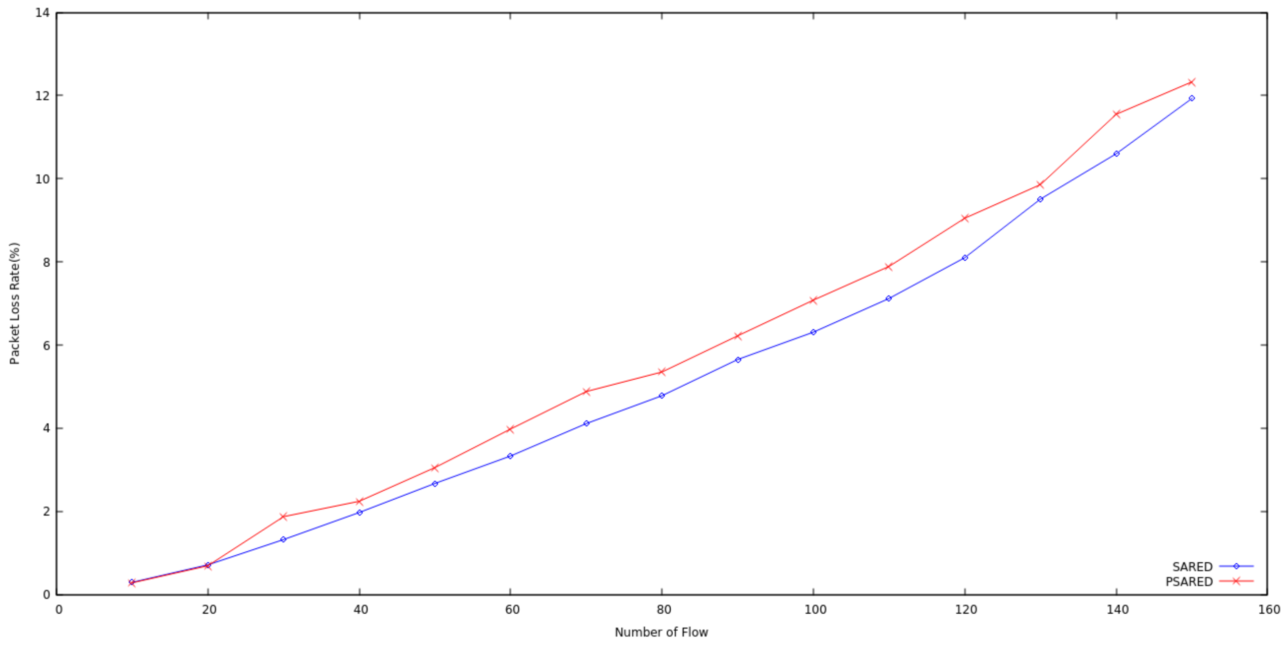

Figure 5: Packet loss rate vs. number of flows

Figure 5 displays a graph of the packet loss rate. The graph shows that when there is a high load, PSARED's packet loss rate is significantly lower than SARED's. The issue is attributed to the heavy load; the average queue length of SARED approached the max_{th}. PSARED's packet loss is primarily due to its drop policy, while SARED's loss can result from its drop policy or force drop.

CONCLUSION

A priority-base congestion control algorithms (PSARED) was develop in this paper. In contrast to RED and certain upgraded versions with drop patterns that are static. PSARED differentiates between critical and non-critical data packets in IoT gateways so as to prevent packet collision and congestion. PSARED improves the performance of SARED in relations of throughput, end to end delay and packet loss rate when both critical and non-critical data packets are present. Later, we will look into PSARED, which is suggested for further investigation and uses a machine learning model to classify critical and non-critical data packets.

REFERENCES

A. Singh. "Congestion Control techniques in Computer Networks," GeeksforGeeks.org

Adamu, A., Surajo, Y., Jafar, M. T. (2021). SARED: SelfAdaptive Active Queue Management Scheme for Improving Quality of Service in Network Systems. Computer Science 22(2), 253–267

Akshatha R. & Vedananda D. (2018). Implementation of Hybrid Modified RED Algorithm for Congestion Avoidance in MANETS. *International Journal for Research in Applied Science and Engineering Technology*, *6*(5), 2414–2419. https://doi.org/10.22214/ijraset.2018.5396
Avoidance in Computer Networks with a Connectionless Network Layer A Binary

Bonald, T., May, M., Bolot, J., Bonald, T., May, M., Analytic, J. B., & Bonald, T. (2015). *Analytic evaluation of RED performance To cite this version : Analytic Evaluation of RED Performance*.

Braden, B. (USC/ISI), Clark, D. (MIT L., Crowcroft, J. (UCL), Davie, B. (Cisco S., Deering, S. (Cisco S., Estrin, D. (USC), … Zhang, L. (UCLA). (1998). *RFC 2309: Recommendations on Queue Management and Congestion Avoidance in the Internet*. 1–18.

Chaudhary, P., & Kumar, S. (2017). A Review of Comparative Analysis of TCP Variants for Congestion Control in Network. *International Journal of Computer Applications*, *160*(8), 28–34. https://doi.org/10.5120/ijca2017913087
Control. Rochester Institute of Technology. Accessed from http://scholarworks.rit.edu

Dempf, G., & Grenzdoerfer, S. (1981). Data Networks. In *AEG-Telefunken Progress (Allgemeine Elektricitaets-Gesellschaft)*. https://doi.org/10.1049/ep.1987.0389
Feedback Scheme for Congestion Avoidance in Computer Networks with a

Feng, W. C., Shin, K. G., Kandlur, D. D., & Saha, D. (2002). The Blue active queue management algorithms. *IEEE/ACM Transactions on Networking*, *10*(4), 513–528. https://doi.org/10.1109/TNET.2002.801399

Floyd S. (2000). Recommendation on using the gentle variant of RED.

Floyd, Sally, & Jacobson, V. (1993). Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, *1*(4), 397–413. https://doi.org/10.1109/90.251892

Forouzan A. Behrouz (2007). Data Communications and Networking 4th edn., McGraw-Hill Company New York, NY 10020, USA.

Gilberto Flores Lucio, Marcos Paredes-farrera, Emmanuel Jammeh, Martin Fleury, and Hashem (1989). Analysis of random drop for gateway congestion control. *Ma, Usa.* Retrieved from http://www.dtic.mil/dtic/tr/fulltext/u2/a218737.pdf

Heinanen, J., Finland, T., Baker, F., System, C., & Weiss, W. (1999). RFC 2597 - Assured Forwarding PHB Group. Retrieved February 10, 2016, from IETF - Network Working Group website: https://www.rfc-editor.org/rfc/pdfrfc/rfc2597.txt.pdf

Henna, S.; Sajeel, M.; Bashir, F.; Asfand-e-Yar, M.; Tauqir, M. A Fair Contention Access Scheme for Low-Priority Traffic in Wireless Body Area Networks. Sensors 2017, 17, 1931. [CrossRef] [PubMed]
http://www.icir.org/oyd/red/gentle.html .
https://www.geeksforgeeks.org/congestion-control-techniquesin-computer-networks/ (assessed Dec. 5, 2020).

Hu, L., & Kshemkalyani, A. D. (2004). HRED: A simple and efficient active queue management algorithm. *Proceedings - International Conference on Computer Communications and Networks, ICCCN*, *00*(C), 387–393. https://doi.org/10.1109/icccn.2004.1401681

Huang, N. F., Jai, G. Y., Chao, H. C., Tzang, Y. J., & Chang, H. Y. (2013). Application traffic classification at the early stage by characterizing application rounds. *Information Sciences*, *232*(22), 130–142. https://doi.org/10.1016/j.ins.2012.12.039

Jain, R. (1990). Congestion Control in Computer Networks: Issues and Trends. *IEEE Network*, *4*(3), 24–30. https://doi.org/10.1109/65.56532 .

Martin J. Reed (2003). *Opnet modeler and ns-2: Comparing the accuracy of*

Misra, V., Gong, W. B., & Towsley, D. (2000). Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. *Computer Communication Review*, *30*(4), 151–160. https://doi.org/10.1145/347057.347421

Obinna Eva, N., & Kabari, L. G. (2018). Comparative Analysis of Drop Tail, Red and NLRED Congestion Control Algorithms Using NS2 Simulator. *International Journal of Scientific and Research Publications (IJSRP)*, *8*(8), 536–543. https://doi.org/10.29322/ijsrp.8.8.2018.p8069

Patel, S. (2013). Performance analysis and modeling of congestion control algorithms based on active queue management. *2013 International Conference on Signal Processing and Communication, ICSC 2013*, 449–454. https://doi.org/10.1109/ICSPCom.2013.6719832

Plasser, E., Ziegler, T., & Reichl, P. (2010). *On the Non-Linearity of the RED Drop Function*.

Ramakrishnan, K. K., & Jain, R. (1988). *A Binary Feedback Algorithm for Congestion*

Sungur Asli (2015). *TCP – Random Early Detection (RED) mechanism for Congestion*

Zhang, Y., Ma, J., Wang, Y., & Xu, C. (2012). *MRED : An Improved Nonlinear RED Algorithm*. *44*(Iccae 2011), 6–11. https://doi.org/10.7763/IPCSIT.2012.V44.2

Zheng B. (2006). DSRED: A New Queue Management Algorithm for the Next Generation Internet. *IEICE Transactions on Communications*, *E89-B*(3), 764–774. https://doi.org/10.1093/ietcom/e89-b.3.764

Zhou, K., Yeung, K. L., & Li, V. O. K. (2006). Nonlinear RED: A simple yet efficient active queue management algorithm. *Computer Networks*, *50*(18), 3784–3794. https://doi.org/10.1016/j.comnet.2006.04.007