ABSTRACT

ORUGANTI, SAI. Performance of Robust Active Queue Management Schemes and Window Adaptation Schemes in IP Network. (Under the direction of Dr. Mihail Devetsikiotis)

The Internet today has emerged as a ubiquitous network consisting of globally-shared resources. Optimal sharing of these resources raises the issue of resource and cost allocation which, in turn, leads to network performance modelling. In our work we emphasize the importance of performance evaluation and monitoring of network resources to achieve their optimal utilization. We analyze the network as a decoupled system consisting of end nodes and routers. We analyze the algorithms running on each component and propose modifications. For the router we study the existing and widely-deployed active queue management scheme, Random Early Detection (RED) and its predecessor, TailDrop scheme. Emphasizing robustness and end-to-end delay, we propose our modification to RED and show that it achieves better results compared to RED. Similarly, for the end nodes, we explore the window adaptation scheme of the widely-deployed cooperative transport protocol, TCP Reno. Stressing packet re-sent ratio and power, we show that our proposed modification of window adaptation schemes achieves better performance than TCP Reno. For a more balanced research we compute the algorithmic complexity of each algorithm to show that better results can be achieved at the expense of increased algorithmic complexity.

Performance of Robust Active Queue Management Schemes and Window Adaptation Schemes in IP Network

by

Sai S. Oruganti

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Operations Research Program

Raleigh

2003

Approved By:

Dr. A. Nilsson	Dr. Y. Viniotis

Dr. M. Devetsikiotis
Chair of Advisory Committee

Dedicated to my parents

Biography

Sai Swaroop Oruganti was born in Jamshedpur, India. He was schooled at Dayanand Anglo Vedic School, Jamshedpur. He attended Bhavan's Sri Ramakrishna Vidyalaya for his intermediate studies, and went on to pursue a Bachelor's Degree at the Indian Institute of Technology, Kharagpur, India. He joined North Carolina State University for his graduate studies in the year 2000.

Acknowledgements

I would like to thank my advisor, Dr. Devetsikiotis for his guidance, stimulating suggestions and encouragement throughout the course of my research work and while writing this thesis. I am indebted to him for introducing me to the exciting research area of Network Performance Modelling and Evaluation. I would like to thank Dr. Nilsson and Dr. Viniotis for serving on my committee. I would like to extend my special thanks to them for their continued support, assistance and helpful comments. I would also like to thank Steve Blake and Zsolt Haraszti from Ericsson IPI, Raleigh, for their comments and assistance. I am very thankful to Ms. Nancy House for her editorial comments and suggestions. I would also like to thank my friends at NC State for their suggestions and constant support. Special thanks go to my parents and friends elsewhere for their encouragement, faith and support.

Publications

- Sai S. Oruganti, Mike Devetsikiotis, Ozdemir Akin and J. Keith Townsend, "Exploiting Traffic Prediction and Adaptive Control to Implement Robust Active Queue Management Schemes," *Center for Advanced Computing and Communications (CACC), Raleigh, NC, Technical Report TR-01/07, July 2003.*

- Sai S. Oruganti and Mike Devetsikiotis, "Analyzing Robust Active Queue Management Schemes: A Comparative Study of Predictors and Controllers," Published in *ICC 2003, Anchorage, Alaska, May 2003.*

- Ozdemir Akin, J. Keith Townsend, Sai S. Oruganti and Mike Devetsikiotis, "Exploiting Traffic Prediction and Adaptive Control to Implement Robust Active Queue Management Schemes," *Center for Advanced Computing and Communications (CACC), Raleigh, NC, Technical Report TR-01/07, July 2002.*

- Sai S. Oruganti, Mike Devetsikiotis, Ozdemir Akin and J. Keith Townsend, "Exploiting Traffic Prediction and Adaptive Control to Implement Robust Active Queue Management Schemes," *Center for Advanced Computing and Communications (CACC), Raleigh, NC, Technical Report TR-01/07, November 2001.*

- Sai S. Oruganti and Mike Devetsikiotis, "Robust AQM Schemes for Correlated and Cooperative Traffic," Submitted to *Computer Communications.*

- Sai S. Oruganti and Mike Devetsikiotis, "Simulation Study of Window Adaptation Schemes under Imperfect Information", In preparation for *IEEE Globecom 2004.*

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation: The Need for Network Analysis

The Internet today has emerged as a ubiquitous network consisting of globally shared resources. A key feature of the Internet is the role of the constituent networks. These networks are integrated entities which actively contribute their resources which range from backbones to regional transmission services to local area networks (LANs) [11]. Pooling resources of so many constituents into a massively interconnected environment raises the issue of resource and cost allocation. Addressing these issues requires an exhaustive evaluation of network performance modelling. Performance monitoring of the network resources can yield a tremendous amount of information about the component's efficiency with respect to the network traffic that it handles. Performance evaluation, on the other hand, can reveal the necessary steps (or precautions) to take to maintain an optimal allocation of resources for ever-changing network traffic.

In our work we considered the issue of performance evaluation of network components for an IP network. We logically divided the IP network into a combination of two components: the end nodes and the routers. We studied the algorithms running on the network components: the end nodes running cooperative transport protocols (like TCP) and the routers implementing active queue management (AQM) schemes. We investigated the currently implemented algorithms in the network components and proposed our algorithms to show that better network performance is a tradeoff between algorithmic complexities.

1.2 Components of an IP Network

For our research we logically divided the IP network as consisting of two components - the AQM schemes at routers and the window adaptation scheme used by the cooperative transport protocols at end nodes.

Active queue management schemes are congestion-avoidance schemes in packet switched networks. They are designed to detect incipient network congestion and proactively drop[1] packets in order to avoid future congestion. The packet dropping is proportional to the congestion it faces. This action of an AQM scheme not only avoids "global synchronization" but also ensures bandwidth proportional to each connection's share.

The window adaptation scheme used by the cooperative transport protocol serves a dual purpose. It carefully injects packets into the network until it claims its share of bandwidth. Upon sensing congestion it reacts by decreasing its packet sending rates. Thus, it participates not only to obtain its fair share of bandwidth but also to ensure that it takes part in relieving congestion.

These two components form a coupled system. The routers drop packets to send the end nodes a signal to slow down, and slowly increase their packet sending rates to avoid congestion. Hence, the action of one network component is dependent on the other.

1.3 Network as a Decoupled System

We observed that, for a best effort network such as an IP network, the router does not distinguish between various traffic classes. Its packet dropping rate depends *exclusively* on its available buffer capacity and on its estimate of future arrivals. Moreover, the behavior of the end nodes is dictated *only* by the end-to-end behavior experienced by their packets. Hence, we inferred that, although the IP network consists of a coupled system of end nodes and routers, their behavior is basically defined only by their observations. So, we had to analyze the network as a decoupled system.

In our work we analyzed each network component independently of the other. We first concentrated on the AQM schemes at the router and treated the rest of the network as a "black-box." We analyzed scheme behavior with respect to robustness and queueing delay and proposed our modifications. We then dealt with the cooperative transport protocols at the end nodes, focusing especially on their window adaptation schemes and modelled the rest of the network as a "black-box." We analyzed scheme behavior with respect to packet loss ratio and power and propose our

[1] Based on their implementation, the routers either mark or drop packets. In our work we have considered both marking and dropping effects. Since dropping is the default action of routers, we refer to packet marking explicitly where indicated.

modifications.

1.4 Summary of the Thesis

In this thesis our efforts can be divided into six parts: In the first part we analyze AQM schemes as a combination of a measurement module and a control module. This method of analysis allows us to carefully observe the effect of each module on the performance of the AQM scheme. For each of the modules, we implement different predictors and controllers and compare the performance of an AQM scheme for various combinations. In our simulations we show that *careful* control decisions based on prediction of future packet arrivals always lead to an increase in performance.

In the second part we study the effects of the fact that traffic generated by TCP sources is correlated in nature. Dependence of future traffic measurements on current value is a result of (positively) correlated traffic, and we use this property of dependence to predict future traffic intensity. We also suggest ways to exploit the dependence using different predictors.

In the third part we emphasize the relevance of using robustness and delay jitter as reliable performance metrics of an AQM scheme rather than goodput. In this regard, we define our idea of robustness. We concentrate our attention on the performance of robust AQM schemes that exploit the correlation that exists in the network traffic. We show that, with our definition of robustness, a positive correlation can be established between robustness, goodput and link utilization. We use goodput, link utilization and algorithmic complexity strictly as additional measures of performance.

In the fourth part we concentrate on the study of window adaptation schemes of cooperative transport protocols running on end nodes. We also mention related work done in the area of developing new window adaptation algorithms to increase TCP throughput. In this part we summarize the IETF recommended guidelines to be followed before proposing a *new* window adaptation scheme for a cooperative transport protocol.

In the fifth part we propose a way to estimate the congestion level for real networks only through the ECN bit marks on the ACKs. Our belief that the ratio of marked packets to the transmitted packets (packet mark ratio) is an indicator of congestion is the basis for our window adaptation scheme. We use an existing predictor to record and "predict" the packet mark ratio and use an existing utility model to determine the optimum window size for the next transmission to reduce the number of re-sent packets.

In the sixth part we combine our observations of AQM scheme analysis and window adaptation analysis to produce results for a combination of robust AQM schemes at the network level and efficient window adaptation scheme at the end node level.

1.5 Thesis Organization

This thesis is organized as follows. Chapter 2 discusses the various existing AQM and window adaptation schemes. Chapter 3 introduces the AQM schemes and describes the need for robust AQM schemes. It also discusses the characteristics and importance of robustness in an AQM scheme. Chapter 4 introduces the essential features of a cooperative transport protocol and reiterates the Internet Engineering Task Force (IETF) recommended guidelines for proposing new cooperative protocols. This chapter illustrates the fact that all new window adaptation schemes have to be compared with their existing counterparts with respect to performance and algorithmic complexity. Chapter 5 provides the results of our work and inferences drawn from it. Finally, in Chapter 6, we summarize our work done in this thesis and provide insight into possible future research areas. In Appendix I we list individual performance graphs for scenarios not covered in the text. In Appendix II we show the algorithmic complexity for each algorithm used in our work.

Chapter 2

Related Work

In this chapter we discuss both components of an IP network, the Active Queue management schemes implemented in the routers and the window adaptation schemes used by the Cooperative Transport Protocols implemented on the end nodes. We also discuss in detail their various attributes and limitations.

2.1 Active Queue Management Schemes

Here we discuss the existing active queue management schemes, their characteristics and their advantages and limitations, where applicable. We also compare each of the schemes in order to assess network performance with respect to fairness in allocation of bandwidth, penalty for user misbehavior, ability to remove bias against bursty traffic and power to avoid global synchronization. For historical reasons we also discuss the operation of a TailDrop scheme, which was the first scheme to be implemented in the routers before the AQM schemes.

2.1.1 TailDrop (DT)

This was the first and, by far, the simplest scheme to be implemented at the routers. A TailDrop (or DropTail) scheme works by accepting all arriving packets into the buffer until the buffer has reached capacity after which it begins to drop packets until some space in the buffer becomes available.

2.1.2 Random Early Detection (RED)

TailDrop has drawbacks, such as: throughput loss due to global synchronization, unsuccessful control of misbehaving users and inability to deal with connections with different round-trip times (RTT). Random early detection (RED) [19, 9] was developed to address these shortcomings. RED is somewhat similar in nature to the DECbit congestion avoidance scheme.

Characteristics

- Four variables are required:

  wq = weight of the queue;

  min_{th} = minimum threshold;

  max_{th} = maximum threshold; and

  max_p = maximum value for packet marking probability p_b.

- For each incoming packet, the new average queue size is calculated as:

  $avg\ queue\ size \leftarrow \alpha * inst\ queue\ size + (1 - \alpha) * avg\ queue\ size.$

  Packet drop decision is then taken as:

  $avg\ queue\ size < min_{th}$ means accept the packet;

  $min_{th} \leq avg\ queue\ size \leq max_{th}$ means accept the packet with a probability; and

  $avg\ queue\ size > max_{th}$ means drop the packet.

- The queue size is calculated by Exponential Weighted Moving Average (EWMA), which means that, depending on the weight factor α, the new queue average will resemble the previous average. This is essential for removing bias against bursty traffic. A high value of α will make the queue average sensitive to bursty traffic, whereas with a low value of α, the average queue value will change slowly with traffic.
- The fraction of packets of a connection marked by RED is, roughly, in proportion to the connection's share of bandwidth. This is how RED achieves its fairness.
- The RED gateway was evaluated with different traffic mixes (FTPs and Telnets), and it was tested for the following goals: congestion avoidance, global synchronization, window reduction at transient congestion, overhead in implementation, maximizing power, achieving fairness and is applicable for network connections with a range of RTTs and goodput.
- RED has been known to have no bias towards bursty traffic, and hence, it is capable of handling transient connections which have a small window and a long delay-bandwidth product.
- RED identifies misbehaving users by randomly marking the incoming packet. Since misbehaving connections are known to send large numbers of packets, they would be the

connections which will have a higher marked fraction of packets. These connections can reduce their windows once they sense a higher fraction of marked packets.

Advantages

- RED is simple and efficient to implement.
- RED can be implemented in high-speed networks since it has little overhead in calculating various parameters.

Limitations

- RED's non-bias against bursty traffic is proven only for FTP sources. It would be beneficial and more informative if it were studied for different traffic mixes, such as a network having FTP, Telnet and UDP sources with different RTTs and window sizes.
- Similarly, in the case of misbehaving users, RED has been tested only on TCP connections that cooperate with the gateways in congestion control. Testing RED for non-cooperative UDP connections will probably clearly reflect RED's ability to control misbehaving users.
- The values of wq, min_{th}, max_{th} and max_p have to be determined for each network, depending on its configuration (window size, RTT, number of nodes) and traffic mixes.

2.1.3 Flow Random Early Detection (FRED)

The limitation of the above RED algorithm is that it ensures fairness by marking a fraction of packets from a connection based on that connection's share of bandwidth. This, however, has only been tested for TCP sources. When given a mixture of different traffic types, RED allows unfair bandwidth sharing as it imposes the same rate loss, irrespective of bandwidths. FRED [30] uses per-active-flow accounting to improve fairness when different traffic types share a router.

Characteristics

- The traffic types have been categorized as non-adaptive, robust and fragile.

  Non-adaptive - This type of traffic takes as much bandwidth as it requires and does not slow down when congestion is detected. It competes with adaptive sources for buffer space and bandwidth. An example includes UDP traffic.

  Robust - This type of traffic is aware of congestion and reduces its window when congestion is detected. When extra bandwidth is available robust traffic increases its window size. It always sends data and its packets are always buffered at the gateway, which ensures fair share of bandwidth. An example includes TCP traffic.

Fragile - This type of traffic is also aware of congestion, but it is either sensitive to packet losses or slower to adapt to more available bandwidth. It has fewer packets buffered at the gateway. An example includes Telnet traffic.

- The following variables are used:

  wq, min_{th}, max_{th} and max_p: as defined earlier;

  min_q and max_q, minimum and maximum number of packets each flow is allowed to buffer;

  avg_{cq}, average per flow buffer count;

  $qlen_i$, queue length (packets buffered) for each flow i; and

  $strike_i$, number of defaults (over-runs) by flow i.

- FRED maintains separate buffer for each active flow. Hence the "bulkiness" of a system increases in proportion to the number of active flows at the gateway.
- Fragile flows: An incoming packet is always accepted if the connection has less than min_q packets buffered and the average buffer size is less than max_{th}. For a flow with buffer size between min_q and max_q, the incoming packet is accepted probabilistically. Hence, FRED protects fragile flows.
- Heterogeneous robust flows: When the number of active connections is small, it might be possible for some flows to have more than min_q packets buffered leading to queue size greater than max_{th}, thus forcing FRED to randomly drop packets (by imposing the same rate loss on each defaulter). To avoid this packet drop, which is independent of the bandwidth, FRED dynamically increases the limit min_q to avg_{cq}.
- Non-adaptive flows: FRED keeps a count of the number of times a connection buffers more than max_q packets in the strike variable. Flows with high strike values are not allowed to buffer more than avg_{cq} packets. Since non-adaptive flows are bound to have high strike values, they will not be allowed to consume a large bandwidth. This allows adaptive flows to increase their windows when spare bandwidth is available, and hence get their fair share.
- FRED also has the ability to support many flows. Every time the buffer needs to be allocated, the inactive flows are given priority over the active ones.

Advantages

- FRED provides more equitable bandwidth sharing by implementing per-flow accounting for each flow and selectively dropping packets from flows that take more than their share of bandwidth.
- FRED is able to provide fairness to various connections having a range of RTT and

window sizes.

Limitations

- FRED uses many state variables to keep account of each flow. This makes it very "bulky" to implement.

2.1.4 Stabilized RED (SRED)

The previous variant of RED, FRED, suffers from the drawback that it has to keep an extensive list of all connections that have packets buffered at the gateway. Moreover, for each active connection it has to maintain its min and max buffer sizes and strike values. This variant of RED requires buffer space in proportion to the number of active connections at the gateway. The basic goal of SRED [35] is to increase the occupancy of the (FIFO) buffer, independently of the number of active flows, by estimating the number of active flows.

Characteristics

- Zombie list: This is a list created to augment the buffer's memory. It stores a list of few recently seen flows (zombies). This list stores information about each packet along with the time stamp of its arrival. It also implements *Count*, which is a measure of number of hits. As long as the zombie list is empty, all packets are added to it along with their time stamp. When the zombie list is full, each arriving packet is compared with a randomly selected zombie from the list.

  Hit: When the arrival packet's flow matches the zombie. *Count* is increased by one and the timestamp is reset to the arrival time of packet in the buffer.

  Miss: When the two are not of the same flow. With a probability p the flow identifier of packet is written on that zombie and count decreased by one. It remains unchanged with a probability $1 - p$.

  The drop probability, however, does not depend on a hit or a miss. If the buffer occupancy demands it, then a packet is dropped.
- Relationship between hits and number of active flows:

  Π_i - probability that a packet belongs to flow i;

  $\Pr\{\text{hit of } k^{th} \text{ packet occurs}\} = \sum_1^N \Pi_i^2$;

  For N active flows of identical traffic intensity, $\Pi_i = \frac{1}{N}$; and

  $\Pr\{\text{hit of } k^{th} \text{ packet occurs}\} = \frac{1}{N}$.

  However, for non-identical flows, we have

$\frac{1}{N} \leq \sum_1^N \Pi_i^2 \leq 1$.

- Simple SRED and Full SRED

  In simple SRED the buffer is divided into 3 parts, and depending on the instant buffer size, the packet is either kept or dropped with a probability p_{zap} which is a function of

  q - instantaneous queue size and

  Pr{packet hit}

  In simple SRED the hits are used to estimate the number of active flows, which are then used to set the dropping probabilities.

  In full SRED the hits are used directly to calculate the dropping probabilities. The dropping probability for full SRED is $p_{zap} * (1 + hit(k)/P(k))$. The value of $hit(k)$, for k^{th} packet, is 1, if a hit occurs, and 0, otherwise.

Advantages

- Estimating of the number of active connections without maintaining per-active-flow accounting really reduces the "overhead" that is incurred in the case of FRED. Maintaining a zombie list is far simpler than keeping a list of all active flows.
- SRED achieves a great deal in stabilizing the buffer occupancy so that by adjusting the drop probability the buffer occupancy can be controlled. For number of flows (N) less than 256, the buffer occupancy is independent of the number of flows; but, for greater values of N, the buffer occupancy gradually increases with N.
- The hit-or-miss scheme can be used to identify misbehaving flows without the need to keep per-flow accounting. Flows with high hit count can be monitored and controlled.

Limitations

- The buffer occupancy is decided by p_{zap} and p_{max}, and these values probably have to be determined for each network configuration to maximize occupancy.

2.1.5 RED with the Unresponsive Flow Identification

This variant of RED is basically designed to promote end-to-end congestion control in a network [17]. The increased use of non-congestion-controlled traffic on the Internet causes unfair sharing of bandwidth and buffer space and might eventually lead to congestion collapse. During congestion this variant tries to identify each high-bandwidth flow as unresponsive, not-TCP friendly and disproportionate-bandwidth flow and tries to restrict its bandwidth. Instead of trusting transport protocols the network itself participates in congestion avoidance and control.

This paper discusses the problems of fairness and the dangers of congestion collapse. It also suggests that presently there are no concrete incentives for implementing end-to-end congestion control and misbehaving flows manage to get more than the allotted share of bandwidth and the routers have to strictly enforce certain schemes, like weighted round robin (WRR), to ensure fairness.

Characteristics

- First approach - High-bandwidth flows are divided into 3 categories:
  - Not TCP-friendly - A flow is not TCP-friendly when its arrival rate is more than the arrival rate of a conformant TCP connection. In other words:

    T = maximum sending rate for a TCP connection;
    B = TCP connection sending packets of B bytes;
    R = RTT + queuing delays; and
    p = packet drop probability.

    $$T \leq \frac{1.5 * \sqrt{2/3} * B}{R * \sqrt{p}}. \tag{2.1}$$

    Therefore, any flow sending more than T Bps data can be treated as a non TCP-friendly flow. This test of TCP-friendliness does not imply that a flow responds to every packet drop as a conformant TCP. It simply tests whether, during congestion, a flow is using more bandwidth than the most aggressive conformant TCP would.
  - Unresponsive flows - A flow is unresponsive when an increased packet drop from that flow does not lead to a decrease in its window size. Equation (2.1) shows that if the long-term packet drop rate increases by a factor of k, then the arrival rate should decrease by a factor roughly equal to $\sqrt{k}$. If a flow does not follow this guideline, then it can be considered an unresponsive flow. This test is only applied to high-bandwidth flows.
  - Flows using disproportionate bandwidth - Flows that use significantly larger portions of bandwidth during times of congestion come under this category. If we let n be the number of flows with packet drops in the recent interval, the first component of a disproportionate bandwidth test defines a disproportionate bandwidth flow as one whose fraction of aggregate arrival rate is greater than $log(3n)/n$.

The second component of this test takes into account the level of congestion. Here a disproportionate bandwidth flow is defined as one whose arrival rate is greater than $c/\sqrt{p}$ Bps for some constant c.

- Second approach - Another approach is to use per-flow scheduling mechanisms such as, WRR or fair queuing, to isolate all best-effort flows at the routers and to prevent them from using a large share of bandwidth.

2.1.6 CHOKe

All of the above-mentioned RED variants incur some overhead by maintaining certain types of state information. This factor makes them either simple to implement or provides fairness, but not both. This variant of RED provides a stateless active buffer management scheme called CHOKe (CHOose and Keep for responsive flows, CHOose and Kill for unresponsive flows) [36, 38].

Characteristics

- For each incoming packet:
  $avg.\ queue\ size \leq min_{th}$ means allow the packet;
  $min_{th} \leq avg.\ queue\ size \leq max_{th}$ means compare with a randomly selected packet (drop candidate packet)from the queue. If they have the same flow ID, both are dropped else the incoming packet is accepted; and
  $avg.\ queue\ size > max_{th}$ means drop the packet.
- The algorithm can also be modified so that instead of one drop candidate packet, we can select $m > 1$ packets. This definitely improves CHOKe's performance. This is especially true when there are multiple unresponsive flows, as more drop candidate packets would increase the chances of packets being dropped from the queue.
- Another way to modify the above algorithm would be to divide the region ($max_{th} - min_{th}$) into k regions and select a different m for each region.

Advantages

- CHOKe has been tested for three network configurations:
  Single congested link;
  Multiple congested link; and
  Multiple drop candidates and misbehaving flows.

In all of the cases CHOKe has been tested for different traffic mixes and has been found to achieve

fair queuing with minimal implementation overhead. It would be interesting to test it on more complicated network topologies.

2.1.7 Weighted RED

Weighted RED (WRED) [1] combines the capabilities of the RED algorithm with IP precedence to provide for preferential traffic handling of higher priority packets. Packets with higher IP precedence are less likely to be dropped than packets with a lower precedence. WRED can selectively discard lower priority traffic when the interface begins to get congested. Thus, higher priority traffic is delivered with a higher probability than lower priority traffic. In other words, WRED has differentiated performance characteristics for different classes of service.

WRED is used in the core routers of a network. The edge routers assign IP precedence to packets as they enter the network. WRED uses these precedences to determine how to treat the packet.

Characteristics

- The entire algorithm of WRED is similar to RED with the following changes noted below:
- WRED treats non-IP traffic as precedence 0, the lowest. Precedence allocation for Cisco class is:
  Gold - highest precedence of 7 - guaranteed latency and delivery for mission critical applications. An example includes voice sent over the Internet as an IP packet.
  Silver - medium precedence of 3 and 4, which means guaranteed delivery
  Bronze - low precedence of 0, which means best effort. Examples include e-mail and ftp.

- Average queue size calculated as
  $avg = (old\_avg * (1 - 2^{-n})) + (current\_queue\_size + 2^{-n})$.
  As explained previously, a moderately high value of n will prevent WRED from fluctuating a lot, and a moderately low value of n will make the buffer sensitive to bursty traffic.

Advantages

- Unlike other variants, WRED attempts to anticipate and avoid congestion rather than controlling it.

Limitations

- WRED is useful only when the bulk of traffic is TCP/IP because TCP/IP sources are adaptive, and hence, they will respond to congestion.

2.1.8 Double Slope RED (DSRED)

Every RED variant that has been mentioned above is either simple to implement or provides fairness, but not both. As mentioned before, SRED takes care of both but has to implement a simple data structure called "zombie list." Although SRED is scalable, it suffers from low throughput. In fact adaptive RED also has its drawbacks, as dynamic/adaptive determination of RED parameters complicates buffer management of high-speed routers.

DSRED [42] modifies the original RED slightly, in that, instead of having a single linear drop probability function, it has double linear drop probabilities.

Characteristics

- The entire buffer space (of size N) is divided into four parts: min_{th} is represented by K_l; max_{th} is represented by K_h; and $K_l < K_m < K_h$.
- The algorithm is the same as RED's except that the drop probability is calculated in the following manner:

  $p_d(i)$ = average (avg) packet drop probability at state i;
  $p_a(i)$ = average (avg) packet accepting probability at state i; and
  $p_a(i) = 1 - p_d(i)$.

$$p_d(avg) = \begin{cases} 0 & avg < K_l, \\ \alpha * (avg - K_l) & K_l \leq avg \leq K_m, \\ 1 - \gamma + \beta * (avg - K_m) & K_m \leq avg \leq K_h, \text{ and} \\ 1 & K_h \leq avg \leq N, \end{cases}$$

  where,
  average queueing delay $D = \sum_{i=0}^{N} \frac{(i+1)V(i)}{\mu} p_a(i)$,
  $\alpha = \frac{2*(1-\gamma)}{K_h - K_l}$, $\beta = \frac{2*\gamma}{K_h - K_l}$, $avg = (1-w) * avg + w_q$, and
  γ = mode selector for adjusting drop function slopes.

Advantages

- Compared with RED and its variants, this variant of RED shows marked improved performance with respect to queuing delay, average queue size and packet drop. Its results are compared with similar network configuration as mentioned in [19] for both

heavy and light load.

Limitations

- As this variant is compared with [19], so it, too, carries the same limitations as [19].

2.1.9 Self-Configuring RED Gateway

As mentioned in [19], the self-configuring RED gateway [15] depends on five parameters. They are min_{th}, max_{th}, max_p, w_q and $f(t)$, packet drop probability function.

With the exception of DSRED, however, in all of the variants, the parameter values are set and remain constant throughout the entire simulation, no matter how the traffic load changes. Although a few variants have managed to demonstrate a high degree of fairness, high throughput and scalability with constant parameters, it is clear that adjusting a few of these parameters dynamically for different network traffic and load would lead to better attainment of the goals mentioned above. This paper proposes a method to dynamically change the parameters in order to increase link utilization and decrease packet loss.

The algorithm implements the dynamic adjustments by the Multiplicative Increase Multiplicative Decrease (MIMD) scheme.

Characteristics

- Three extra variables are defined. They are α, β and status.
- The algorithm follows.

  After every Q_{avg} update:

  If ($min_{th} < Q_{avg} < max_{th}$),

  status = between;

  If ($Q_{avg} < min_{th}$ and (status != below),

  status = below, and

  $max_p = max_p/\alpha$; and

  if ($Q_{avg} > max_{th}$ and (status != above),

  status = above, and

  $max_p = max_p * \beta$.

- The adaptive RED is tested against two max_p values of static RED (conservative detection: $max_p = 0.016$; aggressive detection: $max_p = 0.250$) with number of sources switching between 8 and 32.

It is seen that in case of static RED with aggressive detection, the queue remains almost utilized when the number of sources is 32, but remains underutilized when the number of sources is 8.

However, for conservative detection, the queue remains almost utilized for 8 connections but fluctuates between packet loss and underutilization.

In case of adaptive RED, with a starting max_p, the RED queue adapts well, such that it remains almost utilized throughout the simulation, even when the number of sources keeps changing.

- The adaptive RED is also compared with static RED for throughput and packet loss, and its performance lies between the two (conservative and aggressive) static REDs for both the throughput and packet loss.

Advantages

- This gateway studies the two types of congestion avoidance schemes: RED with packet drop and RED with Explicit Congestion Notification (ECN).

Limitations

- As usual, Self Configuring RED Gateway does not deal with non-adaptive sources and different network configurations.

2.1.10 Adaptive RED

Adaptive RED [18] is largely influenced by the above-mentioned, self-configuring RED. A few changes are made to the algorithm as stated above. Two parameters, max_p and w_q, are studied for the sensitivity of RED. RED's performance (Q_{avg} and throughput) is a function of both w_q and max_p. Hence, a careful selection of these parameters is required to achieve good throughput and reasonable, average queue lengths.

This algorithm implements an Additive Increase Multiplicative Decrease (AIMD) scheme.

Characteristics

- max_p is varied to ensure that the Q_{avg} lies approximately midway between min_{th} and max_{th}.
- max_p lies in the range of $[0.01, 0.5]$, that is, in the range of $[1\%, 50\%]$
- The algorithm follows.

  At every *interval* seconds:

If (avg > target and $max_p \leq 0.5$),

$max_p = max_p + \alpha$; or else

if (avg < target and $max_p \geq 0.01$),

$max_p = max_p * \beta$;

where

$interval = 0.5$ sec;

target $= [min_{th} + 0.4 * (max_{th} - min_{th}), min_{th} + 0.6 * (max_{th} - min_{th})]$;

$\alpha = \min(0.01, max_p/4)$; and

$\beta = 0.9$.

- w_q is set as

  $w_q = 1 - e^{-1/C}$. C = link capacity in packets/sec

- Oscillations: Simple RED (with one flow) exhibits extreme variations of the average queue length over the period of simulation. This variation greatly reduces, in case of adaptive RED, as max_p keeps on adjusting to the traffic load and, hence, the oscillations lie in a small range. The results are similar in case of more realistic multiple traffic flows.
- Effects of queue weight: Sensitivity of RED to w_q shows that for both low and high values of w_q the average queue size suffers and, hence, the automatic setting for w_q is prescribed so that an optimum balance can be sought.
- Routing changes: This section studies the effect of sharp changes in the load due to routing changes. The results show that as soon as an output link becomes available, the sources fill in the queue to maximize the link utilization.
- Setting average queue size: The optimal average queue size is a function of both throughput and delay and, hence, there is a relative tradeoff between the two. The tradeoff can be decided by the characteristics of aggregate traffic and its burstiness.

Advantages

- This gateway performs well and achieves high link utilization and throughput with low queuing delays.

Limitations

- This gateway has no results for mixed traffic. A mixture of traffic loads might show interesting results.

2.2 Existing and Proposed Window Adaptation Schemes

In this section we mention the window adaptation scheme of the Internet Engineering Task Force (IETF) recommended cooperative transport protocol, TCP Reno, for the Internet. We also mention various other window adaptation schemes and discuss their relative features.

2.2.1 Window Adaptation Scheme of Transmission Control Protocol (TCP)

A series of congestion collapses, which first started in October of 1986, led to the development of a cooperative transport protocol that would address the issue of congestion control. The proposed Transmission Control Protocol (TCP) [5, 24] has become the dominant transport protocol of today's Internet. The widely-deployed TCP variant, TCP Reno[1], does not generally guarantee a fair or efficient allocation of bandwidth among connections. Moreover, TCP Reno exhibits oscillatory behavior, which results in inefficient use of available bandwidth due to retransmissions. It is for this reason several other variants like NewReno [21], SACK [33], Vegas [10] and models based on utility and game-theoretic frameworks have been proposed.

2.2.2 Pricing-Based Window Adaptation Scheme of TCP

In [27] the authors propose maximization of total user utility in a distributed environment using only the information available at the end hosts. The authors acknowledge the fact that the network and users cannot dynamically exchange pricing information. The authors also show that their algorithm converges to a socially optimum value. However, the authors do not provide any analysis of the tradeoffs between their method and the widely-used TCP Reno.

2.2.3 ECN-Enabled TCP

Since in the real networks, neither the user nor the network can instantaneously exchange pricing information, hence, we have to rely on indirect methods of estimating the network congestion level. In [28] the authors assume a generic form of packet marking where the congestion notification is issued, using explicit congestion notification (ECN) bits. In the ECN framework the data packets contain a field which is used by the user to determine if congestion has occurred. Initially the value in the field is set to 0. If congestion develops at the router, then the router sets the value in the field to 1 instead of dropping the packet. This way the router does not lose goodput and still conveys the

[1] For simplicity, we sometimes refer the window adaptation scheme of TCP Reno as TCP Reno. The same notation applies to other TCP variants and to our proposed Obj-TCP.

end user of congestion. The receiving user acknowledges the packet receipt through ACKs. These ACKs have the same field and have values as shown on the packets. That is, a marked packet will have a marked ACK and a non-marked packet will have a non-marked ACK.

2.2.4 Utility-Based Models

In [25] the authors formulate the TCP sending rate as a utility-based problem where they assume that the data packets contain a field R, and the ACK packets contain a field E. The value in R lets the router know the current sending rate of that source, and the value in field E lets the source know the total number of congested links on its path. Implementation of this method is clearly not possible within the currently proposed ECN architecture.

In [31] the authors have formulated the source flow rate problem as a primal maximization problem and the network optimization problem as the dual minimization problem. The authors perform a good analysis of the synchronous and asynchronous cases. However, their scheme would work only if network resources actively participate in recording the total rate going through them and convey the information to the sources, which then, accordingly, would adjust their prices. Use of such a scheme is not possible with today's limited support (only ECN) of network resources for such an information transfer.

In [6] the authors consider a fluid model of the network and set up the utility and pricing functions as a logarithmic and linear function of the flow rate. The pricing function also linearly depends on the total queueing delay on its path. However, the authors assume that the total queueing delay on its path can be considered independent of the individual flow rate of the source. This might not be totally accurate when the rest of the users make a similar assumption about total queueing delay.

2.2.5 Game Theory-Based Models

In [3] and [7] the authors model the TCP behavior on each end node as a noncooperative game where each node attempts to maximize its throughput by modifying its congestion control behavior.

Noncooperative games are decentralized control problems and determination of Nash equilibrium in such a scenario requires coordination among the other noncooperative users. This would require that all users are aware of each others' strategies and, hence, make their decisions accordingly. This situation is highly impossible in real networks and, hence, cannot be implemented.

Chapter 3

Analysis of Robust Active Queue Management Schemes

3.1 Features of Robust AQM Schemes

Most of the proposed AQM schemes require frequent tuning of various parameters for optimal performance. This tuning of parameters depends on several factors such as network configuration, resources available and the nature and intensity of traffic expected to pass through the gateway. It is usually desirable for the routers to display steady performance in spite of changes in network conditions and adjust the parameters themselves without much assistance from a network operator. Robust AQM schemes take into account such fluctuations and require very little input from a network operator. With the above arguments and the discussions in [8], we are of the opinion that robust AQM schemes should take precedence over strictly optimal ones.

A robust AQM technique should be capable of switching from one traffic environment to another without sustained loss of performance. A robust AQM technique should be capable of handling traffic environments such as [8]:

- small number of high-bandwidth flows;
- large number of low-bandwidth flows;
- heterogeneous round trip times;
- heterogeneous bandwidths due to upstream constraints (broadband vs. modem access rate constraints);
- heterogeneous flow durations; and

- non-stationary flow duration distribution.

Figure 3.1 illustrates the difference in performance between a hypothetical DropTail scheme and an AQM scheme, AQM1, under the same network traffic conditions. It also shows the difference in performance between the same AQM scheme but under different traffic conditions, AQM1 and AQM2. For a given queueing delay, an AQM scheme should perform better than a DropTail scheme.

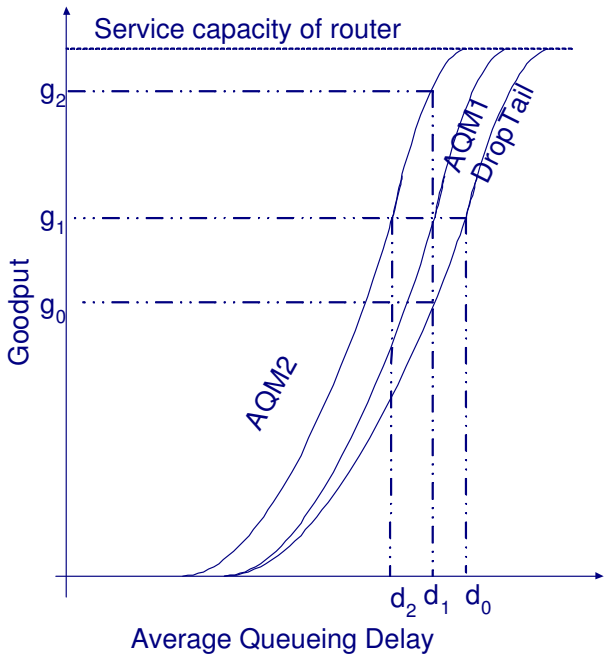


Figure 3.1: The graph compares the behavior of an AQM scheme, AQM1, with a DropTail scheme under similar traffic conditions. Similarly, it compares the behavior between the same AQM scheme but under different traffic conditions, AQM1 and AQM2. Given a delay, an AQM1 scheme should produce a better goodput than a DropTail scheme. Similarly, given a goodput, an AQM1 scheme should have a smaller delay than a DropTail scheme. (Source: Authors' impression of the scheme)

3.2 Characteristics and Prediction of TCP Traffic at the Router

3.2.1 Adaptive, Bursty and Chaotic Nature of TCP Traffic

It has been observed in [16] and [32] that TCP traffic can be interpreted as "pseudo self-similar" in nature, which means that TCP displays self-similarity or *scaling* only on short-time scales. It is the nature of TCP to saturate the outgoing link of a router, and it achieves this saturation by ramping up its packet sending rate until it reaches its maximum window size or experiences congestion, whichever occurs earlier. Similarly, packet losses (or delayed acknowledgements) lead to a decrease in packet sending rate. This window scaling phenomenon reflects the adaptive nature of TCP. Also, TCP traffic shows a bursty nature. A stream of packets arrive from a source, which is then followed by a period of silence. This behavior is quite different from the behavior of UDP traffic where a steady flow of packets is guaranteed by the UDP source. The third characteristic of TCP traffic is that it is chaotic [41]. Although periodicity is induced due to round-trip time and retransmission timeout calculations, in the event of continued packet losses, the exponential back-off behavior of TCP changes this periodicity, making the packet-sending behavior chaotic. Therefore, in order to predict the aggregate behavior of TCP traffic at a router, we have to take into account its adaptive, bursty and chaotic nature.

3.2.2 Motivation for a Predictive AQM Technique

The discussions in Chapters 3.1 and 3.2.1 highlight the fact that the next generation AQM schemes should not only be robust in their performance but also should take into account the pseudo self-similar traffic generated by TCP. The traffic values at the present time contain a significant amount of information about the traffic values in the future. This property of short-range dependence can be used to *predict* future traffic intensity. Combining complete traffic information with rate-based adaptive controllers, it is possible to design AQM schemes that are more robust and perform better than the one implemented at the current time.

3.3 Definition and Quantification of Robustness

Robustness signifies insensitivity against small deviations in the assumptions [23]. In this paper we have quantified AQM robustness as the ability of a router to maintain its performance even under harsh changes in its environment. Referring to Figure 3.1, the *robustness* of an AQM scheme, under two different types of network traffic conditions (AQM1 and AQM2) and with an

average queueing delay of d_1 is:

$$\mathbf{R}_g = \frac{g_2 - g_1}{g_2}. \tag{3.1}$$

With this definition it is clear that more robust schemes have smaller robustness index values. We would also like to emphasize that this value gives only a comparison metric and *does not* give a normalized value of robustness.

3.4 Logical Modules of an AQM Scheme

In this section we dissect the operation of an AQM scheme into two logical modules:

- Measurement module, or *Predictor*; and
- Control module, or *Controller*.

The purpose of predictors is to capture traffic intensity and estimate future arrivals over a prediction interval. The accuracy of prediction depends a great deal on the length of the sampling interval [39]. Our sampling interval was 10ms.

The information collected by the predictors is used by the control module to *adjust the aggressiveness* of packet drops in the next interval.

3.5 Measurement Module or *Predictor*

We focus on the four following predictors:

- Predictive Congestion Control;
- Double-Threshold Moving Window;
- Weighted Double-Threshold Moving Window; and
- Phase Lag.

3.5.1 Predictive Congestion Control (PCC)

The detailed operation of this predictor is mentioned in [37]. In this paper we have used the same notations as mentioned in [37]. The traffic arrival rates are divided into eight intervals based on its past mean (μ) and standard deviation (σ) values[1]. These intervals (or quantized levels) from $l = 1$ to $l = 8$, are, respectively:
$(-\infty, \mu - 3\sigma), [\mu - 3\sigma, \mu - 2\sigma), [\mu - 2\sigma, \mu - \sigma), [\mu - \sigma, \mu), [\mu, \mu + \sigma), [\mu + \sigma, \mu + 2\sigma), [\mu + 2\sigma, \mu + 3\sigma)$

[1]For the purpose of simulation eight divisions were found to be sufficiently granular to capture the traffic rates.

and $(\mu + 3\sigma, \infty)$.

Based on these intervals, an 8×8 conditional probability matrix $\mathbf{M}(l_{t1}, l_{t2}) = P\{X_{t2}|X_{t1}\}$ is created corresponding to each of the eight current quantized levels mapping into the same eight quantized levels into the future. For every slot in the training period (that consists of the sub-slots (l_{t1}, l_{t2})), the corresponding value of $\mathbf{M}(l_{t1}, l_{t2})$ is increased by one. Finally, the values of $\mathbf{M}$ are normalized across rows to give the conditional probability $P\{X_{t2}|X_{t1}\}$.

After the training period the probability matrix $\mathbf{M}$ should be updated at suitable intervals. For our simulations we updated the matrix at the end of every slot.

3.5.2 Double-Threshold Moving Window (DTMW)

This predictor is a type of low-pass but nonlinear finite impulse response (FIR) filter. The details of its operation and its original application are mentioned in [22]. It consists of a moving window of size N where each slot in the window records the traffic rate for that slot. DTMW maintains two thresholds, $T1$ and $T2$. At each measurement interval the arrival traffic is measured and compared with the first threshold $T1$. If the arrival rate equals or exceeds the threshold $T1$ then a bit is set in the moving window corresponding to that slot. A control action for the next slot can then be based on the second threshold $T2$ and the summation of all bits in the N slots. The schematic diagram of a DTMW predictor is redrawn from [22] and shown in Figure 3.2.

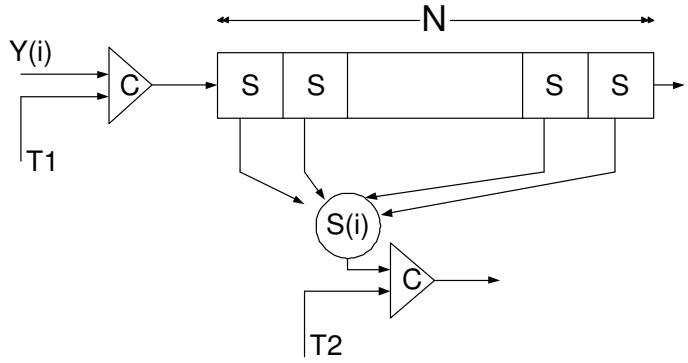


Figure 3.2: Flow diagram of DTMW predictor. (Source: Reproduced from [22])

3.5.3 Weighted Double-Threshold Moving Window (wDTMW)

This is a modified version of the DTMW predictor. Instead of assigning equal weightage to the observations over the N slots in a moving window, higher priority is assigned to more recent observations and lower priority to older ones. This type of measurement captures the continuous occurrence of high traffic rates and is relatively unbiased towards the window size N. Occurrences of high traffic rates at close proximity to each other is an indication that future traffic will follow similar trends. Similarly, for large window sizes, all observations in the past affect the current packet-marking decision. This method of marking may not be optimal, especially for bursty sources which show high correlation to past data. This implies that the window size N should be large enough to capture the correlation of the past data and, at the same time, small enough to exclude data of no importance to the present packet-marking decision. In order to address these problems, observations are assigned weights such that these weights decrease as the observations become older.

The weights of observations are assigned according to a polynomial of binary power. Let n be an integer with values between 1 and N, where the oldest slot in the window has value 1 and the latest slot has value N. The weight polynomial is constructed as:

$$\begin{aligned} \mathbf{P} &= \sum_{i=N}^{1} (2^{i-1} * X_i) \\ &= (2^{N-1} * X_N) + (2^{N-2} * X_{N-1}) + \ldots + (2^0 * X_1) \end{aligned} \tag{3.2}$$

where X_n is bit status of slot n and is defined as:

$$X_n = \begin{cases} 1 & \text{if arrival rate at slot n} \geq \text{T1 and} \\ 0 & \text{otherwise.} \end{cases}$$

The packet-marking decision is taken based on $\mathbf{P}$ and $T2$.

3.5.4 Phase Lag (PL)

This is a representation of an ideal predictor where, instead of *predicting* the future traffic rate, its value is obtained from a *lookahead* into the simulation model. It works as follows: for a phase lag predictor with prediction interval length h, traffic generation is started at a time $t = -h$ but is not fed into the simulation system until at time $t = 0$. In this way a router can *see* the future traffic rates that are going to arrive in the next h intervals. Since such a scheme discounts for

prediction errors, it can be used as a benchmark to compare the performance of other predictors. Although this is not realizable in practice, we use it to set bounds on performance based on exact knowledge of traffic arrivals. The schematic diagram of a Phase Lag predictor is shown in Figure 3.3.

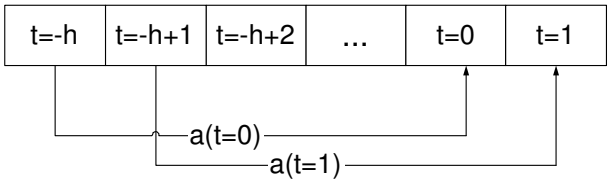


Figure 3.3: Figure of an idealized prediction scheme enforced by a phase lag predictor. (Source: Authors' impression of the scheme.)

3.6 Control Module or *Controller*

We study the following two controllers:

- Estimated Future Average Queue Length; and
- Least Mean Square Fixed Queue Occupancy.

The Least Mean Square Fixed Queue Occupancy (LMSFQO) is an existing rate-based controller, and the details of its operation are cited in [20]. We propose a modification to the EWMA of queue length calculation to achieve a different control action exhibited by Estimated Future Average Queue Length controller (EFAQL). Finally, we design a rate-based controller that fixes the average queue occupancy, as calculated by EWMA scheme, to a certain desired level. The details of each controller are mentioned in the following subsections.

3.6.1 Estimated Future Average Queue Length (EFAQL)

This controller is an extension of the EWMA of queue length. Although it has been used both as a predictor and as a controller in the traditional RED, we implement it here strictly as a controller. It makes a packet-marking decision based on past information of average queue length and estimated future arrival rates. Its control action is similar to the one in [19] and is mentioned

below:

$$p = \begin{cases} 0 & Q_{Avg}(t=n) < Min_{th}, \\ \frac{Q_{Avg}(t=n)-Min_{th}}{Max_{th}-Min_{th}} * Max_p & Min_{th} \leq Q_{Avg}(t=n) < Max_{th}, \text{ and} \\ 1 & Max_{th} < Q_{Avg}(t=n), \end{cases}$$

where

p = drop probability for the next interval, $n+1$,

Max_{th}, Min_{th} = maximum and minimum threshold values for average queue size,

Max_p = maximum drop probability, and

$Q_{Avg}(t=n)$ = average queue size at the interval, n.

The estimated future average queue length is calculated as:

$$\begin{aligned} Q_{Avg}(t=n) &= w1 * Q_{Avg}(t=n-1) + \\ & \quad w2 * Q_{Avg}(t=n+M) + (1-w1-w2) * Q_{Inst}(t=n) \end{aligned} \tag{3.3}$$

In (3.3) $w2$ is the weight attached to the future average queue length obtained from the estimate of future arrival rates in the measurement intervals $(n+1, n+M)$. Future arrival rates are estimated from the predictors. $Q_{avg}(t=n+M)$ is calculated recursively as:

$$\begin{aligned} Q_{Avg}(t=n+M) &= (1-u1) * Q_{Avg}(t=n+M-1) \\ &+ u1 * Q_{Inst}(t=n+M) \end{aligned} \tag{3.4}$$

where, $u1 = 0.002$ and,

$$\begin{aligned} Q_{Inst}(t=n+M) &= Q_{Inst}(t=n+M-1) + \widehat{Ar}(t=n+M) \\ &- Se(t=n+M) - Dr(t=n+M) \end{aligned} \tag{3.5}$$

where, $\widehat{Ar}(t=n+M)$ is the predicted arrival at time $t=n+M$ and $Q_{Inst}(t) := Max(0, Q_{Inst}(t))$. For DTMW and wDTMW, $\widehat{Ar}(t=n+M)$ was calculated as:

$$\widehat{Ar}(t=n+1) = \begin{cases} T1 & \text{if sum of bits} \geq \text{T2, and} \\ \frac{T1}{2} & \text{otherwise.} \end{cases}$$

Equation (3.5) represents buffer dynamics at time $t = n + M$. A special case of (3.3) is when $w2$ becomes equal to zero. For this value it is reduced to the ordinary RED.

3.6.2 Least Mean Square Fixed Queue Occupancy (LMSFQO)

The operation of this controller is to keep the queue occupancy close to a target fraction, Q_{fixed}, of the maximum queue size. This controller does not allow the instantaneous queue occupancy to deviate largely from a predetermined value, and so implementing such controllers helps to achieve low-buffer size variations and low delay jitter. The detailed operation of this controller is described in [20]. The instantaneous queue length at slot $n + 1$ is represented as:

$$Q(t = n + 1) = Q(t = n) + Ar(n + 1) - Se(n + 1) - Dr(n + 1) \tag{3.6}$$

where $Ar(n + 1), Se(n + 1)$ and $Dr(n + 1)$ are the total number of arrived, serviced and dropped packets, respectively, in the slot $n + 1$. The distance, d_{n+1}, of this instantaneous queue length from the target queue occupancy is:

$$d_{n+1} = Q(t = n + 1) - Q_{fixed} \tag{3.7}$$

and the squared sum, J, of this distance over the M future slots is defined as:

$$J = \sum_{i=1}^{M} d_{n+i}^2 = \sum_{i=1}^{M} (Q(t = n + i) - Q_{fixed})^2. \tag{3.8}$$

The aim of the controller is to find a dropping sequence $\{Dr(n + 1), Dr(n + 2), \cdots, Dr(n + M)\}$ for the M future slots that achieves a minimum for J.

In [20], the authors point out that the instantaneous queue length is restricted as $Q(t = n+1) \geq 0$ and so J cannot be differentiated over its domain. Instead they obtain the infimum for J. The procedure of finding the infimum for J to obtain the dropping sequence is mentioned in detail in [20]. Here, we show only the final result of packet dropping probability.

The control action for the next sampling interval is:

$$p = \begin{cases} 0 & Q < Se + Q_{fixed} - \hat{Ar}, \\ \frac{Q+\hat{Ar}-Se-Q_{fixed}}{\hat{Ar}} & Se + Q_{fixed} - \hat{Ar} < Q < Se + Q_{fixed}, \text{ and} \\ 1 & Q > Se + Q_{fixed}. \end{cases}$$

where

p = drop probability for the next interval, $k + 1$,

Q = instantaneous queue size at current interval, k,

Se = service rate at current interval, k, and

$\hat{Ar}$ = estimated arrival rate of next interval, $k + 1$.

Chapter 4

Analysis of Window Adaptation Schemes

In this chapter we describe the characteristics of a generalized window adaptation scheme of a cooperative transport protocol and its widely-deployed implementation, TCP Reno. We also mention the IETF recommended guidelines for proposing new cooperative protocols. Finally, we discuss the need for a more objective implementation of TCP Reno and propose our version of window adaptation for TCP, which we call Objective TCP or Obj-TCP.

4.1 Function of a Window Adaptation Scheme

The function of a window adaptation scheme is to utilize the full bandwidth it requires and, at the same time, to actively participate in reducing network congestion. A window adaptation scheme estimates network congestion after the receipt of acknowledgements (ACKs) and carefully injects its packets into the network in order to avoid congestion. Upon the successful transmission of all its packets, it increases its packet sending rate to its maximum allowed value. On the other hand, upon congestion detection, it immediately backs off by decreasing its sending rate. This behavior ensures that the network congestion is controlled and avoided. A window adaptation scheme estimates the network congestion by a combination of packets losses and packet round-trip times.

For our work we select the window adaptation scheme used by TCP Reno which is the widely-deployed cooperative transport protocol in today's Internet. We compare its behavior with our proposed window adaptation scheme and discuss their relative characteristics with respect to packet re-sent ratio, power and algorithmic complexity. We call the TCP using our scheme, Objective-TCP or Obj-TCP.

4.2 Operation of TCP Reno

The detailed operation of TCP Reno is mentioned in [40]. The pseudo code for TCP Reno is shown in Appendix II. TCP Reno contains four intertwined algorithms: slow start, congestion avoidance, fast retransmit, and fast recovery. The widespread implementation of TCP Reno in the Internet is due to the fact that TCP Reno quickly increases its congestion window to detect network capacity. Its elegant congestion detection scheme allows it to detect real congestion based on the order of acknowledgements (ACKs) it receives. Hence, TCP Reno injects packets into the network without choking its current sending rate even at the receipt of duplicate ACKs.

4.3 Objective-Driven TCP (Obj-TCP)

Obj-TCP measures the network congestion at the end of every transmission instant and then uses these values to estimate the congestion at the next transmission instant. The estimated value of congestion will determine the best window size which will minimize the packet losses for the next sending instant. One way for the Obj-TCP to infer the network congestion is through the ratio of number of marked packets to the number of transmitted packets. In our work we have carried out a black box analysis of the network whereby sources can infer "incomplete" information about network congestion through the number of ECN bits on the ACKs it receives. Retransmission timeout (RTO) is also an indicator of congestion; however, in our work we have modelled Obj-TCP to behave similarly to TCP Reno in case of RTO.

The standards track IETF mechanisms mentioned for all TCP implementations is reproduced in the next chapter. We would like to state that in our work we have not violated the standards track IETF mechanisms proposed for TCP implementation. Rather, we propose a modification to the window updating scheme based on history of packet losses.

4.4 Testing and Evaluation of a New Cooperative Protocol

As shown in Chapter 2, several utility-based, game-theory-based and window-based schemes have been proposed which mention different window adaptation techniques that promise better performance. It has been shown that the flow rate governed by such schemes achieve convergence and global optimum values in the limit. And also, such window adaptation schemes promise enhanced throughput. We observe that in all of these schemes no performance comparison has been done with the TCP, which is the widely-deployed transport protocol; and it still remains to be shown how

these methods fare with respect to the currently-deployed TCP Reno.

Since TCP Reno is the dominant transport protocol of the Internet, hence it becomes necessary that all new TCP-like protocols have to be tested before they can be recommended for use in the Internet. Testing methods serve to compare and evaluate the working of the proposed cooperative protocol with that of existing TCP variants. Detailed performance evaluation of a new cooperative protocol will reveal the protocol's mechanisms, dynamics and interactions with other traffic sharing the network path. We have followed the testing methods mentioned in [4].

The standards track IETF mechanisms mentioned for all TCP implementations are:

- Basic Congestion Control;
- Extensions for High Performance;
- Selective Acknowledgements;
- Delayed Acknowledgements; and
- Nagle Algorithm.

In our work we have emphasized the suitability of our proposed scheme with respect to high performance. We propose a different window calculation algorithm that leads to higher performance compared to TCP Reno. This means Obj-TCP still pursues the basic congestion control mechanism of fast retransmit and fast recovery whenever affected by congestion.

4.5 Calculation of Optimal Window Size

The window updating scheme, at any transmission slot $k+1$, for our proposed Obj-TCP works as follows. We use the following notation:

W_k = Number of packets sent (window size) during transmission slot k;
Wm_k = Number of packets marked during transmission slot k;
W_{max} = Maximum window size;
ρ_k = Packet marked ratio at transmission slot k;
$\widehat{\rho}_{k+1}$ - Estimated packet marked ratio for transmission slot $k+1$;
U_k = Utility at transmission slot k;
$\widehat{U}_{k+1}$ = Estimated net utility for transmission slot $k+1$; and
$\widehat{C}_{k+1}$ = Estimated cost at transmission slot $k+1$.

We consider a simple utility function (mentioned in [6]) and an estimated cost which is a function of the number of estimated marked packets $\widehat{W}m$ in the next transmission slot. The net utility

for the next transmission slot is then given by

$$\widehat{U}_{k+1} = U_{k+1} - \widehat{C}_{k+1} = log(W_{k+1}) - \widehat{W}m_{k+1} = log(W_{k+1}) - \widehat{\rho}_{k+1} * W_{k+1} \qquad \forall\ k \geq 0. \quad (4.1)$$

Prediction of $\widehat{\rho}_{k+1}$ is mentioned in the next subsection. A good approximation of $\widehat{\rho}_{k+1}$ implies that we can find a near optimal window size, W_{k+1}, for the next transmission slot which will maximize $\widehat{U}_{k+1}$. Differentiating (4.1) we obtain

$$W_{k+1} = \frac{1}{\widehat{\rho}_{k+1}} \qquad \widehat{\rho}_{k+1} > 0 \quad (4.2)$$

or for integer values of W_{k+1} we have

$$W_{k+1} \approx \lceil \frac{1}{\widehat{\rho}_{k+1}} \rceil \qquad \widehat{\rho}_{k+1} > 0 \quad (4.3)$$

where $\lceil n \rceil$ denotes the smallest integer greater than or equal to n. Since window size in any slot k is constrained as $1 \leq W_k \leq W_{max}$, ρ_k is constrained as $\varepsilon \leq \rho_k \leq 1$, where $\varepsilon = \frac{1}{W_{max}}$.

4.6 Implementation of Obj-TCP

Obj-TCP maintains a training period and an implementation period. In the training period the Obj-TCP records the ratio of marked packets to build a history in order to estimate the ratio for the next transmission slot. In our experiments we have discovered that the DTMW predictor with a window size of 2 is a good tradeoff between the complexity of the system and the performance increase. More details on the DTMW predictor can be found in [22, 34]. The pseudo-code for the Obj-TCP follows

While (time $\leq$ Training Period) do the following:

1. At the end of every transmission slot, k, get W_k and Wm_k; and
2. Calculate ρ_k.
3. If ($\rho_k \leq T1$)
   $D[index] \leftarrow 0$,

   else
   $D[index] \leftarrow 1$.

4. Set

$\rho[0] \leftarrow \rho[1]$,
$\rho[1] \leftarrow \rho_k$,
$T1 \leftarrow \frac{1}{2} * (\rho[0] + \rho[1])$, and
$index \leftarrow index + 1$.

While (time > Training Period) do the following:

1. At the end of every transmission slot, k, get W_k and Wm_k; and
2. Calculate ρ_k.
3. Perform a moving window operation on D

$D[0] \leftarrow D[1]$.

4. If $(\rho_k \leq T1)$

$D[1] \leftarrow 0$,

else

$D[1] \leftarrow 1$.

5. Set

$\rho[0] \leftarrow \rho[1]$,
$\rho[1] \leftarrow \rho_k$, and
$T1 \leftarrow \frac{1}{2} * (\rho[0] + \rho[1])$.

6. Estimate $\widehat{\rho}_{k+1}$ as:

$$\widehat{\rho}_{k+1} = \begin{cases} Min(\rho[0], \rho[1]) & , (D[0] + D[1]) = 0, \\ \frac{1}{2} * (\rho[0] + \rho[1]) & , (D[0] + D[1]) = 1, and \\ Max(\rho[0], \rho[1]) & , (D[0] + D[1]) = 2. \end{cases}$$

7. The window size W_{k+1} is calculated as:

$$W_{k+1} = \begin{cases} Min(\lceil \frac{1}{\widehat{\rho}_{k+1}} \rceil, W_{max}) & , \rho_{k+1} > 0 \\ \overline{W} & , else \end{cases}$$

where $\overline{W}$ is obtained by the usual TCP Reno window calculation under packet loss.

Chapter 5

Experimental Results and Inferences

In this chapter we provide details of comparative results of the various AQM schemes and of the window adaptation of cooperative transport protocols. We begin by explaining our carefully designed experiments. We simulate a simple network to test our ideas. We model our test traffic on the dominant internet traffic which is heavy-tailed *http* traffic. We also mention how we collect and analyze the simulation data to remove any bias due to warm-up or randomness. In order to give credibility to our results, we also present the 95^{th} percentile confidence intervals for all performance metrics.

In this chapter we have attached plots for selected scenarios of AQM schemes. The rest of the plots of goodput performance and robustness of each combination is shown in Appendix I.

5.1 Test Network Configuration

For our experiments we model a simple network as shown in Figure 5.1. The network consists of 30 sources each sending their packets to the same destination. All sources are connected to their destination through a single router. The output link of this router acts as a bottleneck. The network has the following specifications:
Number of sources - 30;
All links of capacity - 20Mbps;
Router service capacity - 20Mbps; and
Buffer capacity - 100 packets (1 packet = 1000 Bytes).
The OFF period duration and File size distribution are obtained from [13].
OFF period $\sim Pareto(1.21, 2)$seconds;

File size $\sim Pareto(1.21, 2)$*10000 Bytes;

RTT $\sim U(80, 120)$ msec; and

Sampling time - 10msec.

The simulation is carried out on a TCP stack which models the widely-used TCP Reno with delayed

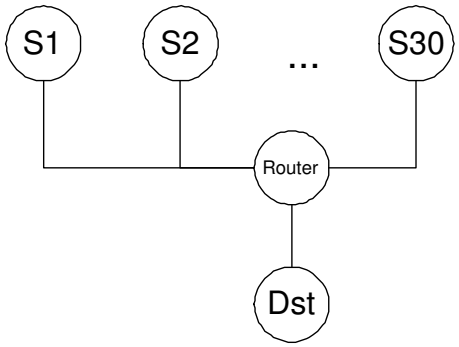


Figure 5.1: The network consists of 30 independent sources connected to the same destination through a single router. This router acts as a bottleneck.

ACKs. It is assumed that ACKs are never lost. The end nodes and the router support ECN bit. The increase in congestion at the router results in packet marking instead of dropping for moderate congestion and packet dropping for heavy congestion. The marked packets are then received by the destination which sends ACKs for these packets with an ECN mark. The sources detect congestion through the ECN marks on the ACKs and change their window size accordingly.

The modelled traffic is http traffic, the dominant network traffic [14]. The document sizes and user "think times" are assumed to be heavy-tailed. At the start of the simulation all of the sources remain in the OFF state which is distributed as mentioned above. After the source switches to the ON state, it receives a file size request which is distributed as mentioned above. During the ON state the sources engage in continuous page transfers. After the file transfer is complete, each source switches back to the OFF state. This cycle continues till the end of the simulation.

For the Obj-TCP the window size calculation algorithm of the above setup was modified to implement the algorithm proposed by us in Chapter 4.6.

5.2 Data Collection and Analysis

The statistics that were collected during the simulation for testing the AQM schemes were *goodput*, *average queueing delay* and *delay jitter*. A robustness graph like the one shown in Figure

3.1 was created by varying the buffer capacity from 100 packets to 1000 packets and recording the corresponding goodput. For each predictor-controller combination, several runs of simulation were conducted in order to remove bias in measurements during the warm-up period and to calculate the 95^{th} percentile confidence intervals during the steady-state period[1].

Similarly, the statistics collected during the simulation for comparison between the two window adaptation schemes were *packet re-sent ratio* and *power* for varying number of sources. As in the case of AQM schemes, in this case also we provide the 95^{th} percentile confidence intervals for each performance metric.

5.3 Statistical Accuracy of Experimental Results

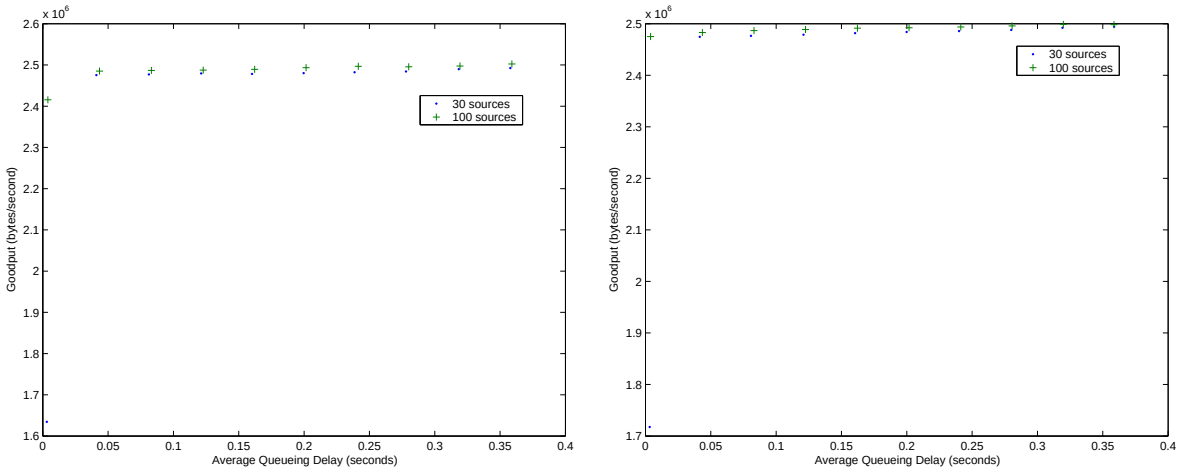
In our experiments we have taken several measures to ensure the consistency and validity of our results. Each result in the following tables and graphs is an average of several runs. In the graphs we have presented the metrics of power and packet re-sent ratio for 30, 60 and 90 sources. Each point on the graph is an average of several observations. In the tables we have provided the average values along with their 95^{th} percentile confidence interval for the metrics of power and packet re-sent ratio for 30 sources.

Detailed analysis of the detection of the warm-up period by the use of the moving window method and the calculation of 95^{th} percentile confidence interval is mentioned in [29].

5.4 Results and Inferences for AQM Schemes

In this section, we present and discuss the results for robustness, delay jitter, goodput, link utilization and algorithmic complexity. Exhaustive results for the various predictor-controller combinations are presented in tabular fashion, which we believe, gives a clearer idea of their relative merits and tradeoffs. To maintain the brevity and to display the Goodput variation with respect to $w2$ and Average queueing delay, we have attached only the plots for schemes based on DTMW. Plots for other combinations display similar trends. Also, we refer to Estimated Future Average Queue Length and Least Mean Square Fixed Queue Occupancy as 'EFAQL' and 'LMSFQO' respectively throughout the rest of this paper. We calculate the robustness of different predictor-controller combinations for all of the scenarios (except 4 and 6) mentioned in Chapter 3.1. Figures 5.2 through 5.4 show the plots obtained for DropTail scheme and schemes based on various controller combinations with DTMW for a small number of high-bandwidth flows and a large number of low-bandwidth

[1]Similarly, several runs were conducted for each value of $w2$ mentioned in (3.3)



(a) DTMW-EFAQL (b) DTMW-LMSFQO

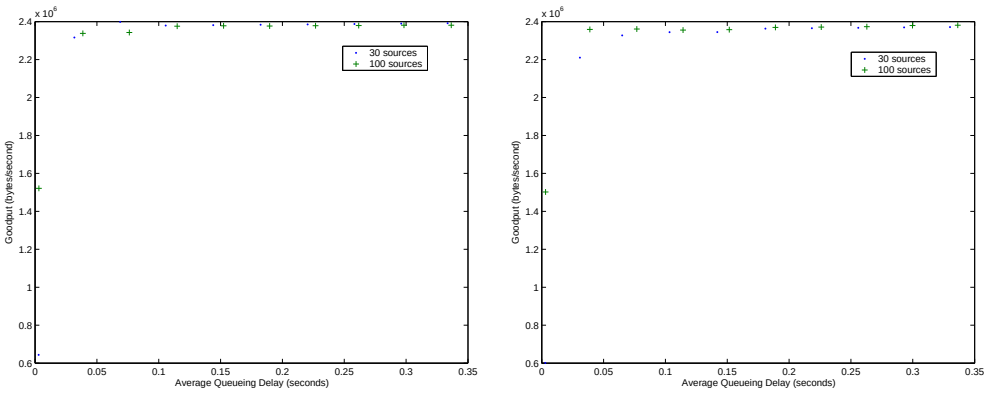
Figure 5.2: Plot of Goodput vs. Average queueing delay of DTMW predictor with (5.2(a)) EFAQL and (5.2(b)) LMSFQO controller for a fixed RTT setup.

flows with a fixed and variable RTT setup. A small number of high-bandwidth flows were simulated by 30 sources with maximum window size of 128 and a large number of low-bandwidth flows were simulated by 100 sources with a maximum window size of 16. The value of robustness between two traffic scenarios, for an average queueing delay of 0.2 seconds[2], is presented in Table 5.1.

Since a lower robustness index indicates better performance, the results show that realistic traffic scenarios, under variable RTT sources, are less robust than their ideal counterparts. This is due to a high degree of variability inherent in real traffic. Table 5.1 shows that AQM schemes with stochastic-based traffic prediction (PCC) are usually more robust when the control is based on EWMA of queue length. Similarly, AQM schemes with traffic prediction based on finite impulse response filters (DTMW) are more robust when the control is based on fixed-queue occupancy. Along the same lines we observe that for most part the robustness results in Table 5.1, between fixed and variable RTT setup for the same combination of predictor and controller, are strongly and positively correlated with goodput values in Tables 5.3 and 5.4 (with the exception of the phase-lag predictor with an EFAQL controller). Also, for the same RTT setup, robustness is usually higher for AQM schemes with higher goodput.

Delay jitter was obtained for parameters displaying the best performance in robustness. The average queue occupancy is between 5 and 15 packets with EFAQL and 90 packets with LMSFQO. LMSFQO help control the delay variations more effectively than EFAQL, which is evident from the

[2]In some cases the simulation results did not go beyond 0.2 seconds due to bounds in programming; in those cases robustness index values for 0.1 seconds was quoted



(a) DTMW-EFAQL (b) DTMW-LMSFQO

Figure 5.3: Plot of Goodput vs. Average queueing delay of DTMW predictor with (5.3(a)) EFAQL and (5.3(b)) LMSFQO controller for a variable RTT setup.

Table 5.1: Robustness, x10^{-3}, for a fixed RTT setup (in parentheses, results for a variable RTT setup). The robustness of a DropTail scheme is 4x10^{-3} (8x10^{-3}).

(Robustness)	EFAQL	LMSFQO
PCC	1.16 (83)	18.62 (140)
DTMW	5.32 (2.86)	3.27 (2.56)
mDTMW	5.32 (9.12)	3.23 (4.1)
PL	5.75 (0.388)	6.67 (30.3)

fact that even with high average queue occupancy, LMSFQO displays comparable delay jitter values. This can be explained by the fact that the maximum waiting time in the queue in bounded if the controller is LMSFQO, which results in smaller delay variations. Whereas, the EFAQL controller allows the instantaneous buffer occupancy to vary widely resulting in degraded performance of delay jitter. The results for different predictor-controller combinations are presented in Table 5.2. Delay jitter values for RED and DropTail (in units of 10^{-6}) are 85.3 $\pm$ 9548.5$\times 10^{-6}$ and 138.1 $\pm$ 7934.8$\times 10^{-6}$ seconds, respectively.

The goodput obtained for different predictor-controller combinations, for a fixed RTT of 100ms, is presented in Table 5.3; and for variable RTTs, the goodput is presented in Table 5.4. The first row lists all predictors, and the first column lists all controllers. Goodput of RED and DropTail, for a fixed RTT setup, is 2 469 668 $\pm$ 3700 bytes/sec and 2 428 928 $\pm$ 4480 bytes/sec, respectively. Similar values for RED and DropTail for a variable RTT setup are 1 718 002 $\pm$ 42040 bytes/sec and 1 652 542 $\pm$ 10820 bytes/sec, respectively. The values in parentheses show percentage improvement over RED. EFAQL with $w2 = 0$ becomes RED. The graphs corresponding to these

Table 5.2: Delay jitter, in units of 10^{-6} seconds, for various predictor-controller combinations for a fixed RTT setup.

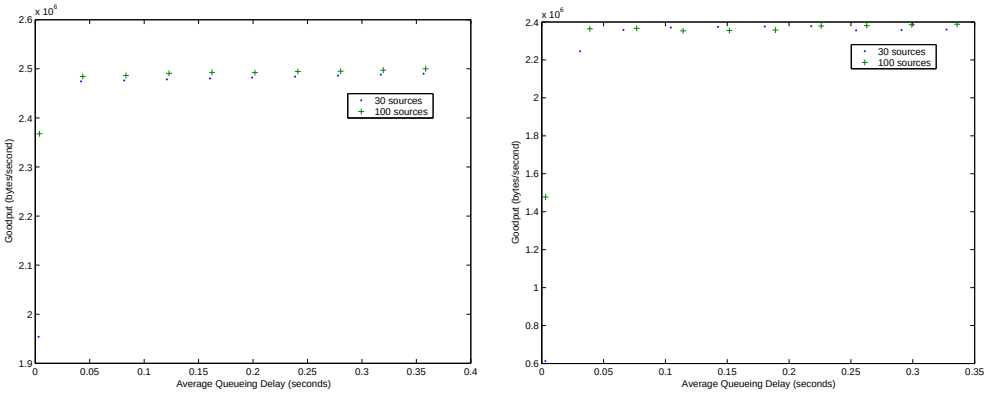
(Delay jitter)	EFAQL	LMSFQO
PCC	$21.83 \pm 13.37 \times 10^{-6}$	$56.4 \pm 286.18 \times 10^{-6}$
DTMW	$41.51 \pm 548.08 \times 10^{-6}$	$36.9 \pm 1606.27 \times 10^{-6}$
wDTMW	$41.63 \pm 1207.6 \times 10^{-6}$	$57.5 \pm 675.04 \times 10^{-6}$
PL	$53.76 \pm 783.13 \times 10^{-6}$	$95.0 \pm 3349.7 \times 10^{-6}$

Table 5.3: Comparison of goodput, in units of bytes/sec, for a fixed RTT setup. The first term in each cell shows goodput in excess of 2 420 000; that is, for PCC-EFAQL combination, the goodput is (2 420 000 + 50 870) = 2 470 870 bytes/sec.

(Goodput)	EFAQL	LMSFQO
PCC	50 870 ± 5 140 (0.05)	20 776 ± 2 657.6 (-1.17)
DTMW	59 530 ± 1 359.2 (0.4)	58 390 ± 1 317.8 (0.35)
wDTMW	57 684 ± 1 800 (0.32)	56 230 ± 2 420 (0.27)
PL	44 367 ± 4 260 (-0.21)	180 890 ± 4 340 (5.31)

Table 5.4: Comparison of goodput, in units of bytes/sec, for a variable RTT setup. The first term in each cell shows goodput in excess of 1 400 000; that is, for PCC-EFAQL combination, the goodput is (1 400 000 + 659 596) = 2 059 596 bytes/sec.

(Goodput)	EFAQL	LMSFQO
PCC	659 596 ± 43 320 (19.8)	148 316 ± 32 000 (-9.88)
DTMW	324 494 ± 37 440 (0.38)	338 844 ± 50 940 (1.21)
wDTMW	336 006 ± 33 800 (1.05)	333 726 ± 34 580 (0.92)
PL	82 020 ± 4 588 (-13.74)	258 726 ± 33 660 (-3.45)



(a) DropTail - fixed RTT

(b) DropTail - variable RTT

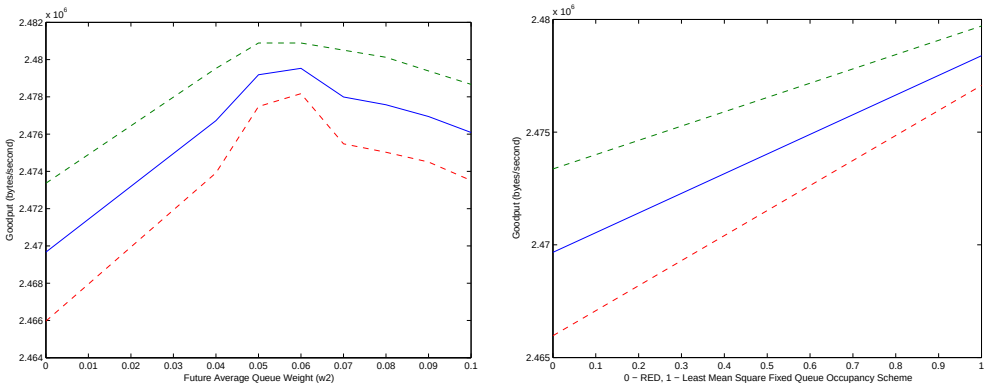
Figure 5.4: Plot of Goodput vs. Average queueing delay of DropTail for (5.4(a)) fixed and (5.4(b)) variable RTT setup.

tables, for the DTMW predictor, are shown in Figures 5.5 and 5.6. As stated earlier, the results of goodput are presented here solely to support our assertion that the robustness of AQM schemes is not achieved at the expense of degraded goodput. The results of goodput for various predictor-controller combinations show that the performance is usually better when traffic characteristics are exploited. However, the percentage improvement of goodput over RED, with PAQM schemes is mild in most of the cases. Moreover, Figures 5.5 and 5.6 show that although there is gain in goodput with increasing values of $w2$ in (3.3), the goodput is quite sensitive to the weight of predicted future arrivals. These observations impose a bound on the optimistic performance of an AQM scheme since they expose the limitations in greatly improving the goodput of AQM schemes even if the underlying traffic structure is exploited to predict the future arrivals. We have used the minimum and maximum threshold values and buffer size as mentioned in [20]. Although we have been conservative in using a relatively small buffer size of 100 packets, we know that for larger buffer sizes the results are expected to be better.

We also present the results of link utilization for various combinations of predictors and controllers. We measure the link utilization values to assess the efficiency of robust AQM schemes. Link utilization is defined as:

$$link\_utilization = \frac{\frac{total\_bytes\_transmitted}{elapsed\_time}}{link\_capacity}. \tag{5.1}$$

Equation (5.1), which gives the rate of data transfer as a percentage of link capacity, gives an indication of the amount of drops that are triggered. Packet size is an important factor in determining



(a) DTMW-EFAQL (b) DTMW-LMSFQO

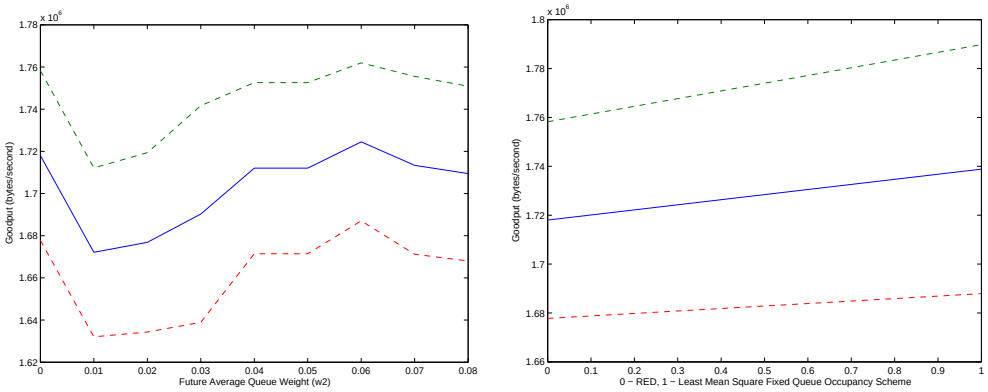
Figure 5.5: Plot of Goodput vs. $w2$ of DTMW predictor with (5.5(a)) EFAQL and (5.5(b)) LMSFQO (at 1) controller for a fixed RTT setup. The dotted lines represent the 95% confidence interval of the solid line.

Table 5.5: Link utilization values, in percent, for a fixed RTT setup (in parentheses results for a variable RTT setup).

(% Utilization)	EFAQL	LMSFQO
PCC	98.8 (82.7)	95.4 (62.0)
DTMW	99.1 (69.0)	99.1 (70.0)
wDTMW	99.1 (69.4)	99.0 (69.3)
PL	98.4 (59.3)	98.5 (62.0)

link utilization. Small packet sizes (64 bytes) will produce low utilization values since high processor overhead is associated with examining many packets [2]. The utilization values for RED and Predictive AQM schemes are mentioned in Table 5.5. These utilization values are for a buffer size of 100 packets. Link utilization values for a DropTail scheme and RED are 97.2% (66.1%) and 98.3% (68.1%), respectively. The result in parentheses is utilization value for a variable RTT setup, and the value outside parentheses is for a fixed RTT setup. Referring to Table 5.5, it can be seen that the utilization values for Predictive AQM schemes are almost always greater than either RED or DropTail for any RTT setup. Since the measurement of utilization does not make any assumption about the configuration of network and the end hosts, this performance metric can be used quite reliably in calculating the efficiency of an AQM scheme.

To conclude our analysis of the robust AQM schemes and to impart an objective approach to our findings, we have calculated the algorithmic complexity of the AQM schemes for the various combinations of predictors and controllers. The algorithmic complexity of DropTail scheme or RED is $O(1)$. This means that they are of constant order and, hence, their execution time is independent



(a) DTMW-EFAQL (b) DTMW-LMSFQO

Figure 5.6: Plot of Goodput vs. $w2$ of DTMW predictor with (5.6(a)) EFAQL and (5.6(b)) LMSFQO (at 1) controller for a variable RTT scheme. The dotted lines represent the 95% confidence interval of the solid line.

Table 5.6: Algorithmic complexity of the AQM schemes.

(Complexity)	EFAQL	LMSFQO
PCC	$O(N)$	$O(N)$
DTMW	$O(N)$	$O(N)$
wDTMW	$O(N)$	$O(N)$
PL	$O(N)$	$O(N)$

of any other system parameter.

Table 5.6 shows the combined complexity for the different combinations of predictors and controllers. Interestingly, the complexity of all predictors is the same and is of order $O(N)$, and the complexity of all controllers is the same and is of order $O(1)$. The combined complexity of any predictor controller combination is $O(N+1)$ or more precisely $O(N)$. The complexity calculation of each module is mentioned in detail in Appendix II.

5.5 Results and Inferences for Window Adaptation Schemes

For a comparative analysis between the window adaptation schemes of TCP Reno and Obj-TCP, we use the performance metrics of packet re-sent ratio, power and computational complexity of algorithms. The definition and importance of each of these metrics is explained in the following subsections. We evaluate the performance for three scenarios where three different AQM schemes

were implemented at the router. The AQM schemes we select are TailDrop (DT), Random Early Detection (RED) and Double-Threshold Moving Window-Least Mean Square Fixed Queue Occupancy (DTMW-LMSFQO). In [34] we had shown that the DTMW-LMSFQO combination of predictor and controller exhibits the highest value of robustness among all AQM schemes. In consistence with our previous work in this thesis, we select the DTMW-LMSFQO combination at the router to study the performance of Obj-TCP under robust network conditions. Also, in the following subsections we present graphs of selective scenarios. The remaining graphs are shown in Appendix I.

Packet re-sent ratio is a performance metric which we define to evaluate the efficiency of a window adaptation scheme in terms of bandwidth reuse. We define packet re-sent ratio as the ratio of the number of retransmitted packets to the total number of packets transmitted by the source.

$$\text{Packet Re-sent Ratio} = \frac{Total\ Retransmitted\ Packets}{Total\ Transmitted\ Packets}$$

The results in Table 5.7 show that Obj-TCP achieves a reduction in the number of retransmitted packets as compared to TCP Reno. Moreover, the reduction is pronounced when the AQM scheme implemented at the router is very robust. Lower packet re-sent ratio implies that the Obj-TCP uses relatively less bandwidth and injects a relatively smaller number of packets into the network, a possible use for security applications that favor less packet losses. Figure 5.7 shows the packet re-sent ratio for TCP Reno and Obj-TCP for a DTMW-LMSFQO AQM scheme at the router for increasing load. The packet re-sent ratio increases as the number of competing flows increase. However, it is interesting to note that the packet re-sent ratio for an Obj-TCP always remains better than TCP Reno, even for increasing load. Similar results for TCP Reno and Obj-TCP for RED and TailDrop AQM schemes are shown in Figures 7.5(a) and 7.5(b) in Appendix I.

Table 5.7: Packet Re-sent Ratio, in percent, for TCP Reno and Obj-TCP under various AQM schemes implemented at the router.

(Packet Re-sent Ratio, %)	TCP Reno	Objective TCP
TailDrop	9.9 ± 3.4	7.8 ± 0.2
RED	4.0 ± 1.6	1.7 ± 1.0
DTMW-LMSFQO	3.7 ± 1.2	0.9 ± 0.5

Power is a standard performance metric which can be used by the end nodes to evaluate the efficiency of their algorithms to transmit a file in minimum time under general network operating conditions [26]. Higher power of a window adaptation scheme means that the scheme is able to send its entire file in less overall time including the queueing delays and retransmission timeouts.

$$\text{Power} = \frac{Throughput}{Total\ Time\ Taken}$$

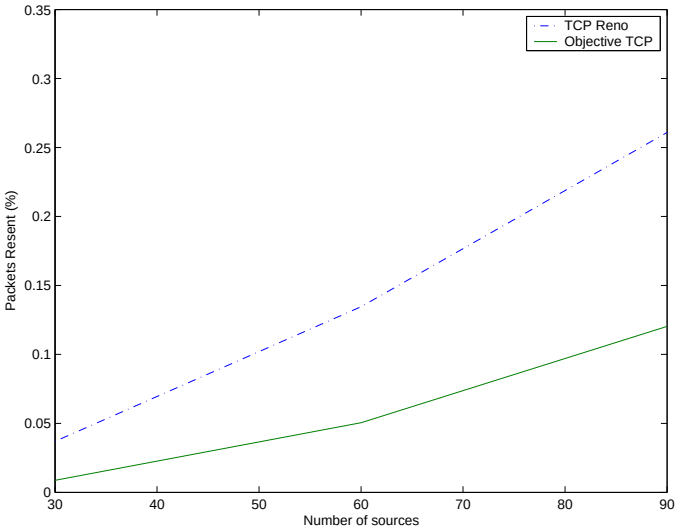


Figure 5.7: This figure shows the dependence of packet re-sent ratio on the number of sources for TCP Reno and Obj-TCP implemented on the end nodes and DTMW-LMSFQO active queue management scheme implemented at the routers. The figure illustrates that Obj-TCP clearly exhibits a lower packet re-sent ratio than TCP Reno for any number of sources.

Table 5.8 shows the difference in power between TCP Reno and Obj-TCP for various AQM schemes. The results reveal that, for any AQM scheme, Obj-TCP always displays a higher power than TCP Reno. Moreover, Obj-TCP shows an increase in power roughly by a factor of 1.28. Figure 5.8 shows the difference in power observed between TCP Reno and Obj-TCP for a DTMW-LMSFQO AQM scheme at the router for increasing load. For increasing competing flows at the router, the transmission time for each source increases resulting in a decrease in power. Similar results for TCP Reno and Obj-TCP for RED and TailDrop AQM schemes are shown in Figures 7.5(c) and 7.5(d) in Appendix I.

Table 5.8: Power, in bytes/second^2, for TCP Reno and Obj-TCP under various AQM schemes implemented at the router.

(Power, Bytes/Second^2)	TCP Reno	Objective TCP
TailDrop	1051 $\pm$ 51	1323 $\pm$ 90
RED	1194 $\pm$ 72	1515 $\pm$ 88
DTMW-LMSFQO	1370 $\pm$ 54	1600 $\pm$ 91

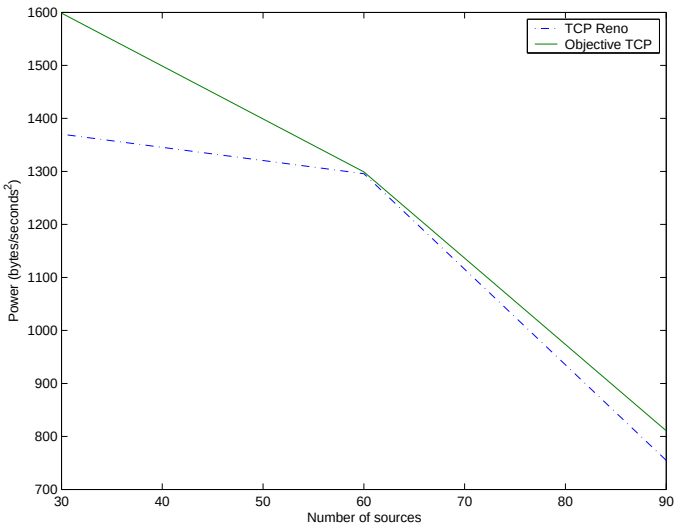


Figure 5.8: This figure shows the relationship between power and the number of sources for TCP Reno and Obj-TCP implemented on the end nodes and DTMW-LMSFQO active queue management scheme implemented at the routers. The figure illustrates that Obj-TCP clearly exhibits a higher power than TCP Reno for any number of sources.

For an objective evaluation of TCP Reno and Obj-TCP algorithms, we calculate and compare their algorithmic complexity. We carry out the complexity analysis as mentioned in [12]. We show that by a simple first-order increase in the complexity of TCP Reno's window calculation algorithm, we get Obj-TCP which displays an increase in performance over TCP Reno. Table 5.9 shows that TCP Reno has a complexity of constant order and Obj-TCP's complexity is proportional to the size of the DTMW predictor. In effect, better performance of TCP entails a relatively more complex congestion detection algorithm. Selection of TCP Reno or Obj-TCP, or any utility-based model, is a tradeoff between performance and complexity.

Table 5.9: Algorithmic complexity of window adaptation schemes.

	Complexity
TCP Reno	$O(1)$
Objective TCP	$O(N)$

Chapter 6

Summary and Future Work

In this chapter we provide a brief summary of the work done in this thesis and future possibilities for research.

6.1 Summary

Our efforts in this work can be divided into six parts: In the first part we have analyzed AQM schemes as a combination of a measurement module and a control module. This method of analysis allows us to carefully observe the effect of each module on the performance of the AQM scheme. For each of the modules we implemented different predictors and controllers and compared the performance of an AQM scheme for various combinations. In our simulations we saw that *careful* control decisions based on prediction of future packet arrivals always lead to an increase in performance.

To summarize our simulation results for the AQM schemes:

- With a proper selection of the weight of future observations, Predictive AQM schemes achieve better delay bounds than the traditional (non-PAQM) ones.
- AQM schemes with stochastic-based traffic prediction (PCC) are usually more robust when the control is based on EWMA of queue length. Similarly, AQM schemes with traffic prediction based on finite impulse response filters (DTMW) are more robust when the control is based on fixed queue occupancy.
- More robust AQM schemes usually have higher goodput for comparable average queueing delay values. We also believe that Explicit Congestion Notification (ECN) based packet marking, rather than packet dropping, would help boost the goodput performance of an

AQM scheme.

- Excessive dependence on predicted future arrivals *always* results in deterioration of performance. This sets bound on the optimistic performance of AQM schemes even using prediction.
- It is inappropriate to evaluate an AQM scheme based only on the goodput performance, which is highly sensitive to the duration of flows (*long-lived* ftp versus *short-lived* http). Robustness, on the other hand, is relatively independent of the traffic flow structure and so can be used reliably for AQM performance evaluation.
- Delay jitter is another important yardstick of performance. Large delay jitter values imply large queue length variations which renders the AQM scheme less robust to accommodate *delay-sensitive* traffic like telnet. Goodput and link utilization were used strictly as additional measures of performance.
- Predictive AQM schemes, for most cases, display higher link utilization values than the traditional ones.
- Predictive AQM schemes clearly show an improvement in performance over nonpredictive ones. This improvement in performance comes in the form of increased algorithmic complexity.

It is known that the precision of prediction degrades as the prediction interval increases. For the same prediction interval, Figures 5.5 and 5.6 show that even though the prediction of future packet arrivals helps to achieve better performance, it performs worse than RED when the current packet marking decisions are too heavily relied upon for future prediction.

In the second part we studied the effects of the fact that the traffic generated by TCP sources is correlated in nature. Dependence of future traffic measurements on current value is a result of (positively) correlated traffic, and we have used this property of strong and even short-range dependence to predict future traffic intensity. We have also suggested ways to exploit the short-range (if not "strong") dependence using different predictors.

In the third part we emphasized the relevance of using robustness and delay jitter as reliable performance metrics of an AQM scheme rather than goodput. In this regard, we have defined our idea of robustness. We have concentrated our attention on the performance of robust AQM schemes that exploited the strong correlation that exists in network traffic. We have shown that, with our definition of robustness, a positive correlation exists between robustness, goodput and link utilization.

We have considered delay jitter as another important yardstick of performance. Large delay jitter values imply large queue length variations which renders the AQM scheme not so robust to accommodate *delay-sensitive* traffic like telnet. We have used goodput, link utilization and

algorithmic complexity strictly as additional measures of performance. We have used link utilization to evaluate the AQM scheme efficiency as it remains independent of the network configuration and, hence, a reliable performance metric. We have measured the link utilization values of predictive schemes over non-predictive ones. Our results indicate that predictive schemes usually have higher link utilization values than the nonpredictive schemes. Algorithmic complexity of AQM schemes allowed us to compare the relative performance and the execution cost of an algorithm. Selection of an algorithm is a tradeoff between its complexity and performance. We strongly support the use of robustness and delay jitter as performance metrics as they tend to provide better evaluation of an AQM scheme. Extensive simulations over varied traffic environments in this paper have supported our propositions.

In the fourth part we concentrated on the study of window adaptation schemes of cooperative transport protocols running on end nodes. We observed that much work has been done in the area of developing new window adaptation algorithms to increase the throughput of TCP. In this part we mentioned in detail the IETF recommended guidelines to be followed before proposing a *new* window adaptation scheme for a cooperative transport protocol. We argued that a new window adaptation scheme not only has to conform to the IETF recommended guidelines, but also has to be compared with other existing protocols for its complexity. Complexity is an important issue in proposing a new window adaptation scheme as the selection of any utility-based model over the existing TCP is a tradeoff between their performance and their complexity. We noticed that most of the proposed schemes do not show the complexity tradeoff of their proposed algorithms.

In the fifth part we proposed a way to estimate the congestion level for the real networks only through the ECN bit marks on the ACKs. Our belief that the ratio of marked packets to the transmitted packets (packet mark ratio) is an indicator of congestion is the basis for our window adaptation scheme. We used a DTMW predictor to record and "predict" the packet mark ratio. We then used an existing utility model and the estimated value of packet mark ratio to estimate the optimum window size for the next transmission to reduce the number of re-sent packets. Unlike other schemes, where elaborate congestion level estimations have been proposed, we estimated the network congestion level only through ECN bit marks and, hence, we called our scheme as congestion level estimation under "incomplete" information. We proposed an Objective driven TCP whose purpose was to reduce the number of marked packets. We then compared its performance to TCP Reno.

In the sixth part we combined our observations of AQM scheme analysis and window adaptation analysis to produce results for a combination of robust AQM schemes at the network level and efficient window adaptation scheme at the end node level.

Our results confirm our view that the analysis of coupled network components, routers and end nodes, can be done independent of each other.

6.2 Future Work

Extension to our work can be, once again, broadly categorized under two areas: one involving the AQM schemes and the other involving the window adaptation scheme of cooperative transport protocols.

6.2.1 Future Areas of Research in AQM Schemes

Although much work has been done in the area of AQM schemes, the selection of one AQM scheme over the other is still an open question. In this regard we mention some interesting research areas that are worth exploring.

- Predicting web traffic by building their regression models instead of predicting by heuristic methods. Such regression models will contain the dependence structure of the various web traffic that comprises the aggregate network traffic which can then be used to make more precise judgement about packet mark or drop.
- Studying the effect of round trip times (RTT) on queue dynamics. By isolating the network dependencies, that is interactions due to other users, a correlation can be built between the structure of RTT for one user and the queue length variation due to it. Such a knowledge will allow the AQM schemes to adjust their aggressiveness (packet mark/drop rate) in a more scientific manner.
- Instead of a general network topology, effort should be expended to analyze a specific topology. For example studying a BellSouth network or the NC State campus network would reveal the precise traffic characteristics of the users (like long file transfers or short SMS messages) and allow us to investigate the important metrics related to such traffic. This, in turn, would facilitate the design of good AQM schemes that take into account such traffic characteristics.

6.2.2 Future Area of Research in Window Adaptation Schemes

All cooperative protocols implement a delay feedback loop to adjust their window sizes depending on input from the network. Instead of adjusting their window size by a heuristic algorithm they can employ stochastic feedback control under disturbance to determine the optimal window size for the next transmission depending on the response from the network. To exploit the stochastic control, the interactions of the other users have to be modelled as disturbances while keeping the objective to optimize some cost by a proper selection of window sizes.

Chapter 7

Appendix I: Individual Performance Graphs

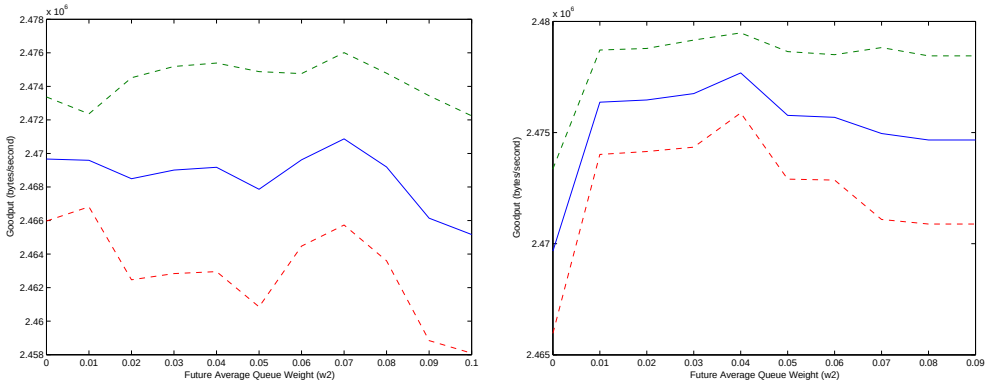
In Chapter 5.4 we present the graphs of Goodput Performance against Average Queueing Delay and against $w2$ the weight of future average queue length for a DTMW predictor. In this chapter we present the remaining performance graphs of the various other predictor controller combinations.

The plots from 7.1(a) to 7.1(c) have estimated future average queue length controller, and the plots from 7.1(d) to 7.1(f) have least mean square fixed queue occupancy controller (at 1). The dotted lines represent the 95% confidence interval of the solid line.

The plots from 7.2(a) to 7.2(c) have estimated future average queue length controller, and the plots from 7.2(d) to 7.2(f) have least mean square fixed queue occupancy controller (at 1). The dotted lines represent the 95% confidence interval of the solid line.

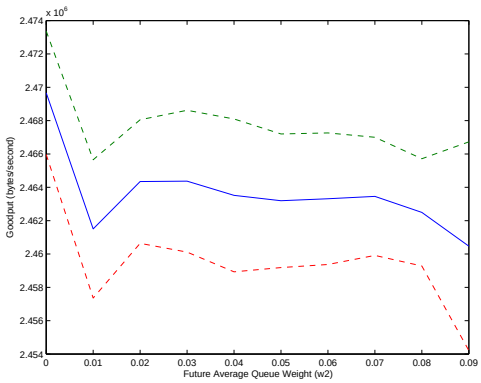
The plots from 7.3(a) to 7.3(c) have estimated future average queue length controller, and the plots from 7.3(d) to 7.3(f) have least mean square fixed queue occupancy controller. The traffic environments were: a small number of high-bandwidth sources and a large number of low-bandwidth sources. The sources had fixed RTTs.

The plots from 7.4(a) to 7.4(c) have estimated future average queue length controller, and the plots from 7.4(d) to 7.4(f) have least mean square fixed queue occupancy controller. The traffic environments were: a small number of high-bandwidth sources and a large number of low-bandwidth sources. The sources have variable RTTs.

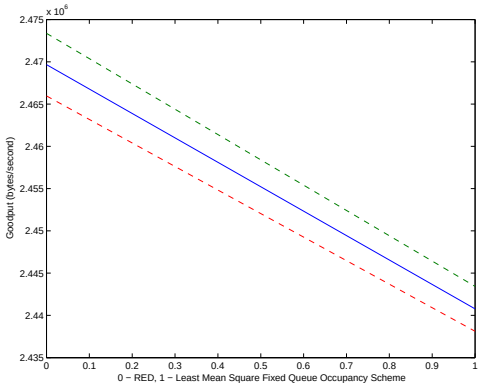


(a) Predictive Congestion Control

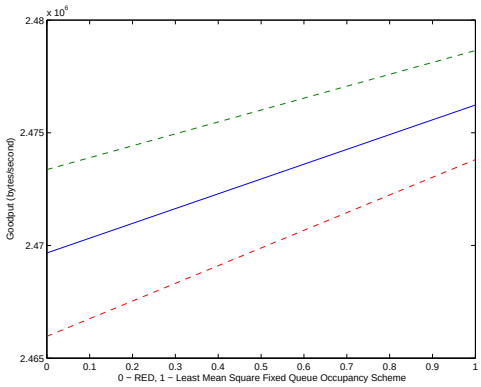
(b) mDTMW



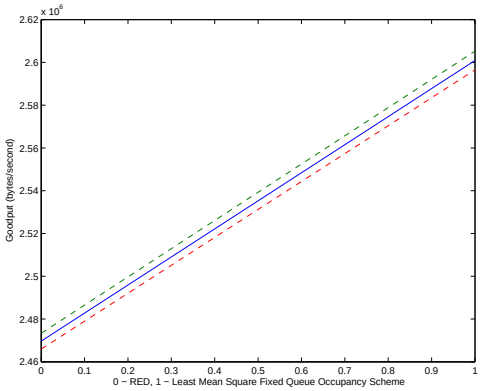
(c) Phase Lag



(d) Predictive Congestion Control

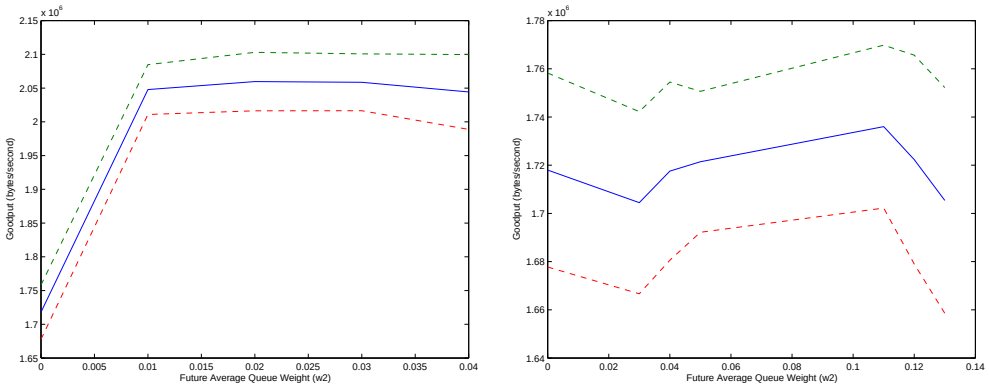


(e) mDTMW



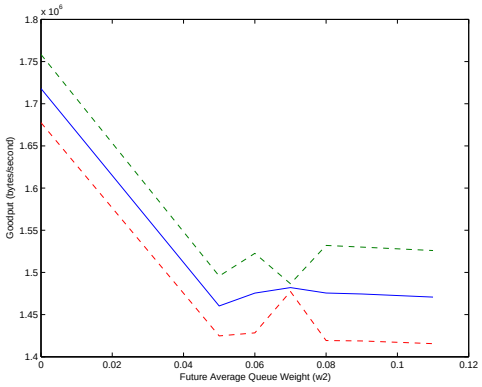
(f) Phase Lag

Figure 7.1: Goodput performance of an AQM scheme for increasing weights of the three predictors under a fixed RTT scheme.

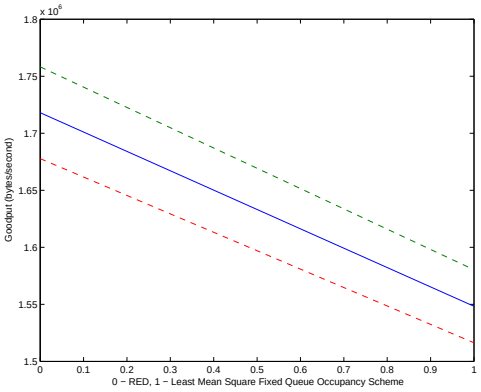


(a) Predictive Congestion Control

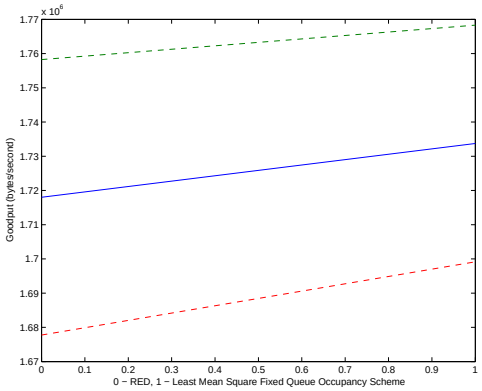
(b) mDTMW



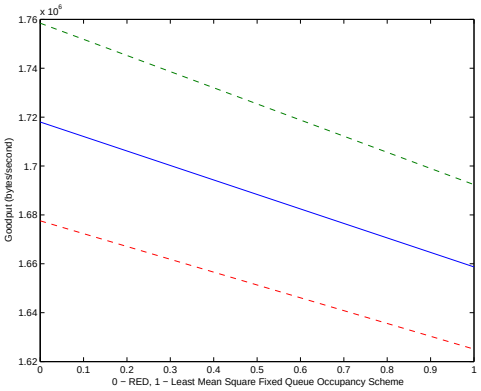
(c) Phase Lag



(d) Predictive Congestion Control

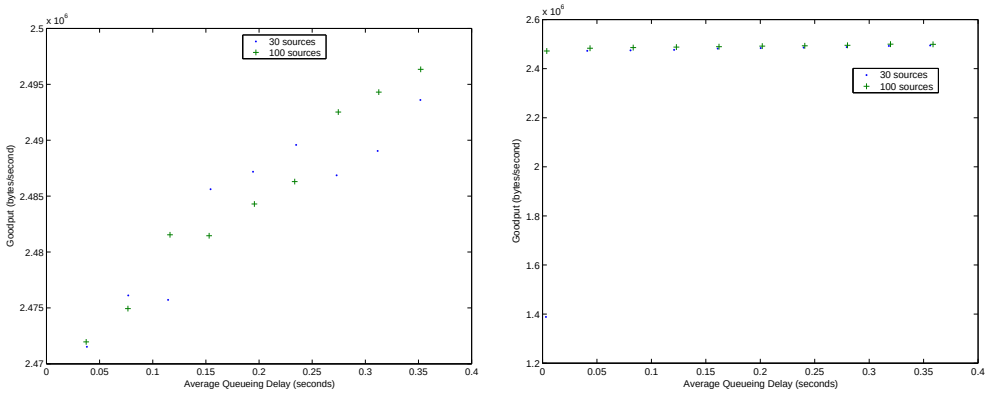


(e) mDTMW



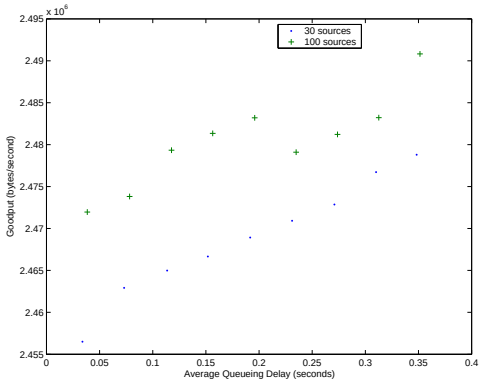
(f) Phase Lag

Figure 7.2: Goodput performance of an AQM scheme for increasing weights of the three predictors under a variable RTT scheme.

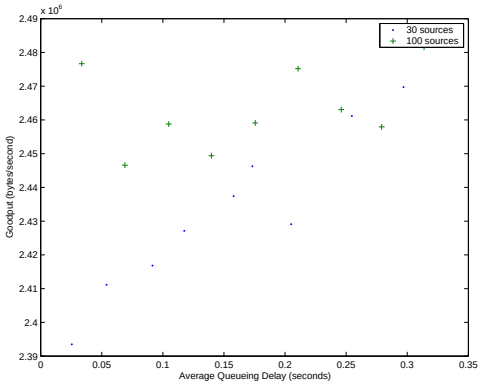


(a) Predictive Congestion Control

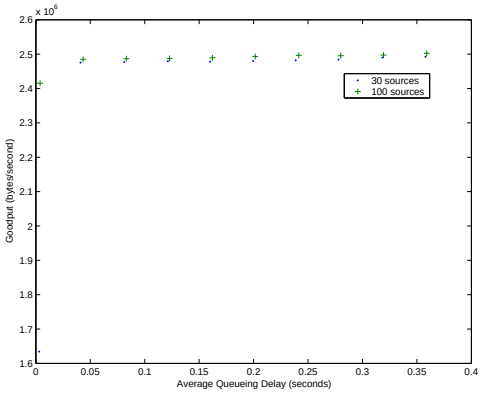
(b) mDTMW



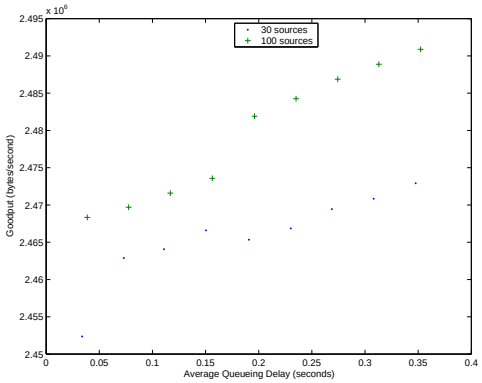
(c) Phase Lag



(d) Predictive Congestion Control

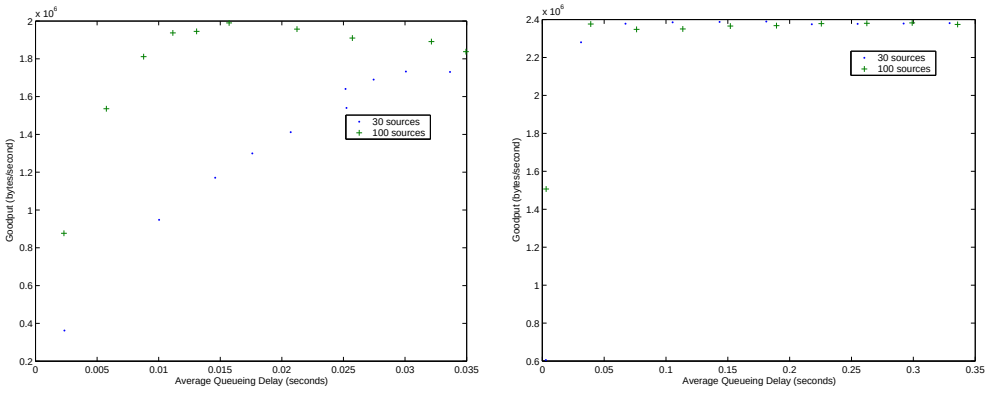


(e) mDTMW



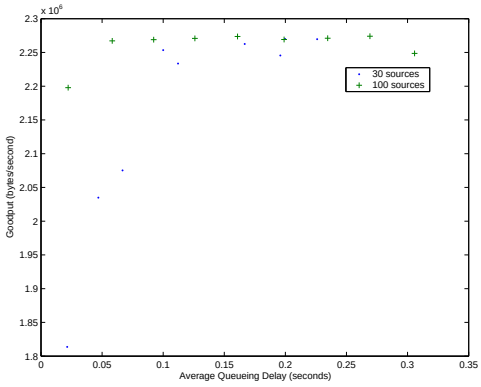
(f) Phase Lag

Figure 7.3: Variation in performance of the three predictors for increasing values of averaging queueing delay.

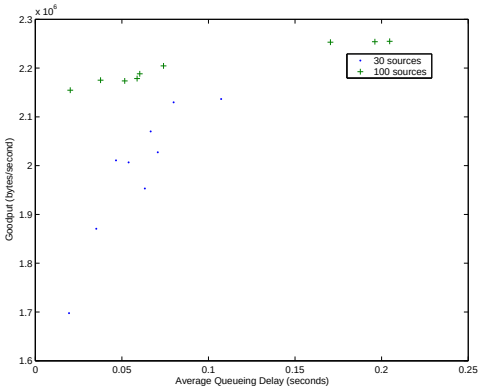


(a) Predictive Congestion Control

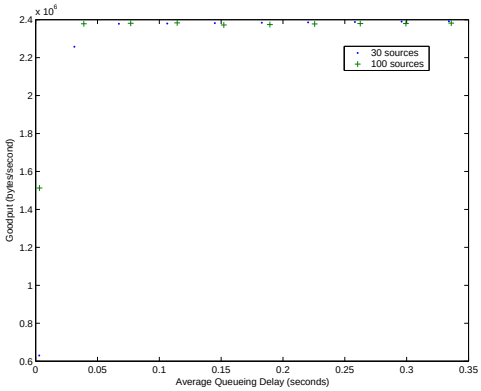
(b) mDTMW



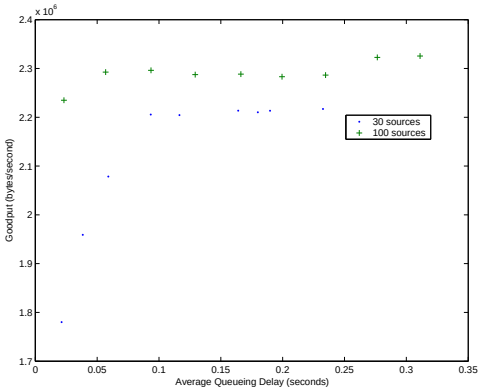
(c) Phase Lag



(d) Predictive Congestion Control

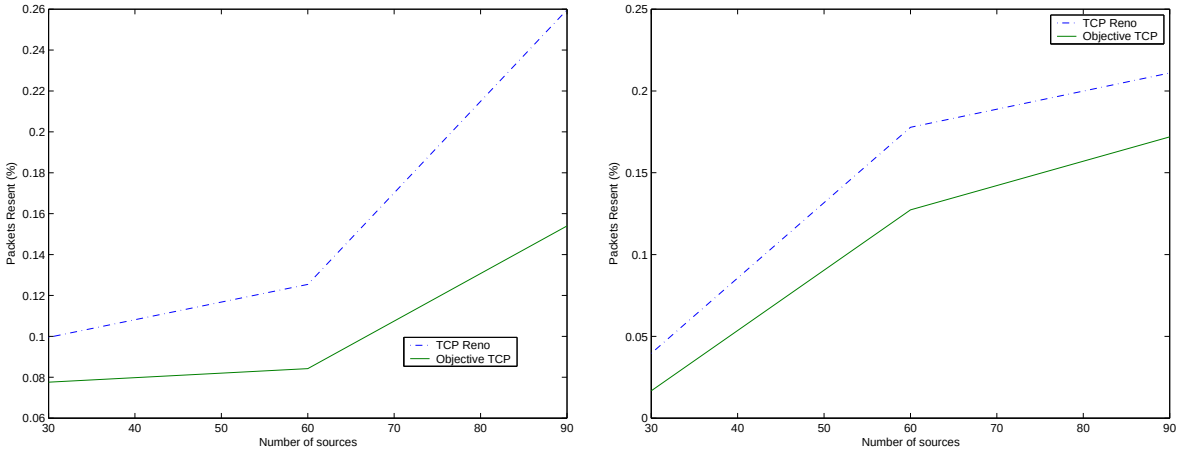


(e) mDTMW



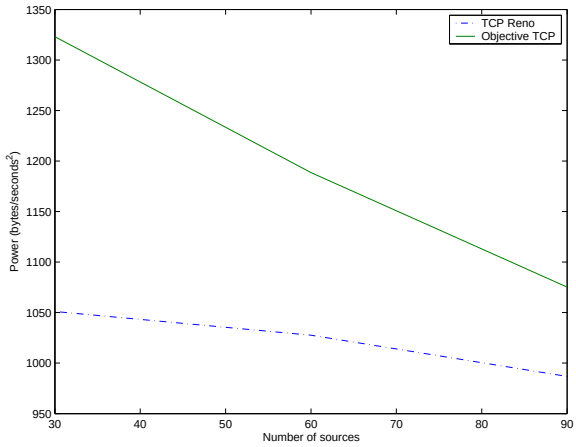
(f) Phase Lag

Figure 7.4: Variation in performance of the three predictors for increasing values of averaging queueing delay.

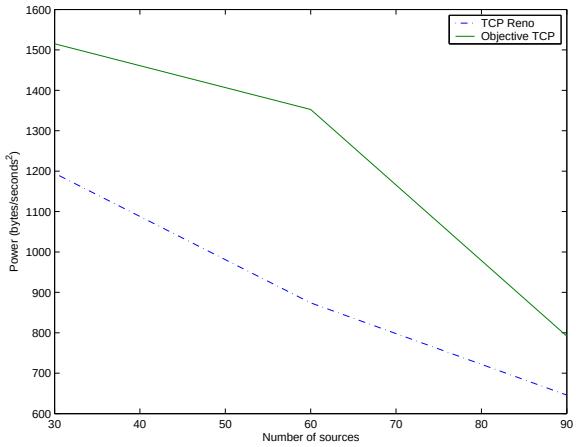


(a) Re-sent Drop Tail

(b) Re-sent RED



(c) Power Drop Tail



(d) Power RED

Figure 7.5: This figure shows the dependence of packet re-sent ratio on the number of sources for TCP Reno and Obj-TCP implemented on the end nodes and DT and RED active queue management schemes implemented at the routers. The figure illustrates that Obj-TCP performs better than TCP Reno.

Chapter 8

Appendix II: Computational Complexity Calculation

We use the following notations throughout our work:

p = Drop probability for the next slot;
Ar = Arrival rate at current slot;
q = Instantaneous queue size at current slot;
q_f = Instantaneous queue size at a future slot;
q_{fixed} = Target fraction of instantaneous queue size at current slot;
avg = Weighted average queue size at current slot;
avg_f = Weighted average queue size at a future slot;
Se = Service rate at current slot;
Ar_f = Expected arrival rate for next slot;
Min_{th}, Max_{th} = Minimum and maximum thresholds for weighted average queue size;
D[.] = Double Threshold Moving Windows;
$T1, T2$ = Thresholds for Double Threshold Moving Window;
Arr[.] = Array of arrival rates for past slots;
$M(a, b)$ = Conditional probability matrix. The probability of future arrival quantum being b given that the current arrival quantum is a; and
PL[.] = Array of arrival rates for future slots.

8.1 Double-Threshold Moving Window

At each sub slot (1ms):

1. For $i \leftarrow 2$ *to* N,
   $D[i-1] \leftarrow D[i]$, and
   $Arr[i-1] \leftarrow Arr[i]$.

2. If $(Ar > T1)$
   $D[N] \leftarrow 1$,
   Else
   $D[N] \leftarrow 0$.

3. Set,
   $Arr[N] \leftarrow Ar$, and

   $T1 \leftarrow \frac{1}{N} \sum_1^N Arr[i]$.

4. Set,
   $T2 = \sum_1^N D[i]$,

   $U \sim U(0,1)$,

   If $(U \leq \frac{T2}{N})$

   $Ar_f \leftarrow T1$,
   Else
   $Ar_f \leftarrow \frac{1}{2}T1$.

5. Expected arrival rate for the next slot is Ar_f.

Here N is the window length. $D[0]$ and $D[N]$ are windows for oldest and latest arrival rates, respectively.
Complexity: $O(N)$

8.2 Weighted Double-Threshold Moving Window

At each sub slot (1ms):

1. For $i \leftarrow 2$ *to* N,
   $D[i-1] \leftarrow D[i]$, and
   $Arr[i-1] \leftarrow Arr[i]$.

2. If $(Ar > T1)$
   $D[N] \leftarrow 1$,
   Else
   $D[N] \leftarrow 0$.

3. Set,
   $Arr[N] \leftarrow Ar$, and

   $T1 \leftarrow \frac{1}{N} \sum_1^N Arr[i]$.

4. Set,
   $T2 = \sum_1^N (D[i] * 2^{i-1})$,

   $U \sim U(0,1)$,

   If $(U \leq \frac{T2}{N})$

   $Ar_f \leftarrow T1$,
   Else
   $Ar_f \leftarrow \frac{1}{2}T1$.

5. Expected arrival rate for the next slot is Ar_f.

Here N is the window length. $D[0]$ and $D[N]$ are windows for oldest and latest arrival rates respectively.
Complexity: $O(N)$

8.3 Predictive Congestion Control

At each slot (10ms) consisting of two sub slots, $s1, s2$ (5ms) get arrival rates at the sub slots $a1, a2$:

1. Set,
   $M(a1, a2) \leftarrow M(a1, a2) + 1$,
   $temp \leftarrow 0$, and
   $Ex(s1) \leftarrow 0$.

2. For $i \leftarrow 1$ *to* N,
   $temp \leftarrow temp + M(a1, i)$.

3. For $i \leftarrow 1$ *to* N,
   $M(a1, i) \leftarrow \frac{M(a1,i)}{temp}$, and

   $Ex(s1) \leftarrow M(a1, i) * i$.

4. Expected quantum level for the next slot is $\leftarrow Ex(s2) = \sum_1^N (M(a2, i) * i)$.

5. For $i \leftarrow 1$ *to* N,
   Find arrival rate corresponding to quantum $Ex(s2)$, and
   Set Ar_f.

6. Expected arrival rate for the next slot is Ar_f.

Here N is the level of quantas into which the arrival rate has been divided (also the matrix size).
Complexity: $O(N)$

8.4 Phase Lag

At each slot (10ms):

1. $Ar \leftarrow PL[1]$

2. For $i \leftarrow 2$ *to* N,
   $PL[i-1] \leftarrow PL[i]$.

3. Set,

   $PL[N] \leftarrow total\ arrivals\ at\ future\ N+1^{th}\ slot$, and

   $Ar_f \leftarrow PL[1]$.

4. Expected arrival rate for the next slot is Ar_f.

Here N is the window length.
Complexity: $O(N)$

8.5 Random Early Detection

At each slot (10ms):

1. If $(q > 0)$

   $avg \leftarrow avg + w_q * (q - avg)$,

   Else

   $avg \leftarrow avg * (1 - w_q)^{(time - idle\_time)/s}$.

2. If$(min_{th} \leq avg < max_{th})$

   $count++$,

   $p_b \leftarrow \frac{max_P}{max_{th} - min_{th}} * avg - \frac{max_P}{max_{th} - min_{th}} * min_{th}$.

   If $((count > 0)$ and $(count \geq Approx[U/p_b]))$

   Mark packet, and

   $count \leftarrow 0$.

   If $(count = 0)$

   $U \leftarrow U(0, 1)$.

   Else If $(max_{th} \leq avg)$

   mark the packet, and

   $count \leftarrow -1$.

   Else $count \leftarrow -1$

3. If$(q = 0)$ $idle\_time \leftarrow time$

Complexity: $O(1)$

8.6 Estimated Future Average Queue Length

At each slot:

1. Get future arrival rate, Ar_f
2. Estimated future average queue length $= avg \leftarrow avg * w1 + avg_f * w2 + q * (1 - w1 - w2)$
   $avg_f \leftarrow (1 - u1) * avg_f + u1 * q_f$
   $q_f \leftarrow q_f + Ar_f - Se$
3. If$(avg < min_{th})$
   No packet marking.
   Else If $(min_{th} \leq avg < max_{th})$
   Packet mark probability $= p \leftarrow \frac{max_P}{max_{th} - min_{th}} * avg - \frac{max_P}{max_{th} - min_{th}} * min_{th}$.
   Else
   Packet mark probability $= p \leftarrow 1$.

Complexity: $O(1)$

8.7 Least Mean Square Fixed Queue Occupancy

At each slot:

1. Get future arrival rate, Ar_f
2. Packet mark probability is

$$p = \begin{cases} 0 & q < Se + q_{fixed} - Ar_f, \\ \frac{q + Ar_f - Se - q_{fixed}}{Ar_f} & Se + q_{fixed} - Ar_f < q < Se + q_{fixed}, \text{ and} \\ 1 & q > Se + q_{fixed}. \end{cases}$$

Complexity: $O(1)$

8.8 TCP Reno

Pseudo-code for TCP Reno is:

1. After new ACK is received:
   If ($cwnd < ssthresh$)
   $cwnd \leftarrow cwnd + 1$.
   Else
   $cwnd \leftarrow cwnd + \frac{1}{cwnd}$.

2. If congestion due to timeout
   $ssthresh \leftarrow \frac{cwnd}{2}$, and
   $cwnd \leftarrow 1$.

3. If 3 duplicate ACKs received in a row
   $ssthresh \leftarrow Max(2, \frac{cwnd}{2})$
   Retransmit the lost segment,
   $cwnd \leftarrow ssthresh + 3 * segment\_size$.
   For each additional duplicate ACK,
   $cwnd \leftarrow cwnd + segment\_size$, and
   Transmit a packet if allowed by new value of $cwnd$.
   If new ACK arrives,
   $cwnd \leftarrow ssthresh$, and
   Transmit a packet if allowed by new value of $cwnd$.

Complexity: $O(1)$

8.9 Obj-TCP

The complexity of Obj-TCP is the same as that of DTMW, which is, $O(N)$.

List of References

[1] CISCO Technical Papers : Weighted RED, Congestion Avoidance Overview. 2000.

[2] CISCO White Papers : Performance Measurements of Advanced Queuing Techniques in the Cisco IOS. 2000. http://www.cisco.com/warp/public/614/15.html.

[3] A. Akella, S. Seshan, R. Karp, S. Shenker, and C. Papadimitriou. Selfish behavior and stability of the Internet: A game-theoretic analysis of TCP. *ACM Special Interest Group on Data Communication*, pages 117 – 130, August 2002.

[4] M. Allman and A. Falk. On the Effective Evaluation of TCP. *ACM Computer Communication Review*, 5(29), 1999.

[5] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control, RFC 2581. Academic Press, New York, April 1999.

[6] T. Alpcan and T. Basar. A Utility-Based Congestion Control Scheme for Internet-Style Networks with Delay. *submitted to IEEE Infocom 2003*, 2003.

[7] E. Altman and T. Basar. Multiuser rate-based flow control. *IEEE Transactions on Communications*, 46(7):940–949, July 1998.

[8] S. L. Blake and Z. Haraszti. Personal Communication. In *Ericsson IPI, Centennial Campus, NC State University*, May 2002.

[9] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on Queue Management and Congestion Avoidance in the Internet. *RFC 2309*, April 1998.

[10] Lawrence S. Brakmo and Larry L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, 1995.

[11] H. W. Braun and K. Claffy. Network analysis in support of Internet policy requirements. In *Cooperative Association for Internet Data Analysis - CAIDA , San Diego Supercomputer Center, University of California, San Diego*, 1993.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Massachusetts, 2nd edition, 2001.

[13] M. Crovella, M. Taqqu, and A. Bestavros. Heavy-Tailed Probability Distributions in the World Wide Web. Chapman and Hall, New York, 1998. In A Practical Guide To Heavy Tails, Chapter 1.

[14] S. Deng. Empirical Model of WWW Document Arrivals at Access Link. *IEEE International Conference on Communications*, 3:71–86, 1996.

[15] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin. A Self-Configuring RED Gateway. In *Proceedings of INFOCOM 99*, volume 3, pages 1320–1328, 1999.

[16] D. Figueiredo, B. Liu, V. Misra, and D. Towsley. The Autocorrelation structure of TCP traffic, 2000. Submitted for review.

[17] S. Floyd and K. R. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, 1999.

[18] S. Floyd, R. Gummadi, and S. Shenker. Adaptive RED: An Algorithm for Increasing the Robustness of RED, 2001.

[19] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ ACM Transactions on Networking*, 1(4):397–413, 1993.

[20] Y. Gao, G. He, and J. C. Hou. On Exploiting Traffic Predictability in Active Queue Management. *To appear in IEEE/INFOCOM*, 2002.

[21] J. C. Hoe. Improving the Start-up Behavior of a Congestion Control Sheme for TCP. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, volume 26,4, pages 270–280, New York, August 1996. ACM Press.

[22] C. Huang. Long Range Dependent Traffic: Modeling, Simulation and Congestion Control. In *PhD Thesis : Carleton University*, 1997.

[23] P. J. Huber. *Robust Statistical Procedures*. Regional conference series in applied mathematics ; 27. Society for Industrial and Applied Mathematics, Philadelphia, 2nd edition, 1996.

[24] V. Jacobson. Congestion Avoidance and Control. *ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium in Stanford, CA*, 18, 4:314–329, August 1988.

[25] K. Kar, S. Sarkar, and L. Tassiulas. A Simple Rate Control Algorithm for Maximizing Total User Utility. In *INFOCOM*, pages 133–141, 2001.

[26] L. Kleinrock. Power and deterministic rules of thumb for probabilistic problems in computer communications. In *ICC*, volume 3.

[27] Richard J. La and Venkat Anantharam. Window-Based Congestion Control with Heterogeneous Users. In *INFOCOM*, pages 1320–1329, 2001.

[28] K. Laevens, P. Key, and D. McAuley. An ECN-based end-to-end congestion-control framework: experiments and evaluation, October 2000.

[29] A. M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw Hill International, 3rd edition, 2000.

[30] D. Lin and R. Morris. Dynamics of Random Early Detection. In *SIGCOMM '97*, pages 127–137, Cannes, France, september 1997.

[31] S. H. Low and D. E. Lapsley. Optimization flow control — I: Basic Algorithm and Convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, 1999.

[32] S. Manthorpe, I. Norris, and J.Y.L. Boudec. The second-order characteristics of TCP, October 1996.

[33] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options (Internet draft, work in progress), October 1996.

[34] S. S. Oruganti and M. Devetsikiotis. Analyzing Robust AQM Schemes: A Comparative Study of Predictors and Controllers. *In Proceedings in IEEE International Conference on Communications,ICC,Anchorage, Alaska*, May 2003. http://www4.ncsu.edu/∼ssorugan.

[35] T. J. Ott, T. V. Lakshman, and L. H. Wong. SRED: Stabilized RED. In *Proceedings of INFOCOM*, volume 3, pages 1346–1355, 1999.

[36] R. Pan, B. Prabhakar, and K. Psounis. CHOKE, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation. In *INFOCOM (2)*, pages 942–951, 2000.

[37] K. Park and W. Willinger. Congestion Control for Self-Similar Network Traffic. In *Self-Similar Network Traffic and Performance Evaluation*, pages 446–480, 2000.

[38] K. Psounis, R. Pan, and B. Prabhakar. Approximate fair dropping for variable-length packets. *IEEE Micro*, 21(1):48–56, 2001.

[39] A. Sang and S. Li. A Predictability Analysis of Network Traffic. In *INFOCOM (1)*, pages 342–351, 2000.

[40] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, RFC 2001, January 1997.

[41] A. Veres and M. Boda. The Chaotic Nature of TCP Congestion Control. In *INFOCOM (3)*, pages 1715–1723, 2000.

[42] B. Zheng and M. Atiquzzaman. DSRED: improving performance of active queue management over heterogeneous networks. *IEEE International Conference on Communications,ICC*, 8:2375–2379, 2001.