Digital Innovations & Contemporary Research in SCIENCE & ENGINEERING
Vol. 5 No. 2, June 2017

Article Progress Time Stamps

Article Type: Research Article
Manuscript Received: 12[rd] May, 2017
Review Type: Blind
Review/Acceptance Information Sent : 12[th] June, 2016
Final Acceptance:: 14[th] June, 2017
DOI Prefix: 10.22624

Article Citation Format
F.Y.E Mmue & K.J. Udoh (2017). Comparative Performance Evaluation of Congestion Control Algorithm in Harsh Environment..
Journal of Digital Innovations & Contemp Res. In Sc., Eng & Tech. Vol. 5, No. 2. Pp 137-150

Comparative Performance Evaluation of Congestion Control Algorithm in Harsh Environment.

F.Y.E Mmue
Department of Computer Science
River State University of Science & Technology
Port Harcourt, Nigeria
barifirste@gmail.com

K.J. Udoh
Department of Computer Science
Akwa Ibom State University
Ikot Akpaden, Nigeria
kenjumboudo@yahoo.com

ABSTRACT

Packet congestion is an important issue in the transmission control protocol (TCP) [1]. A Particular router algorithm related to congestion control is the queue management algorithm that manage the length of packet queues by dropping packets when appropriate queue management method as employed by the routers has been extensively studied by researchers and constitute vital issue in congestion contorl.Active queue management (AQM) as an advanced form of router queue management has been proposed as a router based mechanism for early detection of congestion in a network. This paper evaluates the performance of AQM using four popular algorithm: Random early Detection (RED), Flow Random Early Drop (FRED) Blue and stochastic fair blue and appling such baseline as size, and fairness among different traffic flow thoughout delay queue length or (whether different flows get their fair share and resource utilization (whether the link bandwidth is fully utilized). The overall merits of A QM for responsive flows is also explore

Keywords: RED, FRED, BLUE, SFB, throughput, fairness queue size, delay.

1. BACKGROUND TO THE STUDY

Lack of attention to the dynamics of packet forwarding can result in severe service degradation or internet meltdown [2]. A usuful effort made to take care of this lack of attention may result to the development of congestion avoidance mechanisms that may be required in TCP implementation [3] .These mechanisms operate in the host to cause TCP connections to "back off" during congestion. This is TCP flows responsiveness to congestion signals. Primarly it is these TCP congestion avoidance algorithms that prevent the congestion collapse of todays internet [4]

It has becomes clear that the TCP congestion avoidance mechanism while necessary and powerful, are not sufficient to provide good service in all circumstance [5]. some mechanisms are needed in the routers to complement that end point congestion avoidance mechanisms . Basically, two classes of router algorithms that relate to congestion control exixt. Queue management and sheduling algorithms, while queue management algorithms approximately manage the length of packet queues by dropping packets when necessary, scheduling algorithms determine which packet to send next and are used primarily to manage the allocation of bandwidth among flows through, these two router mechanisms are closely related, they address different performances issues[6].

Active queue management is an advanced form of router queue management that can be used with a wide variety of scheduling algorithms, can be implemented relatively efficiently, and will provide significant inherent performance Improvement [7].

2. QUEUE MANAGEMENT

Researchers and the IETF proposed active queue management (AQM) as a mechanisms for detecting congestion inside the network and strongly recommended the deployment of AQM in routers as a measure to preserve and improved performance. AQM algorithms run on routers and detect incipient congestion by typically monitoring the instantaneors or average queue size [8]. When the average queue size exceeds a certain threshhold but is still less than the capacity of the queue, AQM algorithms infer congestion on the link and notify the end systems to back off by proactively dropping some of the packets arriving at a router . Alternatively, instead of dropping a packet, AQM algorithms can also set a specific bit in the header of thet packet and forward that packet toward the receiver after congestion has been inferred upon receieving that packet the receiver in turns set another bit in its next ***ACK*** when the sender receives this **ACK** it reduce it transmission rate as if its packet were lost [9].

2.1 The Need For Active Queue Management .

Traditionally, Technique used to manage router queue length sets maximum length in terms of packet size for each queue, packets are accepted for the queue untill the maximum length is attained, then drop subsequent incoming packets untill the queue decreases . This technique is called "tail drop" since the packet that arrived most recently , that is the one on the tail of the queue is dropped when the queue is full [13]. This technique has two important setbacks [10]:

a. Lock out: A single connection or a few flows may monopolize queue space, preventing other connections from gaining room in the queue. A phenomenon that occur often as the result of synchronization or other timing effects [11].
b. The tail drop discipline: allows queue to maintain a full or almost full status for long periods of time since packet drop only when the queue has become full it is important to reduce the steady-state queue size, and this is perhaps queue management's most important goal.

Besides tail drop, two alternative queue disciplines that can be applied when the queue becomes full [13], "random drop on full" or drop front on full" under the random drop on full discipline, a router drops a randomly selected packet from the queue when the queue is full and a new packet arrives. Under the "drop front on full" discipline, the router drops the packet at the front of the queue when the queue is full and a new packet arrive [14]. Both of these queue discipline solve the lock- out problem but neither solve the full queue problem.

2.2 Goals of Active Queue Management

AQM was designed with primary and secondary goals to achieve in packet transmission. Controlling average queuing delay, while the secondary goals, include.

- Improving fairness for example by reducing biases against bursty low bandwidth flows
- Reducing unnecessary packet drops.
- Reducing global synchronization especially for environments with small-scale statistical multiplexing
- Accommodating transient congestion that last less than a round- trip time[18]. Vitally summarize, an AQM mechanism can provide the following advantages for responsive flow :

❖ Reduce number of packets dropped in routers [11].
❖ Provide lower-delay interactive services .
❖ Avoid lock-out behaviour [15, 16, 17].

The primary purpose of a queue in internet protocol (IP) router is to smooth out bursty arrivals so that the network utilization can be high. Disappointingly queue add delay and cause jilter in heavy traffic cloud communication environment, Delay is the enemy to real time network transmission and communication. Jilter is turned into delay at the receivers playout buffer, and inadvertently causing data packets congestion in traffic network [19].

2.3 Queue Management Algorithm

2.3.1 **RED** [1] Was designed with the objectives to

(1) Minimize packet loss and queuing delay
(2) Avoid global synchronization of sources
(3) Maintain high link utilization and
(4) Remove biases against bursty source. The basic ideal behind RED queue management is to delect incipient congestion early and to convey congestion notification to the end-host, allowing them to reduce their transmission rates before queue in the network overflow and packets are dropped.

To do this, RED maintain an exponentially weighted moving average (EWMA) of the queue length which it uses to delect congestion. When the average queue length exceed a minimum threshold ($^{min}{}_{h}$), packets are randomly dropped or marked with an explicit congestion notification (ECN) bit [20].When the average queue length exceeds a maximum threshold ($^{max}{}_{h}$) all packets are marked or dropped.

While RED is certainly an improvement over traditional drop tail queue, it has several shortcomings: One of the fundamental problems with RED is that they rely on queue length as an estmator of congestion, while the presence of a persistent queue indicates congestion, its length gives very little information as to the severity of congestion that is, the number of competing connections sharing the link. In a busy period, a single source transmitting at a rate greater than the bottleneck link capacity can cause a queue to build up just as easily as a large number of sources can.

Since the RED algorithm relies on queue lengths, it has an inherent problem in determning the severity of congestion. As a result, RED requires a wide range of parameters to operate correctly under different congestion scenarios. While RED can achieve an ideal operating point, it can only do so when it has a sufficient amount of buffer space and it is correctly parameterized.

RED represents a class of Queue managemet mechanisms that does not keep the state of each flow, that is, they put the data from all the flows into one queue, and focus on their overall performance. It is that which originate the problem cause by non-responsive flows. To deal with that, a few congestion control algorithms have tried to separate different kind of data flows for example, fair queue [21], weighted fair queue etc. But their perflow- schedulling philosophy is different with that of RED which we will not discuss here.

For each packet arrival calculate the new average size q_{avg} if min h<q_{avg}<Max $_h$ calculate probability P_a:with probability p_a:Mark/drop the arriving packet else if max $_h$<q_{avg} drop the arriving packet

[General RED algorithm [22]

Vaviables	Parameters
q_{avg};Average queue size	*Min_h: Minimum Threshold for Queue*
p_a:packet marking or dropping probability	*Max_h: Maximum Threshold for Queue*

2.3.2 FRED (Flow Random Early Drop) (FRED)

[2] Is a modified version of RED, which uses per-active-flow accounting to make different dropping decisions for connections with different bandwidth useage. FRED only keeps track of flows that have packets in the buffer, thus the cost of FRED is proportional to the buffer size and independent of the total flow number (including the short- lived and idle flow). FRED can achieve the benefits of per-flow queuing and round robin scheduling with substaintially less complexity. Some other interesting features of FRED include;

(1) penalizing non-adaptive flows by imposing a maximum number of buffered packet and surpassing their share to average per- flow buffer usage.
(2) Protecting fragile flows by deterministically accepting flow from low bandwidth connections.
(3) Providing fair sharing for larger numbers of flows by using “two packet buffer” when buffer is used up.
(4) Fixing several imperfections of RED by calculating average queue length at both packet arrival and departure (which also causes more overhead).

Two parameters are introduced into FRED Min_h and Max_h, which are minimum and maximum numbers of packets that each flow is allow to buffer. In order to track the average per-active flow buffer usage, FRED uses a global variable avg_{cq} to estimate it. It maintains the number of active flows and for each of them, FRED maintains a count of buffer packets q_{len} and a count of time when the flow is not responsive (q_{len}> max_h,) FRED will penalize flows with high strike values. FRED processes arriving packets ucing the following algorithm.

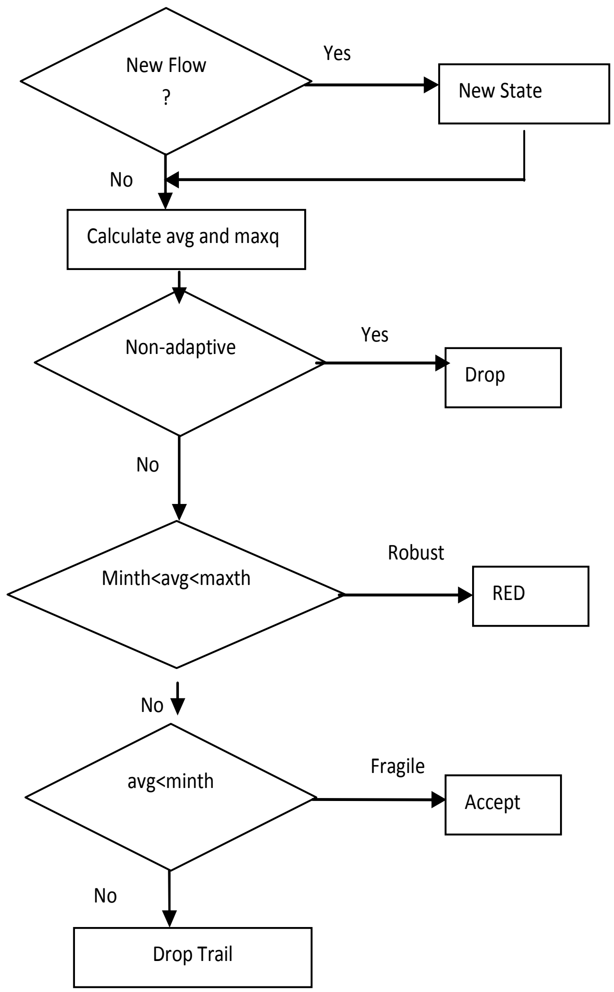


Fig 1: FRED processing arriving packet

BLUE is an active queue management algorithm to manage congestion control by packet loss and link utilization history instead of queue occupancy. BLUE maintains a single probabiting P_a, to mark or drop) packets. If the queue is continually dropping packets due to buffer overflow, BLUE increase P_a, thus increasing the rate at which it sends back congestion notification or dropping packets conversely, if the queue becomes empty or if the link is idle, BLUE decrease it marking probabity. This effectively allow BLUE to " learn" the correct rate it needs to send back congestion notification or dropping packet.

The typical parameters of BLUE are *d1*, *d2* and *freeze- time. d1* determines the amount by which p_a is increased when the queue overflows, while *d2* determines the amount by which p_a is decreased when the link is idle. *Freeze- time* is an important parameter that determines the minimum time interval between two successive updates of p_a. This allows the changes in the marking probability to take effect before this value is updated again. Based on those parameters, the basic BLUE algorithm can be summarized as follows:

Upon link idle event if (now-last update) > freeze time $P_a = P_a - d2$ Last- update= now	Upon packet loss event if (now – last update) > freeze – time) $P_a = p_a + d1$ Last- update= now

2.3.4 SFB Based on BLUE, stochastic fair BLUE (SFB)

SFB Based on BLUE, stochastic fair BLUE (SFB) is a novel technique for protecting TCP flows against non-responsive flows. SFB is a FIFO queuing algorithm that identifies and rate–limits non-responsive flows on accounting mechanisms similar to those used with BLUE. SFB maintain accounting *bins*. The *bins* are organized in *L* level with *N bins* in each level. In addition, SFB maintains *L* independent harsh functions each associated with one level of the accounting *bins* are used to keep track of queue occupancy statistic of packets belonging to a particular *bins.* As a packet arrives at the queue, it is hashed into one of the *bins* in each of the *L* levels. If the number of packets mapped to a *bin* goes above a certain threshold. (ie the size of the *bin*).

The packet dropping probability P_a for that *bin* is increased. If the number of packets in that *bin* drop to zero, P_a is decreased. The observation is that a non- responsive flow quickly drives P_a to 1 in all of the *L bins* it is harshed into. Responsive flow may share one or two *bins* with non-responsive flows, however unless the number of non-responsive flow is extremely large compared to the number of *bins* a responsive flow is likely to harshed into at least one *bin* that is not polluted with non-responsive flows and thus has a normal value. The decision to mark a packet is based on P_{min} the minimum P_a value of all *bins* to which the flow is mapped into. If P_{min} is 1, the packet is identified as belonging to a non- responsive flow and is then rate limited.

```
B (1)(n) LXN array ofbins(L levels N bins per level
calculate hash function value ho, hi………hl-1
enque () update bins at each level
For i= 0 to L-1
If B(hi)(HI). QLENS > BIN- SIZE
B (1)(hi) Pm +=delte
Drop packet
Close if (B(0)(H0).Pm….B(L)(HI) PM )I
P min = min (B(0)(H0). Pm……B(L) (HL). PM)
If (Pmin ==1)
Rate limit ()
Else
Mark/ drop with probability Pmin
```

The typical parameters of SFB algorithm are q_{len}, *Bin-size, d1 d2 freeze-time, NL, Boxtime, H-Interval. Bin-size* is the buffer space of each bin for each *bin.* Q_{len} is the actual queue length of each bin, *d1 , d2,* and *freez-time* have the same meaning as that in BLUE. Beside, *N* and *L* are related to the size of the accounting *bins*, for the *bins* are organized in *L* level with *N bins* in each level. Box time is used by penalty box of SFB as a time interval used to control how much bandwidth those non-responsive flow could take from bottleneck links. *H* interval is the time interval used to change harshing function.

3. PERFORMANCE METRICS

The performance metrics used in this paper are Delay, Packet Loss, Queue Length or Queue size and throughput

3.1 Delay

Delay is the time elapsed while a packet travel from one point (e.g source premise or Network Ingress) to another (e.g destination premise or Network degress). The larger, the value of delay, the more difficult.
It is transport layer protocols to maintain high bandwidths. This characteristic can be specified in a number of different ways, including average delay, variance of delay (jitter), and delay bound. In this paper, we calculated end to end delay.

3.2 Packet Loss

Packets can be lost in a network because they may be dropped when queue in the network node overflows. The amount of packet loss during the steady state is another important property of a congestion control scheme. The larger the value of packet loss, the more difficult it is for transport-layer protocols to maintain high bandwidths, the sensitivity to loss of individual packets, as well as to frequency and patterns of loss among longer packet sequence is strongly dependent on the application itself. This characteristic can be specified in a number of different ways, including loss rate, loss patterns, loss free seconds, and conditional loss probability. In this paper we considered that packet loss would occur only due to the dropping of the packets. There is no loss due to other means.

3.3 Queue Length

A queuing system in network can be described as packet arriving for service, waiting for service, if it is not immediate and if having waited for service, leaving the system after being served. This queue length is very important characteristics to determine how well the active queue management of the congestion control algorithm has been working.

3.4 Throughput

It is the primary performance measure characteristic and most widely used. It measures how soon the receiver is able to get a certain amount of data send by the sender. This is determine as the ratio of the total data received to the end to end delay. Throughput is an important factor which directly impacts the Network Performance.

4. SIMULATION AND COMPARISON

In this section, we will compare the performances of RED, FRED, BLUE and SFB. We use RED and Tail drop as the evaluation baseline. Our simulation configuration is based on *ns-2*. Both RED and FRED have implementation for *ns*-2, BLUE and SFB are originally implemented in a previous version of *ns*, *ns-1* and re-implemented in *ns-2*. In our simulation, ECN support is disabled and "marking a packet" means "dropping a packet" [23].

4.1 Simulation Settings

It is known that different Algorithms have different preferences or assumptions for the network configuration and traffic pattern, one of the basic challenges in designing our simulation, is to select a typical set of network topology and parameters such as link bandwidth, RTT, and gateway buffer size, as well as load parameters such as the numbers of TCP and UDP flow, packet size, TCP window size, traffic patterns,

Digital Innovations & Contemporary Research in SCIENCE & ENGINEERING
Vol. 5 No. 2, June 2017

as the platform for evaluation. In this regard, we make extracts from reading related works, and combine the key characteristics from their simulations.

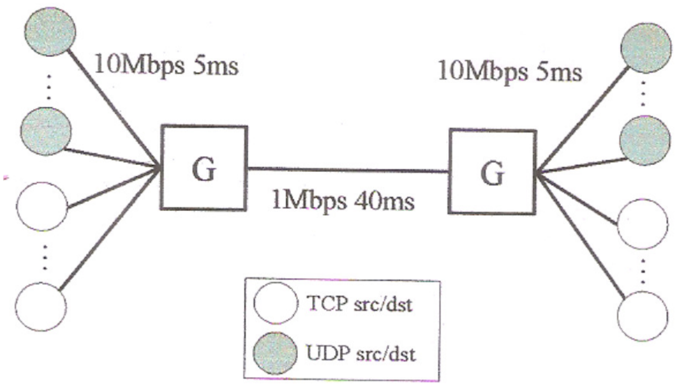


Fig. 2: Simulation Settings.

Figure 2 above is a classic dumb-bell configuration network topology. It is a typical setting that different types of traffic share a bottleneck router. TCP (FTP application in particular), and UDP flows (CBR application in particular) are chosen as typical traffic patterns. In this simulation, we use 10 TCP flows and 1 UDP flow. The bottleneck link in this setting is the link between two gateways. We set TCP window size as 100 packets, and the router queue buffer size in the simulation as 300 packets (the packet size for both TCP and UDP are 1000 bytes). For RED, we set the values for *Min* and *Max*, as 20% and 80% queue buffer size.

4.2 Comparative Analysis

Figure 3 and figure 4 below show the main result of the simulation. The sum of the throughput values for all TCP and UDP flows are not shown here. For all the simulations, the total throughput are reasonably high (about 91.05 percent of the available bandwidth), showing that all the algorithms under investigation provide high link utilization. Figure 3.1 shows the UDP throughput and queue length under simulation. RED and BLUE do not work well under high UDP sending rate. When UDP sending rate is above the bottleneck link bandwidth, UDP flow quickly dominates the transmission on the bottleneck link and TCP flows only share the remaining bandwidth. On the other hand, FRED and SFB properly penalize UDP flow. Figure 3.2 Illustrates the size of queue buffer occupied by UDP flow. It is our observation that buffer usage seems to be a good indicator of link bandwidth utilization. Similar to figure 3.1 RED and BLUE are similar in permissive to non-responsive flows, BLUE uses much less space, FRED and SFB are also the fairest.

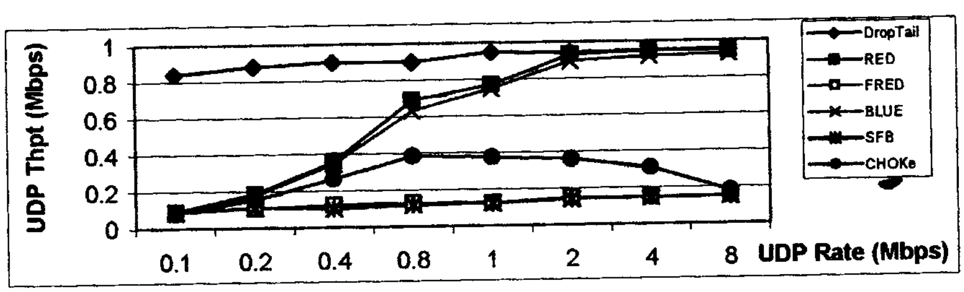


Figure 3(a): UDP flow throughput

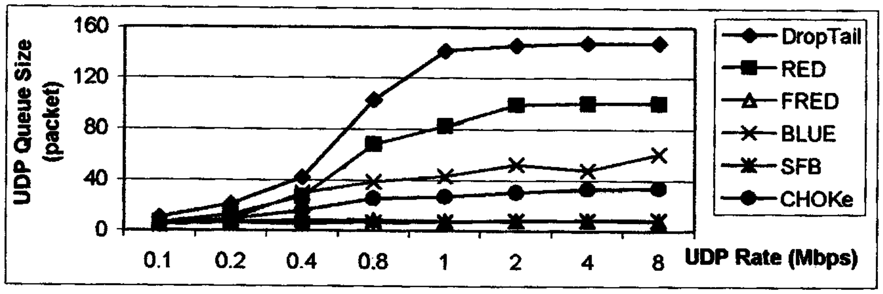
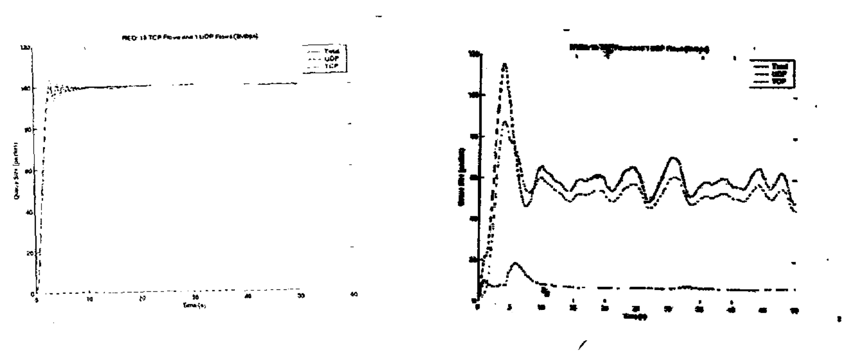


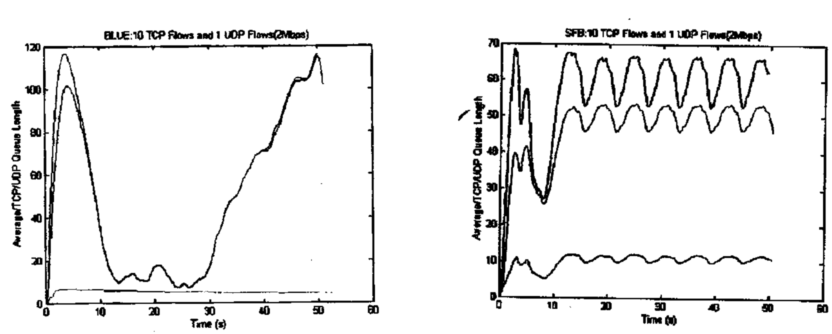
Figure 43(b): UDP flow queue size

Figure 4 illustrates the average queue size for UDP and TCP flows as well as the mean total buffer usage. The difference of the algorithms is clearly shown in the buffer usage plots. We observe that FRED and SFB effectively penalize UDP flow and allow TCP flows to achieve a higher throughput.

We interestingly notice the difference among the total queue sizes. RED, although begins to provide congestion notification when the queue size reaches min_{th}, it only affects TCP flows while UDP keep the same sending rate, which drives the total queue size to max_{th}, quickly, after which all the incoming packets will be dropped, and the total queue size will be kept at max_{th}. FRED, BLUE and SFB are not directly affected by min_{th}, and max_{th}, sendings, so their total queue sizes have no relation with these parameters in figure 4.



(a) (b)



(c) (d)

Figure 4 queue size in different algorithms.

Digital Innovations
& Contemporary Research in
SCIENCE & ENGINEERING
Vol. 5 No. 2, June 2017

Figure 5 plots the actual response time for each achieved in RED, FRED, BLUE and SFB. It is observed that minimum delay occurred in each algorithm is the same. We therefore conclude within reasonable limit that each algorithm would get the same response time provided congestion has been observed because queuing delay would be same for each algorithm if there is no congestion in Network.

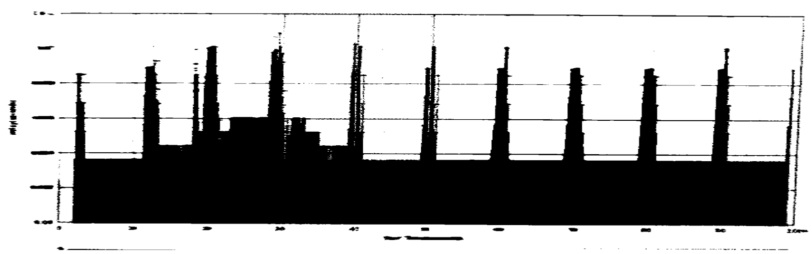


Figure 5 (a) showing the RED Algorithms

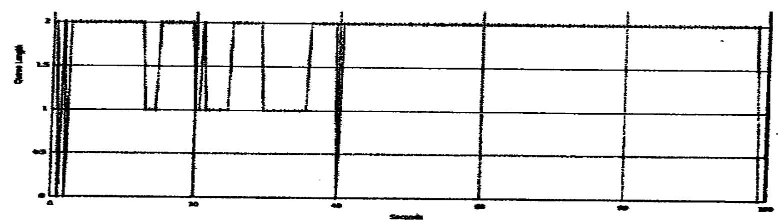


Figure 5 (b) RED queue length

5 ALGORITHM CHARACTERISTICS

5.1 FRED

FRED algorithm focuses on the management of per-flow queue length. The parameter q_{len} is compare with min_{th} and max_{th} and used as a traffic classifier. Fragile flows are those whose $q_{len}<min_{th}$ Robust flows are those whose $min_{th}<q_{len}<max_{th}$, and non-responsive flows are those whose q_{len} was once larger than max_{th}. The min_{th} is set to 2 or 4, but can adapt to average queue length when there are only few robust flows as found in a LAN environment with small RTT and larger buffer size. FRED is very robust in identifying different kind of traffic and providing adaptive flows. Figure 4b shows the queue length of UDP flow and the sum of 10 TCP flows. The UDP queue length was effectively limited to 10 packets, which is approximately the average queue length. The single UDP flow is isolated and penalized without limiting the adaptive TCP flows.

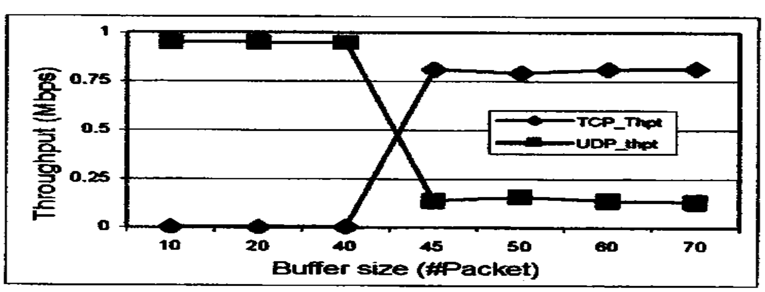


Figure 6 Impact of buffer size to FRED fairness.

Figure 6 shows the impact of buffer size to FRED algorithm. It is clear that FRED works well only when the buffer is larger (larger than 45 packets in this case) enough to hold min_q, packets for each active flow. When the average queue length is larger than max_{th}, FRED degrade into drop tail and cannot preserve fairness.
The fairness of FRED is also illustrated in table 1. The share of UDP flows and TCP flows do not change much as the bottleneck bandwidth increases from 0.5 MbPs to MbPs. After the bandwidth of backbone link is large enough, the UDP flow gets its full share and TCP flows begin to compete with each other.

Table 1 showing the bottleneck bandwidth to FRED link utilization

Botterieck Bandwidth (MbPs)	0.5	1	2	4	8	10	20
TCP Thpt (mbps)	0.42	0.80	1.61	3.14	5.73	7.41	13.94
UDP Thpt (mbps)	0.08	0.16	0.29	0.66	1.82	1.86	1.96
TCP share percent	84%	81%	81%	78%	73%	74%	71%
UDP share percent	13%	15%	15%	17%	24%	10%	9%
TCP share: UDP share	6.93	5.33	5.80	4.60	3.13	3.90	7.77%

The FRED algorithm has an O(N) space requirement (N=buffer size), which was one of the major merit compared with per-flow queuing mechanisms (e.g fair queuing). But with current memory cost, its space requirement is not an important factor. The most significant is the computational resources for each packet. For each arriving packet, FRED need to group the packet into a flow, update information and compute average queue length (also done when a packet is leaving), and decide whether to accept or drop the packet. Summarily, FRED achieves fairness and high link utilization by sharing the buffer size among active flows. It is also easy to configure, and adapt itself to preserve performance under different network environments (different bandwidth, buffer size, flow number), and traffic patterns (non-adaptive flows, robust adaptive flows and fragile flows).

5.2 BLUE

The most significant effect of using BLUE is that congestion control can be performed with a minimal amount of buffer size. Other algorithms such as RED requires a large buffer size to attain the same goal [24]. Figure 7 shows the average and actual queue length of the bottleneck link in our simulation based on the following settings: 50 TCP flows with TCP window size 300 (KB), a bottleneck link queue size 300 (KB). As we observe from figure 7, the actual queue length in the bottleneck is always kept quite small (about 100KB), while the actual capacity is as large as 300KB. Only about 1/3 buffer space is used to achieve 0.94 Mbps bandwidth by TCP flows. The other 2/3 buffer space allows room for a burst of packets, removing biases against bursty sources.

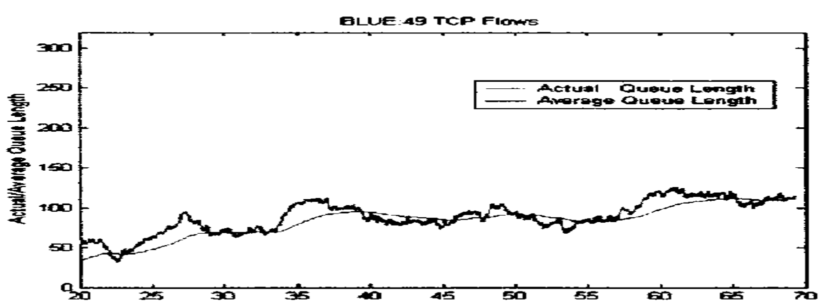


Figure 7 BLUE queue length for TCP flows

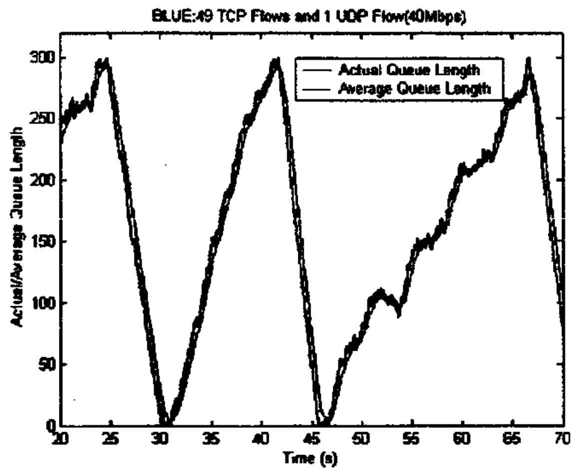


Figure 8: BLUE queue length for TCP and UDP flows.

However, simulation get worse when non responsive flows appear. Figure 8 shows the actual and average queue length of the bottleneck link in our simulation when a 40 mbps UDP flow joins those 49 TCP flows. Here the total throughput (TCP and UDP) achieved is 0.95 mbps, among which 0.01 mbps bandwidth is taken by 49 TCP flows while the UDP flows throughput is as high as 0.94 mbps.

The slow fluctuation of the bottleneck queue length shown in figure 8 is reasonable. At t=40second, the buffer of the bottleneck link is overflowed, so P_a increases to 1 quickly. Hence, all the incoming packets will be dropped and in the nearwhile packets in the queue are dropped. Since P_a does not change until the link is idle, the queue length shrinks to zero gradually. The queue length at t=48s is O. After that, the P_a is decreased by BLUE. Then incoming packets could get a chance to enter queue, and the actual queue length will gradually increase from zero accordingly.

5.3 SFB

Basic SFB characteristics

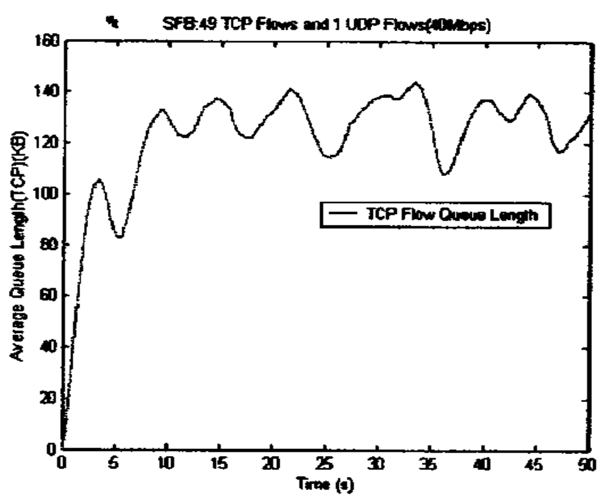


Figure 9: SFB queue length for TCP flows

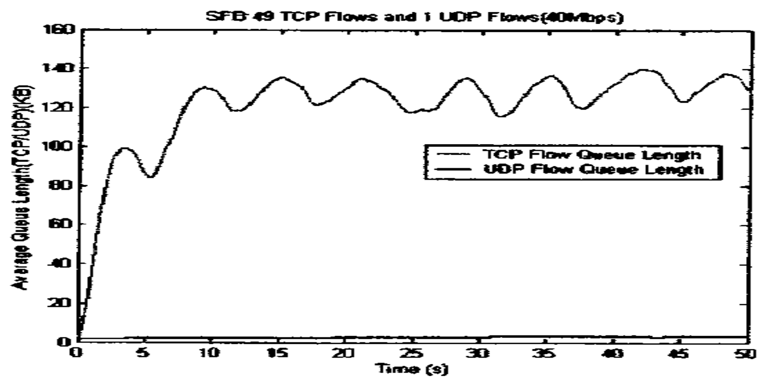


Figure 10: SFB queue length for TCP and UDP flows.

Although SFB is able to accurately identified and rate-limit a single non-responsive flow without affecting the performance of any of the individual TCP flows, as the number of non-responsive flows increases, the number of the parameters *bin* which become "polluted" or have P_a values of 1 increases. As a result of this, the probability that a responsive flow becomes misclassified increases. To overcome this, the moving harsh functions was implemented, that is, by changing the harsh function, responsive TCP flows that happen to map into polluted *bins* will potentially be remapped into at least one unpolluted *bin*. However, in this case, non-responsive flows can temporaily consume more bandwidth than their fair share. To remedy this, two set of harsh functions are used.

Figure 9 shows the TCP flow queue length of the bottleneck link when there is no UDP flow. Here, the 49 TCP flows throughput is 0.93 mbps, while figure 10 shows the case when a 40mbps UDP flow joins. In this case, the UDP flow's throughput is only 0.027mbps while the 49 TCP flows throughput is still quite large which consumes 0.926 mbps bandwidth of the bottleneck link. The UDP queue length is kept very small (about 4.4KB) all the time, so that we could see that due to the effect of SFB's double buffered moving harsh, those non-responsive flows are effectively detected and rate-limited by SFB.

6. CONCLUSION

In this paper, comparison had been made on four congestion control algorithms (RED, FRED, BLUE and SFB) based on the results obtained from the simulation made. Algorithm characteristics of the algorithms compared are also presented to give insight into our observations. We still find it difficult to conclude which algorithm is better in all aspects than another, especially, when we consider the deployment complexity. Summarily, we present the major trends of the comparative evaluation results in a table below.

Table 2 summary of result obtained

Algorithm	Configuration Complexity	Space Requirement	Per-Flow State Information	Fairness	Link Utilization
RED	Difficult	Large	No	Unfair	Good
FRED	Adaptive (Easy)	Small	Yes	Fair	Good
Blue	Easy	Small	No	Unfair	Good
SFB	Difficult	Large	No	Fair	Good

REFERENCES

1. Stevens W; "TCP Slow Start, Congestion Avoidance Fast Retransmit And Fast Recovery Algorithms" RFC 2001, Jan 1997
2. V. Jackson, "Congestion Avoidance And Control"; ACM Sigcomm 88, August 1988
3. Wagle J. "Congestion Control In IP/TCP"; RFC896, January 1984.
4. Gaynor M; "Proactive Packet Dropping Methods For TCP Gateways" October 1996.
5. T. Blaskar Reddy, Ali Ahammed and Reshma Banu, "Performance Comparison Of Active Queue Management Technique" In IJCSNS Vol.9 No.2, February 2009.
6. Shenker, S, Parthridge, C, and R Guerin, "Specification of Guranteed Quality of Service", 1996 Work In Progress.
7. T.V. Lakshman, Arnie Neidhardt, Tennis Ott; "The Drop Front Strategy In TCP Over ATM And Its Interworking With Other Control Features" In FOCOM 96. MA 28.1
8. W. Willinger, M.S Taggu, R. Sherman, D.V. Wilson. "Self – Similarity Through High Variability Statistical Analysis Of Ethernet LAN Traffic At The Source Load", ACM SIGCOMM 95, August 1995
9. Floyd, S; "Connection With Multiple Congested Gateway In Packet Switched Networks", Part 1: One Way Traffic Computer Communication Review, Vol 21 No.5, October 1991
10. G.F. Ali Ahammed, Reshma Banu "Analyzing The Performance Of Active Queue Management Algorithm" International Journal Of Computer Networks And Communication (2010).
11. Demers, A, Kesh, S, and Shenker S, "Analysis and Simulation of a Fair Queueing Research and Experience Vol. 9, 1990.
12. Floyd, S. And Jacobson , V; "Link Sharing And Resource Management Models For Packet Networks"; IEEE/ACM Transactions On Networking Vol. 3, No. 4, August 1995
13. S. Paul P. Gopia and Singh. "Performance Measure of Drop Tail and RED"; Proceeding of ICEP, 2010
14. Monley, R. "Reducing Packet Loss and Latency by Active Queue Management Algorithm" CSCI 3902. Seminar 1, UMM CSCI Twiki; (2003).
15. Aflualiya S. Lapsley D.E Low S.H Randomly Early Marking For Internet Congestion Control" IEEE/ACM Transaction On Networking Vol.15, No.3 (2001).
16. D. Lin and R. Morris
"Random Early Dectation" ACM SIGCOMM Computer Communication Review, Vol. 27 No.4
17. B. Braden et al Recommendation on "Queue Management and Congestion Avoidance in Internet", RFC 2309; (1998)
18. B. Braden, Estrin B. Clark, D, Crow J, Decring, Dfloyd, S, Jacobson, et al Recommendations on "Queue Management And Congestion Avoidance In The Internet", Internet Draft. (1998).
19. Ningning HU, LIUREN, Jichuan Chang, "Evaluation of Queue Management Algorithms" Class Project Report for 15-744 Computer Networks
20. T. Bonald et al "Analytic Evaluation of RED Performance" in Proceeding of IEEE INFOCOM, 2000
21. J. Zhino and R Govindin "Understanding Packet Delivery Performance In Dense Wireless Sensor Networks" In Proceeding Of ACM Subsys. 2003
22. Feng W., Kandlur D, Saha D, Shin K. "A self – Configuring RED Gateway"; In Proceeding of IEEE/INFOCOM. (1999)
23. S. Mascolo, C. Casette, M. Gerla S.S Lee, And M Sanadidi; "TCP Westwood, Congestion Control With Fast Recovery" Technical Report 2000
24. M. Ahman, V. Paxson, and W. Stevens "TCP Congestion Control" RFC 2581; April 1999.