IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-3, Issue-9 E-ISSN: 2347-2693

Performance evaluation of Unresponsive flow Management on Server for better load rebalancing in Internet

Vijith C[1*] and M. Azath [2]

[1*,2]*Department of Computer Science and Engineering, Met's School of engineering, Kerala, India*

www.ijcseonline.org

Received: Aug/22/2015 Revised: Aug/30/2015 Accepted: Sep/24/2015 Published: Sep/30/2015

***Abstract*—** Active Queue Management has specific role in computer networks. In this paper, we investigate the chances of Active Queue Management at serve side. The proposed system reduces packet loss ratio at server side to provide better load balancing at internal buffer levels. The system distinguishes responsive flows from unresponsive flows in a congested Hyper Text Transfer Protocol traffic, dynamically manages them and provides better transfer speed and maximum throughput in network.

***Keywords*—** AQM, CHOKeR, Server, Proxy server, Load balancing

I. INTRODUCTION

The Internet is a decentralized global system of interconnected computer networks that use the standard Internet protocol suite (TCP/IP) to link devices worldwide and carries extensive large amount of information in fraction of seconds among them. When the level of network traffic nears, reaches or exceeds the design maximum, the network is said to be congested. Load rebalancing is a concept come in to play in the network, which is experienced with traffic congestions. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any one of the resources [1].

Unresponsive flows are main constituent of congestions occurred in the network buffers. Unresponsive flows are the information that are not responded yet and may be generated from same or different machines. Those flows make the buffer full and it will lead to overflow situation. This is a crucial scenario where the important flows get refused to join in queue due to overridden non responsive flows. It leads to congestion in network and makes scarcity in resource [21].

Active queue management (AQM) is the intelligent drop of network packets inside a buffer associated with a network interface controller (NIC), when that buffer becomes full or gets close to becoming full, often with the larger goal of reducing network congestion [2][7]. This task is performed by the network scheduler, which for this purpose uses various algorithms such as random early detection (RED), Explicit Congestion Notification (ECN), or controlled delay (CoDel) [3][5][6]. An Active Queue Management system is used to control the length of a queue so that it does not run full, adding its maximum (usually bloated) delay under load. Such management also enables TCP to do its job of sharing links properly, without which it cannot function as intended [8].

There are two kinds of flow management mechanisms that try to achieve the resource fair sharing: scheduling scheme and queue management scheme. Scheduling schemes have generally too much complexity and low scalability to large number of flows. If we plan to provide fair utilization on server, queue management scheme could be a better choice. Queue management scheme not only has less complexity, but also approximates fairness better [7].

CHOKe, Choose and Keep for Responsive flows, Choose and Kill for Unresponsive flows, an Active Queue Management method, is stateless, controls misbehaving flows with a minimum overhead. It is simple to implement, based on queue length and differentially penalizes unresponsive flows using the information of each flow [9][10]. CHOKeW uses "matched drops" created by CHOKe to control the bandwidth allocation, but excludes the RED module for bandwidth differentiation and TCP protection which is important for implementing Quality of Services (QoS)[17]. CHOKeR is advancement to CHOKeW algorithm which overrides the problems of bandwidth differentiation in multiple priority levels and poor performance on bursty traffic in large congested network which are experienced by CHOKeW. CHOKeR does not maintain per flow state information and it uses MISD (Multi step Increase and Single step Decrease) model for congestion avoidance [18].

II. PROPOSED SYSTEM

The system is proposed to deal with unresponsive flows, so that the network performance could be enhanced by efficient congestion control and load rebalancing. Congestion control is arisen by different types of flows through a network in any type of nodes. Among the flows, the main contribution for congestion is shown by non-responsive flows generated by end to end systems. So to remove the fluctuations generated in the network buffer

by the unresponsive flows and as well as normal flows, the proposed system shows an efficient method which uses queue dimensions and other performance metrics for tuning the communication. Proposed system classifies the packets into responsive and unresponsive flows. The dimensions for the classifications are fall in flow information. The dominant unresponsive flows are dropped pre-emptively and the valid responsive flows are limited to enter into buffer without making over flow and underflow conditions. The membership of packets in the active queue is estimated using an AQM methodology called CHOKeR [18]. On the results of applying the queue management algorithms the packets are distinguished and the network is classified.

The System is implemented at server side as a proxy server. The server runs a server application which manipulates http requests and provides http responses. The server is created in open source platform and proxy server handles both responsive and unresponsive flows. In computer networks, a proxy server is a server (a computer system or an application) that acts as an intermediary for requests from clients seeking resources from other servers.

III. MODULAR DESCRIPTION

The system contains three modules for implementation. They are classified as Unresponsive flow management, Active Queue Management and Traffic characterization based on their operations.

Unresponsive flow management

It deals with recognition and management of flows. This module characterizes the entire flow into responsive and unresponsive flows. The flow information is extracted and they are compared based on the present structure. If the incoming flow has same information as in stored flow which is already in the buffer, then the incoming flow is treated as unresponsive and system decides it is carrying duplicate information. So the unresponsive flows are pre-emptively discarded. And the converse, the responsive flows which are well-behaving and carrying unique information is passed to next module for the proper insertion of the flow in the buffer.

Active queue management

It is the second module of the system which deals with internal buffers. It uses CHOKeR methodology for queue management [18]. On the arrival of a responsive flow, the average buffer length is calculated exponentially and it is compared with thresholds. According to the rules defined by the algorithm based on thresholds, the packet may or may not be allowed to enter in the FIFO buffer.

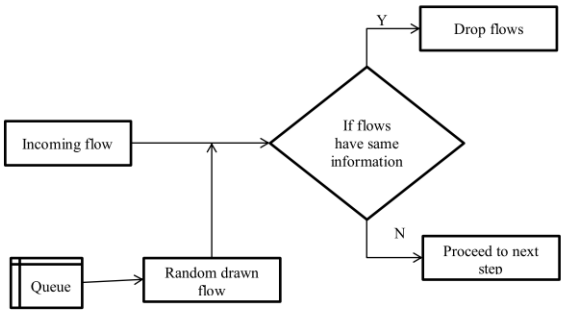

Fig. 1 Unresponsive flow management

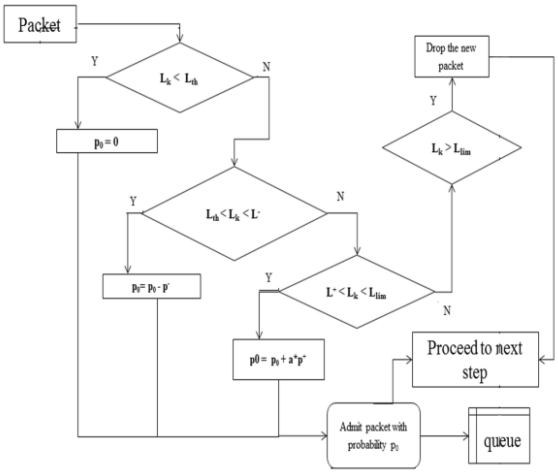

Fig. 2 Active queue management

Traffic characterization

Traffic characterization is nothing but the final stage of the system which takes buffer as input and provides service as output. It does mainly two things, packet processing from the buffer in FIFO order and same time logging the historical data. The packet from head is drawn and it is forwarded to proper destination which can be obtained from the flow information encapsulated with buffer. This section realizes actual port forwarding scheme of the proxy server.

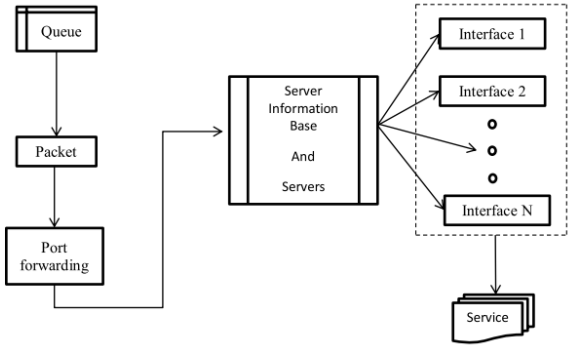

Fig. 3 Traffic characterization

IV. WORKING

The client generates request for particular web service from his machine. When the server receives the request on its ports the system will be invoked automatically. The system checks the incoming flows and filters the packets into responsive and unresponsive flow classes. The flow classification is mainly done by comparing the flow ids compromising source address, destination address and ports. Then the unresponsive flows are pre-emptively discarded and the server traffic information base is updated about the procedures. The responsive flows are forwarded to AQM module which uses CHOKeR algorithm for adaptive queue management. The CHOKeR works on the responsive flows and it drops or insert packets into the queue based on its criteria and methods [18]. After the queuing and discarding of packets, the control is passed to server applications at higher levels which processes service requested by the client that is received from the queue in order.

Since the system is a proxy server, the main objective of algorithm to retrieve packets form heterogeneous clients and forward them based on their destination address. The general algorithm is represented in Fig. 5.

The probability to find eligibility of a packet in buffer is found out using CHOKeR method. The algorithm uses mainly three probabilities $pmin$, $pmax$ and $p0$. The first two probabilities are used for providing MISD (Multi step Increase, Single step Decrease) scheme in server's queue management and the latter is used for drawing multiple random packets from the same buffer [18]. The mathematical model is given below.

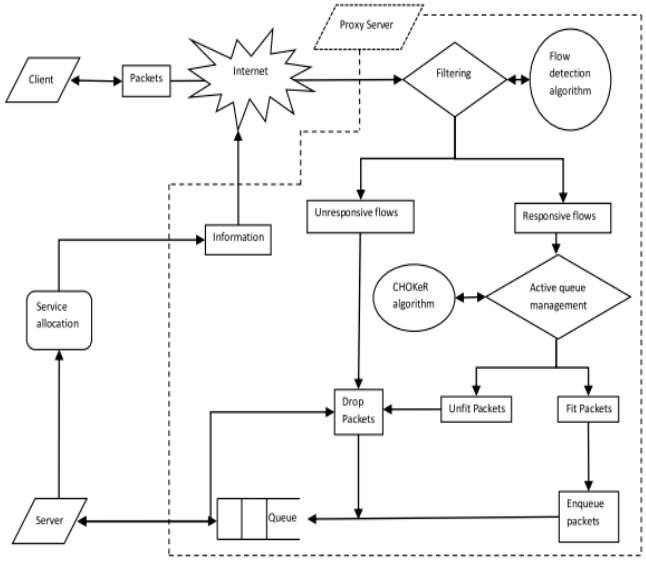

Fig. 4 Process flow

```
for each packet arrival
begin
    insert packet with probability P and then forward packet in FIFO order;
    otherwise drop packet;
end {for}
```

Fig. 5 General algorithm

Case 1: When L_k is between the L_{th} and L_{min}

$$p0 = \max(0, p0 - pmin)$$

Case 2: When L_k is between the L_{max} and L_{lim}

$$p0 = p0 + a * pmax$$

Where 'a' is the number of steps,

$$a = \frac{Lk - Lmax}{Lmax - Lmin}$$

Where L_{th}, L_{min}, L_{max} and L_{lim} are the thresholds adjusted in buffer length.

Fig. 7 gives the central algorithm for proxy server. The proxy server starts with a given host address and port address. Any client which ever wants to communicate with this proxy can use this host address and port to send data. Also the port forwarding thread starts together with proxy. To help forwarding the proxy server is configured with a list of registered servers. For each http requests arrives at assigned port in proxy server, the algorithm fetches the request from port, decodes them and generate respective raw packet from it. Then it applies CHOKeR algorithm on the packet and after proper queue management the packet is either dropped or inserted in FIFO order. If packet is dropped, then the algorithm sends ECN (Explicit Congestion Notification) to the corresponding client [5]. Otherwise the inserted packet is processed in order to respective server by the forwarding thread.

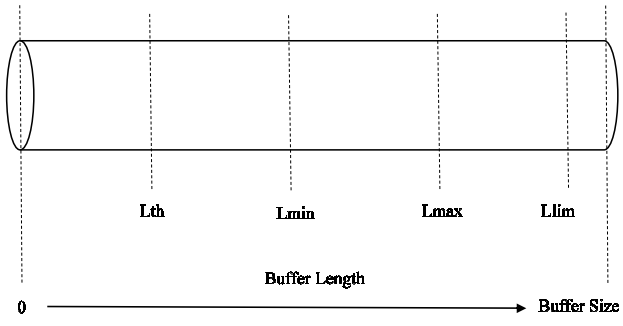

Fig. 6 Buffer thresholds

```
MUfProx :
{algorithm to handle unresponsive flows at server side}
host_address = /*IP address of host*/;
host_port = /*port number of host*/;
begin
    Start_Proxy(host_address , host_port);
    Start_Forward();
    for each http request arrival do
        begin
            http_request = fetch http request;
            packet = Generate_Packet(http_request);
            is_insert = Do_AQM(packet);
            if (is_insert)
                push packet in buffer;
            else
                reply ECN;
        end {for}
end {MUfProx}
```

Fig. 7 Central algorithm

The pseudo code for CHOKeR AQM algorithm is represented in Fig. 8. The parameters L_{th}, L_{min}, L_{max}, L_{lim}, p_{min} and p_{max} are initialized at starting of proxy server. They are manually tuned based on the network congestion level at starting. On each packet arrival the buffer occupancy L_k is compared with the thresholds L_{th}, L_{min}, L_{max} and L_{lim}. As CHOKeR does, if L_k is less than L_{th} the probability for packet drawing (p_0) is kept 0 which means no dropping is needed due to the buffer is congestion free. If L_k exceeds L_{th}, the algorithm assumes there may be chance for congestion. So for values of L_k between L_{th} and L_{min}, the value of p0 reduce to Single step Decrease, till value to zero. It means, the congestion is brought to be reduced. But when L_k exceeds L_{min}, there should be active queue management to provide better load balancing. When L_k exceeds L_{max}, there is a large chance for buffer unavailability which triggers the case of congestion, thus algorithm tries to reduce occupancy below L_{max} using Multistep Increase method. I.e., it changes drawing factor to multiple of steps in buffer length variation from L_{min} to Lmax. When L_k is reduced in the range of L_{min} and L_{max}, the value of p_0 is kept intact, which provides a stable nature to congestion level, but applies active queue management later. If L_k is beyond L_{lim}, means buffer overflow, thus the packet is directly discarded [18].

'*m*' is the number of packets need to be chosen randomly for comparison and drawing. On each draw of packets from buffer, the packet's flow information (source address, source port, destination address, destination port, data payload) is compared with arriving packet's flow information. If they are same, then drawn packet is discarded. Levenshtein distance is used to compare data payload each other where its complexity is $\Theta(\min(m,n))$ [4]. For bursty flows, the buffer will contain large consecutive same packets, which also take in account in this section of comparison and dropping. Finally, arrived packet is allowed to enter in buffer at last position if the algorithm returns success which means the packet is inserted with a probability p0.

```
Do_AQM(packet:record) :
{procedure to do Active Queue Management using CHOKeR}
begin
    if(Lk < Lth) do
    begin
        p0 = 0;
        Lk = Lk + 1a;
    end {if}
    else if (Lk >= Lth and Lk < Lmin)
        p0 = max(0 , p0 - pmin);
    else if (Lk>=Lmin and Lk<Lmax)
        p0 = p0;
    else if (Lk >= Lmax and Lk < Llim) do
    begin
        a = ceil((Lk - Lmax) / (Lmax - Lmin));
        p0 = p0 + (a * pmax);
    end {else if}
    else
        return False;
    m = floor(p0);
    f = p0 - m;
    v = generate random value between(0 , 1);
    if (v < f)
        m = m+1;
    while (m > 0) do
    begin
        m = m - 1;
        random_position = generate random integer value between(0 , buffer size);
        random_packet = get packet from buffer at random_position;
        if (packet = random_packet)
            drop packet from buffer at random_postion;
    end {while}
    return True;
end {Do_AQM}
```

Fig. 8 CHOKeR algorithm

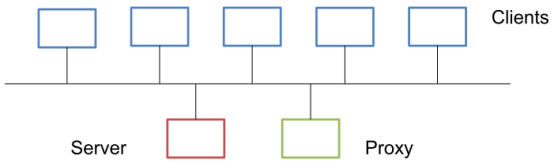

Fig. 9 Simulation topology

V. RESULTS AND DISCUSSION

The system is deployed in python environment and run from Linux based machine. The HTTP server is created using Java and hosted in Local Area Network. The client files are designed using HTML and JavaScript for some of the tests. Three test cases are carried out in this setup and their results are analysed.

Test case I

The system is tested with five different computers in local area network. The each client sends random HTTP requests in an interval of 100 milliseconds. The system is analysed for a duration of 5 minutes. The total responsive flows as well as unresponsive flows are logged and finally tabulated. The scenarios are tested with two fixed probabilities P_{min} = 0.1 and P_{max} = 0.2. Table. 1 shows the results obtained by setting parameters Q_{size} = 50, L_{th} = 10, L_{min} = 12, L_{max}= 17, L_{lim} = 50 and Fig. 10 shows its corresponding graph. Similarly Table. 2 shows the results obtained by setting

parameters Q_{size} = 100, L_{th} = 20, L_{min} = 24, L_{max}= 34, L_{lim} = 100 and Fig. 11 shows its corresponding graph. And finally Table. 3 shows the results obtained by setting parameters Q_{size} = 200, L_{th} = 40, L_{min} = 48, L_{max}= 68, L_{lim} = 200 and Fig. 12 shows its corresponding graph.

Ratio and average ratio are found using following formulae.

$$Ratio = \frac{No.\ of\ Responsive\ flows}{No.\ of\ Unresponsive\ flows}$$

$$Avg.\ Ratio = \frac{Total\ Ratio}{No.\ of\ Clients}$$

$$Detection\ Rate = \frac{Unresponsive\ Flows}{Total\ Flows} * 100$$

It can be found that average ratio is proportionally increases with change in quantity of parameters. From analysing the results we can found that the change in queue size and corresponding parameters also effect the ratio between responsive flow and unresponsive flow. It can be said that the increase in quantity of parameters is inversely proportional to amount of unresponsive flows detected. And it is mainly due to the overflow condition of queue. The results also show that the client sends minimum amount of unresponsive flows for polling the connection.

Source	Resp. Flow	Unresp. Flow	Ratio
192.168.1.66	62	14	4.42
192.168.1.67	63	10	6.30
192.168.1.75	61	19	3.21
192.168.1.76	60	21	2.85
192.168.1.77	64	17	3.76
		Avg. Ratio	4.10

Table. 1 Test case I – queue size: 50

Source	Resp. Flow	Unresp. Flow	Ratio
192.168.1.66	64	13	4.92
192.168.1.67	66	8	8.25
192.168.1.75	63	17	3.70
192.168.1.76	62	19	3.26
192.168.1.77	69	14	4.92
		Avg. Ratio	5.01

Table. 2 Test case I – queue size: 100

Source	Resp. Flow	Unresp. Flow	Ratio
192.168.1.66	74	9	8.22
192.168.1.67	76	5	15.20
192.168.1.75	66	14	4.71
192.168.1.76	68	17	4.00
192.168.1.77	73	11	6.63
		Avg. Ratio	7.75

Table. 3 Test case I – queue size: 200

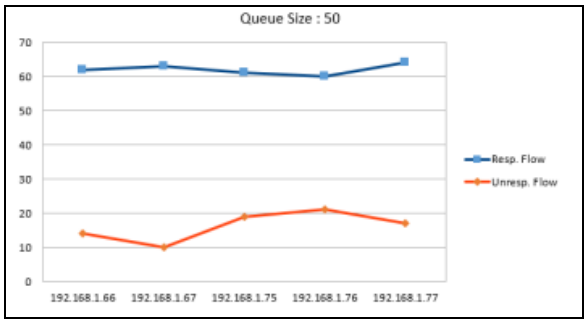

Fig. 10 Test case I – queue size: 50

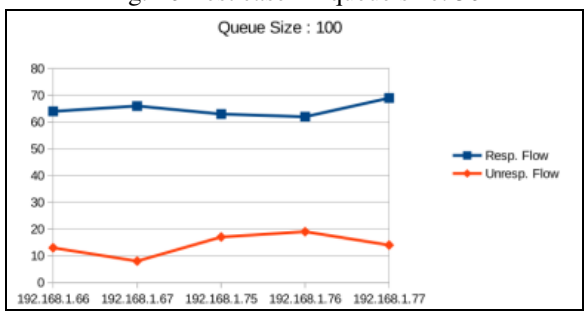

Fig. 11 Test case I – queue size: 100

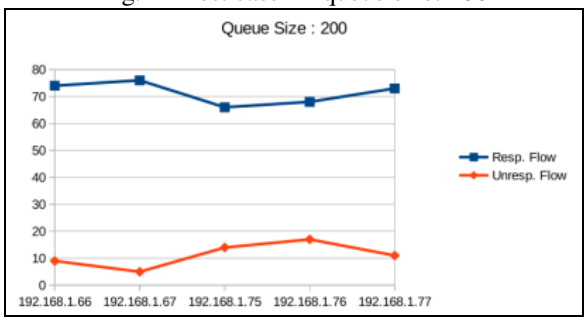

Fig. 12 Test case I – queue size: 200

Detection rate is found out using above given formula, and it gives the success rate of detecting unresponsive flows from a transaction. Detection rate for different queue size is given in Table. 4 and corresponding graph is drawn in Fig. 13. From analysing the graph, it is clear that the system successfully recognizes unresponsive flows from input mixed flows, but the detection rate is diminishing on increase in the queue size and respective parameters.

Source	Queue size		
	50	100	200
192.168.1.66	18	16	10
192.168.1.67	13	10	6
192.168.1.75	23	21	17
192.168.1.76	25	23	20
192.168.1.77	20	16	13

Table. 4 Test case I – unresponsive flow detection percentage

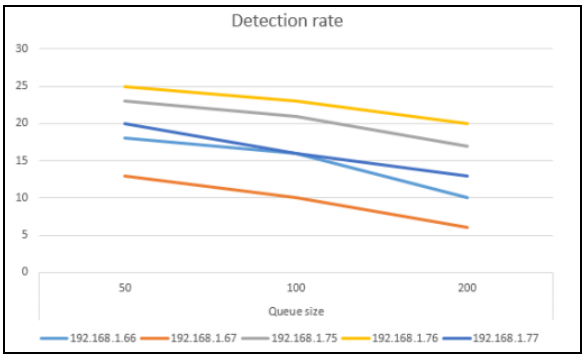

Fig. 13 Test case I – unresponsive flow detection percentage

Test case II

The system is tested with same five different computers in same local area network. The each client tries to download a file of size 2 MB. The system is analyzed till the end of downloading. The total responsive flows as well as unresponsive flows are logged and finally tabulated. The scenarios are tested with two fixed probabilities P_{min} = 0.1 and P_{max} = 0.2. Table. 5 shows the results obtained by setting parameters Q_{size} = 50, L_{th} = 10, L_{min} = 12, L_{max}= 17, L_{lim} = 50 and Fig. 14 shows its corresponding graph. Similarly Table. 6 shows the results obtained by setting parameters Q_{size} = 100, L_{th} = 20, L_{min} = 24, L_{max}= 34, L_{lim} = 100 and Fig. 15 shows its corresponding graph. And finally Table. 7 shows the results obtained by setting parameters Q_{size} = 200, L_{th} = 40, L_{min} = 48, L_{max}= 68, L_{lim} = 200 and Fig. 16 shows its corresponding graph. Ratio and average ratio are found using the same formulae given above. The same results are obtained in this experiment. It can be found that average ratio is proportionally increases with change in quantity of parameters and finally it comes close to average number of responsive flows. From analysing the results we can found that the change in queue size and corresponding parameters also effect the ratio between responsive flow and unresponsive flow. It can be said that the increase in quantity of parameters is inversely proportional to amount of unresponsive flows detected. And it is mainly due to the overflow condition of queue. The results also show that the variation between the amount of responsive flows and as well as unresponsive flows with respect to change of parameters also does not differ well due to the caching feature of HTTP agent.

Source	Resp. Flow	Unresp. Flow	Ratio
192.168.1.66	17	8	2.12
192.168.1.67	21	4	5.25
192.168.1.75	19	7	2.71
192.168.1.76	16	8	2.00
192.168.1.77	17	7	2.42
		Avg. Ratio	2.90

Table. 5 Test case II – queue size: 50

Source	Resp. Flow	Unresp. Flow	Ratio
192.168.1.66	26	4	6.50
192.168.1.67	33	2	16.5
192.168.1.75	23	3	7.66
192.168.1.76	23	4	5.75
192.168.1.77	24	5	4.80
		Avg. Ratio	8.24

Table. 6 Test case II – queue size: 100

Source	Resp. Flow	Unresp. Flow	Ratio
192.168.1.66	28	3	9.33
192.168.1.67	36	2	18.00
192.168.1.75	24	2	12.00
192.168.1.76	26	1	26.00
192.168.1.77	29	4	7.25
		Avg. Ratio	14.51

Table. 7 Test case II – queue size: 200

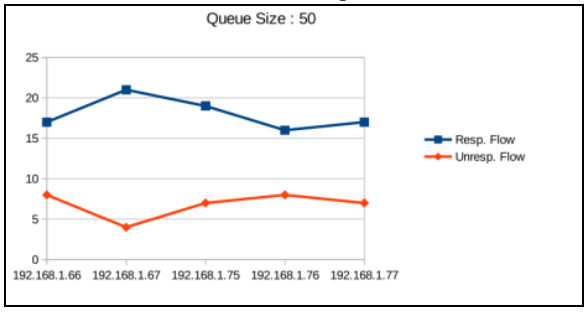

Fig. 14 Test case II – queue size: 50

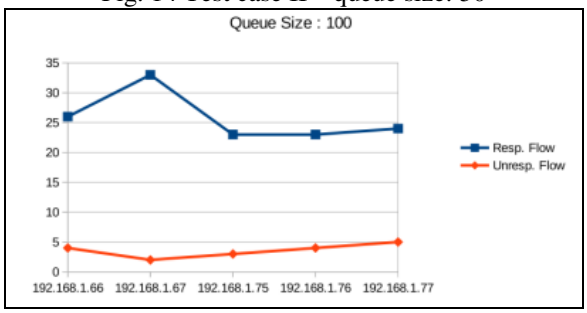

Fig. 15 Test case II – queue size: 100

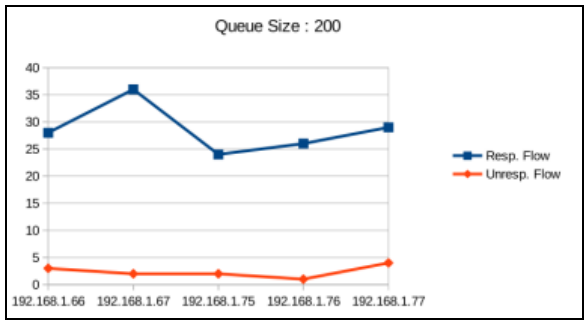

Fig. 16 Test case II – queue size: 200

Detection rate for different queue size is given in Table. 8 and corresponding graph is drawn in Fig. 17. From analysing the graph, it can be said that the results are almost

same as above test case I. i.e., here also the detection rate is diminishing with increase in queue size.

Source	Queue size		
	50	100	200
192.168.1.66	32	13	9
192.168.1.67	16	5	5
192.168.1.75	26	11	7
192.168.1.76	33	14	3
192.168.1.77	29	17	12

Table. 8 Test case II – unresponsive flow detection percentage

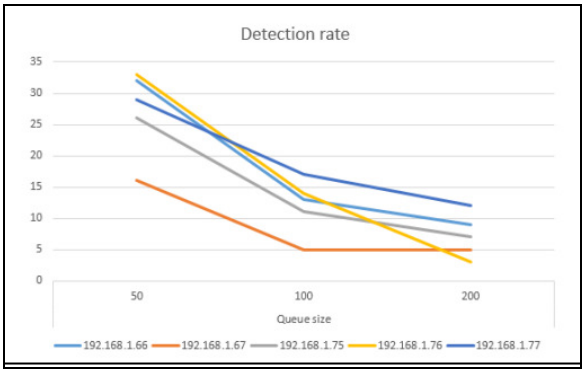

Fig. 17 Test case II – unresponsive flow detection percentage

Test case III

Source	CHOKeR						CHOKe					
	Q.Size : 512		Q.Size : 1024		Q.Size : 2048		Q.Size : 512		Q.Size : 1024		Q.Size : 2048	
	Insertions	Drops	Insertions	Drops	Insertions	Drops	Insertions	Drops	Insertions	Drops	Insertions	Drops
192.168.1.66	560	72	1120	233	2144	515	560	31	1120	134	2144	364
192.168.1.67	600	77	1162	241	2191	526	600	33	1162	139	2191	371
192.168.1.75	546	70	1100	228	2112	507	546	30	1100	131	2112	357
192.168.1.76	558	71	1115	231	2127	510	558	30	1115	132	2127	359
192.168.1.77	576	73	1141	236	2153	516	576	30	1141	135	2153	363

Table. 9 Test case III

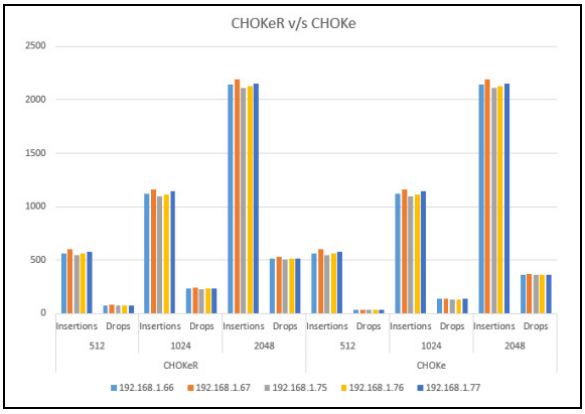

Fig. 18 Test case III

The system is tested with five different computers in local area network. The each client sends random HTTP requests in an interval of 50 milliseconds. The system is analyzed for a duration of 5 minutes. The total number of packet insertions as well as drops are logged and finally tabulated. The scenarios are tested with two fixed probabilities P_{min} = 0.01 and P_{max} = 0.02. The variable parameters are taken into three sets {Q_{size} = 512, L_{th} = 352, L_{min} = 412, L_{max}= 452, L_{lim} = 512}, {Q_{size} = 1024, L_{th} = 712, L_{min} = 812, L_{max}= 912, L_{lim} = 1024} and {Q_{size} = 2048, L_{th} = 1438, L_{min} = 1638, L_{max}= 1838, L_{lim} = 2048}. Table .9 shows the results obtained and Fig. 18 shows its corresponding graph. The experiment is carried out to compare the performance of existing CHOKe algorithm and its descendant CHOKeR algorithm in server side congestion control. The results show that, for the same amount of insertions CHOKeR has larger number of drops than that of CHOKe. That is, CHOKeR detects and drops more unresponsive flows than CHOKe does for the same amount of requests. So it can be found that the system is more congestion tolerable than its ancestors.

VI. CONCLUSION

The system is a proxy server which uses CHOKeR active queue management algorithm to manage unresponsive flows in the internet. The proxy server considers both TCP as well as UDP flows which can be extracted from the HTTP flows in the congested network. The use of virtual queues in the AQM scheme at proxy server makes physical buffers which are present at routers can be kept intact in the presence of misbehaving flows and over flood conditions.

Results show that Active Queue Management is effective at server side. From experimental results, we can found that the change in queue size and corresponding parameters also effect the ratio between responsive flow and unresponsive flow. It can be derived that application of CHOKeR algorithm at HTTP flows is effective as much as that of normal TCP/UDP flows.

Current system can be extended in future to handle special messages like ICMP and also adding hash bins at buffer side increases speed of flow matching.

References

[1] en.wikipedia.org/wiki/IP_address.

[2] B. Kiruthiga and Dr. E. George Dharma Prakash Raj, “Survey on AQM Congestion Control Algorithms”, IJCSMC, Vol. 2, Issue. 2, pp.38–44, Feb **2014**.

[3] en.wikipedia.org/wiki/Fair_queuing.

[4] en.wikipedia.org/wiki/ Levenshtein distance.html.

[5] en.wikipedia.org/wiki/Explicit_Congestion_Notification.

[6] gettys.wordpress.com/active-queue-management-aqm-faq.

[7] G.F.Ali Ahammed, Reshma Banu, "Analyzing the Performance of Active Queue Management Algorithms", IJCNC, Vol. 2, pp. 19, Mar **2010**.

[8] searchnetworking.techtarget.com/definition/load-balancing.

[9] Rong Pan, Balaji Prabhakar, Konstantinos Psounis, "CHOKe: A stateless active queue management scheme for approximating fair bandwidth allocation", INFOCOM 2000, vol.2, pp. 942-951, Mar **2000**.

[10] Ao Tang, Jiantao Wang and Steven H. Low, "Understanding CHOKe: Throughput and Spatial Characteristics", IEEE/ACM Trans. Networking, vol. 12, No. 4, pp. 694-707, Aug **2004**.

[11] Jiang Ming, WU Chumming, Zhang Min and Bian Hao, "CSa-XCHOKe: A Congestion Adaptive CHOKe Algorithm", Chinese Journal of Electronics, Vol.19, No.4, Oct **2010**.

[12] Ying Jiang, and Jing Liu, "Self adjustable CHOKe: an active queue management algorithm for congestion control and fair bandwidth allocation", IEEE computers and comm., Vol.2, No.4, pp. 1018-1024, Jul **2013**.

[13] K.Chitra and Dr. G.Padamavathi, "Adaptive CHOKe: An algorithm to increase the fairness in Internet Routers", IJANA, vol. 01, Issue. 06, pp. 382-386, Apr **2010**.

[14] G. Sasikala and E. George Dharma Prakash Raj, "P-CHOKe: A Piggybacking-CHOKe AQM Congestion Control Method", IJCSMC, Vol. 2, Issue. 8, pp.136–144, Aug **2013**.

[15] Addisu Eshete and Yuming Jiang, "Protection from Unresponsive Flows with Geometric CHOKe", Centre for Quantifiable Quality of Service in Communication Systems, Feb **2012**.

[16] K.Chitra and Dr.G.Padmavathi, "FAVQCHOKE: To Allocate Fair Buffer To A Dynamically Varying Traffic In An Ip Network", IJDPS, Vol. 2, Issue. 1, pp.73–82, Jan **2011**.

[17] Shushan Wen, Yuguang Fang and Hairong Sun, "CHOKeW: Bandwidth Differentiation and TCP Protection in Core Networks", IEEE Trans. Parallel and Distributed Sys. , Vol. 20, NO. 1, pp. 34-47, Jan **2009**.

[18] Lingyun Lu, Haifeng Du and Ren Ping Liu, "CHOKeR: A Novel AQM Algorithm with Proportional Bandwidth Allocation and TCP Protection", IEEE Trans. Industrail Informatics, Vol. 10, No. 1, pp.637–644, Feb **2014**.

[19] Addisu Eshete and Yuming Jiang, "Generalizing the CHOKe Flow Protection", Preprint submitted to Computer Networks, pp.1–28, Feb **2012**.

[20] Shalki Chahar, "Social Networking Analysis", International Journal of Computer Sciences and Engineering, Vol. 02, No. 5, pp.159–163, May **2014**.

[21] Vijith C and Dr. M. Azath, "Survey on CHOKe AQM Family", International Journal of Computer Sciences and Engineering, Vol. 02, No. 11, pp.81–85, Nov **2014**.

AUTHORS PROFILE

Vijith C has completed B Tech in CSE from Sahrdaya College of Engineering and Technology, Thrissur, Kerala, in 2012. Presently he is pursuing his M Tech in CSE from Met's School of engineering, Thrissur, Kerala.

Dr. M. Azath is Head of Department of Computer Science and engineering, Met's School Of Engineering, Mala. He has received Ph.D. in Computer Science and Engineering from Anna University in 2011. He is a member in Editorial board of various international and national journals and also a member of the Computer society of India, Salem. His research interests include Networking, Wireless networks, Mobile Computing and Network Security.

