Computer Communication & Collaboration (Vol. 2, Issue 2, 2014)
ISSN 2292-1028(Print) 2292-1036(Online)
DOIC: 2292-1036-2014-02-001-22

A high-Performance Router: using fair-dropping policy

Seyyed Nasser Seyyed Hashemi (Correspondence author)

Young Researchers Club, Ardabil branch, Islamic Azad University, Ardabil, IRAN

E-mail: naser_seyed_hashemi@yahoo.com

Shahram Jamali

Department of Computer Engineering, Mohaghegh Ardabili University, Ardabil, IRAN

E-mail: jamali@iust.ac.ir

Hadi Moratez

Department of Computer Engineering, Islamic Azad University, Tabriz branch, Tabriz, Iran

E-mail: h.motarez@gmail.com

Abstract: Active queue management (AQM) employs some types of feedbacks to disseminate congestion information to the sources. Packet dropping is the most used feedback on the Internet. This paper discusses that an unfair dropping discipline leads to performance reduction, the unfair bandwidth sharing and the instability on the Internet. Then it proposes a fair dropping AQM algorithm and shows how this fairness improves the network performance, fairness and stability. Extensive packet-level simulations done in ns-2 environment, show that the proposed algorithm, on one hand, presents a fair bandwidth allocation among competitor flows and on the other hand, this fairness is a launcher for improved performance in terms of link utilization, queue size and packet drop rate.

Keywords: Fairness, Network Congestion Control, Active Queue Management

1. Introduction

Congestion Control is one of the critical issues in computer networks and therefore, it has attracted a lot of attentions in recent years [1, 2, 3, 4, 5, 6 and 7]. The congestion control mechanisms on the Internet consist of the congestion window algorithms of transmission control protocol (TCP), running at end-systems, and active queue management algorithms running at the routers, seeking to obtain high network utilization, small amounts of queuing delay, and some degree of fairness among users. TCP sends data packets to network according to additive increase multiplicative decrease (AIMD) algorithm [8] in which sending rate increases one packet per round trip time (RTT) for probing available bandwidth, when no congestion occurs, and multiplicatively decreases to half of past rate, when congestion detected in network. The active queue management algorithm in the routers is responsible for producing congestion information for sources. The router notifies source from network congestion by dropping packets or in some other ways such as explicit congestion notification (ECN) [9]. RED [10] is the most famous AQM algorithm introduced to congestion control in 1993 (Figure 1). After that, various works have been done to enhance RED's performance [11, 12, 13, 14, 15 and 16]. In spite of the vast amount of researches in this field, there are yet many problems such as stability, efficiency and fairness that need more attentions to be solved. Fairness, which simply means allocating the same share to all, has been addressed in some works as [17, 18].

The CHOKe proposed in [17] is a queue management discipline to fair bandwidth allocation which doesn't require any per flow state information. This scheme aims to approximate the fair queuing policy and works with heuristic manner to decide to drop a packet during congestion in a random toss if it finds another packet of the same flow. The FABA [18] is a rate control based queue management algorithm which uses the notion of token buckets per flow for buffer management at the network edge. This scheme performs fair bandwidth allocation depend on permitted rate of token addition in a particular flow bucket. FABA could establish good fair allocation for both adaptive and non-adaptive flows. In [22] a network rate management protocol (RMP) has been proposed that controls the rate of all flows based on the fair target rates computed by the RMP. Upon their suggestion, each non-TCP aggregate flow is policed by its respective edge router and each TCP flow adapts its rate according to the RMP suggested fair target rate. On the other hand, there are many published works that address performance of the Internet congestion control [19, 20, and 21].

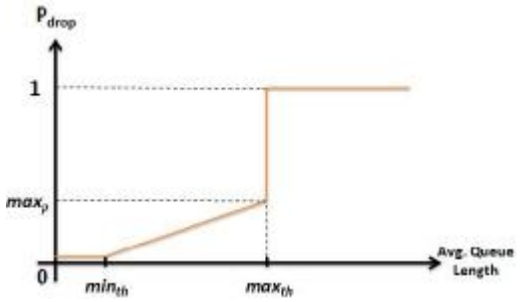


Figure 1 packet dropping function of RED

As we know, TCP tunes its sending rate based on received feedbacks that are often in the form of dropped packets. Hence, it seems that by accurately tuning of these feedbacks, we can direct the congestion control scheme to a stable, high-performance, and fair operational point. In this paper we study the fairness issue from fair dropping point of view and discuss how it provides fair bandwidth allocation along with improved performance metrics. It proposes a modification over the packet dropping pattern of RED algorithm, to achieve a fair packet dropping scheme considering the flows' RTT. We show that this fair dropping scheme not only leads to fair bandwidth sharing among passing flows, but also it will have an interesting and important effect, namely, improved performance on individual sources as well as the whole network. The proposed algorithm is called, FDD-RED (Fair Dropping Discipline over RED).

Rest of this paper is organized as follows. Section 2 presents the proposed algorithm. In section 3, we bring extensive packet-level simulation results and finally our conclusions are given in section 4.

2. FDD-RED: Fair Dropping Discipline for RED

This section addresses two important issues that affect success of an active queue management i.e. the performance and the fairness. The performance can be characterized in terms of parameters such as utilization, queue length and packet drops count. The fairness, on the other hand, is an important feature that is pursued in any congestion control scheme. It means that all flows passing a bottleneck link share the same amount of the bandwidth.

It is clear that packet dropping function in AQM algorithms affects both fairness and performance issues. A flow that encounters more drops will have smaller congestion window size and so lower sending rate and flows that encounter fewer drops will have more sending rate. Hence, it seems that packet dropping function is a place that has enough potential to manage fairness issue.

On the other hand, packet dropping function affects performance metrics such as throughput and queue size. When a flow's packets are dropped it reduces its throughput and consequently the queue will be shortened.

We believe that performance, fairness and stability of the network can be realized by a fair packet dropping function. In an unfair dropping scheme almost all flows are damaged. Obviously flows that encounter many drops will have low throughput and even may face the timeout. On the other hand, when these sources encounter burst drops, some other sources likely won't see any drop and hence they won't be informed about existence of the network congestion. The result is that they don't contribute in the congestion control process and since they continue to send with current rate, in the next round they may be faced with many drops. The network, on the other hand, is damaged in existence of such unfair packet dropping scheme. When AQM drops many packets from a certain flow with congestion window size of *w*, for the first drop the source's window size will be reduced by *w/2*, but for the second drop, the reduction amount will be *w/4* and for the third one it will be *w/8*. This means that in an unfair dropping scheme, while some sources don't receive any reduction commands from the AQM, some other sources receive many reduction commands but their reaction to these commands follows a decreasing trend. In other words, unfairness leads to reduced efficiency of the congestion control feedbacks that prevents the congestion control scheme reaching its target such as high performance, fairness and stability.

A short review over the existing AQM algorithms such as Drop tail and RED shows that they often drop burst packets from a small number of flows. This paper proposes an improvement over RED algorithm that equips it with a fair dropping discipline. The proposed approach, called FDD-RED, also takes into account a well-known drawback of TCP i.e. its unfair behavior against flows with different RTTs and tries to penalize the short RTT flows that utilize more than their fair share. To this end it maintains a Fair Dropping Index, defined as (1) for some recent flows and tries to keep it at the same level for various flows. According to this index a source whose RTT is lower than average of RTTs encounters more drops compared to the source whose RTT is longer than average of RTTs.

$$FDI_i = Drop_i \frac{RTT_{avg}}{RTT_i} \quad for\, i = 1, 2, ..., n \tag{1}$$

Where n is the number of flows that have passed through the queue during the last RTT_{avg} . $Drop_i$ is the number of packets that have been dropped from flow i, RTT_i is RTT of flow i and RTT_{avg} stands for the average of RTTs of all flows passing from the congested link.

Note that, there are several methods available to online estimation of network parameters. For example, according to the methods proposed in [23, 24, and 25] the network parameters such as RTT and number of flow connections (n) can be determined, accurately.

The detailed description of FDD-RED algorithm has been given in Figure 2. Note that this algorithm keeps a limited amount of information only for those flows that are passing during the last RTT_{avg} and hence has low overhead and is scalable.

Variables:
1: lastDropMarkTime[]: an array for last drop/mark time;
2: penalty[]: an array to keep penalty value of flows;
3: RTT[]: an array to keep round trip time of flows;
4: avgRTT: average RTT of flows;

```
5:   pkt: network packet;

Initialization:
1:   lastDropMarkTime[] ← 0;
2:   penalty[]← 0;

New flow arrival function():
1:   for a new flow (newflow) arrival
2:            RTT[newflow] = Estimate new flow's RTT;
3:            avgRTT = Estimated average RTT;

New packet arrival function():
1:   for each Packet(pkt) arrival from flow(flow)
2:            Calculate_drop_probability(pkt);
3:            if pkt must be Drop/Mark then
4:                     PenalizeFlow(flow)
5:   end for

PenalizeFlow function(flow):
1:       fid = select flow that has latest time in lastDropMarkTime[];
3:       penalty[flow] = penalty [flow] + avgRTT/RTT[flow];
4:       penalty [fid]   = penalty [fid] + avgRTT / RTT[fid];
5:       fid = select flow that has greatest value in Penalty[];
6:       pkt = select a packet from flow (fid);
8:       penalty [fid] = penalty [fid] – 1;
9:       drop/mark(pkt);
10:      lastDropMarkTime[fid] = now;
```

Figure 2 The FDD-RED algorithm

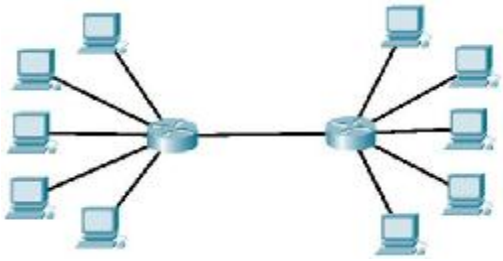


Figure 3 The network topology

3. Packet Level Simulation Results

In order to evaluate the proposed algorithm, we implement it as an extension to RED module of ns-2 simulator [27]. We present a group of simulation results to demonstrate the validity of our design. We demonstrate through extensive simulations that FDD-RED outperforms RED especially in environments with different RTTs. Our simulations also show that FDD-RED drops less packets, dampens oscillations and smoothly converges to high utilization, small queue size and fair bandwidth allocation.

3.1 Simulation Setup

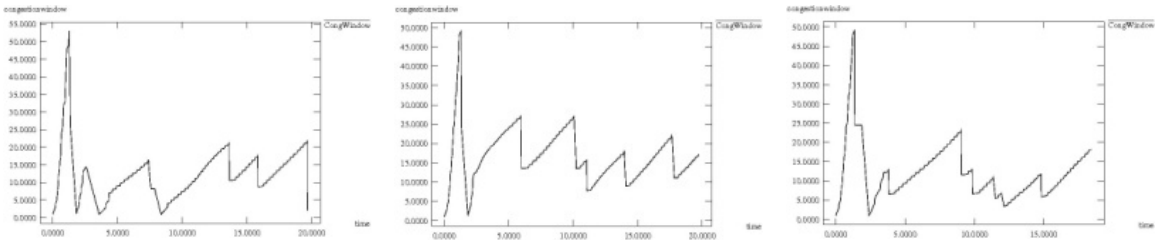
Our simulations use the topology in Figure 3. The bottleneck capacity is 1 Mbps, and the number of flows varies according to the objective of the experiments. The buffer size will be different in various scenarios. The data packet size is 500 bytes. Simulation duration varies depending on the propagation delay but is always as long as 20 seconds. All simulations were run long enough to ensure the system has reached a consistent behavior. The basic parameters of RED are selected as follows: min_{th} = 25 packets, max_{th} = 75 packets, max_p = 0.01 and w_q = 0.002.

ISSN 2292-1028(Print) 2292-1036(Online)
DOIC: 2292-1036-2014-02-001-22

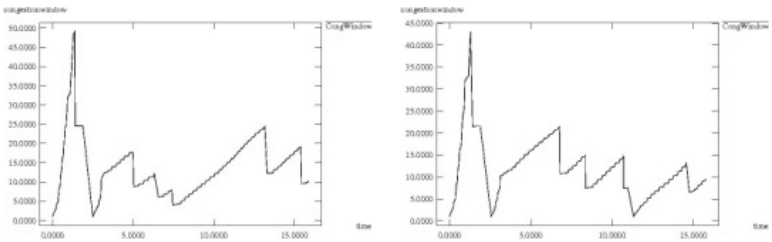
We present three simulation scenarios. In the first scenario we study about fairness and performance of FDD-RED algorithm in a network with identical RTTs. Second scenario studies the impact of the buffer size and number of flows on the performance of FDD-RED and finally scenario 3 examines how different RTTs affect FDD-RED's behavior. In each scenario the network is simulated once under RED algorithm and then under FDD-RED algorithm to compare the results.

3.2 Scenario 1: Fairness study for a Network with Identical RTTs

In this experiment 5 long-lived flows with RTT of 100 ms share the bottleneck in topology of Figure 3 and the buffer size is set to 100 packets in each router. Simulation results are given in Figures 4-8. Figure 4 shows how the congestion windows of various sources evolve during the time, when the queue is managed by RED algorithm. This figure shows clearly that different congestion windows have different trends during the time and hence don't follow a fair manner. This unfairness stems from the fact that RED drop different numbers of packets from various sources and hence they will have different window sizes. But as shown in Figure 5 when router's queue is managed by FDD-RED, various congestion windows follow similar trends and hence converge to a fair point. This fairness has roots in the fair dropping function of FDD-RED.

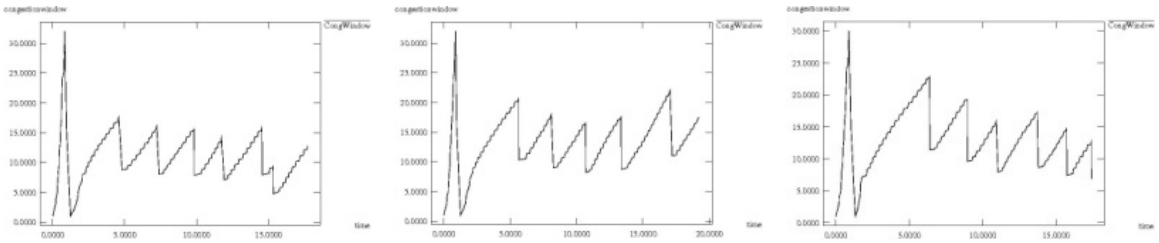


a. TCP window size in node 1 **b.** TCP window size in node 2 **c.** TCP window size in node 3

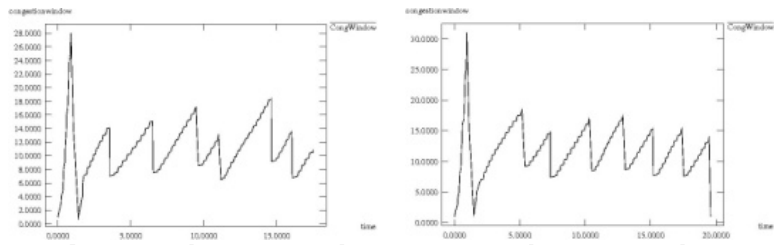


d. TCP window size in node 4 **e.** TCP window size in node 5

Figure 4 the congestion window size of sources with original-RED



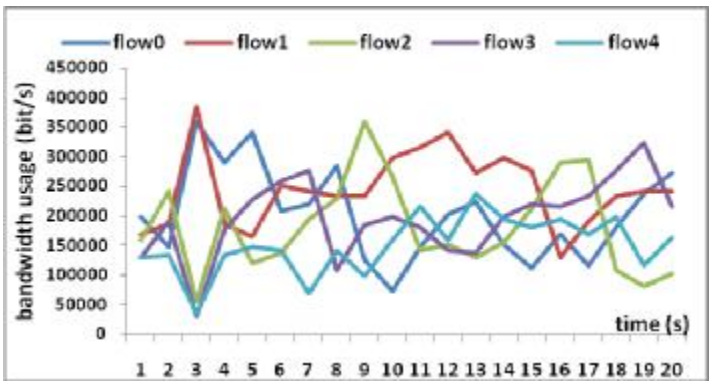
a. TCP window size in node 1 **b.** TCP window size in node 2 **c.** TCP window size in node 3



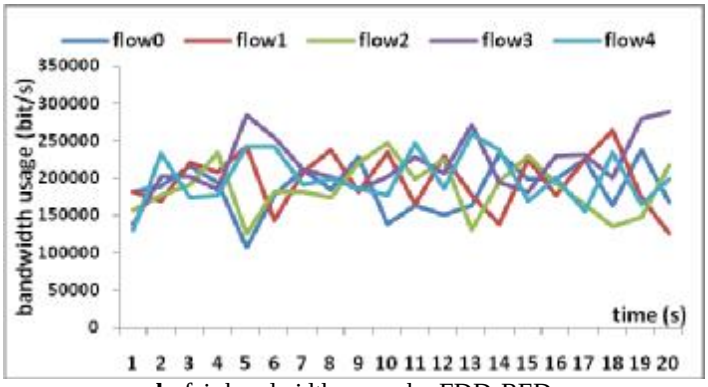
d. TCP window size in node 4 **e.** TCP window size in node 5

Figure 5 the congestion window size of sources with FDD-RED

Figure 6, on the other hand, shows bandwidth usage of various flows over bottleneck link for RED and FDD-RED algorithms. Figure 6.a shows that under governance of RED, various sources use different amounts of bandwidth, but as shown in Figure 6.b, in case of FDD-RED throughput the various sources are near to the fair amount.



a. unfair bandwidth allocation by original RED



b. fair bandwidth usage by FDD-RED

Figure 6 Bandwidth allocations for sources (a) original-RED (b) FDD-RED

Although fairness of FDD-RED was expected due to its fair dropping function, simulation results show another important achievement for FDD-RED i.e. its high performance in compared with RED. Average throughput of RED and FDD-RED have been given in Figure 7 according to which FDD-RED exhibits more fair behavior and its throughput for various flows are identical approximately. Throughput of the bottleneck link has been given in Figure 8 for RED and FDD-RED algorithms. This figure shows that FDD-RED converges more rapidly to a steady-state in which the bottleneck is fully utilized.

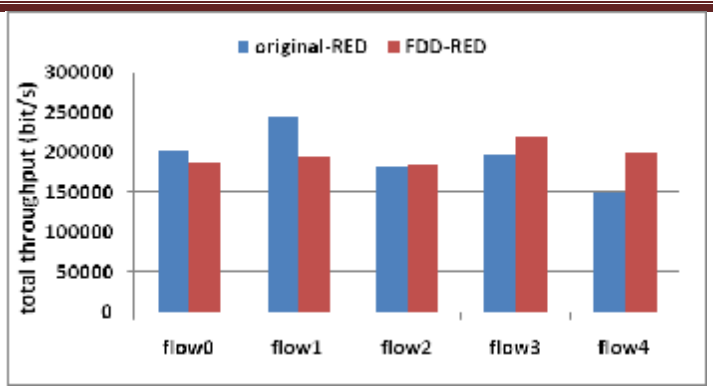


Figure 7 average throughputs for flows

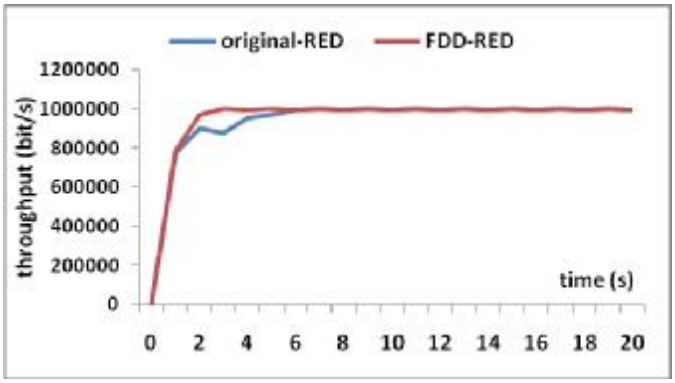
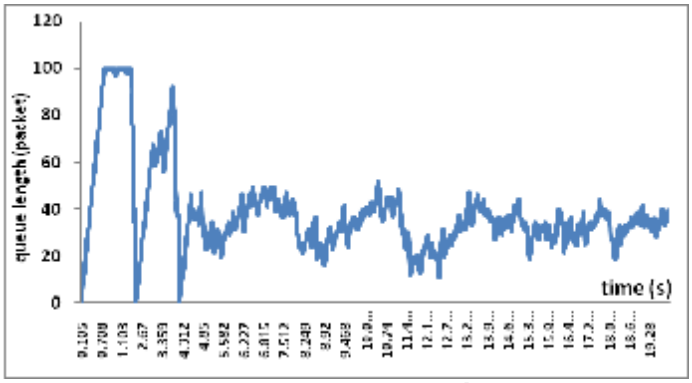
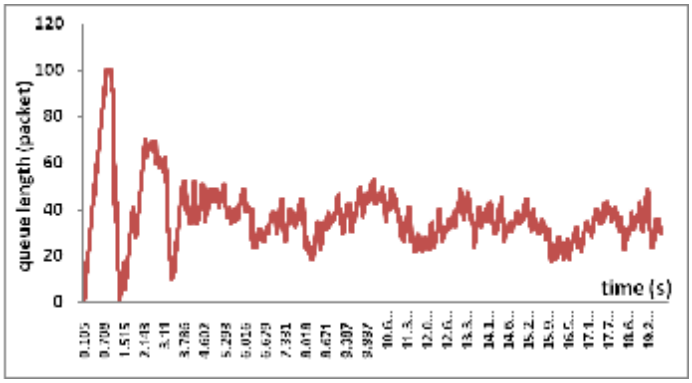


Figure 8 Bottleneck link throughputs

Figure 9 shows instantaneous queue length in bottlenecked link for both RED and FDD-RED. It can be observed that fair behavior of queue management discipline could affect stability of queue length. The queue length fluctuation in FDD-RED is obviously lower than original RED that has been shown in figures 9.a and 9.b.



a. RED's queue evolution



b. FDD-RED's queue evolution

Figure 9 instantaneous queue lengths for (a) original-RED (b) FDD-RED

Table 1 compares number of packets dropped by RED and FDD-RED algorithms. It shows that RED drops more packets than FDD-RED algorithm. Since FDD-RED drops packets uniformly from all passing flows then its feedbacks will be timely and efficient. Consequently there will be no need to further feedbacks in form of more dropped packets.

Table 1 Packet dropping for different RTTs

Algorithm	total dropped (packet)	avg. dropping rate (packet/s)
RED	**149**	**7.45**
FDD-RED	**67**	**3.35**

3.3 Scenario 2: Effect of Flows' Number and Buffer Capacity

This experiment fixes the bottleneck bandwidth at 1 Mbps and RTT at 100 ms and repeats the simulation by various numbers of FTP sources and different buffer sizes. Other parameters have the same values used in the previous experiment. Figure 10 shows the number of dropped packets for different buffer sizes ranging from 10 to 200 packets. According to this figure, FDD-RED drops fewer packets compared with RED. Figure 11, on the other hand, shows the dropped packets number for various flow numbers ranging from 5 to 100 flows. Again it can be found that FDD-RED has lower drop count than RED algorithm for different flow numbers.

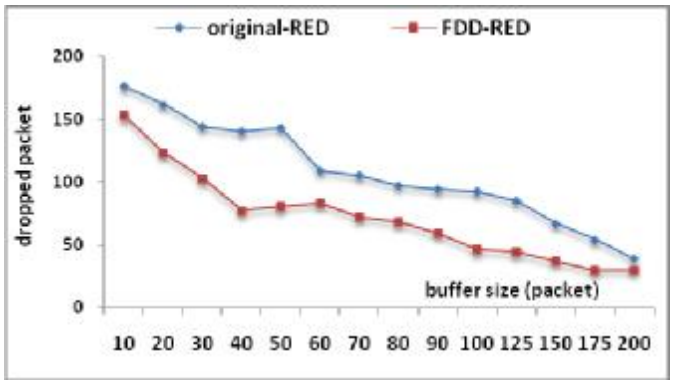


Figure 10 Dropped packet versus buffer size

ISSN 2292-1028(Print) 2292-1036(Online)
DOIC: 2292-1036-2014-02-001-22

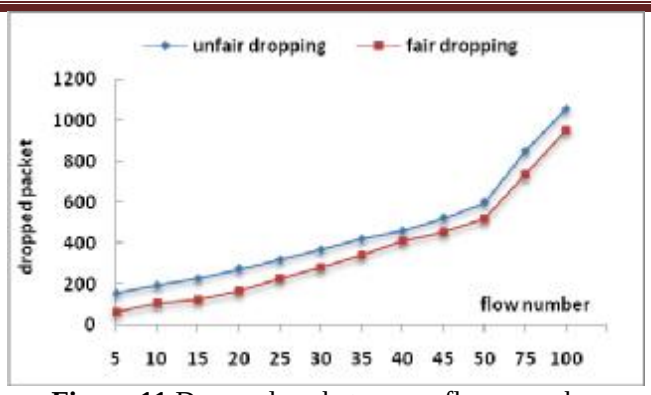


Figure 11 Dropped packet versus flows number

3.4 Scenario 3: Effect of Heterogeneous RTTs

In this scenario the network setup is as in scenario 1, except that it considers heterogeneous RTTs to study about fairness of the proposed algorithm. For this purpose we consider two experiments. In the first experiment all flows have same RTT of 100 ms, but in the second scenario the five flows have RTT values 20 ms, 40 ms, 60 ms, 80 ms and 100 ms. In both experiments the network is simulated once under FDD-RED algorithm and then under RED algorithm and we measure their fairness by using Jain's fairness index (FI) [26] shown in equation (2).

$$FI = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n * \sum_{i=1}^{n} x_i^2} \qquad (2)$$

Where n is the number of current flows and x_i is the average throughput of flow i. Value of FI is always no more than 1 and its larger value indicates better fairness performance. For example when all the competing flows in a network achieve definitely equivalent throughput, FI will be equal to 1.

Table 2 shows the simulation results. In both experiment, FDD-RED is fairer than RED algorithm and exhibits better performance in terms of flows throughput and bottleneck utilization. This better performance results from the decreased drop rate.

Table 2 The simulation result for FI

	Homogeneous RTT			Heterogeneous RTT		
	RTT	Original-RED	FDD-RED	RTT	Original-RED	FDD-RED
Flow0	100 ms	203056 bps	186856 bps	20 ms	295936 bps	306304 bps
Flow1	100 ms	244312 bps	195928 bps	40 ms	202840 bps	212128 bps
Flow2	100 ms	182320 bps	185704 bps	60 ms	161800 bps	189232 bps
Flow3	100 ms	196576 bps	219600 bps	80 ms	244528 bps	168496 bps
Flow4	100 ms	151216 bps	199976 bps	100 ms	89224 bps	124496 bps
Average flows' Throughput	-	195496 bps	197612 bps	-	198865 bps	200131 bps
Aggregated Throughput	-	977480 bps	988064 bps	-	994328 bps	1000656 bps
FI	-	0.967	0.996	-	0.888	0.901

4. Conclusion

This paper studied the Internet congestion control scheme from a novel perspective and proposed that an appropriate packet dropping function can direct the congestion control scheme to its targets. Based on this idea, it proposed a fair packet dropping discipline which distributes dropped packets uniformly among various sources considering their RTT. We implemented the proposed model in ns-2 environment by a modification over RED module. Simulation results showed that the proposed algorithm not only is fairer than RED algorithm but also outperforms RED in terms of throughput, utilization, number of dropped packets, queue size and even stability.

References

[1]. Wang, J. (2005), A Theoretical Study of Internet Congestion Control: Equilibrium and Dynamics, PhD thesis, university of Caltech.

[2]. Guo, S. Liao, X. Li, C. (2008), stability and hopf bifurcation analysis in a novel congestion control model with communication delay. *Nonlinear Analysis: Real World Applications*, 9, 1292-1309.

[3]. Analoui, M. Jamali, S. (2007), Congestion Control In the Internet: Inspiration from Balanced Food Chains In the Nature. *Springer Journal of Network and System Management*, 16, 1-10.

[4]. Jamali, S. Analoui, M. (2011), Globally Stable and High-Performance Internet Congestion Control through a Computational Inspiration from Nature, *Springer Journal Information science*, Science in China, 54, 6, 1251-1263.

[5]. Wang, X. Eun, D. (2007), Local and global stability of TCP-newReno/RED with many flows. *Computer Communications*, 30, 1091-1105.

[6]. Pei, L. Mu, X. Wang, R. Yang, J. (2011), Dynamics of the internet TCP-RED congestion control system. *Nonlinear Analysis: Real World Applications*, 12, 947-955.

[7]. Liu, F. Guan, Z. Wang, H. (2010), Controlling bifucations and chaos in TCP-UDP-RED. *Nonlinear Analysis: Real World Applications*, 11, 1491-1501.

[8]. Chiu, D. Jain, R. (1989), Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17, 1, 1–14.

[9]. Floyd, S. (1994), TCP and Explicit Congestion Notification. ACM *Computer Communication Review*, 24, 10-23.

[10]. Floyd, S. Jacobson, V. (1993), Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions Networking*, 1, 4, 397-413.

[11]. Chen, W. Yang, S. (2009), The mechanism of adapting RED parameters to TCP traffic. *Computer Communication*, 32, 1525-1530.

[12]. Abbasov, B. Korukoglu, S. (2009), Effective RED: an algorithm to improve RED's performance by reducing packet loss rate. *Journal of the Network and computer Applications*, 32, 703-709.

[13]. Zhou, K. Yeung, L. Li, K. (2006), Nonlinear RED: A simple yet efficient active queue management scheme. *Computer Networks*, 50, 3784-3794.

[14]. Xiong, N. Visilakos, V. Yang, T. Wang, Ch. (2010), A novel self-tuning feedback controller for active queue management supporting TCP flows. *Information Sciences*, 180, 2249-2263.

[15]. Cho, H. Fadali, S. Lee, H. (2008), Adaptive neural queue management for TCP networks. *Computers and Electrical Engineering*, 34, 447-469.

[16]. Athuraliya, S. Li, V. Low, S. Yin, Q. (2001), REM: active queue management. *IEEE Network*, 15, 3, 48–53.

[17]. Pan, R. Prabhakar, B. Psounis, K. (2000), CHOKe: a stateless active queue management scheme for approximating fair bandwidth allocation, *In Proceedings of IEEE INFOCOM2000*, Tel-Aviv, Israel, 26–30.

[18]. Kamra, A. Saran, H. Sen, S. Shorey, R. (2004), Fair adaptive bandwidth allocation: a rate control based active queue management discipline. *Computer Networks*, 44, 135-152.

[19]. Wu, H. Ren, F. Mu, D. Gong, X. (2009), An efficient and fair explicit congestion control protocol for high bandwidth-delay product networks. *Computer Communications*, 32, 1138-1147.

[20]. Sall, C. Alaoui, E. Doubabi, S. Warraki, E. (2011), Design of a robust digital controller for congestion control in Internet. Simulation Modeling Practice and Theory, 19, 301-313.

[21]. Barrera, I. Arce, G. Bohacek, S. (2011), Statistical approach for congestion control in gateway routers. Computer Networks, 55, 572-582.

[22]. Rosberg, Z. Matthews, J. Zukerman, M. (2010), A network rate management protocol with TCP congestion control and fairness for all. Computer Networks, 54, 1358-1374.

[23]. karn, P. Partridge, C. (1995), Improving round-trip time estimates in reliable transport protocols. ACM Computer Communication Review, 25, 1, 66-74.

[24]. Ott, T. Lakshman, T. Wong, L. (1999), SRED: stable RED, In Proceedings of IEEE INFOCOM, NY, 3, 1346-1355.

[25]. Zhang, H. Hollot, C. Towsley, D. Misra, V. (2003), A self-tuning structure for adaptation in TCP/AQM networks. ACM Sigmetrics Performance Evaluation Review, 31, 1, 302-303.

[26]. Jain, R. Chiu, D. Hawe, W. (1984), A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. DEC Research Report TR-301.

[27]. UCN/LBL/VINT, Network Simulator-NS2, <http://-mash.cs.berkelay.edu/ns>.