Proportional Integrator with Short-lived flow Adjustments

By

Minchong Kim

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

In

Computer Science

by

____

Jan 2004

APPROVED:

____

Dr. Robert Kinicki, Major Advisor

____

Dr. Mark Claypool, Reader

____

Dr. Michael Gennert, Head of Department

ABSTRACT

The number of Web traffic flows dominates Internet traffic today and most Web interactions are short-lived HTTP connections handled by TCP. Most core Internet routers use Drop Tail queuing which produces bursts of packet drops that contribute to unfair service. This thesis introduces two new active queue management (AQM) algorithms, PISA (PI with Short-lived flows Adjustment) and PIMC (PI with Minimum Cwnd). These AQMs are built on top of the PI (Proportional Integrator). To evaluate the performance of PISA and PIMC, a new simple model of HTTP traffic was developed for the NS-2 simulation. TCP sources inform PISA and PIMC routers of their congestion window by embedding a source hint in the packet header. Using the congestion window, PISA drops packets from short-lived Web flows less than packets from long-lived flows. Using a congestion window, PIMC does not drop a packet when congestion window is below a fixed threshold. This study provides a series of NS-2 experiments to investigate the behavior of PISA and PIMC. The results show fewer drops for both PISA and PIMC that avoids timeouts and increases the rate at which Web objects are sent. PISA and PIMC improve the performance of HTTP flows significantly over PI. PISA performs slightly better than PIMC.

ACKNOWLEDGEMENTS

Throughput my graduate student career, I have been fortunate enough to have the help and support from many people. I would especially like to thank:

- Professor Robert Kinicki, my major advisor, for giving me a chance to have an insight in his research field of network congestion control, and for his patience, advice and guidance for helping me finish this thesis.

- Professor Mark Claypool, my thesis reader, for his insightful comment and advice on this thesis, and for his congo server.

- CC (congestion control) group members for their many useful and insightful discussions about network congestion control mechanisms. Special thanks to Jae Chung, Choong-Soo Lee, Huahui Wu, James Nichols, Mingzhe Li for their technical and academic help.

- Michael Voorhis, and Jesse Banning for providing enough disk space on CS and q1 machines.

Finally, I would most like to thank my parents, my brother and sister for their continual encouragement, guidance and support.

TABLE OF CONTENTS

ABSTRACT………… i

ACKNOWLEDGEMENTS………… ii

LIST OF TABLES………… v

LIST OF FIGURES………… vi

Chapter 1 Introduction………… 1

Chapter 2 Background and Related Work………… 4

2.1 Network Performance Metrics………… 4

2.2 TCP………… 5

2.2.1 Versions of TCP………… 5

2.2.2 Details of TCP Congestion Control………… 6

2.3 AQM for Congestion Control………… 8

2.3.1 Drop Tail (FIFO)………… 8

2.3.2 RED………… 8

2.3.3 SHRED………… 9

2.3.4 RIO and RIO-PS………… 10

2.3.5 PI………… 11

2.3.6 REM………… 13

2.3.7 AVQ………… 13

2.4 Modeling Web Traffic………… 13

2.4.1 Web Traffic Model in SHRED………… 14

2.4.2 Web Traffic Model in RIO-PS………… 14

2.4.3 ON/OFF Pareto………… 14

Chapter 3 Design of Web Traffic Model and Two AQM Algorithms………… 16

3.1 Web Traffic Model………… 16

3.2 PISA………… 17

3.3 PIMC……………………………………………………………………….. 19
Chapter 4 Experimental Methodology and Tools…………………………………... 22
4.1 Network Simulator ns-2………………………………………………………. 22
4.2 Simulation Input…………………………………………………………… 23
4.3 Simulation Output Traces……………………………………………….…. 23
4.4 Data Extraction ……………………………………………………………. 24
4.5 Data Analysis………………………………………………………………. 25
4.6 Experimental Setup and Validations……………………………………….…. 25
4.7 Simulation Scenarios………………………………………………………. 27
Chapter 5 Performance Evaluation and Analysis…………………………………... 29
5.1 PI compared to Drop Tail……………………..…………………………… 29
5.2 Drop Rate……………………………………………………………………... 30
5.3 Queue Length………………………………………………………………. 33
5.4 Packet Delay and Object delay……………………………………………..…. 34
5.5 Utilization………………………………………………………………….. 35
5.6 Experiments with Heavier Congestion……………………………………….. 37
5.6.1 Increased FTP flows…………………………………………….……. 38
5.6.2 Reduced bandwidth………………………………………………….... 41
5.7 Various RTTs of HTTP flows………………………………………………… 44
Chapter 6 Conclusions and Future work…………………………………………… 47
6.1 Conclusions………………………………………………………………… 47
6.2 Future Research……………………………………………………………. 47

Bibliography………………………………………………………………………… 49

LIST OF TABLES

Table 3.1 MinCwnd………………………………………………………………… 21

Table 4.1 Variance of Packet Drop Rate…………………………………………...… 25

Table 5.1 Average Drop Rate of PI and Drop Tail ………………………………..... 30

Table 5.2 Average Cwnd and Drop Rate with 10 FTP and 100 HTTP flows……….... 31

Table 5.3 Utilization of Link Capacity of 10 Mbps………………………………... 36

Table 5.4 Number of Web Objects transmitted during Standard Experiment……….. 37

Table 5.5 Heavier Congestion Scenarios……………………………………………... 37

Table 5.6 Average Cwnd and Drop Rate with 50 FTP flows and 100 HTTP flows on Capacity of 10 Mbps………………………………………………….…….…... 39

Table 5.7 Number of Web Objects transmitted with 50 FTP and 100 HTTP flows …. 41

Table 5.8 Average Cwnd and Drop Rate with 10 FTP and 100 HTTP flows on 42 Capacity of 5 Mbps………………………………………………………………… 43

Table 5.9 Number of Web Objects transmitted with 10 FTP and 100 HTTP flows on Capacity of 5 Mbps………………………………………………………………… 44

Table 5.10 The Number of Web Objects with of FTP RTT of 200 ms………………. 45

Table 5.11 The Number of Web Objects with of FTP RTT of 2000 ms…………….… 45

LIST OF FIGURES

Figure 2.1 Congestion Control………………………………………………………….. 7
Figure 2.2 IP Header…………………………………………………………………….. 9
Figure 2.3 PI Algorithm…………………………………………………………………. 12
Figure 2.4 Pareto Parameter Setting………………………………………………... 15
Figure 3.1 PI Algorithm………………………………………………………………….. 18
Figure 3.2 PISA Algorithm…………………………………………………………. 18
Figure 3.3 Weighted Cwnd Average………………………………………………... 19
Figure 3.4 PIMC Algorithm ………………………………………………………... 20
Figure 3.5 CDF of Object Delay ………………………………………………..... 21
Figure 4.1 The Simulation Procedure with NS-2……………………………………... 22
Figure 4.2 A Sample of NS-2 Output Trace……………………………………….. 24
Figure 4.3 Packet Drops………………………………………………………………. 26
Figure 4.4 Queue Length……………………………………………………………… 26
Figure 4.5 Standard Simulation Topology……………………………………….… 27
Figure 5.1 Packet Drops with PI and Drop Tail…………………….……………… 29
Figure 5.2 PISA FTP Flow Packet Drops with 10 FTP and 100 HTTP flows………… 30
Figure 5.3 PISA HTTP Flow Packet Drops with 10 FTP and 100 HTTP flows……… 30
Figure 5.4 PIMC FTP Flow Packet Drops with 10 FTP and 100 HTTP flows……….. 31
Figure 5.5 PIMC HTTP Flow Packet Drops with 10 FTP and 100 HTTP flows……... 31
Figure 5.10 CDF of CWND for when Packets are Dropped ………………………….. 32
Figure 5.11 CWND of FTP Packets Dropped with 10 FTP and 100 HTTP flows……. 32
Figure 5.12 CWND of HTTP Packets Dropped with 10 FTP and 100 HTTP flows…. 32
Figure 5.13 Queue Length ……………………………………………………………. 33
Figure 5.14 CDF of Queue Length of Packets Dropped with 10 FTP and 100 HTTP flows……………………………………………………………………………… 34
Figure 5.15 CCDF o Queue Length of Packets Dropped with 10 FTP and 100 HTTP flows……………………………………………………………………………… 34
Figure 5.16 Packet Delay of a FTP flow with 10 FTP and 100 HTTP flows…………. 35
Figure 5.17 Packet Delay of a HTTP flow with 10 FTP and 100 HTTP flows……….. 35

Figure 5.18 CDF of Web Object Delay with 10 FTP and 100 HTTP flows…………….. 35
Figure 5.19 CCDF of Web Object Delay with 10 FTP and 100 HTTP flows………… 35
Figure 5.20 PI Utilization……………………………………………………………... 36
Figure 5.21 PISA Utilization…………………………………………………….…. 36
Figure 5.22 PIMC Utilization…………………………………………………………. 36
Figure 5.23 Drop Tail Utilization……………………………………………………… 36
Figure 5.24 PISA FTP Flow Packet Drops with 50FTP and 100 HTTP flows………. 38
Figure 5.25 PISA HTTP Flow Packet Drops with 50FTP and 100 HTTP flows……... 38
Figure 5.26 PIMC FTP Flow Packet Drops with 50FTP and 100 HTTP flows………. 38
Figure 5.27 PIMC HTTP Flow Packet Drops with 50FTP and 100 HTTP flows…….. 38
Figure 5.28 Packet Delay of a FTP flow with 50 FTP and 100 HTTP flows…………. 40
Figure 5.29 Packet Delay of a HTTP flow with 50 FTP and 100 HTTP flows……….. 40
Figure 5.30 CDF of Web Object Delay with 50 FTP and 100 HTTP flows…………... 40
Figure 5.31 CCDF of Web Object Delay with 50 FTP and 100 HTTP flows………… 40
Figure 5.32 PISA FTP Flow Packet Drops with Capacity of 5 Mbps………………... 41
Figure 5.33 PISA HTTP Flow Packet Drops with Capacity of 5 Mbps……………..… 41
Figure 5.34 PIMC FTP Flow Packet Drops with Capacity of 5 Mbps………………... 42
Figure 5.35 PIMC HTTP Flow Packet Drops with Capacity of 5 Mbps……………... 42
Figure 5.36 Packet Delay of a FTP flow with Capacity of 5 Mbps…………………… 43
Figure 5.37 Packet Delay of a HTTP flow with Capacity of 5 Mbps……………….… 43
Figure 5.38 CDF of Web Object Delay with Capacity of 5 Mbps……………….……. 43
Figure 5.39 CCDF of Web Object Delay with Capacity of 5 Mbps…………………... 43

1. Introduction

Traffic generated by the World Wide Web (WWW) dominates the Internet today. Web traffic is transmitted by the Hypertext Transfer Protocol (HTTP) flows via the Transmission Control Protocol (TCP). HTTP flows introduce most of the TCP connections on the Internet. Although the number of HTTP flows on the Internet is dominant, Web traffic occupies a relatively smaller portion of Internet traffic volume when compared to the volume associated with FTP (File Transfer Protocol) traffic and Peer-to-Peer traffic. Empirical data indicates that the average size of each Web object is about 8 – 12KB [GM01] and 2KB [CKS03]. This is tiny compared to huge file transfers, and each embedded object in a Web page is downloaded via a separate TCP connection. Thereby, Web traffic is considered primarily as short-lived flows and FTP traffic often involves long-lived flows.

As Internet traffic volume continues to grow, network congestion is more likely to occur and it becomes more challenging to provide good throughput to millions of Web users under congestion. When multiple input streams from a number of senders arrive at a router whose output capacity is less than the sum of the inputs, the router can be congested. Bursty traffic, ACK compression, and flash crowds cause congestion. The most common router on the Internet, the Drop Tail router, reacts to congestion by dropping a packet due to lack of buffer space no matter what kind of Internet traffic. The result of Drop Tail behavior produces bursts of packet drops that contribute to unfair service, especially when the mixture of long and short flows is transmitted. This causes unfairness for the share of throughput of short-lived flows.

Most active queue management (AQM) techniques do not focus on short flows. Web traffic is one type of traffic transferred on top of TCP. Other kinds of traffic transferred via UDP (User Datagram Protocol) such as video and audio streams are not covered in this thesis. The HTTP traffic, short-lived flows, does not get a fair share of the throughput compared to long-lived FTP flows with the existing AQMs. That is, the short-lived flows are more likely to be less competitive in the war of throughput against long-lived flows when a mixture of short-lived and long-lived flows is transferred on the Internet [GM01]. To reduce the unfairness to Web traffic, a few AQMs have focused on short flows. RIO-PS (RIO with Preferential treatment to Short flows) [GM01] and SHRED (Short-lived flows friendly RED) [HCK02] attempt to improve fairness for short flows by providing less delay and more throughput. However, RED (Random Early Detection) with dropping has a negative effect on the transmission latencies of short-lived (web) flows [CJO01].

This thesis proposes two new AQMs, PISA (PI with Short-lived flow Adjustments), and PIMC (PI with Minimum Cwnd) based on PI to improve Web traffic performance. The Short-lived flow Adjustments (SA) is decoupled from SHRED (Short-lived flows friendly RED) [HCK02] and the minimum threshold idea from RIO-PS (RIO with Preferential treatment to Short flows), is applied to TCP congestion windows and combine with PI (Proportional Integrator) scheme to yield PIMC. Using TCP congestion window (cwnd) from a TCP source, PISA controls the drop probability based on the ratio of a packet's cwnd to average cwnd. The objective is help short-lived flows by dropping fewer HTTP flow packets while dropping FTP flow packets more aggressively. Relying also on the cwnd source hint, PIMC does not drop packets with cwnd below a fixed threshold and drops packets with cwnd larger than the threshold based the PI algorithm.

To conduct thorough comparison of PISA and PIMC with existing AQMs, this investigation developed a new simple Web traffic model that accurately characterize the behavior of TCP congestion window for HTTP short-lived flows.

The performance of PISA and PIMC are investigated with standard experiment setting and three other sets of experiment providing heavy congestion. The NS-2 simulation results using the simple Web traffic model presented in this investigation show that PISA performs better than PI and PIMC. By avoiding timeouts, PISA improves object transmission rate by 22% over PI in moderate congestion and by up to 40% in heavy congestion. PIMC outperforms PIMC, PI, and Drop Tail.

This thesis is organized as follows. Chapter 2 reviews the background of TCP and previous AQM research. Definitions of measurement criteria are presented, various versions of TCP are described with congestion control mechanisms in TCP/IP networks, and a summary of the related work in congestion control schemes is given. Additionally previous Web traffic models are also reviewed. Chapter 3 presents a new simple Web traffic model and introduces PISA, and PIMC. Chapter 4 introduces the simulation methodology deployed in this investigation with the simulation tool NS-2, and described experimental procedures used throughout this study. This chapter also briefly describes preliminary experiments with PI (Proportional Integrator) and simulation scenarios. Chapter 5 presents results and evaluates the performance of PISA and PIMC compared to PI and Drop Tail schemes. The conclusions of this thesis and the future work are presented in Chapter 6.

Chapter 2 Background and Related Work

This chapter provides definitions and background information required for a better understanding of AQM and congestion control issues. Topics discussed include performance measurement criteria, TCP versions and the details of TCP congestion control, pertinent previous AQM research, and a review of previous attempts to model Web traffic via simulations.

2.1 Network Performance Metrics

The performance of AQM in this investigation is evaluated by network performance indicators. The performance measures used in this investigation such as utilization, packet delay, object delay, queue length in a router, packet drop rate, and fairness are defined in this section.

Throughput is defined as the rate at which the packets are sent by a network source in megabits/sec (Mbps). Goodput is the effective data rate received at destination in megabits/sec (Mbps), which does not include retransmitted packets. Throughput includes retransmitted packets so throughput is normally a little higher than goodput. Utilization is defined as the fraction of link capacity being used for transferring data. Utilization can be expressed as a decimal point between 0 and 1 or as a percentage (%).

Delay is the one-way end-to-end transmission time of a packet from a TCP source to a TCP destination. It includes transmission time, propagation delay and queuing delay. In the experiments presented in this thesis, both packet delay and object delay are measured since the performance of HTTP is a key part of this study, delay is measured in the direction from a Web server to a Web client. Packet delay is the elapsed time a packet is sent from a source to a destination. Object delay is the delay of a HTTP object, and measured from a transmission start time to transmission end time. As another time measurement, round-trip time (RTT) is defined as the time required for a data packet to travel from the source to the destination and the time for the responding ACK packet to return to the source. While delay is the time taken in one way, RTT includes transmission time in both directions.

Queue length is defined as the number of packets in the queue for a router outgoing link and is considered as an important performance measurement related to delay.

Packet drop rate is the rate at which the packets are dropped at a router queue in packets per second (packets/sec). Drop rate represents the difference between goodput and throughput. It can be used to compute fairness. Drop ratio is the ratio of the number of packets dropped to total number of packets sent and is used to compare with different router AQMs.

Fairness is defined as how fairly all flows are treated as they traverse a router. There is formal fairness such as Jain's fairness Index and max-min fairness. The fairness in this investigation is considered by FTP and HTTP group measurements. Due to offtime characteristic per HTTP flows, in this study capturing fairness was difficult (see section 5.5).

2.2 TCP

2.2.1 Versions of TCP

Multiple versions of TCP have been developed. TCP Tahoe, known as BSD Network Release 1.0, corresponds to the original implementation of Jacobson's congestion control mechanism [PD00]. Tahoe uses a basic go-back-n model using slow start, congestion avoidance and fast retransmit algorithm. With fast retransmit, after receiving a small number of duplicate ACKs for the same TCP packet, the data sender infers that the packet has been lost and retransmits the packet without waiting for the retransmission timer to expire. Tahoe eliminated the Internet congestion collapse of 1986.

TCP Reno, known as BSD Network Release 2.0, adds the fast recovery mechanism [PD00]. A TCP sender enters fast recovery after receiving three duplicate ACKs. The sender retransmits one packet and reduces its congestion window by half. Instead of slow start, the Reno sender uses additional incoming duplicate ACKs to clock subsequent outgoing packets. This prevents the pipe from going empty after fast retransmit, thereby avoiding the need to slow start after a single packet loss. TCP Reno greatly improves performance in the face of single packet loss, but can suffer when multiple packets are lost. In addition, TCP Reno includes an optimization known as header prediction that optimizes for the common case that segments arrive in order. TCP Reno also supports delayed ACKs acknowledging every other segment rather than every segment although this is a selectable option.

While a partial ACK (an ACK for some but not all of the packets that were outstanding at the start of the fast recovery period) takes TCP out of Fast Recovery in Reno, in TCP New Reno, partial ACKs do not take TCP out of fast recovery. Partial ACKs received during fast recovery are treated as an indication that the packet immediately following the acknowledged packet has been lost and should be retransmitted. Thus, when multiple packets are lost, New Reno can recover without a retransmission timeout.

2.2.2 Details of TCP Congestion Control

TCP congestion control was introduced into the Internet in the late 1980s by Van Jacobson [PD00]. The TCP congestion control algorithms are designed to share a bottleneck link's bandwidth among the TCP connections traversing that link. The idea of TCP congestion control is for each source to determine how much capacity is available in the network, so that it knows how many packets it can have in transit. This section describes the predominant end-to-end congestion control in use today, that implemented by TCP.

The TCP sender is not able to open a new connection to transmit a large burst of data at once. The TCP sender is limited by a small initial value of the congestion window (cwnd). During slow start, the TCP sender increases its cwnd by one for every acknowledgement (ACK) received in a round-trip time. This effectively doubles the cwnd per round-trip time. When the sender's congestion window exceeds the slow start threshold (ssthresh), slow start ends and TCP enters congestion avoidence. Slow start is used at the beginning of a transfer or after a packet loss detected by retransmission timeout (RTO). TCP uses slow start to determine the available capacity for the purpose of avoiding congestion of the network. During congestion avoidance, TCP increments cwnd by one packet per round-trip time (RTT). When an RTO occurs, congestion avoidance ends with setting ssthresh to half of the current cwnd value and then TCP Tahoe enters slow start again. This procedure is repeated until the transmission ends. Figure 2.1 shows an example of how TCP Slow start and Congestion avoidance works.

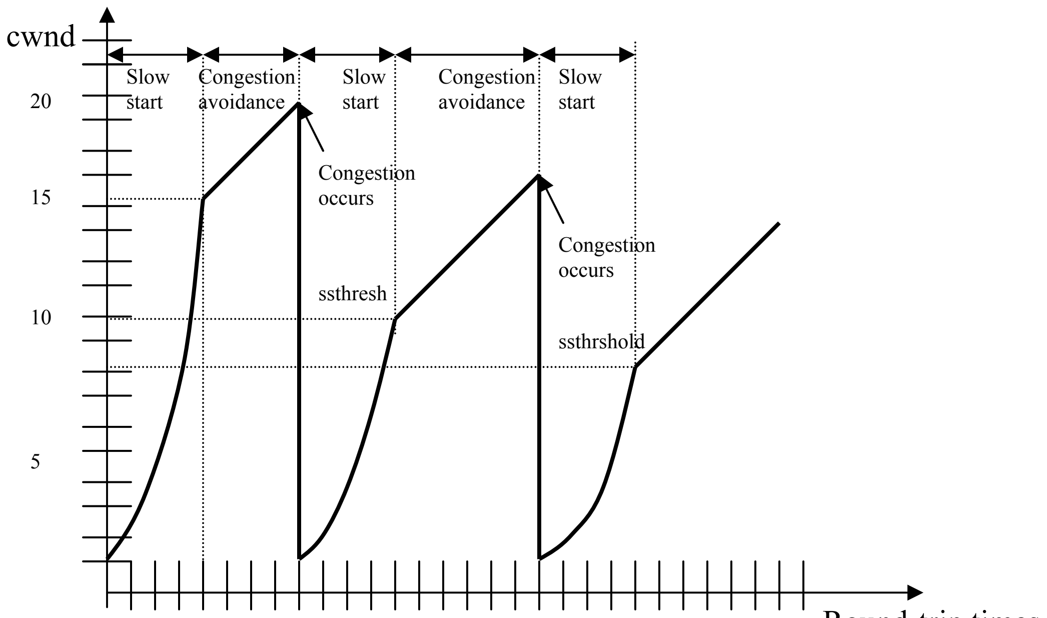

Figure 2.1 Congestion Control [WLOO]

TCP uses fast retransmit to detect packet loss. Once TCP receives three duplicate acknowledgements, TCP assume a packet has been lost. Then TCP retransmits the missing packet without waiting for RTO to expire. After sending the missing packet, TCP Reno sets cwnd to half of current cwnd and performs congestion avoidance, but not slow start. This is the fast recovery algorithm after fast transmit for TCP Reno. It improves throughput under congestion because TCP does not have to transmit with beginning window size from 1. Before sending, TCP sender sets a retransmit timer to determine if a packet has been lost in the network. If TCP does not receive acknowledgement for the packet until the retransmit timer expires, the sender considers it as a packet loss, and sets slow start threshold to half of the current cwnd, and then performs slow start and retransmit the lost packet. If the TCP does not receives acknowledgement for the retransmitted packet before the retransmit timer expires, the retransmit timer uses exponential backoff. That is, the value of the next retransmit timeout increases. Initial time out (ITO) is typically conservatively set to three seconds and RTO is gradually adjusted based on RTT estimates.

2.3 AQM for Congestion Control

A study of short-lived flows [GM01] shows that short-lived flows receive less service compared to long-lived flows due to the conservative nature of the TCP Reno congestion control algorithm described above. Under congestion, short-lived flows tend to receive less than their fair share of throughput without differentiated treatment from an AQM policy. Routers interact with TCP sources to deal with congestion control. Active queue management provides special actions to be taken at router queue. In this section, we present AQMs that classify flows and treat the class of short-lived flows differently than the long-lived flows along with a few well-known AQMs.

2.3.1 Drop Tail (FIFO)

Drop tail, commonly used in most Internet routers, implements first-come-first-served (FCFS) queuing and drop-on-overflow buffer management. The first packet that arrives at a router is the first packet to be transmitted. If a packet arrives at a router whose outgoing link queue is full, then the router discards the packet regardless of which flow the packet belongs to or how important the packet is. Under congestion, drop tail has high utilization, but high delay because of long queuing delay. High delay is not good for Web applications that provide interactive communication.

2.3.2 RED

RED (Random Early Detection) [FJ93], a well-known AQM scheme, detects incipient congestion using average queue length. RED uses the parameter set: minimum threshold (minthresh), maximum threshold (maxthresh), maximum drop probability (maxp), and weight parameter in order to probabilistically drop packets arriving at a router. RED maintains an average queue size, which is computed as an exponentially weighted moving average of the instantaneous queue size. If the average queue length is smaller than minthresh, no packet is dropped. If the average queue length is between minthresh and maxthresh , RED's packet drop probability varies linearly between zero and maxp and the packet could be early dropped depending on the drop probability. If the average queue length is greater than maxth, the drop probability equals one and the packet is dropped. By dropping the packet or marking using ECN (Explicit Congestion Notification), the router indicates incipient congestion to the source. Early dropping helps the RED router keep the average queue level low during congestion periods. However, increasing the maximum drop probability leads the instantaneous queue length to large oscillations. In the results of the Christiansen et al. [CJO01] their investigation of RED for Web traffic shows queuing delay and throughput in RED is sensitive to RED parameters. So an appropriate parameter setting is important to performance. However, finding the optimal RED parameter setting is shown to be problematic.

2.3.3 SHRED

SHRED (SHort-lived flows friendly RED) [HCK02] attempts to give preferential treatment to short-lived flows using a source hint. The source hint, fetched from TOS (Type of Service) field in the IP (Internet Protocol) header in a packet, contains the current congestion window size of a flow.

4-bit Version	4-bit IHL	8-bit Type of service	16-bit Total length
16-bit Identification		3-bit Flags	13-bit Fragment offset
8-bit Time to live		8-bit Protocol	16-bit Header checksum
32-bit Source address			
32-bit Destination address			
32-bit Option + Padding			

Figure 2.2 IP Header

The SHRED queue is managed in the same way as RED does except that minimum threshold and maximum drop probability are computed based on cwnd for each packet. The ratio of a cwnd to the weighted average cwnd is used to adjust drop probability, called short-lived flow adjustments (SA). When the average queue length is between minimum and maximum thresholds, if the ratio is greater than one, that is, cwnd is greater than average cwnd, minimum threshold decreases pushing up a maximum drop probability. Otherwise, minimum threshold increases, pushing down a maximum drop probability, which helps short-lived flows to be treated more fairly with fewer dropping packets by router. Although preferential treatment for short flows has been accomplished, SHRED still has the instability problem on instantaneous queue length and lots of parameters to control since it is rooted in RED.

2.3.4 RIO and RIO-PS

Both RIO and RIO-PS are extensions of RED. RIO (RED routers with In/Out bit) [CW97] is based on the idea of tagging packets as either in or out and treating them differently based on the tags. RIO uses the same mechanism as in RED but is employed with two sets of parameters for dropping packets, one set for in packets and the other set for out packets. Upon each packet arrival at the router, the router checks whether the packet is tagged as in or out. If it is an in packet, the router calculates the average queue length (avg_in) for in packets. If the arriving packet is an out packet, the router computes the average queue length (avg_total) for all both in and out arriving packets. The probability of dropping an in packet depends on avg_in, and the probability of dropping an out packet depends on avg_total. As in RED, the three parameters the minimum threshold (min_in), the maximum threshold, and the maximum drop probability (P_max_in) for in packets defines normal operation [0, min_in], congestion avoidance [min_in, max_in], and congestion control [max_in,∞] phases for in packets. Similarly, three corresponding phases for out packets are defined. With these two sets of parameters and phases, RIO decides whether or not it drops a packet. By classifying packets into in and out, RIO discriminates against out packets in times of congestion.

RIO-PS (RIO with Preferential treatment to Short flows) [GM01] is inspired by RIO. RIO-PS works with edge routers maintaining all the per-flow information and core routers managing per-class flows. Specifically, a counter in an edge router tracks how many packets have been currently transferred for each flow. If the counter exceeds a certain threshold, called minimum threshold (MinThresh), RIO considers the flow to be long and classifies the packets as long packets. This method classifies packets within the threshold as packets from a short flow. To detect the end of a flow, the per-flow state information is maintained and updated periodically. If no packets from the flow is observed in the period of time units, a core router considers the flow terminated and removes its information entry. The threshold can be a static or dynamic value. In RIO-PS, per-flow state information is used to adjust the threshold dynamically to balance the number of short and long flows. That is, the ratio of the number of short flows to that of long flows is controlled by an edge router.

2.3.5 PI (Proportional Integrator)

While RED uses average queue length, PI (Proportional Integrator) [HMT$^+$01] uses instantaneous queue length and regulates queue length to a desired queue reference value (qref). The drop probability of PI is proportional to queue length mismatches. The difference between the current queue length and a desired target queue length, and difference between a previous queue length and a desired target queue length determines drop probability and the drop probability is accumulated. That is, weighted subtraction of the previous queue mismatch from current queue mismatch is added to the previous drop probability. If the result of the subtraction is positive, drop probability gets larger than previous drop probability and smaller otherwise. Figure 2.3 gives the basics of the PI algorithm.

```
PI coeffieients : a = 0.00001822, b = 0.00001816

Once every period
   calculate probability P with qref and qlen
   P =  a * ( qlen – qref ) - b * ( qold – qref )  +  Pold
  drop(P)

  Pold = P
  qold = qlen
```
Figure 2.3 PI Algorithm

qref is a desired queue reference value and qlen is an instaneous queue length. P and qlen are saved for the next PI drop probability calculation. By adjusting drop probability based on queue length, PI keeps queue length close to the desirable target queue length and maintains a stable queue length. Moreover, it prevents the queue in a router from overflowing. However, PI can result in low queuing delay at the expense of large number of packets dropped. As shown in [HMT$^+$01], PI shows better utilization and lower queuing delay.

However, preliminary PI simulations run early in this investigation demonstrate that PI overacts when there are many flows by dropping many packets. That is, to keep the queue length at a targeted queue reference value, PI drops more packets than other AQM schemes. Moreover, while drop probability is computed at each packet arrival epoch in other AQM schemes, PI uses a frequency rate to change the drop probability once per period. Hence during this period all flows have the same drop probability instead of having the drop probability that reflects the characteristics of the flow, which does not help fairness. Since the link capacity remains constant, drop probability must be increased. By dropping, queuing delay become lower and round trip time delay gets reduced. However, for PI this results in higher numbers of retransmissions due to timeouts. This should yield higher transmission completion time for PI.

Due to lack of time marks were not considered for new algorithms. PI with ECN [HMT$^+$01] shows that the performance of PI is better than that of RED in high utilization and low delay but it may not result in more efficient performance with dropping, especially in the case of short lived flows.

2.3.6 REM

REM (Random Exponential Marking) [ALL$^+$01] maintains a variable, price to measure congestion. Price is updated periodically and determines the marking probability based on rate mismatch and queue mismatch. Rate mismatch is the difference between input rate and link capacity, and queue mismatch is the difference between queue length and target. When either the input rate exceeds the link capacity or the queue length is greater than target queue length, the weighted sum is positive. When the number of users grows, the input rate mismatch and queue mismatch increases, raising price and finally marking probability. When the source rates are small, the mismatches become negative, reducing price and marking probability. REM stabilizes queue length around a small target.

2.3.7 AVQ

As a rate-based scheme, AVQ (Adaptive Virtual Queue) [KS01] maintains a virtual queue whose capacity is less than the actual capacity of the link and updated by link utilization and packet arrival rate. On a packet arrival, the packet is marked if it overflows the virtual buffer and is enqueued in the virtual queue otherwise. The motivation behind AVQ is that when the link utilization is below the desired utilization, the virtual queue increases and marking gets less aggressive. Otherwise, the virtual queue decreases and marking gets more aggressive. AVQ regulates utilization instead of queue length as RED, PI, and REM and governs the queue without an explicit drop probability unlike the other AQM schemes.

2.4 Modeling Web Traffic

Web traffic models used in AQMs that attempt to treat HTTP flows differently from FTP flows are surveyed in this section.

2.4.1 Web Traffic Model in SHRED

The Web traffic simulator used in SHRED [HCK02] uses the built in Pareto II function [NS201] to generate Web reply object size with the Pareto shape parameter of 10 Kbytes, the maximum object size of 2 Mbytes, and the minimum object size of 12 bytes. The model sends Web pages, composed of multiple Web objects, to a traffic sink and waits for an amount of time determined by page generation rate before sending the next page. It modeled HTTP version 1.0 where each object in the Web page is a separate TCP connection. Unlike the standard HTTP procedure that the client first requests the primary container page and then subsequently issues separate requests to transfer each embedded object, all the objects in a Web page in this model are downloaded concurrently.

2.4.2 Web Traffic Model in RIO-PS

The Web traffic model used in RIO-PS randomly selects clients to initiate sessions to reflect surfing several Web pages of different sizes of randomly chosen Web sites. It models HTTP 1.0 such that each page containing several objects requires a TCP connection for delivery. To request a page, the client sends a request packet to the server, the server responds with an acknowledgement and then start to transmit the web page requested by the client. Exponential distributions are used for interpage and interobject arrivals and bounded Pareto distribution is used for object size with a shape parameter of 1.2.

2.4.3 ON/OFF Pareto

Another way to construct Web traffic is to use the Pareto On/Off Traffic [NS201]. This model is an application embodied in the OTcl class Application/Traffic/Pareto of NS-2. It generates traffic according to a Pareto On/Off distribution. Packets are sent at a fixed rate during “ON” periods, and no packets are sent during “OFF” periods. Both on and off periods are taken from a Pareto distribution with constant size packets. A Pareto On/Off traffic generator can be created with the following NS settings.

```
set p [new Application/Traffic/Pareto]
$p set burst_time_ 500ms
$p set idle_time_ 500ms
$p set rate_ 200k
$p set packetSize_ 210
$p set shape_ 1.5
```

Figure 2.4 Pareto Parameter Setting

Burst_time is a mean burst time, idle_time is a mean idle time, rate_ is a sending rate during burst transmission, packetSize_ is a fixed application packet size, and shape_ is Pareto shape parameter. Given the mean burst time and the Pareto shape parameter, the next burst length in units of a packet is computed. Using the mean idle time and the Pareto shape parameter, the next idle time is computed and the generator goes to sleep for the next idle time. This procedure is repeated.

The PI paper [HMT$^+$01] use this On/Off Pareto to model HTTP flows in their experiments. As preliminary experiments with Pareto On/Off Pareto were run, it was observed that a new TCP connection is not established for each new on burst transmission and is not disconnected when the transmission is completed. Moreover, the congestion window is not reset to one for a new burst transmission, which does not realistically model HTTP behavior. The modeling of timeouts is not completely accurate during off time periods. In this study, cwnd needs to behave as it would for real Web traffic so that AQMs developed in this investigation can apply an accurate cwnd.

After reviewing, the Web traffic models above it was decided that a more accurate and realistic Web traffic model was needed for this investigation. In the version 1.0 of HTTP traffic modeled in this study, each web object is transferred with a separate TCP connection and the cwnd of each new connection is always set to an initial value of 1. AQMs in this study use cwnd to classify short-lived flows from long-lived flows. Hence, correctly modeling cwnd is important to this investigation. To deal with the timeouts properly, a Web object transmission time is used instead of the Pareto on time.

Chapter 3 Design of Web Traffic Model and Two AQM Algorithms

The goal of this thesis is to investigate new congestion control algorithms at core routers that cooperate with TCP sources to provide good performance, and fair treatment for long and short flows when congestion occurs at bottlenecked links. To reach the goal two new algorithms and a simple Web traffic Model were developed. This chapter describes a simple Web traffic model and introduces two new AQM techniques.

3.1 Web Traffic Model

While FTP traffic can include very large files, HTTP traffic typically consists of relatively small objects embedded in a Web page. HTTP traffic is classified as short-lived TCP flows. Typically a short TCP flow has less than 20 packets to transmit [GM01]. Mah [MB97] reports the maximum object sizes are rather large (over 1 MB) and the mean object reply size between 8 and 10 KB are much larger than the median object reply sizes. Mah claims that these characteristics of Web object size distributions are consistent with heavy-tailed (with a large amount of the probability mass in the tail of the distribution). He found the distributions of Web object sizes above 1KB are reasonably well-modeled by Pareto distributions with Pareto shape parameter ranging from 1.04 to 1.14. Guo and Matta [GM01] uses a bounded Pareto function with shape parameter of 1.2 to generate Web replies.

The Web traffic model developed for this investigation models HTTP 1.0 that opens and closes a new TCP connection for each object embedded in a Web page and one object per a Web page. Each TCP connection is established resetting cwnd to 1. Once a transmission of a single object is completed, TCP disconnects. The variable size of an object is randomly generated by Pareto II function in NS-2. The Web traffic model implements ontime and offtime. Ontime is the time taken to transfer a single object and offtime is the object interarrival time. Object size distribution is modeled using a Pareto distribution because it has been shown that object size distribution are heavy-tailed.

While offtime in SHRED uses exponential interarrival times, the model in this study uses deterministic interarrival time. This makes it easier to analyze the impact of many HTTP flows. It is important to model many HTTP flows which more accurately affects the real world.

In the first version of the simple Web traffic model, clients sent a HTTP request to a Web server and then the Web server responded to the client and sent the object requested in the same way standard HTTP does. This is a more than realistic model of Web traffic than the newer version. However, as we ran experiments with this model, the amount of Web traffic transmitted for the whole simulation time was small and not enough to compare performance with FTP traffic. Even though the number of HTTP flows were increased from 50 to 100 and the number of FTP flows were decreased to 10, the load generated by the Web traffic did not still reach a sufficient amount needed for investigation. Because of the time taken by clients and the server to establish each TCP connections, actual HTTP transmit rate for the simulation duration was too small. To produce more HTTP traffic in the limited time, the Web traffic model was modified to send only one object per connection from the servers to the clients without modeling the HTTP request-response mechanism.

3.2 PISA

The AQMs investigated in this thesis are PISA (PI with Short-lived flow Adjustments) and PIMC (PI with Minimum Cwnd) based on PI. PISA (PI with Short-lived flow Adjustments) decouples short-lived flow adjustments (SA) from SHRED and employs it with PI. The initial thought was that PISA would be an improvement over PI, but PIMC was also evaluated as an alternative scheme.

To create PISA, the cwnd source hint and average cwnd calculation from SHRED is added to PI to create PISA. A TCP source adds a hint of its current congestion window size to the packet. When the packet arrives at a PISA router, the source hint is taken from the IP header in the same manner SA does. The hint is used with average cwnd to classify short-lived flows and long-lived flows. The ratio of current cwnd to average cwnd classifies short-lived flows and differentiates treatment from long-lived flows. If the congestion window is greater than average congestion window PISA increases the PI drop probability by as much as the ratio. Otherwise, it yields a lower probability than the PI drop probability. The drop probability is calculated based on the ratio of cwnd to average cwnd and PI drop probability. The PI drop probability is computed every period and the PISA drop probability is calculated by referring to the PI drop probability, a global variable, on each packet arrival.

```
PI coefficients : a = 0.00001822, b = 0.00001816

Once each period
  calculate drop probability, P, with qlen (instaneous
queue length), qref, qold (previous queue length), and
Pold (previous PI drop probability) :
  P = a * ( qlen – qref ) - b * ( qold – qref ) + Pold
  Pold = P;
  qold = qlen;
```

Figure 3.1 PI Algorithm

```
PISA coefficient : α is in rage {0.1, 3.0}

for each packet arrival
    updateAvg(cwnd)
    calculate probability Psa with probability P:
    Psa = α * P * (cwnd / cwnd_avg)
    If (Psa > 1) Psa = 1
    drop(Psa)
```

Figure 3.2 PISA Algorithm

PISA always refers to the same value as the PI drop probability within a period. Figure 3.1 and 3.2 presents the PISA algorithm. The important idea to note is that the PISA drop probability, Psa, is not saved by PI in Pold. This means that PISA will be a little weaker at keeping the queue size close to qref.

α is a PISA parameter to determine how much of the ratio is applied in adjusting the PI probability, P, to yield the PISA drop probability Psa. α can vary between 0.1 and 3.0 Preliminary experiments show that PI reduces cwnd under congestion which causes the ratio of cwnd and average cwnd to be smaller. To make bigger impact of the ration, 3 is selected for α in this investigation. α should not be zero because α of zero makes PISA work as Drop Tail. In PISA, packets of short-lived flows are dropped less than those of long-lived flows. The average cwnd used in this scheme is a weighted average that weighs cwnds of new arrival packets and the previous cwnd average with a weight parameter. Based on the parameter, the average cwnd can be controlled to reflect the impact of a new cwnd on the weighted average. A stable value of the weight parameter, 0.002, is found in SHRED experiments [HCK02] and after several preliminary PISA experiments the value of 0.02 was selected because the average cwnd value does not fluctuate as much with this setting.

$$\boxed{cwnd_{avg} = ((1.0 - weight) * cwnd_{avg}) + (weight * cwnd_{new})}$$

Figure 3.3 Weighted Cwnd Average

3.3 PIMC

Most of the time, HTTP, short-lived flows, have only a few packets per object and thus these flows reach only a small TCP congestion window (cwnd) size for each connection. Flows with a small number of packets have no available sampling data to estimate an appropriate RTO value for the first control packets such as SYN, SYN-ACK, and the first data packet. With this characteristic of short-lived flows, losing SYN or SYN-ACK packets costs an initial timeout (ITO) value as RTO. This large timeout period decreases throughput. If cwnd is less than four, a dropped packet will be unable to trigger three duplicate ACKs for fast retransmit. Thus in this situation, a packet loss will always require a timeout. This causes the flows to experience longer response times and yield high delays. For these reasons, flows need a minimum size of four for cwnd to use fast retransmit instead of RTO. For MinCwnd of PIMC, MinCwnd should be greater than 4, the minimum cwnd size for fast retransmit. Moreover, if one of the last three packets of short-lived flows is dropped, the TCP source does not send enough additional packets to trigger three duplicate ACKs and an RTO occurs.

PIMC uses the same cwnd source hint as PISA but does not compute an average cwnd. To keep the performance of all TCP flows high, PIMC has a cwnd threshold. If the cwnd is less than the threshold, a packet is not dropped. Otherwise, a packet gets dropped based on the PI drop probability. PIMC does not drop packets whose cwnd is less than a threshold, MinCwnd, which causes the queue length to grow. However, the PIMC drop probability for packets with cwnd exceeding a threshold is the same as the PI drop probability without a new value for qold. By doing so, packets with a small cwnd are protected from dropping so that high throughput is expected. PIMC uses PI coefficients, a = 0.00001822 and b = 0.00001816, implemented in PI experiments [HMT+01]. Figure 3.4 gives the PIMC algorithm.

```
for each packet arrival
   if (cwnd <= MinCwnd) then
      enque packet
   else
     use PI calculation to decide whether to drop a packet
     Pmc =  a * (qlen – qref) – b * (qold – qref ) + pold
```

Figure 3.4 PIMC Algorithm

Starting with MinCwnd at 4, preliminary experiments were conducted for a MinCwnd = 4,5,6, and 7. Figure 3.4 shows that 95% of objects have a 7 delay second in an experiment with a MinCwnd of 7 while 88% of objects have an object delay of 7 seconds with MinCwnd of 4. Median of object delay with MinCwnd of 7 is 0.42, which is slightly higher than median of 0.40 of object delay with MinCwnd of 4 in Table 3.1. The experiment with MinCwnd of 7 shows better results in object delay, so 7 was selected as the value for MinCwnd.

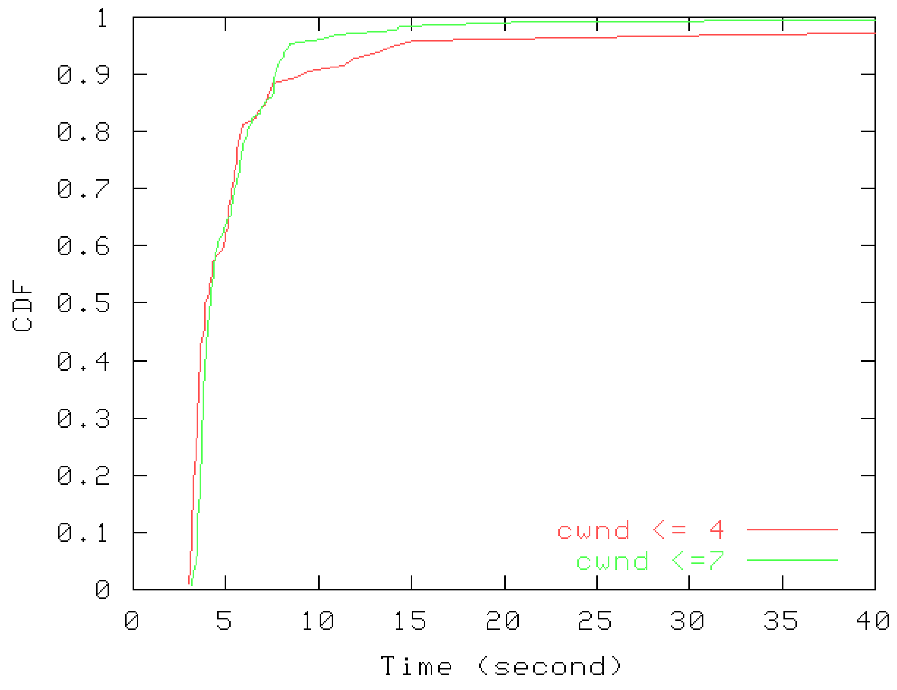

Figure 3.5 CDF of Object Delay

MinCwnd	Drop rate (packets/second)	Median object delay
4	7.286	0.40
7	7.126	0.42

Table 3.1 MinCwnd

Chapter 4 Experimental Methodology and Tools

This chapter includes NS-2 (Network Simulator), simulation input and output traces, data extraction and analysis, experimental setup and validations, and simulation scenarios with simulation network topology and simulation design specification

The experiments in this study are performed through procedure. The simulation script written in OTcl is run in NS-2 and trace files are generated as a result of the simulation. The data is extracted from the trace files and is plotted in a graph to analyze the performances. The simulation process is presented in Figure 4.1

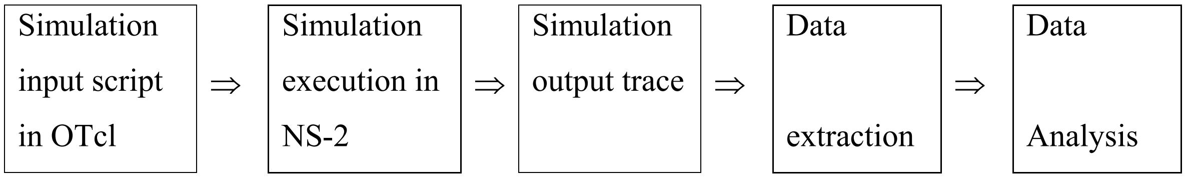

Figure 4.1 The Simulation Procedure with NS-2

For the simulation processes, the Network Simulator is used.

4.1 NS-2 Network Simulator

The Network simulator version 2 (NS-2) [NS201], written in C++ and Otcl, is an object-oriented, and discrete event driven network simulator. NS-2 developed as the VINT (Virtual InterNetwork Tested) project at Lawrence Berkeley National Laboratory (LBL), Xerox PARC, the University of California, Berkeley, and the University of Southern California/ISI [NS201] and is mainly used in the network research community. NS-2 simulates a variety of IP networks and including network protocols such as TCP, and UDP (User Datagram Protocol), traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanisms such as Drop Tail, RED, PI and AVQ. NS-2 also supports simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

4.2 Simulation Input

To run a simulation in NS-2, configuration and behaviors expected to be simulated are described in the form of an OTcl script. Basically, the topology is defined and nodes, agents, applications are instantiated and attached in the input script. These simulation objects in OTcl script are mirrored in classes in C++, the compiled hierarchy. The applications such as FTP, and Telnet traffic sources and the traffic distributions such as CBR (Constant Best Rate), Pareto, and exponential are specified. The start time and end time of the simulation are set in the script. During the simulation time, the events are generated and scheduled by time. Each event includes a packet arrival from a source to a router queue, drop, enqueue, dequeue, an arrival at a destination, a generation of an ACK packet, and timeouts. The input script also sets trace files keeping track of packets and other specific information such as instantaneous queue length.

4.3 Simulation Output Traces

The simulation is traced during the simulation time by using trace objects and monitor objects. The monitor objects collect data for basic information about the simulation. For example, the monitor objects are implemented as counters to count total number of packets, drops, and bytes received. In contrast, the trace objects collect the data for specific information. It keeps track of packets in the process of transmission and contains event number, time, source node, destination node, packet type, packet size, flow id, source address, destination address, sequence number and packet id for each packet arrival at a queue in a router, drop or en-queue, de-queue and departure. In this study, packet-based traces are needed to understand the simulation comprehensively so the data is collected by using the trace object. An output trace generated by the trace object in NS-2 has a fixed format shown in Figure 4.2

r	:	receive	(at to_node)
+	:	enqueue	(at queue)
-	:	dequeue	(at equeu)
d	:	drop	(at queue)
	:	src addr	node.port (ex.3.0)
	:	dest addr	node.port (ex. 0.0)

event	time	From node	To node	Pkt type	Pkt Size	flags	fid	Src addr	Dest addr	Seq num	Pkt id
r	10.000512	0	56	http	1040	-------	119	57.0	56.0	1	2911
+	10.000512	56	0	ack	40	-------	119	56.0	57.0	1	2911
+	10.002041	1	17	ack	40	-------	8	16.0	17.0	18	2871
-	10.002041	1	17	ack	40	-------	8	16.0	17.0	18	2871
+	10.002114	0	2	tcp	1040	-------	1	3.0	2.0	19	2454
-	10.002114	0	2	tcp	1040	-------	1	3.0	2.0	19	2454
r	10.006009	85	1	http	1040	-------	133	85.0	84.0	25	2878
r	10.006286	85	1	http	1040	-------	133	85.0	84.0	26	2879
r	10.006681	18	0	ack	40	-------	9	18.0	19.0	19	2880
r	10.00853	1	15	ack	40	-------	7	14.0	15.0	18	2843
+	10.00853	15	1	tcp	1040	-------	7	15.0	14.0	23	2912
-	10.00853	15	1	tcp	1040	-------	7	15.0	14.0	23	2912
r	10.008832	0	138	http	1040	-------	160	139.0	138.0	7	2427
+	10.008832	138	0	ack	40	-------	160	138.0	139.0	7	2913
-	10.008832	138	0	ack	40	-------	160	138.0	139.0	7	2913

Figure 4.2 A Sample of NS-2 Output Trace

4.4 Data Extraction

Once the simulation is done, the traced data is extracted for computation of performance metrics. The data extraction modules developed in C and perl generate reports on utiliazation, drop rate, delay and other statistical data such as drop ratio and average congestion window size for each type of flows.

4.5 Data Analysis

The data extracted from the traced files are fed into the data analysis tools, Gnuplot and MS excel to produce graphs based on the data. The graphs show the performance of the simulation results clearly so that the performance metrics are compared and analyzed.

4.6 Experimental Setup and Validations

In Figure 4.3 and 4.4, packet drops and instantaneous queue length with PIMC are shown for 2400 seconds of an experiment. Table 4.1 presents the variances in the drop rate for PIMC over a variety of interval ranges. Notice that the variances are each 500 second interval range from about 1.70 to 1.81 for FTP flows and from about 0.61 to 0.70 for HTTP flows. The differences between the variances of the 500 second intervals are smaller than the differences in variance of the first five 100 second intervals. Table 4.1 shows that by 500 seconds of simulation, the variance has settled down. Thus, each simulation experiment in this investigation was run for 500 seconds.

Time Interval	FTP	HTTP
0 – 100 sec	2.037106	0.588126
100 – 200 sec	1.781901	0.866419
200 – 300 sec	1.557411	0.759637
300 – 400 sec	1.772006	0.434962
400 – 500 sec	1.758998	0.700268
0 – 500 sec	1.788308	0.668003
500 – 1000 sec	1.704418	0.703262
1000 – 1500 sec	1.814922	0.616865
1500 – 2000 sec	1.739448	0.675673
0 – 1000 sec	1.743351	0.686251
1000 – 2000 sec	1.773690	0.640179

Table 4.1 Variance of Packet Drop Rate

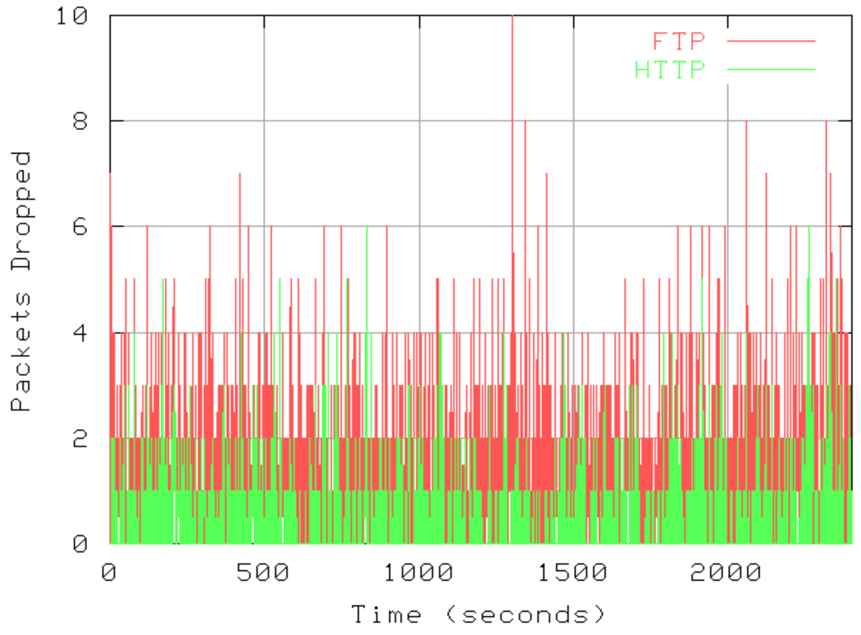

Figure 4.3 Packet Drops

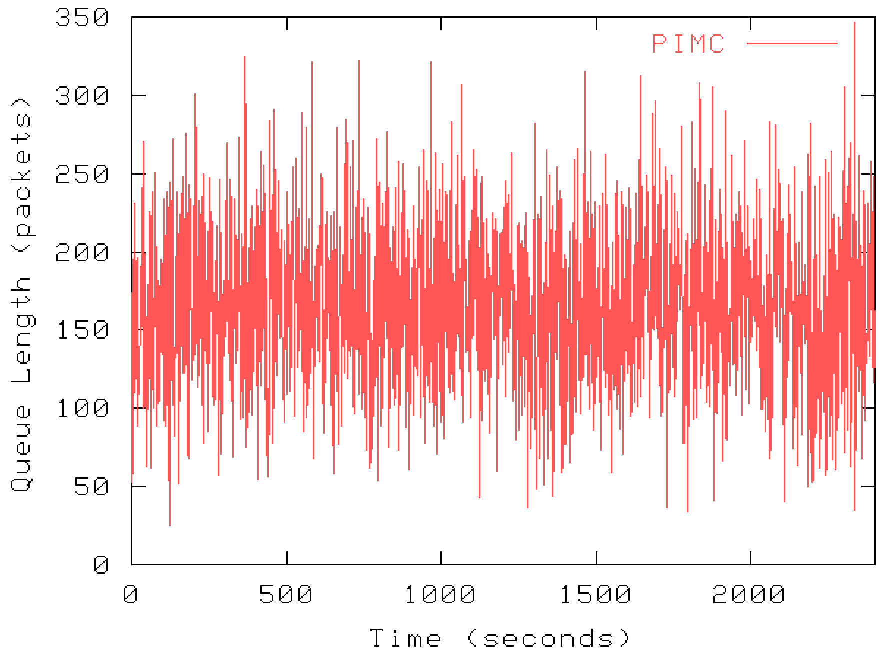

Figure 4.4 Queue Length

4.7 Simulation Scenarios

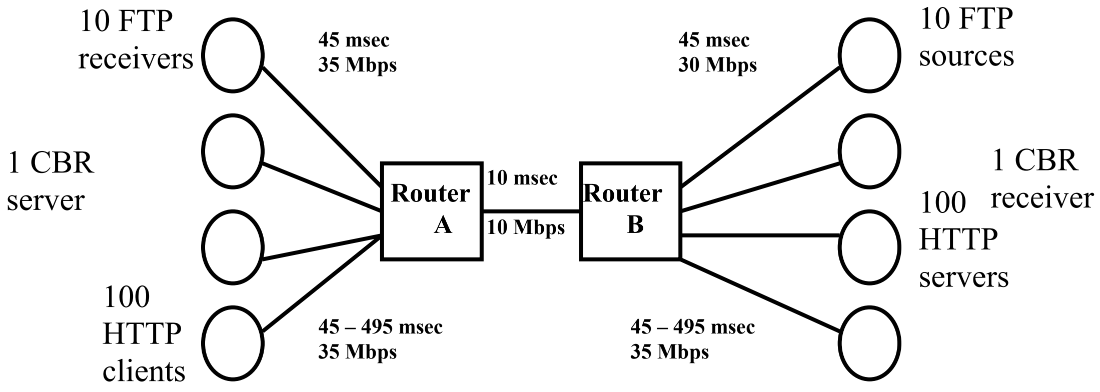

Figure 4.5 Standard Simulation Topology

The simulation network topology (see Figure 4.5) consists of two routers, a number of sources and sinks. The two routers, router A and router B are connected by a link of bandwidth of 10 Mbps and a propagation delay of 10msec. The link from router B to router A is managed by AQMs such as PI, PISA, and PIMC with a queue size of 800 packets and from router A to router B managed by Drop Tail with a queue size of 800 packets. 10 FTP receivers, 100 HTTP clients, and 1 CBR server are linked to router A at bandwidth of 35 Mbps and propagation delay of 45 ms. 10 FTP sources, 100 HTTP servers and 1 CBR receiver are linked to router B with a bandwidth of 30 Mbps and propagation delay of 45 ms. CBR traffic goes from router A to router B while FTP and HTTP traffic goes from router B to router A. 10 FTP flows and 100 HTTP flows travels on the topology. To make a realistic model of congestion on the bottleneck, different types of congestion such as reverse traffic of CBR is transmitted to create realistic congestion for ACKs. Through preliminary experiment with Reno and Newreno, it was shown that Newreno provided better performance. Thus TCP Newreno is used. The maximum cwnd is set unlimited, the default value of NS-2 and ssthreshold is initially set to 50 in all the experiments.

The Web traffic model in this experiment sets the maximum size of object to 2 Mbytes, average size of object to 10 Kbytes, referenced from SHRED [HCK02], and minimum size of object to 1 Kbyte in this experiment and uses 1.2 as the Pareto shape parameter based on findings in [GM01] and [MB97]. Offtime is set to 0.5 seconds and ontime is the one burst time taken to transfer one object.

Experiments in this investigation use exactly the same settings used in PI experiments [HMT$^+$01] to be fair. PISA and PIMC algorithms use PI coefficients $a = 1.822(10)^{-5}$ and $b = 1.816(10)^{-5}$, a queue size of 800, and a desired queue reference value of 200 packets. As we experimented with various values of α between 0 and 3, the value of 3 showed slightly better performance with a bigger impact on short-lived flows. These setups are for standard experiments and other simulations were run with heavier congestion with their setups (see section 5.6).

Chapter 5 Performance Evaluation and Analysis

In this chapter, the performance of PISA and PIMC is evaluated by comparison with PI and Drop Tail. PI is observed with Drop Tail, and the behaviors of PISA and PIMC are analyzed. Except for experiments under heavier congestion, all the other experiments are run with 10 FTP flows and 100 HTTP flows, link capacity of 10 Mbps, queue reference of 200 packets, and a queue size of 800 packets. This is the standard experimental setting.

5.1 PI compared to Drop Tail

PI is a stable AQM with low queue length and low delay, but it has high drops to keep the queue at the target queue reference. An experiment with PI and Drop Tail counts packets dropped per second and Figure 5.1 depicts that the number of packet drops of HTTP flows on PI is distinguishably higher than that of Drop tail. While the average drop rate of Drop Tail is almost zero, PI drops about 4.5 packets out of 1260.22 packets a second (0.38%). Dropping more packets is unfair to short-lived flows because dropping causes timeouts with short-lived flows.

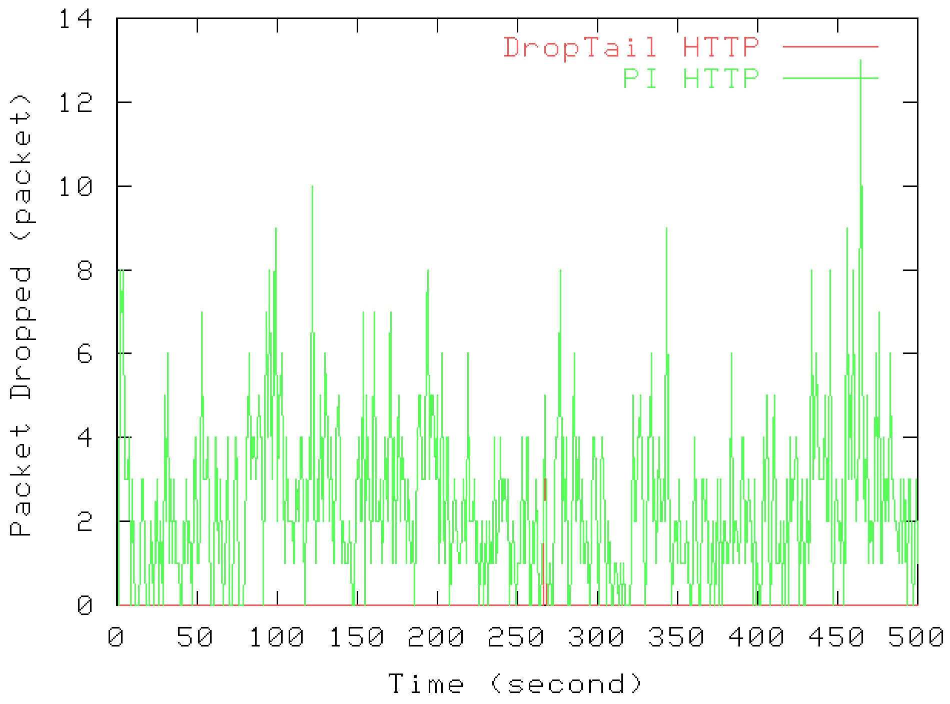

Figure 5.1 Packet Drops with PI and Drop Tail

	FTP flows (packets/sec)	HTTP flows (packest/sec)	All flows (packest/sec)
Drop tail	0.004	0.010	0.014
PI	2.024	2.412	4.436

Table 5.1 Average Drop Rate of PI and Drop Tail

5.2. Drop Rate

Drop rate is measured using the standard experimental setting. Figure 5.2, and 5.3 present the number of packet drops every second with PISA compared to PI. PISA drops FTP flows slightly more aggressively but HTTP flows less aggressively than PI, which helps HTTP flows to attain lower delay.

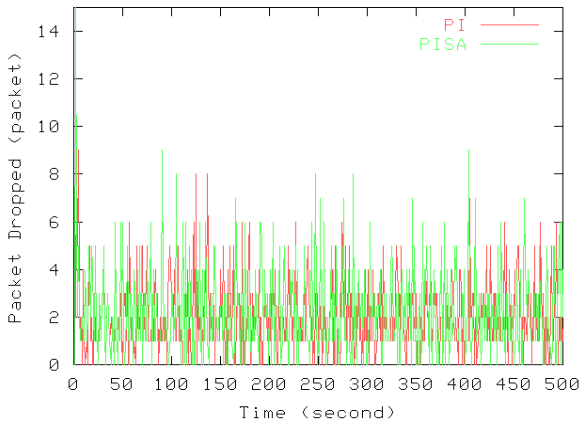

Figure 5.2 PISA FTP Flow Packet Drops with 10 FTP and 100 HTTP flows

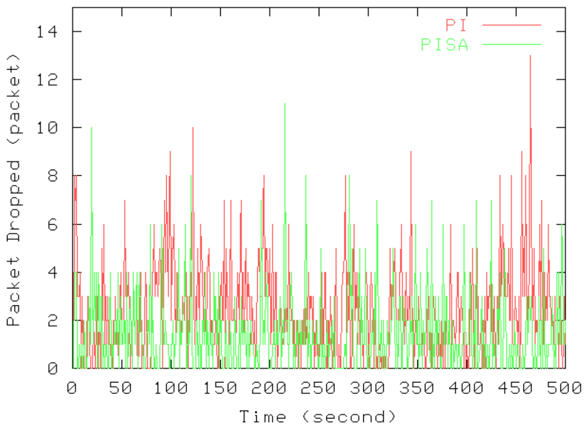

Figure 5.3 PISA HTTP Flow Packet Drops with 10 FTP and 100 HTTP flows

In Figure 5.4 and 5.5, PIMC drops both FTP and HTTP flows less than PI. While PIMC drops FTP flow packets at a similar rate to PI, for HTTP flows PIMC decreases the drop rate significantly. Since the drop rate decrement of HTTP flows is relatively higher than that of FTP flows, PIMC also helps HTTP.

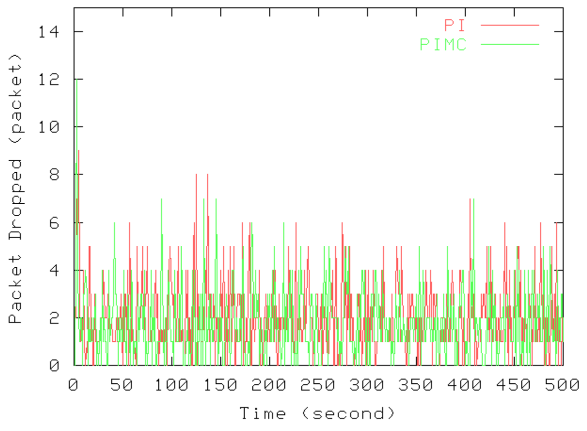

Figure 5.4 PIMC FTP Flow Packet Drops With 10 FTP and 100 HTTP flows

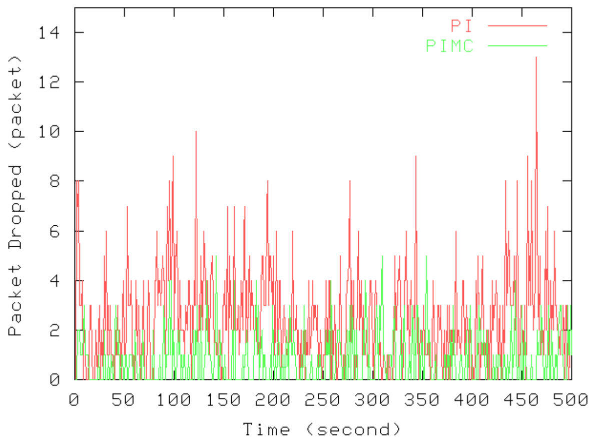

Figure 5.5 PIMC HTTP Flow Packet Drops With 10 FTP and 100 HTTP flows

As seen in Table 5.2, PISA compared to PI increases the drop rate of FTP flows by 0.2 packets per second and decreases that of HTTP flows by 0.7 packets per second. PIMC decreases drop rates of FTP and HTTP flows. The PIMC drop rate of HTTP is decreased by 65% of PI's drop rate. PISA has the smallest queue length. For average cwnd of FTP flows, PIMC has higher than PI and PISA and PISA is lower than PI. PIMC has a slightly higher average cwnd of HTTP flows than those of PI and PISA and an HTTP flow average cwnd of PISA higher than that of PI.

	FTP cwnd	HTTP cwnd	FTP packets dropped (packets/sec)	HTTP packets dropped (packets/sec)	ALL packets dropped (packets/sec)	FTP Drop Ratio (%)	HTTP Drop Ratio (%)	Average Queue Length (packet)
PI	24.99	11.20	2.024	2.412	4.436	0.35	0.35	197.01
PISA	23.95	13.83	2.258	1.468	3.726	0.35	0.22	162.16
PIMC	28.89	14.35	1.792	0.830	2.622	0.26	0.14	194.88
Drop Tail	193.15	20.52	0.004	0.010	0.014	0.00	0.00	505.01

Table 5.2 Average Cwnd and Drop Rate with 10 FTP and 100 HTTP flows

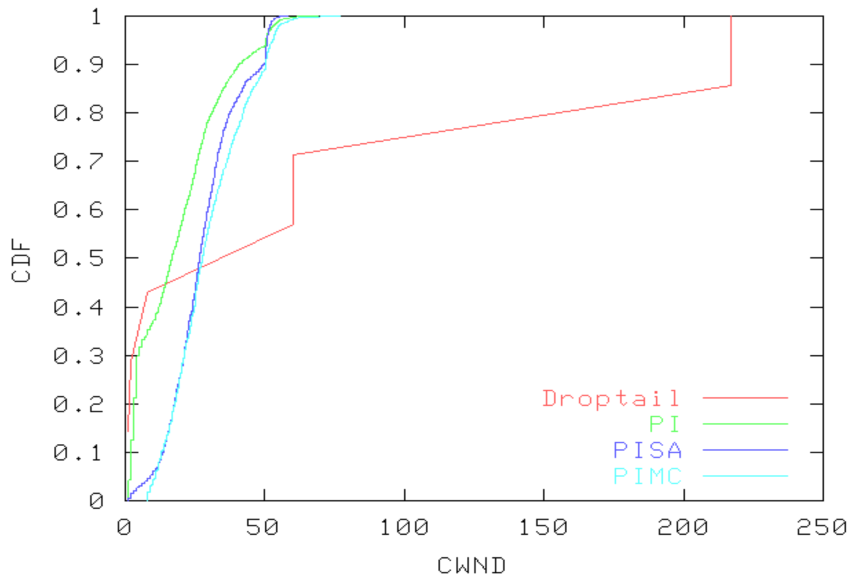

Figure 5.10 CDF of TCP CWND for when Packets are Dropped

As seen in Figure 5.10, PIMC does not drop any packets with cwnd less than MinCwnd of 7. 100 % of packets dropped have cwnd greater than 7. Since PISA does not differentiate packets with a fixed threshold such as MinCwnd, there are about 0.05% of packets dropped that have cwnd less than 7. Except for the 5% of all packets, PISA behaves similar to PIMC. In PI, more packets dropped have a smaller cwnd than other AQMs. Drop Tail has only 5 HTTP packet drops and 2 FTP packet drops that are presented at each point in Figure 5.10.

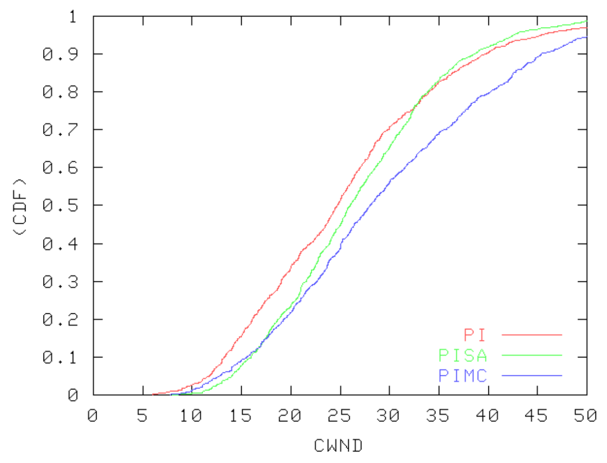

Figure 5.11 CWND of FTP Packets Dropped with 10 FTP and 100 HTTP flows

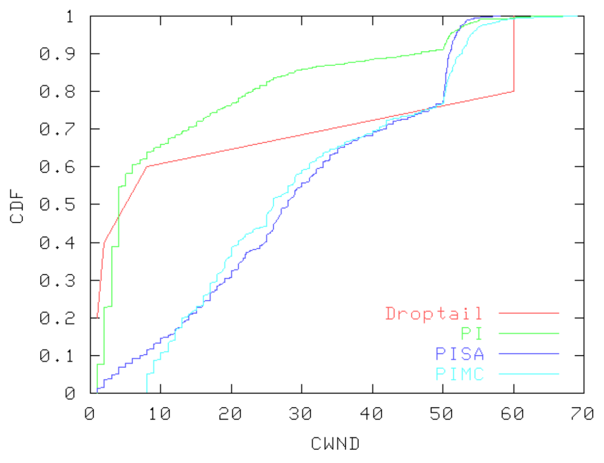

Figure 5.12 CWND of HTTP Packets Dropped with 10 FTP and 100 HTTP flows

The two FTP packets have cwnds of 217, much higher than that of other AQMs. For this reason, the cwnd of FTP packets dropped in Drop Tail are not included in Figure 5.11. The cwnd of Drop Tail includes only 5 packets dropped. Their cwnds are 1, 2, 8, 60.084805, and 60.101448 and the five points are connected in Figure 5.12 as a CDF. PI, PISA, and PIMC behave similarly and PIMC maintains slightly larger cwnd than others, which indicates that PIMC drops packets with larger cwnd more aggressively. Figure 5.12 implies that PISA and PIMC drop more HTTP packets with larger cwnd. This behavior should help improve the throughput for HTTP flows.

5.3. Queue Length

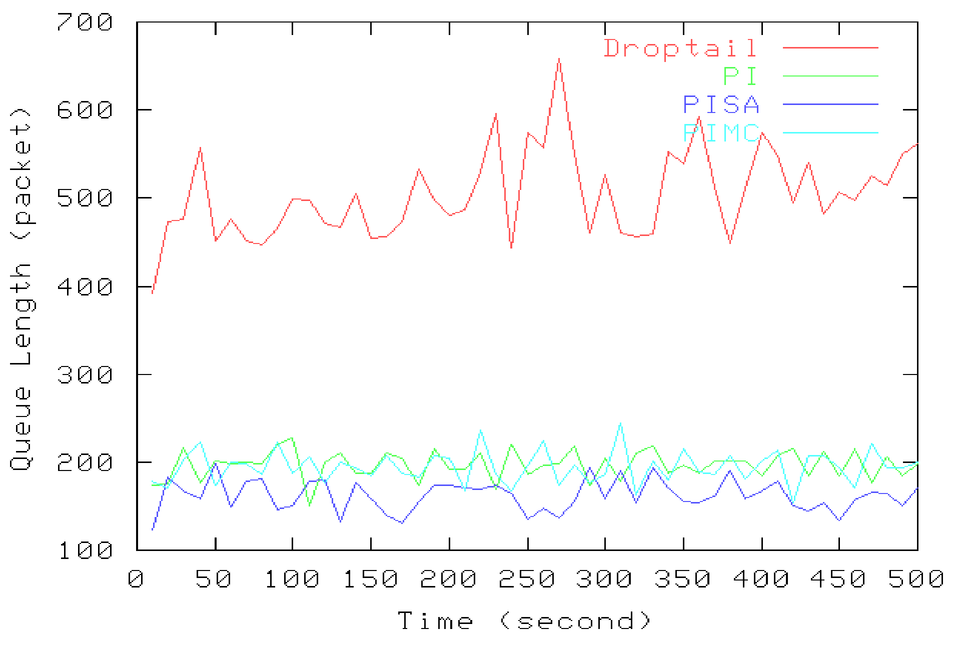

Figure 5.13 Queue Length

Figure 5.13 shows PI, PISA, and PIMC have stable queue lengths and PISA has a lower queue length than the other three algorithms. PISA drops fewer HTTP packets and drops more aggressively FTP packets. More packets belonging to FTP flows get sacrificed to keep the queue length stable. In PIMC, queue length is increased by packets with cwnd less than minimum threshold of 7 because they are always queued. If a number of packets with small cwnd arise fast, then the queue length grows fast. When packets with cwnd larger than minimum threshold come in, the queue length could be reduced because PIMC drops strongly the packets based on current queue length to keep the queue length close to the desired queue reference value.

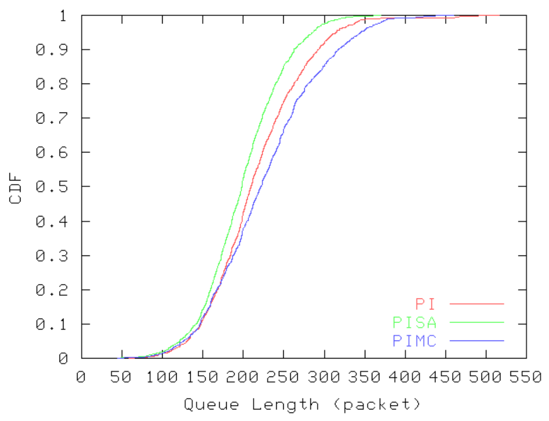

Figure 5.14 CDF of Queue Length of Packets Dropped with 10 FTP and 100 HTTP flows

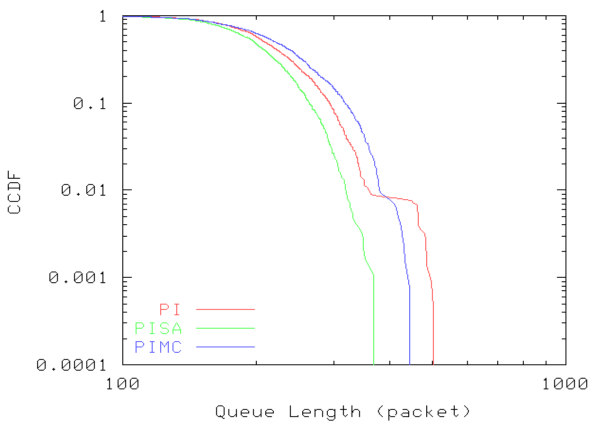

Figure 5.15 CCDF of Queue Length of Packets Dropped with 10 FTP and 100 HTTP flows

PISA keeps the queue length smallest of the three AQMs. 95% of packets' queue length is within 300 with PISA in Figure 5.14 and the largest queue length of packets dropped is smaller than those of PIMC and PI in Figure 5.15. With this smallest queue length, low delay is expected with PISA.

5.4 Packet Delay and Object delay

Figure 5.16 and 5.17 shows packet delays for PI, PISA, and PIMC for FTP and HTTP flows respectively. As expected from the previous section, PISA has the lowest packet delay for both FTP and HTTP flows. This is due to its exhibiting the smallest queue length. In contrast, Drop Tail has the longest packet delay for FTP and HTTP flows because of a large queue length. PIMC has similar behavior for packet delay with FTP flows to PI while PIMC has lower packet delay than PI for most of the HTTP flows.

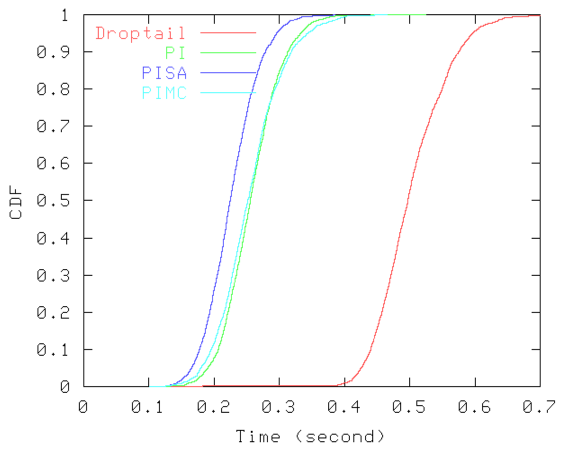

Figure 5.16 Packet Delay of a FTP flow with 10 FTP and 100 HTTP flows

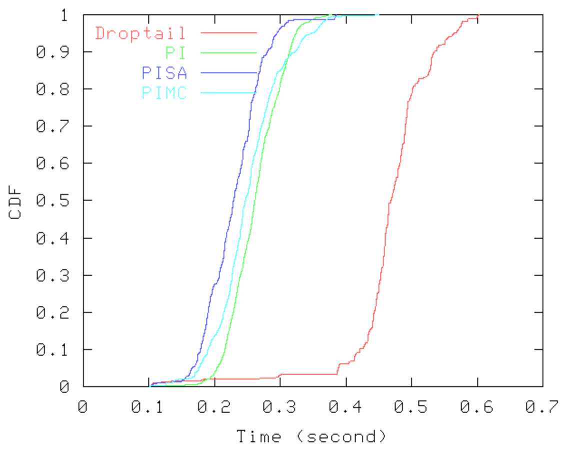

Figure 5.17 Packet Delay of a HTTP flow with 10 FTP and 100 HTTP flows

In addition to packet delay, PISA has the smallest object delay as seen in Figure 5.18. Specifically, 90% of packets' delay in PISA are within about 1 second. Because PISA drops fewer packets with a small cwnd, those packets can avoid time outs, which results in short response time and low delay. A few, heavy-tailed flows have higher delay with PISA in Figure 5.19.

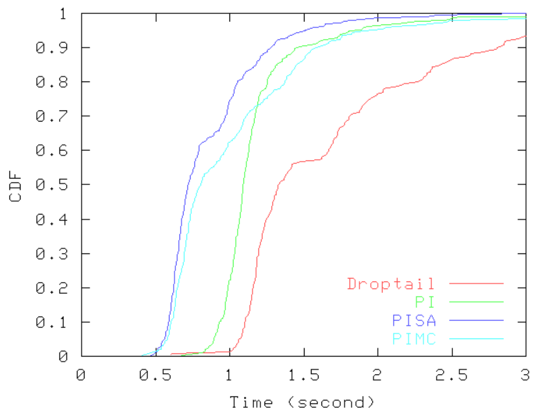

Figure 5.18 CDF of Web Object Delay with 10 FTP and 100 HTTP flows

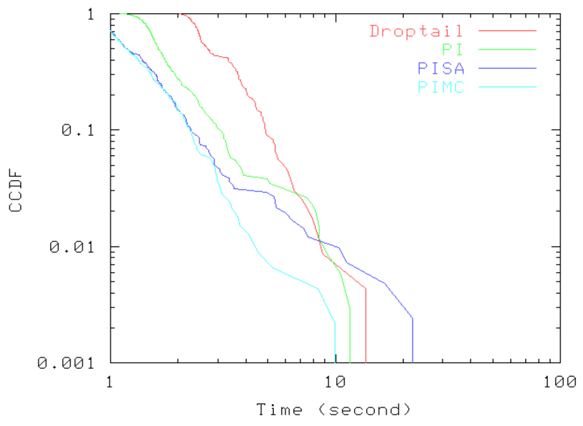

Figure 5.19 CCDF of Web Object Delay with 10 FTP and 100 HTTP flows

5.5 Utilization

As seen previously, PISA has a higher drop rate on FTP flows and a lower drop rate on HTTP flows compared to PI. Nonetheless, utilization of PISA for HTTP flows is lower than that of PI and utilization of PISA for FTP flows is higher than that of PI. Similarly, PIMC shows higher utilization with FTP flows and lower utilization with HTTP flows than PI although they drop fewer packets of both FTP and HTTP flows than PI. Utilizations of a link capacity of 10 Mbps with AQMs are shown in Figure 5.20, 5.21, 5.22, and 5.23.

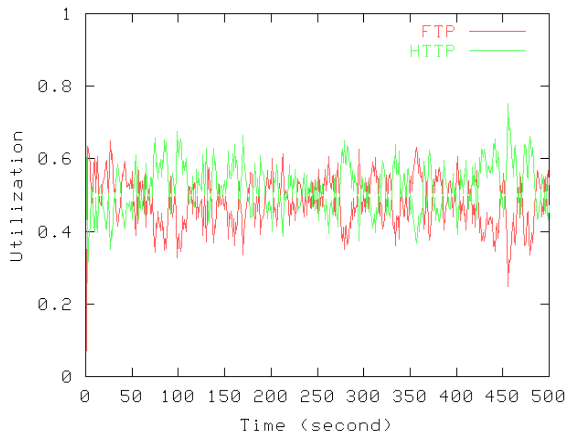

Figure 5.20 PI Utilization

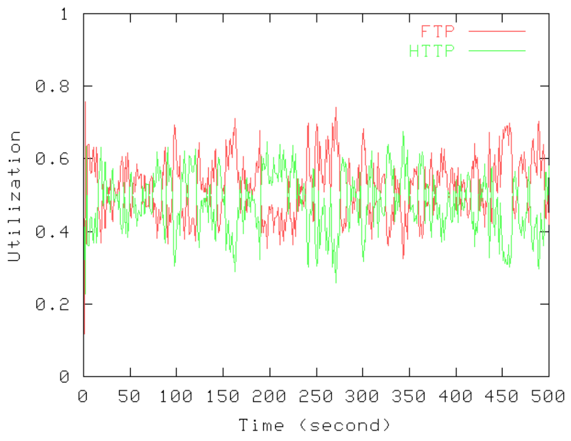

Figure 5.21 PISA Utilization

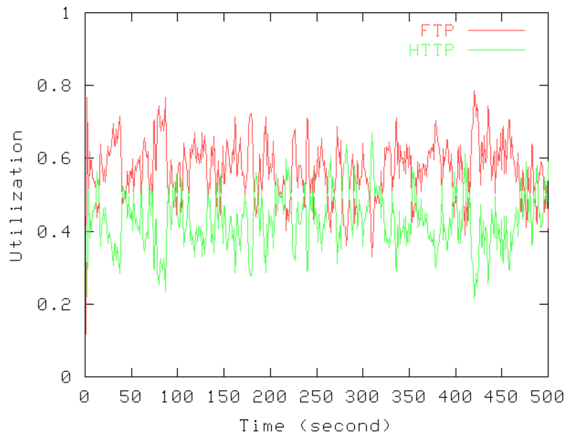

Figure 5.22 PIMC Utilization

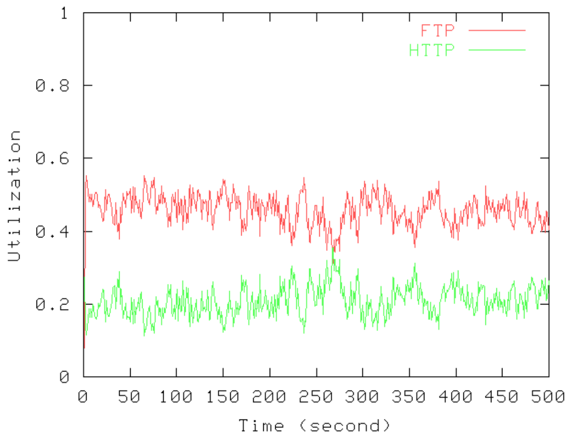

Figure 5.23 Drop Tail Utilization

	FTP flows	HTTP flows	Total flows
PI	0.479	0.520	0.999
PISA	0.522	0.477	0.999
PIMC	0.568	0.431	0.999

Table 5.3 Utilization of Link Capacity of 10 Mbps

Utilization is not a good measure to compare with FTP flows because HTTP flows spend time on connections and are idle for offtimes. Moreover, object delays are not much larger than the 0.5 seconds of idle time as shown in Figure 5.18. As seen in Table 5.4, PISA transmitted 6261 more Web objects than PI did in 500 seconds, which means PISA spends more time to establish 6261 more connections and idle 6261 times. This decreases utilization for HTTP flows and FTP flows gain more share of bandwidth. PIMC had 18% more objects and PISA had 22% more objects than PI in 500 seconds.

	Number of Object	Improvement
PI	28313	0%
PISA	34574	22%
PIMC	33357	18%

Table 5.4 Number of Web Objects transmitted during Standard Experiment

5.6 Experiments with Heavier Congestion

To investigate the behavior of the new AQMs when the bottlenecked link becomes more congested, two additional scenarios were simulated. Table 5.5 scenarios the setting used in these additional NS simulations.

	FTP flows	HTTP flows	Bandwidth	Queue Reference	Queue size
Increased FTP flows	50 flows	100 flows	10 Mbps	200 packets	800 packets
Reduced Bandwidth	10 flows	100 flows	5 Mbps	80 packets	320 packets

Table 5.5 Heavier Congestion Scenarios

5.6.1 Increased FTP flows

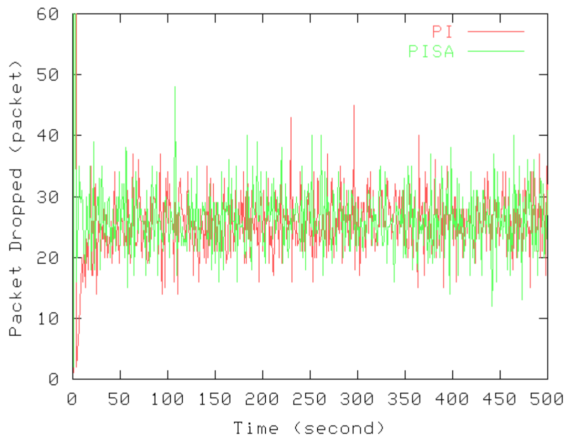

Figure 5.24 PISA FTP Flow Packet Drops with 50 FTP and 100 HTTP flows

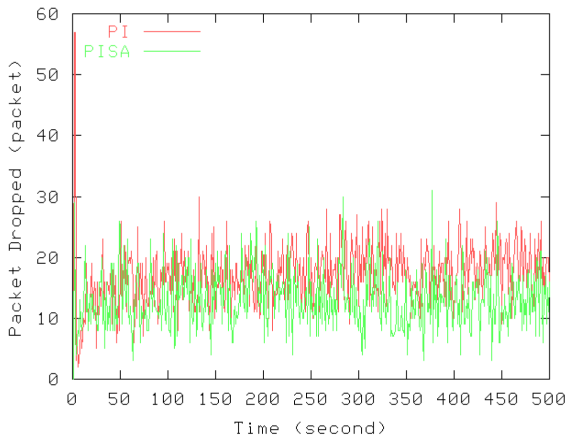

Figure 5.25 PISA HTTP Flow Packet Drops with 50 FTP and 100 HTTP flows

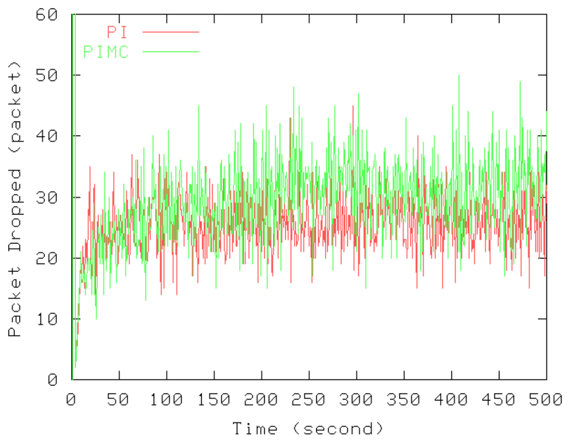

Figure 5.26 PIMC FTP Flow Packet Drops with 50 FTP and 100 HTTP flows

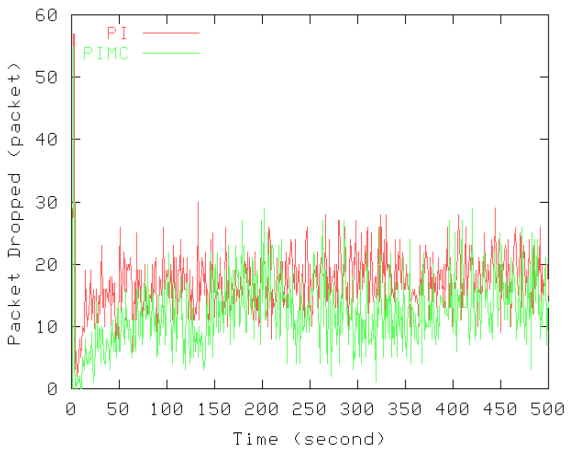

Figure 5.27 PIMC HTTP Flow Packet Drops with 50 FTP and 100 HTTP flows

By increasing the number of FTP flows from 10 to 50, PISA and PIMC drop more FTP packets than PI and fewer HTTP packets than PI. PIMC does perform better than PISA. Since most HTTP flows have a small cwnd, they are not dropped even in congestion so the drop rate of HTTP flows with PIMC is the lowest. For FTP traffic, PIMC has the highest drop rate. The increased number of FTP flows increases the queue length because PIMC does not drop initial packets belonging to the increased FTP flows. That is, no matter what type of TCP flows, increasing the number of TCP flows increases queue length. Increased queue length increases the drop probability of packets with cwnd larger than MinCwnd for PIMC.

	FTP cwnd	HTTP cwnd	FTP packet dropped (packets/sec)	HTTP packet dropped (packets/sec)	All packet dropped (packets/sec)	FTP Drop Ratio (%)	HTTP Drop Ratio (%)	Average Queue length (packet)
PI	7.72	5.47	26.654	17.166	43.820	3.4	3.3	207.93
PISA	7.34	6.03	27.038	13.084	40.122	3.4	2.6	202.41
PIMC	7.28	6.28	30.616	12.372	42.988	3.9	2.4	218.48
Drop Tail	21.78	11.44	8.440	6.862	15.302	0.3	0.8	691.05

Table 5.6 Average Cwnd and Drop Rate with 50 FTP and 100 HTTP flows on Capacity of 10 Mbps

PISA drops 0.5 more packets/second of FTP flows and 4 fewer packet/seconds of HTTP flows. PIMC drops 4 more packets/second of FTP flows and 5 fewer packets/second of HTTP flows. In total flows, PISA reduces 3.7 packets/second of drop rate and PIMC decreases 1 packet/second of drop rate.

The results in Table 5.6 are sums for all FTP flows and all HTTP flows. All AQMs with 50 FTP and 100 HTTP flows have higher drop rates than those with standard experiment. PISA with 50 FTP and 100 HTTP flows behaves similarly to PISA with standard experiment while PIMC with congestion behaves differently from PIMC with standard experiment. Unlike PIMC with standard experiment, PIMC with 50 FTP and 100 HTTP flows has a higher drop rate for FTP flows and a lower drop rate for HTTP flows. Although the queue length with 50 FTP and 100 HTTP flows increases, PISA still has the lowest queue length and queue length of PIMC is higher than that of PI. When compared to Table 5.2 increases of 5 times in FTP flows has higher rate more than 5 times for 50 FTP flows but more than 8 times for HTTP flows.

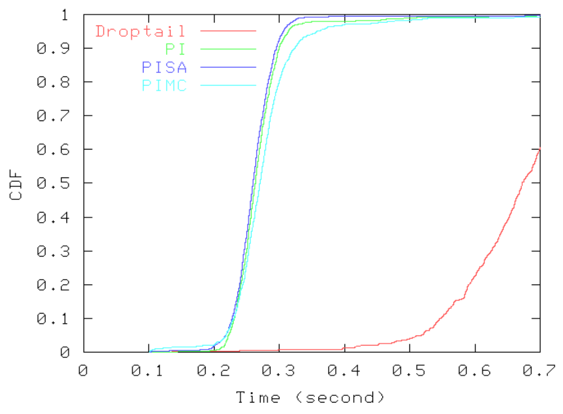

Figure 5.28 Packet Delay of a FTP flow with 50 FTP and 100 HTTP flows

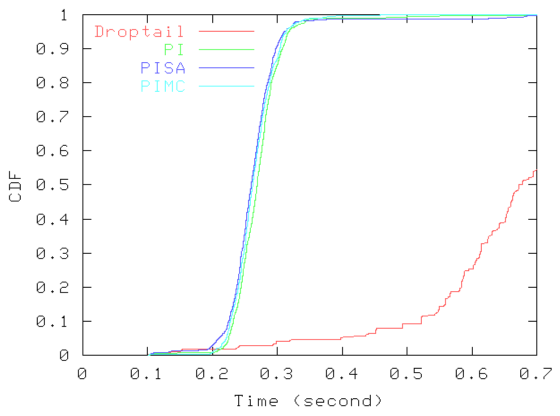

Figure 5.29 Packet Delay of a HTTP flow with 50 FTP and 100 HTTP flows

For packet delay, PISA has similar packet delay to PI for a FTP and a HTTP flow. 99% of FTP flow packets' delay with PISA and PI is within 0.3 seconds while 90% of those with PIMC is within 0.3 seconds. Packet delays with 50 FTP and 100 HTTP flows for both a FTP and a HTTP flow are higher than those with standard experiment. PISA with 50FTP and 100 HTTP flows still have the lowest packet delay among other AQMs. As analysis, it appears that packet delay improvement of PISA and PIMC is less compared to PI, but Drop Tail is terrible.

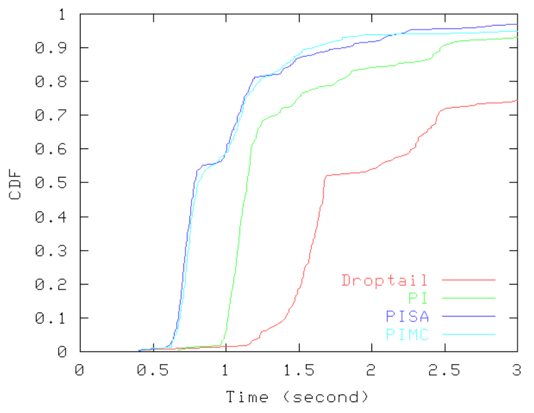

Figure 5.30 CDF of Web Object Delay with 50 FTP and 100 HTTP flows

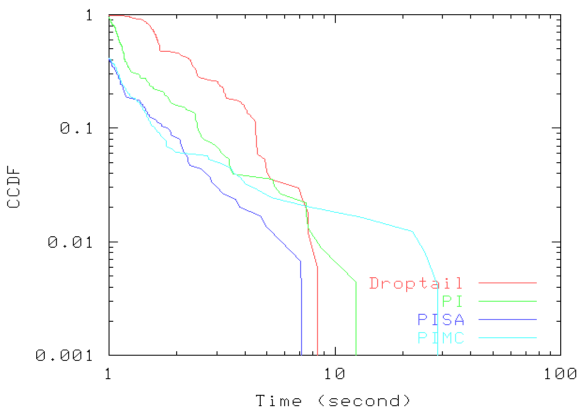

Figure 5.31 CCDF of Web Object Delay with 50 FTP and 100 HTTP flows

Object delays of PISA and PIMC are mostly smaller than PI and Drop Tail. PISA and PIMC have similar object delay. In Figure 5.30 70% of jobs with PISA and PIMC are transmitted within 1 second while the same percentage of jobs with PI is transferred within 1.2 seconds. PISA has the smallest object delay in heavy-tailed flows in Figure 5.31. Even though there are 40 more FTP flows, object delay of PISA with 50 FTP and 100 HTTP flows does not make much of a difference from PISA with standard experiment. For heavy-tailed flows, PISA with 50 FTP and 100 HTTP flows has the lowest object delay while PISA with standard experiment has a higher object delay than PIMC.

	Number of Object	Improvement
PI	20777	0%
PISA	27580	32.7%
PIMC	27597	32.8%

Table 5.7 Number of Web Objects transmitted with 50 FTP and 100 HTTP flows

The number of Web objects transmitted with 50FTP and 100 HTTP flows is reduced compared to that in standard experiment because of heavy congestion. Even under heavy congestion both PISA and PIMC are able to increase the object transmission rate by about 33%. PISA and PIMC improve the performance of HTTP flows about even.

5.6.2 Reduced Bandwidth

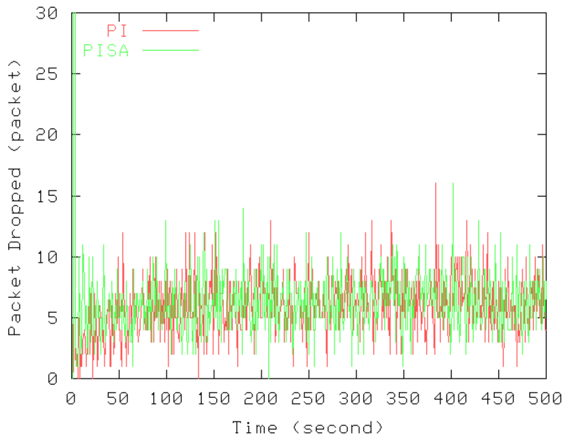

Figure 5.32 PISA FTP Flow Packet Drops with Capacity of 5 Mbps

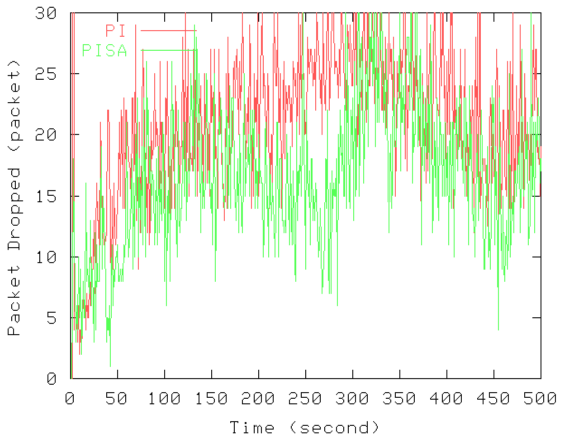

Figure 5.33 PISA HTTP Flow Packet Drops with Capacity of 5 Mbps

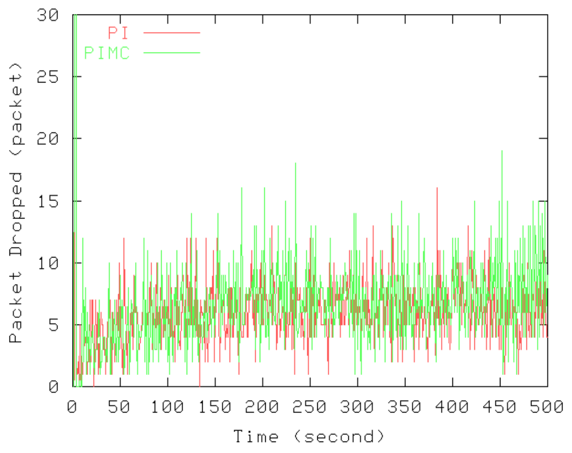

Figure 5.34 PIMC FTP Flow Packet Drops with Capacity of 5 Mbps

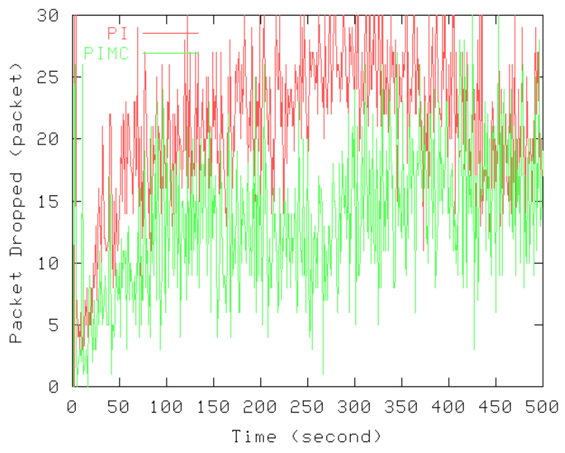

Figure 5.35 PIMC HTTP Flow Packet Drops with Capacity of 5 Mbps

	FTP cwnd	HTTP cwnd	FTP Packet dropped (packets/sec)	HTTP Packet dropped (packets/sec)	All Packet dropped (packets/sec)	FTP Drop Ratio (%)	HTTP Drop Ratio (%)	Average Queue Length (packet)
PI	7.36	4.94	6.188	21.148	27.336	4.1	4.1	87.53
PISA	7.06	5.68	6.356	16.614	22.970	3.9	3.2	81.90
PIMC	7.66	6.19	6.948	13.292	20.240	4.3	2.6	104.03

Table 5.8 Average Cwnd and Drop Rate with 10 FTP and 100 HTTP flows on Capacity of 5 Mbps

In the next set of simulations, the link capacity of the bottleneck is decreased to 5 Mbps with a queue size of 320 packets and the desired queue reference to 80 packets. In this scenario, PISA and PIMC decrease the drop rate of both FTP slightly and that of HTTP flows aggressively.

PISA increases drop rates of FTP flows by 0.17 packets/sec and decreases that of HTTP flows by 4.53 packets/second respectively. PIMC increases drop rates of FTP by 0.76 packets/second and decreases significantly that of HTTP flows by 7.86 packets/second respectively. For total flows, PIMC has the smallest drop rate. All AQMs with 5 Mbps link capacity have about three times higher drop rates than AQMs with standard experiment. PIMC with 5 Mbps link capacity drops more FTP flows than PI while PIMC with the standard experiment drops fewer packets from FTP flows than PI. PIMC has

20% more queue length.

For a FTP flow, PISA, PIMC and PI show similar packet delays. For a HTTP flow, PISA is slightly better than other AQMs. For both a FTP and a HTTP flow, 90% of packets' delays of PISA and PIMC with 5 Mbps are within 0.5 seconds while those with the standard experiment are within 0.3 so packet delay is increased by reduced bandwidth of 5 Mbps. PIMC behaves more similarly to PIMC with 5 Mbps than that with the standard experiment. Like the standard experiment with 10 Mbps, PISA has the lowest packet delay for the 5 Mbps link capacity.

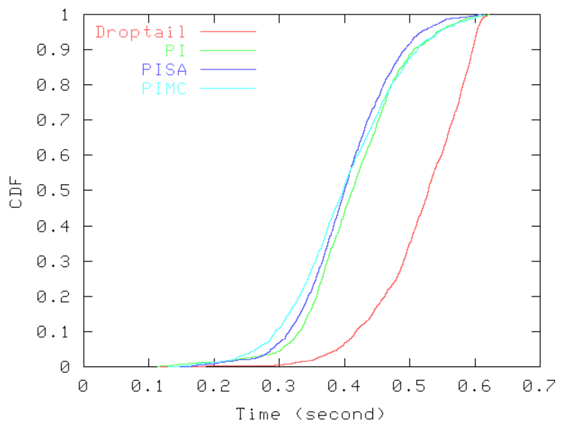

Figure 5.36 Packet Delay of a FTP flow with Bandwidth of 5 Mbps

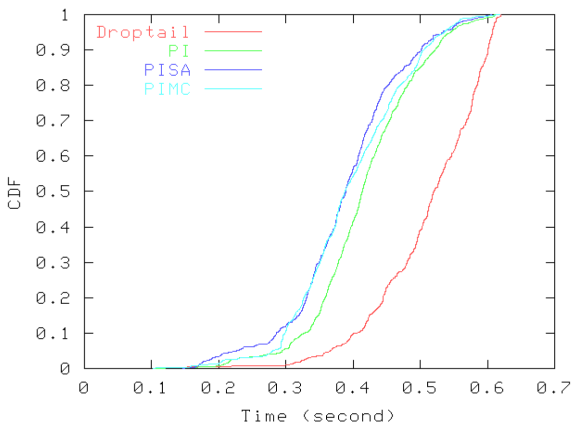

Figure 5.37 Packet Delay of a HTTP flow with Bandwidth of 5 Mbps

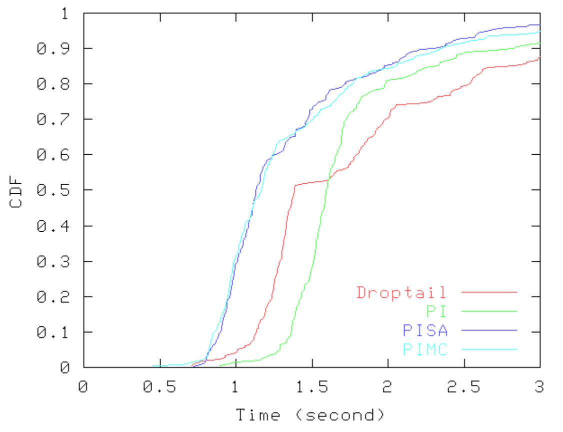

Figure 5.38 CDF Web Object Delay with Bandwidth of 5 Mbps

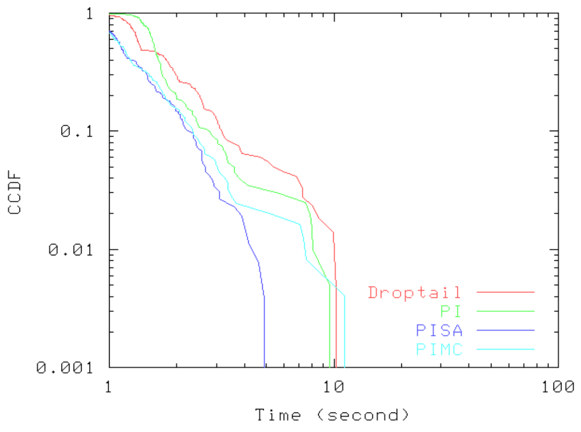

Figure 5.39 CCDF Web Object Delay with Bandwidth of 5 Mbps

PISA and PIMC behave similarly for object delay except for heavy-tailed flows. PISA is significantly better for heavy-tailed flows. All AQMs with 5 Mbps link capacity have longer object delays than that those with the standard experiment. In PISA and PIMC, 90% of Web object with 5 Mbps has object delay of 2.5 seconds while that with standard experiment the 90% performance has all 1.5 seconds. For heavy-tailed flows, object delay of PISA with 5 Mbps is less than 10 seconds while that of PISA with 10 Mbps is about 10 seconds.

	Number of Object	Improvement
PI	20256	0%
PISA	28644	41%
PIMC	28421	40%

Table 5.9 Number of Web Objects transmitted with 10 FTP and 100 HTTP flows on Capacity of 5 Mbps

Due to heavier congestion, the number of objects transmitted with capacity of 5Mbps is fewer than that with standard experiment. However, by avoiding timeout of dropped packets, PISA improves object sending rate significantly by 41% and PIMC does by 40%. Thus, PIMC and PISA outperform PI in a higher congested situation. PISA performs slightly better than PIMC.

5.7 Varying the RTTs of HTTP flows

In this set of simulation, 10 different groups of HTTP flows were run with 10 FTP flows with RTT of 200 ms. Each HTTP group has different RTTs ranging from 200 ms to 2000 ms. A group with RTT of 100 ms is called robust flows and the other group with RTT of 2000 ms is called fragile flows because robust flows send packets to the router much quicker than the fragile flows. Fragile flows need larger cwnd to send packets than robust flows. Most AQMs are inherently unfair to fragile flows without information on RTTs at routers. This experiment is run to see if PIMC with MinCwnd of 7 would be more unfair to fragile flows that are very far away versus PI and PISA.

The number of Web objects transmitted in each group is shown in Table 5.10. Robust flows with PIMC sends 0.3% more Web objects than PI and fragile flows with PIMC transfer 5.7% of total Web objects, the same as PI's and PISA's. PIMC does not make HTTP flow performance worse.

RTT	PI (Object)	PI(%)	PISA(object)	PISA(%)	PIMC(Object)	PIMC(%)
200 ms	3606	18.4	4614	18.7	4152	18.7
400 ms	2884	14.7	3710	15.1	3352	15.1
600 ms	2387	12.2	3019	12.2	2852	12.2
800 ms	2092	0.7	2573	10.4	2411	10.4
1000 ms	1783	9.1	2305	9.3	2101	9.3
1200 ms	1620	8.3	2039	8.3	1949	8.3
1400 ms	1462	7.4	1835	7.4	1734	7.4
1600 ms	1341	6.8	1589	6.4	1589	6.4
1800 ms	1210	6.2	1472	5.9	1447	5.9
2000 ms	1115	5.7	1410	5.7	1346	5.7
Total	19500	100.0	24566	100.0	22933	100.0

Table 5.10 The Number of Web Objects with of FTP RTT of 200 ms

RTT	PI (Object)	PI(%)	PISA(object)	PISA(%)	PIMC(Object)	PIMC(%)
200 ms	3603	18.6	4639	19.0	4153	18.0
400 ms	2841	14.6	3635	14.8	3380	14.6
600 ms	2370	12.2	2971	12.1	2868	12.4
800 ms	2078	10.7	2600	10.6	2413	10.4
1000 ms	1823	9.4	2253	9.2	2210	9.5
1200 ms	1623	8.3	1966	8.0	1926	8.3
1400 ms	1446	7.4	1823	7.4	1732	7.5
1600 ms	1316	6.7	1626	6.6	1583	6.8
1800 ms	1138	5.6	1514	6.2	1451	6.2
2000 ms	1121	5.7	1375	5.6	1331	5.7
Total	19359	100.0	24402	100.0	23047	100.0

Table 5.11 The Number of Web Objects with of FTP RTT of 2000 ms

Table 5.11 shows the experiment with 10 FTP flows with RTT of 2000 ms and 10 different groups of HTTP flows Each HTTP group has the same various RTTs as in previous experiment. Fragile flows of PIMC have 5.7 %, the same as that of PI and do not decrease the performance of fragile flows.

Chapter 6 Conclusions and Future Research

6.1 Conclusions

The majority of the traffic on the Internet is Web traffic, made up of short HTTP 1.0 connections that are short-lived flows. Since typical AQM schemes do not consider the duration of TCP flows, dropping is more likely to cause retransmission timeouts and high response time with short-lived flows. This thesis presents two AQMs, PISA (PI with Short-lived flow Adjustments) and PIMC (PI with Minimum Cwnd) based on PI to improve Web traffic performance. Using a cwnd source hint, PISA computes the ratio of current cwnd to average cwnd and adjusts the drop probability based on the cwnd ratio. Using cwnd as a minimum threshold, PIMC does not drop when a flow's cwnd is small. Simulations were conducted with a standard experiment setting and three other experimental setting to provide heavy congestion to evaluate the performance of PISA and PIMC versus PI and Drop Tail schemes.

The results of this thesis show that PISA performs better than PIMC and PI. Under moderate congestion, PISA sends 22% more objects with a drop rate of 3.73 packets/second and PIMC transmit only 18% more objects with a drop rate of 2.62 packets/second. Under heavy congestion conditions, PI drops packets at the rate of 27.34 packets/second. Since PISA only drops packets at 22.97 packets/second and PIMC only drops packets at 20.24 packets/second, PISA and PIMC are able to transmit HTTP objects at a significantly higher rate. PISA sends 41 % more objects while PIMC transmits 40 % more objects than PI. All of experiments show that PISA performs better with HTTP flows than PIMC and PI by avoiding timeouts for packet drops. Thus, this thesis recommends PISA for a HTTP flow friendly AQM.

6.2 Future Research

The basic algorithms of PISA and PIMC have been demonstrated in the simulation topology with FTP flows and HTTP flows along with reverse traffic of CBR for all 500 seconds. The investigation and development of the simple Web traffic model in NS for this research took much longer than was anticipated. Thus, this left much less time to conduct experiments. The following is a list of possible directions for future research.

1. To further investigate the performance of PISA and PIMC in more complicated network configurations.
2. To further refine PISA and PIMC by employing ECN. The expectation is that ECN will only improve PISA and PIMC even more than this study showed.
3. To evaluate the adaptability of PISA and PIMC to sudden changes in traffic profile. Rather than all flows starts at the same time and run together until the end of simulation, sudden drops or increases in the number of flows should be considered.
4. To refine the PISA algorithm for setting proper values of PISA's parameter α, which may need to be adjusted according to the number of flows.
5. To investigate the idea of using a variable MinCwnd in PIMC.
6. To investigate the period of PI drop probability computation.
7. To implement and study SA and MC with AVQ and REM.
8. To make object arrival times variable using exponential and other non-deterministic distribution.
9. To enhance the simple Web traffic model to be more realistic in terms of multiple number of objects per Web page and to consider container pages.

Bibliography

[ALL$^+$01]S. Athuraliya, V. H. Li, Steven H. Low and Qinghe Yin, “REM: Active Queue Management,” IEEE Network, Vol. 15, No. 3, May/June 2001, pp. 48 – 53.

[CJO01] M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith, "Tuning RED for Web Traffic," IEEE/ACM Transactions on Networking, Vol. 9, No. 3, (June 2001), pp. 249-264.

[CKS03] Félix Hernández-Campos, Kevin Jeffay, F. Donelson Smith, “Tracking the Evolution of Web Traffic”, 1995-2003. MASCOTS 2003, pp 16-25

[CW97] D. Clark and W. Fang, “Explicit Allocation of Best-Effort Packet Delivery Service”, IEEE/ACM Transactions on Networking, Vol. 6, NO. 4, August 1998, pp.362 – 372.

[FJ93] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Tractions on Networking, Vol. 1 No. 4, August 1993, pp. 397 – 413.

[GM01] L. Guo and I. Matta, "The War between Mice and Elephants," In Proceedings of the 9$_{th}$ IEEE International Conference on Network Protocols, Nov. 2001.

[HCK02] M. Hartling, M. Claypool and R. Kinicki, "Active Queue Management for Web Traffic," Technical Report WPI-CS-TR-02-20, Worcester Polytechnic Institute, May 2002.

[HMT$^+$01] C. V. Hollot, V Misra, D. Towsley and W. Gong, "On Designing Improved Controllers for AQM Routers Supporting TCP Flows," IEEE/INFOCOM, Vol. 3, April 2001, pp. 1726 – 1734.

[JA91] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley and Sons, 1991.

[KS01] S. Kunniyur and R. Srikant, "Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management," ACM SIGCOMM, (San Diego California, Aug. 2001), pp. 123-134.

[MB97] B.A Mah, "An Empirical Model of HTTP Network Traffic," In Proceedings of INFOCOM, pp. 592-600, Apr. 1997.

[NS201] ns-2 Network Simulator. http://www.isi.edu/nsnam/ns, September 2001.

[NS2HTTP] HTTP/1.0 traffic Generator for ns-2. http://www.isi.edu/nsnam/ns/WWW.html, September 2001.

[PD00] L. L. Peterson and B. S. Davie, *Computer Networks, a Systems Approach*, 2_{nd} Ed., Morgan Kaufmann, San Francisco, 2000.

[WL00] Leon-garcia Windjaja, *Communication Networks*, 2_{nd} Ed., Mc GrawHill, Boston, 2000.

[WW02] C. Williamson and Q.Wu, "A Case for Context-Aware TCP/IP," 2002