Proceedings of the Second International Conference on Mobile e-Services (ICOMeS) Oct. 27 – 29, 2009. Volume 2, ISBN: 978-2902-43-8

SIMULATION OF AN ACTIVE QUEUE MANAGEMENT IN A QUALITY OF SERVICE INTERNET USING RANDOM EARLY DETECTION (RED)

[a]OYETUNJI M.O., [b]OLADEJI F. O [c]UWADIA C O. [d]EMUOYIBOFARHE O.J, [e]AKINWUNMI A. O

[ae]Department of Computer Science and Information, Technology Bowen University, Iwo, Osun State, Nigeria.

[bc]Department of Computer Science University of Lagos, Akoka, Lagos State, Nigeria.

[d]Departmeent of Computer Science and Engineering Ladoke Akintola University of Technology Ogbomoso, Oyo State, Nigeria.

All correspondence should be directed to:

kunleoyetunji2002@yahoo.com 08034291396

ABSTRACT

This paper work presents simulation of an active queue management in a quality of service Internet using differentiated service random early detection (DiffServ RED) algorithm for congestion avoidance in packet switched network. RED algorithm is designed to accompany a transport-layer congestion control protocol such as TCP. We explored how the algorithm, earlier designed by [8], could be used to achieve differential packet dropping required by real-time traffic in the extended TCP/IP protocol for the Internet.

The queue manager operates by setting thresholds (minimum and maximum) for the average queue size to detect incipient congestion. When the average queue size is below the minimum threshold, all arriving packets are properly enqueued for dispatch to their destinations. Any time the average queue size exceeds a preset minimum thresholds, the gateway drops packets at random or marks each arriving packet with a certain probability, where the exact probability is a function of the average queue size. A case when the average queue size exceeds the set maximum threshold, the gateway drops all other incoming packets.

Differential packet dropping demands that sources violating their agreed sending profiles should have some of its packets discarded during incipient congestion to reduce global synchronization. This study simulated using ns 2, how RED logic marks or drops at random compliant and non-compliant packets means to highly increase the throughput and drastically reduce losses in order to step down congestion over the network. The RED gateway has no bias against bursty traffic and avoids global synchronization of many connections decreasing their windows at the same time.

KEYWORDS: Differentiated Service Random Early Detection, Quality of Service, Gateway, Packets, Congestion, Average Queue Size and Queue Manager.

1.0 INTRODUCTION

The expansion of the Internet and the accessibility to low cost broadband services are making multimedia traffics over Internet a reliable option. The real time multimedia application such as video over Internet Protocol (VIP), Voice over Internet Protocol (VoIP), web learning and remote login need reliable and efficient Quality of Service (QoS). An approach to extend the TCP/IP led to the introduction of Integrated Service (IntServ) architecture which is characterized by resource reservation by router [1] for each session, leading to scalability problems when thousand of applications are requesting reservation at one time. The weakness of IntServ brought about scalable differentiated services (DiffServ) model, which provides for aggregates traffic to be classified and conditioned at the boundary of the network on the basis of performance. The DiffServ has instant impact on real time applications with respect to QoS and Traffic Engineering (TE).

Of recent, measurements taken have shown that the growing demand for network bandwidth has driven loss rates up across various links in the Internet [4]. In order to stem the escalating packet loss rates caused by an exponential increase in network traffic and congestions, the IETF is considering the deployment of explicit congestion notification (ECN) [2], [5], along with active queue management techniques such as RED (Random Early Detection) [1], [3].

The basic idea behind RED queue management is to detect incipient congestion early and to pass on congestion notification to the sources, allowing them to trim down their transmission rates before queues in the network overflow and packets are marked.

To do this, RED maintains an exponentially weighted moving average of the queue length which it uses to discover congestion. When the average queue length is below the minimum threshold (MINtr), packets flow at an increasing rate. When the average queue length exceeds a minimum threshold (MINtr), packets are randomly dropped or marked with an ECN bit. When the average queue length exceeds a maximum threshold (MAXtr), all packets are dropped or marked. The basic idea is that one should not wait until the buffer is full before detecting congestion but start detecting it before the buffer overflows.

The aim of this study is to explore the architecture of a differentiated Service (DiffServ) in order to manage buffer before it fills up. To achieve the set goals, we carried out an in dept study on an active queue management scheme-Diffserv RED to know how one can manage buffer before it fills up. Also, we designed a topology for the DiffServ RED and as well simulate it using a network simulator (NS2.31) to provide an effective tool for dissipating information about the active queue manager. We used network animator to view the simulation result.

2.0 THEORETICAL FRAMEWORK

2.1 General Overview

As Internet expansion increases, it becomes obvious that existing congestion control solutions deployed in the Internet Transport Control Protocol (TCP) are becoming ineffective. It is also generally accepted that these solutions cannot easily scale up [9]. As stated in section one, more users use this for the delivery of real time video and voice services with very poor QoS. Based on these facts, it is becoming clear that new approaches for congestion control must be investigated.
The Internet Engineering Task Force (IETF) has proposed two basic QoS provisioning techniques to manage and reserve the QoS resources in the Internet:per-flow-based Integrated Service (IntServ) [11] and aggregation-bas Differentiated Services (DiffServ). The requirement in IntServ to keep a flow-specific state for each flow raises some concerns. First, the amount of state information increases with the number of flows. The router may need high storage space and high processing power. Second, the IntServ model may make the router much more complex since they need to implement RSVP protocol, admission control and packet classifier.
IETF came up with DiffServ to build on the weakness of IntServ scalability. Scalability problem solved in IntServ was achieved in two ways: First, per flow service is replaced with aggregated service. Second, complex processing is moved from core router to edge router. A customer wishes to receive the service must have Service Level Agreement (SLA) with its service provider. SLA is a service contract between a customer and a service provider that specifies the forwarding service that the customer will receive. SLA includes a traffic conditioning agreement (TCA), which gives detail service parameter such as service level, traffic profile, marking and shaping.

2.2 Differentiated Services Module in NS

The network simulator DiffServ architecture provides quality of service by dividing traffic into different categories, marking each packet with a code point that indicates its category, and scheduling packets according to their code points. The DiffServ module in NS currently by default defines four classes of traffic, each of which has three drop precedences. Those drops precedences enable differential treatment of traffic within a single class. A single class of traffic is enqueued into one physical RED queue, which contains three virtual queues (one for each drop precedence). Different RED parameters are used for the virtual queues, causing packets from one virtual queue to drop more frequently than packets from another. A packet with lower drop precedence is given better treatment in times of congestion because it is assigned a code point that corresponds to a virtual queue with relatively lenient RED parameters.

According to [7] "The DiffServ module in ns has three major components:

Policy: *Policy is specified by network administrator about the level of service a class of traffic should receive in the network.*

Edge routers: *Edge routers marks packets with a code point according to the policy specified.*

Core routers: *Core routers examine packets' code point marking and forward them accordingly.*

The DiffServ functionality in ns is captured in a Queue object. A DiffServ queue (dsREDQueue) derived from the base class Queue, contains the abilities:

- *to implement multiple physical RED queues along a single link;*
- *to implement multiple virtual queues within a physical queue, with individual set of parameters for each virtual queue;*
- *to determine in which physical and virtual queue a packet is enqueued, according to policy specified.*

Six different policy models are defined in ns , which are:

- *TSW2CM (TSW2CMPolicer): uses a CIR and two drop precedences. The lower precedence is used probabilistically when the CIR is exceeded.*
- *TSW3CM (TSW3CMPolicer): uses a CIR, a PIR, and three drop precedences. The medium drop precedence is used probabilistically when the CIR is exceeded and the lowest drop precedence is used probabilistically when the PIR is exceeded.*
- *Token Bucket (tokenBucketPolicer): uses a*

  CIR and a CBS and two drop precedences. An arriving packet is marked with the lower precedence if and only if it is larger than the token bucket.

- *Single Rate Three Color Marker*

(srTCMPolicer): uses a CIR, CBS, and an EBS to choose from three drop precedences.

- *Two Rate Three Color Marker*

  (trTCMPolicer): uses a CIR, CBS, PIR, and a PBS to choose from three drop precdences.

  Scheduling modes supported are Weighted

  Interleaved Round Robin (WIRR), Round Robin (RR), and Priority (PRI). For Priority scheduling, priority is arranged in sequential order with queue
  0 having the highest priority."

RIO Based [13] scheme has been proposed as simple means of Active Queue Management (AQM) at the core routers. The basic of the RIO (RED with in/out) mechanism is RED based differentiated dropping of packets during congestion at the router. The RIO scheme utilizes a single queue. Two sets of RED parameters are maintained, one each for in-profile and out-ofprofile packets. The drop probabilities of the inprofile are obviously lower than out-of-profile packets.

All policies are stored in the policy table in class PolicyClassifier. This table is an array that includes fields for the source and destination nodes, a policer type, a meter type, an initial code point, and various states information as shown below:

The rates CIR and PIR are specified in bits per second:

CIR: committed information rate

PIR: peak information rate

The buckets CBS, EBS, and PBS are specified in bytes:

CBS: committed burst size

EBS: excess burst size

PBS: peak burst size

C bucket: current size of the committed bucket

E bucket: current size of the excess bucket

P bucket: current size of the peak bucket

2.3 Red Algorithm

The most popular algorithm used for DiffServ implementation is RED (Random Early Discard) as earlier discussed by [8]. RED simply sets some Minimum and Maximum dropping thresholds for each class. In case the buffer queue size exceeds the minimum threshold, RED starts dropping packets randomly based on a probability which depends on queue length. If the buffer queue size exceeds the maximum threshold then every packet is dropped, i.e. drop probability is set to 1. The RED implementation for DiffServ defines that we have different thresholds for each class. Thus, the RED gateway has two separate algorithms. The algorithm for computing the average queue size determines the degree of burstiness that will be allowed in the gateway queue. The algorithm for calculating the packet-marking probability determines how frequently the gateway marks packets, given the current level of congestion. The general RED gateway algorithm is given in figure
2.1

For each packet arrival

Calculate average queue size AVG

If $MINtr \leq AVG < MAXtr$

Calculate probability p_a

With probability p_a

Mark the arriving packet

Else if $MAXtr \leq AVG$

Mark every arriving packet.

Figure 2.1: General algorithm for RED gateways.

The goal is for the gateway to mark packets at fairly evenly spaced intervals, in order to avoid bias against bursty traffic and global synchronization. The detailed algorithm for the dsRED gateway is given in figure 2.2

The gateway's calculations of the average queue size take into account the period when the queue is idle by estimating the number *m* of small packets that *could* have been transmitted by the gateway during the *idle period*. After the idle period the gateway computes the average queue size as if *m* packets had arrived to an empty queue during that period.

As average queue size (AVG) varies from minimum threshold (MINtr) to maximum threshold (MAXtr), the packet-marking probability p_b varies linearly from 0 to MAXtr:

Pb ←Max$_p$ (AVG- MINtr) / (MAXtr - MINtr)

The final packet marking probability Pa increases slowly as the COUNT increases since the last marked packet:

Pa ←P$_b$ / (1-COUNT.P$_b$)

The gateway marks each packet that arrives at the gateway when the average queue size (AVG) exceeds MAXtr.

Initialization:

AVG ←0

COUNT← -1

For each packet arrival

Calculate new average queue size AVG:

If the queue is non empty

AVG← (1-Wq) AVG + W$_q$q

Else

M← f (time-q_time)

AVG← (1-Wq)^m AVG

If MINtr ≤ AVG < MAXtr

Increment COUNT

Calculate probability Pa:

Pb ← Max$_p$ (AVG- MAXtr) / (MAXtr - MINtr)

Pa ← Pb / (1-COUNT. Pb) With probability Pa:
Mark the arriving packet

COUNT←0

Else if MAXtr ≤AVG

Mark the arriving packet

COUNT←0

Else COUNT← -1

When queue becomes empty
q_time ←time
SAVED VARIABLES
AVG: average queue size
q_time: start of the queue idle time COUNT: packet since last marked packet (pkt)
FIXED PARAMETER

Wq: queue weight

MINtr: minimum threshold for queue

MAXtr: Maximum threshold for queue

Max$_p$: maximum value for P$_b$

OTHER

Pa: current pkt marking probability q: current queue size time: current time f(t): a linear function of the time t

Figure 2.2: Detailed Algorithm for RED Gateways.

3.0 EXPERIMENTAL CONFIGURATION AND SIMULATION

3.1 Overview of NS

NS-based scripts were written in NS-2 to simulate an active queue manager- DiffServ RED. The simulator uses two languages because it does two things. First, detailed simulations of protocols require a system programming language which can manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks, run-time speed is of great *important.*

Second, a large part of network research involves slightly varying parameters or configurations or quickly exploring a number of scenarios. In these cases, iteration time (change the model and rerun) is more important. NS meets both of these needs with two languages, C++ and OTcl. C++ is fast to run but slower to change, making it suitable for detailed protocol implementation. OTcl runs much slower but can be changed very quickly making it superb for simulation configuration.

3.2 Design Topology

As stated above, the design of this study focused on simulation of a differentiated service random early detection to maximise link usage and at the same time guarantee desired output interms of high throughput, low loss rate and almost 95% average fairness. The design followed clearly stated below:

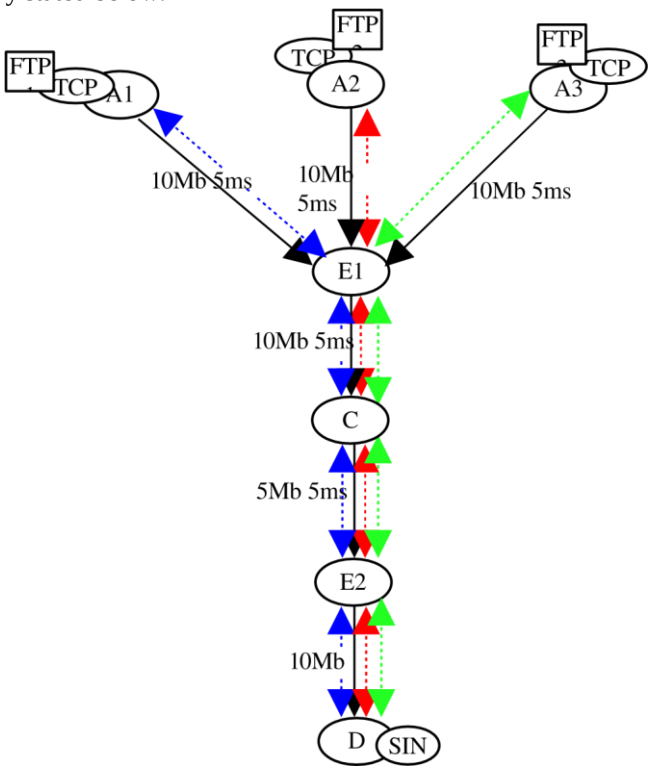

Fig 3.1: A Simple Network Topology

This network topology consists of 7 nodes (A1, A2, A3, E1, C, E2 and D) as shown in figure 3.1. This topology shows how three sources A1, A2 and A3 are connected to an edge router E1 with duplex link between A1 and E1, A2 and E1, and A3 and E1. Also, node D is connected to edge router E2 via duplex link. The destination D serves as the sink. The edge router E1 is connected to core router C, likewise core router C to edge E2 via simplex link.

The duplex links between A1 and E1, A2 and E1, A3 and E1, and E2 and D have 10 Mbps of bandwidth and 5ms of propagation delay. The simplex link between E1 and C has 10Mbps of bandwidth and 5ms of propagation delay. The simplex link between C and E2 has 5Mbps of bandwidth and 5ms of propagation delay which serves as the bottleneck. Each source node uses a DropTail queue type, while the edges and the core nodes uses RED queue type.

TCP agents are attached to A1, A2 and A3 as tcp1, tcp2 and tcp3 as shown in figure 3.1 and connections are established between TCP agents and sink agent. Sink agent is attached to D. The maximum size of a packet that a TCP agent can generate is set to 1KByte. A TCP sink agent generates and sends acknowledgement (ACK) packets to the senders (TCP agents) and frees the received packets. FTP traffic generator is attached to TCP agents respectively, The FTP generators (ftp1, ftp2 and ftp3) are set to start at 0.0s and stop at 85sec.

3.3 Parameter Settings for Diffserv RED

The policer used for the experiment is token bucket which uses a CIR and a CBS with two drop precedence. The edge router does the marking for the incoming packets to differentiate between compliant packets and non complaint packets. An arriving packet is marked with a lower precedence if and only if it is larger than the token bucket. All flows matching the source to destination pair are treated as a single traffic aggregate (A1 to D, A2 to D and A3 to D)

The token bucket identified source A1 with DSCP 10 downgraded to 11, source A2 with DSCP 20 downgraded to 21, and source A3 with DSCP 30 downgraded to 31. The Committed Information Rate (CIR) for the three sources (A1, A2, A3) is 1Mbps with Committed Burst Rate (CBS) of 10Kbytes. Also, the packet size and the sending rate are 1Kbytes and 3Mbps respectively. The queue limit between E1 and core was 60 packets.

3.3.1 Random Early Detection (RED)

Configuration

The RED with in-profile and out-profile (RIO-C) is an active queue management for edge and core routers. Each physical queue is configured to have two virtual queues, one for in-profile (with DSCP 10, 20 & 30) and the other is for out-profile (with DSCP 11, 21 & 31) traffic. So, we have Queue 1 with CP (10 & 11), Queue 2 with CP (20 & 21), and Queue 3 (30 & 31). The RED parameters for the simulation are listed in the table below:

TABLE 3.1: DIFFERENTIATED SERVICE RED PARAMETERS

RED Parameters		Minimum Threshold (MINtr)	Maximum Threshold (MAXtr)	Maximum Drop Probability (Max_p
Physical Queue	Virtual Queue			
0	0	20	40	0.02
0	1	10	20	0.10
1	0	20	40	0.02
1	1	10	20	0.10
2	0	20	40	0.02
2	1	10	20	0.10

From table 3.1, as the precedence value increases, the RED parameters become harsher. Any source violating the admission control rule would have its packets downgraded to lower class which may likely be discarded.

A packet is marked with a code point at the edge router according to the policy specified before it is put into the corresponding physical and virtual queue. Policy, for each different traffic aggregate, has an associated policer type, meter type, and initial code point. Core router does packet forwarding to the next edge router, for proper delivery to the destination. The bottleneck queue is between core router (C) and edge router (E2). This gives room to queue monitoring and evaluation of packets loss at the bottleneck. The fluctuation of average queue size gives room for busty traffic.

4.0 RESULT ANALYSIS

The experiment was carried out for 85 seconds and the statistics (outputs) were taken after every 20 seconds. The dispatch policy employed was strictly priority.

4.1 Analysis Based on Fairness

With reference to Jain's Fairness Index, the formula stated below was used to calculate the fairness among the queues

$$\text{Fairness} = \frac{\Sigma (x_i)^2}{N * x_i^2}$$

Where: x_i is queue i throughput and N is the number of queue.

Based on the "printStats" result recorded after 80 seconds, the total number of packets serviced in all the three Queue were 44869 (15166 from source1, 15149 from source2 and 14554 from source3) with an average fairness of 95.28% at the fairness rate of 0.9528 (0.9572 for queue1, 0.9598 for queue2 and 0.9417 for queue3) . The experiment showed that the dsRED router was fair to all the three queues with almost the same fairness rate. Based on the result, the diffServ shows no bias against bursty traffic.
The figure 3.1 shows the pictorial representation of the fairness rate among the queues.

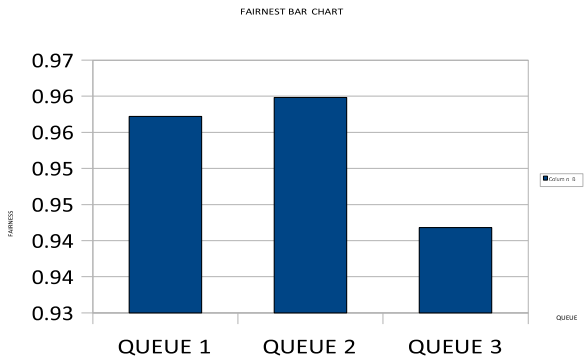

Fig 4.1: Fairness bar chart

4.2.2 Analysis Based on Loss

Based on diffServ RED throughput using priority

scheduling policy, the total loss of 588 packets (588000 bytes) was recorded out of 45 427 packets (45 427 000 bytes) received at the edge router. The average packet loss percentage was just about 1.229% which is very small. The packets lost were mainly from non-compliant packets of the three queues. The total number of packets dropped lately were 255. 303 packets were dropped due to RED early drop. The loss rates were estimated and plotted against the queues. The figure 3.2

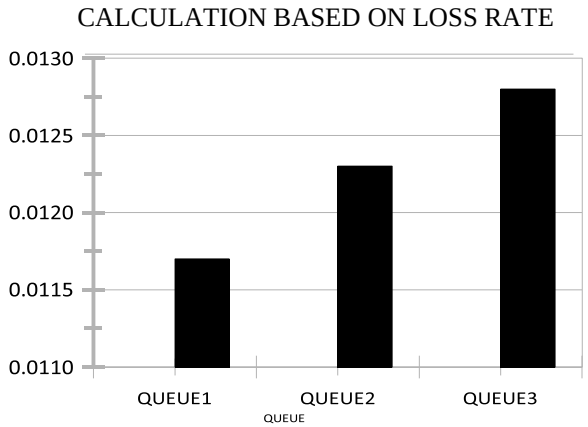

Fig 4.2: Loss rate bar chart

4.2.3 ANALYSIS BASED ON THROUGHPUT

The throughput result for dsRED was estimated and recorded using priority scheduler after running the simulation for 85seconds. When the statistics of the early drops (drops due to random dropping of dsRED) and late drops (packets dropped when average queue size is greater than maximum threshold) were taken, very few packets were dropped from compliant and non complaint packets which increases the overall throughput to 44869 packets (44 869 000bytes) out of 45427 packets (45 427 000bytes) received at the edge router (E1). The buffer at the edge router (E1) using dsRED was properly managed to avoid buffer overflow which immensely contributed to very high throughput. The percentage of the work done (throughput) out of

the total packets received at the edge router E1 was 98.77%. Based on the computation, the ratio of throughput of compliant to non-compliant packets was 6:4. Fig 4.3 show the throughput obtained in the course of the simulation.

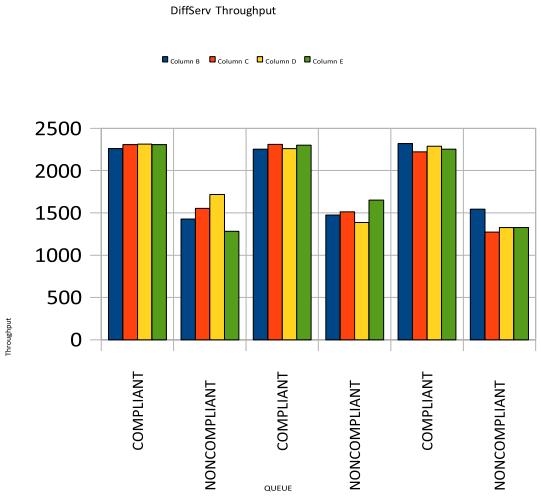

Fig 4.3: Throughput bar chart

4.2.4 No Global Synchronization

The rate at which RED gateways mark packets depends on the level of congestion. During low congestion, the gateway has a low probability of marking each arriving packet, and as congestion increases, the probability of marking each packet increases. In order to prevent all the sources from slowing down the rate of transmitting packets at the same time unlike Drop Tail and Random drop gateway, RED gateways mark or drop packets at random to prevent global synchronization. DiffServ RED has better throughput than drop tail because of global synchronization avoidance as demonstrated in Figure 4.4 using network animator.

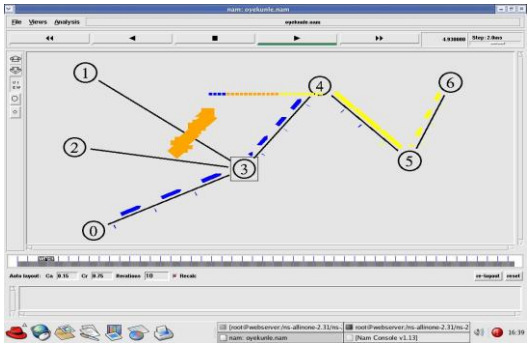

Fig 4.4: Network animator for the simulation

5.0 CONCLUSION

This study has enabled us to explore the architecture of diffserv RED to classify and condition aggregates traffic using the compliant and non compliant differentiated service code point (DSCP). It gears up the quality of service in the network by maximizing the throughput. The increase in throughput drastically reduces losses in the course of the simulation. Also, we were able to detect incipient congestion and as well manage the buffer effectively before it fills up by controlling the value of an average queue size at the gateway. The fairness among the queues was very high which gave no room for bias against busty traffic.

6.0 REFERENCE

[1] R. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendations on queue management and congestion avoidance in the Internet,", RFC 2309, Apr 1998.

[2] S. Floyd, "TCP and explicit congestion notification," Comput. Commun. Rev., vol. 24, no. 5, pp. 10–23, Oct. 1994.

[3] "Random early detection gateways for congestion avoidance," IEEE/ACM Trans. Networking, vol. 1, pp. 397–413, Aug. 1993.

[4] V. Paxson, "End-to-end internet packet dynamics," in Proc. ACM SIGCOMM, Sept. 1997, pp. 139–152.

[5] K. K. Ramakrishan and R. Jain, "A binary feedback scheme for congestion avoidance in computer networks," ACM Trans. Comput. Syst., vol. 8, no. 2, pp. 158–181, May 1990.

[6] Floyd, S., and Jacobson, V., "*On Traffic Phase Effects in Packet-Switched Gateways, Internetworking:* Research and Experience," V.3 N.3, September 1992, p.115-156.

[7] ns-2 Manual The ns Manual , The VINT Project collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC Kevin Fall, Editor Kannan

[8] Floyd, S., and Jacobson, V., *"Random Early Detection for Congestion Avoidance"* V.I N.4, August 1993, p. 397-413

[9] V. Jacobson, R. Braden, D. Borman, TCP Extensions for High Performance, RFC 1323, May 1992.

[10] S. Blake et al. "An architecture for Differentiated Services", December 1998, RFC 2475

[11] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview.

[12] Albeto Leon-Garcia and Indra Widjaja, "Communication Network" Second Edition, 2004, Pages 539-561,707720.

[13] Seddigh N., Nandy B., Pieda P., "Bandwidth Assurance Issues Flow for in a Differentiated Service Network ", Globecon, March 1999.