"Design and Analysis of AQM Techniques for Network Congestion Control"

A Report Submitted to the
BIJU PATTNAIK UNIVERSITY OF
TECHNOLOGY, ROURKELA

In Partial Fulfilment of the Requirement for the Award of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE & ENGINEERING
BY

PRATIK KUMAR PANDEY 1301227131
RAJESH KUMAR SAW 1301227109
ARPITA VIDYANTA 1301227288

Under the supervision of
Prof. SUKANT KISHORO BISOYI

Department of Computer Science & Engineering,
C V RAMAN COLLEGE OF ENGINEERING, BBSR

April 5, 2017

"Design and Analysis of AQM Techniques for Network Congestion Control"

A Report Submitted to the
BIJU PATTNAIK UNIVERSITY OF
TECHNOLOGY, ROURKELA

In Partial Fulfilment of the Requirement for the Award of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE & ENGINEERING
BY

PRATIK KUMAR PANDEY 1301227131
RAJESH KUMAR SAW 1301227109
ARPITA VIDYANTA 1301227288

Under the supervision of
Prof. SUKANT KISHORO BISOYI

Department of Computer Science & Engineering,
C V RAMAN COLLEGE OF ENGINEERING, BBSR

April 5, 2017

CERTIFICATE

This is certify that the project entitled, "Design and Analysis of AQM Techniques for Network Congestion Control" submitted by PRATIK KUMAR PANDEY, RAJESH KUMAR SAW and ARPITA VIDYANTA is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Technology (Computer Science & Engineering) is a research work carried out by them under our supervision during period 2016-2017.The thesis has fulfilled all the requirements as per regulation of the University.

Date: / /

	Prof. SUKANT K. BISOYI	Dr. B. PATI
(External)	(Project Guide)	(HOD,CSE)

DECLARATION

This thesis is a presentation of my original research work. Wherever contributions of others are involved, every effort is made to indicate that clearly. With due reference to the literature and acknowledgement of collaborative research and discussion. The work was done under the guidance of Prof. SUKANT KISHORO BISOYI. The results embodied in this thesis have not been submitted to any other university or Institute for the award of any degree or diplomas.

PRATIK KUMAR PANDEY
RAJESH KUMAR SAW
ARPITA VIDYANTA

Department of Computer Science and Engineering

C.V. RAMAN COLLEGE OF ENGINEERING

ACKNOWLEDGEMENT

First and foremost, we would like to express our sincere gratitude to our guide Prof. S.K. Bisoyi for helping us all the time whenever we needed help and guidance. We are grateful to him for having supporting us the way he did, given us intellectual freedom, yet showing a possible approach whenever we needed help. Needless to say, without his guidance and support this work would not have been possible. We am grateful to our parents and guide Prof. S.K. Bisoyi for having supported us through this past year whenever we felt down as a result of our work not turning out the way we expected it to. We would also like to thank Dr. B Pati for helping us with the proof reading of this thesis.

PRATIK KUMAR PANDEY
RAJESH KUMAR SAW
ARPITA VIDYANTA

CONTENT

Chapter 5

5. NEURAL NETWORK

Chapter 6

6. PROPOSED WORK

Chapter 7

7. RESULT AND ANALYSIS

Chapter 8

8. CONCLUSION AND FUTURE WORK

LIST OF FIGURES

ABSTRACT

Congestion occurs in the network when the incoming data packets are more than the capacity of the network. To control congestion, TCP is used as transport layer protocol which is window-sized based. This specification provides a high utilization of the available bandwidth and is at the basis of the "congestion control" mechanism: when a bottleneck link discards some packets because of an overflow in its routing buffer or a packet is lost or arrives corrupted at the end-point, A (the sender) reduces its congestion window and hence decreases the load in the sender perspective. Traditionally, Droptail is used as queue management technique along with TCP. Droptail creates many problems like synchronization and buffer bloat. To overcome this problem, many active queue management (AQM) techniques have been developed as network algorithm.

In this project, we have developed three new AQM techniques named as Proposed AQM, MPID and AN-MPID. We have divided the project into two parts. In first part of our project, we have developed a new AQM called Proposed AQM and compared its performance with Proportional Integral (PI), Random Early Marking (REM) and Intelligent Adaptive Proportional Integral (IAPI). Then the stability of the proposed AQM is analyzed with IAPI by varying different network parameters such as number of TCP connections (N), bottleneck bandwidth (C), Round trip time (RTT) and Target queue reference (q_{ref})

In second part of our project, we have modified the existing Proportional Integral Derivative (PID) controller and named as MPID. And then we have implemented feed-forward neural networks with MPID to develop AN-MPID. Then we used activation function to develop AN-MPID (Act-1), AN-MPID (Act-2) and AN-MPID (Hybrid). Then we compared the performance of the developed AQMs with PID controller. The simulation experiment result shows that our proposed algorithm performs better than existing algorithm.

Keywords- AQM, Stability, Oscillation, Neural Network, NS2.

Chapter 1
Introduction

1.1 Background and Motivation

Over the last decade Internet has caught people's attention. More and more number of people are becoming a part of it. The entire world is going online. Which is a great thing but we cannot overlook the fact that with increase in the number of Internet members, the capacity of the Internet to feed its members is not increasing. Thus, due to explosion of phones, tablets, video traffic, there has been an increase in the network congestion. And with it comes the challenge of controlling that congestion.

The important issue with Internet is the Congestion control. Congestion in the network occurs when the link bandwidth exceeds the capacity of the available routers. The primary role of router is to switch packets from the input links to the output links through buffer. Due to congestion or more number of packets in the router buffer, results in long delay in data delivery and wasting of resource to loss of packets.

It is therefore clear that in order to maintain good network performance, certain mechanisms must be provided to prevent the network from being congested for any significant period of time. Two approaches to handling congestion are *congestion control* (*or recovery*) and *congestion avoidance*. The former is reactive in that congestion control typically comes into play *after* the network is overloaded, i.e., congestion is detected. The latter is proactive in that congestion avoidance comes into play *before* the network becomes overloaded, i.e., when congestion is expected.

To prevent congestion from happening we employ certain queue management techniques or algorithms to avoid congestion by dropping packets before the buffer becomes full and thereby informing the sender that congestion has occurred and hence it needs to cut down its transmission rate of data packets. These are called Active Queue Management algorithms or AQM.

There are lot of AQM techniques have been studied in recent literatures. These schemes can be classified into three types. The first includes queue length that includes AQM schemes that are based on queue length measurements. This category is also called queue-based AQM scheme, which uses the average (or instantaneous) queue length to compute the packet dropping/marking probability. Random Early Detection (RED) [1], its modification, and PI control algorithm [2], Dynamic RED [3], PD-Controller [4], belong to this category. The second group includes AQM schemes that use put traffic rate measurements. These are called rate-based AQM schemes. Blue [5], GREEN [6] and adaptive virtual queue (AVQ) [7] are rate-based AQM schemes. The third group includes AQM schemes that use both queue length and rate. The instantaneous queue length and input traffic rate are both deployed to determine the packet dropping/marking probability in the category. Random Exponential Marking (REM) [8], Virtual Rate Control

(VRC) [9] and Yellow are representative algorithms. The primary objectives of AQM include: (1) high link utility and high goodput (exclude the duplicate packets received). On one hand, buffer emptiness should be avoided; on the other hand, it is also necessary to prevent buffer overflow or longer queue length. Otherwise, the timeout of TCP [23] flows would occur frequently, resulting in unnecessary retransmissions, thus low goodput for them. (2) Low queuing delay. This is helpful to prevent TCP timeout and also attractive for the ever-increasing real-time applications. (3) Simple and efficient. AQM must be simple and scalable to be deployed in high-speed routers. (4) Stability and robustness. AQM should be stable and robust under dynamic environments, i.e. retain good performance even if the network parameters, such as the number of flows N, round-trip time RTT.

In our minor project, we had discussed about the various existing AQM techniques. Along with it, we had proposed two new algorithms serving the same purpose but displaying better performance. That is, we proposed two new algorithms, MPID and Neuron PI. Further we compared them with the existing AQM techniques and plotted the graph of queue length v/s time of each existing AQM techniques without proposed technique.

1.2 Objective

The basic objective of the project is to develop new AQM techniques to efficiently control the congestion and improve the performance of networks.

1.3 Contributions

The contributions of this project includes the following:

(i) We have developed a new AQM technique called Proposed AQM. Then we compare it along with existing AQM techniques like PI, REM and IAPI using NS2 simulator. Then we analyze the stability of proposed controller under varying parameter conditions.

(ii) We modified the existing PID controller to develop a new technique called MPID. Then we used an adaptive neuron based MPID called AN-MPID.

(iii) We used activation functions with AN-MPID to develop AN-MPID (Act-1), AN-MPID (Act-2) and AN-MPID (Hybrid). Then we analyze with PID and MPID in NS2 simulator to see the difference in their performance. And we concluded that An-MPID (Hybrid) showed the best results.

1.4 Project outline

The Chapter 2 briefs the introduction to congestion in the network and some basic concepts about TCP. In Chapter 3, the review of literature is done. Different AQM techniques are explained in chapter 4. The Chapter 5 deals with detailed concepts of Neural Network. The proposed work is

stated and explained in Chapter 6. Simulation Results and Analysis is detailed in Chapter 7 in two parts. Part 1 deals with the analysis of performance of PI, REM, IAPI and Proposed AQM. Part 2 deals with the analysis of performance of PID, MPID, AN-MPID, AN-MPID (Act-1), AN-MPID (Act-2) and AN-MPID (Hybrid). Conclusion and future work is suggested in Chapter 8. And finally the references are listed.

Chapter 2

Basic Concepts

2.1 Internet Congestion Control

The phenomenal growth of the Internet in terms of the volume of economic activity and traffic it carries over the past fifty years represents a remarkable example of the scalability of the Internet architecture, which in spite of the growth, remains in many respects, fundamentally unchanged. One of the most enduring features of the Internet since the late 1980s has been reliance on TCP's flow-rate fairness. The Transmission Control Protocol (TCP) is one of the key Internet protocols and is responsible for managing end-to-end connections across the Internet. Since that time and still today, TCP remains one of the most important congestion control mechanisms in use in the Internet. In light of its success in supporting the immense growth in Internet traffic and infrastructure investment, and in the face of the substantial changes in market/industry structure and regulatory policy that have changed the Internet's role in the economy so dramatically

Identifying when a network is congested depends upon the definition of congestion one employs. Loosely speaking, everyone would agree that "congestion" is the state of N network overload. However, this is not a precise definition adequate for characterized exactly when or for how long a network is congested.

More precise, but different, definitions are supplied by various authors and sub disciplines in networking. Each uses the term "congestion" to describe different (but related) phenomena. Each of these meanings of congestion is useful in its own right. But regrettably the particular definition is not always clear in discussions. Congestion control provides for a fundamental set of mechanisms for maintaining the stability and efficiency of the Internet. Congestion control has been associated with TCP since Van Jacobson's work in 1988, but there is also congestion control outside of TCP (e.g., for real-time multimedia applications, multicast, and router-based mechanisms). Among the ways to classify congestion control algorithms are:

- By type and amount of feedback received from the network: Loss; delay; single-bit or multi-bit explicit signals
- By incremental deploy ability: Only sender needs modification; sender and receiver need modification; only router needs modification; sender, receiver and routers need modification.
- By performance aspect: high bandwidth-delay product networks; lossy links; fairness; advantage to short flows; variable-rate links
- By fairness criterion: max-min, proportional, "minimum potential delay"

In general, congestion in pure datagram networks must be kept at the periphery of the network, where the above mechanisms can handle it. Congestion in the Internet Backbone is problematic.

Cheap fiber-optic lines have reduced costs in the Internet backbone allowing it to be provisioned with enough bandwidth to keep congestion at the periphery.

Practical network congestion avoidance

Connection-oriented protocols, such as the widely used TCP protocol, generally watch for packet errors, losses, or delays (see Quality of Service) to adjust the transmit speed. Various network congestion avoidance processes, support different trade-offs.

TCP/IP congestion avoidance

The TCP congestion avoidance algorithm is the primary basis for congestion control in the Internet.

Problems occur when concurrent TCP flows experience port queue buffer tail-drops, defeating TCP's automatic congestion avoidance. All flows that experience port queue buffer tail-drop begin a TCP retrain at the same moment – this is called TCP global synchronization.

Active queue management

Active queue management (AQM) is the reorder or drop of network packets inside a transmit buffer that is associated with a network interface controller (NIC). This task is performed by the network scheduler

2.2 Transmission Control Protocol

TCP [23] is a standard that defines how to establish and maintain a network conversation via which application programs can exchange data. TCP works with the Internet Protocol (IP), which defines how computer send packets of data to each other. Together TCP and IP are the basic rules defining the Internet. TCP is defined by the Internet Engineering Task Force (IETF) in the Request for Comment (RFC) standards document.

TCP is a connection oriented protocol, which means a connection is established and maintained until the application programs at each end have finished exchanging messages. It determines how to break application data into packets that networks can deliver, sends packets to and accepts packets from the network layer manages flow control, and because it is meant to provide error-free data transmission-handles retransmission of dropped or garbled packet as well as acknowledgment of all packets that arrive.

In our network system, due to TCP protocol, the sender keeps on increasing its data packet transmission. But once packet drop occurs, the sender does not get acknowledgment from the dropped packet and it realizes that some kind of problem has occurred and it reduces its data packet transmission rate.

The Transmission Control Protocol provides a communication service at an intermediate level between an application program and the Internet Protocol. It provides host-to-host connectivity at the Transport Layer of the Internet model. An application does not need to know the particular mechanisms for sending data via a link to another host, such as the required packet fragmentation

on the transmission medium. At the transport layer, the protocol handles all handshaking and transmission details and presents an abstraction of the network connection to the application.

At the lower levels of the protocol stack, due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets may be lost, duplicated, or delivered out of order. TCP detects these problems, requests retransmission of lost data, rearranges out-of-order data and even helps minimize network congestion to reduce the occurrence of the other problems. If the data still remains undelivered, the source is notified of this failure. Once the TCP receiver has reassembled the sequence of octets originally transmitted, it passes them to the receiving application. Thus, TCP abstracts the application's communication from the underlying networking details.

TCP is used extensively by many applications available by internet, including the World Wide Web (WWW), E-mail, File Transfer Protocol, Secure Shell, peer-to-peer file sharing, and streaming media applications.

2.2.1 Flow Control

TCP uses an end-to-end flow control protocol to avoid having the sender send data too fast for the TCP receiver to receive and process it reliably. Having a mechanism for flow control is essential in an environment where machines of diverse network speeds communicate. For example, if a PC sends data to a smartphone that is slowly processing received data, the smartphone must regulate the data flow so as not to be overwhelmed.

TCP uses a sliding window flow control protocol. In each TCP segment, the receiver specifies in the *receive window* field the amount of additionally received data (in bytes) that it is willing to buffer for the connection. The sending host can send only up to that amount of data before it must wait for an acknowledgment and window update from the receiving host.

When a receiver advertises a window size of 0, the sender stops sending data and starts the *persist timer*. The persist timer is used to protect TCP from a deadlock situation that could arise if a subsequent window size update from the receiver is lost, and the sender cannot send more data until receiving a new window size update from the receiver. When the persist timer expires, the TCP sender attempts recovery by sending a small packet so that the receiver responds by sending another acknowledgement containing the new window size.

If a receiver is processing incoming data in small increments, it may repeatedly advertise a small receive window. This is referred to as the silly window syndrome, since it is inefficient to send only a few bytes of data in a TCP segment, given the relatively large overhead of the TCP header.

2.2.2 Congestion Control

The final main aspect of TCP is congestion control. TCP uses a number of mechanisms to achieve high performance and avoid congestion collapse, where network performance can fall by several orders of magnitude. These mechanisms control the rate of data entering the network, keeping

the data flow below a rate that would trigger collapse. They also yield an approximately max-min fair allocation between flows.

Acknowledgments for data sent, or lack of acknowledgments, are used by senders to infer network conditions between the TCP sender and receiver. Coupled with timers, TCP senders and receivers can alter the behavior of the flow of data. This is more generally referred to as congestion control and/or network congestion avoidance.

Modern implementations of TCP contain four intertwined algorithms: slow-start, congestion avoidance, fast retransmit, and fast recovery (RFC 5681).

In addition, senders employ a *retransmission timeout* (RTO) that is based on the estimated round-trip time (or RTT) between the sender and receiver, as well as the variance in this round trip time. The behavior of this timer is specified in RFC 6298. There are subtleties in the estimation of RTT. For example, senders must be careful when calculating RTT samples for retransmitted packets; typically they use Karn's Algorithm or TCP timestamps (see RFC 1323). These individual RTT samples are then averaged over time to create a Smoothed Round Trip Time (SRTT) using Jacobson's algorithm. This SRTT value is what is finally used as the round-trip time estimate.

Enhancing TCP to reliably handle loss, minimize errors, manage congestion and go fast in very high-speed environments are ongoing areas of research and standards development. As a result, there are a number of TCP congestion avoidance algorithm variations.

Chapter 3

Literature Study

This chapter provides a comprehensive review of the existing works in congestion control. Active Queue Management (AQM) aims to detect congestion in the network before it becomes severe by overfilling the router queue. It means that the router tries to reduce the sending rate of the traffic sources by dropping or marking packets. There exist two approaches to indicate congestion: Packets can be dropped and packets can be marked. First strategy requires cooperation of the endpoints and latter generates additional overhead through resending. Endpoints have to react on marked packets as they have been dropped and decrease their throughput. With this the same improvement of bandwidth utilization can be achieved, but without additional overhead costs. Additionally some AQM mechanisms aim to reduce the bandwidth of greedy flows by dropping their packets at higher rates. In this chapter we give a survey on Active Queue Management algorithms, which are suitable for Peer-to-Peer networks.

C.V. Hollot, Vishal Misra, Don Towsley and Wei-Bo Gong [2] studied a previously developed linearized model of TCP and AQM. They used classical control system techniques to develop controllers well suited for the application and came up with Proportional Integral (PI) controller. They provided guidelines to design these stable linear controllers. Then they verified their guidelines through non-linear simulations and proved it to be a robust controller. Then came Jinsheng Sun, Guanrong Chen, King-Tim Ko, Member, Sammy Chan and Moshe Zukerman[4] who described a Proportional-Derivative (PD) control algorithm as a new Active Queue Management scheme for TCP/IP congestion control. Its performance is extensively evaluated by simulations. The result demonstrate that the PD-Controller AQM is stable and robust against traffic load fluctuations, UDP and HTTP disturbances. Then Azadegan and M.T.H. Beheshti[10] proposed a new Proportional Integral Derivative (PID) congestion controller design for transmission control protocol It could efficiently control the TCP traffic. They did it via the use of only the queue length at router needless to know the TCP window size which is not accessible at the router side. The stability condition was presented by applying linear matrix inequality (LMI).

Wei Zhang, Liansheng Tan, Gang Peng[1] applied a time-delay control theory to analyze the mechanism of packet dropping at router and the window updating in TCP source in TCP congestion control for a TCP/RED model. They derived explicit conditions under which the TCP/RED system is asymptotically stable. They discussed the convergence of the buffer queue lengths in the router. Their results suggested that if the network parameters satisfy certain conditions, the TCP/RED system is stable and its queue length can converge to any target. Further, Jinsheng Sun, Sammy Chan, King-Tim Ko, Guanrong Chen and Mosche Zukerman [11], introduced a novel and robust active queue management scheme based on a Proportional Integral-

Derivative(PID) controller, called Neuron PID that uses an adaptive neuron to tune its parameters. They demonstrated the simulations that showed Neuron PID stabilized the router queue length regardless of the round trip delay. M.Yaghoubi Waskasi, M.J. Yazdanpanah, N. Yazdani [12] in their work, they applied an adaptive Proportional Integral controller based on Artificial Neural Networks(ANN) to AQM for the objective of congestion avoidance and control in middle nodes. The proposed controller is simple and can be easily implemented in high speed routers. It adapts its parameters dynamically with respect to change in the system. Which results in better response compared to linear controllers due to the nonlinear nature of New Neuron PI. Jinsheng Sun, Sammy Chan, Moshe Zukerman [13] proposed an Intelligent Adaptive Proportional Integral controller (IAPI) whose Parameter values is provided based on previous details on PI and additional empirical studies. The performance of IAPI was evaluated by simulations and the results demonstrated that it was stable and robust under various scenario including a case involving a multiple bottleneck.

Chapter 4

Active Queue Management

4.1 Introduction

In this section, we will review existing proposals for AQM algorithms. There is a large body of AQMs proposed and we will limit our review to some key AQMs which represent the key paradigms. In following chapter, we will analyze the key AQMs in more detail. Let us have a closer look into the structure of an AQM algorithm as shown in Fig. 4.1. The AQM algorithm controls the arrival rate of packets into the queue, by ECN marking or packet dropping to generate the congestion signal that controls the source rate. In the optimizations problem analogy, determining the right feedback signal is akin to finding the right price that controls the demand of the sources to match to the target link capacity. Different AQMs solve the underlying utility optimization problem, however with differences in properties such as steady state backlog in the link, speed of converge, transient response, and stability regions. We will now examine these differences and discuss the aims of AQMs in general. The aims of flow control include (a) fairness (b) utilization (c) quality of service (d) good transient properties and (e) scalability.

Fairness principally refers to the rate allocation that occurs in steady state, and is chiefly controlled by the way links signal congestion, and the source algorithms. For the Internet, fairness is utility maximization. Utilization refers to the AQM's ability to drive the link at some target capacity, and controls application throughput. However, the greater the utilization, the greater the queuing delays experienced high utilization, causing large queuing delays, and explore new proposals for AQMs which can better control utilization and delay. In the context of AQM design, Quality of Service (QoS) means packet delay and packet loss. We have discussed the link between utilization and delay. One way packet loss occurs is when the buffer overflows, so it is important for the AQM to have good dynamic properties, and quickly control the source rate to the target capacity. If the controller is too slow, the network may experience underutilization or buffer overflows when the transmission rate is not matched to the capacity. If the controller is too fast, it may become unstable, and cause rates to oscillate. For the Internet to be scalable and fully distributed, the AQM algorithm must only use local information to achieve the listed aims. Therefore to determine the appropriate dropping or marking rate M, the AQM can only observe information such as the packet arrival rate X, the backlog in the queue B, and link capacity C. The problem in designing an AQM algorithm is how to combine this state information to generate the appropriate feedback signal, so that the source rates can be controlled quickly to achieve the desired utilizations of the link.

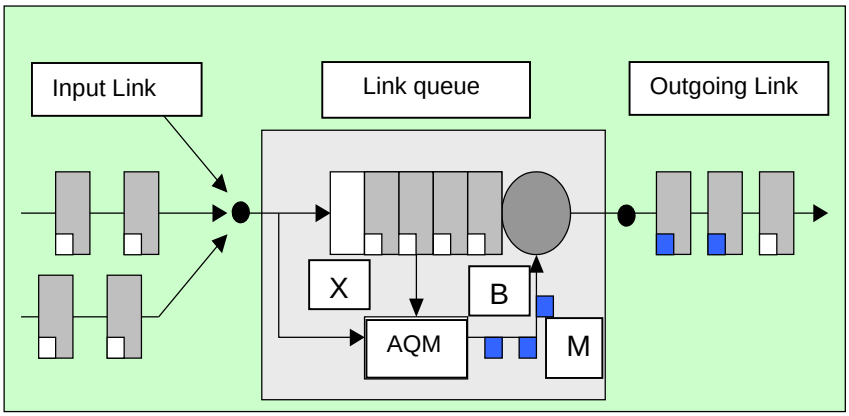

Figure 4.1 Active Queue Management

4.2 Passive Queue Management

PQM is the traditional way for controlling queue length at the routers; it is still one of the popular configurations used due to its simplicity. In PQM, a maximum length is set for each queue at the router and this accommodates all arriving packets till the maximum queue length is reached. Once the buffer overflows, packets are dropped until the queue size is again below the maximum as a result of some departures. PQM was a popular queue management technique for many years due to its simplicity, although it results in some serious disadvantages such as the global synchronization problem and lock-out phenomena [14]. There are many PQMs introduced in the literature and the most well-known algorithm is Drop Tail (DT). There are other PQM algorithms similar to Drop-Tail that can be applied when the queue becomes full such as Drop Front (DF) and Random Drop (RD) respectively

4.2.1 Drop Tail (DT)

Drop tail is the traditional way for controlling the queue length at routers and the most frequently used in network gateways [15,16]. In this mechanism a maximum length is set for each queue at the router which accommodates all arriving packets until the maximum queue length is reached. Once the buffer overflows packets start dropping until the queue size is again below the maximum as a result of some departures [17]. The DT algorithm reacts only when the buffer is full, which causes a high packet loss rate, especially since the buffer maybe full most of the time, and this also results in long delays. Global synchronization occurs when all connections slow down their transmission rate at the same time when several packets are dropped over a long period as a result of the full buffer. When connections decrease the sending window size at the same time, the buffer will be empty for a time interval which results in low link utilization. Also the DT technique suffers from the lock-out phenomenon where the queue is monopolized by some unresponsive connection and other connections may

not easily use the queue. The DropTail mechanism interacts badly with interactive network applications because the DropTail queues are always full or close to full most of time and packets are continuously dropped when the queue capacity is reached and this leads to poor performance. Nowadays, the use of interactive applications over the Internet, such as voice and video conferencing has increased. These applications require low end-to-end delay and jitter. When the buffer is almost full for long periods of time, the delay will of course be large which makes the DT algorithm illustrated for such applications. As Drop Tail is widely used because of its simplicity and the low cost of management, in chapter four these advantages of DT will be used to develop a new technique to maintain the delay and meet multimedia application requirements.

4.2.2 Drop Front

The Drop Front algorithm is similar to the Drop Tail algorithm as both mechanisms drop packets when the queue is full. The Drop Front algorithm drops packets from the queue when the buffer capacity is reached while Drop Tail drops packets from the tail of the queue.

Drop Front on full, drops a packet at the front of the queue when a new packet arrives into the full queue. This algorithm solves the lock-out phenomenon and this is the only advantage that Drop Front offers over Drop Tail. On the other hand, similar to the Drop Tail algorithm, Drop front suffers from long delay, global synchronization and other problems caused by full queues.

4.2.3 Random Drop

The Random Drop algorithm drops packets randomly when the buffer is full in an attempt to make a space for new arriving packets. The Random Drop algorithm is a complex algorithm in comparison with Drop Tail and drop front as this algorithm requires connection management as the probability of a packet being dropped from a specific connection is proportional to the percentage share of that connection in the router buffer. Once a packet is dropped randomly the queue has to shift packets forward, leaving a space at the end of the queue for the new arriving packet this process results in a huge amount of computation. After all, this algorithm could drop packets that have spent some time already in the queue waiting for service just to make a space for another new packet which is an obvious waste of processing time. As explained in, Random Drop lacks fairness where connections unresponsive to congestion keep their high sending rate and monopolize the queue, leaving a small share of bandwidth to slow and responsive connections [1]. Random Drop is more complicated than Drop Tail and Drop Front yet still can cause a full buffer for a long time, which results in delay and jitter. Moreover random drop wastes processing time explained in, Random Drop lacks fairness where connections unresponsive to congestion keep their high sending rate and monopolize the queue, leaving a small share of bandwidth to slow and responsive connections [1]. Random Drop is more complicated than Drop Tail and Drop Front yet still can cause a full buffer for a long time, which results in delay and jitter. Moreover

random drop wastes processing time. Also the DT technique suffers from the lock-out phenomenon where the queue is monopolized by some unresponsive connection and other connections may not easily use the queue. The Drop Tail mechanism interacts badly with interactive network applications because the Drop Tail queues are always full or close to full most of time and packets are continuously dropped when the queue capacity is reached and this leads to poor performance. Nowadays, the use of interactive applications over the Internet, such as voice and video conferencing has increased. These applications require low end-to-end delay and jitter. When the buffer is almost full for long periods of time, the delay will of course be large which makes the DT algorithm ill-suited for such applications. As Drop Tail is widely used because of its simplicity and the low cost of management, in chapter four these advantages of DT will be used to develop a new technique to maintain the delay and meet multimedia application requirements.

4.3 Active Queue Management (AQM)

Traditional queue management suffers from many drawbacks which seriously affects the performance of networks and in many cases does not meet the required QoS. Active Queue Management (AQM) was proposed in order to solve the above- mentioned problems caused by DT and other traditional queue managements. The solution to the full queues problem caused by DT is to drop the packets before the queue becomes full and subsequently senders slow down their transmission rate because of this congestion notification, thus preventing the queue from overflowing. In other words, by using AQM mechanisms, sources are informed early about congestion and can react accordingly. AQM is a pro-active approach, and is highly recommended for the Internet intermediate devices. AQM has been an active research area in the last two decades due to the rapid growth of the Internet and the high demand for QoS in some sensitive applications. A large number of AQM algorithms have been introduced in the literature for different purposes. The main objectives of AQM are to achieve high throughput and low average queuing delay in the network. AQM algorithms can be classified as follows: stabilizing router queues, approximating fairness among flows, controlling unresponsive high-bandwidth flows, and improving performance for short connections.

4.3.1 Random Early Detection (RED)

Random Early Detection (RED) is an AQM algorithm that was first introduced by Jacobson and Floyd [17], and since then it has been the cornerstone of most studies of AQM. RED has led to a substantial increase in the efficiency of using the network resources. Although some recent studies [18,19] show that the original RED has serious performance problems, due to the improvement of network performance RED has been recommended by the Internet Engineering Task Force (IETF) as the default active queue management scheme in Internet routers, with the following statement. "Internet routers should implement some active queue management mechanism to manage queue lengths, reduce end-to-end latency, reduce packet dropping, and avoid lock-out phenomena within the Internet. The default mechanism for managing queue lengths to meet these goals is RED. Unless a developer has reasons to provide another equivalent mechanism, we recommend that RED be used."

In recent study Floyd and Kohler [20] introduced the Datagram Congestion Control Protocol (DCCP). CCID 4 uses TCP- Friendly Rate Control for Small Packets (TFRC- SP) a variant of TFRC designed for applications that intend to send small size packets. The idea behind using TFRC-SP is to produce a similar bandwidth as TCP flow using packets up to 1500 bytes with less congestion. CCID 4 uses for senders that send small packets and would like TCP-Friendly sending rate possibly with (ECN), while changing abrupt rate. However CCID 4 is still for experimental use according to Floyd and Kohler. Random Early Detection, from its name, has two important objectives. First, it tries to keep the queue away from the congestion limit by using some signals as the indicators of congestion. Secondly, it drops packets randomly according to a probability function which increases as the average queue size increases. This ensures that RED is not biased against any connection or any bursty traffic

and stops greedy users from monopolizing the queue. Moreover, RED provides a better notion of the congestion since it drops packets randomly according to the average queue size rather than the instantaneous queue size. Thus, if the average queue size is high, there will be a high probability of persistent congestion. Also, RED overcomes the disadvantage of traditional queue management such as eliminating global synchronization and solves the high latency problems by controlling the average queue length.

The RED algorithm consists of two main parts, the first step is the calculation of the average queue size *AVG-Q*, and the second step is to make a decision on whether to drop the incoming packet or not by calculating the drop probability *Pa*. For each arrival of a new packet to the queue, the algorithm in Figure 3.1 is performed. According to this algorithm the average queue size *AVG_Q* is calculated upon the receipt of a new packet then *AVG_Q* is compared to the queue's two thresholds L_1 and L_2. If *AVG-Q* is more than the second packet is dropped immediately; else if *AVG-Q* is between the first threshold the L_1 , and threshold queue value, L_2 , the arriving the second threshold the L_2 , and the packet is dropped with a probability *Pa*. If the *AVG- Q* is less than the first threshold L_1, the packet is queued.

1. If a packet arrives do
2. Calculate the Average Queue Size *AVG_Q*
3. If *AVG_Q* ≥ L_2 then
4. Drop the packet
5. Else if L_2 ≤ *AVG_Q* ≤ L_2 then
6. Calculate drop probability P_a
7. Drop the packet with probability P_a
8. End if
9. End

4.3.2 Adaptive Random Early Detection (ARED)

The RED algorithm and Gentle RED can achieve high throughout and also low average delay. However, in these algorithms the *AVG_Q* is very sensitive to the level of congestion and parameter configuration. For instance, if the max_p is high or the congestion is light, the average queue size is close to the first threshold, while average queue size is close to the second threshold when the max_p is small or the congestion is heavy. Unfortunately, RED and Gentle RED do not predict the average queue size in advance due to the static configuration of the $max_{p.}$ Adaptive RED was first introduced by Feng et al 1997 [21]. In 2001 Floyd, Gummadi, and Shenker [18] made some algorithmic modifications to Feng's proposal, at the same RED gateways to remove the difficulties with parameter settings that are raised in RED and which affect performance. Moreover, ARED can achieve a precise target average queue size in different traffic scenarios. The principal ideas behind ARED [18] is to focus on dynamically adjusting the RED control parameter, *Pmax*, according to the changes in the network load, thus giving a more stable queue size, which means a predictable queuing delay. A target queue length is first set normally at midway between the maximum and the minimum queue size. ARED's algorithm as shown below in figure (4.3) where the *Pmax* is calculated every time

interval.

1. Every 0.5 seconds do
2. If *AVG_Q* > L and *Pmax* ≤ 0.5 then
3. Incase *Pmax* =*Pmax*+ α
4. Else if *AVG_Q* < L and *Pmax* ≥ 0.1 then
5. Decrease *Pmax* =*Pmax* * β
6. End if
7. End do

According to the above algorithm, when the average queue size is more than the target queue value L, ARED becomes more aggressive by increasing the maximum drop probability by the factor α so that *AVG_Q* size reduces back to the target value. Conversely, when the average queue size is smaller than the target queue value L, ARED becomes more conservative by decreasing the maximum drop probability by a factor β so that the sources can send more packets. Thus ARED has added more parameters to RED, which are α, β and the target value L.

4.2 PI: Proportional Integral Controller

A methodology to efficiently calculate the packet drop Probability and that was the base of new kind of AQM. According to it, Probability of packet drop depends upon present and past values of queue length.

$$p(t) = K_p * e(t) + K_i * e(t-1) + p(t-1)$$

Where,
K_p : Proportional gain
K_i: Proportional integral gain
$p(t)$: Probability to drop packet at time t
$e(t)$: Current error
$e(t-1)$: Previous error

PI controller [2] will eliminate forced oscillation and steady state error resulting operation of on-off controller and P controller respectively. However, introducing integral mode has a negative effect on speed of the response and overall stability of the system. This, PI controller will not increase the speed of response. It can be expected since controller does not have means to predict what will happen with the error in near future. The problem can be solved by introducing derivative mode which has ability to predict what will happen with the error in near future and thus to decrease a reaction time of the controller.

4.2.4 PID: Proportional Integral Derivative Controller

A new kind of AQM by taken into consideration of present past and future error in queue length and calculate the packet drop probability.

$$p(t) = K_p * e(t) + K_i * e(t-1) + K_d * e(t-2) + p(t-1)$$

Where,
K_p: Proportional gain
K_i: Integral gain
K_d: Derivative gain
$p(t)$: Probability to drop packet at time t
$e(t)$: Current error
$e(t-1)$: Previous error
$e(t-2)$: Previous to previous error

PID [10] controller has all the necessary dynamics fast reaction on charge of the controller input, increase in control signal to lead error towards zero and suitable action inside control error are at eliminate oscillation. Derivative mode improves stability of the system and enables increase in gain K and decrease in integral time constant Ti, which increases speed of the controller response. However, the problem is it cannot predict the packet drop probability in a non-linear system.

4.2.5 REM: Random Exponential Marking

REM [8] differs from RED [1] only in two design questions: it uses a different definition of congestion measure and a different marking probability function.

The first idea of REM is to stabilize both the input rate around link capacity and the queue around a small target, regardless of the number of users sharing the link. Each output queue that implements REM maintains a variable we call price as a congestion measure. This variable is used to determine the marking probability, as explained in the next subsection. Price is updated, periodically or asynchronously, based on rate mismatch *(i.e., difference between input rate and link capacity) and queue mismatch (i.e., difference between queue length and target). The price is incremented if the weighted sum of these mismatches is positive, and decremented otherwise. The weighted sum is positive when either the input rate exceeds the link capacity or there is excess backlog to be cleared, and negative otherwise. When the number of users increases, the mismatches in rate and in queue grow, pushing up price and hence marking probability. This sends a stronger congestion signal to the sources which then reduce their rates. When the source rates are too small, the mismatches will be negative, pushing down price and marking probability and raising source rates, until eventually, the mismatch are driven to zero, yielding high utilization and negligible loss and delay in equilibrium. The buffer will be cleared in equilibrium if the target queue is set to zero.

The second idea of REM is to use the sum of the link prices along a path as a measure of congestion in the path, and to embed it into the end-to-end marking probability that can be observed at the source. The output queue marks each arrival.

However, one thing is clear that no algorithm has been successful in achieving small queuing delay, small packet loss ratio and small round trip time. This is surprising in terms of the importance of these matrices in real-life multimedia applications. The rest of the project concentrates on filling up this gap and maintaining the delay, packet loss ratio and round trip time by developing two new algorithms through a buffer that uses dynamic threshold the position of which is linked to the delay via analytic equations that have been developed.

4.2.6 BLUE

The BLUE [5] algorithm proposed by Feng et al. uses packet loss and eventually occurring idle times of the output link as indicator of congestion and tries to prevent high loss rates and to reduce the queue length oscillations. The main idea is to increase the marking probability upon each packet's arrival event by d_1 and to decrease it upon each idle event by d_2. If the time elapsed between two events is larger than a time-period called *freeze-time* d_1, e.g. by factor of 10. The *freeze-time* parameter should be randomized to avoid global synchronization.

An extension to BLUE called Stochastic Fair BLUE use Bloom Filters to identify nonresponsive flows with small space consumption. The approach uses L levels with N bins on each level. L independent hash functions map each flow identifier to L bins, one bin per layer. Each arriving packet increases the size of its bins on all levels. If a bin overflows the dropping probability assigned to it is increased or decreased if the bin becomes empty. The dropping probability of a packet a minimum of dropping probabilities of all its bin. A flow with marking probability of one is considered to be non-responsive and its transmission rate is limited. As flows share some bins there may be some false-positives which in turn can be alleviated by exchanging the hash functions periodically. Setting proper values for L and N is an open question.

Chapter 5

Neural Network

5.1 Introduction

Inspired from the natural neural network of human nervous system but do not actually simulate neurons. ANN typically contains many fewer than the 1011 neurons that are in the human brain. Dr. Robert Hetch-Nielsen (inventor of 1st neurocomputer) defines ANN as:

"A computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs. "

ANNs are composed of multiple nodes, which imitate biological neurons of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its activation or node value. The neural network structure can be viewed in figure 5.1

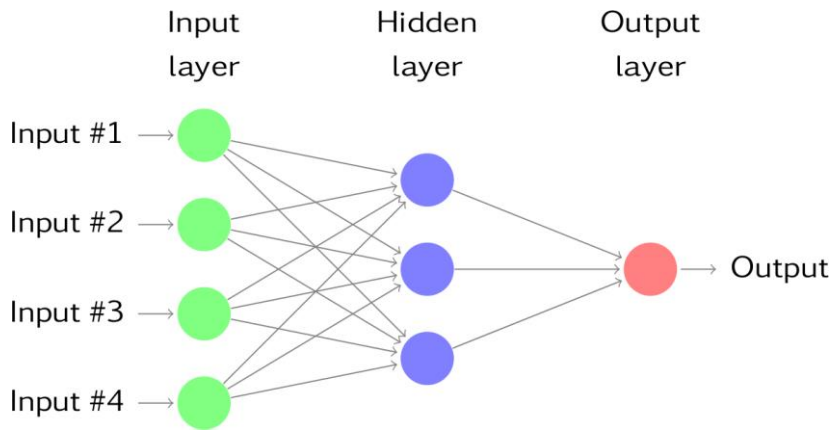

Figure 5.1 Neural network structure

5.2 Advantage of ANN over conventional computing:

- Parallel Processing:

In traditional computers processing is sequential. The artificial neural network (ANN) is an is an inherently multiprocessor-friendly architecture (Without much modification, it goes beyond one or even two processors of the von Neumann architecture.) The neural network can accomplish a lot in less time

- Learning Strategy:

Traditional computers have to learn by rules, while artificial neural networks learning Example, by doing something and then learning from it (i.e. self-learning).

- Self-Programming:

The conventional computers learn by doing different sequences or steps defined in an algorithm, neural networks are continuously adaptable by truly altering their own programming. It could be said that conventional computers are limited by their parts, while neural networks can work to become more than the sum of their parts.

5.3 Applications of neural networks:

Neural Network is applied in the field of Character Recognition. The neural network can be used to recognize handwritten characters. Also, in Pattern Recognition such as in radar Systems and face identification, etc. Further in, Data processing and Compression. Also in Stock market prediction. As the day-to-day business of the stock market is extremely complicated. Many factors weigh in whether a given stock will go up or down on any given day. Since neural networks can examine a lot of information quickly and sort it all out, they can be used to predict stock prices. Also, function approximation and medical diagnosis etc.

5.4 Basic Concepts

5.4.1 Learning Strategy

Learning Strategy is the network's approach to make decisions based on observations and previous outcomes. It is an essential feature of neural network that makes self-learning possible.

1. Supervised Leaning: It involves a teacher that is smarter than ANN itself. For example, let's take the facial recognition example. The teacher shows the network a bunch of faces, and the teacher already knows the name associated with each face. The network makes its guesses, then the teacher provides the network with the answers. The network can then compare its answers to the known "correct" ones and make adjustments according to its errors.

2. Unsupervised Learning: Required when there isn't an example data set with known answers. Imagine searching for a hidden pattern in a data set. An application of this is clustering, i.e. dividing a set of elements into groups according to some unknown pattern.

3. Reinforcement Learning: This Strategy is built on observation. The ANN makes a decision with an outcome and observes its environment. If the Observation is negative, the network can

adjust its weights in order to make a different decision the next time. Reinforcement learning is common in robotics. At time t, the robot performs a task and observes the results. Did it crash into a wall or fall off a table? Or is it unharmed?

5.4.2 Activation function:

The activation function of a node defines the output of that node given an input or set of inputs. In artificial neural networks this function is also called transfer function (not to be confused with a linear system's transfer function).The output of a neuron (y) is a function of the weighted sum $y = f(v)$ is called as activation function. Some of the activation functions are:

1. Linear function:

$$\boldsymbol{f(v) = a + v = a + \sum_{i=0}^{m} wi * xi}$$

Where, a is called a bias

2. Step Function

$$\boldsymbol{f(v) = \begin{cases} 1 & if\ v \geq a \\ 0 & otherwise \end{cases}}$$

Where, a is called threshold

5.5 Types of Neural Network

1. Feed Forward ANN: Perceptron are arranged in layers and allows the information/signal to flow from input to output layer (i.e. unidirectional). There is no connection among perceptron in the same layer. There is no feedback (or loops) .i.e. output of any layer does not affect the same layer. They are extensively used in pattern recognition /generation. It is also referred as top-down or bottom-up ANN.

2. Feedback ANN: Feedback networks can have signals travelling in both directions by introducing loops in the network. More complicated and powerful. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent

5.6 Back Propagation of errors (or Back Propagation) Algorithm:

Back propagation is a common method of training artificial neural network and used in conjunction with an optimization method (such as gradient descent). The algorithm repeats a two phase cycle, propagation and weight update. When an input vector is presented to the network, it is propagated forward through the network, layer by layer, until it reaches the output layer. The output of the network is then compared to the desired output using loss function, and an error

value is calculated for each of the neurons in the output layer. The error values are then propagated backwards, starting from the output, until each neuron has an associated error value which roughly represents its contribution to the original output. Back propagation uses these error values to calculate the gradient of the loss function with respect to the weights in the network. In the second phase, this

Algorithm:

The back propagation learning algorithm can be divided into two phases: propagation and weight update.

PHASE I: Propagation

Each propagation involves the following steps:

1. Forward propagation of a training pattern's input through the neural network in order to generate the network's output value(s).
2. Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.

PHASE II: Weight Update

For each weight, the following steps must be followed:

1. The weight's output delta and input activation are multiplied to find the gradient of the weight.
2. A ratio (percentage) of the weight's gradient is subtracted from the weight.

This ratio (percentage) influences the speed and quality of learning; it is called the learning rate. The greater the ratio, the faster the neuron trains, but the lower the ratio, the more accurate the training is.

Loss function:

The loss function is a function that maps values of one or more variables onto a real number intuitively representing some "cost" associated with the event. For back propagation, the loss function calculates the difference between the input and its Expected output.

Gradient descent:

Gradient descent is a first-order iterative optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point.

Chapter 6

Proposed Work

6.1 Introduction

One of the inherent weaknesses of some of the existing proposed active queue management schemes is that congestion notification does not directly depend on the number of connections multiplexed over the link. In order for early detection to work in congested networks congestion notification must be given to enough sources so that the offered load is reduced sufficiently to avoid packet loss due to buffer overflow.

We have divided our work into two parts. In part 1, we have proposed a new AQM technique called Proposed AQM. In our part 2, we have modified the existing PID controller to develop MPID. Then we have used an adaptive neuron with MPID to develop AN-MPID. Further, we have used activation functions to develop AN-MPID (Act-1), AN-MPID (Act-2) and AN-MPID (Hybrid). The proposed algorithms are implemented in NS2 simulator. The algorithm are analysed with the performance metrics such as Queue length(packets), Packet drop, Packet receive, Average delay (ms), Average Queue length (packets), Average throughput and RTT. And then compared the Results of proposed AQM with the existing AQM techniques.

6.2 Proposed AQM

6.2.1 Description

Proposed AQM controller based AQM can be described as:

$$p(t) = K_p * e(t) + K_d * \big(e(t-1) - e(t)\big) + p(t-1)$$

where, $p(t)$ is the packet mark/drop probability. And K_p is the Proportional gain. K_d is the Derivative gain. The gains are multiplied to the current error, $e(t)$ and the previous error, $e(t-1)$ respectively. We calculate the value of the current error and the previous error with respect to the queue length.

Where,
$e(t) = q_{len} - q_{ref}$

Here, $e(t)$ is the current error in queue length, q_{len} is the current queue length, q_{ref} is the targeted or the reference queue length. Our objective is to keep the q_{ref} as close as to our q_{ref}.We calculate the previous error

$$e(t-1) - e(t) = q_{old} - q_{len}$$

Here, $e(t-1)$ is the previous error in queue length, q_{old} is the previous value of queue length, and q_{len} is the current queue length. The values of $e(t)$ and $e(t-1)$ are further incorporated to calculate the drop probability. We need to calculate the drop probability at every packet arrival to keep the queue length closer to the reference queue length and avoid massive fluctuation. We can sum up the working of Proposed AQM in the following two main functions called calculate_p() and enqueue() of the algorithm.

/* Parameters */
$p(t)$ The calculated packet drop probability
p_{old} Previous packet drop probability
K_p The proportional gain constant
K_d The derivative gain constant
q_{ref} Targeted queue length
q_{len} Current queue length
q_{old} Previous queue length
$p(t-1)$Previous drop probability
$e(t)$ Current queue length error
$e(t-1)$ Previous queue length error
U Random variable

calculate_p() /*Called at every tm seconds */

1. $e(t) = q_{len} - q_{ref}$
2. $e(t-1) = q_{old} - q_{ref}$
3. $e(t-1) - e(t) = q_{old} - q_{len}$
4. $p(t) = K_p e(t) - K_d * ((e(t-1) - e(t)) + p(t-1)$
5. if $(p < 0)$
6. $p = 0;$
7. if $(p > 1)$
8. $p = 1;$
9. $p(t-1) = p(t);$
10. $q_{old} = q_{len}$
11. return $p;$

enqueue()/*Called at each packet arrival*/

1. $p(t) = p(t-1) * qlen/$qref
2. $U(t)$ = Random:: uniform()
3. If $(U > p)$
4. Enqueue the packet
5. Else
6. Drop the packet

6.3 MPID

In this section we have explained our proposed AQM named MPID and stated the packet drop probability $p(t)$.
The A PID controller-based AQM can be described by

$$\mathrm{p(k)} = k_p * e(t) + k_i * \sum_{i=0}^{n} e(i) \; + \; k_d * \big(e(t) - e(t-1)\big) \ldots\ldots\ldots (1)$$

The traditional PID controller do not withstand non-linearity. Further, it fails to give efficient performance in time varying systems.

Hence to overcome this limitation, we have modified equation (1) to make the controller flexible & efficient even in non-linear and time varying systems. The modification gives arise to the new equation of over new AQM controller MPID. The packet dropping /marking probability of MPID controller-based AQM can be expressed as:

$$p(t) = p(t-1) + k_p * e(t) + k_i * e(t-2) + k_d * (e(t-2) - e(t-1)) \ldots (4)$$

We can explain the equation (4) as follows:

$p(t)$ is the packet dropping/marking probability

$p(t-1)$ is the previous packet dropping/marking probability which is initially taken as zero. Proportional gain k_p , Integration gain k_i and Derivative gain k_d are the parameters of the controller. The queue length error at time t in equation (4) is given as

$$e(t) = q(t) - q_{ref}$$

Where, $q(t)$ is the instantaneous queue length and q_{ref} is the targeted or reference queue length.

The previous queue length error at time $\mathrm{t} - 1$ in equation (4) is given as

$$e(t-1) = q(t-1) - q_{ref}$$

Where $q(t-1)$ is the queue length at time $t-1$.

The previous to previous queue length error at time $t-2$ in equation (4) is given as

$$e(t-2) = q(t-2) - q_{ref}$$

Where $q(t-2)$ is the queue length at time $t-2$.

So, $e(t-2) - e(t-1)$ in equation (4) becomes

$$e(t-2) - e(t-1) = q(t-2) - q_{ref} - q(t-1) + q_{ref}$$

$$= q(t-2) - q(t-1)$$

In our controller, we have taken the values of the parameters and terms as:

6.4 AN-MPID

In this section, we have used an adaptive neuron in our MPID to develop a new AQM algorithm named AN-MPID. A MPID controller based AQM scheme can be described as by:

$$p(t) = p(t-1) + k_p * e(t) + k_i * e(t-2) + k_d * (e(t-2) - e(t-1)) \ldots (1)$$

Our MPID shows efficient results under varying conditions. It brings the queue length closer to the targeted queue length at a faster rate. Yet, we are looking for a controller than can withstand heavy load or connections and still give the same efficient result.

Thereby we have come up with a solution. We have used an adaptive neuron to tune in the parameters of the existing controller considering the difference between the current queue length and the desired queue length. We formulated a controller called AN-MPID.

Contrast to equation (1), we include a neuron output S(t) , which can be written as

$$S(t) = K * \sum_{i=0}^{n} w_i(t) * x_i(t) \ldots\ldots\ldots (2)$$

Where,

$K > 0$ is the neuron proportional coefficient,

$x_i(t)$ (i=1, 2,.....m) denotes the neuron inputs and

$w_i(t)$ are connection weights of $x_i(t)$ determined by learning rule.

According to Hebb [22],

$$w_i(t+1) = w_i(t) + d_i * y_i(t) \ldots\ldots (3)$$

In equation (3), we can describe each term as

$w_i(t+1)$ is the connection weight at $t+1$. $w_i(t)$ is the connection weight at t.

$d_i > 0$ is the learning rate, i.e., the rate at which our neuron learns and adapts when fed with inputs.

$y_i(t)$ is the learning strategy, i.e., the actions that are consciously deployed by our neural system to learn.

The learning strategy in equation (3) can be defined as

$$y_i(t) = z(t) * S(t) * x_i(t)$$

Where $z(t)$ is teacher signal.

Our neuron self learns the surrounding or fed information following the learning strategy under the guidance of teacher signal.

The neuron inputs thus becomes,

$$x_1(t) = e(t)$$

$$x_2(t) = e(t-1)$$

$$x_3(t) = e(t-2) - e(t-1)$$

And the gains are,

$w_1(t) =$ Adaptive Proportional gain,

$w_2(t)$ =Adaptive Integration gain and
$w_3(t)$ =Adaptive Derivative gain.

So, equation (1) becomes

$$p(t) = p(t-1) + \frac{\left[K * \sum_{i=0}^{3} w_i(t) * x_i(t)\right]}{\sum_{i=0}^{3} w_i(t)}$$

$$w_i(t+1) = w_i(t) + d_i * \mathrm{e}(t) * p(t) * x_i(t)$$

Where,
$\mathrm{e}(t)$ used as teacher signal $\mathrm{z}(t)$
$d_1 = k_p$
$d_2 = k_i$
$d_3 = k_d$
In our Neuron MPID controller we have used the values of parameters as:
$w_1 = w_2 = w_3 = 0.001$ and K=0.01

6.5 AN-MPID (Act-1)

In this section, we used an activation function with our previously proposed AQM, AN-MPID to develop a new algorithm named as AN-MPID (Act-1).

A MPID controller based AQM scheme can be described as by
$p(t) = p(t-1) + k_p * e(t) + k_i * e(t-2) + k_d * (e(t-2) - e(t-1))$..... (1)

Implementation of neural networks in our controller helps in making conscious and cognitive decision. Information is contained in the overall activation state of the network. Knowledge is thus represented by the network itself, which is quite literally more than the sum of its individual components. We use neural network concepts to withstand the non-linearity feature of networks in real life. Thus by introducing an adaptive neuron in our existing controller we come up with MPID N2 algorithm. And hence to maintain the multi-layer network, we incorporate activation functions. We can see the implementation of adaptive neuron in our controller in figure 6.1

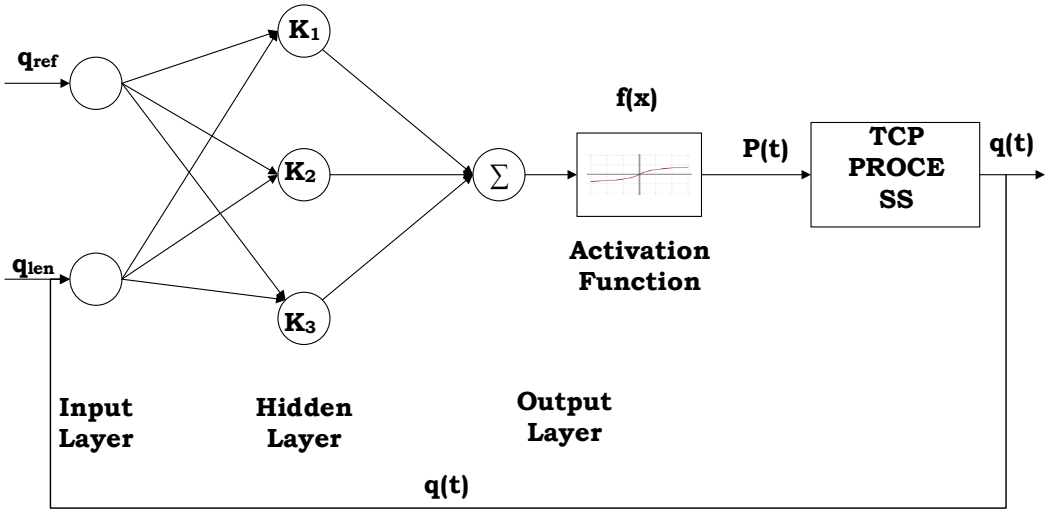

Figure 6.1 Our Neural Network Model

The activation function used in this algorithm is the hyperbolic function which is defined as

$f(x) = \tanh(kx) = \frac{1-e^{-2kx}}{1+e^{-2kx}}$, $f'(x) = 1 - f(x)^2$

The packet dropping/marking probability function becomes,

$p(t) = f(x)$…….. (2)

Where ,

$x = k_p * e(t) + k_i * e(t-2) + k_d * (e(t-2) - e(t-1))$

Now,

$k_{pnew} = k_{pold} - \eta_p \frac{\partial E}{\partial k_p}$

$k_{inew} = k_{iold} - \eta_i \frac{\partial E}{\partial k_i}$

$k_{dnew} = k_{dold} - \eta_d \frac{\partial E}{\partial k_d}$

Where, $E = \frac{1}{2}\sum\big(e(t)\big)^2 = tuning\ function$

$\frac{\partial E}{\partial k_p} = \frac{\partial E}{\partial q}\frac{\partial q}{\partial p}\frac{\partial p}{\partial x}\frac{\partial x}{\partial k_p} = -\big(q - q_{ref}\big)\frac{\partial q}{\partial p}f'(x)e(t)$

$\frac{\partial E}{\partial k_i} = \frac{\partial E}{\partial q}\frac{\partial q}{\partial p}\frac{\partial p}{\partial x}\frac{\partial x}{\partial k_i} = -\big(q - q_{ref}\big)\frac{\partial q}{\partial p}f'(x)e(t-2)$

$\frac{\partial E}{\partial k_d} = \frac{\partial E}{\partial q}\frac{\partial q}{\partial p}\frac{\partial p}{\partial x}\frac{\partial x}{\partial k_d} = -\big(q - q_{ref}\big)\frac{\partial q}{\partial p}f'(x)(e(t-2) - e(t-1))$

And,

$\frac{\partial q}{\partial p} = 1$, $\eta_p = 3 \times 10^{-8}$, $\eta_i = 2 \times 10^{-8}$, $\eta_d = 2 \times 10^{-8}$

So finally we define the packet dropping/marking probability of AN-MPID (Act-1) as,

$$p(t) = p(t-1) + k_{pnew} * e(t) + k_{inew} * e(t-2) + k_{dnew} * (e(t-2) - e(t-1))$$

6.6 AN-MPID (Hybrid)

In this section we have used another activation function called soft sign to develop a new AQM called AN-MPID (Hybrid). A MPID controller based AQM scheme can be described as by

$$p(t) = p(t-1) + k_p * e(t) + k_i * e(t-2) + k_d * (e(t-2) - e(t-1)) \ldots\ldots (1)$$

When it comes to robustness, we always look for something more. Something more that our controller can give us. And that is exactly where the concept of neural network kicks in. For difficult problem that do not yield the traditional algorithmic approaches, we depend upon neural network. Our congestion in the network is one such fuzzy problem. Hence we have come one with a new algorithm called MPID N02 by self tuning the classical PID.

In this algorithm we have used an activation function. Activation function is the decision function in our neural network. It produces non-linear decision boundary given the combination of the weighted inputs A multilayer neural network without an activation function would be equivalent to a linear regression. And since we need a system to withstand non-linearity, we have introduced a non-linear function. There are many activation functions, popular ones being sigmoid, soft sign etc.

We have used a soft sign function as it is easily differentiable. The function is defined as

$$f(x) = \frac{x}{1+|x|}, \; f'(x) = \frac{1}{(1+|x|)^2}$$

Hence, the packet dropping/marking probability of our controller will be

$$p(t) = f(x) \ldots\ldots (2)$$

We have substituted the value of x with

$$X = k_p * e(t) + k_i * e(t-2) + k_d * (e(t-2) - e(t-1))$$

$$k_{pnew} = k_{pold} - \eta_p \frac{\partial E}{\partial k_p}$$

$$k_{inew} = k_{iold} - \eta_i \frac{\partial E}{\partial k_i}$$

$$k_{dnew} = k_{dold} - \eta_d \frac{\partial E}{\partial k_d}$$

Where, $E = \frac{1}{2}\sum\big(e(t)\big)^2 \quad (tuning\ function)$

$$\frac{\partial E}{\partial k_p} = \frac{\partial E}{\partial q}\frac{\partial q}{\partial p}\frac{\partial p}{\partial x}\frac{\partial x}{\partial k_p} = -\left(q - q_{ref}\right)\frac{\partial q}{\partial p}f'(x)e(t)$$

$$\frac{\partial E}{\partial k_i} = \frac{\partial E}{\partial q}\frac{\partial q}{\partial p}\frac{\partial p}{\partial x}\frac{\partial x}{\partial k_i} = -\left(q - q_{ref}\right)\frac{\partial q}{\partial p}f'(x)e(t-2)$$

$$\frac{\partial E}{\partial k_d} = \frac{\partial E}{\partial q}\frac{\partial q}{\partial p}\frac{\partial p}{\partial x}\frac{\partial x}{\partial k_d} = -\left(q - q_{ref}\right)\frac{\partial q}{\partial p}f'(x)(e(t-2) - e(t-1))$$

And,

$$\frac{\partial q}{\partial p} = 1 \quad , \quad \eta_p = 3 \times 10^{-8} \quad , \quad \eta_i = 2 \times 10^{-8} \quad , \quad \eta_d = 2 \times 10^{-8}$$

So finally we define the packet dropping/marking probability of AN-MPID (Hybrid) as,

$$p(t) = p(t-1) + \frac{[\,k_{pnew} * e(t) + k_{inew} * e(t-2) + k_{dnew} * (e(t-2) - e(t-1))]}{k_{pnew} + k_{inew} + k_{dnew}}$$

Chapter 7

Results and Analysis

7.1 Experimental Setup

In this we analyze the performance of different AQM techniques using NS2 simulator. We have divided our project into two parts. In Part 1, we compare our Proposed AQM with the existing AQM techniques PI, REM, IAPI. In Part 2, we compare the other proposed AQM MPIDN02 with the existing AQM PID, and our proposed AQM MPID, AN-MPID, AN-MPID (Act-1), AN-MPID (Act-2), AN-MPID (Hybrid). We measured queue length, packet send, packet receive and drop, average delay and average throughput of the network. In this a single bottleneck topology is created to analyze the performance of these protocols. We used Gnu plot for plotting different graphs. We have set the number of connections, N=200; Queue reference, q_{ref}=200 packets Sampling time, w=170 sec; Packet size=1000 bytes and Queue limit=1125 packets. And for *Source* to router1 and router2 to *destination*, we have set bandwidth=100Mb, delay=10ms. And from Router1 to Router2, we have set the Bottle neck bandwidth, C=15Mb and delay=20ms. And for REM, we have set Queue/REM set pbo_ 200.0. For AN-MPID, the values of parameters as $k_p = 0.01745, k_i = 0.01732\,, k_d = 0.01742$

7.2 Result Analysis

In this section we summarize the results gathered from our different simulations experiments. The results of AQM techniques are shown below as Part 1 and Part 2.

PART 1: Performance of proposed AQM

In Part 1, a critical comparative study has been made between Proposed AQM and AQM techniques PI, RED, IAPI that have been presented in Chapter 4.This comparison focuses on performance evaluation of the proposed AQM based on performance metrics such as queue length, average queue length, packet send, packet receive, packet drop, delay and average throughput of the mentioned by varying various network parameters. The various network parameters taken are number of TCP connections, N=100 and N=2000. Also, targeted queue length, q_{ref}=100 packets and q_{ref}=500 packets. Also, Bottle neck Bandwidth, C=5 Mb and C=40 Mb. Also, for round trip time RTT=60 ms and RTT=120 ms. The experimental scenario stated in figure 7.40

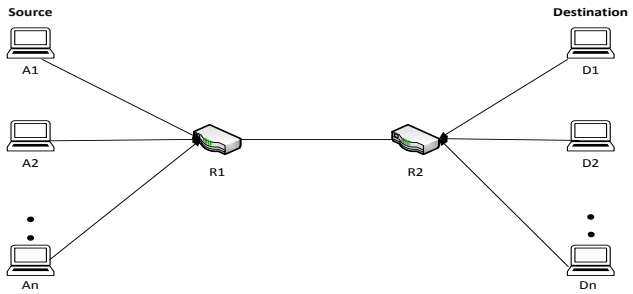

Figure 71: Network Topology

Case 1: Comparison between different AQM techniques

We calculate the queue length of PI controller and find that the queue length takes a lot of time to reach the targeted queue length. And even when it does, it is fluctuating massively around the targeted queue length. This is presented in figure 7.2.

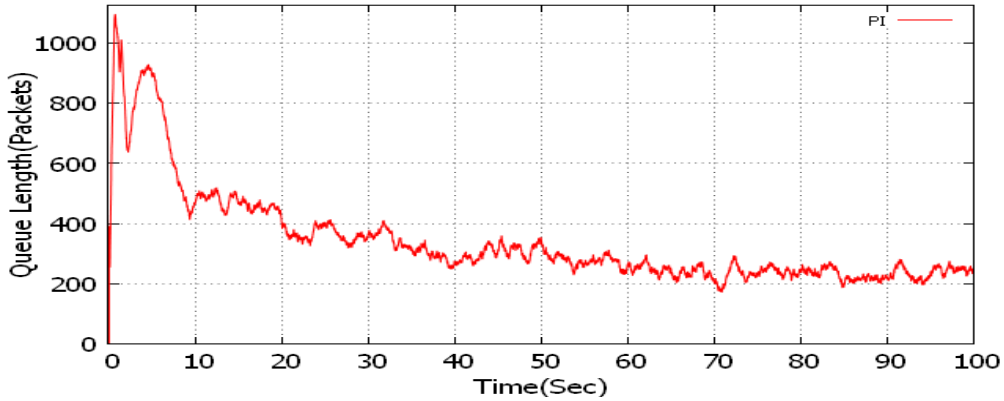

Figure 7.2: Queue length of PI controller

When we calculate the queue length of IAPI controller, we find that the queue length comes closer to the targeted queue length faster than PI controller. Yet it takes a lot of time. This is shown in figure 7.3.

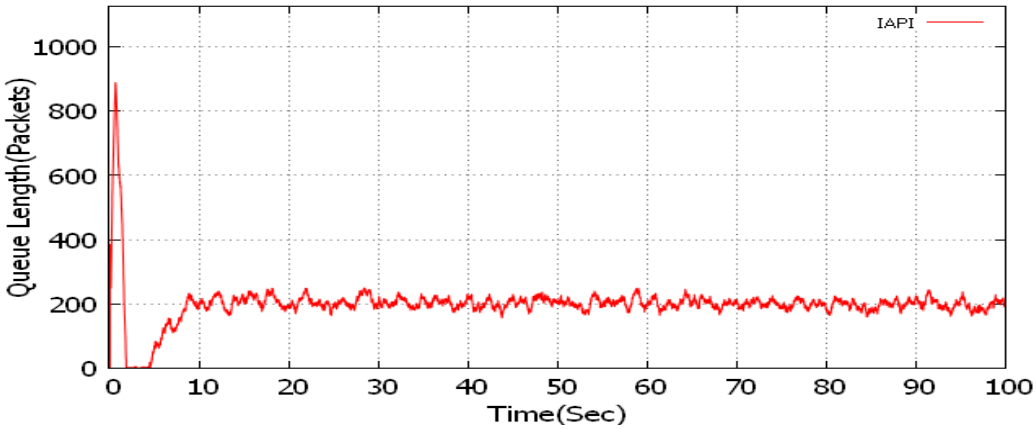

Figure 7.3 Queue length of IAPI controller

The queue length of REM controller is calculated and shown that it takes less time than PI and IAPI to come closer to the reference queue length. Even when it comes closer, it is still not very stable. This can be seen in figure 7.4.

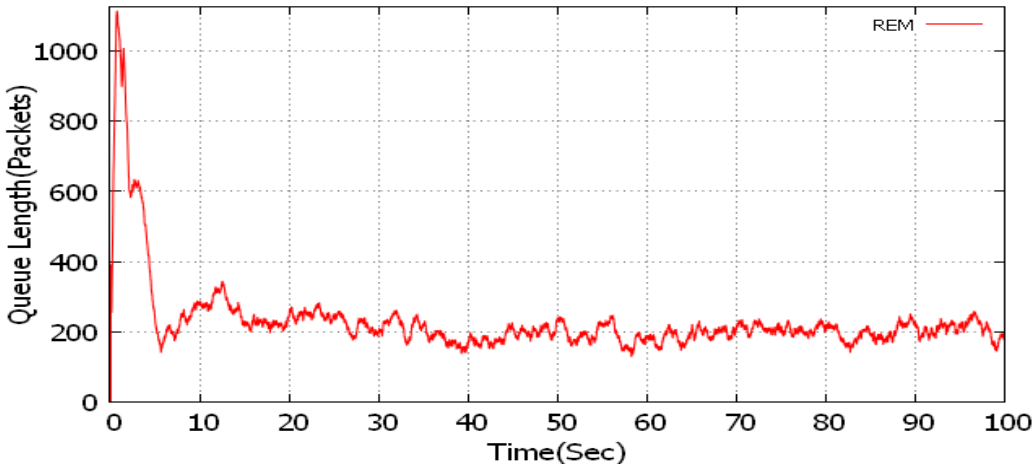

Figure 7.4 Queue length of REM controller

We calculate the queue length of our Proposed AQM and find out that it takes less time than PI, IAPI and REM to come closer to the reference queue length. It attains the queue reference in least time and maintains the stability around it. This is presented in figure 7.5.

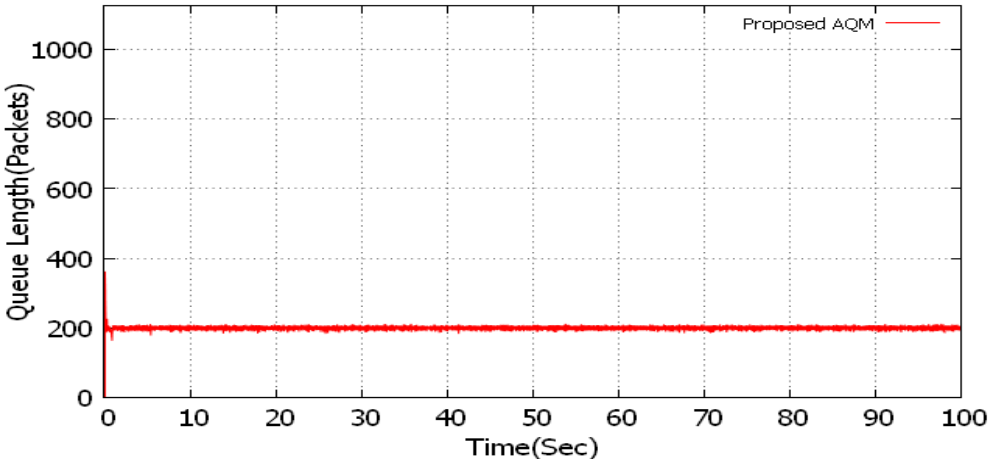

Figure 7.5: Queue length of Proposed AQM

How fast the queue length of the controllers PI, IAPI, REM and Proposed AQM reach the targeted queue length comparison has been made in figure 7.6. And as we can see in the figure that Proposed AQM is the most stable one when it comes to acquiring the targeted queue length.

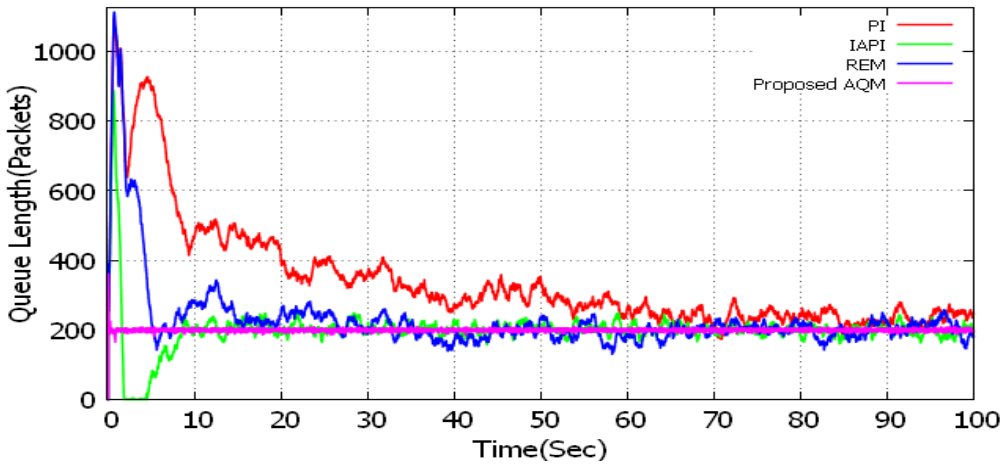

Figure 7.6: Queue length of PI, IAPI, REM and Proposed AQM

A comparison between PI, IAPI, REM and Proposed AQM in terms of the amount of average throughput (Mbps) is presented in figure 7.7. We can see that our Proposed AQM has the highest average throughput.

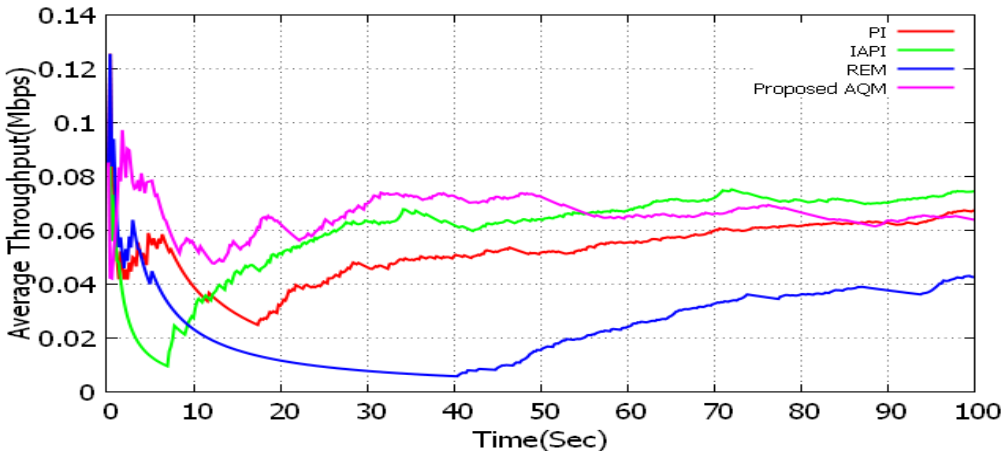

Figure 7.7: Average Throughput (Mbps) of PI, IAPI, REM and Proposed AQM

A comparison between PI, IAPI, REM and Proposed AQM in terms of average queue length (packets) is made and presented in figure 7.8. We can see that our Proposed AQM has the most stable average queue length.

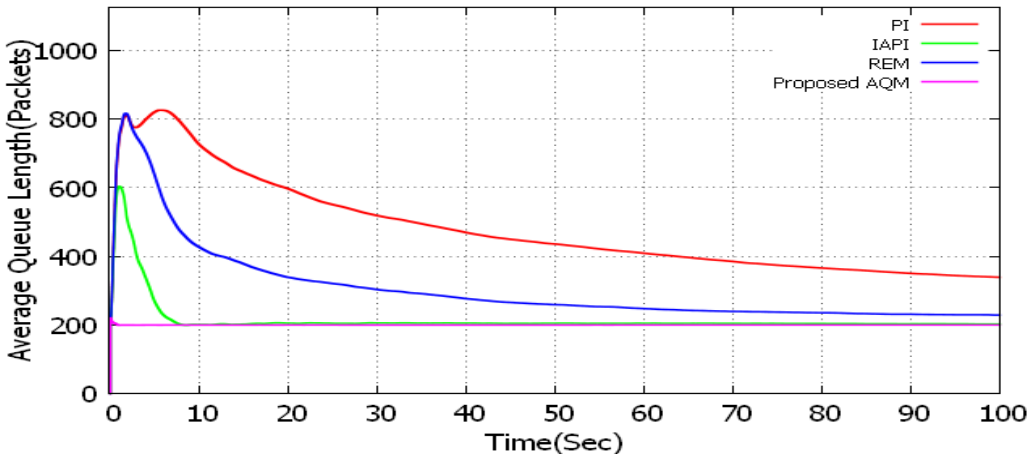

Figure 7.8: Average Queue length (packets) of PI, IAPI, REM and Proposed AQM

Case 2: Performance under constant TCP connections

We take the number of TCP connections, N as 100 and keep the rest of the experimental set up as it is. Then we make a critical comparison study between the existing IAPI controller and our Proposed AQM based on performance matrices such as queue length (packets), average throughput (Mbps) and average queue length (packets).

The queue length of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.9.

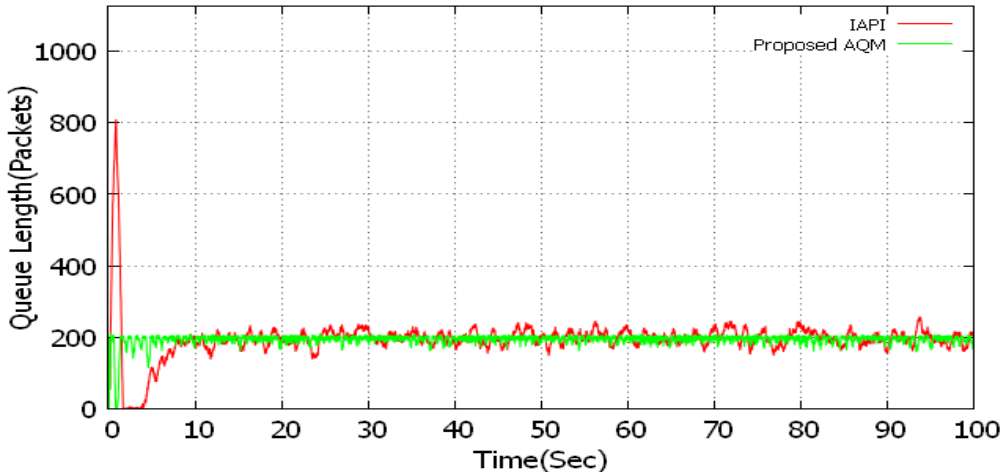

Figure 7.9: Queue length (packets) of IAPI and Proposed AQM for N=100

The average throughput (Mbps) of IAPI and Proposed AQM is calculated. We find that our Proposed AQM gives higher throughput as compared to IAPI. This is presented in figure 7.10.

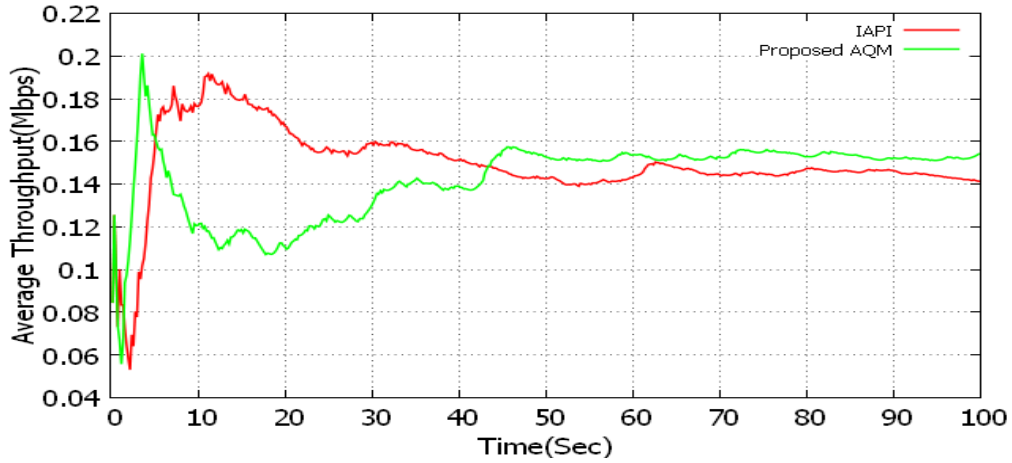

Figure 7.10: Average Throughput (Mbps) of IAPI and Proposed AQM for N=100

The average queue length of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.11.

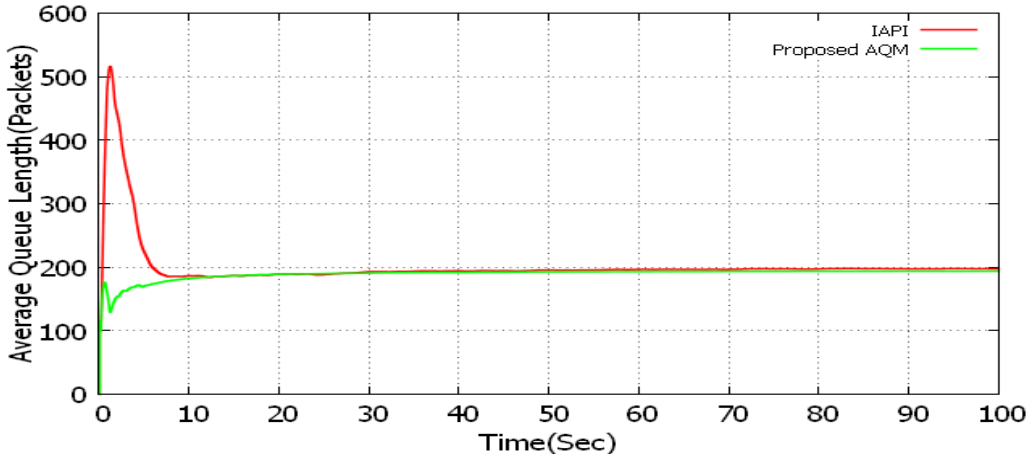

Figure 7.11 Average Queue length (packets) of IAPI and Proposed AQM for N=100

We take the number of TCP connections, N as 2000 and keep the rest of the experimental set up as it is. Then we make a critical comparison study between the existing IAPI controller and our Proposed AQM based on performance matrices such as queue length (packets), average throughput (Mbps) and average queue length (packets).

The queue length of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.12.

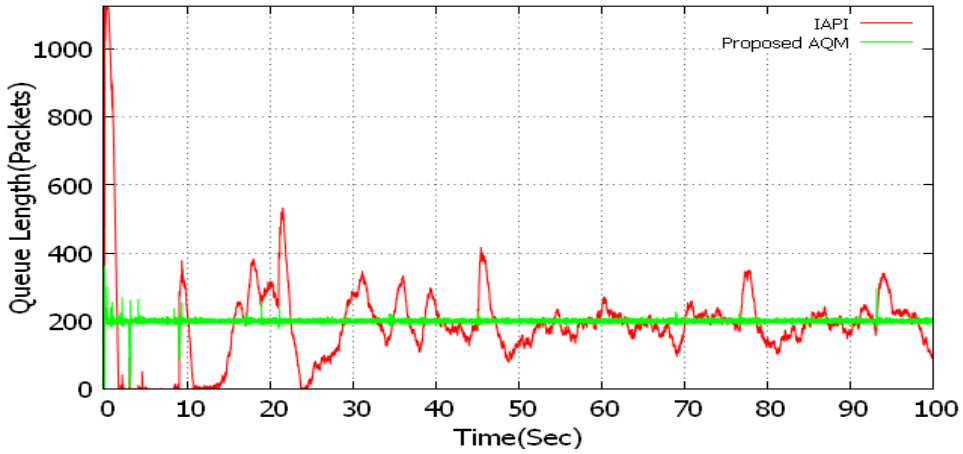

Figure 7.12 Queue Length (packets) of IAPI and Proposed AQM for N=2000

The average throughput (Mbps) of IAPI and Proposed AQM is calculated. We find that our Proposed AQM gives higher throughput as compared to IAPI. This is presented in figure 7.13.

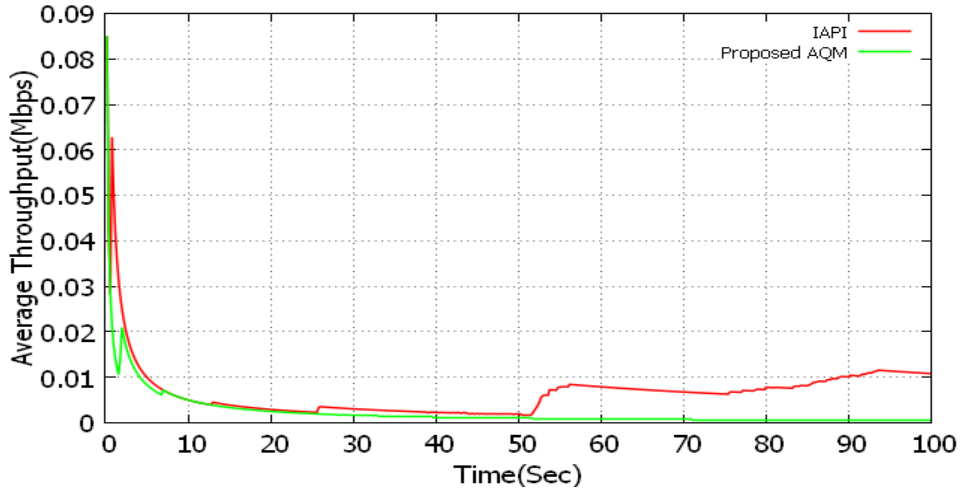

Figure 7.13: Average Throughput (Mbps) of IAPI and Proposed AQM for N=2000

The average queue length of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.14.

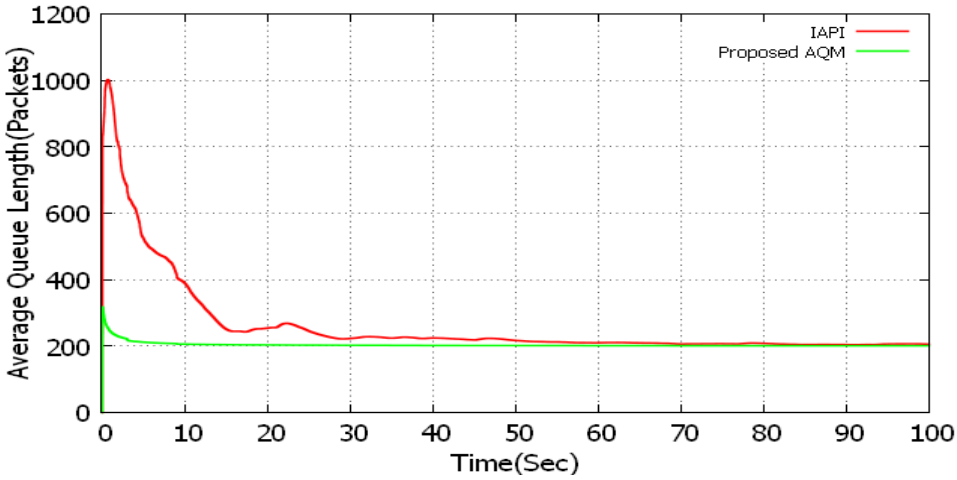

Figure 7.14: Average Queue length (packets) of IAPI and Proposed AQM

Case 2: Performance under different bottleneck bandwidth.

We take the bottleneck bandwidth, C as 5 Mb and keep the rest of the experimental set up as it is. Then we make a critical comparison study between the existing IAPI controller and our Proposed AQM based on performance matrices such as queue length (packets), average throughput (Mbps) and average queue length (packets).The queue length (packets) of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.15.

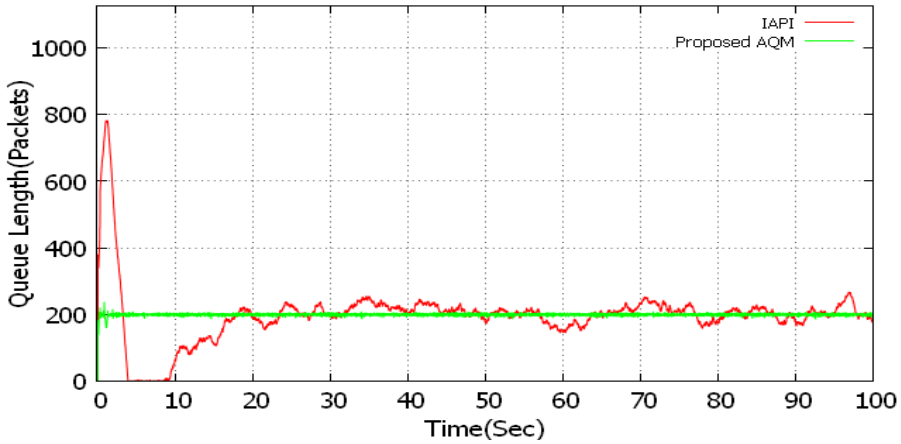

Figure 7.15: Queue length (packets) of IAPI and Proposed AQM for C=5Mb

The average throughput (Mbps) of IAPI and Proposed AQM is calculated. We find that our Proposed AQM gives higher throughput as compared to IAPI. This is presented in figure 7.16.

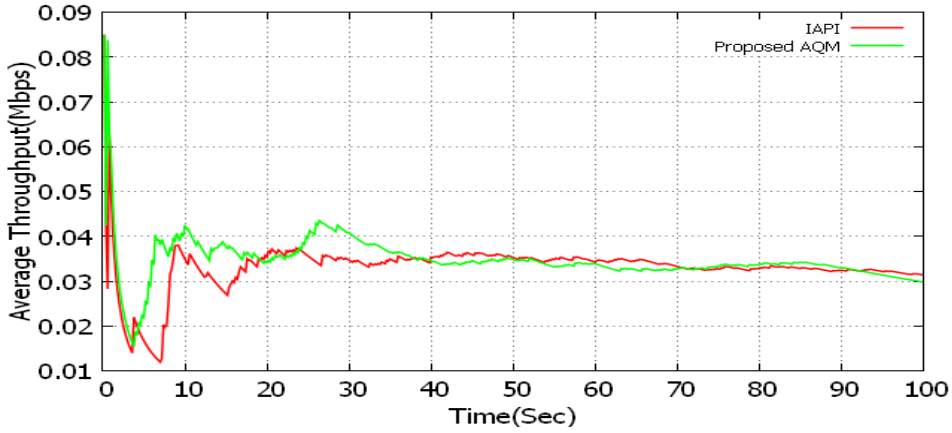

Figure 7.16: Average throughput (packets) of IAPI and Proposed AQM for C=5Mb

The average queue length of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.17.

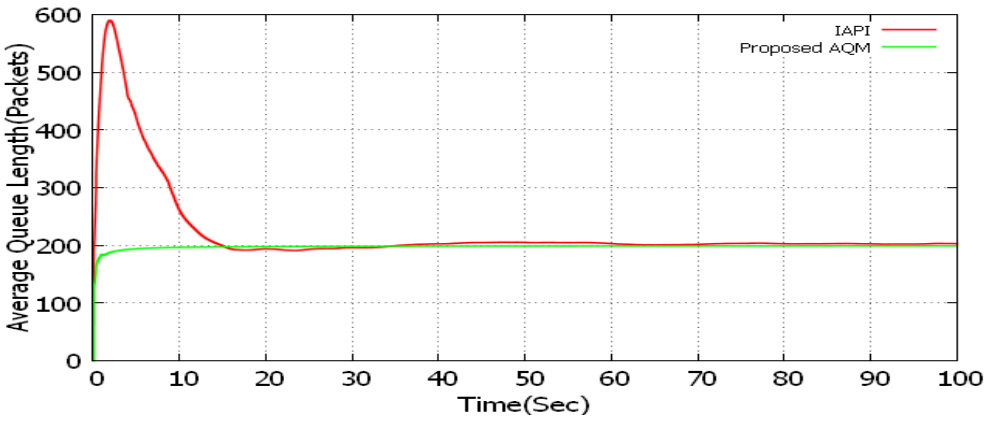

Figure 7.17: Average queue length (packets) of IAPI and Proposed AQM for C=5Mb

We take the bottleneck bandwidth, C as 40Mb and keep the rest of the experimental set up as it is. Then we make a critical comparison study between the existing IAPI controller and our Proposed

AQM based on performance matrices such as queue length (packets), average throughput (Mbps) and average queue length (packets).

The queue length (packets) of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.18.

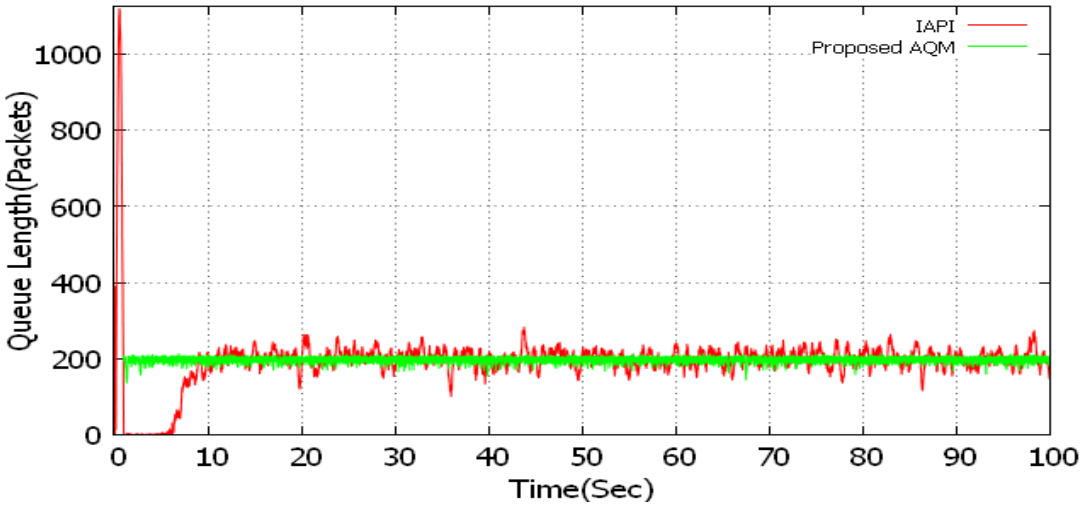

Figure 7.18 Queue length of IAPI and Proposed AQM for C=40Mb

The average throughput (Mbps) of IAPI and Proposed AQM is calculated. We find that our Proposed AQM gives more stable throughput as compared to IAPI. This is presented in figure 7.19.

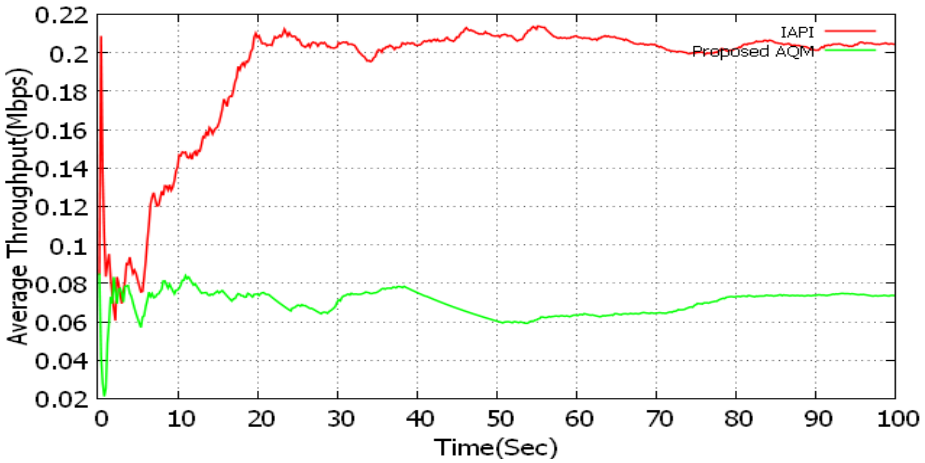

Figure 7.19: Average throughput of IAPI and Proposed AQM for C=40Mb

The average queue length of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.20.

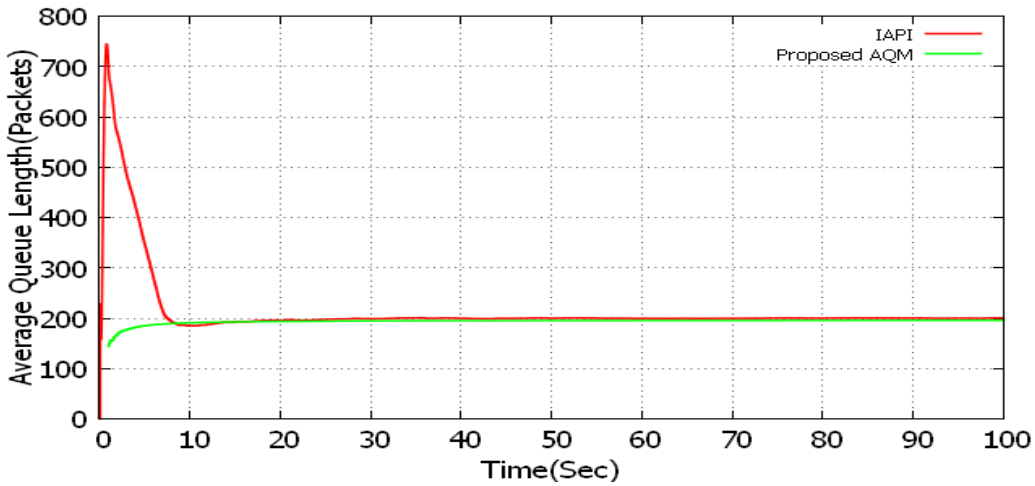

Figure 7.20: Average Queue length (packets) of IAPI and Proposed AQM for C=40Mb

Case 3: Performance under different Round Trip Time

We take the Round Trip Time, RTT as 60 ms and keep the rest of the experimental set up as it is. Then we make a critical comparison study between the existing IAPI controller and our Proposed AQM based on performance matrices such as queue length (packets), average throughput (Mbps) and average queue length (packets).

The queue length of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.21.

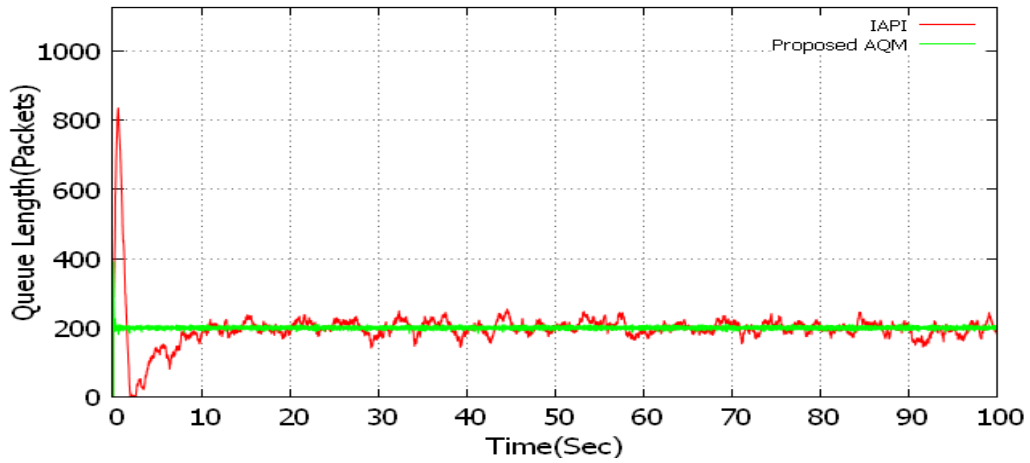

Figure 7.21: Queue length (packets) of IAPI and Proposed AQM for RTT=60ms

The average throughput (Mbps) of IAPI and Proposed AQM is calculated. We find that our Proposed AQM more stable throughput as compared to IAPI. This is presented in figure 7.22.

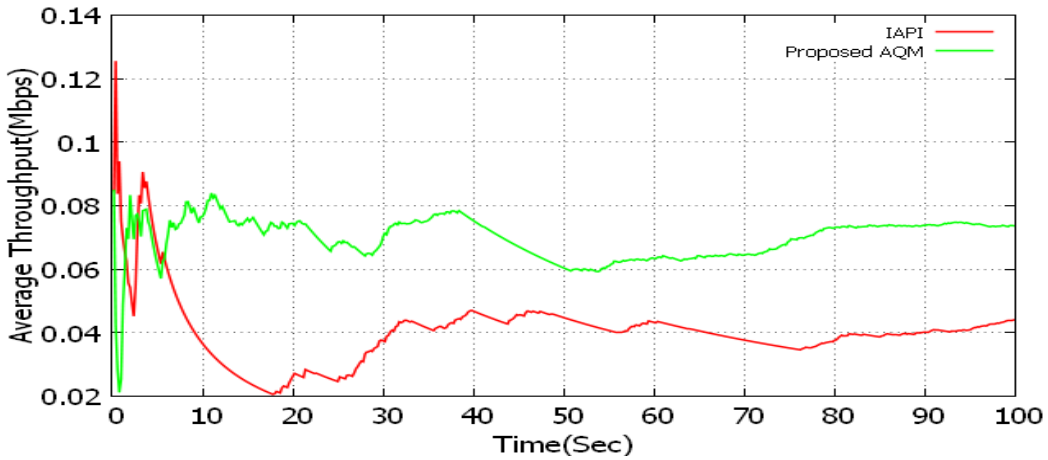

Figure 7.22: Average throughput (Mbps) of IAPI and Proposed AQM for RTT=60ms

The average queue length of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.23.

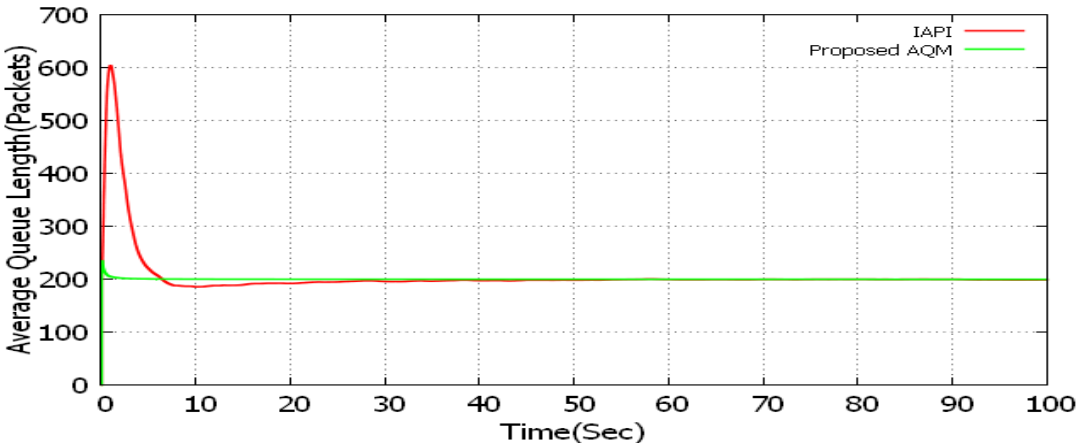

Figure 7.23: Average Queue length (packets) of IAPI and Proposed AQM for RTT=60ms

We take the Round Trip Time, RTT as 120 ms and keep the rest of the experimental set up as it is. Then we make a critical comparison study between the existing IAPI controller and our Proposed AQM based on performance matrices such as queue length (packets), average throughput (Mbps) and average queue length (packets).

The queue length of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.24.

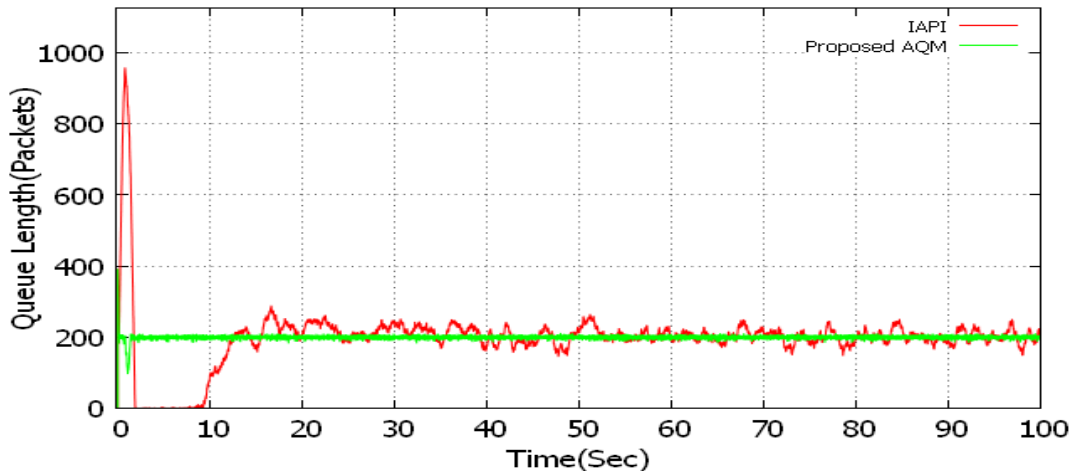

Figure 7.24: Queue length (packets) of IAPI and Proposed AQM for RTT=120ms

The average throughput (Mbps) of IAPI and Proposed AQM is calculated. We find that our Proposed AQM gives more stable throughput as compared to IAPI. This is presented in figure 7.25.

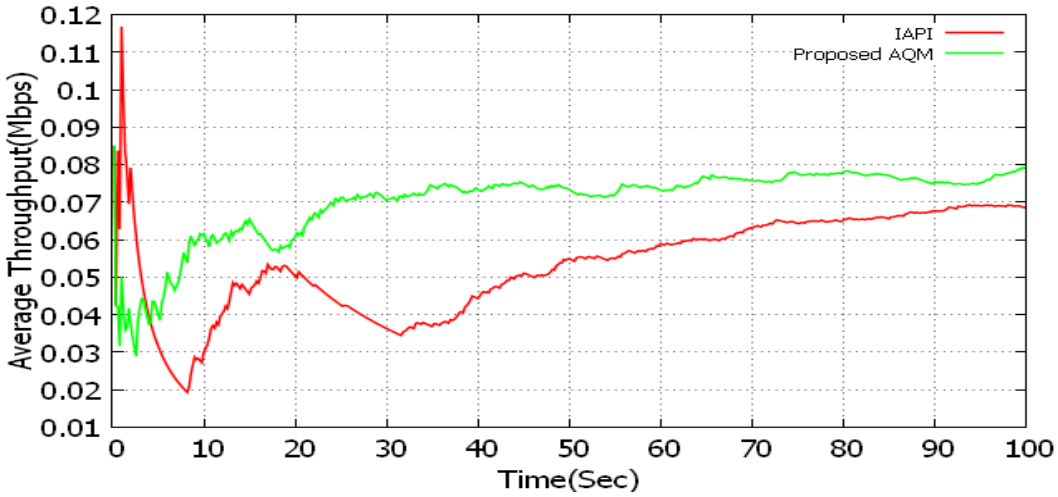

Figure 7.25: Average Throughput (Mbps) of IAPI and Proposed AQM for RTT=120ms

The average queue length of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.26.

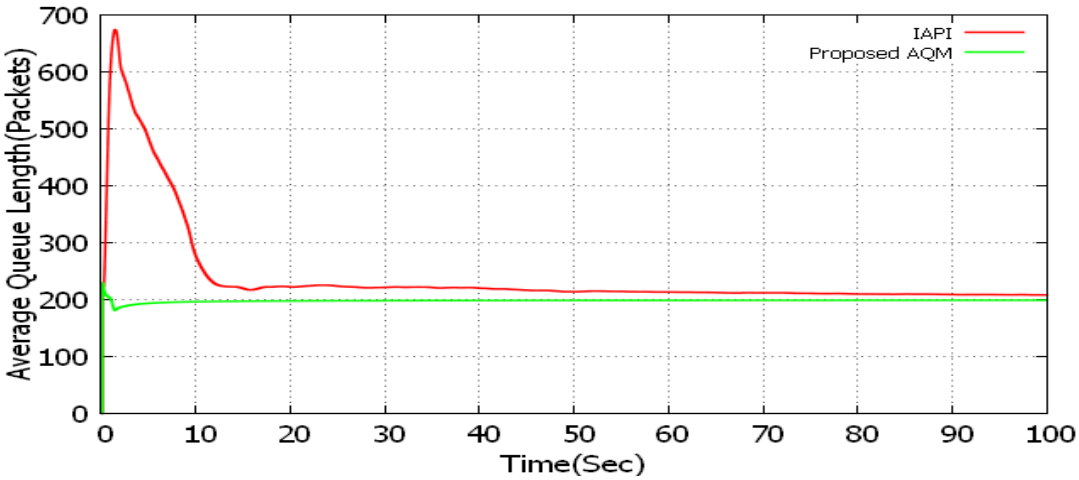

Figure 7.26: Average Queue Length (packets) of IAPI and Proposed AQM for RTT=120ms

Case 4: Performance under different Reference Queue length

We take the targeted or reference queue length (packets), q_{ref} as 100 and keep the rest of the experimental set up as it is. Then we make a critical comparison study between the existing IAPI

controller and our Proposed AQM based on performance matrices such as queue length (packets), average throughput (Mbps) and average queue length (packets).

The queue length of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.27.

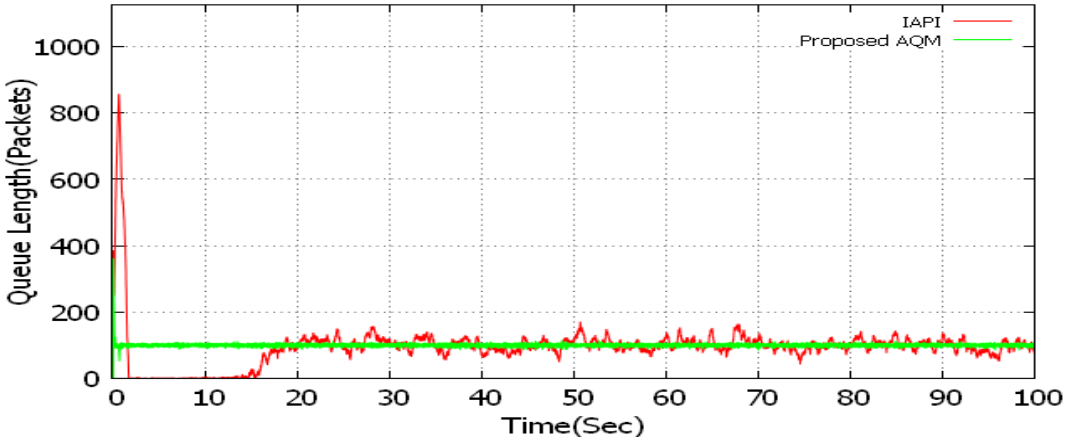

Figure 7.27: Queue Length (packets) of IAPI and Proposed AQM for q_{ref}=100 packets

The average throughput (Mbps) of IAPI and Proposed AQM is calculated. We find that our Proposed AQM gives higher throughput as compared to IAPI. This is presented in figure 7.28.

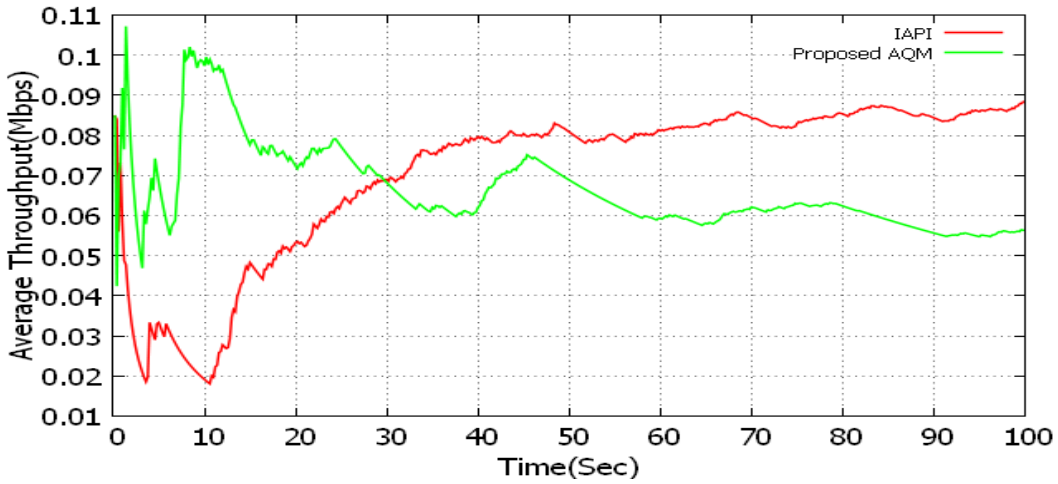

Figure 7.28: Average Throughput (Mbps) of IAPI and Proposed AQM for q_{ref}=100 packets

The average queue length of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted

queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.29.

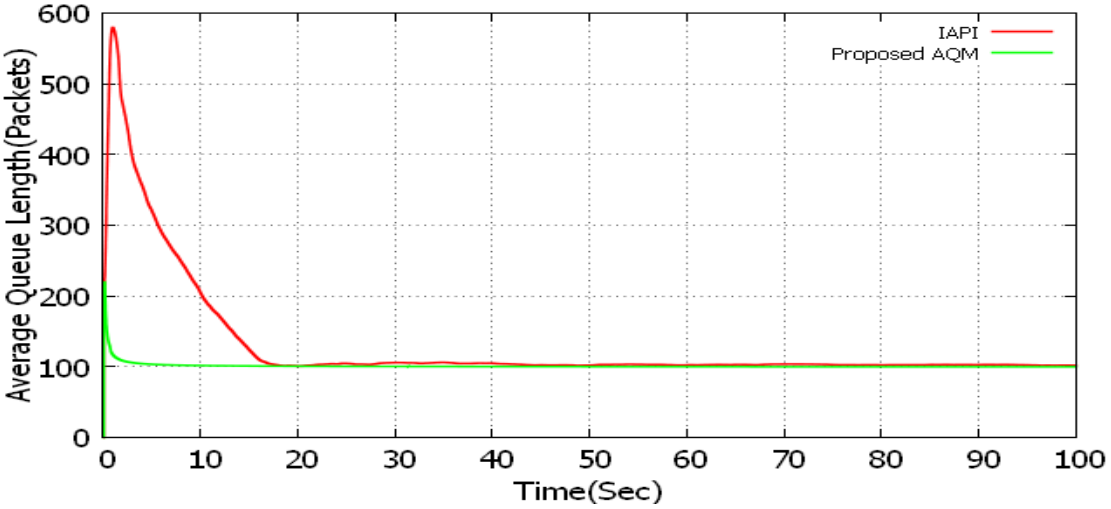

Figure 7.29: Average Queue Length (packets) of IAPI and Proposed AQM for q_{ref}=100 packets

We take the bottleneck bandwidth, C as 5Mb and keep the rest of the experimental set up as it is. Then we make a critical comparison study between the existing IAPI controller and our Proposed AQM based on performance matrices such as queue length (packets), average throughput (Mbps) and average queue length (packets).

The queue length (packets) of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.30.

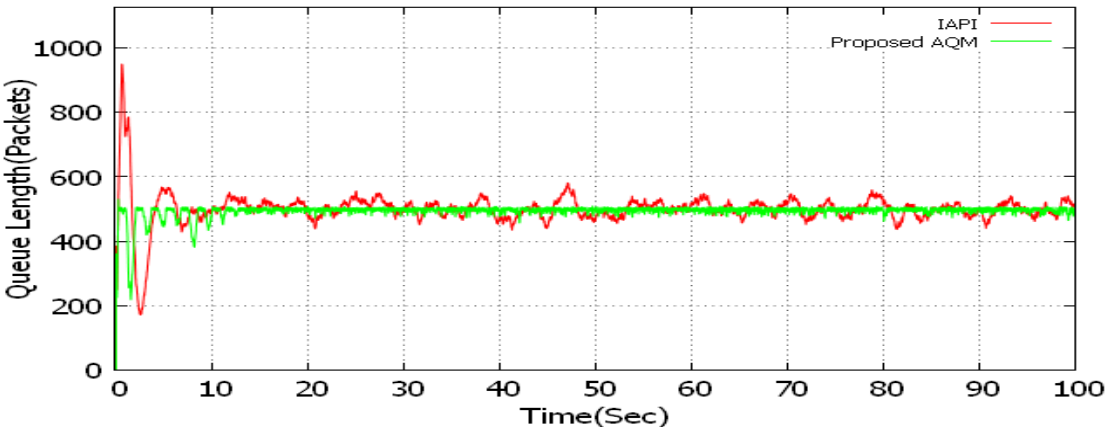

Figure 7.30: Queue length (packets) of IAPI and Proposed AQM for q_{ref}=500 packets

The average throughput (Mbps) of IAPI and Proposed AQM is calculated. We find that our Proposed AQM gives higher throughput as compared to IAPI. This is presented in figure 7.31.

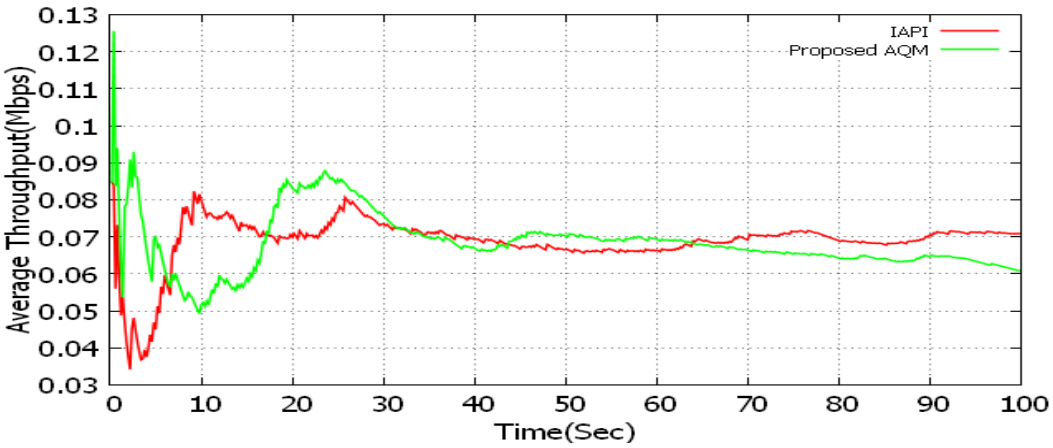

Figure 7.31: Average Throughput (Mbps) of IAPI and Proposed AQM for q_{ref}=500 packets

The average queue length of IAPI and Proposed AQM is calculated. We find that our Proposed AQM attains the targeted queue length faster than IAPI. Also, it fluctuates around the targeted queue length far less than IAPI. This shows our Proposed AQM is more stable. This is presented in figure 7.32.

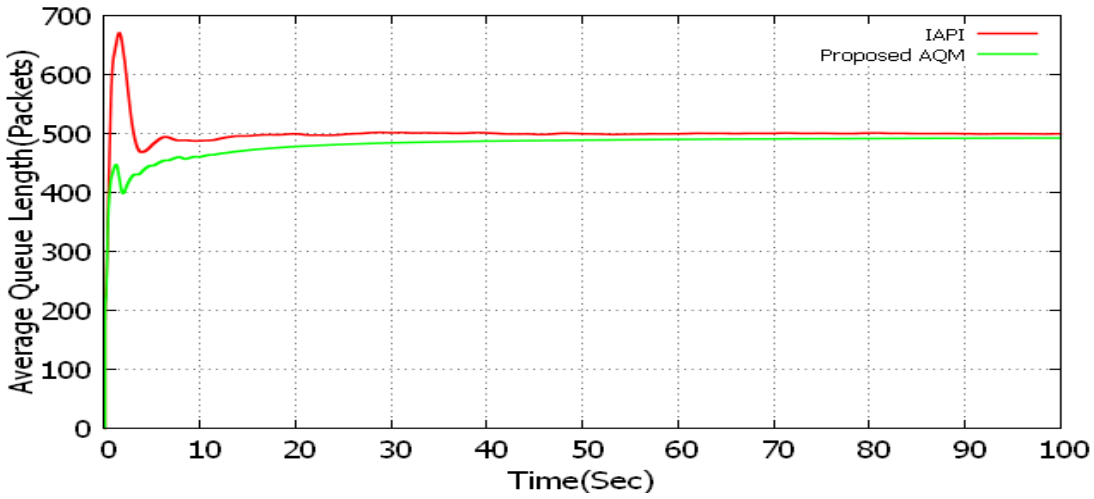

Figure 7.32: Average Queue Length of IAPI and Proposed AQM for q_{ref}=500 packets

PART 2: Performance of Neuron based AQM

In Part 2, a critical comparative study has been made between MPID, AN-MPID, AN-MPID (Act-1), AN-MPID (Act-2), AN-MPID (Hybrid) and AQM technique PID that has been presented in Chapter 4. This comparison focuses on performance evaluation of the proposed AQM based on performance metrics such as queue length, average queue length, packet send, and packet receive, packet drop, delay and average throughput of the mentioned AQMs. The network topology is stated in figure 7.1.

We calculate the queue length (packets) of PID controller and find that it is highly unstable as it takes a lot of time to reach the targeted queue length. Also, it fluctuates a lot around the targeted queue length. This is presented in figure 7.33.

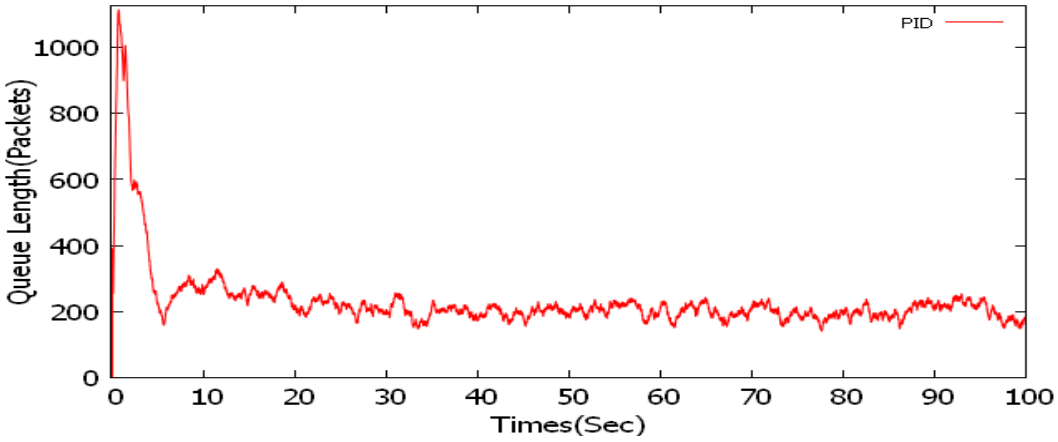

Figure 7.33: Queue length (packets) of PID controller

Then we calculated the queue length (packets) of MPID controller. It reaches the reference queue length mush faster than PID. But it still varies around the reference queue length. This is shown in figure 7.34.

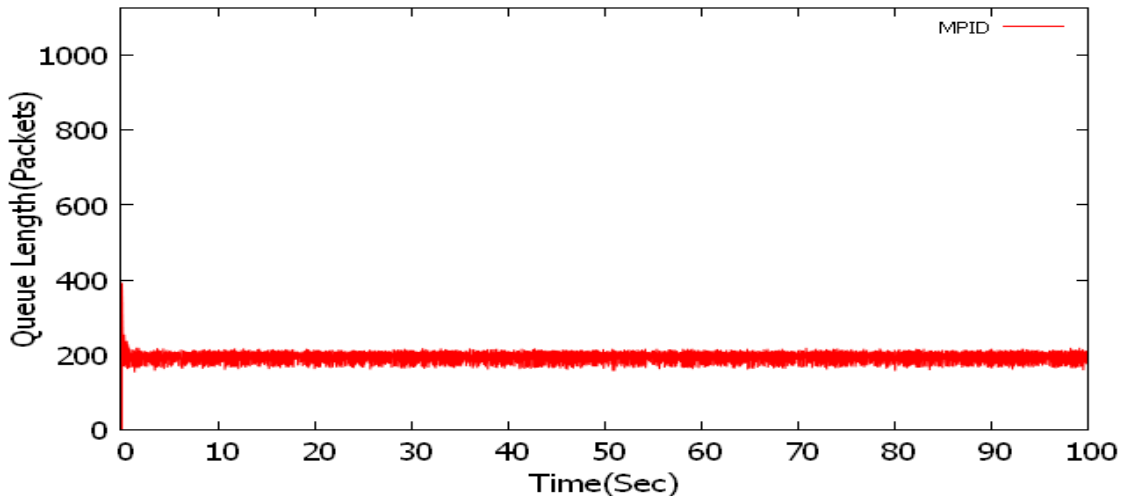

Figure 7.34: Queue length (packets) of MPID controller

We use an adaptive neuron in MPID to develop AN-MPID. We analyze the performance of AN-MPID in terms of its queue length. The analysis is shown in figure 7.35.

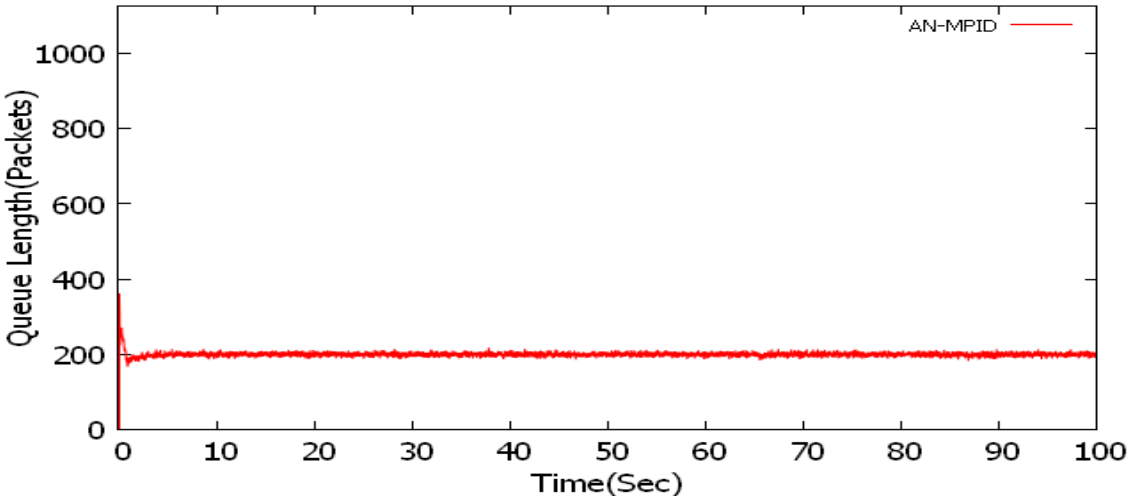

Figure 7.35: Queue length (packets) of AN-MPID

We further use activation function to develop AN-MPID (Act-1). We analyze the performance of AN-MPID (Act-1) in terms of queue length and can see that it is more stable than the above mentioned AQM techniques as it reaches the targeted queue length faster. This can be seen in figure 7.36.

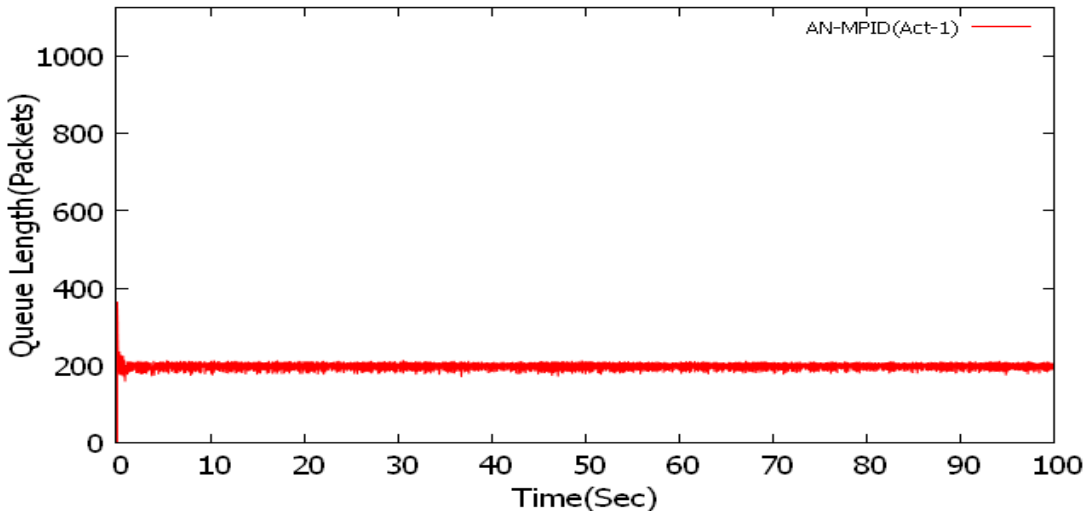

Figure 7.36: Queue length (packets) of AN-MPID (Act-1)

We use different activation function to develop AN-MPID (Act-2). We analyze the performance of AN-MPID (Act-2) in terms of queue length. We can see that even though it gives better results than PID, MPID, and AN-MPID. It shows very less variation in performance in comparison to AN-MPID (Act-1). We can see this in figure 7.37.

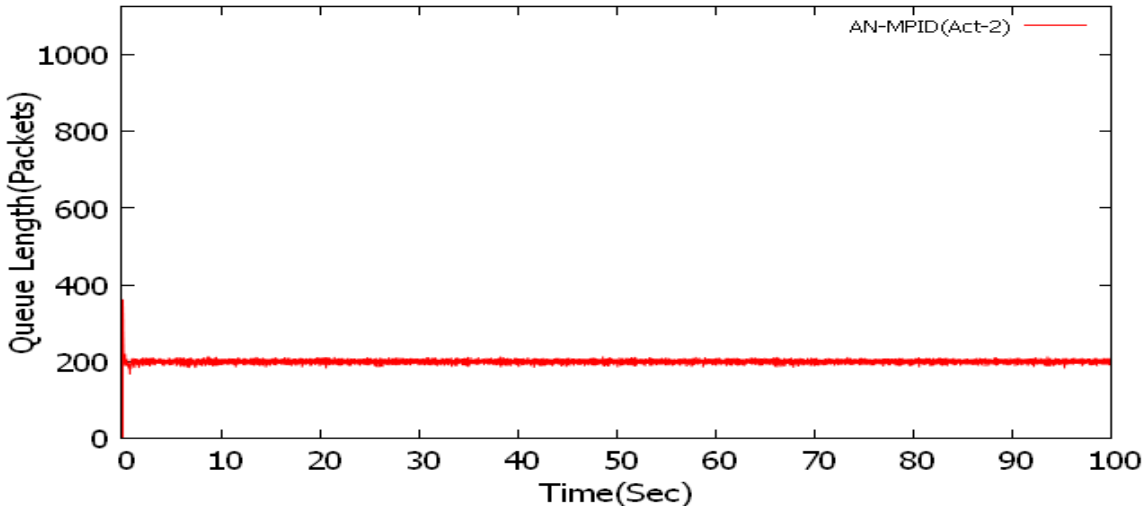

Figure 7.37: Queue length (packets) of AN-MPID (Act-2)

We have further developed another AQM technique called AN-MPID (Hybrid) which is a hybrid of AN-MPID (Act-1) and AN-MPID (Act-2). We analyze the performance of AN-MPID (Hybrid) in terms of queue length and find that it is the most stable one in the lot as it reaches the reference queue length fastest and varies the least around the reference queue length. This can be seen in figure 7.38.

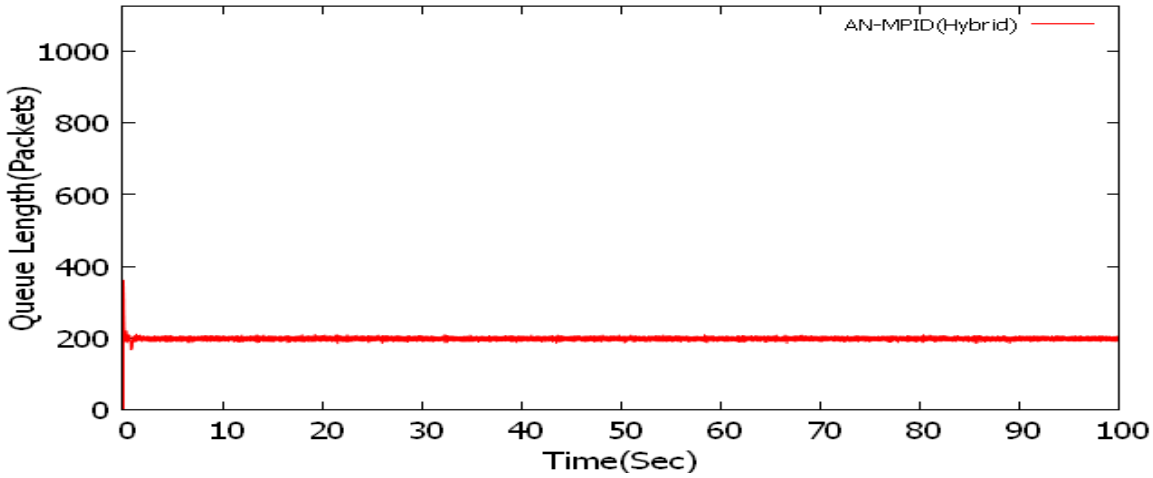

Figure 7.38: Queue length (packets) of AN-MPID (Hybrid)

We have graphically represented the Average Throughput (Mbps) values for PID, MPID, AN-MPID, AN-MPID (Act-1), AN-MPID (Act-2) and AN-MPID (Hybrid) to make a clear comparison between them with respect to the throughput offered by them. This can be seen in figure 7.39.

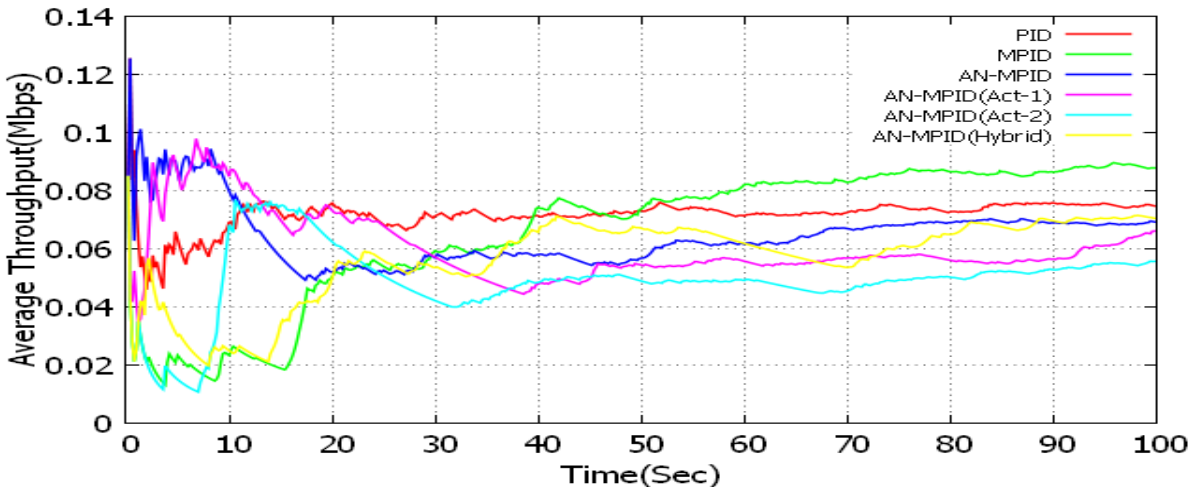

Figure 7.39: Average Throughput (Mbps) of AQMs

We have graphically represented the Average Queue length (packets) values for PID, MPID, AN-MPID, AN-MPID (Act-1), AN-MPID (Act-2) and AN-MPID (Hybrid) to make a clear comparison between them with respect to the stability offered by them. We can see that AN-MPID (Hybrid) attains the reference queue length faster than other techniques and remains stable around the reference queue length. This can be seen in figure 7.40.

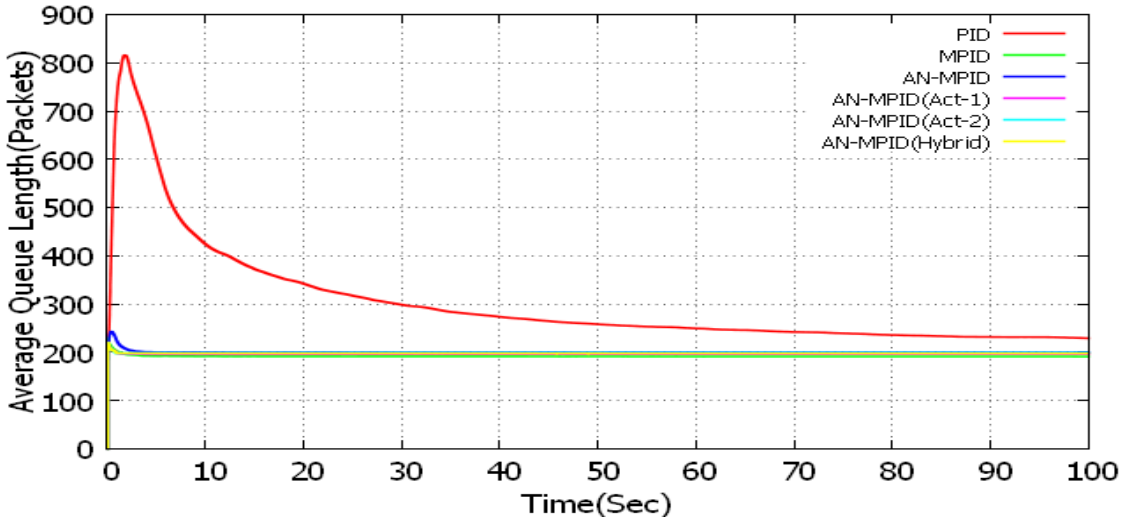

Figure 7.40: Average Queue Length (packets) of Different AQMs

We have further made a comparison between the queue lengths of our developed AQMs MPID, AN-MPID, AN-MPID (Act-1), AN-MPID (Act-2) and AN-MPID (Hybrid) in figure 7.41.

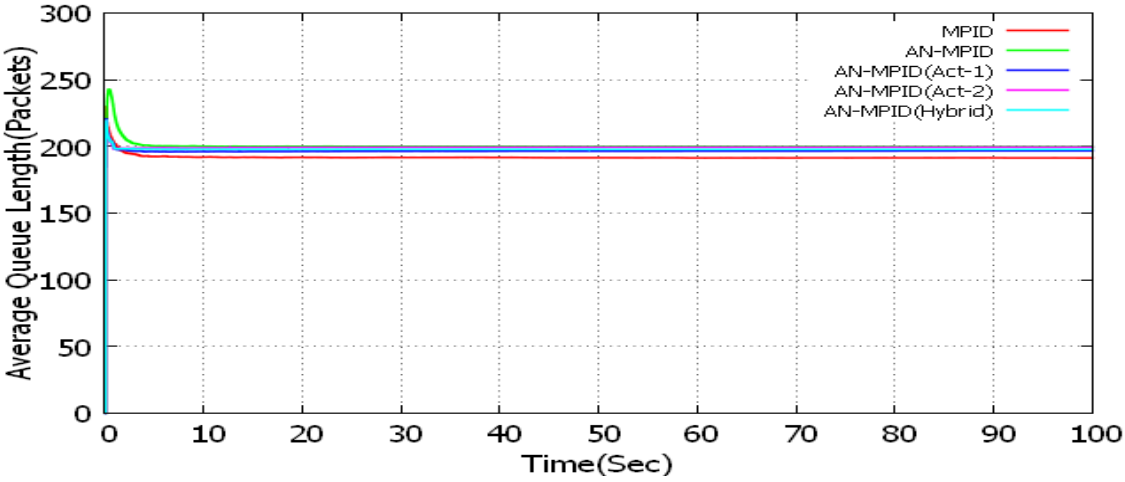

Figure 7.41: Average Queue Length (packets) of Proposed AQMs

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this project, we developed three new AQM techniques named as Proposed AQM, MPID and AN-MPID. In first part of our project, we developed a new AQM called Proposed AQM and compared its performance with Proportional Integral (PI), Random Early Marking (REM) and Intelligent Adaptive Proportional Integral (IAPI). Then the stability of the proposed AQM was analyzed with IAPI by varying different network parameters such as number of TCP connections (N), bottleneck bandwidth (C), and Round trip time (RTT) and target queue length (q_{ref}). We concluded that our Proposed AQM show better performance then others and achieve stability under dynamic network environment.

In second part of our project, we modified the existing Proportional Integral Derivative (PID) controller and named as MPID. And then we have used concepts of neural networks with MPID to develop AN-MPID. Then, we used activation function to develop AN-MPID (Act-1), AN-MPID (Act-2) and AN-MPID (Hybrid). Then we compared the performance of the developed AQMs with PID. We concluded that MPID, AN-MPID, AN-MPID (Act-1), AN-MPID (Act-2) and AN-MPID (Hybrid) performed better than the existing PID controller. However neuron based AQM such as AN-MPID, AN-MPID (Act-1), AN-MPID (Act-2) and AN-MPID (Hybrid) performs better than MPID. The simulation experiment result shows that our neuron based controllers performs better than PID algorithm with respect to stable queue length, and high throughput

8.2 Future Work

Our work was limited to investigating and designing various AQM techniques to avoid congestion in network router. And also analyzing their performances under varying network conditions. Further research work could be carried out in analysis of the stability of the proposed AQMs by using control theory. Further, our proposed AQMs can be physically configured and implemented on physical routers by creating image file of the proposed AQMs. Further, fuzzy based neural network can be implemented in the proposed algorithms. Concluded work can be used for further research.

Author's Publications

S.K. Bisoy, P.K. Pandey, A. Vidyanta, R.K. Saw, " Design of an Active Queue Management based on Neural Networks for Congestion Control", IEEE Communications Letters (submitted).

REFERENCES

[1] Omer Dedeoglu, "Impacts of RED in a Heterogeneous Environment", B.S., Electrical Engineering, Bilkent University, May, 2003.

[2] X. Chang and J.K. Muppala and J. Yu, "A robust nonlinear PI controller for improving AQM performance", Proc. IEEE ICC (2004) 203-205

[3] Shahid Akhtar, Andrea Francini, Dave Robinson, Randy Sharpe, "Interaction of AQM schemes and adaptive streaming with Internet traffic on acess networks.", Proc. IEEE 2014, Global Communication Conference

[4] J.Sun, K.T.Ko, G. Chem, S.Chab and M. Zukerman, "PD-Controller: A new active queue management scheme". Proc. IEEE Globecom 2003, vol. 6, San Francisco.pp. 3103-3107, 2003.

[5]W.Feng, D.Kandlur, D.Sahaand K.Shin, "BLUE: A New class of Active Queue Management Algorithms.", Tech.Rep. UM CSE-TR-387-99,1999.

[6]Wu-Chun Feng, A. Kapadia, S. Thulasidasan, "GREEN: a proactive queue management over a best-effort network", Global Telecommunication Conference, 2002.

[7] S.S. Kunniyur, R. Srikant, "An adaptive virtual queue (AVQ) algorithm for active queue management", IEEE/ACM Transactions on Networking, 2004.

[8] E. Hashem, Random Drop Congestion Control, M.S. Thesis, Massachusetts Institute of Technology, Department of Computer Science, PhD thesis, 1990.

[9] Hyuk Lim, Kyung-Joon Park, Eun-Chan Park, Chong-Ho Choi, "Virtual Rate Control Algorithm for active queue manangement in TCP flows.", Proc. IEEE, 2002

[10] Y. Du and N. Wang , " A PID controller with neuron tuning parameters for multi-model plants", In: proceedings of 2004 International Conference on Machine Learning and Cybernetics, vol. 6, pp. 3408-3411,2004.

[11] Jinsheng Sung, Sammy Chan, King-Tim-K, Mosbe Zukerman, "Neuron PID: A Robust AQM Scheme, Proc. IEEE

[12] M. Yaghoubi Waskasi, M.J. Yazdanpanah, N. Yazdani, "A New Active Queue Manangement Algorithm Based on Neural Networks PI", Proc IFAC, 2005

[13]Jinsheng Sun, Sammy Chan, Moshe Zukerman, "IAPI: An Intelligent Adaptive PI Active Queue Management Scheme.", Proc. IEEE, 2012.

[14] B. Braden and e. al, "Recommendation on Queue Management and Congestion Avoidance in the Internet," IETF RFC 2309, April 1998.

[15] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers", in ACM SIGCOMM 2004, August 2004. Extended version: Stanford HPNG Technical Report TR04-

HPNG-060800.
[16] G. Bajk, I. Moldovn, O. Pop, and J. Br, "TCP flow control algorithms for routers", in SIGCOMM'04, Aug 2004.
[17] Omer Dedeoglu, "Impacts of RED in a Heterogeneous Environment", B.S., Electrical Engineering, Bilkent University, May, 2003.

[18] S. Floyd and V. Jacobson. "Random Early Detection Gateways for Congestion Avoidance". IEEE/ACM Transactions on Networking, vol 1(4), pp 397–413, 1993.

[19] Dong Lin and Robert Morris, "Dynamics of Random Early Detection." In Proceedings of ACM SIGCOMM, August 1997.

[20] Khalid S.R., Alawfi "Active Queue Management based on the use of Self Similarity Long Range Dependency found in Internet Traffic", PhD thesis, University of Bradford 2006.

[21] W. Feng, D. Kandlur, D. Saha, and K. Shin. "Techniques for Eliminating Packet Loss in Congested TCP/IP Network". U. Michigan CSE-TR-349-97, November 1997.

[22]Y.Du and N. Wang, "A PID controller with neuron tuning parameters for multi-model plants", Proceedings of 2004 International Conference on Machine Learning and Cybernetics, vol. 6, pp. 3408-3411, 2004.

[23] Guodong Wang, Yongmao Ren, Jun Li, "An effective approach to alleviating the challenges of Transmission Control Protocol", Proc. IEEE, 2014.