ANLRED: A Robust AQM Mechanism for Congestion Avoidance

Manasa S.
Department of Information Science and Engineering
NMAM Institute of Technology
Nitte, Karnataka, India.

ABSTRACT

Internet over the past few years has undergone dramatic changes in terms of scale, penetration rate and the diversity of applications. The demand for continuous network connectivity is profilerating. Moreover, it has been observed that the usage of real time applications like Voice over IP (VoIP) and Live Streaming has increased drastically. Passive Queue Management (PQM) mechanisms in the routers do not react to *congestion* till the buffers overflow. This has two severe consequences: (i) large queueing delays that hurt the performance of real time traffic because such traffic is sensitive to delay and (ii) a large number of consecutive packet drops which affect the network stability. Recently, there has been a lot of interest in the deployment of Active Queue Management (AQM) mechanisms in modern Internet routers to overcome drawbacks of buffer overflow. Although Random Early Detection (RED) is the most widely studied AQM mechanism, it is highly sensitive to parameter settings. In this paper, we propose a robust AQM mechanism named Adaptive Nonlinear RED (ANLRED) which minimizes the parameter sensitivity of RED. Results obtained using *ns*-2 in a wide variety of Internet scenarios show that ANLRED improves the overall performance of the network in terms of link utilization while maintaining an acceptable mean queue length and minimal packet drop rate. Moreover, ANLRED implementation requires minimum algorithmic changes and hence, is easy to deploy.

General Terms:

Network Congestion, Queue Management

Keywords:

Active Queue Management, RED, Congestion Avoidance

1. INTRODUCTION

Internet in the present era has become a gigantic source of information and one of the most preferred means of communication. The success of Internet can be partly attributed to the congestion control mechanisms implemented in Transmission Control Protocol (TCP). TCP, although modified largely, continues to be the workshorse for Internet applications. Tremendous growth in the range of bandwidth, increase in Bit-Error Rates and increased diversity of applications however, have recently posed challenges to TCP.

TCP provides congestion control by four main algorithms: Slow Start, Additive Increase/Multiplicative Decrease (AIMD), Fast Retransmit and Fast Recovery (FR-FR) [17]. Slow start and AIMD are used for dynamically changing the size of a congestion window (*cwnd*). Slow Start increases the *cwnd* exponentially to quickly bring a newly started flow to the desired speed. In steady state, TCP uses AIMD to vary the *cwnd* in conjunction with FR-FR. FR-FR are triggered in the event of a packet loss and are used to quickly recover from the state of congestion. These algorithms, though modified several times in the recent past, have been the cornerstones of TCP research.

Congestion avoidance mechanisms differ from congestion control mechanisms, since former are proactive while latter are reactive. Though AIMD is also known as *Congestion Avoidance* algorithm, it is a misnomer since AIMD does not try to *avoid* congestion proactively [23]. Henceforth, we consider AIMD algorithms as *Congestion Control mechanisms* and AQM mechanisms as *Congestion Avoidance mechanisms* since AQM mechanisms proactively inform the sender about network state and *avoid* congestion. The deployment of AQM mechanisms in the Internet has significantly increased in the recent past, because PQM mechanisms have a few limitations such as:

(1) Lock-out [12]: PQM mechanisms (e.g.: tail-drop) allow a single connection or a few connections to monopolize the buffer space in the router queues. This results in unfair sharing of the network resources among the connections, thereby giving rise to fairness problems.

(2) Global Synchronization [8]: Traditional tail-drop gateways do not provide an early congestion notification. This leads to *global synchronization*, a phenomenon in which all senders sharing the bottleneck gateway reduce their sending rate at the same time, thereby under-utilizing the network resources.

(3) Bufferbloat [10]: Since memory costs have reduced in the recent past, modern routers are designed with extremely large buffers. As a result, today's Internet suffers from poor network performance because TCP variants implemented in modern operating systems are *end-to-end protocols* and hence, do not reduce the sending rate unless a packet drop is encountered. Since the packet drop occurs only when these large buffers overflow, queueing delay experienced by each packet increases drastically, thereby degrading the Quality of Service for delay sensitive applications such as DNS queries, VoIP and

other multimedia applications. This problem has been termed as *Bufferbloat*.

AQM mechanisms have been extensively studied to monitor and limit the growth of the queue at routers. These mechanisms *avoid* congestion by proactively informing the sender about congestion, either by dropping a packet or by marking a packet. Random Early Detection (RED) [8] is the most widely deployed AQM mechanism in the routers and Explicit Congestion Notification (ECN) [22] is the most popular marking mechanism used in conjunction with RED. A lot of research studies have demonstrated that the performance of RED largely depends on appropriate setting of the following four parameters: minimum threshold (min_{th}), maximum threshold (max_{th}), queue weight factor (w_q) for exponential weighted moving average and maximum drop probability (max_p). Optimal values for these parameters differ for different scenarios and are dependent on other factors such as number of flows passing through same bottleneck gateway, packet size, etc. Table 1, reproduced from [23] presents the values of above mentioned parameters used in Cisco 12000 Series routers that implement a modified RED called Weighted RED (WRED). C is the capacity of the link in packets where mean packet size is 1500 bytes.

Table 1. WRED parameter setting in Cisco 12000 Series Routers

Link Speed	C	min_{th}	max_{th}	w_q	max_p
DS3	3666	110	367	9	1
OC3	12917	388	1292	10	1
OC12	51666	1550	5167	12	1

This paper proposes a robust AQM mechanism which aims to minimize the parameter sensitivity of RED by making minimal algorithmic modifications. The proposed algorithm is named as Adaptive Nonlinear RED (ANLRED) Algorithm. Unlike other RED based AQM mechanisms, ANLRED does not introduce new parameters to achieve better performance since adding new parameters further complicates the original design of RED.

The remainder of the paper is organized as follows: Section 2 describes the related work that aims to improve the performance of RED. Section 3 presents the design of ANLRED algorithm. Results and analysis of the same is demonstrated in Section 4 and Section 5 concludes the paper with possible future directions.

2. RELATED WORK

The parameter sensitivity of RED has been addressed by several researchers and as a result, RED has been extended and enhanced by adopting many different approaches. The basic mechanism of RED, however, still remains the same.

On arrival of every packet, RED gateways calculate the average queue size (*avg*) using Exponential Weighted Moving Average (EWMA). If *avg* is less than min_{th}, the packet is enqueued. If *avg* is more than max_{th}, the packet is dropped[1]. However, if *avg* is between min_{th} and max_{th}, the packet is dropped randomly with a certain probability. RED, therefore, has two computational parts: computation of *avg* and calculation of packet drop probability (p_d). The following equations show *avg* and p_d calculation of RED respectively:

[1] In presence of ECN, router may choose to mark the packet instead of dropping. Hence, the terms dropping and marking are used interchangeably.

$$avg = ((1 - w_q) \times oldavg) + (w_q \times cur\_q) \quad (1)$$

where *oldavg* = average queue size during previous packet arrival
cur_q = current queue size

$$p_d = \begin{cases} 0 & avg < min_{th} \\ \frac{avg - min_{th}}{max_{th} - min_{th}} \times max_p & min_{th} \leq avg < max_{th} \\ 1 & avg \geq max_{th} \end{cases} \quad (2)$$

The probability with which a packet is dropped is a linear function of *avg*. Hence when *avg* varies from min_{th} to max_{th}, the drop probability varies from 0 to max_p. If *avg* increases above max_{th}, drop probability becomes 1 i.e. all incoming packets are dropped. Figure 1 shows the marking function of RED.

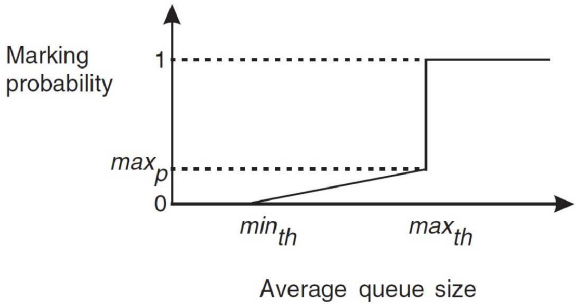

Fig. 1. Marking function of RED

The effectiveness of RED highly depends on the appropriate setting of its parameters. However, it is difficult to find the appropriate values of parameters that enable RED to perform equally well in different scenarios. RED may in fact perform worse than PQM if its parameters are not correctly tuned. We now discuss the issues and prior work done in appropriately setting these parameters.

2.1 Setting max_p

The choice of max_p significantly affects the performance of RED. If max_p is too small, the number of active packet drops becomes less and hence, cannot prevent the queue overflow. If max_p is too large, the number of active packet drops becomes more and significantly affects the throughput.

Feng [7] demonstrates that the choice of max_p depends not only on the bandwidth delay product but also on the number of flows. An algorithm called Self Configuring RED is developed and implemented in [7] to vary max_p parameter based on the average queue length dynamics. The main idea is to modulate the packet dropping behavior of RED by monitoring the variations in the *avg*. If the *avg* oscillates around min_{th}, the value of max_p is decreased to make RED less aggressive. Similarly, if the *avg* oscillates around max_{th}, the value of max_p is increased to make RED more aggressive. max_p is carefully varied so as to keep the *avg* between min_{th} and max_{th}. This algorithm performs well in different traffic scenarios since it reduces the oscillations in the instantaneous queue length.

As an extension to Self Configuring RED, an Adaptive RED (ARED) is developed in [9] and its effectiveness over the origi-

nal RED[2] is demonstrated in terms of improved throughput and reduced oscillations in the queue size. Unlike Self Configuring RED, ARED is designed to keep the *avg* in *target range* between min_{th} and max_{th} and thus, max_p is varied accordingly. Moreover, ARED automatically sets min_{th}, max_{th} and w_q parameters. The choice of *target queuing delay*, which determines the trade-off between delay and link utilization, is left to the network operators.

ARED follows an AIMD policy to vary max_p. While ARED adopts a conservative approach to vary max_p, Refined Adaptive RED (Re-ARED) [14] adopts an aggressive approach to bring *avg* within its *target range* more *quickly*. However, like ARED, Re-ARED too employs an AIMD policy to adapt max_p. On the contrary, a modified ARED algorithm based on Multiplicative Increase Multiplicative Decrease (MIMD) policy to adapt max_p is designed in [19]. However, the results show that MIMD policy to adapt max_p yields similar results as the AIMD policy.

2.2 Setting w_q for EWMA

The *avg* in RED is required to filter out the transient congestion while, at the same time, detect congestion that has persisted for several RTTs. If w_q is too small, the AQM may fail to detect the incipient congestion and lead to overall performance degradation by causing queues to overflow. If w_q is too large, *avg* tracks the instantaneous queue and leads to more oscillations in the queue, thereby degrading the performance of AQM.

ARED, as discussed above, automatically sets the w_q as a function of the link bandwidth. It is shown in as

$$w_q = 1 - exp(\frac{-1}{C}) \tag{3}$$

where C is the link capacity in packets/second, computed for packets of the specified default size.

An ARED based algorithm[3] that adaptively varies w_q along with max_p is designed in [24]. The main goal of the algorithm is to modulate the aggressiveness of RED by varying the value of w_q based on the changes in *avg*. If the change in *avg* is negligible, smaller value for w_q (w_{q1}) is chosen to give more weight to the *oldavg*. On the other hand, if the change in *avg* is significant, larger value for w_q (w_{q2}) is chosen to give more weight to the instantaneous queue length. However, w_{q1} and w_{q2} are fixed values and must be predefined.

Similar mechanisms, Stabilized ARED (SARED) [13] and Self Tuning RED [5] focus on assigning different queue weights, w_q, to ARED instead of one fixed queue weight. The major limitation of these approaches is that they introduce several new parameters to achieve performance gain. Setting these additional parameters adds to the complexity.

2.3 Setting min_{th} and max_{th}

It is recommended that the min_{th} for a RED router that carries only TCP traffic should be around five packets. max_{th} should be at least three times min_{th}. However, a different set of values are required for min_{th} and max_{th} to achieve fairness when non-TCP traffic (e.g., UDP traffic) coexists with the TCP traffic. This approach is adopted by Class-Based Threshold RED (CBT-RED) [20]. CBT-RED sets the min_{th} and max_{th} thresholds according to the traffic type and its priority. UDP traffic is assigned a separate drop threshold than the one assigned for TCP traffic.

Balanced RED (BRED) [3] achieves fairness among TCP and UDP traffic by regulating the bandwidth of a flow based on the other active flows. However, it requires per-flow accounting and hence, has scalability and deployment issues.

ARED automatically sets min_{th} based on the Eq.(4). max_{th} is set to three times of min_{th}.

$$min_{th} = max(5, \frac{d_t * C}{2}) \tag{4}$$

where d_t represents the *target queuing delay* set by the network operators.

Eq.(4) ensures that for high bandwidth links, min_{th} is set sufficiently high and for low bandwidth links, min_{th} is set equally low.

2.4 Calculating packet drop probability (p_d)

It is observed that sharply increasing the p_d to 1 when *avg* crosses max_{th} (see Fig.1) results in high number of packet drops. Hence, a modified RED known as Gentle RED (GRED) is recommended that varies the p_d from max_p to 1 when *avg* varies from max_{th} to twice max_{th} so as to reduce the number of packet drops. Figure 2 shows the marking function of GRED.

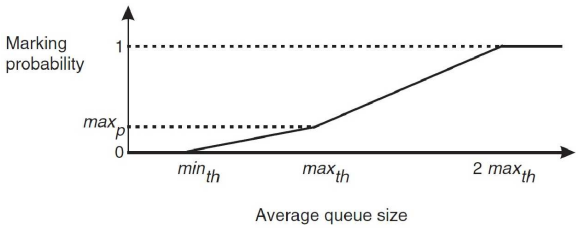

Fig. 2. Marking function of Gentle RED

Stabilized RED (SRED) [15] has been designed to make the router queue stable over a wide range of load levels. Instead of calculating the *avg*, SRED drops packets depending on the instantaneous queue length and the number of active flows. Eq.(5) and Eq.(6) show the p_d and the final packet drop function (p_{sred}) of SRED respectively. B represents the buffer capacity. SRED achieves the goal of stabilizing the queue, however, suffers from low throughput even for a small number of active flows.

$$p_d = \begin{cases} max_p & \frac{B}{3} \leq cur\_q < B \\ \frac{1}{4}max_p & \frac{B}{6} \leq cur\_q < \frac{B}{3} \\ 0 & 0 \leq cur\_q < \frac{B}{6} \end{cases} \tag{5}$$

$$p_{sred} = \begin{cases} p_d & \textit{for large number of active flows} \\ \frac{p_d}{65536}(numberof flows)^2 & \textit{for small number of active flows} \end{cases} \tag{6}$$

Double Slope RED (DSRED) [26] implements two linear drop functions with different slopes to improve the throughput and delay of RED. The main idea is to divide the queue between min_{th} and max_{th} into two segments and use a separate linear function for each segment. DSRED adapts to the level of congestion by

[2]Original RED refers to the RED proposed in [8]

[3]We call this as w_q+max_p algorithm.

changing the slope of the drop function. The equation governing the packet drop probability (p_d) is given by Eq.(7).

$$p_d = \begin{cases} 0 & avg < min_{th} \\ \alpha(avg - min_{th}) & min_{th} \leq avg < mid_{th} \\ 1 - \gamma + \beta(avg - mid_{th}) & mid_{th} \leq avg < max_{th} \\ 1 & avg \geq max_{th} \end{cases} \tag{7}$$

where α and β are given by Eq.(8) and Eq.(9) respectively. mid_{th} is a threshold for *avg* to change the slope of the drop function and γ is used as a mode selector to adjust the slopes of the drop function.

$$\alpha = \frac{2(1-\gamma)}{max_{th} - min_{th}} \tag{8}$$

$$\beta = \frac{2\gamma}{max_{th} - min_{th}} \tag{9}$$

Nonlinear RED (NLRED) [27], on the other hand, replaces the linear packet dropping function of RED by a nonlinear quadratic function to improve the effectiveness of RED. (See Eq.(10))

Though DSRED and NLRED outperform the original RED, their parameter sensitivity remains same as the original RED because they do not vary max_p and use the default value of w_q.

$$p_d = \begin{cases} 0 & avg < min_{th} \\ (\frac{avg - min_{th}}{max_{th} - min_{th}})^2 \times max_p & min_{th} \leq avg < max_{th} \\ 1 & avg \geq max_{th} \end{cases} \tag{10}$$

2.5 Other RED Variants

There are a few RED based AQM mechanisms that not only take *avg*, but also consider the *instantaneous queue size*. Examples of such mechanisms include Modified RED [6] and Effective RED [2]. Appropriately setting thresholds for *instantaneous queue size* is a challenging issue in these mechanisms. Moreover, since these mechanisms are completely based on the original RED, their parameter sensitivity remains same as that of the original RED.

Other AQM mechanisms based on RED include: Dynamic RED (DRED) [4], RED with Preferential Dropping (RED-PD) [18], Exponential RED [16], Loss-ratio based RED (LRED) [25], RED based on Neural Networks (NN-RED) [11], etc. There are some concerns on the suitability of these AQMs since they do not eliminate the parameter sensitivity of RED. Moreover, they are complex to deploy than the original RED algorithm. We summarize this section (See Table 2) by classifying RED variants based on the parameters they focus to improve the performance of RED.

Table 2. Classification of RED Variants

max_p	w_q	min_{th} and max_{th}	p_d
Self Configuring RED	w_q+max_p algorithm	CBT-RED	GRED
ARED	ARED	BRED	SRED
Re-ARED	SARED	ARED	DSRED
ARED with MIMD	Self Tuning RED		NLRED

3. ADAPTIVE NONLINEAR RED ALGORITHM

ANLRED varies max_p adaptively based on the change in avg. Although a similar approach is adopted by Re-ARED, the packet drop rate with Re-ARED is high since it varies max_p aggresively [23]. Moreover, Re-ARED does not achieve optimal performance for a few scenarios where traffic load is high [23]. ANLRED is designed to provide robust performance in a wide variety of scenarios. ANLRED algorithm consists of two parts:

(1) Setting max_p - to improve link utilization and
(2) Calculating p_d - to control packet drop rate.

ANLRED combines the advantages of two algorithms in one: first part is borrowed from Re-ARED and second part is borrowed from NLRED.

3.1 Setting max_p in ANLRED

The following algorithm presents the approach adopted by ANLRED to vary max_p. Note that like ARED and Re-ARED, ANLRED also varies max_p within a range of 1% to 50%.

Algorithm 1: Adapting max_p in ANLRED Algorithm

$every\ interval\ seconds$:
if $avg < target$ **and** $max_p \geq 0.01$ **then**
  $decrease\ max_p$
  $\beta = 1 - (0.17 \times \frac{target - avg}{target - min_{th}})$
  $max_p = max_p \times \beta$
end
else if $avg > target$ **and** $max_p \leq 0.5$ **then**
  $increase\ max_p$
  $\alpha = 0.25 \times max_p \times \frac{avg - target}{target}$
  $max_p = max_p + \alpha$
end

Variables used in the algorithm:
avg: average queue size
β *: decrease parameter*
α *: increase parameter*

Fixed Parameters in the algorithm:
interval = 0.5 seconds
target = target for average queue size:
[min_{th} + 0.48 × (max_{th} - min_{th}), min_{th} + 0.52 × (max_{th} - min_{th})]

3.2 Calculating p_d in ANLRED

Eq.(11) shows the NLRED based approach adopted by ANLRED to calculate p_d, where max_p is obtained from 3.1:

$$p_d = \begin{cases} 0 & avg < min_{th} \\ (\frac{avg - min_{th}}{max_{th} - min_{th}})^2 \times max_p & min_{th} \leq avg < max_{th} \\ 1 & avg \geq max_{th} \end{cases} \tag{11}$$

ANLRED's approach to combine the advantages of two algorithms in one not only keeps the deployment complexity low, but also improves the link utilization significantly while maintaing acceptable mean queue length and minimal packet drop rate. The next section

highlights the performance gain of ANLRED as compared to other popular RED based AQMs.

4. SIMULATION RESULTS AND ANALYSIS

In this section, we compare the performance of ANLRED with RED, ARED and NLRED in a wide variety of Internet scenarios. We have used *ns*-2 [1] and TCP Evaluation suite for simulating Internet like scenarios. TCP Reno with SACK and CUBIC TCP are selected for HTTP and FTP traffic since the former is a default transport protocol used in Microsoft Windows family of operating systems and the latter is a default transport protocol in Linux kernel > 2.6. Background traffic is generated by using UDP flows.

A single bottleneck dumbbell topology with two-way traffic is designed for all the simulations. The bottleneck bandwidth is set to 10Mbps with bottleneck RTT set to 32ms unless specified. Non-bottleneck bandwidth is set to 20Mbps with RTT is set to 4ms. The scenario consists of 5 forward-FTP flows unless specified, 5 reverse-FTP flows, 15 HTTP flows generated using PackMime generator, 5 audio flows, 5 forward-streaming flows and 5 reverse-streaming flows. The parameters chosen above are based on the simulation scenario designed in [21]

The desirable properties of an optimal AQM mechanism are: high link utilization, minimum queue occupancy and least packet drop rate. Thus, we mainly concentrate on analyzing three parameters viz. link utilization of bottleneck link, queue size at bottleneck router and packet drop rate.

4.1 Varying Bottleneck Bandwidth

This scenario verifies robustness of ANLRED in a wide range of bottleneck bandwidth starting from 1 Mbps to 1 Gbps. Fig. 3 highlights the advantages of ANLRED in terms of efficient link utilization. Link utilization with ANLRED is significantly better than RED, ARED and NLRED. While other AQMs fail to achieve even 30% link utilization, ANLRED succeeds in achieving more than 60% link utilization, even in high bandwidth scenarios. Fig. 4 highlights that mean queue length with ANLRED is slightly higher than other AQMs. However, note that it is in the acceptable range and is stable around 19%. It can be verified from Fig. 5 that unlike other RED based AQMs, ANLRED does not improve the link utilization at the cost of increased packet drop rate. Packet drop rate with ANLRED remains almost similar to those of other algorithms. Fig. 6 through Fig. 8 represent similar performance measurements obtained with CUBIC TCP.

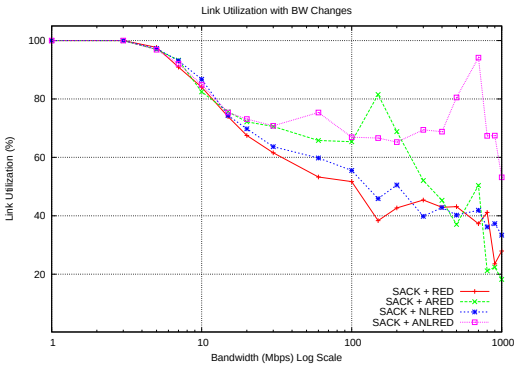

Fig. 3. Link Utilization vs Varying Bottleneck Bandwidth (TCP SACK)

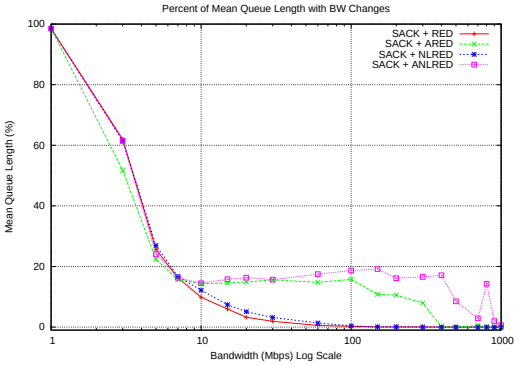

Fig. 4. Mean Queue Length vs Varying Bottleneck Bandwidth (TCP SACK)

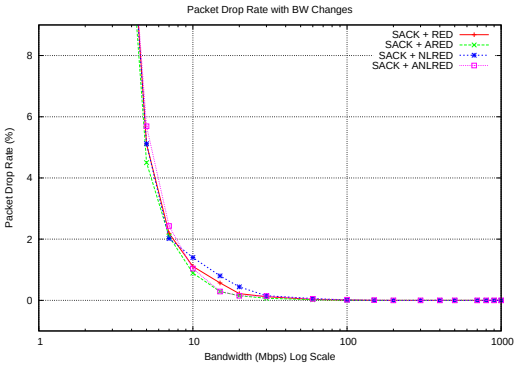

Fig. 5. Packet Drop Rate vs Varying Bottleneck Bandwidth (TCP SACK)

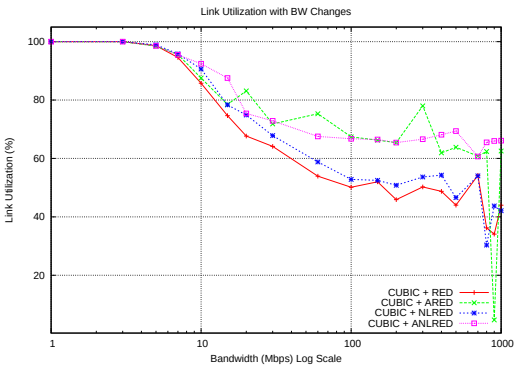

Fig. 6. Link Utilization vs Varying Bottleneck Bandwidth (CUBIC TCP)

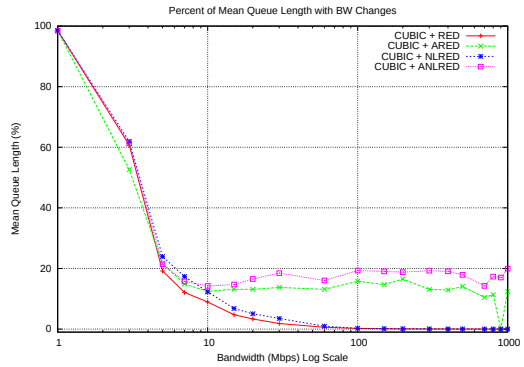

Fig. 7. Mean Queue Length Varying Bottleneck Bandwidth (CUBIC TCP)

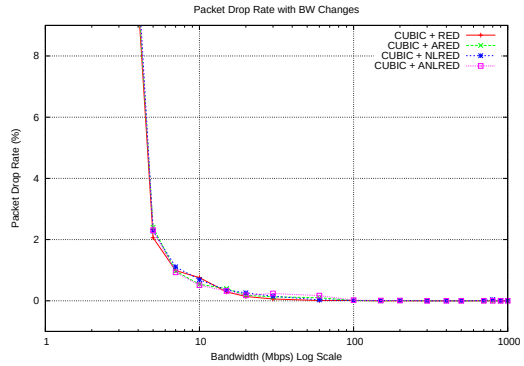

Fig. 8. Packet Drop Rate Varying Bottleneck Bandwidth (CUBIC TCP)

4.2 Varying Number of FTP flows

This scenario varies the number of FTP flows from 1 to 1000 and highlights the robustness of ANLRED in varying traffic loads.

When number of forward FTP-flows are less, the bottleneck link utilization is expected to be low and when number of forward FTP-flows are more, the bottleneck link utilization is expected to be high because of increase in the traffic load. It can be observed from Fig. 9 that even when number of FTP flows are less - ANLRED utilizes the available bandwidth much more efficiently than other algorithms. Fig. 10 and Fig. 11 show that the performance of ANLRED is similar to other algorithms in terms of mean queue length and packet drop rate. This implies improvement in link utilization with ANLRED does not affect the other network parameters.

Fig. 12 through Fig. 14 show that network performance improves slightly when CUBIC TCP is used with other AQMs, however, ANLRED's performance is stable and better than the other algorithms.

4.3 Varying RTT

In this scenario, RTT is varied from 1 ms to 1 second to capture the behavior of AQMs with varying amount of bursts. Fig. 15 through Fig. 17 show that except RED - all other AQMs including ANLRED exhibit similar performance. Although RED has lesser mean queue length and packet drop rate than other AQMs, it does not exhibit similar behavior when bandwidth is varied or number of flows are varied. Moreover, it must be noted that average RTT in

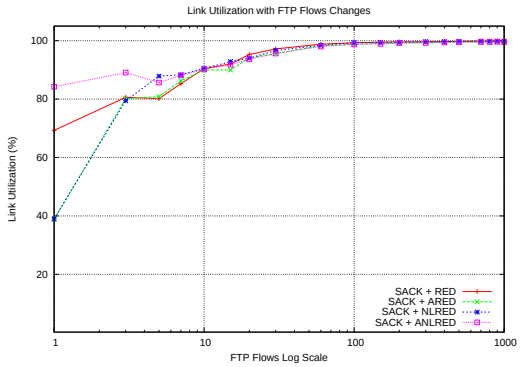

Fig. 9. Link Utilization vs Number of FTP flows (TCP SACK)

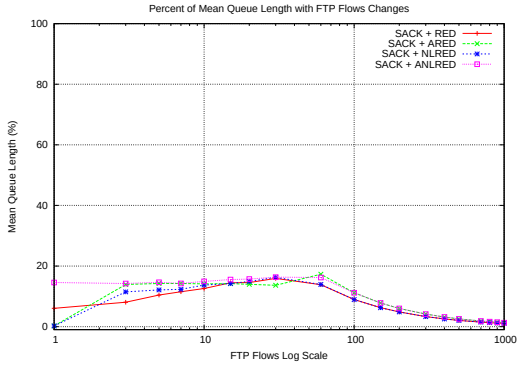

Fig. 10. Mean Queue Length vs Number of FTP flows (TCP SACK)

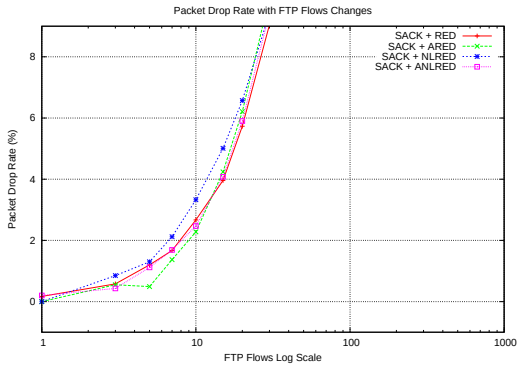

Fig. 11. Packet Drop Rate vs Number of FTP flows (TCP SACK)

the Internet is around 100 ms to 300 ms [21]. In that case, it can be observed that all AQMs have similar performance.

Same experiments were repeated with CUBIC TCP and observations were similar. Fig. 18 through Fig. 20 demonstrate the results.

5. CONCLUSIONS AND FUTURE WORK

In this paper, a robust AQM mechanism named ANLRED is proposed to enhance the network performance in terms of link utilization and packet drop rate. Based on the simulation results, it is ob-

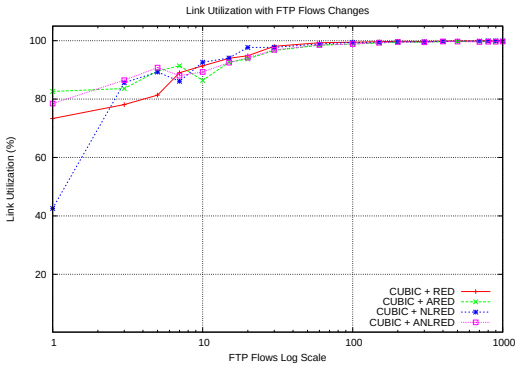

Fig. 12. Link Utilization vs Number of FTP flows (CUBIC TCP)

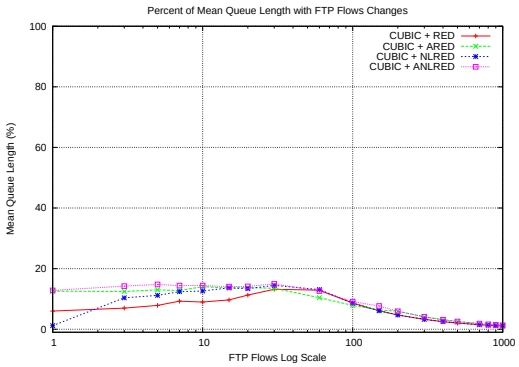

Fig. 13. Mean Queue Length vs Number of FTP flows (CUBIC TCP)

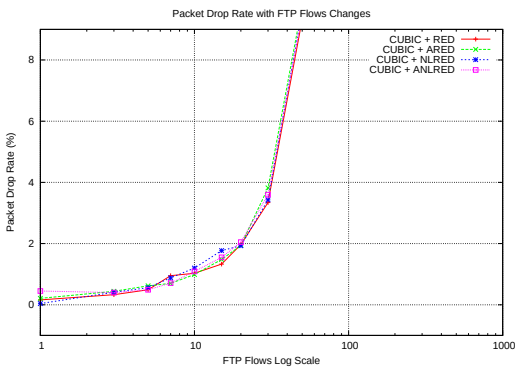

Fig. 14. Packet Drop Rate vs Number of FTP flows (CUBIC TCP)

served that ANLRED exhibits robust performance in a wide variety of Internet scenarios when compared to RED, ARED and NLRED. ANLRED, however, occupies slightly more queue space than other AQM algorithms. The future work includes optimizing ANLRED to occupy the minimum possible queue space without affecting the link utilization and packet drop rate.

6. REFERENCES

[1] UCN/LBL/VINT, Network Simulator - 2 (ns-2), 2011. Available from http://www.isi.edu/nsnam/ns/.

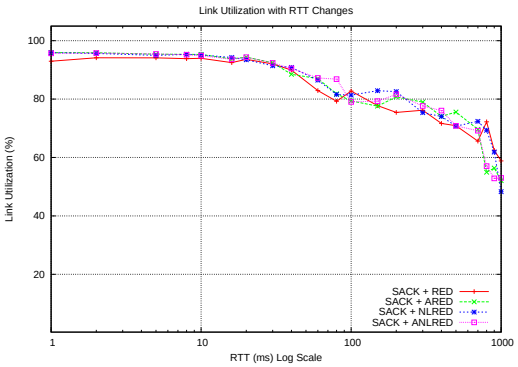

Fig. 15. Link Utilization vs Varying RTT (TCP SACK)

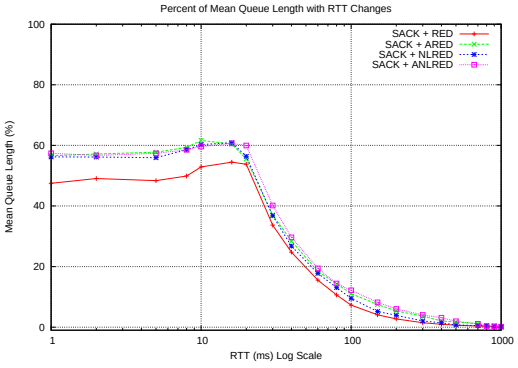

Fig. 16. Mean Queue Length vs Varying RTT (TCP SACK)

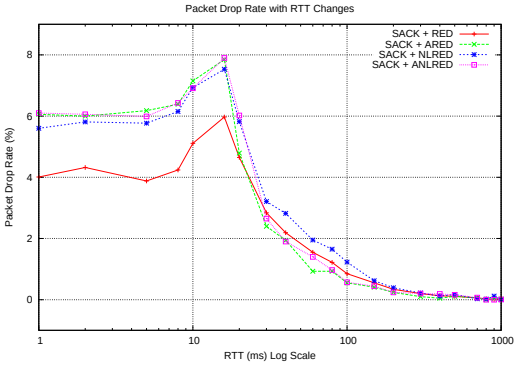

Fig. 17. Packet Drop Rate vs Varying RTT (TCP SACK)

[2] Babek Abbasov and Serdar Korukoglu. Effective RED: An Algorithm to Improve RED's Performance by Reducing Packet Loss Rate. *Journal of Network and Computer Applications*, 32(3):703–709, 2009.

[3] Farooq M. Anjum, Ros Tassiulas, Farooq M. Anjum, and Ros Tassiulas. Balanced-RED: An Algorithm to achieve fairness in the Internet. In *Proc. IEEE INFOCOM '99*, 1999.

[4] James Aweya, Michel Ouellette, and Delfin Y. Montuno. A control theoretic approach to Active Queue Management. *Computer Networks*, 36:203–235, July 2001.

[5] Jianyong Chen, Cunying Hu, and Zhen Ji. Self-Tuning Ran-

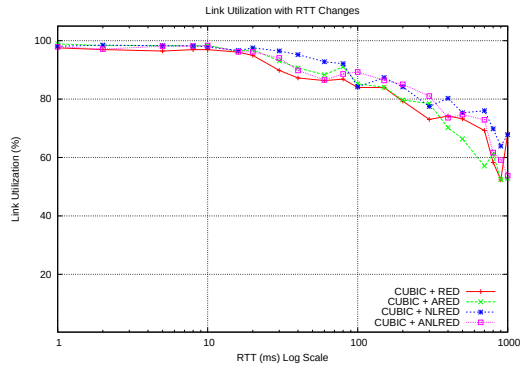

Fig. 18. Link Utilization vs Varying RTT (CUBIC TCP)

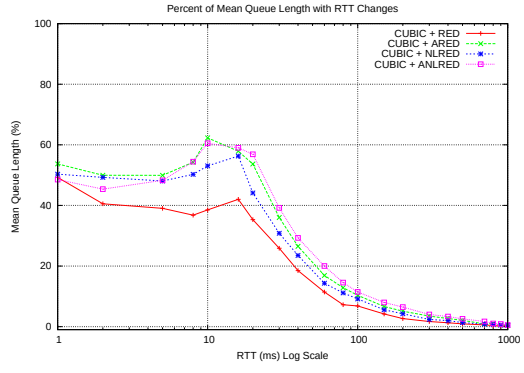

Fig. 19. Mean Queue Length vs Varying RTT (CUBIC TCP)

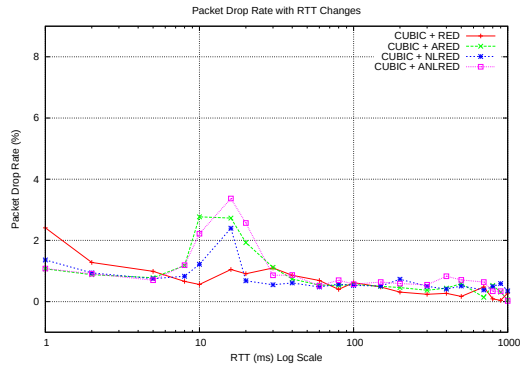

Fig. 20. Packet Drop Rate vs Varying RTT (CUBIC TCP)

dom Early Detection Algorithm to Improve Performance of Network Transmission. *Mathematical Problems in Engineering*, 2011, 2011.

[6] Gang Feng, A.K. Agarwal, A. Jayaraman, and Chee Kheong Siew. Modified RED Gateways under Bursty Traffic. *IEEE Communications Letters*, 8(5):323–325, May 2004.

[7] W.-C. Feng, D.D. Kandlur, D. Saha, and K.G. Shin. A Self-Configuring RED Gateway. In *Proceedings of IEEE INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1320–1328 vol.3, Mar 1999.

[8] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1:397–413, August 1993.

[9] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker. Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management. Technical report, August 2001.

[10] J. Gettys. Bufferbloat: Dark Buffers in the Internet. *IEEE Internet Computing Magazine*, 15:96, June 2011.

[11] B. Hariri and N. Sadati. NN-RED: An AQM Mechanism based on Neural Networks. *Electronics Letters*, 43(19):1053–1055, 13 2007.

[12] M. Hassan and R. Jain. High Performance TCP/IP Networking: Concepts, Issues and Solutions, 2004. Pearson, Inc.

[13] H. Javam and M. Analoui. SARED: Stabilized ARED. In *International Conference on Communication Technology, 2006, ICCT '06*, pages 1–4, Nov. 2006.

[14] Tae-Hoon Kim and Kee-Hyun Lee. Refined Adaptive RED in TCP/IP Networks. In *Proceedings of International Joint Conference, SICE-ICASE, 2006*, pages 3722–3725, Oct. 2006.

[15] Teunis Ott Lakshman, T. V. Lakshman, and Larry Wong. SRED: Stabilized RED. In *Proceedings of INFOCOM*, pages 1346–1355, 1999.

[16] Shao Liu, T. Basar, and R. Srikant. Exponential-RED: A Stabilizing AQM Scheme for Low and High-Speed TCP Protocols. *IEEE/ACM Transactions on Networking*, 13(5):1068–1081, Oct. 2005.

[17] W. Stevens M. Allman, V. Paxson. TCP Congestion Control, April 1999. RFC 2581.

[18] Ratul Mahajan and Sally Floyd. Controlling High Bandwidth Flows at the Congested Router. In *Proceedings of IEEE ICNP '01*. IEEE, 2001.

[19] Richard Marquez, Isbel González, Niliana Carrero, and Yuri Sulbarán. Revisiting Adaptive RED: Beyond AIMD Algorithms. In *Proceedings of the 1st EuroFGI international conference on Network control and optimization*, NET-COOP'07, pages 74–83, Berlin, Heidelberg, 2007. Springer-Verlag.

[20] Mark Parris, Kevin Jeffay, and F. Donelson Smith. Lightweight Active Router-Queue Management for Multimedia Networking. In *Multimedia Computing and Networking, SPIE Proceedings Series*, pages 162–174, 1999.

[21] Dipesh M. Raghuvanshi, B. Annappa, and Mohit P. Tahiliani. On the Effectiveness of CoDel for Active Queue Management. In *Proceedings of Third International Conference on Advanced Computing & Communication Technologies*, ACCT '13, pages 107–114. IEEE Computer Society, 2013.

[22] K. K. Ramakrishnan and S. Floyd. The Addition of Explicit Congestion Notification (ECN) to IP, 2001. RFC 3168.

[23] Mohit P. Tahiliani, K. C. Shet, and T. G. Basavaraju. CARED: Cautious Adaptive RED Gateways for TCP/IP Networks. *Journal of Network and Computer Applications*, 35(2):857–864, March 2012.

[24] Rahul Verma, Aravind Iyer, and Abhay Karandikar. Active Queue Management using Adaptive RED. *IEEE/KICS Journal of Communications and Networks*, 5(3), 2002.

[25] Chonggang Wang, Jiangchuan Liu, Bo Li, Kazem Sohraby, and Y. Thomas Hou. LRED: A Robust and Responsive AQM Algorithm Using Packet Loss Ratio Measurement. *IEEE Transactions on Parallel and Distributed Systems*, 18(1):29–43, Jan. 2007.

[26] Bing Zheng and M. Atiquzzaman. DSRED: An Active Queue Management Scheme for Next Generation Networks. In *Proceedings of the 25th Annual IEEE Conference on Local Computer Networks LCN '00*, Washington, DC, USA, 2000. IEEE Computer Society.

[27] Kaiyu Zhou, Kwan L. Yeung, and Victor O. K. Li. Nonlinear RED: a simple yet efficient Active Queue Management Scheme. *Computer Networks*, 50:3784–3794, December 2006.