Enhancing Performance of Internetworking with Novel RED-AT Congestion Control Algorithm

Z.M.Patel, Dr. U.D.Dalal

Abstract— Congestion Control on Internet has drawn attention of many researchers since its presence from early 80's. This lead to development of congestion control algorithms such as Droptail, RED and its variants, Random Exponential Marking (REM), Adaptive Virtual Queue (AVQ) etc. Out of these, RED has been promising and widely used in router for long. This paper proposes RED with Adaptive Threshold (RED-AT) algorithm which is based on the current and few past observations of average queue size to adjust thresholds. Thus it can correctly predict the behavior of queue dynamic and make appropriate adjustment in upper and lower thresholds. It is observed the proposed RED-AT scheme mitigates oscillations in average and instantaneous queue size and enhances link utilization. Simulation results show improvement in throughput, jitter and packet loss performance of a router.

Index Terms— Congestion control, RTT, Adaptive RED, RED-AT, Window control,bandwidth, ACK.

1 Introduction

Congestion on internet has been under constant investigation since its presence is first detected in mid 80s. The sophistication of internetworking would be useless if we do not cater the problem of congestion. This problem is now becoming more and more relevant with substantial increase in internet traffic in last decade. Congestion collapse [1] is that state that a packet switched data network can reach when little or no useful communication happens. Congestion usually occurs at "choke points" in the networks when total incoming traffic to a node exceeds outgoing bandwidth capacity. Congestion control deals with controlling data traffic entry into a communication network so as to avoid overloading of routers/gateways and hence avoids congestion collapse.

The congestion control schemes can be classified into two categories: *source based* and *gateway based*. In a source based scheme, the presence of congestion is detected through the effects of congestion, e.g., packet loss, increased round trip time, changes in the throughput gradient etc., rather than the congestion itself e.g. overflowing queues. After detection of congestion, transport layer at sender reduces window size to reduce sending rate. Gateway based scheme detects incipient of congestion through large queue size and notifies host by marking or dropping packets. The RED [2] is popular gateway based congestion management method and designed to work with transport layer protocol such as TCP. A model of RED feedback control system with TCP flows has been investigated in [3]. RED tends to drop packet randomly with increasing probability with increase in congestion. This drop event acts as feedback signal to hosts and they reduce their sending rate using window control of transport layer protocol.

- *Z.M.Patel is currently pursuing PhD degree program in Electronics Engineering from NIT, Surat, India. E-mail: zmp@eced.svnit.ac.in*
- *Dr. U.D.Dalal is an Associate Professor in Dept. of Electronics Engineering, NIT Surat, Inida. E-mail: udd@eced.svnit.ac.in*
(*This information is optional; change it according to your need.*)

RED aims to control the average queue size by indicating to the end hosts when they should temporarily slow down transmission of packets. RED takes advantage of the congestion control mechanism of TCP. By randomly dropping packets prior to periods of high congestion, RED tells the packet source to decrease its transmission rate. Assuming the packet source is using TCP, it will decrease its transmission rate until all the packets reach their destination, indicating that the congestion is cleared. You can use RED as a way to cause TCP to slow down transmission of packets. TCP not only pauses, but it also restarts quickly and adapts its transmission rate to the rate that the network can support.

Though many algorithmic modifications to basic RED are suggested to enhance performance, most of them are less effective in preventing high packet loss and large jitter. The proposed algorithm RED-AT uses current and few past values of average queue size to modulate thresholds. Our scheme also restricts variations in thresholds within certain bounds and ensures less frequent adjustment of thresholds. As a result, we observe relatively smaller variations in amplitude of instantaneous and average queue size. This improves jitter performance of router since variations in delay are reduced. Moreover, by preventing packet drops at low to medium traffic load, high throughput and low packet loss are achieved.

The rest of the paper is organized as follows. Transport layer based end-to-end congestion control mechanism such as TCP Tahoe, TCP Reno and TCP Westwood are discussed in Section 2. Section 3 explains basic RED operation and its extensions such as adaptive RED [4][5] and reconfigurable threshold RED. Section 4 presents adjustable threshold algorithm named RED-AT proposed by us. Simulation results and related discussion are given in Section 5. Finally, the concluding remarks are made in Section 6.

2 End to End Congestion Contnrol

The algorithm proposed by Van Jacobson [1] for internet congestion control based on end-to-end principle has been quite successful in avoiding congestion collapse. His work was based on adjustment of *TCP congestion window* to respond to

International Journal of Scientific & Engineering Research Volume 5, Issue 11, November-2014
ISSN 2229-5518

network congestion. This window ensures that packets enter into the network at the same rate that they are exiting with a full window of packets in transit. A connection in this state is said to be in equilibrium and congestion collapse is unlikely. After Jabobson, many TCP based end-to-end congestion control algorithms were proposed to improve network stability, throughput and fairness and to keep network utilization high. In following sections, we shall discuss end-to-end TCP based congestion control schemes.

2.1 TCP Tahoe and Reno

TCP congestion control [6] under Tahoe comprises of three parts: Slow start, Congestion Avoidance (CA) and Fast Retransmit. Reno algorithm [7] is improvement over Tahoe that adds *Fast Recovery* operation. The operation of both Tahoe and Reno is based two variables, congestion window (*cwnd*) that determines number of outstanding segments at any time and slow start threshold (*ssthresh*) is the value up to which congestion window can grow exponentially. The *cwnd* and *ssthresh* are used to throttle TCP sending rate to match available network bandwidth.

The slow start begins in the exponential growth phase initially with a *cwnd* of 1 segment and increases it by one Segment Size (SS) for each new *ACK* received (Fig. 1). If the receiver sends an ACK for every segment, this behavior effectively doubles the window size each round trip of the network. If the receiver supports delayed ACKs, the rate of increase is lower, but still increases by a minimum of one maximum SS each round-trip time. This behavior continues until the *cwnd* reaches the size of the receiver's advertised window or until a loss occurs. Once the *cwnd* reaches the *ssthresh*, TCP goes into congestion avoidance mode where each new *ACK* increases the *cwnd* by *1/cwnd*. This results in a linear increase of the *cwnd*.

When a loss occurs due to timeout, half of the current *cwnd* is saved as *ssthresh* and slow start begins again from its initial *cwnd*. If three duplicate ACKs are received (i.e., four ACKs acknowledging the same packet, which are not piggybacked on data, and do not change the receiver's advertised window), Reno will halve the congestion window (instead of setting it to 1 MSS like Tahoe), set the slow start threshold equal to the new congestion window, perform a *fast retransmit*, and enter a phase called *fast recovery*. In fast recovery, TCP retransmits the missing packet that was signalled by three duplicate ACKs, and waits for an acknowledgment of the entire transmit window before returning to congestion avoidance. If an ACK times out, slow start is used.

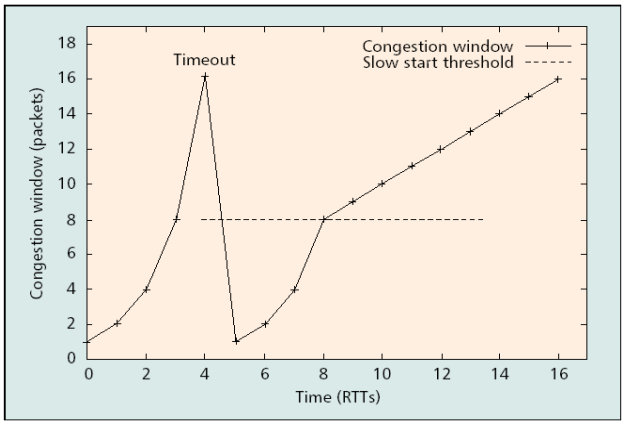

Fig. 1. TCP congestion window dynamics [6].

New Reno [8] is an improved version of Reno that avoids multiple reductions of the *cwnd* when several segments from the same window of data get lost. The work of [9] presents analytic model for the throughput of a TCP New Reno bulk data transfer as a function of round-trip time (RTT) and loss rate. This proposed model can predict steady-state TCP NewReno throughput for a wide range of network conditions.

2.2 TCP Vegas

TCP Vegas [10] was the first attempt to introduce mechanism of detecting network congestion before packet losses. The working of TCP Vegas is based on measurement of RTT rather than lost packets to gauge network capacity. It anticipate onset of congestion by computing the difference between the actual input rate (*cwnd*/*RTT*) and expected input rate (*cwnd*/RTT_{min}) where *cnwd* is current window size, RTT is actual round trip time and RTT_{min} is minimum round trip time of a packet. The difference in these rates can be translated in to the difference between the window size and the number of acknowledge packets denoted as *Diff* given below

$$Diff = (\exp ected\ rate - actual\ rate)\, RTT_{\min}$$
$$= \left(\frac{cwnd}{RTT_{\min}} - \frac{cwnd}{RTT} \right) RTT_{\min} \quad (1)$$

Based on *Diff*, source updates its window size as

$$cwnd = \begin{cases} cwnd + 1; & if\ Diff < cwnd + \alpha \\ cwnd - 1; & if\ Diff > cwnd + \beta \\ cwnd; & otherwise \end{cases} \quad (2)$$

Fig. 2 illustrates the behaviour of TCP Vegas. If the difference is smaller than a threshold *cwnd*+α then the *cwnd* is additively increased, whereas if the difference is greater than another threshold *cwnd*+β then the *cwnd* is additively decreased; finally, if the difference is smaller than *cwnd*+β and greater

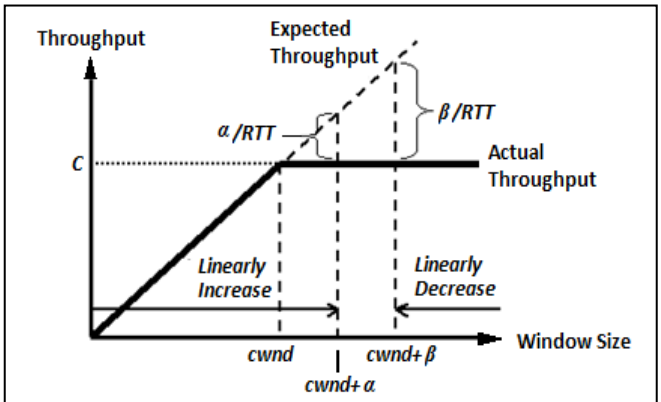

Fig. 2. Window size control in TCP Vegas

International Journal of Scientific & Engineering Research Volume 5, Issue 11, November-2014
ISSN 2229-5518

than *cwnd*+α, then the *cwnd* is kept constant. TCP vegas tries to keep at least α packets but no more than β packets in a queue. The reason behind this is that TCP Vegas attempts to detect and utilize the extra bandwidth whenever it becomes available without congesting the network. This when there is only one connection, the window size of TCP Vegas converges to a point that lies between *cwnd*+α and *cwnd*+β. This mechanism is fundamentally different from that used by TCP Reno. TCP Reno always updates its window size to guarantee full utilization of available bandwidth, leading to constant packet losses where a TCP Vega does not cause any oscillation in window size once it converges to an equilibrium point.

TCP Vegas does not have any mechanism that handles the rerouting of connection. If the route of a connection is changed by a gateway switch, then without an explicit signal from switch, the end host cannot directly detect it. If the new route has a shorter propagation delay, this does not cause any serious problem because most likely some packets will experience shorter round trip delay and RTT_{min} will be updated. On the other hand, if the new route for the connection has a longer propagation delay, the connection will not be able to tell whether the increase in the round trip delay is due to congestion in the networks or change in the route. It was observed that the performance of TCP Vegas decreases significantly when the network RTT exceeds 50ms.

2.3 TCP Westwood+

TCP Westwood [11] is modification to TCP New Reno [8] and it applies to sender-side-only. Its working is based on the idea of estimating available end-to-end bandwidth by counting and filtering the returning ACKs and adaptively adjusts the *ssthresh* and *cwnd* after congestion episode i.e. after timeout or after detecting three duplicate ACKs. Unlike TCP Reno, which *halves* the congestion window after three duplicate ACKS, TCP Westwood sets *cwnd* and *ssthresh* which are consistent with estimated available bandwidth at the time congestion occurs. This ensures fast recovery with better throughput and delay performance. TCP Westwood+ works even in the presence of ACK compression.

The amount of data acknowledged between ACK reception is then used to compute the bandwidth of the link for the considered ACK interval. The calculated bandwidth is then passed through Tustin approximation low pass filter to filter out the high frequency components. A simplified form of the filter [11] that is used in the implementation of the protocol is given as:

$$\hat{b}_k = a\hat{b}_{k-1} + \frac{(1-a)}{2}[b_k - b_{k-1}] \qquad (3)$$

Where a is a weighing factor set to 0.9 and b_k and $\hat{b}_k$ are the instantaneous and the average measurement of the available bandwidth respectively at the time instant t_k.When three DUPACKs are received, both the congestion window (*cwnd*) and the slow start threshold (*ssthresh*) are set equal to the estimated bandwidth times the minimum measured round trip time (*RTTmin*); when a coarse timeout expires the *ssthresh* is set as before while the *cwnd* is set equal to one. The pseudo code of the Westwood+ algorithm is given in Fig. 3.

The TCP Westwood+ is particularly effective in wireless links where bursty losses due to erroneous radio channel are often misinterpreted as a symptom of congestion by current TCP schemes and thus lead to an unnecessary window reduc-

a) On ACK reception:
   cwnd is increased accordingly to the Reno algorithm;
   the end-to-end $\hat{b}_k$ is computed;
b) When 3 DUPACKs are received:
   ssthresh =max(2, ($\hat{b}_k$ * *RTTmin*) / *seg_size*);
   cwnd = *ssthresh*;
c) When coarse timeout expires:
   ssthresh = max(2, ($\hat{b}_k$ * *RTTmin*) / *seg_size*);
   cwnd = 1;

Fig. 3. TCP Westwood Algorithm

tion. TCP Westwood+ is able to discriminate the of packet loss (wireless channel error or congestion) by estimating end-to-end bandwidth. TCP Westwood+, TCP New Reno and Vegas TCP are evaluated and compared in [12] to investigate their effectiveness in terms of goodput, fairness and friendliness. The results show that Westwood+ remarkably improves utilization of wireless links. An NS-3 implementation of [13] shows the tradeoffs between TCP Westwood and TCP Westwood+ in terms of congestion and aggressiveness.

4 Gateway Based Congestion Control

End-to-end congestion control mechanisms that we discussed in previous section act after the occurrence of network congestion at overflowing gateways. This would lead to large queue size at gateways and significantly increase average delay in the network. It would be more efficient to detect incipient of congestion at gateway itself and provide feedback end hosts either by dropping packets or marking congestion bits in packets. Only the gateway has a unified view of the queuing behaviour over time. In addition, a gateway is shared by many active connections with a wide range of roundtrip times, tolerances of delay, throughput requirements, etc.; decisions about the duration and magnitude of transient congestion to be allowed at the gateway are best made by the gateway itself.

A simple method for a gateway to notify congestion to sources is to drop packets when queue becomes full, this is called *Tail Drop*. The drawback of Tail Drop gateway is global synchronization; a phenomenon where all senders who share common bottleneck router/gateway link slows down at the same time resulting in sharp decrease in link utilization. The tail-drop routers are also biased against the bursty flows because when a burst of packets from a sender arrives on fully occupied queue a sustained packet drop belongs same source occurs. To circumvent these problems, a gateway/router should play active role in detecting/preventing congestion. This is known as active queue management (AQM). The most popular AQM method is *Random Early Detection* (RED) [2] that

International Journal of Scientific & Engineering Research Volume 5, Issue 11, November-2014
ISSN 2229-5518

effectively handles problem of global synchronization and bias against bursty flows. Its principle is to monitor average queue size and keep it low by dropping packets with some probability before queue gets full. In this way, it achieves high link utilization and low average delay simultaneously. In [14], various variants of RED techniques have been presented and analyzed with respect to queue length stability and delay.

An alternative method to notify sources in RED is to modify congestion bits in packet header which explicitly sends feedback about network congestion level. This is called *Explicit Congestion Notificacion* (ECN). ECN [15] is an optional feature that may be used between two ECN-enabled endpoints when the underlying network infrastructure also supports it. When ECN is successfully negotiated, an ECN-aware router may set a mark in the IP header instead of dropping a packet in order to signal impending congestion. The receiver of the packet echoes the congestion indication to the sender, which reduces its transmission rate as though it detected a dropped packet. As expected, ECN reduces the number of packets dropped by a TCP connection, which, by avoiding a retransmission, reduces latency and especially jitter. Congestion control scheme proposed in [16] combines feedback of ECN bits and available bandwidth estimate to adjust behavior of TCP sources to achieve high efficiency.

4.1 Basic RED

RED algorithm for AQM works on monitoring average queue size and dropping packets with increasing probability as average queue size increases. The objective is to detect incipient of congestion well in advance and notify end hosts, allowing them to reduce their sending rate that avoids router queue overflow and excessive packet dropping. The value of average queue size is compared with two thresholds; upper threshold min_{th} and upper threshold max_{th}. When average queue size increases above some value called min_{th} packets are randomly dropped with probability that increases from 0 at min_{th} to maximum value max_p at max_{th}. All incoming packets are dropped if average queue size exceeds max_{th}. With early congestion notification and burst absorption, RED simultaneously achieves low average queuing delay and high throughput. Although RED is useful in detecting congestion, its working is very sensitive to parameters such as min_{th}, max_{th}, max_p and w_q. Moreover, RED performance depends on number of connections multiplexed across the link. To address shortcoming of RED, adaptive mechanism are researched by [4] and [5].

4.2 Adaptive RED

The original adaptive RED (ARED) given by [4] performs directly on link utilization and packet drops rather than instantaneous or average queue size. It maintains a single probability, p_m, which it uses to mark (or drop) packets. The algorithm increments p_m if packets are being excessively dropped due to buffer overflow, thus increasing the rate at which it sends back congestion notification to hosts. Conversely, if the queue buffer becomes empty or if the link is idle, it decreases marking probability p_m. This effectively allows the algorithm to send back congestion notification at the correct rate. Fig. 4 shows ARED algorithm where *last_update* is the last time when p_m is updated, *freez_time* is determines the minimum time interval between two successive updates of p_m and d_1 & d_2 are amount by which p_m is incremented/decremented when queue buffer overflows/empties. The d_1 was selected larger than d_2 to react quickly to substantial rise in traffic load. The paramters *freez_time*, d_1 and d_2 control how quickly the marking probability changes over time.

Upon packet loss event:
If (now – *last_update*) > *freez_time*)
$p_m = p_m + d_1$;
last_update = now;
Upon link idle event:
If (now – *last_update*) > *freez_time*)
$p_m = p_m - d_2$;
last_update = now;

Fig. 4. Pseudocode for Adaptive RED algorithm [4]

Another proposal [5] tunes parameter max_p for making RED adaptive. It is revised version of ARED with several algorithmic modifications and can be implemented as simple extension to RED routers. The overall idea is quite similar to original ARED of keeping average queue size between min_{th} and max_{th} by adapting max_p but with following differences.

- max_p is intended to keep average queue size halfway between min_{th} and max_{th}
- Adaptation of max_p is in small steps and slow over the period typically greater than RTT
- max_p is constrained within range [0.01, 0.5] and AIMD is adopted.

Instead of using multiplicative increase and decrease, [5] uses additive increase and multiplicative decrease (AIMD) strategy for adjusting max_p.

4.3 Reconfigurable Threshold RED

The RED behaviour also depends on thresholds. Too small values of thresholds would cause many timeouts and load the network heavily; on the other hand, if the thresholds are too large, large size of packet queue at router will incur large delay even if network is lightly loaded. Many researchers have focused on adaption in thresholds to improve RED performance. The proposal of [17] claims to improve throughput and packet drop rate performance with adaptive threshold technique. The basic idea is that the average queue size should reach the maximum threshold when or before the instantaneous queue size reaches the maximum buffer size. The value of min_{th} is initially set to 4 packets and max_{th} is set to 2 * min_{th} according to rule-of-thumb [18]. During operation, lower threshold min_{th} is varied depending on burst and number of flows. When the min_{th} reached the maximum value called *target*, max_{th} is changed dynamically as per equation below

$$\max_{th} = \max_{th} + (1-w)^{nk} a_0 + (1-(1-w)^{nk}) q_0 \qquad (4)$$

International Journal of Scientific & Engineering Research Volume 5, Issue 11, November-2014
ISSN 2229-5518

where, n=no. of nodes
k= burst size
a_0 = current value of average queue size
q_0= instantaneous value of queue

Work of [19] proposes an algorithm named Preferential Dynamic Threshold – RED (PDT-RED). In this method, packets are classified into several types. Each type of packet has a priority set in header by sending host. The minimum and maximum thresholds are varied with unused buffer space and priority of a packet.

5 Proposed RED-AT algorithm

As pointed out earlier that operation of RED is very sensitive to its parameters such as max_p, min_{th}, max_{th}, w_q, etc. The fix value often degrades the performance of RED. There is no single set of RED parameters that can work well under difference congestion scenarios; therefore, adaptive strategies have been devised for practical implementation of RED. Adaptive mechanism suggested by [4] is based on link utilization and packet loss rather than average or instantaneous queue size to change packet marking probability.

To achieve better performance in terms of drop rate and link utilization, we adjust thresholds adaptively based on values of average queue size over a few past observations and hence this proposed method is termed as Random Early Detection with Adaptive Threshold (RED-AT). In this method, the thresholds are adjusted around their nominal values (min_{th0} and max_{th0}) within finite upper and lower limits. As shown in Fig. 5 min_{th} can be adjusted between two bound i.e. min+ and min⁻ whereas max⁺ and max⁻ are bounds around max_{th}. Initially minimum and maximum threshold are set to nominal values. During operation, thresholds are tuned depending on value of average queue size observed over past few periods.

The drop probability plot is divided into three regions. In region a, where average queue size is low, we gradually increase min_{th} by amount equal to fraction of dynamic range of min_{th} factor α and inrease max_{th} by fraction of dynamic range of max_{th}. This makes sure that algorithm is gentle and less aggressive under low traffic. In region c, where average queue size is high, we gradually decrease min_{th} by factor α and max_{th} by factor of β. This makes sure that the average queue size does not become exceedingly large and hence delay. The value of β is smaller than α because link underutilization can occur if congestion management is too conservative or too aggressive but packet loss occur only when congestion management is too conservative. The pseudocode for adaptive threshold adjustment is given in Fig. 6. The following terms and symbols are used in the pseudocod.

- q_{avg}→ average queue size
- min_{th0}, max_{th0} → initialized value of min_{th} and max_{th} respectively
- min^+ , min^- → upper and lower bound on dynamic range of min_{th}
- max^+ , max^- → upper and lower bound on dynamic range of max_{th}

Our method not only uses current average value but also few past average values of queue size to decide the region of

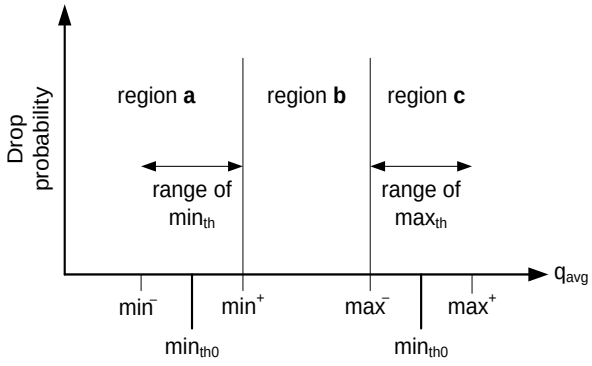

Fig. 5. Bounds on upper and lower thresholds.

```
Upon every packe arrival:
if (qavg < min+) {
    ++cnt1;   cnt2=0; cnt3=0;
    if(cnt1 > 4) {
        minth += α (min+ – min–) ;
        maxth += β (max+ – max–) ;
      if (minth > min+)   minth = min+ ;
      if (maxth > max+)   maxth = max+ ;
      cnt1=0;
    }
 }
Else if (qavg > max–) {
      ++cnt2;  cnt1=0;  cnt3=0;
          if (cnt2 > 4) {
          maxth – = β (max+ – max–) ;
          minth – = α (min+ – min–) ;
          if (maxth < max–)   maxth = max– ;
          if (minth < min–)   minth = min– ;
          cnt2=0;
          }
    }
Else
{     ++cnt3;    cnt1=cnt2=0;
      If (cnt3 > 4) {
            minth = minth0 ;
            maxth = maxth0 ;
         cnt3=0;
  }
}
```

Fig. 6. Pseudocode of proposed RED-AT Algorithm

International Journal of Scientific & Engineering Research Volume 5, Issue 11, November-2014
ISSN 2229-5518

operation and corresponding threshold adjustment action. These new threshold values will be maintained at least for few periods (till next region of operation is decided). This avoids overly updates in thresholds upon every observation of average queue size and thus reduces oscillations in queue. The swing in instantaneous and average queue size Therefore, proposed RED-AT scheme exhibits lower variations in delay and hence offer low jitter. Moreover, it renders lower packet drop rate by keeping min_{th} at higher value under low to medium traffic load.

6 Simulation Results and Discussions

The main objective of the simulation experiments is to verify that RED-AT keeps the oscillation in average queue size within limits under varying traffic load conditions. Simulations also verify that proposed scheme indeed improves the throughput, drop rate and jitter performance. We conducted simulations on ns-2 [20] network simulator to evaluate the performance of RED-AT, conventional RED and adaptive RED. The network topology of our set up is shown in Fig. 7 where senders (S_1 to S_n shown as circles) are connected to one end to router R1 and sink node is connected to router R2. Hence, the congested link under consideration is between R1-R2. Above mentioned three AQM scheme will be implemented at router R1 which has queue buffer size of 50 packets. Traffic is generated by TCP sources connected to node S_1 to S_n. In our experiments, mix traffic is generated with 70 FTP sources and 30 best effort (HTTP) sources. Unless otherwise stated, we assume that all packets generated by the senders are 1000 bytes long.

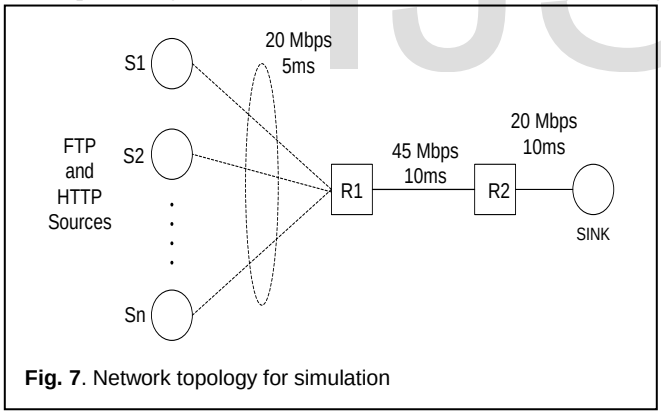

Fig. 7. Network topology for simulation

6.1 Experiment 1

In this experiment we investigate dynamic behaviour of conventional RED and proposed RED-AT congestion control scheme with moderately congested link. Fig. 8 and Fig. 9 show instantaneous and average queue size variation with time considering number of TCP flows N=80, max_p=0.1 and w_q=0.002 for RED and RED-AT respectively. In both cases, lower threshold min_{th}=10 packets and upper threshold max_{th}=30 packets are considered. From results, we noticed that the oscillations in both instantaneous and average queue size are reduced when RED-AT is used as compared to RED. Due to small variation in average queue size, RED-AT limits delay variations. Moreover, RED-AT keeps average queue size remains slightly higher than RED since RED-AT avoids excessive packet drops

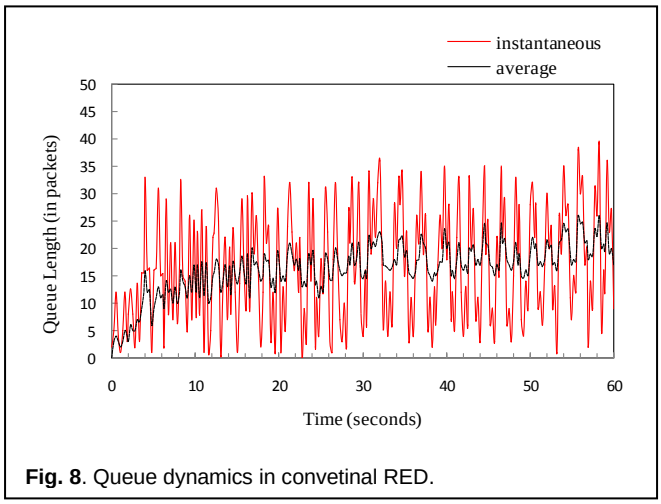

Fig. 8. Queue dynamics in convetinal RED.

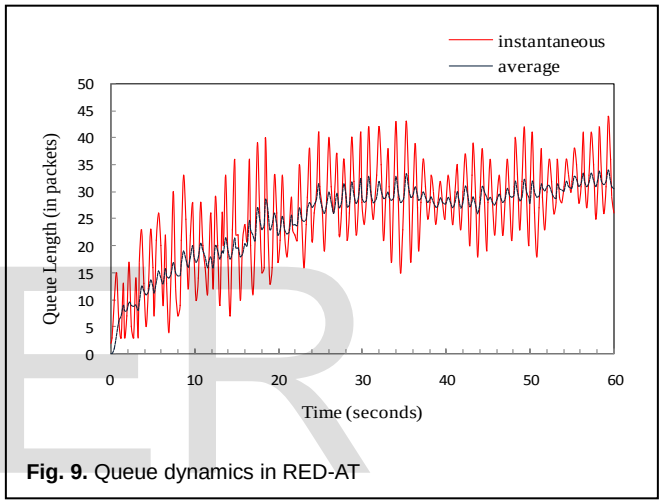

Fig. 9. Queue dynamics in RED-AT

by adjusting thresholds. This enhances utilization and proves throughput.

6.2 Experiment 2

In this experiment, we evaluate the performance of RED, ARED and RED-AT in terms of throughput, jitter and drop rate by varying number of TCP flows from 5 to 100. Each point on the result plot is obtained from 150s simulation run and statistics are collected/recorded simultaneously in result file. Once again, all three schemes use parameters w_q =0.002, max_p =0.1, min_{th} =10 packets and max_{th}=30 packets.

The simulation results reveal the ability of proposed RED-AT scheme to avoid excessive packet drop rate as compared to other two schemes as seen from Fig. 10. This is because RED-AT prevents early initiation of packet drop at low queue size and keeps average queue size away from maxth. Comparing RED and ARED, at low to medium congestion, packet drop rate of RED closely follows ARED but in high traffic conditions, ARED exhibits lower drop rate than RED. The throughput performance shown in Fig. 11 illustrates success of RED-AT in maintaining higher throughput. The throughput of RED declines sharply under high traffic load whereas throughput of ARED and RED-AT converges under high traffic load. As illustrated in experiment 1, variation in instantaneous and average queue size in case of RED-AT is much smaller than RED

International Journal of Scientific & Engineering Research Volume 5, Issue 11, November-2014
ISSN 2229-5518

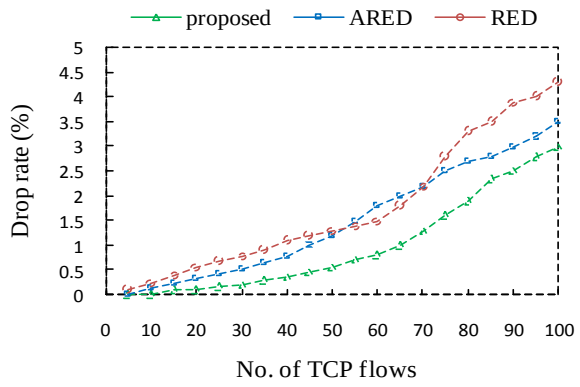

Fig. 10. Drop rate vs. number of flows

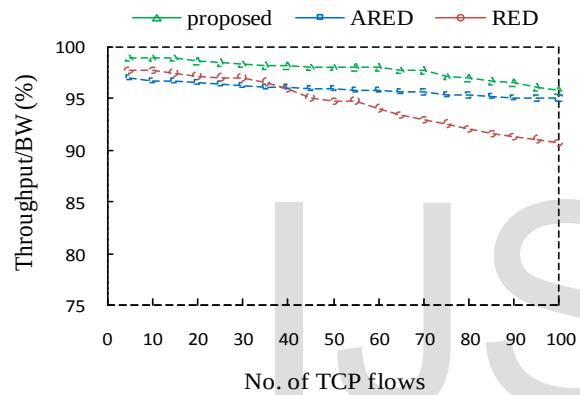

Fig. 11. Throughput vs. number of flows

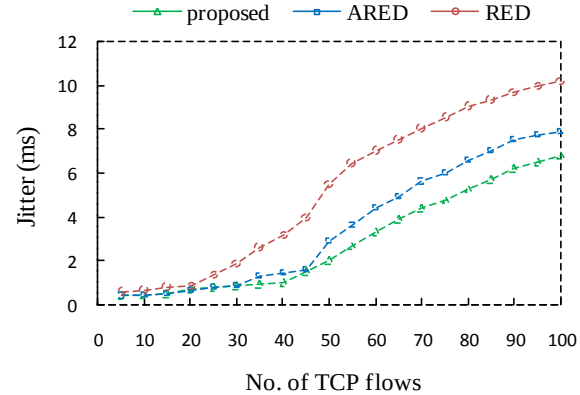

Fig. 12. Jitter vs. number of flows

and ARED. Hence RED-AT offers fewer variations in delay which is reflected in the jitter plot shown in Fig. 12. Under low to medium traffic load, RED-AT and ARED have roughly same jitter but at higher traffic load RED-AT has lower jitter than ARED. Due to large fluctuations in queue size, RED has highest jitter.

7 Conclusion

In this paper, we explored the opportunity of improving basic RED mechanism by adaptively adjusting the thresholds. Unlike other schemes which work on current value of instantaneous or average queue size, the proposed RED-AT algorithm uses current as well as few past values of average queue length to correctly predict the queue behavior and adjust the thresholds accordingly. It also restricts the dynamic range of threshold within bounds. This has resulted in reduced swing in the oscillations of actual and average queue size. RED-AT is also able to keep high utilization of link by maintaining larger average queue length, thus exhibiting high throughput compared to RED and ARED. Due to more stablility in instantaneous and average queue, RED-AT offers lowest jitter. Besides it keeps drop rate lower by keeping high value of low threshold under low to medium traffic load.

References

[1] V.Jacobson, "Congestion Avoidance and Control," *Proc. SIGCOMM'88*, vol.18, no.4, pp.314-329, Aug. 1988.

[2] S.Floyd and V.Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. on Networking*, vol.1, no.4, pp. 397-413, Aug. 1993.

[3] V.Firoiu and M.Borden, "A Study of Active Queue Management for Congestion Control," *Proc. of IEEE INFOCOM 2000*, vol.3, pp.1435-1444, Mar 2000.

[4] W.Feng, D.Kandlur, D.Saha and K.G.Shin, "Blue: An alternativeapproach to active queue management," *Proc. of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'01)*, Port Jefferson, NY, USA, pp. 41–50, June 2001.

[5] S.Floyd, R.Gummadi and S.Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management," Technical Report, ICSI, 2001.

[6] G.Xylomenos, G.Polyzos, P.Mahonen and M.Saaranen, "TCP Performance issues over Wireless Links," *IEEE Communication*, vol. 39, no.4, pp.52-58, 2001.

[7] V.Jacobson, "Berkeley TCP evolution from 4.3-Tahoe to 4.3 Reno," *Proc. 18th IETF*, Vancouver, Canada, pp.365-376, Aug. 1990.

[8] S.Floyd, T.Henderson and A. Gurtov, "The New Reno Modification to TCP's Fast Recovery Algorithm," RFC 3782, April 2004.

[9] Nadim Parvez, "An Analytic Throughput Model for TCP NewReno," *IEEE/ACM Transaction on Networking*, vol.18, no.2, pp.448-461, 2010.

[10] L.S.Brakmo, S.W.O'Malley and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," *SIGCOMM Computer Comm. Rev.*, vol. 24, no. 4, pp.24-35, 1994.

[11] S.Mascolo, C.Casetti, M.Gerla, M.Sanadidi and R.Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," *Proc. ACM MOBICOM*, pp.287-297, July 2001.

[12] L.A.Grieco and S.Mascolo, "Performance Evaluation and Comparison of Westwood+, New Reno, and Vegas TCP Congestion Control," *SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp.25-38, 2004.

[13] S.Gangadhar, T.A.N.Nguyen, G.Umapathi, and J.P.G.Sterbenz, " TCP Westwood(+) Protocol Implementation in ns-3," *Proc. ICST Conference on Simulation Tools and Techniques*, , ICST, Brussels, Belgium, pp.167–175, 2013.

International Journal of Scientific & Engineering Research Volume 5, Issue 11, November-2014
ISSN 2229-5518

[14] M.Cheng and X.Ma, "Performance Evaluation of Queue Management Methods for Congestion Control," *Journal of Information & Computational Science*, vol. 9, no. 6, pp.1599–1608, 2012.
[15] K.K.Ramakrishnan and S.Floyd, "The addition of explicit congestion notification (ecn) to IP," IETF RFC 3168, Jan. 2001.
[16] J.Wang, P.Dong, J.Chen, J.Huang, S.Zhang and W.Wang, "Adaptive Explicit Congestion Control Based on Bandwidth Estimation for High Bandwidth-delay Product Networks," *Elsevier Journal of Computer Communications*, vol. 36, no.10-11, pp.1235-1244, June 2013.
[17] M. Murshed, "Adaptive RED with Dynamic Threshold Adjustment," MS Research Report, Dept. of Electrical and Computer Engineering, Iowa State Univ., Ames, Iowa, 2005.
[18] S.Floyd, "RED: Discussions of setting parameters," available at http://www.icir.org/floyd/REDparameters.txt. 1997.
[19] L.Sun and L.Wang, "Novel RED Scheme with Preferential Dynamic Threshold Deployment," *Proc. of IEEE International Conference on Computational Intelligence and Security Workshop CISW 2007*, pp.854-857, Dec. 2007.
[20] TheNetwork Simulator-NS-2, http://www.isi.edu/nsnam/ns.