Dual Degree Dissertation

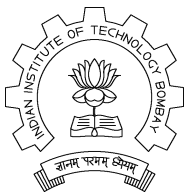
Congestion Management & Bandwidth Allocation for Best-effort Traffic in Packet Switched Networks

submitted in partial fulfillment of the
requirements for the degree of

Bachelor of Technology
in *Electrical Engineering*
&
Master of Technology
in *Communications & Signal Processing*
(under the Dual Degree Programme)
by

Abhishek Jain
Roll no. 98D07040

under the guidance of
Prof. Abhay Karandikar



Department of Electrical Engineering
Indian Institute of Technology, Bombay
Powai, Mumbai-400076.
June, 2003.

Dissertation Approval Sheet

The Dual Degree Dissertation titled "**Congestion Management and Bandwidth Allocation for Best-effort Traffic in Packet Switched Networks**" submitted by **Abhishek Jain** (Roll Number: 98D07040) may be accepted.

Guide: **Chairman:**

Internal Examiner: **External Examiner:**

Date: 24/06/2003

Congestion Management & Bandwidth Allocation for Best-effort Traffic in Packet Switched Networks

Abhishek Jain

Dual Degree Thesis

Acknowledgements

No thesis is the complete work of any one particular author. In my journey towards a successful completion of my Dual Degree thesis, I was assisted and shown the path by numerous people. I would like to take this opportunity to thank and express my gratitude towards some of them.

First and foremost, I would like to praise Prof. Abhay Karandikar, with whom I have been working for the past 38 months on various projects big and small, for all the valuable guidance, philosophy and most importantly the support and encouragement given during times of difficulty. I have learnt a lot and still have plenty to learn from him. Once a guide, always a guide.

Work has not always been the incentive to come to lab. Here, I would like to show appreciation towards Shruti Mahajan and Harish Ramamurthy for making the final year sojourn in the lab, a very eventful and gratifying experience.

I am also indebted to Prof. Rakesh Lal, Prof. U B Desai, Prof. Deepankar Sarkar and Electronic Design Project partners, Premal Shah, Hariharan Narayanan and Shruti Mahajan, who conjointly, through the year long work (prior to my thesis) helped inculcate the *never compromise* and *always strive for perfection* attitude in me. This was the first time in my life, when I put in days and days of effort, just to go the extra mile. This has helped me a lot in all my subsequent endeavors.

I am very much obliged to Rahul Verma for getting me started in APACE (Chapter 2) and Ajay Kumar Singh for Learning TCP (Chapter 4), which was his brain child.

On the whole, it was an experience worth carrying forward in all walks of life. As they say,

"The best things in life are always free."

Abhishek Jain

Abstract

In this thesis, we aim at reducing or eliminating wherever feasible, the side effects and uncalled-for under link utilization, packet delay and packet loss owing to congestion. We propose three new algorithms for congestion management and bandwidth allocation, all addressing the same basic issue of improvement in the performance of *best-effort* traffic in the Internet.

We propose an Active Queue Management (AQM) strategy, called *Adaptive Prediction based Approach for Congestion Estimation in Active Queue Management (APACE)* that takes a packet drop decision at the router by predicting the instantaneous queue length at a future instant of time using adaptive filtering techniques. APACE is competent in controlling the oscillations in the instantaneous queue. It also achieves a higher link utilization, lower packet delay and lower packet loss rate than the existing schemes in single as well as multiple bottleneck links. In addition APACE is not very sensitive to parameter settings, adapts quickly to changes in traffic and can achieve a given delay and/or link utilization or packet loss by varying just one of its parameters, thus giving the network operator a better tool to manage congestion, guarantee packet delays and improve the performance of the network.

We, then propose a modification to the TCP algorithm called *Learning TCP (LTCP)* that is not specific to any particular type of network network *ie,* wireless or wired or a mix of both) and hence can be employed in a wide variety of networks to improve the end-to-end performance of TCP. LTCP attempts to learn the cause of packet loss adaptively and subsequently takes appropriate corrective measures.

Our bandwidth allocation tool, *Scepter*, aims at improving the performance of best-effort TCP traffic by enabling traffic aggregates operate very close to their allocated bandwidth. Scepter will enable Internet Service Providers (ISPs) to allocate its customers, the bandwidth they demand (could be 2Mbps, 1.85Mbps or anything) with very high precision at a very small time scale. Scepter achieves its objectives by

modification in the token bucket of the traffic marker at the edge router.

Contents

List of Figures

CHAPTER

1

Introduction & Motivation for the Thesis

There are no problems…,
But only issues that need to be resolved

In this chapter, we explain the following:

- *Issues in the present-day Internet*
- *Adaptive Prediction based Approach for Congestion Estimation in Active Queue Management (APACE)*
- *Learning TCP (LTCP)*
- *Scepter: A Bandwidth Management Tool*

Chapter 1

Introduction & Motivation for the Thesis

1.1 Present-day Internet

Internet is fast becoming an inseparable part of our lives with its ever increasing applications such as emails, multimedia streaming, Internet telephony, video conferencing, etc adding value and glamour to our daily lives. Imagine millions of computers[1] all across the globe connected to each other, communicating, exchanging messages in less than a fraction of a second. Imagine yourself video conferencing with someone 20,000 km away from you. Let us try to understand the global Internet topology. How exactly does one connect to some other computer at a distant location? Computers at one's home, offices, labs, etc are connected to a switch owned by an Internet Service Provider (ISP) either directly or through a central switch in case of Local Area Networks (LAN's). The ISP's then connect them to the backbone routers connecting various regions of the world. A similar network topology exists on the receiver side of the network also, thus connecting one computer or another. Packetized data traverses all these nodes typically in less than a fraction of a second, thus enabling us to communicate efficaciously. As every good has a dark side to it, packets do not always traverse across these nodes without encountering the devil named *"Congestion"* which leads to packet drops, loss of data, increased packet delay, under link utilization and many such problems. So,

[1] By a computer, we mean any electronic device capable of connecting to the Internet.

"Why do we have congestion?"

The answer to the above, though contradictory is:

"Our protocols are designed, so as to create congestion"

Not convincing, isn't it? Consider the present Transmission Control Protocol (TCP) [1], the most widely used transport layer protocol. TCP increases its transmission rate till it causes congestion leading to packet drops. The protocol has been designed, so as to maximize the link utilization by trying to utilize as much of the resources as possible. As they say every dark cloud has a silver lining, the protocol after inducing congestion that has lead to packet drops of many others and itself, drastically reduces its transmission rate in order to be friendly and fair to other fellow sources. This act of kindness further aggravates the problem leading to severe underutilization of the link and global synchronization. However, we cannot also blame everything on TCP. The dynamics of the Internet are quite complicated. Allocating bandwidth to clients, ensuring maximum utilization of link, minimizing packet delays and losses, etc are all very difficult issues to manage.

The end hosts, *ie,* the Desktop computers communicate with each other using the transport layer protocols. These protocols provide for a logical communication between the end hosts. It creates a virtual environment in which both the end hosts communicate as if they are connected physically. The packets from the desktop computers then encounter the various switches and routers[2]. Of the many functions, a typical router performs, we shall restrict ourselves only to queue or buffer management aspect of it initially. In the later part, we will focus our attention on a specific bandwidth allocation mechanism at the ingress[3] router. Through these routers and switches, the packet eventually reaches its destination. In this thesis, we attempt to manage the congestion at various levels in the network and improve the performance of best-effort responsive traffic, for instance TCP traffic.

[2]We use the terms routers and switches interchangeably, since the difference amongst the two is irrelevant for our discussion. The algorithms that we will be discussing are applicable to both likewise.

[3]An ingress router is the first router through which a data packet enters a network from another network.

1.2 Contributions of the thesis

In this thesis, we aim at reducing or eliminating wherever feasible, the side effects and uncalled-for under link utilization, packet delay and packet loss owing to congestion. We propose three new algorithms addressing the same basic issue of improvement in the performance of *best-effort* traffic in the Internet. All of the three schemes attempt to reduce the unwanted effects owing to congestion, though in different context. We, now discuss the motivation for and the algorithms themselves in brief.

1.2.1 Adaptive Prediction based Approach for Congestion Estimation in Active Queue Management (APACE)

When the incoming packet rate is more than the outgoing rate, packets are temporarily kept in the router's memory, also known as the *buffer*. We say that such a packet has been buffered. In a DropTail mechanism, the incoming packet is kept in the buffer as long as there is space in the buffer. If the buffer is full, all incoming packets are dropped. The traffic source, which typically uses the TCP protocol, on encountering a packet loss reduces its window size[4] to half or one. Since all incoming packets are dropped, it is likely that many of the transmitting sources would experience a packet loss resulting in all of them reducing their transmission rates, which may lead to under utilization of the link in the near future. It takes considerable time for a TCP source to recover owing to its conservative additive increase policy. We note that a DropTail gateway uses buffer overflow as an indicator of congestion. The buffer remains almost always near its capacity, resulting in high delays and high packet loss rates. Given the very nature of TCP, this behaviour hampers TCP's transmission rates in particular and also has problems like global synchronization associated with it.

To counter some of the drawbacks in a DropTail gateway, Random Early Detection (RED), one of the initial Active Queue Management (AQM) policies was proposed by Floyd *et al* in [2]. Active Queue Management (AQM) policies are mechanisms for congestion avoidance, which pro-actively drop packets in order to provide an early congestion notification to the sources. RED uses an exponentially weighted

[4]Window size corresponds to the maximum amount amount of unacknowledged data, a TCP source can transmit.

average queue as a measure of congestion and starts dropping the packets randomly in anticipation of congestion based on the average queue even if the packet can be accommodated in the buffer. As a result, only a few TCP sources encounter packet loss and hence the overall link utilization remains high. Though better than DropTail, the performance of RED depends heavily on its parameter settings which is still an inexact science. We discuss RED and other subsequent AQM schemes and the problems associated with it in more detail in Chapter 2.

There has always been a need for an AQM policy that would keep the instantaneous queue stable, achieve higher link utilization keeping delay and packet loss rate low. In addition, the scheme should not be very sensitive to its parameter values and should perform over a broad traffic distribution. These are exactly the issues that our scheme APACE addresses successfully.

In Chapter 2 we propose a new Adaptive Prediction based Approach for Congestion Estimation in Active Queue Management (APACE) that predicts the instantaneous queue length at a future time using adaptive filtering techniques. We compare the performance of APACE with other existing AQM schemes like RED [2], Stabilized RED (SRED) [3], Adaptive Virtual Queue (AVQ) [4] and Predictive AQM (PAQM) [5] in networks having single and multiple bottleneck links. We show that APACE is indeed able to control the oscillations in the instantaneous queue. Moreover, APACE performs remarkably well by achieving higher link utilization[5] and a lower packet loss for a given delay[6], especially in networks with multiple bottleneck links.

APACE is not very sensitive to parameter settings, adapts quickly to changes in network conditions and can achieve a given delay or packet loss by varying just *one* parameter, thus giving the network operator a better tool to manage congestion, give delay guarantees and improve the performance of the network. In fact link utilization in APACE remains remains almost constant at a high value irrespective of its parameter settings, thus enabling the network operator to improve on other performance metrices (such as delay and packet loss) keeping the link utilization high. We explain the effects on the performance of APACE owing to its parameters in Chapter 3.

[5]By link utilization, we refer to the number of packets transmitted successfully on the particular link.

[6]By delay, we refer to the average queuing delay.

1.2.2 Learning TCP (LTCP)

In our efforts to combat the unwanted repercussions of congestion at the end hosts, in Chapter 4, we propose a modification to the TCP algorithm to improve the end-to-end performance in heterogeneous[7] networks. The performance of TCP deteriorates when it operates over the networks in which packet losses can also occur due to reasons other than congestion. This degradation in performance is caused by the rigidness of always assuming the cause of packet loss as congestion and subsequent significant reduction in window. Earlier research work have proposed many approaches on improving TCP over such networks, but have focused on particular network characteristics, *ie,* for wireless channel, mobility of host, etc. We propose an approach, Learning TCP (LTCP) in which a TCP attempts to learn the cause of packet loss adaptively and takes appropriate measures subsequently. LTCP is a comprehensive algorithm that can be applied to a wide variety of networks.

1.2.3 Scepter: A Bandwidth Management Tool

The current token bucket mechanism used to mark the traffic though adequate for shaping non-responsive constant rate traffic fails to guarantee, even the committed bandwidth, to responsive best-effort TCP flows. It can only give long-term bandwidth guarantees is unable to do so at shorter time scales.

Motivated by the fact, that in order to improve the goodput[8] of TCP aggregates, we do not want that all TCP sources in a given aggregate drop their window sizes at the same time. This may be inevitable in a heavy congestion scenario, but we would like to minimize the occurrence of such a phenomenon. The motivation for this, as mentioned before, is that the TCP source takes considerable time to recover from a drop in window owing to packet loss because of its conservative additive increase policy. Thus, in order to allocate and guarantee bandwidth to a TCP aggregate, we need a token bucket (Section 5.1) that instead of dropping all incoming packets for paucity of tokens should anticipate congestion and preemptively start dropping packets randomly. Most importantly, we need to keep the number of tokens in the

[7] A heterogeneous network need not necessarily have a wireless link. It can be an entirely wired network or a totally wireless network or a mix of both.

[8] In this thesis, goodput is the number of packets per unit time that have successfully reached the destination host.

bucket at a constant level. This would imply that, the number of tokens coming into the bucket, which corresponds to the bandwidth to be allocated, is equal to the number of tokens being consumed, thus ensuring a guaranteed bandwidth to the aggregate.

Scepter aims to improve the performance of best-effort TCP traffic by enabling aggregates operate very close to their allocated bandwidth. Scepter will enable Internet Service Providers (ISPs) to allocate its customers, the bandwidth they demand (could be 2Mbps, 1.85Mbps or anything) with very high precision and minimum over provisioning of the network resources by the ISP. Scepter achieves its objectives by modification in the token bucket of the traffic marker at the ingress router. We describe our this bandwidth management tool, Scepter in Chapter 5

1.3 The Road Ahead

In this thesis, we suggest alternatives to the most commonly used protocols and algorithms. We hope that they would, therefore find immense utility and have maximal impact in an effort towards improving Quality of Service (QoS) of best-effort responsive traffic in the Internet. We conclude and discuss directions for future work in Chapter 6.

CHAPTER

2

Adaptive Prediction based Approach for Congestion Estimation in Active Queue Management (APACE)

The Queue Principle: *The longer you wait in line, the greater the likelihood that you are standing in the wrong line*

In this chapter, we explain the following:

- *A new Active Queue Management (AQM) scheme called APACE*
- *Concept of Operating Point*
- *Significance of the various parameters in APACE*
- *Comparison of APACE with other existing AQM schemes*

Chapter 2

An Adaptive Prediction based Approach for Congestion Estimation in Active Queue Management (APACE)

2.1 Introduction to APACE

TCP (and its variants) remains the dominant end-to-end congestion control mechanism deployed in the Internet. The essence of this mechanism is that a TCP source adjusts its window size based on an implicit feedback about the congestion in the network. This implicit feedback is in the form of lack of receipt of acknowledgment from the receiver within a certain time out interval or the receipt of three duplicate acknowledgments. Either of these feedbacks is taken as an indication of packet loss. In a simple DropTail node, a packet gets dropped whenever the buffer is full. In networks with large Round Trip Times (RTT) and subsequently longer time out periods, the TCP congestion control has slower response to a packet drop. Thus it is desirable that the sender be notified early so as to adjust its rate pre-emptively in order to avoid congestion in the bottleneck node. Active Queue Management (AQM) policies attempt to estimate the congestion at a node and signal the incipient congestion by dropping packet(s) before the buffer is full. A responsive congestion control strategy then reduces its transmission rate. This helps in avoiding further congestion and is

expected to reduce the packet loss rate and keep the average queue size low. But if packets are dropped aggressively, then the capacity of the node may remain underutilized. An AQM policy thus has two components - one component estimates the congestion and another component takes the packet drop decision. The performance, therefore, depends upon how aggressive or conservative the estimate of congestion is and also on how aggressively the packets are dropped based on this estimate.

In this chapter, we propose and analyze extensively a new AQM strategy called APACE. We also attempt to give a general framework in terms of *operating points* (Section 2.3) for evaluating any AQM policy. We use this framework later to compare APACE with other existing AQM schemes like RED [2], SRED [3], AVQ [4] and PAQM [5] in networks having single and multiple bottleneck links. Our simulation results indicate that the APACE scheme is able to learn the changes in network conditions much faster than others. It is able to make a better decision regarding a packet drop. Moreover, APACE performs remarkably well by achieving higher link utilization and a lower packet loss for a given delay in networks with multiple bottleneck links. APACE is able to effectively control the oscillations in the instantaneous queue. APACE is not very sensitive to its parameter settings and can achieve a given delay or packet loss and/or link utilization by varying just *one* parameter, thus giving the network operator a better tool to manage congestion, give delay guarantees and improve the performance of the network. In fact, link utilization in APACE remains almost constant at a high value irrespective of its parameter settings, thus enabling the network operator to improve on other performance metrices (such as delay and packet loss) keeping the link utilization high.

The rest of the chapter is organized as follows. In Section 2.2, we discuss some of the related work in the area of queue management. We then explain the concept of operating point in Section 2.3, the APACE algorithm in Section 2.4, discuss prediction accuracy in Section 2.5, significance of various parameters in Section 2.6. We compare the performance of our scheme with RED, SRED, AVQ, and PAQM in Section 2.7 for single bottleneck link scenarios and in Section 2.8 for a multiple bottleneck link scenario. We discuss some relevant issues in Section 2.9 and finally conclude in Section 2.10.

2.2 Related Work

The RED scheme was initially described and analyzed in [2] with the main aim of providing "congestion avoidance" by dropping packets in anticipation of congestion. The performance of the RED algorithm depends significantly upon the setting of each of its parameters, *ie,* w_q, max_p, min_{th} and max_{th}. Though significant progress has been made towards the understanding of tuning RED parameters, it is a difficult problem and has not yet been solved satisfactorily. The difficulty in tuning the parameters of RED under different network conditions has limited its effectiveness. Experiments have shown that it is difficult to find appropriate values of parameters that will enable RED gateways to perform equally well under different congestion scenarios. Incorrectly tuned parameters may in fact cause RED to perform worse than Drop tail. In [6, 7], the authors have questioned the benefits of RED by performing experiments on testbeds. The authors have also recommended more research with realistic network settings in order to understand RED better before its wide scale deployment in the Internet. They also argue that RED performs well only under single bottleneck gateways and heavy TCP traffic. These are also the cases for which most of the simulations have been performed and reported. The performance of RED and most other schemes under multiple bottleneck gateways has not yet been satisfactorily studied. Various authors [8, 9, 10] have given guidelines and proposals for setting RED parameters or to adaptively vary them.

In [11], Hollot *et al.* have studied the problem of tuning RED parameters from a control theoretic stand point. The aim was to improve the throughput by controlling oscillations in the instantaneous queue.

Feng et al. [12] proposed a mechanism for adaptively varying one of the RED parameters, max_p, with the aim of reducing the packet loss rates across congested links. Floyd et al. in [13] discusses the algorithmic modifications to the self-configuring RED algorithm [12] for tuning max_p adaptively. Their objective was to control the average queue length around a pre-decided target. The choice of the target queue size, left to the network operator, determines the trade-off between delay and link utilization. Controlling the average queue size, however has a limited impact on regulating the packet loss rate. Balanced RED (BRED) [14] and Fair RED [15] aim to improve the fairness of RED by maintaining per-active-flow state information.

Adaptive Virtual Queue (AVQ) [4] tries to decouple congestion measure from the

performance measure. Stochastic Fair Blue (SFB) [16] attempts to enforce fairness among a large number of flows. It handles and rate limits non-responsive flows effectively using an extremely small amount of state information. CHOKe (CHOose and Keep for responsive flows and CHOose and Kill for unresponsive flows) [17] also aims to ensure fairness to each of the flows that share the outgoing link.

Predictive AQM (PAQM) [5] tries to exploit traffic predictability in the calculation of the packet dropping probability. The authors have shown that the correlation structure present in long-range dependent traffic can be used to accurately predict the future traffic. PAQM enables the link capacity to be fully utilized without incurring excessive packet loss by stabilizing the instantaneous queue to a desired level. This scheme, however is very computation intensive.

All these schemes aim and achieve a particular target, but we need an AQM policy that would keep the instantaneous queue stable, achieve higher link utilization keeping delay and packet loss rate low. In addition, the scheme should not be very sensitive to its parameter values, should perform over a broad traffic distribution and must not be very computation intensive. These are exactly the issues that APACE addresses successfully.

2.3 Operating Point

An AQM policy can be used to control the performance metrices such as link utilization, the average queuing delay and the packet loss rate. An AQM policy should give the network operator the freedom to specify the required performance metrices and should be able to meet the requirements to its best. In other words, one should be able to specify the *operating point* that one wants to achieve given a particular network scenario. The operating point, for instance can be specified in terms of link utilization, average queuing delay and packet loss rate that one wants to achieve. We denote such an operating point by (t^*, d^*, l^*), where t^* is the target link utilization, d^* is the target average queuing delay and l^* denotes the target packet loss rate. In order to keep our representation simple and be able to visualize the graphs in two dimension, in this thesis we define operating points as (t^*, d^*) and $(l^*, d^*,)$. It is possible that a desired operating point might not be achievable. In that case, the aim of the AQM policy should be to approach this operating point as closely as possible.

In case of a Drop Tail gateway, there is only one operating point for a given

network scenario since the packets are dropped only when the buffer is full. In RED and APACE, the parameters can be suitably adjusted resulting in greater choices of operating points. Setting the appropriate parameters is difficult in case of RED, but as will be illustrated in this thesis, in APACE we need to vary only one parameter to achieve a given operating point.

2.4 Proposed Scheme

In APACE, we estimate the congestion by predicting the instantaneous queue length at a future time instant. This estimate is based on the queue lengths at the previous packet arrivals. The decision to drop any packet is based on the predicted value of the instantaneous queue length rather than the average queue length, as in the case of RED. As will be shown later in this chapter, this makes the scheme more responsive especially in scenarios with changing network conditions. We now explain the APACE scheme in detail.

2.4.1 Predicting the Instantaneous Queue

We predict the instantaneous queue length using the Normalized Least Mean Square (NLMS) algorithm [18]. Our simulation results show that the NLMS predictor can be used to get a good estimate of the instantaneous queue length under a large set of network scenarios (different kinds of sources, topologies *etc*). Moreover, the algorithm takes only a few iterations to converge and adapts well under changing network scenarios.

The instantaneous queue length prediction is made at every packet arrival. Let M denote the order of the NLMS filter used for prediction, $q(n)$ the instantaneous queue length at the nth packet arrival and $\bar{q}(n)$ a Mx1 vector of the instantaneous queue lengths of the past M packet arrivals. The instantaneous queue length after N_0 packet arrivals from the n^{th} packet is predicted based on $\bar{q}(n)$. We call N_0 the prediction parameter. We denote the predicted queue length by $\hat{q}(n+N_0)$. $\hat{q}(n+N_0)$ is calculated as follows:

$$\hat{q}(n+N_0) = \bar{w}_q^T(n) * \bar{q}(n) \tag{2.1}$$

In the above equation, $\bar{w}_q(n)$ denotes a Mx1 weight vector. These weights are

updated dynamically based on the error between the predicted and the actual queue length. The error, $e(n)$ in the prediction is computed as

$$e(n+N_0) = q(n+N_0) - \bar{w}_q^T(n) * \bar{q}(n) \tag{2.2}$$

The queue weights are updated using the following equation:

$$\bar{w}_q(n+1) = \bar{w}_q(n) + \mu(n) * \bar{q}(n) * e(n) \tag{2.3}$$

In the NLMS algorithm, $\mu(n)$ is calculated using the following equation:

$$\mu(n) = \frac{\mu_0}{1 + \bar{q}^T(n)\bar{q}(n)} \tag{2.4}$$

The queue weights are initially set to a fixed value and are later updated using the above equations. Typically, the weights are initially set to 0. μ_0 has been set to 0.01. A small value of μ_0 implies guaranteed convergence of the NLMS algorithm though at a slower rate. A large value of μ_0 though increases the rate of convergence, may cause it to diverge.

2.4.2 Taking a Packet Drop Decision

As stated earlier, the decision to drop the incoming packet is based on the predicted value of the instantaneous queue length. The incoming packet is dropped with a probability p that is calculated based on $\hat{q}(n+N_0)$. The algorithm for dropping the incoming packet(s) is illustrated in Figure 2.1. Let B denote the maximum buffer size. If $\hat{q}(n+N_0) < \alpha * B$, no packet is dropped ($\alpha$ is a positive constant less than 1). If $\hat{q}(n+N_0) > B$, every incoming packet is dropped. If $\alpha * B \leq \hat{q}(n+N_0) \leq B$, the incoming packet is dropped with a probability p, which is a function of the predicted queue size. For the purpose of simulations, we vary the probability p linearly from 0 at αB to max_p at B. The motivation behind linearly increasing the packet dropping probability is to make the scheme more aggressive as the predicted queue length increases. The packet dropping probability, p can thus be expressed as:

$$p \leftarrow \frac{max_p(\hat{q}(n+N_0) - \alpha * B)}{(1-\alpha) * B} \tag{2.5}$$

On every packet arrival

- Predict instantaneous queue size (after N_0 packet arrivals):
  $\hat{q}(n+N_0) \leftarrow \bar{w}_q^T(n) * \bar{q}(n)$

- Calculate packet dropping probability p:

  - If $\alpha * B \leq \hat{q}(n+N_0) \leq B$

    $p \leftarrow \frac{max_p(\hat{q}(n+N_0)-\alpha*B)}{(1-\alpha)*B}$

  - else if $\hat{q}(n+N_0) \geq B$
    $p = 1$

  - else if $\hat{q}(n+N_0) \leq \alpha * B$
    $p = 0$

- Update queue weights using the NLMS algorithm

Figure 2.1: The APACE algorithm

2.5 Prediction Accuracy

The first issue that needs to be addressed is whether the NLMS algorithm can predict the instantaneous queue length accurately. We test the performance of NLMS algorithm under different network loads. The simulations have been performed using the network simulator, **ns v2.1b8a** [19]. The network topology shown in Figure 2.2 has been used for simulations.

Figure 2.3 illustrates the mean square error in the actual and the predicted value of the instantaneous queue size (learning curve). The plot has been obtained for 25 TCP sources. The packet loss rate is relatively high $\approx (5-10)\%$. The mean square

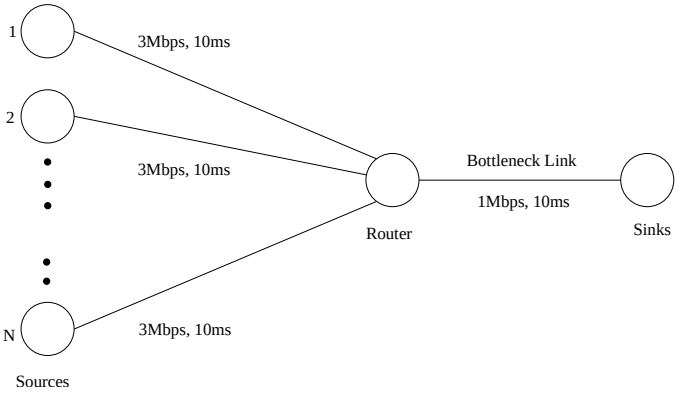


Figure 2.2: Network Topology

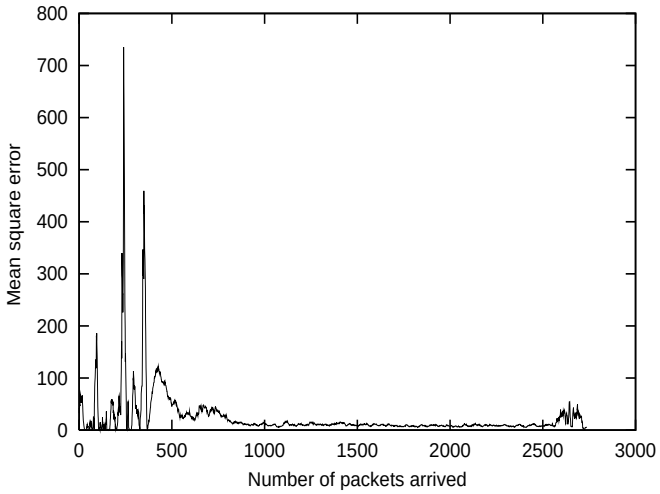


Figure 2.3: Learning curve

error has been averaged over 100 sample paths. As can be seen from the figure, the NLMS algorithm converges quickly and is able to predict the instantaneous queue to a reasonable accuracy (residual error[1] $\approx$ 8 square packets). We get similar learning curves even for fluctuating network loads (described in Section 2.7), though the prediction in case of sustained heavy traffic is better as compared to other scenarios with lower or fluctuating network loads because under conditions of heavy congestion, the aggregate incoming traffic characteristics do not vary much. Moreover, even under fluctuating network loads, the algorithm is able to adapt to the network conditions in a few iterations only and its estimation remains robust. The error in the queue

[1]By residual error, we refer to the steady state mean square error in predicting the instantaneous queue.

prediction for the same is shown in Figure 2.4.

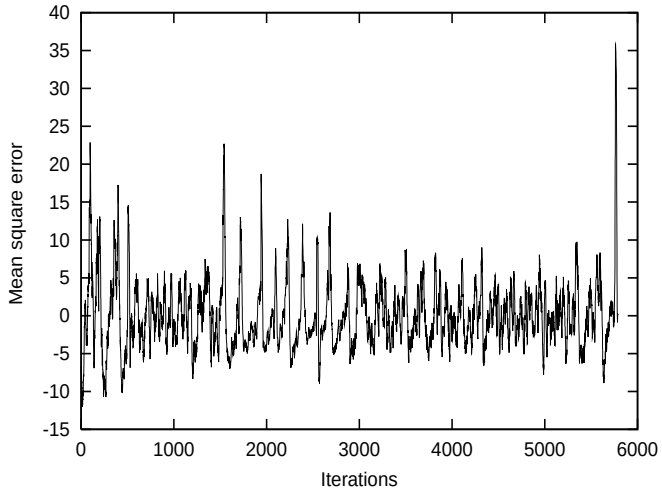


Figure 2.4: Error for fluctuating traffic scenario

Based on the above discussion and simulation results, we can conclude that the NLMS algorithm does a good prediction of the instantaneous queue for various traffic scenarios.

2.6 Significance of various parameters

The various parameters of the scheme are listed in Table 2.1. Our simulation results

Table 2.1: APACE Parameters

Parameter	Function
M	Order of the filter
N_0	Prediction parameter for the filter
max_p	Maximum dropping probability
α	Decides the lower threshold below which no incoming packet(s) is dropped
β	Decides the upper threshold above which every incoming packet is dropped

indicate that the order of the filter, M does not have a significant impact on the

performance of the scheme. However, for large values of $M(\geq 50)$, the NLMS algorithm does not converge. This is because the NLMS algorithm typically takes $20M$ iterations to converge and during this period the traffic arriving at the queue might vary substantially for large M. Moreover, the computation complexity of the scheme increases with increasing M. Therefore, it is advisable to take a low value for the order of the filter. We suggest that $M \approx (5 - 20)$ is a good value for the order of the filter and the performance of the scheme is not sensitive to M in this range. For the purposes of simulations, we have chosen $M = 10$.

The prediction parameter, N_0 decides the trade-off between the accuracy of the prediction and how early the prediction is made, *ie*, a lower value of N_0 means that the prediction is made over a relatively shorter time scale. In such a scenario, the prediction error is typically low. On the other hand, by taking a large value of N_0, the prediction is made over a longer time scale but the prediction error might be large. The value of N_0 should ideally depend on the round trip time to the source as well. N_0 should be large enough so that the effect of dropping a packet results in an early congestion notification to the source. This implies that for large RTTs, one should keep a large value of N_0 though this might result in a greater residual error in prediction. Keeping a small value of N_0 in scenarios with large RTTs might result in a delay in the congestion estimation (since the prediction instant is not far enough in future) as well as a delay in the congestion notification (owing to a large RTT) to the source.

Our simulation results in Chapter 3 also illustrate that N_0 does not have a strong bearing on the performance (in terms of packet loss rate, average queuing delay and link utilization) of the scheme. We have reported simulations results for $M = 10$ and $N_0 = 15$. An extensive simulations study in support of the claims made can be found in the next chapter.

The parameter max_p governs how aggressively the packets are being dropped, based on the predicted value of the instantaneous queue length and α determines the buffer occupancy at which we should start dropping packets. The effects of these parameters on the performance has been explained in detail in the next chapter. Extensive simulations by varying all the parameters indicate that the scheme gives similar performance under a wide range of parameter settings. The scheme adapts itself to the network conditions well, thus reducing the importance of initial parameter settings. We suggest $max_p = 0.2$, $\alpha = 0.3$, $N_0 = 15$, $M = 10$ as the default values.

2.7 Comparison with other Queuing Strategies

In this section we compare the performance of APACE with other queuing schemes like RED, SRED, PAQM and AVQ. The network topology shown in Figure 2.2 is used for performing the simulations. The results for multiple bottleneck scenario is explained in the next section. The number of TCP sources, N, is varied to achieve different incoming traffic loads. Packet size has been fixed to 500 bytes. The buffer size at the router is of 20 packets for SRED and 50 for all others. The reason for choosing 20 is the fact that SRED always tries to keep the buffer close to full. The results (except for the instantaneous queue) have been averaged over 20 sample paths.

2.7.1 Instantaneous queue length stability

We first address the issue of instantaneous queue stability. In our simulation 40 TCP sources are switched on randomly in the first two seconds and the simulation is performed for 40 seconds. Figure 2.5 illustrates the instantaneous queue size at the gateway for various queuing strategies. We have used $w_q = 0.002$, $min_{th} = 5$, $max_{th} = 15$, $max_p = 0.1$ for RED, default APACE parameters, $M = 1000$, $\alpha = 1/M$, and $p_{max} = 0.15$ for SRED, $\gamma = 0.98$, $\alpha = 0.10$ for AVQ and $Q_{opt} = 20$ for PAQM as the queue in APACE is stable at 20.

As is evident from the plots, RED is unable to control the oscillations in the instantaneous queue, while APACE and PAQM provide reasonable stability to the instantaneous queue. Moreover, even under steady heavy traffic, the instantaneous queue in RED becomes empty frequently (as shown in Figure 2.5) leading to severe under utilization of the link. In addition, the average queue size in RED deviates significantly from the instantaneous queue. This entices us to look for better indicators of congestion than the average queue length. The decision to drop a packet based on the average queue size also introduces delay in the estimation of congestion as the learning is slower. As a result, RED might take a wrong decision regarding a packet drop. SRED has problems of global synchronization similar to DropTail since it always keeps its buffer close to full which keeps overflowing.

We next perform the simulation under fluctuating network loads. We switch on 40 sources within a small interval of time. After about 10 seconds of the simulation, 36 of these sources are switched off resulting in a drastic decrease in the incoming traffic. At about 20 seconds from the start of the simulation, 20 new TCP sources are switched on

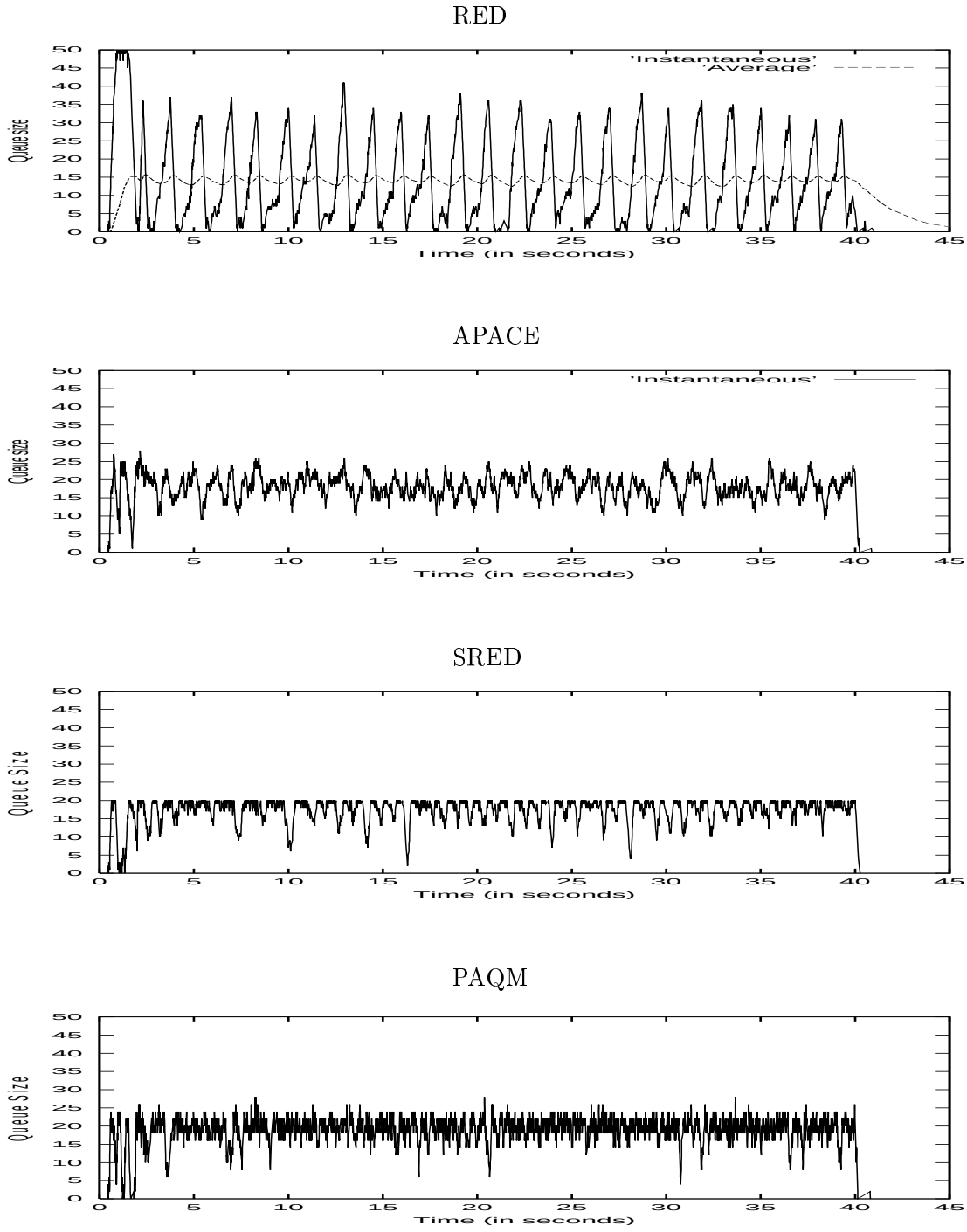


Figure 2.5: Instantaneous queue at RED, APACE, SRED and PAQM routers under heavy traffic conditions

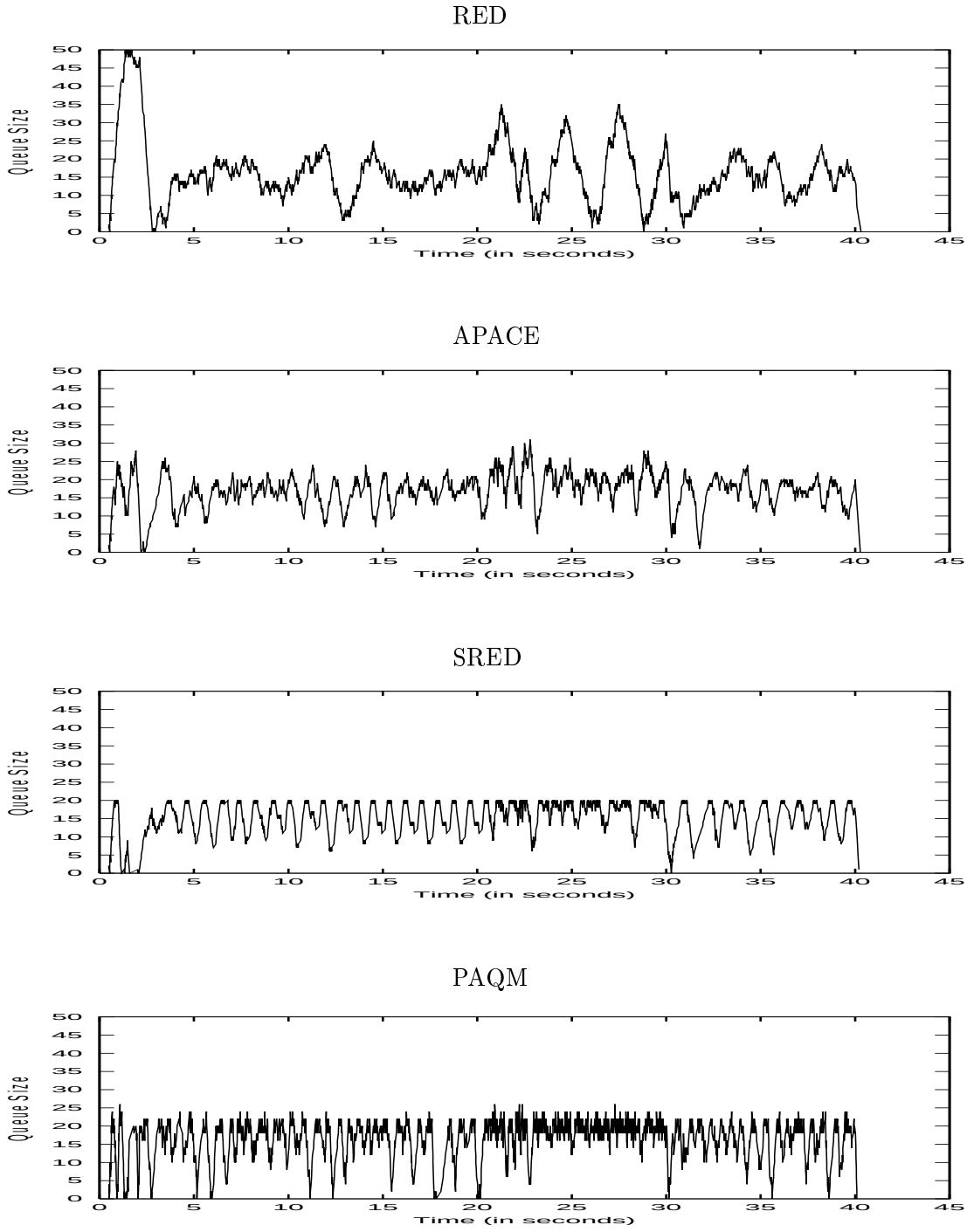


Figure 2.6: Instantaneous queue at RED, APACE, SRED and PAQM routers under fluctuating traffic conditions

resulting in a sudden increase in the incoming traffic and 10 seconds later additional 16 TCP sources are switched on. These 40 TCP sources run till 40 seconds from the start where the simulation is terminated. Even under such fluctuating network loads, the NLMS algorithm adapts very well and is able to predict the instantaneous queue accurately. The error in prediction of the instantaneous queue is plotted in Figure 2.4. We note that at points where there is a sudden change in the incoming traffic, the prediction error increases. However, the NLMS algorithm is able to converge and predict the instantaneous queue accurately and quickly.

The instantaneous queues are shown in Figure 2.6. The instantaneous queues are kept again around 20 for the sake of comparison. Conclusions similar to above can be drawn from it. Though the queue in SRED seems to be more stable, there are other problems like global synchronization attached to it as explained above. APACE is indeed able to adapt well to the changes in network conditions and in maintaining the stability of the instantaneous queue.

2.7.2 Link utilization

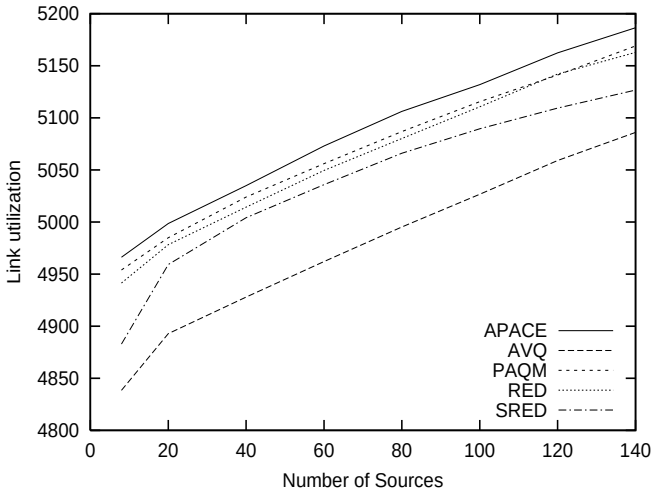


Figure 2.7: Link utilization vs. number of sources in a single bottleneck scenario

Figure 2.7 shows the number of packets transmitted successfully at the bottleneck node as the number of TCP connections are increased. We observe that APACE is able to successfully transmit more packets than any other scheme under inspection.

2.7.3 Packet loss rate

Figure 2.8 illustrates the fraction of packets dropped at the bottleneck node as the number of TCP connections are increased. As can be seen from the figure, APACE shows a consistent improvement in the packet loss rate. This improvement is more prominent under heavy network loads (larger number of TCP sources) because under mild congestion scenarios, the packet loss, as such, is quite low.

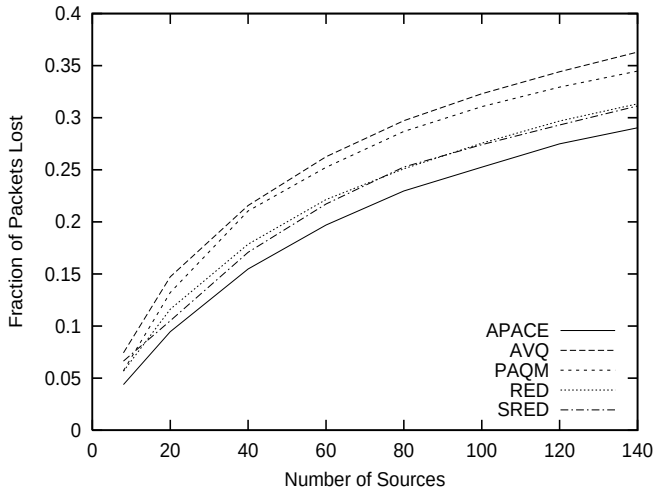


Figure 2.8: Fraction of packets lost vs. number of sources in a single bottleneck scenario

2.7.4 Trade-off Comparison with RED

It should be noted that it is unfair to compare APACE and RED by fixing any one set of parameters. In fact, the above statement holds true for any scheme whose performance needs to be compared with RED. One might achieve an entirely different performance for some other setting of RED parameters. Moreover, using only one performance metric to compare any scheme with RED is also not entirely correct because finally there is a trade-off between various performance metrics such as delay-link utilization or delay-loss. Also we need to take into consideration the effects of various parameters on the performance metrics.

Hence, we now compare the performance of APACE scheme with RED in terms of the delay-link utilization trade-off curves (refer Figures 2.9, 2.10) and delay-loss trade-off curves (refer Figures 2.11, 2.12) that can be achieved. The buffer size is 250

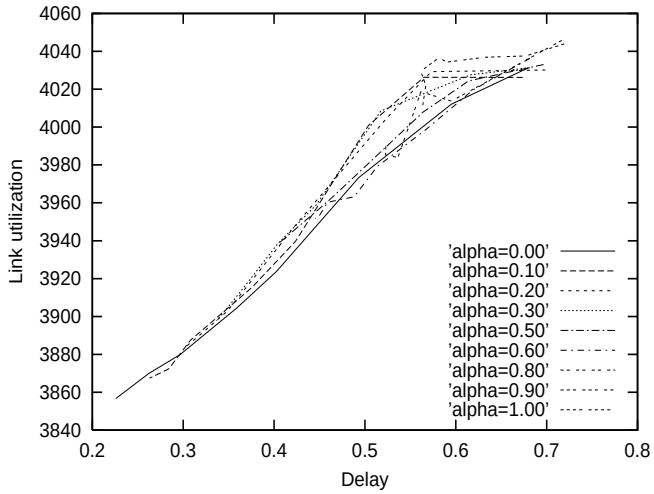


Figure 2.9: Delay-link utilization trade-off curves for single bottleneck scenario APACE

packets. The simulations have been performed with 80 TCP sources.

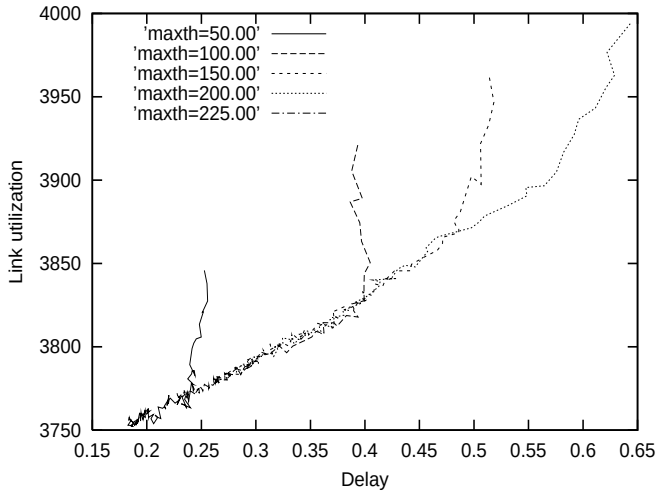


Figure 2.10: Delay-link utilization trade-off curves for single bottleneck scenario with RED

Each curve corresponds to a different value of max_{th} (α) for RED (APACE). The extreme right point on each curve corresponds to $max_p = 0$ and as we move along the curve, we get points corresponding to high values of max_p. It is worth noting that the plots for different values of α are close to each other and in particular, the plot with $\alpha = 0$, encompasses the complete range of the delay-link utilization trade-off plane covered by the other plots (for different values of α). Therefore, by setting $\alpha = 0$

and varying only max_p, one can span all the achievable delay-link utilization trade-off operating points. An operating point below the delay-link utilization trade-off curve is an *achievable* operating point, while the one above it is *unachievable*. By an achievable operating point, we mean that given a particular delay-link-utilization pair that one wants to achieve, an operating point with a higher link-utilization and lower delay can be achieved. To get the suitable setting of APACE parameters, that can achieve a desired operating point, a simple method is to join the given operating point to the origin and extend the line. The point of intersection of this line segment with the delay-link utilization trade-off curve(s) gives the values of the APACE parameters and the corresponding average queuing delay and link utilization.

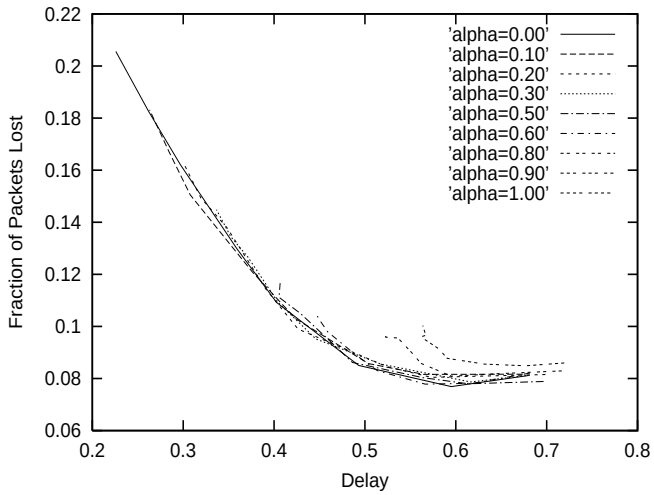


Figure 2.11: Delay-loss for trade-off curves for single bottleneck scenario with APACE

As can be seen from Figures 2.9 and 2.10, APACE gives a better link utilization and also lower delay than RED. Also any operating point that is achievable by RED can be achieved by APACE as well by a suitable setting of parameters. Moreover, for the APACE scheme (refer Figures 2.9, 2.11) one can cover the entire range of operating points by keeping $\alpha = 0$ and varying max_p only. The above observations are also true when the buffer size is 50 packets. The same however cannot be said for RED and both the parameters, max_p and max_{th} need to be varied (Figure 2.10) in order to achieve a certain operating point. This makes APACE scheme easy to adapt to network conditions. Varying only max_p *adaptively* based on the network traffic, we can achieve a better AQM strategy than the existing ones.

We have compared the performance with only RED here, mainly to illustrate

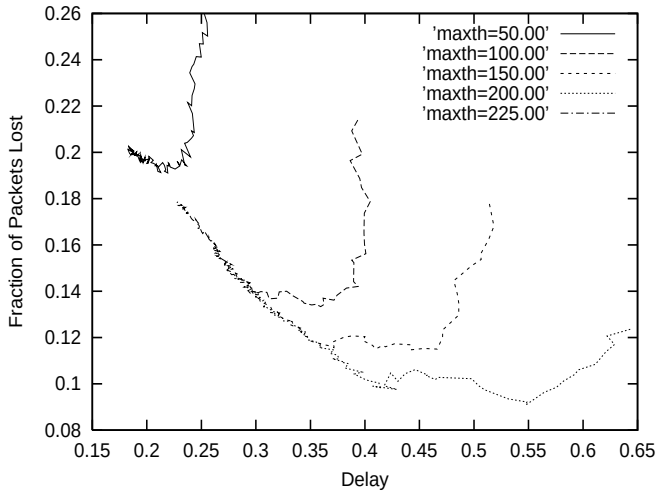


Figure 2.12: Delay-loss trade-off curves for single bottleneck scenario with RED

the significance of the concept of trade-off curves. Trade-off comparisons with other schemes is reported in the next section for multiple bottleneck link scenario, which is also more important. From the above we can conclude that the APACE gives a better link utilization with lower delay and only one parameter max_p needs to be varied to achieve any given feasible operating point.

2.8 Performance under Multiple Bottleneck Links

We now compare the performance of the various AQM schemes in networks having multiple congested bottleneck links. The network topology is shown in Figure 2.13. There are "N" TCP sources connected to router 1 and a cross traffic of 25 TCP sources each flows from router 2 to router 3 and from router 4 to router 5 respectively. Packet size has been fixed to 500 bytes and buffer size to 50 packets. The results (except for instantaneous queue) have been averaged over 20 sample paths.

Routers 2 and 4 are the most congested nodes and hence show similar behavior. We have chosen router 2 for detailed study. We verify our earlier observation by extensive simulations similar to those in Chapter 3 that values of N_0 and M do not affect the performance significantly. The results are infact, so similar to the single bottleneck case in terms of dependence on N_0 and M, that we choose not to report them in this thesis to avoid repetition. We choose the same parameter settings as before, except that $Q_{opt} = 40$ for PAQM and a buffer size of 40 packets for SRED at

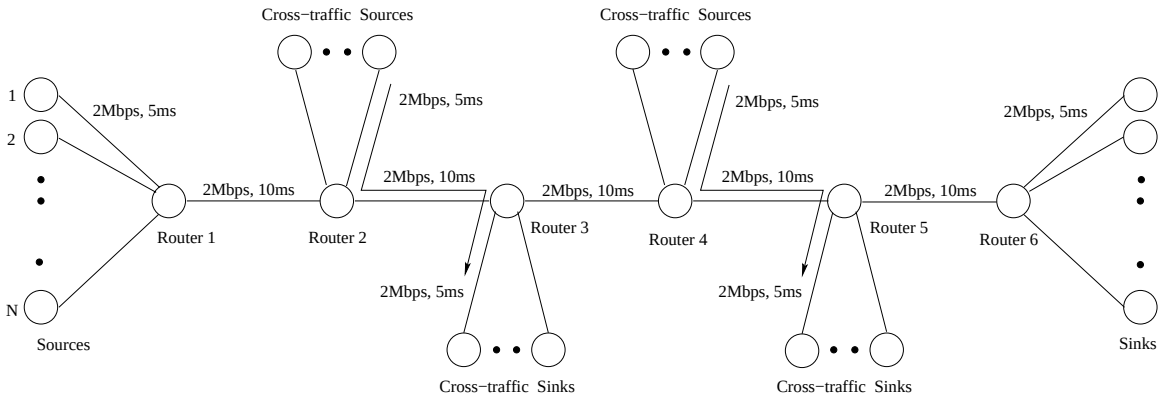


Figure 2.13: Network topology for multiple bottleneck links

all the five nodes.

2.8.1 Instantaneous Queue, Link utilization and Packet loss

The instantaneous queue length results are similar to those obtained for single bottleneck link. The queue occupancy is around 40 packets instead of 20 previously. We observe that APACE is able to keep the instantaneous queue stable in agreement to our observations for the single bottleneck case. This implies that the packet delay remains almost constant under the APACE scheme.

In addition APACE also gives better link utilization and lower packet loss as shown in Figures 2.15 and 2.16. Link utilization and fraction of packets lost are illustrated in Figures 2.15 and 2.16 respectively, as the number of TCP connections is increased. We observe that in networks with multiple bottleneck links, APACE is able to achieve high link utilization with low packet loss rate and a stable queue that keeps the delay bounded.

2.8.2 Trade-off Curves

The delay-link utilization and delay-loss trade-off curves for the various schemes are shown in Figures 2.17 and 2.18 respectively. The curves for APACE correspond to $\alpha = 0$ and varying max_p. Though APACE has better curves for other values of α, we plot the curves for $\alpha = 0$ while comparing it with others to illustrate the fact that we can indeed fix α at 0 or some other small value and vary only max_p and still get performance better than other schemes. For RED, the curves correspond to

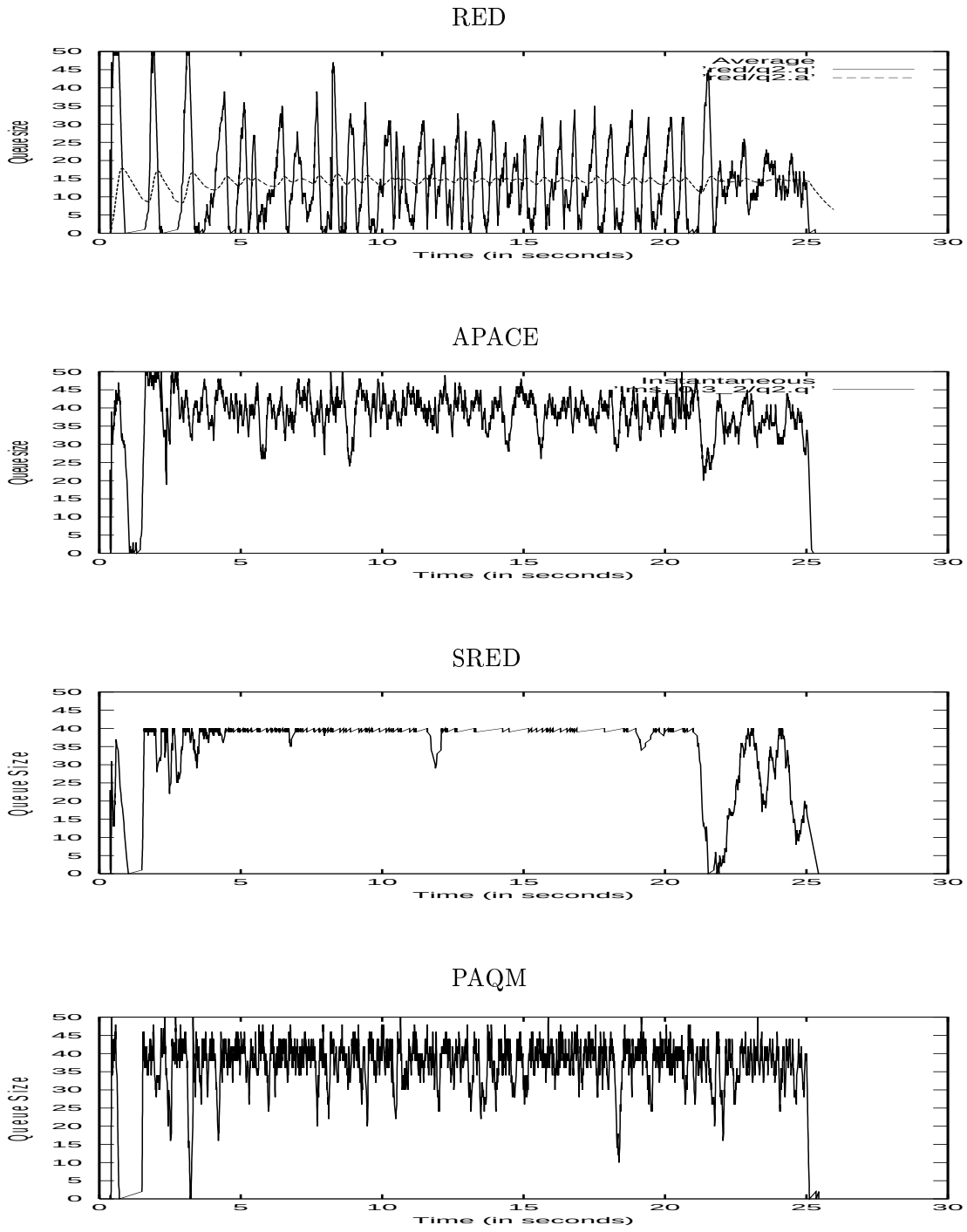


Figure 2.14: Instantaneous queue at RED, APACE, SRED and PAQM routers in a multiple bottleneck scenario

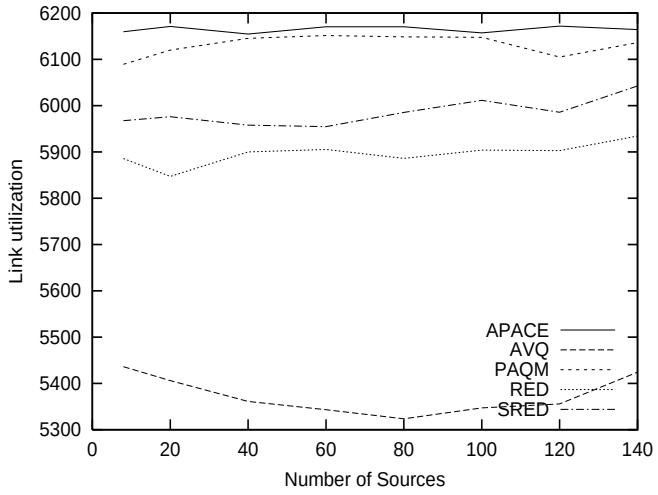


Figure 2.15: Link utilization vs. number of sources in a multiple bottleneck scenario

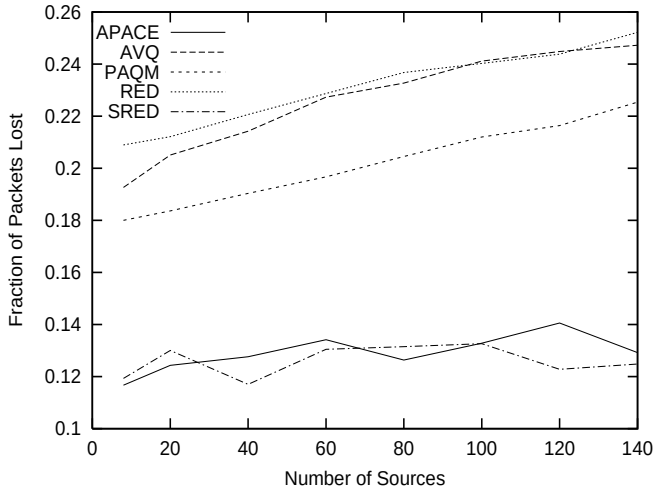


Figure 2.16: Fraction of packets lost vs. number of sources in a multiple bottleneck scenario

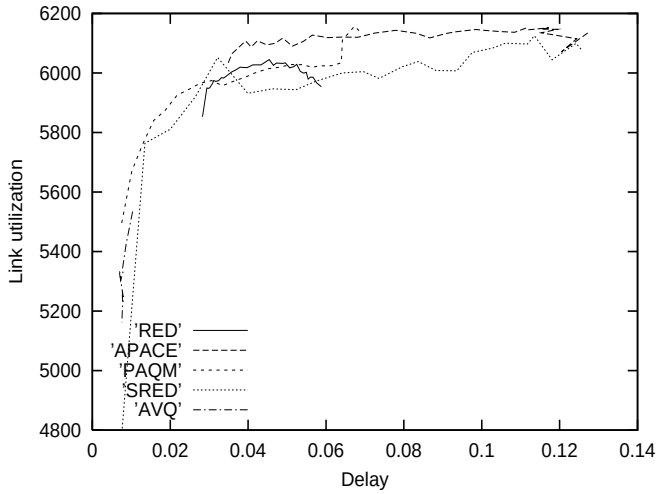


Figure 2.17: Delay-link utilization trade-off curves for multiple bottleneck scenario

$max_{th} = 15$ and varying max_p. Q_{opt} in PAQM and buffer size in SRED is varying from 3 to 50 as we move from left to right and α is varying from 0.05 to 0.99 in AVQ.

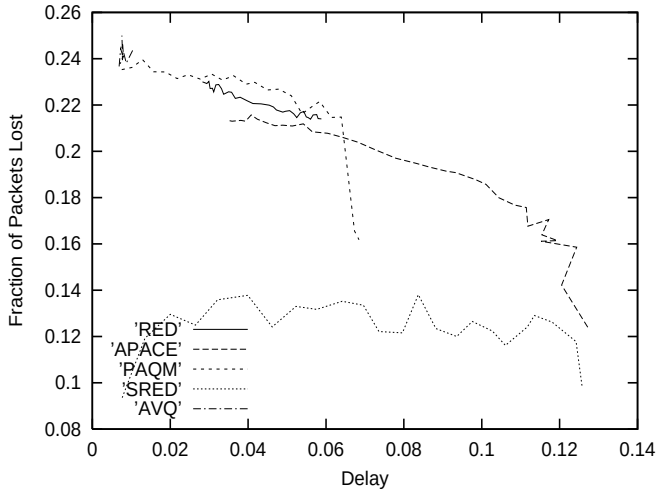


Figure 2.18: Delay-loss trade-off curves for multiple bottleneck scenario

We observe that for a given delay APACE achieves the highest link utilization at a much lower packet loss rate. Though the trade-off curves for SRED look better, it is associated with other problems like global synchronization owing to its frequent overflow of buffer. In addition link utilization in APACE remains almost constant at a high value indicating that it is quite independent of its parameter settings and hence the network operator can focus more on the delay and packet loss rate without worrying much about link utilization.

Note that the drop in the link utilization-delay trade-off curve in Figure 2.17(a) for $max_{th} = 15$ which is the default value for RED indicates that we would achieve lower link utilization and higher packet delays at least for the traffic scenario considered here (which is common) for higher values of max_p. This suggests that varying max_p adaptively may sometimes worsen the performance of RED under multiple bottleneck links. This parameter dependence in RED and its unexpected behavior is a cause of concern given that RED is being deployed extensively in routers these days.

2.8.3 Stability of Link Utilization

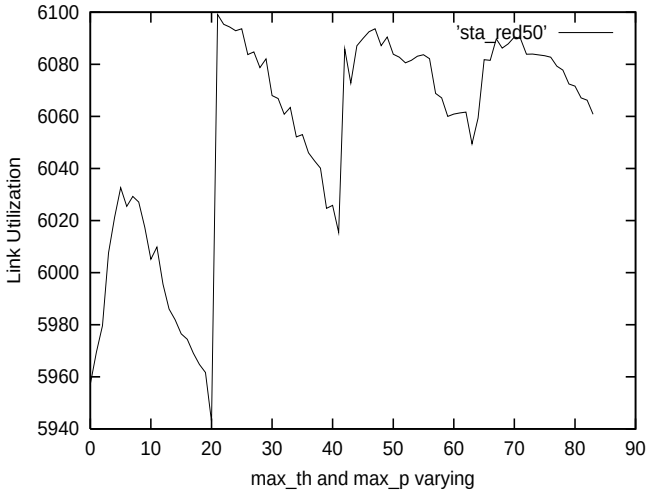


Figure 2.19: Link utilization of RED with various parameter settings

Figure 2.19 shows the link utilization of RED at router 2 as its parameters are varied. For RED we have taken 4 values for max_{th} *ie* 15, 30, 45, 50. For every value of max_{th}, we have varied max_p from 0.00 to 1.00. For these different values, we have plotted the link utilization on the Y-axis. Similarly for APACE, we have varied max_p from 0.00 to 1.00 and α from 0.0 to 1.0. The link utilization is plotted on the Y-axis of Figure 2.20. We observe that the link utilization remains fairly constant for the different values of its parameters as compared to RED. In fact the initial part of Figure 2.19 corresponds to $max_{th} = 15$, which is also its default value. The average packet delay and packet drop rate varies in both the schemes. Given this flexibility in parameter adjustment for link utilization in addition to the fact that the instantaneous queue remains fairly constant and its level of occupancy can be

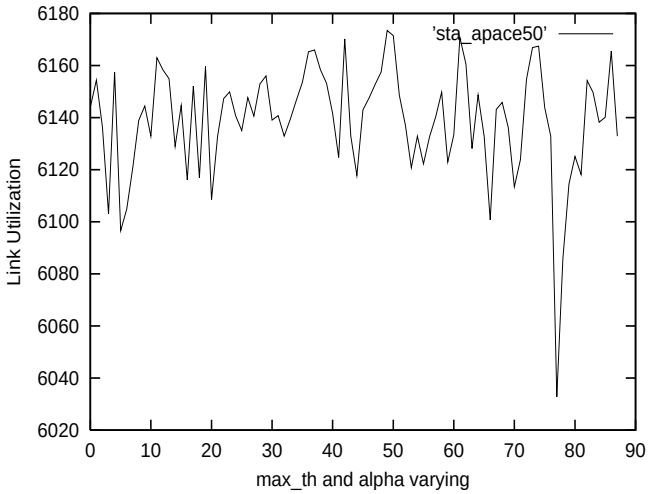


Figure 2.20: Link utilization of APACE with various parameter settings

controlled by varying only max_p can help substantially the service provider to give delay guarantees and achieve better link utilization.

2.9 Discussion

One can also consider a generalized version of the APACE scheme where the upper threshold for dropping the packets is kept as $\beta * B$ rather than B. Here β is a positive constant greater than 1. Such a scheme is able to achieve a greater range of operating points (with lower packet loss rates). However, it also introduces an additional parameter β. The APACE scheme presented is a special case of this scheme with $\beta = 1$. However, our preliminary simulation results with the generalized scheme show that the performance gain by the introduction of parameter β is not significant. We therefore suggest that $\beta = 1$ is a good enough value for most practical purposes.

Also, as discussed, the average queue size (in RED) might not give a true indication of the congestion. This suggests that a modified version of RED in which the decision to drop any packet is based on both the instantaneous and the average queue size might give a better performance as compared to RED. A simple way to this is to look at the difference between the instantaneous queue length and the average queue length before making a drop decision. The maximum packet dropping probability, max_p can then be varied based on this difference ($q_{instantaneous} - q_{average}$). For example, if the absolute value of the difference is greater than a certain threshold, max_p can

be decreased or increased based on whether the difference is negative or positive.

In the current version of the scheme, the prediction is made at every packet arrival. A scenario where the number of packet arrivals is large might lead to a lot of computational overhead. To overcome this problem, one might consider a scheme where the prediction is done at periodic time intervals, though this might come at the expense of lower prediction accuracy.

2.10 Conclusions

In this chapter, we have presented a novel AQM scheme based on (predicted) instantaneous queue length. We conclude that algorithms like NLMS can indeed be used to predict the instantaneous queue length. The APACE scheme performs better than existing AQM schemes as it adapts faster to changes in network conditions and is able to keep the instantaneous queue stable. APACE also gives higher link utilization and a much lower packet loss rate as compared to PAQM in addition to the instantaneous queue stability comparable to PAQM. In addition the scheme is able to achieve better operating points than that of PAQM and others (in terms of delay-link utilization or delay-loss trade-off) for both single and multiple bottleneck links. The link utilization of APACE remains more or less constant at a high value, in networks with multiple bottlenecks, with change in its parameters that can be varied suitably to achieve a given delay and packet loss rate without worrying much about link utilization. Moreover this can be achieved by varying just one of its parameters *ie.*, max_p. The performance of APACE is predictable depending on its parameters, unlike RED which sometimes can behave unexpectedly. With APACE we can thus achieve an AQM strategy better than most others. APACE with max_p being varied adaptively is expected to perform even better and is an area of future research.

CHAPTER 3

APACE - parameters in detail

Only two things are infinite, the universe and graphs in this chapter, and I'm not sure about the former

In this chapter, we explain the following:

- *Effects of N_0 and M on the performance of APACE*
- *Effects of max_p*
- *Effects of alpha (α)*

Chapter 3

APACE - parameters in detail

In this chapter, we provide extensive simulation results supporting our earlier claims regarding the performance of APACE being independent of the values of N_0 and M. We, then explain the effects of the parameters max_p and α on various performance metrics including packet loss rate, average queuing delay and link utilization. We use the same simulation set up as in Figure 2.2. We verify our claims for three types of network scenarios. The harsh scenario consists of 25 TCP sources with about $5-10\%$ packet loss rate. The mild scenario has just 5 TCP sources and $1-2\%$ packet loss rate. We have also done simulations for an extreme scenario, in which there are 80 TCP sources and about $20-25\%$ packet loss rate. We conduct simulations for buffer sizes of 50 and 250 packets and for both, single and multiple bottleneck networks. The mean packet size is 500 bytes.

3.1 Effects of N_0 and M

Though we have done extensive simulations for all the above mentioned scenarios [20], we report in this thesis the results for the harsh scenario with buffer size of 50 packets only. The results are and the conclusions that can be drawn from the other scenarios are identical and hence we skip them in order to avoid repetition. The complete set of simulations is can be accessed at [20].

Figures 3.1, 3.2 and 3.3 illustrate the probability of packet loss, packet delay and link utilization with max_p as the independent variable. Each figure has 12 sub-figures, each of which corresponds to a particular pair of (N_0, M). The prediction parameter, N_0 is a measure of how far in time, the prediction is made. Computational complexity

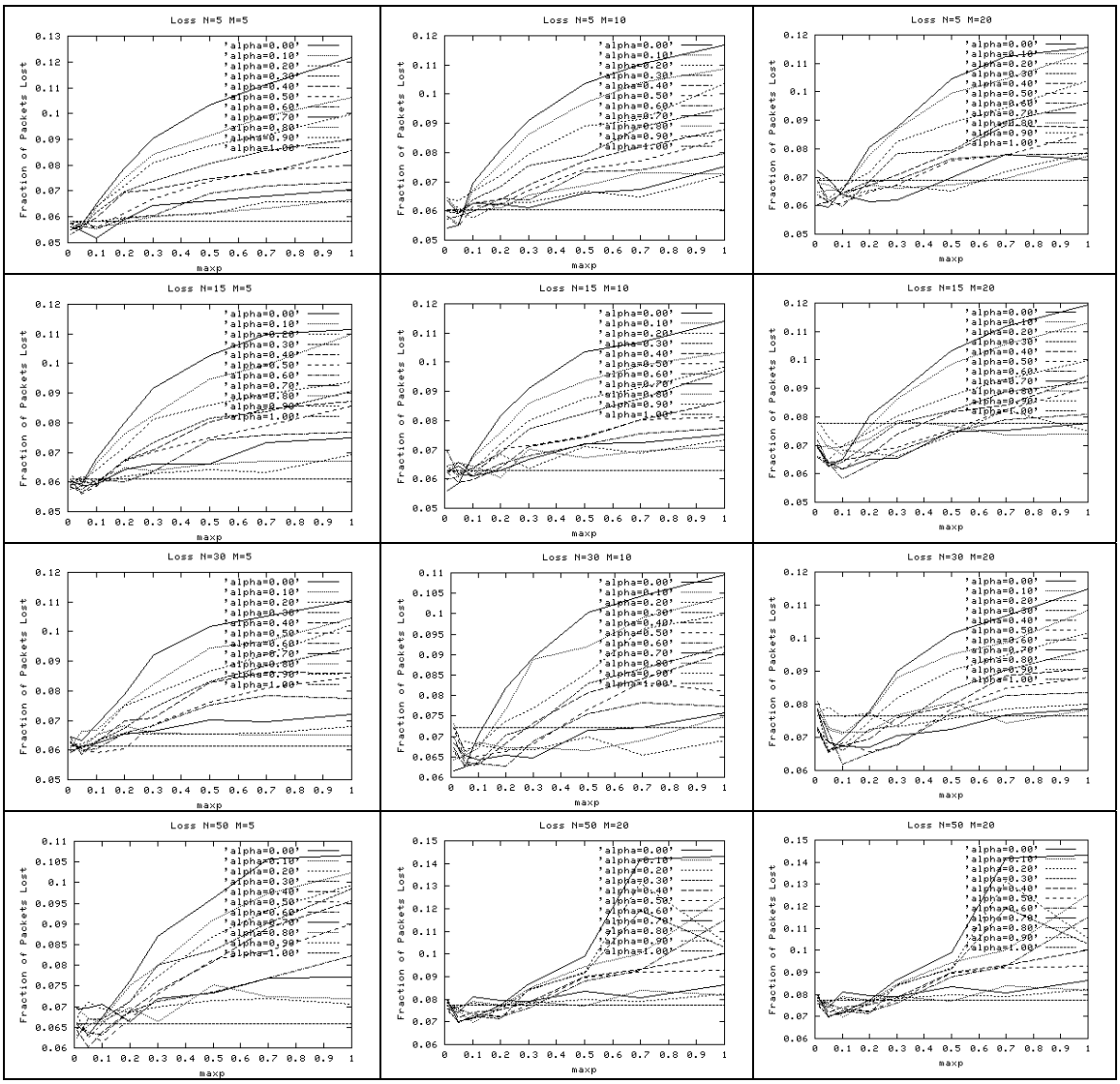


Figure 3.1: Fraction of packets lost vs. max_p for different α and (N_0, M)

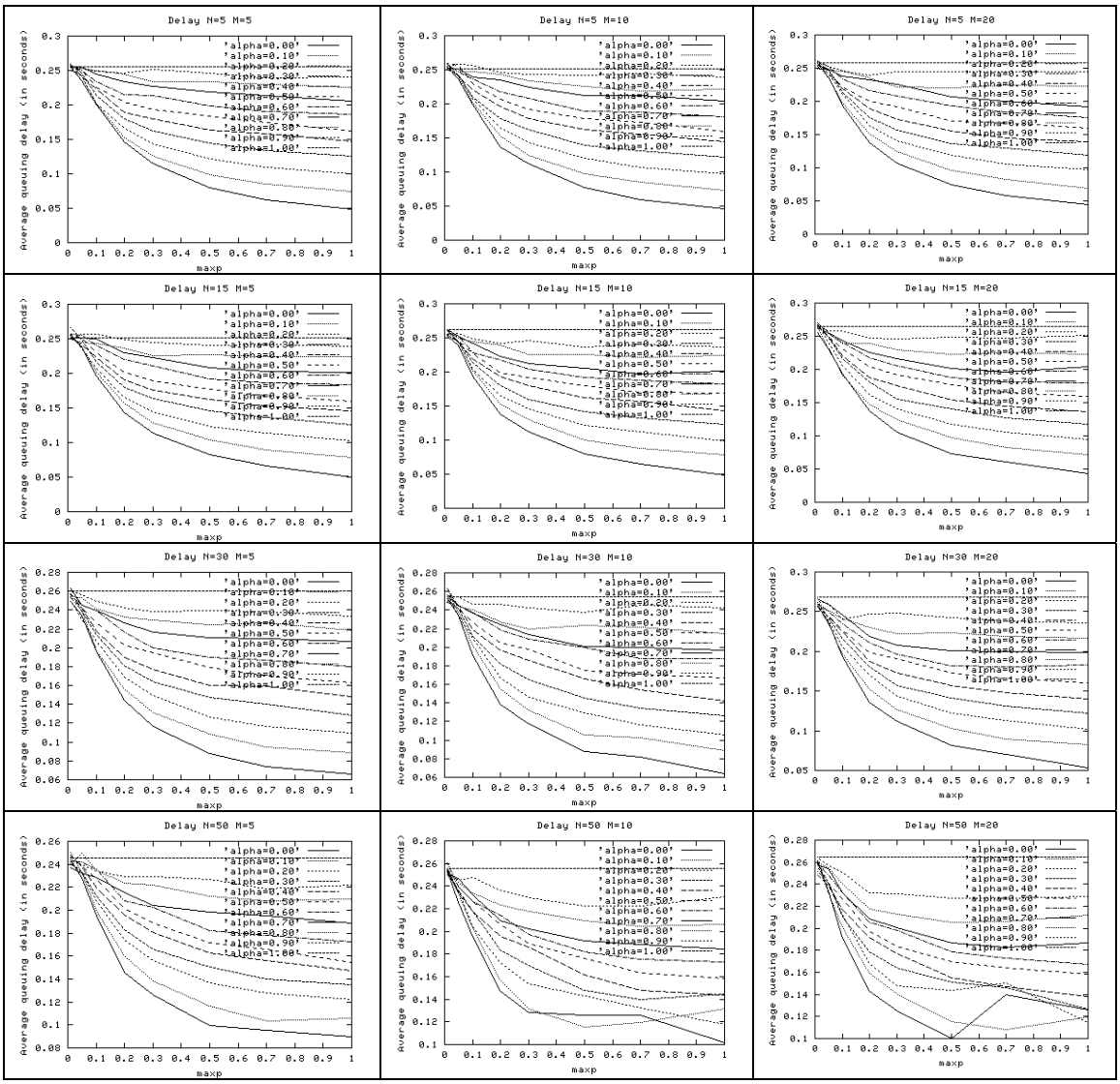


Figure 3.2: Packet delay vs. max_p for different α and (N_0, M)

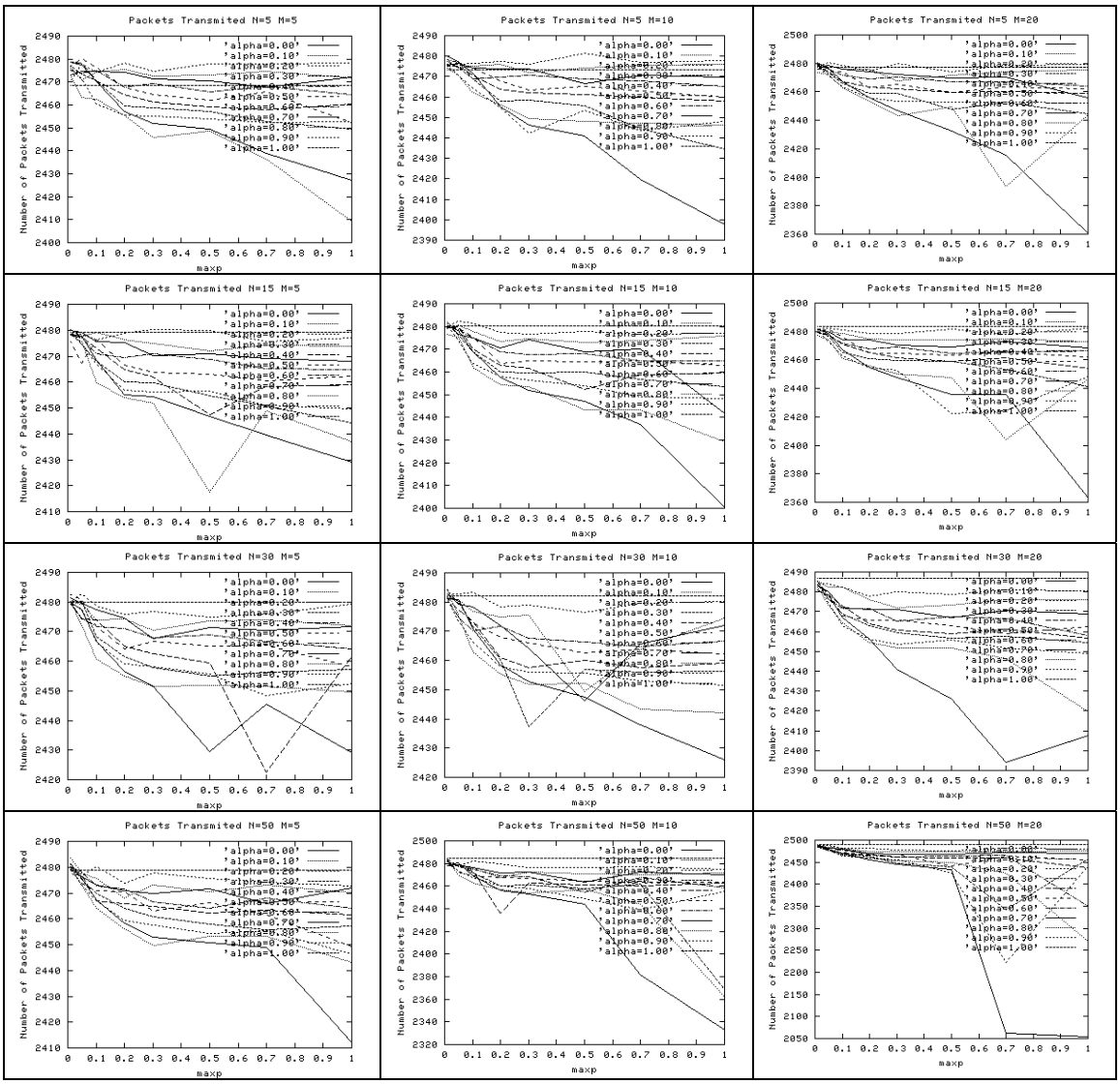


Figure 3.3: Link utilization vs. max_p for different α and (N_0, M)

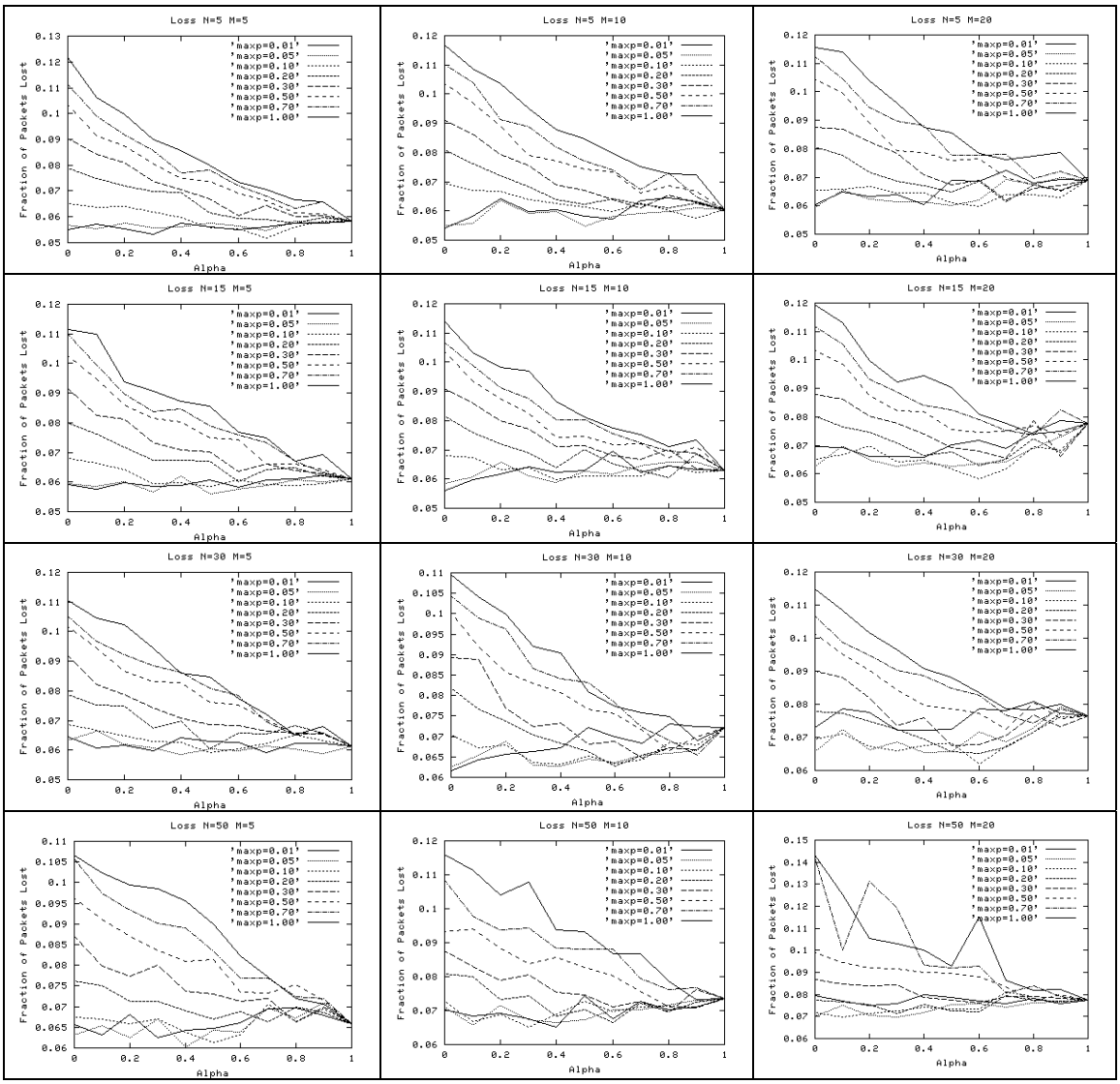


Figure 3.4: Fraction of packets lost vs. α for different max_p and (N_0, M)

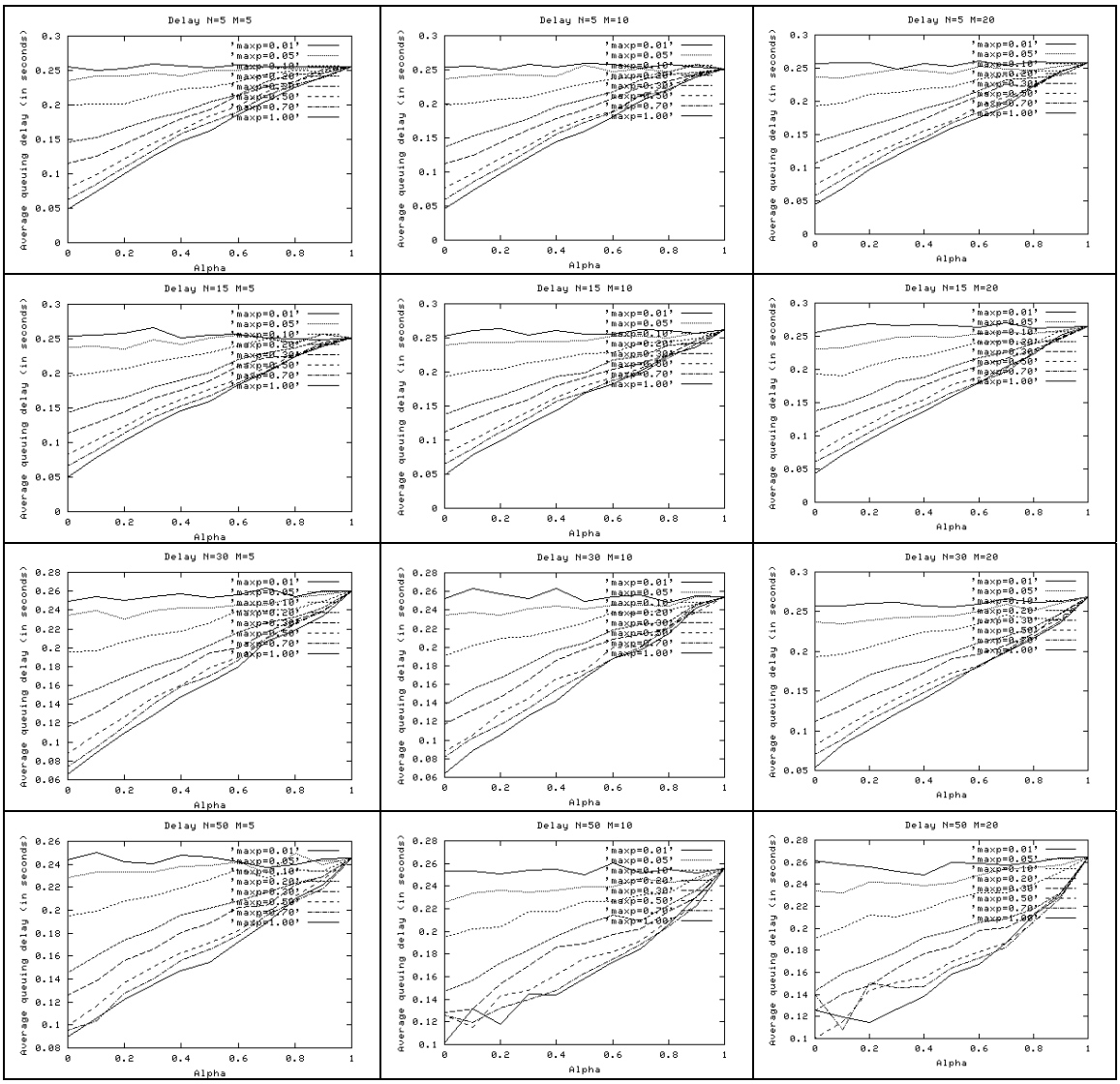


Figure 3.5: Packet delay vs. α for different max_p and (N_0, M)

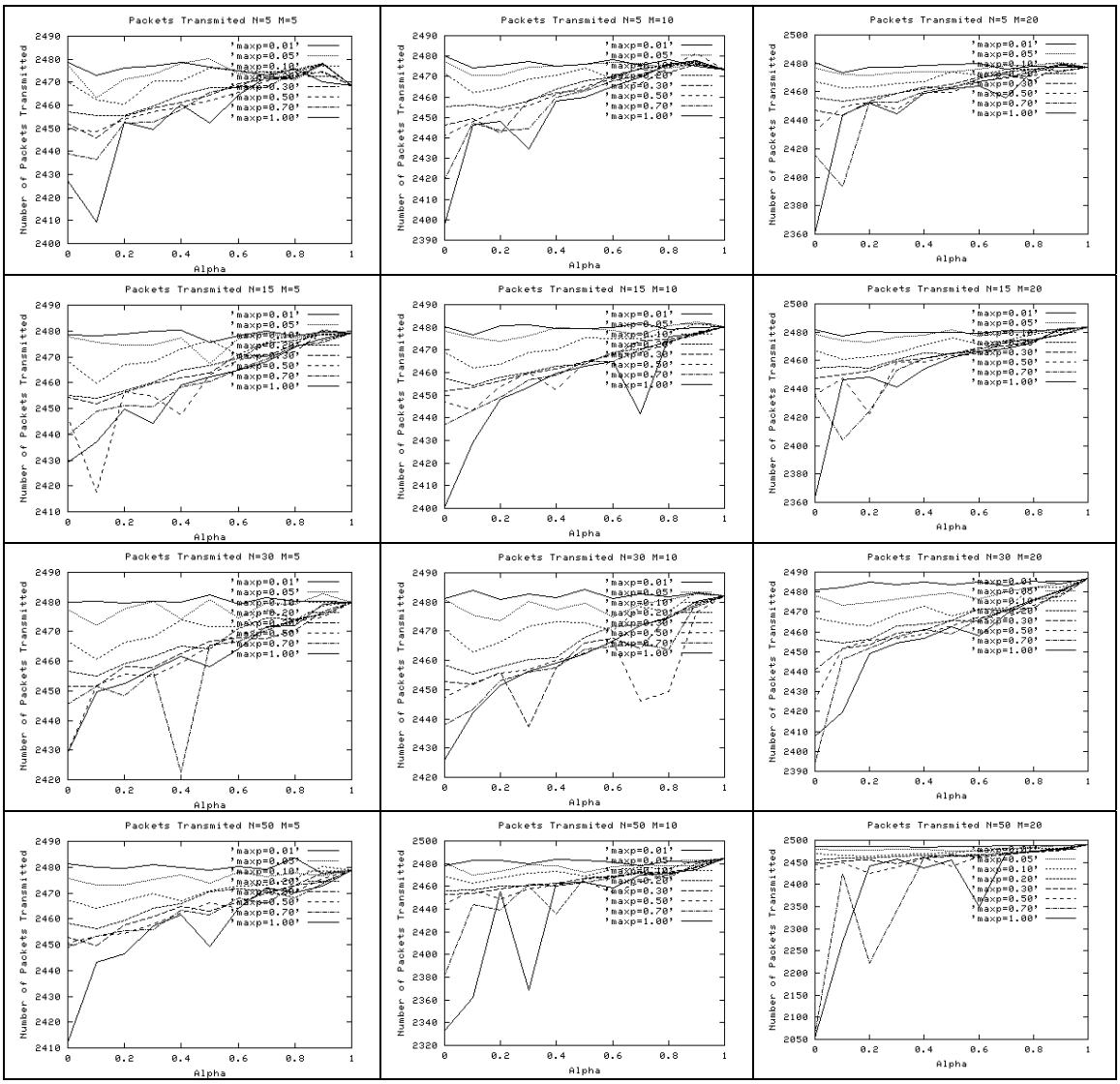


Figure 3.6: Link utilization vs. α for different max_P and (N_0, M)

and rate of convergence depends on M, which is the order of the NLMS filter. Refer to Section 2.6 for more elaborate definitions of the various parameters. N_0 takes the values $\{5, 15, 38, 50\}$, while M takes the values $\{5, 10, 20\}$ for each value of N_0. In each of the sub-figures, max_p is the free axis and each curve in the sub-figure corresponds to a different value of α. From the three figures, we observe that the value of N_0 and M do not affect any of the performance criterion, *viz,* link utilization, delay or packet loss rate significantly. We choose $N_0 = 15$ and $M = 10$ as our default values as they give us slightly better performance over other values.

Similarly Figures 3.4, 3.5 and 3.6 illustrate the probability of packet loss, packet delay and link utilization with α as the independent variable. Every curve in each of the sub-figures corresponds to a different value of max_p. From these three figures, we again take note that the variation in the performance is insignificant with change in either N_0 or M.

3.2 Effects of max_p

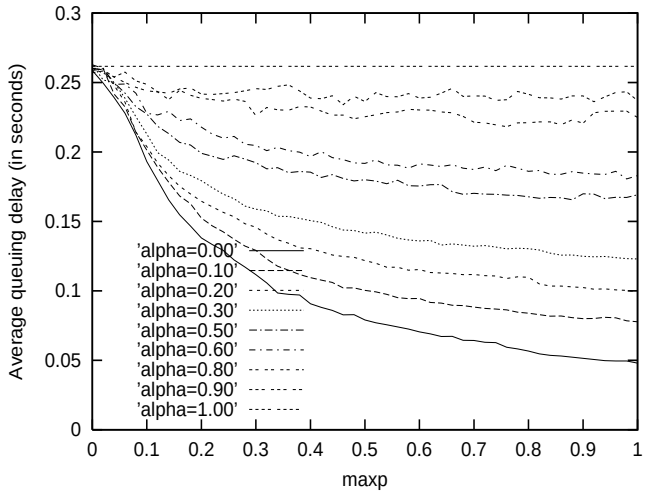


Figure 3.7: Average queuing delay vs. max_p (different α), harsh scenario

The parameter max_p governs how aggressively the packets are dropped, based on the predicted value of the instantaneous queue length. Figure 3.7 illustrates the variation of the average queuing delay as a function of max_p in a harsh scenario (25 TCP sources and $5 - 10\%$ packet loss rate). The different plots, as indicated, correspond to different values of α. The plots show that the average queuing delay

is a non-increasing function of max_p. Moreover, for relatively high values of α ($\geq$ 0.5), the delay stabilizes above a certain value of max_p ($\approx$ 0.2). For low values of α, the delay keeps decreasing even for high values of max_p. In a mild scenario (5 TCP sources and $1-2\%$ packet loss rate) (see Figure 3.8), even for low values of α, the average queuing delay stabilizes beyond $max_p = 0.1$.

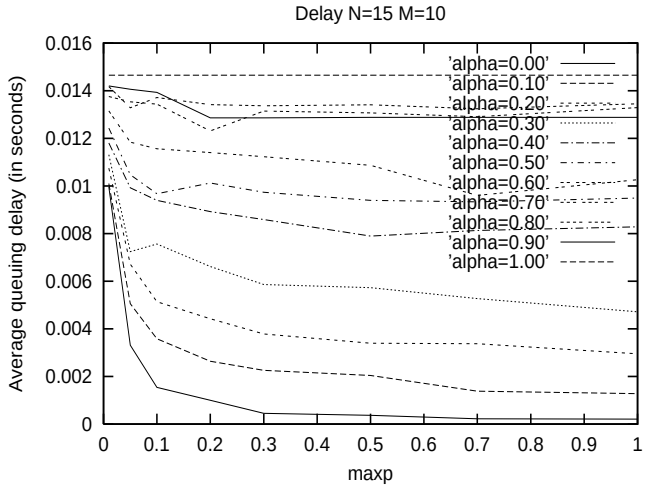


Figure 3.8: Average queuing delay vs. max_p (different α), mild scenario

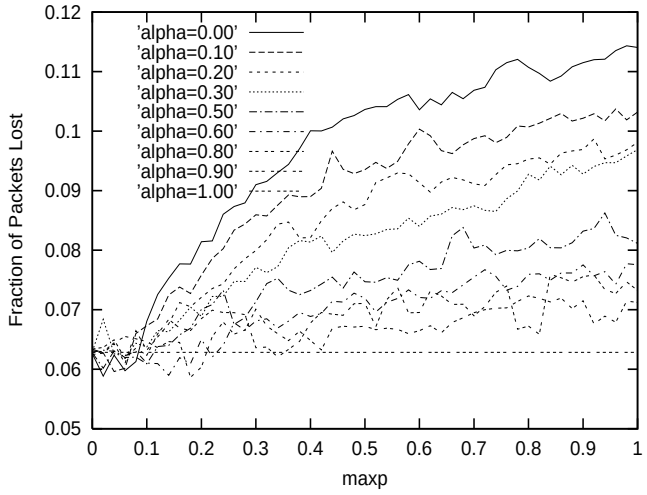


Figure 3.9: Packet loss rate vs. max_p (different α), harsh scenario

Figure 3.9 illustrates the variation of the packet loss rate against max_p in a harsh scenario. The plots show that the packet loss rate increases with increasing max_p. This is intuitive because a higher value of max_p results in a more aggressive dropping

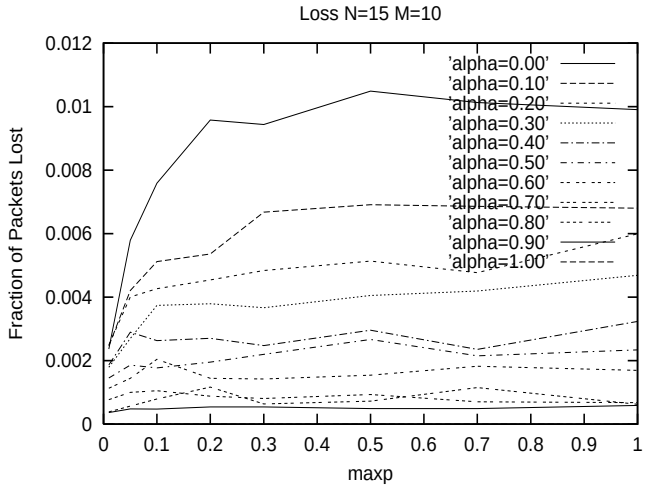


Figure 3.10: Packet loss rate vs. max_p (different α), mild scenario

of packets leading to a greater packet loss rate. Moreover, the increase is more prominent for low values of α. For large values of α (> 0.5), the packet loss rate stabilizes beyond $max_p \approx 0.2$. However, in a mild scenario (Figure 3.10) the packet loss rate is independent of max_p for a large range of α ($\alpha > 0.2$).

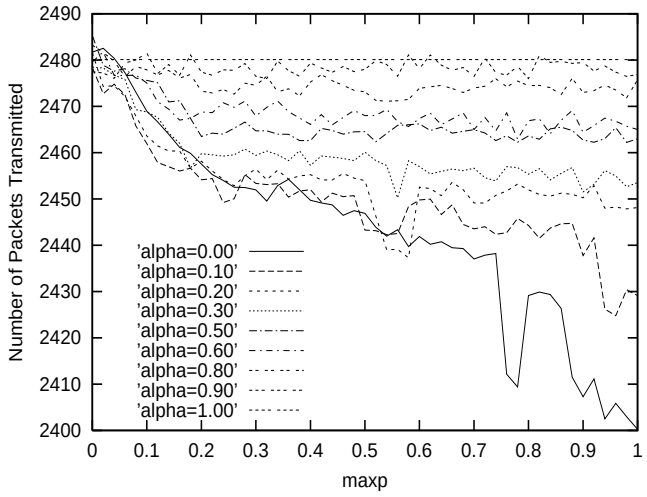


Figure 3.11: Number of packets transmitted vs. max_p (different α), harsh scenario

Figure 3.11 depicts the number of packets transmitted as a function of max_p in a harsh scenario. From the plots, it is evident that typically the number of packets transmitted decreases as max_p is increased. Though for high values of α, the number of packets transmitted is more or less independent of max_p. We observe a similar trend for the mild scenario too (see Figure 3.12).

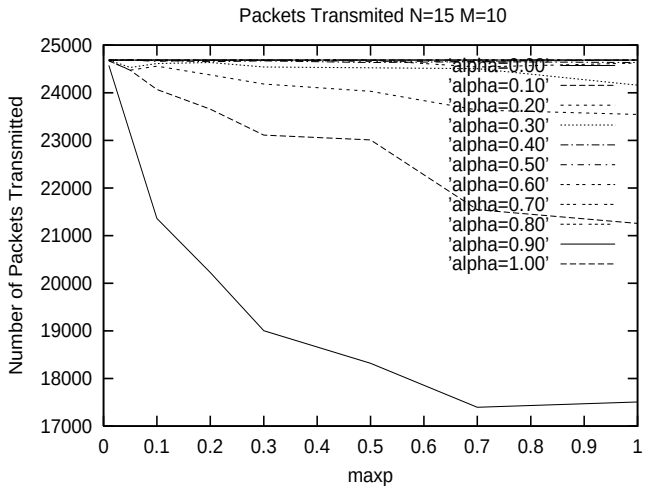


Figure 3.12: Number of packets transmitted vs. max_p (different α), mild scenario

3.3 Effects of α

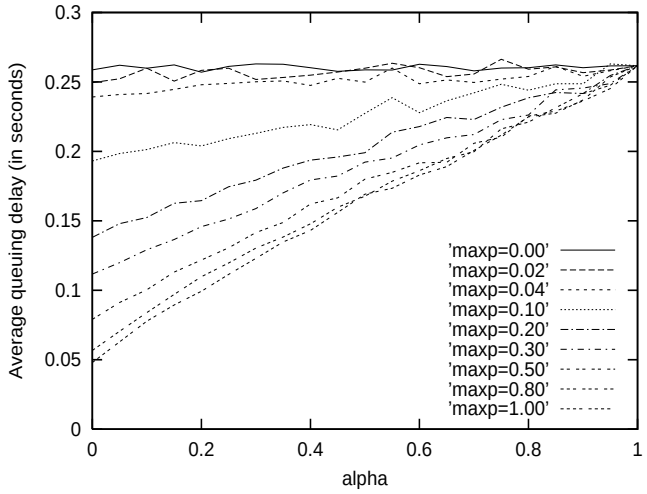


Figure 3.13: Average queuing delay vs. α (different max_p), harsh scenario

In this section, we explain the effect of α on various performance metrics. Figure 3.13 illustrates the variation of average queuing delay with α. The different plots, as indicated, correspond to different values of max_p. The plots correspond to a harsh scenario. As can be seen from the figure, in most cases the average queuing delay increases with increasing α. This effect is more prominent for high values of max_p (≥ 0.2). For low values of max_p, the average queuing delay is independent of α. In a mild scenario (Figure 3.14), the plots corresponding to different max_p are close to

each other (especially for $max_p > 0.01$). Though here too, the average queuing delay increases with increasing α.

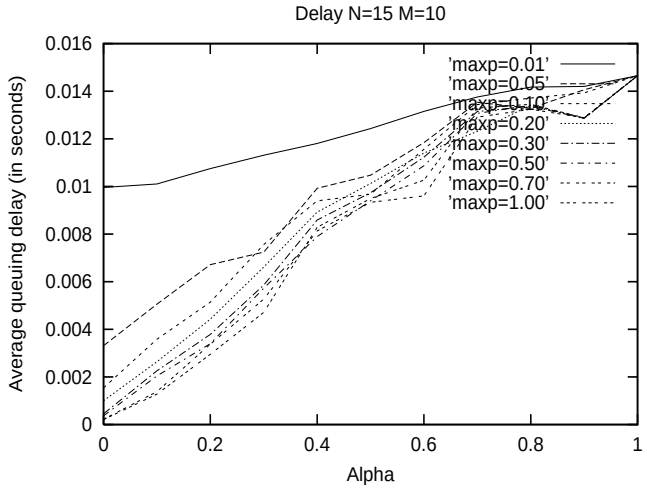


Figure 3.14: Average queuing delay vs. α (different max_p), mild scenario

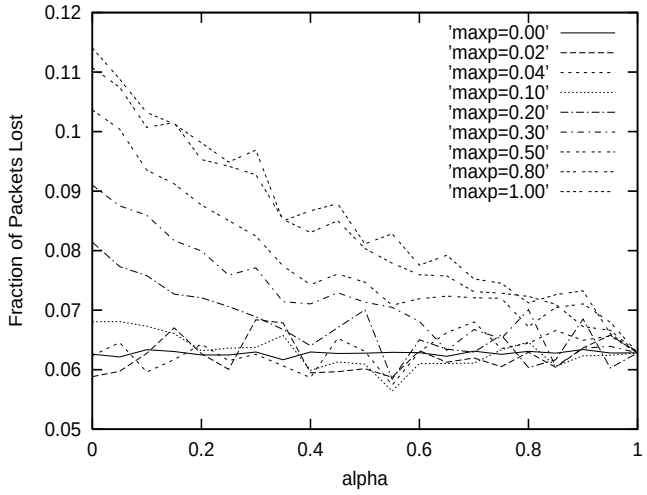


Figure 3.15: Packet loss rate vs. α (different max_p), harsh scenario

Figure 3.15 shows the packet loss rate as a function of α (in a harsh scenario). For high max_p (≥ 0.2), the packet loss rate reduces with increasing α. However, this trend is prominent for high values of max_p only. For low values of max_p, the packet loss rate is independent of α. Also as α approaches 1, the packet loss rates for different values of max_p start converging. From the plots for mild scenario (Figure 3.16), we can infer that the packet loss rate is independent of max_p for high values

(> 0.1) of max_p. Therefore, we can say that in a mild scenario, max_p does not play a significant role in controlling either the packet loss rate or the average queuing delay. The above observation is more precise for values of max_p greater than 0.1.

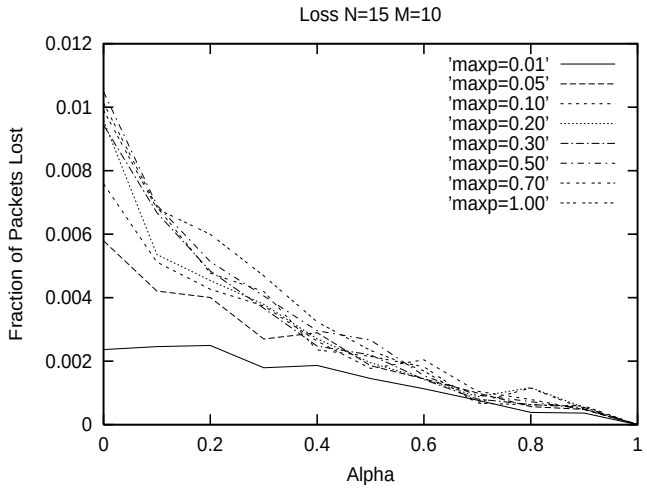


Figure 3.16: Packet loss rate vs. α (different max_p), mild scenario

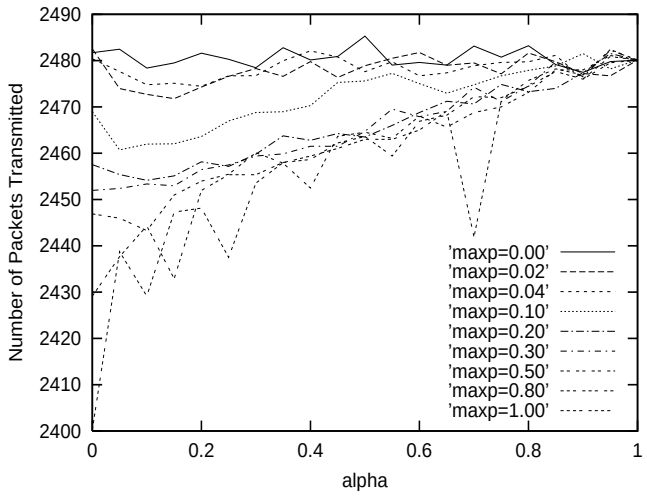


Figure 3.17: Number of packets transmitted vs. α (different max_p), harsh scenario

In Figure 3.17, the number of packets transmitted (in a harsh scenario) is plotted against α. From the plots we can infer that unless max_p is very high (≥ 0.7), the number of packets transmitted is independent of α. For high values of max_p, the number of packets transmitted is lower for low values of α. Also for high values of α, the number of packets transmitted is independent of max_p. In a mild scenario, the

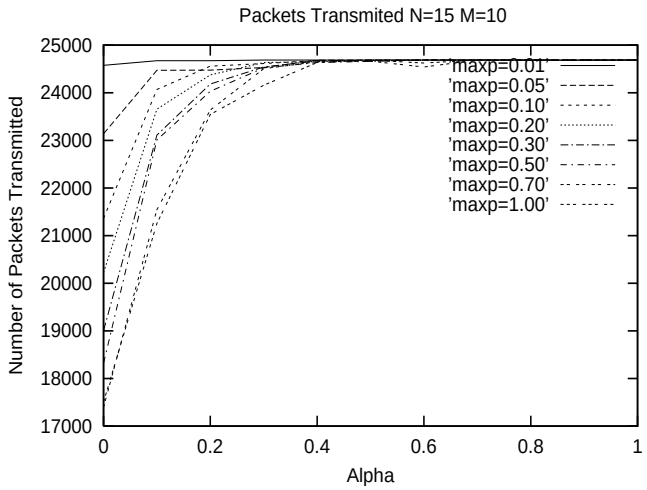


Figure 3.18: Number of packets transmitted vs. α (different max_p), mild scenario

number of packets transmitted stabilizes beyond $\alpha = 0.3$. For $\alpha < 0.3$, the number of packets transmitted is less for lower values of α. Also the curves corresponding to different max_p are close to each other which further validates the observation that max_p does not have a significant role to play in mild scenarios.

3.4 Conclusions

The above detailed study of the parameters of APACE indicates that the behaviour of APACE for its various parameter settings is limited and predictable. We have verified out claim that the performance is independent of the choice of N_0 and M. This is mainly due to the inherent adaptive nature of the algorithm used for prediction of queue length. These conclusions, along with our inferences regarding parameter settings of APACE from Chapter 2 make APACE a very convenient AQM scheme with only one parameter max_p to be varied to achieve any given performance (operating point).

CHAPTER

4

Learning TCP (LTCP)

Those who don't learn from history, are doomed to repeat it

In this chapter, we explain the following:

- *The Learning TCP (LTCP) algorithm*
- *Mathematical analysis of LTCP*
- *Performance evaluation with respect to*
  - *Throughput Enhancement*
  - *Faimess*
  - *Friendliness*

Chapter 4

Learning TCP (LTCP)

4.1 Motivation for LTCP

The TCP estimates the available bandwidth on a link through its window increase-decrease policy and adjusts its window control behavior accordingly. But this method (and all it's subsequent enhancements) are not the best, for learning in today's perspective as losses do not solely occur due to network bandwidth limitations.

TCP's flow and congestion control mechanisms are based upon the assumption that packet loss is an indication of congestion. This rigidity in always assuming the cause of packet loss as congestion and subsequent significant reduction in window is one of the primary factors in degradation of TCP's performance over the networks where the assumption fails to hold. These includes networks such as wireless LAN, wireless WAN, satellite links, etc, where to make up for the random losses owing to errors in the wireless channel, a simple retransmission of the packet(s) is sufficient and there is no need for the source to drop its window size. Thus, there is a need for the TCP, which would differentiate and try to learn the reasons for packet loss and act accordingly, which would attempt to learn the network rather than assume, which would be universal and work for all types of networks, *ie,* wired networks, wireless networks, DiffServ networks, etc, which is exactly what LTCP attempts to accomplish.

4.2 Introduction to LTCP

LTCP attempts to learn the cause of packet loss and takes action appropriately. It is not specific to any particular network characteristics (*ie,* wireless or wired) and hence it can be applied in a wide variety of heterogeneous networks including fully wired networks.

We first characterize packet losses into two types based on the desired action that should be taken by the sender upon their occurrence. Type A losses require the sender to retransmit the lost packets and also reduce its transmission rate. Congestion losses are type A losses. Type B losses require the sender to retransmit the lost packets only and there is no need for window reduction. These are primarily the random losses occurring in the network owing to fading in the wireless channel, shadowing, etc. An ideal approach should correctly determine the type of each packet loss and take appropriate actions subsequently.

LTCP attempts to learn the type of packet loss using prediction based on probability p_b, which is the estimated conditional probability that the packet loss is of type B, given a packet loss over the link where TCP is operating. LTCP updates p_b based on accuracy of predictions, thus being adaptive to the changes in link characteristics. We have assumed that if the transmission window size is not reduced in case of packet loss due to congestion (this will happen if we predict incorrectly a congestion loss as type B loss), then it is highly probable that there will be atleast one more packet loss in the next 'n', where $n > 1$ round trip times (RTTs).

Earlier research work have proposed many approaches on improving TCP over such networks, but have focused on a particular network characteristics, *ie,* for wireless channel, mobility of host, etc. These schemes are not universal in the sense that they assume a particular kind of network and therefore cannot be used in heterogeneous networks that can be fully wired, or completely wireless or a combination of both.

The algorithm is explained in detail in Section 4.4. We have performed the mathematical analysis of throughput enhancement by LTCP and also derived an upper bound on throughput enhancement that can be achieved in Section 4.5. We have implemented LTCP in network simulator ns-2 [19] and have performed simulations for different loss scenarios and reported the results in Section 4.6 . We also show that LTCP is fair and TCP friendly in the same section.

4.3 Related Work in TCP

Most modern TCP implementations [21] incorporate algorithms introduced by Van Jacobson [1] into 4.3 BSD to fix the original 1988 congestion collapse. Wireless Media are more prone to transmission losses due to fading, shadowing, etc. TCP was designed with reference to a wired medium, which has much lower bit error rates than the wireless medium. Random losses or type B losses according to our terminology have been the most widely addressed issue in literature, and many solutions aim at alleviating this deficiency of TCP. As mentioned earlier, retransmission of lost packets in not an issue at all, it is that the lost packet triggers the congestion avoidance mechanism, which essentially leads to unnecessary substantial reduction in the sender window. A transient error, thus leads to TCP back off and it is not able to sustain a good throughput level [23]. There have been several improvements to TCP. TCP SACK was primarily proposed to alleviate the inefficiency in handling multiple drops in a single window of data [23, 24]. TCP Forward Acknowledgment (TCP FACK) attempts to decouple the TCP congestion control algorithm from data recovery [25]. However, it has never really been tested for wired-cum-wireless environments, and is more or less targeted to improving TCP's performance when losses are of type A rather of type B [26]. Proposals like Delayed ACKs [27, 28] and ACK pacing [29, 30] mainly aim at dealing with congestion-related losses and, in view of [31], it may not be suitable for wired-cum-wireless environment. TCP Westwood [32] attempts to estimate the available bandwidth in the network, however it does not attempt to learn the reason behind the packet loss. It simply reduces the window to maximum estimated bandwidth on an packet loss. Proposals like Explicit Congestion Notification (ECN) [33, 34] and Explicit Loss Notification (ELN) [35] are meant for wired-cum-wireless domain, but calls for more functionality in the routers. Most of the schemes focus on a particular problem and do not address the need for a general approach. LTCP on the other hand is an universal algorithm which can be used for both, wired as well as wireless networks without requiring any modifications at any of the routers in the network.

4.4 Learning TCP

In this section, we describe the LTCP algorithm in detail. For every packet loss detected by LTCP, it predicts with probability, p_l (learning probability), that the packet loss is of type B and not due to congestion. If it predicts the loss to be of type B, then it simply retransmits the lost packet and the appropriate subsequent packets without any reduction in the congestion window size. If it predicts the loss to be of type A, then normal TCP action is taken. As LTCP is finding out the cause of packet loss probabilistically, it is likely to make errors in prediction. In case, the actual packet loss is due to congestion and LTCP predicts it as type B (not congestion), then it is highly probable that LTCP will incur again at least one packet loss in the next n RTTs ($n > 1$) as it continues to transmit without reducing the congestion window. This assumption is more appropriate for large window sizes. Hence we define p_{inc} as the ratio of window at the time of packet loss detection and maximum window achieved on the link. LTCP uses these information for adjusting the value of p_b adaptively and thus learns about network characteristics. Also we need to take some conservative action (we reduce the window size to one) to compensate for the error made in prediction (type A loss predicted as type B). In case of type B loss being predicted as type A (congestion), LTCP will update the value of p_l appropriately and behave as normal TCP and will not perform worse. The detailed algorithm with pseudo code is illustrated in Figure 4.1.

4.5 Mathematical Analysis of LTCP

4.5.1 Ideal LTCP

We calculate maximum performance enhancement by assuming that LTCP behaves ideally *ie,* the prediction is always correct.

Let,

p = Packet loss probability.

p_a = Probability of type A loss given a packet loss has occurred.

p_b = Probability of type B loss given a packet loss has occurred.

k = Number of packets acknowledged per TCP ack.

Th_{tcp} = Throughput of TCP.

Initialization

- initialize p_l to some constant value
- $p_l = constant$; /*variable used for predicting p_b*/
- Check_Prediction = 0;

Learning p_b()

- a packet loss is detected;
- **if** Check_Prediction == 0 **then**
  - predict packet loss type using prediction probability p_l;
  - **if** (Packet loss cause = Type A) **then**
    - take TCP congestion control action;
  - **else**
    - retransmit lost packet and successive packets with no reduction in sender window
    - set Check_Prediction = 1;
    - Set Check_Prediction_Timer = n*RTT; /*n=1.5 in our implementation*/
  - **end if**
- **else**
  - Check_Prediction == 1; /*Previous prediction that packet loss is of type B is incorrect */
  - decrease p_l;
  - cancel Check_Prediction_Timer;
  - set Check_Prediction = 0;
  - take a conservative action; /*We make $congestionwindow = 1$*/
- **end if**

CheckingPredictionTimeOut()

- previous calculation: packet loss is of type B is correct
- increase p_l with probability p_{inc} ;
- set Check_Prediction = 0;

Figure 4.1: Window Control and learning p_b

Th_{ltcp} = Throughput of LTCP.
E = Enhancement = Th_{ltcp} / Th_{tcp}.

Calculating the throughput of ideal LTCP similar to [36] we get:

$$Th_{ltcp} = \frac{\frac{1-p_b p}{p_b p} + \frac{2+k}{3k} + \sqrt{\frac{8(1-p_b p)}{3kp_b p} + \left(\frac{2+k}{3k}\right)^2}}{RTT\left(\frac{2+k}{6} + \sqrt{\frac{2k(1-p_b p)}{3p_b p} + \left(\frac{2+k}{6}\right)^2} + 1\right)} \tag{4.1}$$

which can be approximated as:

$$Th_{ltcp} = \frac{1}{RTT}\sqrt{\frac{3}{2kp_b p}} \tag{4.2}$$

Similarly we get the approximate expression for TCP throughput as:

$$Th_{tcp} = \frac{1}{RTT}\sqrt{\frac{3}{2k(p_a + p_b)p}} \tag{4.3}$$

From (4.1), (4.2) and (4.3)

$$E = \sqrt{1 + \frac{p_b}{p_a}} \tag{4.4}$$

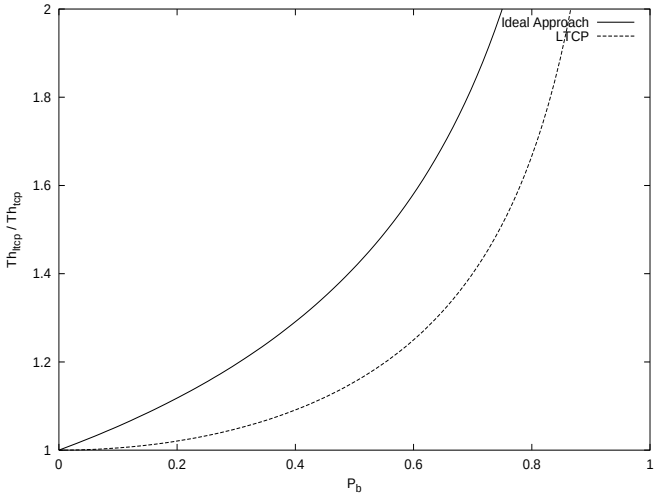


Figure 4.2: Enhancement by Ideal LTCP with Pb

In case of only wired networks ($p_b = 0$), and thus $E = 1$, *ie,* LTCP achieves same throughput as TCP and thus is TCP friendly. E will never be less than 1 and will always be > 1 in the presence of type B losses. Figure 4.2 gives the plot

of enhancement, E, against p_b. The curve for the ideal approach illustrates the maximum possible gain, given the source knows exactly which of the packet loses are of type A and which are of type B respectively. This curve corresponds to the maximum enhancement that can be achieved. LTCP performs slightly inferior as losses in consecutive windows (*ie,* within n RTT's) are assumed to be type A (owing to congestion) losses.

4.5.2 Practical LTCP

In real life scenarios LTCP will not be able to predict the type of loss accurately for all losses. The measure of how well LTCP performs depends on how close to p_b can it adapt p_l and how accurate is the prediction. To analytically compare the performance of a practical LTCP with TCP, we need to find the probability of window drop.

$$\begin{aligned} & Prob[Decrease\ in\ window \mid loss] \\ = & Prob[type\ A\ loss\ predicted\ as\ type\ A] \\ & + Prob[type\ A\ loss\ predicted\ as\ type\ B] \\ & + prob[type\ B\ loss\ predicted\ as\ type\ A] \end{aligned} \tag{4.5}$$

We have assumed that whenever type A loss is predicted as type B, LTCP will detect the error in prediction in the next 'n' (we have kept $n = 1.5$) RTTs. It will take a conservative action and reduce its window. Hence we have included the corresponding term (second term on the right hand side of Equation 4.5) in our calculation. This implies that type A losses will always lead to reduction in window size as expected and how well we predict type B losses will determine the performance gain we achieve. Thus:

$$\begin{aligned} Prob[Decreaseinwindow|loss] &= p_a(1-p_l) \\ &+ p_a p_l + p_b(1-p_l) \\ &= p_a + p_b(1-p_l) \\ &= 1 - p_b p_l \end{aligned} \tag{4.6}$$

Thus we have an enhancement in throughput equal to:

$$E = \frac{1}{\sqrt{1-p_b p_l}} \tag{4.7}$$

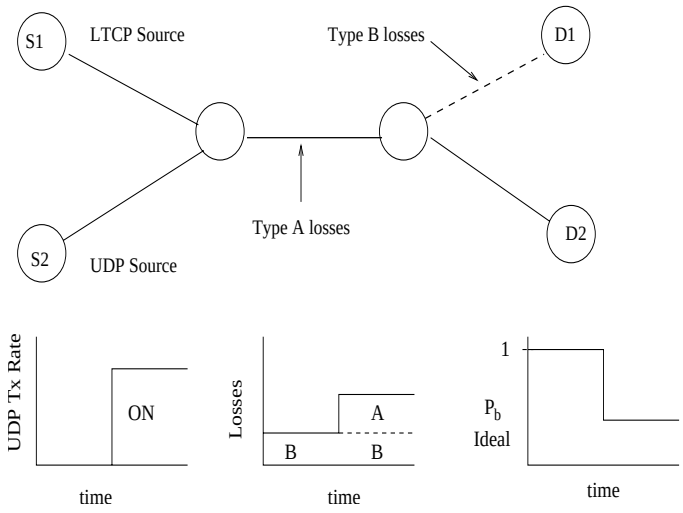


Figure 4.3: Simulation Topology

We observe that keeping $p_l = 1$, *i.e.,* predicting all packet losses as type B loss may maximize throughput, but this will also make us TCP unfriendly which is not desirable. Another point worth mentioning is that the conservative action we take after we detect that a type A error was predicted as type B may rather impair the throughput (we are dropping the window to 1 in our implementation to compensate for the extra window of packets sent).

4.6 Simulation Study

4.6.1 Simulation Setup

The simulations have been performed using the network simulator, **ns v2.1b8a** [19]. A representative network topology as shown in Figure 4.3 was used to test LTCP in various scenarios. All sources start randomly and packet size has been fixed to 1000 bytes.

Congestion that leads to type A losses are induced in the network using a Constant Bit Rate (CBR) source with high sending rate. This congestion occurs at the bottleneck link to which all the TCP/LTCP sources are transmitting. The sending rate of CBR source is metric to the congestion in the bottleneck link. It's sending rate is to be less than the bandwidth of the bottleneck link, but high enough to cause congestion at the bottleneck link.

Type B losses which are random in nature with bursts are induced using a *Two*

State Markov Loss Model which alternates between a good and a bad state, dropping packets with a certain probability when in the bad state. By varying the good state duration, bad state duration and the loss probability of bad state we can achieve any type B loss probability. The bad state duration is kept less than the average RTT of all the sources ,so that the random losses are confined only to a single window.

We use the *File Transfer Protocol or FTP* based applications over TCP or LTCP. These applications have sufficient data to be sent ,in order to ensure that the connection never starves for data. The values chosen for the *Round Trip Time (RTT)* are approximately 60 ms and 600 ms. The 60 ms value represents a Wireless Local Area Network (WLAN) environment where RTT values are in the order of milliseconds. The large values of 600 ms models a Wireless Wide Area Network (WWAN) environment.

The buffer size at bottleneck link is kept as 150 packets to avoid any influence in throughput due to the buffering and scheduling mechanisms. The buffering mechanism is*Random Early Detection* and the Scheduling mechanism is *First in-First out* as these are dominantly used in the Internet. Link capacity for the small round trip time (RTT) of 60ms was taken to be 10Mbps to represent a WLAN link, while for a large RTT of 600ms it was taken as 100Kbps, to represent a WWAN link.

For WLAN environment, simulation is carried out for 200 seconds and for WWAN environment for is 1000 seconds.

4.6.2 Throughput Enhancement

The CBR source is switched off permanently and bandwidth of bottleneck link and queue length is increased sufficiently to avoid type A losses altogether. The Markov loss model simulating type B loss is initiated. In a network with only type B losses, the throughput of TCP degrades drastically as the probability of type B loss increases, while LTCP flows are able to achieve higher throughput than the TCP flows of same RTT as shown in Figure 4.4.

4.6.3 Fairness and Friendliness

Figure 4.5 shows that a LTCP source is friendly with other TCP sources. The plot has one LTCP source and five TCP sources. The LTCP source achieves the same throughput as all other TCP sources.

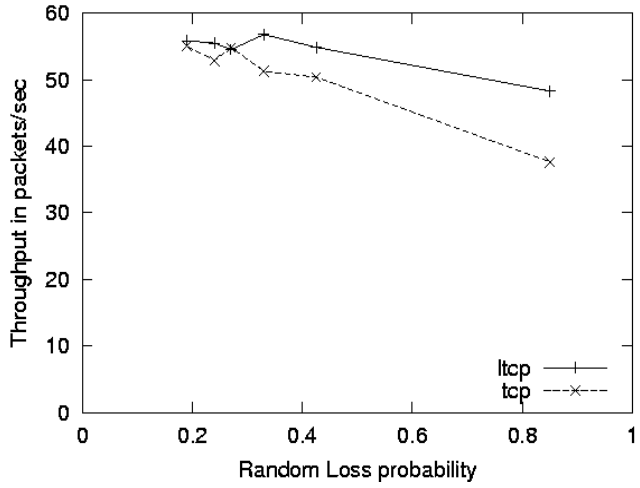


Figure 4.4: Throughput enhancement of LTCP over TCP as type B losses increase

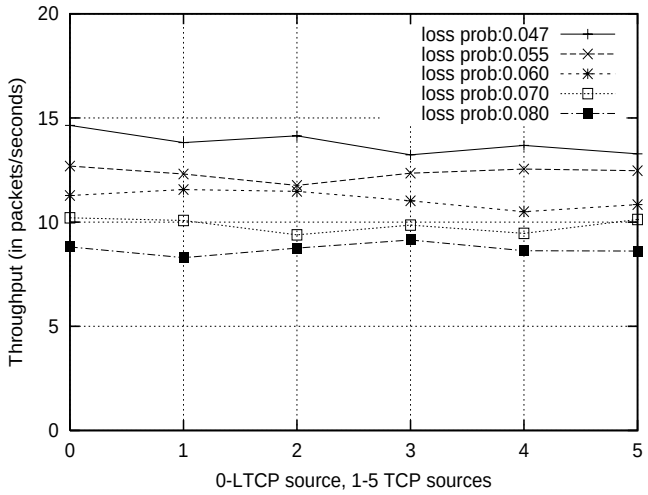


Figure 4.5: Throughput of a LTCP and five TCP sources, all having the same RTT (600ms) and sharing a common bottleneck link with receiver advertised window of 15 packets

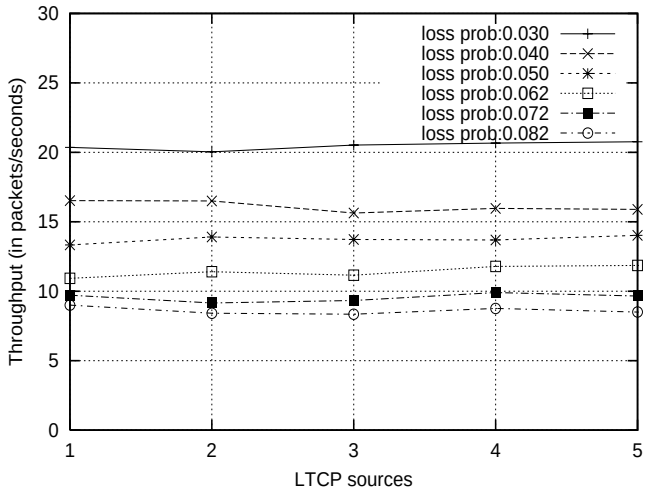


Figure 4.6: Throughput of five LTCP sources having the same RTT (600ms) and sharing a common bottleneck link with receiver advertised window of 15 packets

In addition to being friendly to other TCP sources, the LTCP source is also be fair to other LTCP sources. Figure 4.6 shows five LTCP sources with same RTT sharing the same bottleneck link. Both the types (A and B) of losses occur in the network. We observe that LTCP source with same RTT's achieve the same throughput. Similar behavior is observed for lower RTTs also. Hence we conclude LTCP adheres to the behavior of TCP in terms of fair share of bandwidth among peers.

4.7 Conclusions

In this chapter, we have proposed a modification to the existing TCP algorithm to improve end-to-end performance in heterogeneous networks, which may be completely wired or entirely wireless or a combination of both. LTCP attempts to learn the reason for packet loss in the network and then acts accordingly. Such learning can also be applied to incorporate various other networks like DiffServ networks, etc.

4.8 Future Directions - some thoughts on improving the performance of TCP

In this section, we throw light on some of our thoughts regarding improving TCP's performance in general and over Differentiated Services networks in particular. We

plan to achieve the above by modifying suitably the algorithms currently used in any of the following ways:

1. The Explicit Congestion Notification (ECN) [33, 34] scheme can be extended by reserving another bit for it. The additional bit may be used to indicate any of the following:

   (a) *IN/OUT bit:* The additional bit can be used to indicate whether the packet dropped/marked was an IN/OUT packet. The TCP at the hosts can then reduce their window size appropriately. They should not make drastic reduction in the window if the packet marked was an OUT packet.

   (b) *Retransmitted packets:* A loss of retransmitted (*i.e.,* packets that have been lost once before) packets severely deteriorates the throughput of the TCP source and may lead to severe under-utilization of the link. This is one of the major problems that TCP is currently facing since the TCP window is reduced to one in such a scenario. A mechanism by which a source would mark a retransmitted packet and an appropriate AQM policy at the router which will give preference to such packets will definitely help improve the overall link utilization.

   (c) *Slow Start, etc:* If the source sends an indication when its in slow start, or other conditions in which a unnecessary packet loss may prove very costly, the AQM policy can take care of it and give such packets a higher preference. It will also help us achieve better fairness for sources with larger RTT's.

2. The ingress routers currently use the token bucket scheme for marking TCP traffic. We argue that since TCP reduces its window size upon a packet loss, the packet shaper at the ingress router should take this fact into account before marking the packet as OUT. The packet shaper should be aware of the feedback action which will result in case of packet drop if it marks it as OUT. The ingress router is also the most appropriate place to make changes from implementation point of view. The later part of the report is exclusively devoted to improving the performance of the token bucket at the ingress routers.

CHAPTER

5

Bandwidth Allocation & Rate Control

The imperial authority symbolized by a ***Scepter***

In this chapter, we address the following issues:

- *Bandwidth allocation techniques & applications*
- *Classic token bucket mechanism*
- *Issues in allocating bandwidth to TCP aggregates*
- *Reasons for failure of the classic token bucket mechanism*
- ***Scepter**, an alternate algorithm for precise bandwidth allocation for TCP aggregates*

Chapter 5

Bandwidth Allocation and Rate Control

Managing Bandwidth still remains a largely unsolved issue. Lack of QoS guarantees impairs human interaction in applications such as tele-conferencing, Internet phones, video and audio streaming, etc.

The service provider would want ideally to give delay and jitter[1] guarantees to the subscriber, simultaneously maintaining high utilization of its link resources. However these type of delay and jitter sensitive flows form a very small fraction of Internet traffic. As of today, most of the traffic in the Internet is carried by TCP and this is where bandwidth allocation is most relevant. Consider an example of an ISP that wishes to provide different customized bandwidth to its various customers. Some may want 2Mbps, some other may require 1.75Mbps. Given that most of its customers will have largely TCP traffic, the real challenge in front of the ISP is to allocate and guarantee precisely the required bandwidth with minimum over provisioning of resources. Giving guarantees and allocating bandwidth to TCP flows is a very difficult problem due to the very nature of the TCP protocol. Adding access bandwidth may not always be a good idea. The nature of TCP is to expand traffic flows until all of the bandwidth is consumed. Therefore an increase in bandwidth does not solve the problem of congestion, window drops, global synchronization and under utilization of the link. Moreover, an increase in bandwidth may benefit the larger sources and may not necessarily improve the performance of the smaller, less aggressive sources. The ISP, therefore needs a tool which would give it more freedom and precision to

[1] Jitter is the variation in the latency experienced by different packets in the same stream.

allocate bandwidth to both TCP as well as non TCP sources. It should, if necessary be able to allocate additional bandwidth to exactly one aggregate of the many.

5.1 Token Bucket: Classic Bandwidth Allocation Techniques and Applications

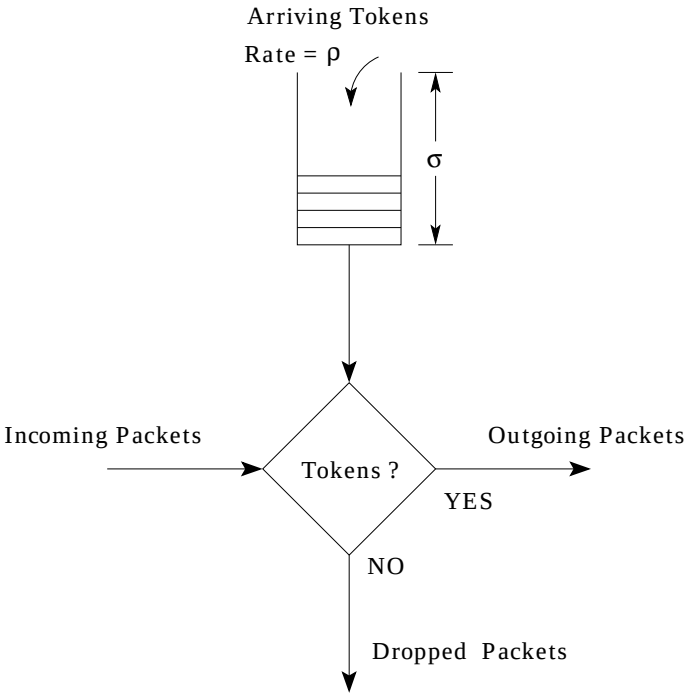


Figure 5.1: The token bucket mechanism

Token Bucket, also known as Leaky Bucket (refer to Figure 5.1) is currently the most widely used mechanism to allocate bandwidth. Though schedulers, classifiers and many others play a crucial role in bandwidth provisioning, we will restrict our focus only to token bucket in order to identify easily the effects of the proposed token bucket mechanism. A token bucket flow is defined by two parameters (ρ, σ), where ρ denotes the rate at which the tokens enter the bucket which also corresponds roughly to the long-term average rate allocated by the network and σ is the maximum bucket size which also corresponds to the longest burst a source can send. A packet which enters the system is allowed to go out only if it has tokens worth the number of bytes in the packet. When the bucket overflows, excess incoming tokens are discarded. If there are insufficient numbers of tokens in the bucket, then, one of the following may

happen:

- The packet is dropped.
- The packet is marked in a particular way.
- The packet may be kept in a buffer and transmitted later when sufficient number of tokens are available.

For the purpose of discussion, let us assume that those packets which have insufficient number of tokens are dropped. We observe that the long term transmission rate of the source is ρ which can sometimes deviate on a smaller time scale as dictated by σ. The number of bits transmitted in any interval of length t is less than $\rho t + \sigma$.

Given a source, say a video, we would like to come up with parameters ρ and σ, that best describe it such that no other descriptor has both a smaller ρ and σ.

5.2 Motivation for Scepter

A change at the edge router requires minimal effort and investment as compared to a change at the core routers or the transport protocol at the millions of individual computer hosts. The current token bucket mechanism used to shape the traffic though adequate for shaping non-responsive constant rate traffic fails to guarantee, even the committed bandwidth, to responsive TCP flows. This is primarily because, the current token bucket implementation allows a packet to pass through, if it conforms to the service level agreements and drops it, if it exceeds the agreed upon rate. In the presence of TCP aggregates, invariably the window sizes of individual sources keep increasing to a point till it exceeds the limit and then the token bucket at the marker starts dropping the incoming packets. The root cause of the problem is, that the TCP sources at the end points are unable to distinguish, whether a packet was dropped owing to congestion or due to exceeding the allocated bandwidth. In the later case, dropping the window size drastically is totally unwarranted for. So, when the bucket is empty and it starts dropping all incoming packets, most of the transmitting TCP sources encounter packet loss as was explained in Section 1.2. The drastic reduction in their individual window sizes and hence transmission rates of most of the individual TCP flows in the aggregate disregarding the cause for packet drop incapacitates the aggregate to send traffic even at the allocated rate. One solution to this could be

to let the source know the cause of packet loss, but this would require a change at all end sources (Computers, etc) and in addition to the edge routers. This option, though feasible, is not that practical. Instead we propose an alternate token bucket marking mechanism which significantly reduces the number of packets dropped in an aggregate by pre-emptively notifying the aggregates (not sources) in advance without requiring any change in the source TCP protocol. Only a few TCP sources, which are picked randomly, in the aggregate, drop their window sizes, thus facilitating the aggregate to maintain its overall transmission rate up to the desired level.

5.3 The Idea: Keep the number of tokens in the bucket constant

If somehow, we are able to maintain the number of tokens in the bucket for a given aggregate at some constant level, it would imply that the number of tokens going in the bucket is equal to number of tokens going out of the bucket. If the rate of flow of tokens in the bucket corresponds to the allocated rate, then the network is giving the aggregate exactly the bandwidth allocated to it. Though maintaining the token level in the bucket constant is a trivial problem for non-responsive aggregates as one can just drop the excess packets. The issue becomes more challenging for responsive flows, such as TCP, which indeed constitute a bulk of today's Internet traffic. As explained

5.4 Impact

Token bucket, today is employed extensively in almost all types of bandwidth allocation mechanisms used by Internet Service Providers (ISP) for data traffic, IP telephony, video conferencing, metro access routers, etc. An improvement in it, would therefore have consequential effects.

5.5 Scepter

Scepter aims to improve the performance of best-effort traffic by enabling aggregates operate very close to their allocated bandwidth. We achieve our objectives by modi-

fying the token bucket of the traffic marker at the edge router.

Our algorithm guarantees TCP aggregates a pre-specified bandwidth over very small time intervals unlike the present token bucket mechanism where it keeps fluctuating and only an aggregate rate over a very long time interval is guaranteed.

The idea is to treat the bucket as a queue and then apply the various AQM strategies for keeping the instantaneous queue stable in the context of bucket. Number of bytes in the queue corresponds to the bucket size less the number of tokens in the bucket. A full queue implies an empty token bucket and an empty queue implies a bucket full of tokens. To guarantee a TCP aggregate a constant bandwidth throughout, we must be able to maintain the token level in the bucket constant and put in tokens in the bucket at a constant rate that correspond to the bandwidth allocated to it. This ensures that the aggregate is constantly getting a rate equal to the rate of flow of tokens in the bucket. A lot of work on AQM focuses on maintaining the instantaneous queue constant. We take cue from some of these [5, 20] in designing our algorithm. The algorithm essentially consists of two modules, the *estimator* and the *stabilizer*.

5.5.1 Estimator

As was discussed in Chapter 2, NLMS is indeed able to predict the incoming traffic quite accurately. In Scepter also, we use a similar algorithm to predict the incoming traffic at a future instant of time based on which we will take a decision, whether or not to allow the packet to pass through. We will predict the traffic at a future instant in time at every packet arrival, exactly as we did in APACE.

5.5.2 Stabilizer

The purpose of this module is to keep the number of tokens in the bucket at a constant level, so as to guarantee the allocated bandwidth to the source aggregate. We are currently in the *implementation stage* of this module and are considering the following options:

Probability of packet drop independent of target token level in the bucket

This module maintains the stability of the queue using a probability drop function similar to that of APACE in Section 2.4. "max_p", as was explained in Chapter 2 is the maximum probability with which an incoming packet can be dropped in anticipation of congestion, given that there are sufficient number of tokens available in the bucket for the same to be forwarded. This will lead to a stabilization of the bucket[2] at any arbitrary value similar to the case in APACE. We believe that such an algorithm will perform better, since the level at which the bucket remains stable is not very critical to us (the level remaining constant is much more critical) and by not specifying the target token level in the bucket, we end up providing more freedom to the algorithm to adjust according to the traffic needs.

Probability of packet drop dependent on target token level in the bucket

This module maintains the stability of the queue using a simple controller that drops packets in proportion to the difference between the target number of tokens and the actual number of tokens in the bucket. Here, we try to keep the number of tokens in the bucket at a pre-specified target level. We use the following equation to adaptively vary the value of the packet dropping probability with the number of tokens in the bucket:

$$p = \frac{max_p}{B}[T_{target} - T_{current} + T_{predicted}] \tag{5.1}$$

where,
p = Calculated packet loss probability.
max_p = A constant less than 1.
B = Bucket size.
T_{target} = The target token level in the bucket at which we want the bucket to be stable.
$T_{current}$ = Number of tokens in the bucket currently.
$T_{predicted}$ = Predicted number of tokens in the bucket at a future instant of time.

[2]By stabilization of the bucket, we mean that the number of tokens in the bucket remain constant.

5.6 Conclusions & Future Work

The algorithm is still in its implementation stages. Though intuitively it should perform better than the classic token bucket, we are yet to provide with any factual results supporting our claims. A detailed study on the performance of Scepter is an area of future research. Scepter, if successful, can change the way in which bandwidth is allocated and managed in the internet.

CHAPTER

6

Conclusions & Future Work

It's more like the future each day

In this chapter, we explain the following:

- *Summary of the contributions of the thesis*
  - *APACE*
  - *LTCP*
  - *Scepter*
- *Directions for future work*

Chapter 6

Conclusions and Future Work

In our efforts to combat the ill-effects of congestion, we have in this thesis suggested three new algorithms. We now summarize the contributions and give directions for future work on them.

In this thesis, we have presented a novel AQM scheme based on (predicted) instantaneous queue length. We conclude that algorithms like NLMS can indeed be used to predict the instantaneous queue length. The APACE scheme performs better than existing AQM schemes and it adapts faster to changes in network conditions and is able to keep the instantaneous queue stable. APACE gives higher link utilization, a much lower packet loss rate and comparable delay as compared to PAQM in addition to the instantaneous queue stability comparable to PAQM. In addition, the scheme is able to achieve better operating points than all other existing schemes (in terms of delay-link utilization or delay-loss trade-off) for both single and multiple bottleneck links. The link utilization of APACE remains more or less constant at a high value with change in its parameters in networks with multiple bottlenecks. The parameters can thus be adjusted suitably to achieve a given delay and packet loss rate without worrying much about link utilization. Moreover this can be achieved by varying just one of its parameters *ie.*, max_p.

APACE with max_p being varied adaptively is expected give even better performance and is an area of future research. One possibility is to vary max_p in proportion to the difference between the predicted queue and the target queue in order to keep the the queue even more stable at a pre-defined level. There can be other alternate ways to vary max_p, in order to enhance link utilization, packet delay and/or packet loss in particular. Also, in the current version of the scheme, the prediction is made

at every packet arrival. A scenario where the number of packet arrivals is large might lead to a lot of computational overhead. To overcome this problem, one might consider a scheme where the prediction is done at periodic time intervals, though this might come at the expense of lower prediction accuracy.

LTCP attempts to improve the end-to-end performance of TCP in heterogeneous networks which is general and not specific to a particular type of network. LTCP is able to learn the cause of packet loss in the network. It is fair and also TCP friendly. How to adapt LTCP and apply it to other form of networks like DiffServ, etc is an area of future study.

Though intuitively, Scepter should perform better than the classic token bucket, it is still in its implementation stages. A detailed study of the algorithm is left as a topic of future research. If successful, it could well be a major break through in bandwidth allocation and management in the Internet.

Bibliography

[1] Van Jacobson, Michael J. Karels, "Congestion Avoidance and Control", *ACM Computer Communication Review; Proceedings of ACM SIGCOM*, August, 1988. vol.18-4, pp. 314-329, 1988.

[2] Sally Floyd and Van Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *ACM/IEEE Transactions on Networking*, vol. 1, no. 4, pp. 397-413, August 1993.

[3] Teunis J. Ott and T. V. Lakshman and Larry H. Wong, ""SRED: Stabilized RED", *Proceedings of INFOCOM*, pp. 1346-1355, 1999.

[4] Srisankar Kunniyur and R. Srikant, "Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management", *Proceedings of ACM SIGCOMM*, pp. 123-134, August 2001.

[5] Yuan Gao, Guanghui He, and Jennifer C. Hou, "On Exploiting Traffic Predictability in Active Queue Management", *Proceedings of IEEE INFOCOM*, 2002.

[6] T. Bonald , M. May and J. Bolot, "Analytic Evaluation of RED Performance", *Proceedings of IEEE INFOCOM*, pp. 1415-1424, 2000.

[7] Martin May, Jean Bolot, Christophe Diot and Bryan Lyles, "Reasons not to deploy RED", *Seventh International Workshop on Quality of Service*, June 1999.

[8] Sally Floyd, "RED: Discussions of Setting Parameters", `http://www.aciri.org/floyd/REDparameters.txt` (Last accessed in May 2003.)

[9] Van Jacobson, K. Nichols, and K. Poduri, "RED in a Different Light", `www.cnaf.infn.it/~ferrari/papers/ispn/red_light_9_30.pdf`, September 1999 (Last accessed in June 2002.)

[10] H. Ohsaki and M. Murata, "Steady state Analysis of the RED Gateway: Stability, Transient Behaviour, and Parameter Setting", *IEICE Transcations on Communications*, vol. E85-B, vo. 1, January 2002.

[11] C. V. Hollot, Vishal Misra, Don Towsley, and Wei-Bo Gong, "A Control Theoretic Analysis of RED", *Proceedings of IEEE INFOCOM*, pp. 1510-1519, 2001.

[12] Wu-chang Feng, Dilip D. Kandlur, Debanjan Saha and Kang G. Shin, "A Self-Configuring RED Gateway", *Proceedings of IEEE INFOCOM*, pp. 1320-1328, 1999.

[13] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management", `www.icir.org/floyd/papers/adaptiveRed.pdf`, August 2001 (Last accessed in May 2003.)

[14] F Anjum, and L. Tassiulas, "Fair Bandwidth Sharing among Adaptive and Non-Adaptive Flows in the Internet" *Proceedings of IEEE INFOCOM*, pp. 1412-1420, 1999.

[15] Don Ling, Robert Morris, "Dynamics of Random Early Detection" *Proceedings of ACM SIGCOMM*, 1997.

[16] Wu-chang Feng and Dilip D. Kandlur and Debanjan Saha and Kang G. Shin, "Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness", *Proceedings of IEEE INFOCOM*, pp. 1520-1529, 2001.

[17] Rong Pan and Balaji Prabhakar and Konstantios Psounis, "CHOKE, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation", *Proceedings of IEEE INFOCOM*, pp. 942-951, 2000.

[18] Simon Haykin, *Adaptive Filter Theory*, Third Edition, Prentice-Hall, 1996.

[19] "Network SImulator", `http://www.isi.edu/nsnam/ns/`,

[20] Abhishek Jain, Rahul Verma and Abhay Karandikar, "An Adaptive Prediction based Approach for Congestion Estimation in Active Queue Management (APACE)", *Technical report, Department of EE, IIT Bombay*, `www.ee.iitb.ac.in/uma/~abhishek/research/apace.html`, December 2002.

[21] G.Wright and W.R.Stevens, *TCP/IP Illustrated, Volume 2:The Implementation*, Addison Wesley ,1995.

[22] W.C.Y.Lee, *Mobile Communications Design Fundamentals*, 2^{nd} Edition, John Wiley ans Sons,1993.

[23] V.Tsaoussidis *et al*, "Energy/Throughput Tradeoffs of TCP Error Control Strategies", *Proceedings of 5th IEEE Symposium Computer and Communications*, July 2000.

[24] Kevin Fall and Sally Floyd, "Simulation based Comparisons of Tahoe, Reno and SACK TCP", *ACM Computer Communication Review*, vol. 26, no.3, July 1996.

[25] M.Mathis and J.Madhvi, "Forward Acknowledgment: Refining TCP Congestion Control", *Proceedings of ACM SIGCOMM*, August 1996.

[26] Kostas Pentikousis, "TCP in Wired-cum-Wireless Environments", *IEEE Communications Surveys*, October 2000.

[27] S.Bradner, "Keywords for Use in RFCs to indicate Requirement Level", *BCP 14, RFC 2119*, March 1997.

[28] R.Barden, "Requirements for Internet Hosts - Communication Levels", *STD 3, RFC 1122*, October 1989.

[29] M.Mathis, "The Ratehalving Algorithm fir TCP congestion Control", `http://www.psc.edu/networking/rate_halving.html`, April 2001 (Last accessed June 2003.)

[30] J.Hoe, "Startup Dynamics of TCP's Congestion Control and Avoidance Schemes", *Master's Thesis*, MIT 1995.

[31] Amit Aggarwal, Stefan Savage and Thomas Anderson, "Understanding the Performance of TCP Pacing", *Proceedings of IEEE INFOCOM*, pp. 1157-1165, 2000.

[32] M. Gerla, M. Y. Sanadidi, R. Wang, A. Zanella, C. Casetti and S. Mascolo "TCP Westwood: Congestion Window Control Using Bandwidth Estimation", *Proceedings of IEEE Globecom*, vol. 3, pp. 1689-1702, 2001.

[33] K. Ramakrishnan and S. Floyd, "A Proposal to Add Explicit Congestion Notification (ECN) to IPv6 and to TCP", *Internet Draft draftkksjf -ecn-03.txt*, 1998.

[34] K. K. Ramakrishnan, Sally Floyd and D. Black, "The Addition of Explicit Congestion Notification to IP", *Internet Draft*, `http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-ecn-03.txt`, March 2001 (Last accessed in April 2003.)

[35] K.Pentikousis,H.Badr and B.Karmah, "TCP with ECN:Performance Gains for Large TCP Transfers", *SBCS-TR-2000/01, Department of Computer Science, SUNY at Stony Brook*, March 2001.

[36] Jitendra Padhye, Victor Firoiu, Don Towsley, Jim Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation", *Proceedings of ACM SIGCOMM*, pp. 303–314, 1998.