Efficient Packet Discarding Schemes for TCP/IP Based Networks

by

June D. Yiu

S.B., Massachusetts Institute of Technology (1998)

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1999
[June 1999]

© June D. Yiu, MCMXCIX. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

ENG

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUL 15 1999
FROM MIT LIBRARIES
LIBRARIES

Author .....
Department of Electrical Engineering and Computer Science
May 21, 1999

Certified by .....
Kai-Yeung Siu
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Efficient Packet Discarding Schemes for TCP/IP Based Networks

by

June D. Yiu

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 1999, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In recent years, increasing demand of the Internet has resulted in scarcity of network resources, compromising performance of traditional, mission critical applications [4]. There is a tremendous need to devise mechanisms to control congestion, as well as provide differentiated services to satisfy the varied requirements of different users. In 1993, Floyd and Jacobson proposed a technique called Random Early Detection (RED) as a method for providing congestion avoidance at the gateway. RED has shown to work very well with TCP to maintain high throughput and low delay in the networks [6]. One drawback of RED is that the randomization algorithm is not amenable to simple hardware. In this thesis, we present several alternate packet discarding schemes that aim to emulate RED and have simple hardware implementation. Simulation results are presented to illustrate that our scheme, Periodic Discarding Scheme with Sampling (SPD), exhibits comparable performance to RED. In addition, we also present how packet discarding schemes can be employed in the routers to provide differentiated services. We consider two differentiated services models - End-to-End QoS and Hop-by-Hop CoS. End-to-End QoS supports differentiated services by regulating traffic at the edge routers, while the Hop-by-Hop CoS model controls traffic at the network core routers. Our simulation results show that with greedy sources, the End-to-End QoS model coupled with connection admission control performs better than the Hop-by-Hop CoS model. However, when using bursty sources, the Hop-by-Hop CoS model exhibits better performance than End-to-End QoS model.

Thesis Supervisor: Kai-Yeung Siu
Title: Associate Professor

Acknowledgments

I wish to express my greatest gratitude to my thesis advisor, Professor Kai-Yeung Siu, for his guidance, patience, and encouragement throughout my research. His technical insights and dedication have helped to bring about the completion of this thesis.

Many thanks to Yuan Wu, who has helped me tremendously with his technical knowledge and numerous research ideas.

Thanks to all the brothers and sisters in the Hong Kong Student Bible Study Group and Boston Chinese Evangelical Church for their continuous support in my daily walk of Christian life.

Thanks to the following people who have made my life at MIT enjoyable: to Vanessa, for being the greatest roommate and friend; to Connie, for bringing me laughter, sharing my burdens and being my best companion in many adventures; to Charlotte, for her prayer support and "JC" sessions; to Ching, Edmond and Hubert, for providing academic help and advice throughout my years at MIT; to Joyce and Xuxia, for being my Course 6 comrades for five years; to Edwin, for his love and patience.

To my dear sister, Elaine, for answering all my "emergency" phone calls. I am truly grateful for her love and understanding. She has always been the best advisor and listener in my life.

To my parents, for their continuous encouragement and unconditional love.

To my Lord Jesus Christ,

" It is God who arms me with strength and makes my way perfect."

Psalm 18:32

Contents

1 Introduction and Motivation **10**

1.1 Asynchronous Transfer Mode 10

1.1.1 UBR 11

1.1.2 VBR 11

1.1.3 CBR 11

1.1.4 ABR 12

1.2 TCP Congestion Control Mechanism 12

1.3 Congestion Avoidance Mechanism 14

1.4 Related Works 15

1.4.1 DECbit 15

1.4.2 Random Early Detection 15

1.4.3 Early Packet Discard 17

1.4.4 Generic Cell Rate Algorithm 18

1.4.5 Differentiated Services 18

1.5 Performance Matrix 19

1.5.1 Throughput 19

1.5.2 Fairness 20

1.6 The Simulator and Analysis Tool 20

1.6.1 USER 20

1.6.2 TCP 21

1.6.3 HOST 21

1.6.4 SWITCH 21

1.6.5 PPLINK 21
1.6.6 Configuration 21
1.7 Thesis Plan 22

2 Periodic Discard Algorithms **23**
2.1 Periodic Discard Schemes and its variations with FIFO queues 24
2.1.1 Periodic Discard Scheme 24
2.1.2 Periodic Discard with Sampling 24
2.1.3 Periodic Discard with Sampling - Improved Version 25
2.2 Periodic Discard Scheme with
Per-VC Technique 26
2.2.1 Scheme A 26
2.2.2 Scheme B 26
2.2.3 Scheme C 27
2.2.4 Scheme D 27
2.3 Simulation Model and Parameters 28
2.3.1 Schemes: PRED, SPD, SPDI (Bursty Sources) 28
2.3.2 Schemes: A, B, C and D (Greedy Sources) 30
2.4 Simulation Results 31
2.4.1 Simulation Results for Periodic Discard Schemes with FIFO queues 31
2.4.2 Simulation Results for Periodic Discard Schemes with Per-VC Technique 34
2.5 Concluding Remarks 35

3 Differentiated Services **47**
3.1 End-to-End QoS 48
3.1.1 GCRA 48
3.1.2 Boundary Provisioning 50
3.1.3 Interior Provisioning 50
3.2 Hop-by-Hop CoS 50

3.2.1 Boundary Provisioning 51
3.2.2 Interior Provisioning 51
3.3 Simulation Model and Parameters 51
3.3.1 End-to-End QoS without CAC 52
3.3.2 End-to-End QoS with CAC 54
3.3.3 Hop-by-Hop CoS 55
3.4 Simulation Results 56
3.4.1 End-to-End QoS without CAC 56
3.4.2 End-to-End QoS with CAC 57
3.4.3 Hop-by-Hop CoS 57

4 Conclusions **66**

List of Figures

1-1 A 10-VC peer-to-peer configuration 22

2-1 Average Queue Size vs. Dropping Probability 27

2-2 A 10-VC peer-to-peer configuration 28

2-3 Simple Periodic Discard Scheme 36

2-4 Periodic Discard with Sampling (Interval = 50ms) 37

2-5 Improved Periodic Discard with Sampling (Interval = 50ms) 38

2-6 Improved Periodic Discard with Sampling (Interval = 100ms) 39

2-7 RED: 10 TCPs, BW = 10Mbps 40

2-8 RED: 30 TCPs, BW = 150Mbps 41

2-9 SPD: 30 TCPs, BW = 150Mbps 42

2-10 Per VC Queue with PRED: Scheme A 43

2-11 Per VC Queue with PRED: Scheme B 44

2-12 Per VC Queue with PRED: Scheme C 45

2-13 Per VC Queue with PRED: Scheme D 46

3-1 A flow chart description of the leaky-bucket algorithm 51

3-2 A Network Configuration for Differentiated Services 52

3-3 End-to-End QoS without CAC: Greedy Sources 60

3-4 End-to-End QoS without CAC: Greedy Sources 61

3-5 End-to-End QoS with CAC: Greedy Sources 61

3-6 Hop-by-Hop CoS: Greedy Sources 62

3-7 End-to-End QoS with CAC: Bursty Sources 63

3-8 End-to-End QoS with CAC: Bursty Sources 64

3-9 Hop-by-Hop CoS: Bursty Sources 65

List of Tables

2.1 SPD for various different values of N_{low} and N_{high} 32

2.2 Periodic Discard Scheme and its variations (BW = 10 Mbps) 33

2.3 A throughput comparison of SPD and RED (BW = 150 Mbps) . . . 34

2.4 Periodic Discard Scheme with Per-VC technique (BW = 150 Mbps) . 35

3.1 Throughput for TCP traffic under End-to-End QoS without CAC (Greedy Sources) 58

3.2 Throughput for TCP traffic under End-to-End QoS with CAC (Greedy Sources) 58

3.3 Throughput for TCP traffic under Hop-by-Hop CoS (Greedy Sources) 58

3.4 Throughput for TCP traffic under End-to-End QoS (Bursty Sources) 59

3.5 Throughput for TCP traffic under Hop-by-Hop CoS (Bursty Sources) 59

3.6 Total Throughput in the Two DS Models 59

Chapter 1

Introduction and Motivation

Asynchronous transfer mode (ATM) has become a tremendously important technology as it emerged as the standard for supporting future Broadband Integrated Services Digital Network (B-ISDN). While ATM was originally conceived as a carrier of integrated services networks, most traffic carried over the Internet today is still data. Clearly, there is a need for ATM to support the huge legacy of existing data applications, in which transmission control protocol (TCP) is generally used as the transport layer protocol [9]. Hence, many studies have been done on TCP performance over ATM. In particular, extensive research has been done on traffic control in ATM networks to improve TCP performance. The objective of this research is to investigate various efficient packet discarding schemes for the gateways of ATM networks that aim to provide traffic control, and how these schemes can be used to support differentiated services.

1.1 Asynchronous Transfer Mode

ATM switching technology transmits all information in small, fixed-size packets called cells. The cells are 53 bytes long - 5 bytes of header followed by 48 bytes of payload. ATM networks are connection-oriented. When a connection is established, a route from the source machine to the destination machine is chosen as part of the connection setup and remembered. These end-to-end connections are called Virtual Circuits

(VCs). Cells that transmit along a VC will never arrive out of order. ATM Adaptation Layer (AAL) is the interface layer that support TCP/IP, current Internet protocol, over ATM switching technology. One function of ATM AAL is segmentation and reassembly. AAL breaks down variable length packets into fixed size cells at the source, and then cells are reassembled into packets at the destination. There are five service categories provided by ATM - UBR, VBR, CBR, and ABR [11].

which give gateway control mechanism a better view of the traffic sources [7].

1.1.1 UBR

The Unspecified Bit Rate (UBR) service is intended for data applications that want to use any available bandwidth and is not sensitive to cell loss or delay. UBR service provides no traffic related guarantees. Instead, it relies on the higher-level cell loss recovery and retransmission mechanisms, such as the window flow control employed by TCP. The network congestion control and flow control mechanisms deployed by TCP aim at using any available bandwidth. Hence, TCP can be easily adapted to the UBR service class. In Chapter 2, we will study packet discarding schemes that aim at improving the performance of TCP over UBR service.

1.1.2 VBR

The Variable Bit Rate (VBR) is divided into two subclasses, rt-VBR and nrt-VBR. The real-time VBR service category is intended for services that have variable bit rates combined with tightly constrained delay and delay variation, such as interactive compressed video. The non-real-time VBR service category is intended for non-real-time applications that have bursty traffic characteristics and required timely delivery.

1.1.3 CBR

The Constant Bit Rate (CBR) class is intended to emulate a dedicated transmission link. This class of service guarantees transmission of data at a constant bit rate. It

is used by connections that request a static amount of bandwidth that is available continuously during the time of connection.

1.1.4 ABR

The Available Bit Rate (ABR) is intended to allow a source to adjust its rate, based on the feedback from switches within the network [10]. This feedback is communicated to the source through control cells called Resource Management Cells, or RM-cells. It is expected that an end-system that adapts its traffic according to the feedback will experience a low cell loss ratio and obtain a fair share of available bandwidth according to a network specific resource allocation scheme [11].

1.2 TCP Congestion Control Mechanism

The basic strategy of TCP is to send packets without reservation into the network and then react to observable events that occur. TCP congestion control was introduced into the Internet in the late 1980s by Van Jacobson following a congestion collapse of the Internet. Congestion collapse occurs when an increase in the network load results in a decrease in the useful work done by the network [5]. It is caused by TCP connections unnecessarily retransmitting packets that are either in transit or has already been received at the receiver. Currently, the TCP congestion control mechanism has four intertwined algorithms: slow start, congestion avoidance, fast retransmit, and fast recovery [2]. In the TCP congestion control mechanism, two variables are maintained per-connection. The congestion window (*cwnd*) is a sender-side limit on the amount of data the sender can inject into the network before acknowledgment (ACK) is received, while the receiver's advertised window (*rwnd*) is receiver-side limit on the amount of outstanding data. In addition, a state variable, the slow start threshold (*ssthresh*, is maintained to determine whether the slow start or congestion avoidance algorithm is used to control data transmission. If *cwnd* is less than *ssthresh*, the slow start algorithm is used to increase the value of *cwnd*. However, if *cwnd* is greater than or equal to *ssthresh*, the congestion avoidance algorithm is used. The initial

value of *ssthresh* is set to the receiver's advertised window. Slow start algorithm is used by the data sender at the beginning of a transfer to avoid transmitting an inappropriately large burst of data [12].

Slow start begins by first initializing *cwnd* to one segment. Each time an ACK is received, the congestion window is increased by one segment. This continues until the value of *cwnd* is greater than or equal to *ssthresh*, or a loss of packets is detected. During slow start, *cwnd* doubled for every round-trip time (RTT) and therefore provides an exponential increase. When the value of *cwnd* is greater than or equal to *ssthresh*, the congestion avoidance algorithm is used to increase *cwnd*. This algorithm increases the size of *cwnd* more slowly than does slow start. During congestion avoidance, *cwnd* is increased by $1/cwnd$ for each receiving ACK. Therefore, if one ACK is received for every data segment, *cwnd* will increase by roughly 1 segment per round-trip time. Congestion avoidance is used to slowly probe the network for additional capacity [1].

TCP's default mechanism to detect lost segment is a timeout [10]. When the sender does not receive an ACK for a given packet within the expected amount of time, the segment will be retransmitted. The retransmission timer (RTO) is based on an estimation of the RTT. TCP uses a lost segment as an indication of congestion in the network. TCP is required to send an immediate acknowledgment (a duplicate ACK) when an out-of-order segment is received. The purpose of this duplicate ACK is to let the sender know that a segment is received out-of-order and which sequence number is expected. From the sender's point of view, a duplicate ACK can be caused by a dropped segment or just a reordering of segment. However, if three of more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost. We then retransmit the missing segment, using fast retransmit, without waiting for RTO to expire. The congestion window is adjusted using fast recovery. The *ssthresh* is first set to one-half of the value of *cwnd*. The value of *cwnd* is then set to *ssthresh* plus 3 times the segment size. For each duplicate ACK that has been received, the value of *cwnd* is inflated by 1 segment. TCP is able to transmit new data when permitted by *cwnd*. Thus, TCP is able to keep sending data into the network

at half of its original rate. When an ACK for the retransmitted packet arrives, the value of *cwnd* is set back to *ssthresh*.

TCP handles congestion differently based on the way the congestion is detected. If the retransmission timer causes a packet to be resent, TCP reduces *ssthresh* to half the current *cwnd* and set the value of *cwnd* to one segment (entering slow start). However, if a segment is retransmitted through fast retransmit, *cwnd* is set to half the current value of *cwnd* and congestion avoidance is used to send new data. The difference reflects the fact that when retransmitting due to duplicate ACKs, packets are still flowing through the network and can therefore infer that the congestion is not very severe [2]. However, when a packet is resent due to the expiration of the retransmission timer, TCP cannot infer anything about the state of the network, and hence must continue conservatively by sending new data using the slow start algorithm.

1.3 Congestion Avoidance Mechanism

TCP provides a mechanism that controls congestion when it happens, as opposed to trying to prevent congestion in the first place. In fact, TCP increases the load it injects into the network until congestion occurs, and then it backs off from this point. In other words, TCP needs to create packet losses to probe the available bandwidth of the connection. A different approach is to predict when congestion is about to happen and then reduce sender's rate before packets start being discarded. We called this approach congestion avoidance, as distinguished from congestion control [10]. The most effective detection of congestion can occur in the gateway itself. Only the gateway has a unified view of the queuing behavior over time [6]. Hence, added functionality at the gateway can assist the end node in the anticipation of congestion. In our research, we will study several congestion avoidance mechanisms that use packet discarding algorithms.

1.4 Related Works

In this section, we will discuss several congestion avoidance mechanisms that have been proposed for packet-based networks. The techniques proposed for adaption of packet-based congestion avoidance mechanisms to ATM (cell-based) networks will also be discussed.

1.4.1 DECbit

DECbit is an early example of a congestion avoidance mechanism. DECbit gateway sends explicit feedback to end sources when the average queue size at the gateway exceeds a certain threshold. DECbit gateway gives feedback by setting a binary congestion bit in the packets that flow through the gateway. When a packet arrives at the gateway, the average queue length for the last (busy + idle) cycle plus the current busy cycle is calculated. The gateway sets the congestion bit in the header of arriving packets when the average queue length exceeds one. The source, then, uses window flow control, and updates its window once every two round-trip times. The window is decreased exponentially if at least half of the packets in the last window had the congestion indication bit set. Otherwise, the window is increased linearly. In DECbit scheme, there is no distinction between the algorithm to detect congestion and the algorithm to set the congestion indication bit. When a packet arrives at the gateway and the average queue size exceeds one, the congestion bit is set in the header of that packet. Because of this method of marking packets, DECbit can exhibit a bias against bursty traffic [6].

1.4.2 Random Early Detection

Random Early Detection gateway (RED) were introduced by Floyd and Jacobson in 1993. RED gateway uses an implicit signaling of congestion by packet dropping. Instead of waiting for the queue to become full and start discarding each arriving packet, the gateway decides to drop arriving packets with a drop probability calculated each time the average queue size exceeds a certain threshold. The average queue

size is calculated for each arriving packet using a low-pass filter:

$$avg := (1 - w_q) \cdot avg + w_q \cdot q \tag{1.1}$$

The queue weight w_q is a constant between 0 and 1, while q represents the sampled current queue size. The average queue size is calculated for comparison with the minimum threshold (min_{th}) and the maximum threshold (max_{th}). When the average queue exceeds the maximum threshold, all arriving packets are marked. On the other hand, no packets are marked if the average queue size is less than the min_{th}. If the average queue size is between min_{th} and max_{th}, each arriving packet is marked with probability p_a, where p_a is a function of the average queue size avg. Each time a packet is marked, the probability that a packet is marked from a particular connection is roughly proportional to that connection's share of the bandwidth at the gateway [6].

The calculations of the average queue size take into account the period when the queue is empty by estimating the number m of packets that could have been transmitted by the gateway during the idle period [6]

As avg fluctuates from min_{th} to max_{th}, packet marking probability p_b varies linearly from 0 to max_p:

$$p_b := \frac{max_p \cdot (avg - min_{th})}{(max_{th} - min_{th})}. \tag{1.2}$$

The final packet-marking probability p_a slowly increases as the *count* increases since the last marked packet:

$$p_a := \frac{p_b}{1 - count \cdot p_b} \tag{1.3}$$

Where *count* is the number of arriving packets since last marked packet when avg remained between the min_{th} and max_{th}. It is being set to zero every time a packet is marked. All marked packet will get dropped by the gateway.

The objective is for the gateway to drop packets frequently to control the average queue size, but at a regular interval to avoid global synchronization and biases [6].

Global Synchronization is a general network phenomena that results when all connections are notified to reduce their sending rates at the same time. The probability that the RED gateway selects a particular connection to notify during congestion is approximately proportional to that connection's share of the bandwidth at the gateway [6]. This approach avoids biases against bursty traffic. Since the rate at which the gateway marks packets depends on the level of congestion, global synchronization can be avoided. It has been shown that RED is able to improve network throughput and fairness in bandwidth utilization among contending TCP connections substantially [6]. One drawback of RED is that the randomization algorithm is not amenable to simple hardware. In our research, we are going to investigate various alternative packet discarding algorithms that have simple hardware implementation and exhibit performance comparable to that of RED.

1.4.3 Early Packet Discard

Though RED is designed for packet-based networks, we can easily adopt to ATM UBR service. Studies have shown that, however, UBR without any ATM layer congestion control mechanism yields unsatisfied performance. Much of the degradation is due to the transmission of corrupted packet due to cell losses. In view of this problem, Early Packet Discard (EPD) algorithm for UBR service has been proposed. The idea of EPD is to discard an entire packet before buffer overflows, such that the bandwidth is utilized for the transmission of only non-corrupted packets. In fact, EPD represents a general class of packet discarding algorithms, which can be applied to any packet-based protocol running over ATM. One problem of the EPD scheme is that it tends to allocate less bandwidth for connections with longer round-trip time and for connections traversing multiple congested nodes. Furthermore, connections with bulky data tend to get more bandwidth than highly bursty traffic connections under EPD.

1.4.4 Generic Cell Rate Algorithm

So far, we have discussed several congestion avoidance schemes that are mainly used to support ATM UBR service. The Generic Cell Rate Algorithm (GCRA), however, is a scheme that is used to support VBR service. VBR is an ATM service class that intend for application with bandwidth requirements that vary with time. With VBR traffic, it is generally not possible for the source to slow down, even in the event of congestion, due to the inherent real-time or semi-real-time nature of the information source [13]. Hence, for VBR service, a concrete traffic contract is negotiated before the connection is admitted. The traffic contract specifies a number of QoS parameters whose values are negotiated by the provider and the customer. The two parameters that GCRA has are Peak Cell Rate (PCR) and Cell Delay Variation Tolerance (CDVT). PCR is the maximum cell rate at which the source is planning to send. CDVT specifies how much variation will be presented in cell transmission times. GCRA works by checking every cell to see if it conforms to the parameters for its connection. GCRA can also substitute the parameter PCR with Sustained Cell Rate (SCR) to check for cell conformant to the mean cell rate. The algorithm details will be described in Chapter 3. We can use GCRA in the routers to support service differentiation.

1.4.5 Differentiated Services

The Internet Engineering Task Force has drafted a framework for providing differentiated services. In their framework, individual application flows are classified into different service classes by means of packet marking [3]. The packets are marked in the DS-field upon entry to a DS capable network. Classifiers separate traffic based on the DS-field of the submitted packets. The framework provides building blocks to support differentiated services. We are going to study and compare two differentiated services model that are based on this framework, End-to-End QoS and Hop-by-Hop CoS.

1.5 Performance Matrix

To evaluate the effectiveness of a resource allocation scheme, two broad metrics of measuring network performance are used - Throughput and Fairness.

1.5.1 Throughput

Network performance is optimized when the total throughput is maximized and the average queuing delay at bottleneck links are minimized [7]. The throughput of a switch is given by the number of bits that can be transmitted over the network in each second, while delay corresponds to how long it takes for a single bit to transmit from one end of a network to the other. Maximizing throughput and minimizing delay are two conflicting criteria that need to be compromised. To increase total throughput, we can inject as many packets into the network as possible to drive the utilization of all the links up to 100%. We want to avoid the possibility of a link becoming idle, because an idle link necessarily hurts throughput [10]. However, as the number of packets in the network increases, the length of the queues at each router also increases. Packets, in turn, are delayed longer in the network. To describe this relationship, power is proposed as a metric for evaluating the effectiveness of a congestion avoidance schemes. Power is defined as

$$Power = \frac{throughput}{delay} \tag{1.4}$$

Since we employ packet discarding algorithms in the switch, queue lengths at each routers are under control. Throughput becomes our major focus as we assess the performance of our congestion avoidance mechanism. In this thesis, we will refine our definition of throughput to be the number of packets that are delivered to the receiver, excluding duplicate packets.

1.5.2 Fairness

The fair allocation of resources depends mainly on users' demands. The simplest scenario is when users are able to receive equal shares of the bandwidth when they have equal demands [7]. Another valid definition would be that each user should be entitled some percentage of available bandwidth, according to the particular resource allocation policy.

Though fairness is usually specified qualitatively, a quantitative measure was proposed by Raj Jain that can be used to quantify the fairness of a congestion avoidance mechanism [8]. Given a set of TCP flows throughput (x_1, x_2, ... , x_n), Jain's fairness index is defined according to the following function

$$f(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2} \tag{1.5}$$

The fairness index always lies between 0 and 1. The fairness index is one when the throughput of all the flows are the same. In our analysis, we are only going to assess the performance of the congestion avoidance mechanisms qualitatively.

1.6 The Simulator and Analysis Tool

The simulations are done using the MIT Network Simulator (NetSim). NetSim is an event-driven simulator composed of various components that can send messages to one another. The set of components used are users, TCPs, hosts, switches, and links. Each component contains status indicators that can be used for performance assessment.

1.6.1 USER

The USER components simulate the source of a network connection. They generate data in burst or continuously for the TCP components. The USER's burst size and the spacing between consecutive bursts determine the network dynamics.

1.6.2 TCP

The TCP components simulate TCP reno version with fast retransmit and fast recovery. They keep track of their throughputs, retransmission percentages, and end-to-end delays. These parameters are used for analysis of the performance of TCP during network congestion.

1.6.3 HOST

The HOST components perform segmentation and reassembly of TCP packets. They have input and output buffers that help to absorb traffic transients.

1.6.4 SWITCH

The SWITCH components simulate ATM switching technology. It performs cell switching between their input and output ports. The SWITCH components contain traffic control algorithms for congestion avoidance. Most of our algorithms are implemented in the SWITCH components.

1.6.5 PPLINK

PPLINK components represent point-to-point links. The parameters associated with links are transmission and propagation delays.

1.6.6 Configuration

A sample network model using our simulation components is illustrated in Figure 1-1. First, the USER components generate data burst to the TCP components. The data burst formed packets which are then sent to the HOST components for segmentation. The ATM switches perform cell switching between their input and output ports. Then, on the receiving side, cells are reassembled into packets and passed back to the TCP components and finally to the USER components at the destination.

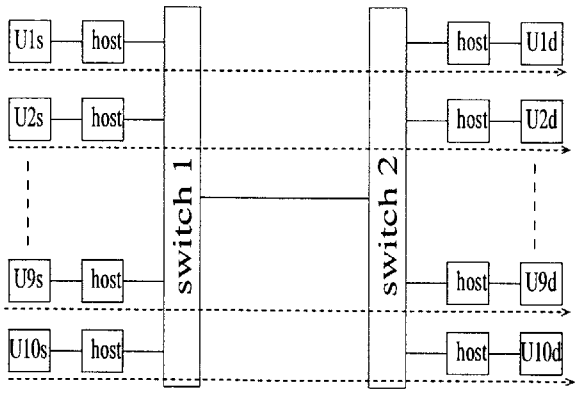


Figure 1-1: A 10-VC peer-to-peer configuration

1.7 Thesis Plan

This thesis is an analysis of performance of various packet discarding schemes used as a traffic controller at the gateway. The simulation studies are conducted using the MIT Network Simulator (NetSim). The network topologies and parameters used are described in details for each simulation.

Chapter 2 is an investigation on various packet discarding algorithms that have simple hardware implementation and exhibit performance comparable to that of RED. This report summarizes the results of our investigation toward this objective. We will first give a brief description of the various schemes we studied and then present and analyze the simulation results.

In Chapter 3, we will investigate two different differentiated services models - End-to-End QoS and Hop-by-Hop CoS. We will describe the packet discarding schemes that are used in these models to support service differentiation. We will then present the simulation results comparing the two models.

Chapter 2

Periodic Discard Algorithms

It is well known that when TCP flows go through a congested network, they will exhibit some undesirable synchronization problems that will significantly degrade the network utilization. In general, the synchronization problem occurs when TCP flows go through a bottleneck link, packets will get dropped when the buffer overflows, causing most of the TCP flows to go to the slow start phase simultaneously, during which the link will be underutilized until most of the TCP flows ramp up their rates to cause network congestion again. This synchronization of dynamics among the TCP flows will result in oscillation of the network load and underutilization of the network bandwidth. The problem becomes particularly severe in networks with long round trip time.

In view of this problem, Floyd and Jacobson proposed a technique called Random Early Detection (RED), which selects a random set of TCP packets to be discarded during period of congestion, thereby avoiding the synchronization problem. While their randomization technique improves network throughput substantially and also fairness in bandwidth utilization among contending TCP flows, the specific algorithm they proposed is not amenable to simple hardware implementation.

In this chapter, we will investigate various alternative packet discarding algorithms that have simple hardware implementation and exhibit performance comparable to that of RED. It summarizes the results of our investigation toward this objective. We will first give a brief description of the various schemes we studied and then present

the simulation results comparing their performances.

2.1 Periodic Discard Schemes and its variations with FIFO queues

2.1.1 Periodic Discard Scheme

Periodic Discard Scheme (PRED) is an algorithm that tries to emulate the techniques of RED without the explicit use of randomization. The simplest form to implement Periodic Discard is to set three threshold values: Minimum Threshold (min_{th}), Maximum Threshold (max_{th}), and Counter Threshold (N). A switch drops incoming packets whenever the buffer exceeds the given max_{th}. To be more specific, the first cell of any incoming packet will be discarded when the total queue size at a switch exceeds max_{th}. Once the first cell of a packet is discarded, the remaining cells of the packet will also be discarded, even when the switch queue is reduced to below the Maximum Threshold. However, a cell will not be discarded by the algorithm if the first cell of the same packet is not discarded, unless the entire buffer is full. The incoming packets will get marked when the buffer exceeds the given min_{th} but below the max_{th} threshold. We maintain a state variable, Packets Counter (X), that stores the number of incoming packets that are marked. When the counter exceeds the given counter threshold N, a packet will get dropped. The counter is being reset to zero whenever a packet gets dropped. Consequently, a packet will get dropped for every N marked packets.

2.1.2 Periodic Discard with Sampling

Periodic Discard with Sampling (SPD) is a slight variation of the Periodic Discard Scheme described previously. Instead of having one fixed Counter Threshold, we use different values for the Counter Threshold based on the average queue size. The queue size is sampled at regular time intervals specified by the parameter, Sampling

Interval (SI). The average queue size is then calculated using the sampled queue size as follows:

$$avg := (1 - w_q) \cdot avg + w_q \cdot q \tag{2.1}$$

The queue weight w_q is a constant between 0 and 1, while q represents the sampled current queue size. We defined two possible Counter Threshold values, N_{low} and N_{high}. When the current queue size is above or at the mid point between Minimum Threshold and Maximum Threshold, we set the Counter Threshold N to N_{low}. When the current queue size is below the midpoint value between Minimum Threshold and Maximum threshold, N is set to N_{high}. In effect, the Counter Threshold changes as the computed average queue size fluctuates. The packets dropping frequency becomes higher as the average queue size grows.

2.1.3 Periodic Discard with Sampling - Improved Version

The improved version ($SPDI$) differs from the original version SPD in the method for determining the counter threshold value. Instead of having two possible Counter Threshold values, this version calculates the Counter Threshold using the following formula:

$$P_b = 1/200 \cdot (Q_{avg} - min_{th})/(max_{th} - min_{th}) \tag{2.2}$$

$$N = \text{integer part of } 1/P_b \tag{2.3}$$

The discarding probability versus the computed average queue size is illustrated in Figure 2-1. This improved version allows finer granularity in computing the Counter Threshold value, and hence, should better reflect the average queue size of the switch.

2.2 Periodic Discard Scheme with Per-VC Technique

When implementing switch buffer, we can either make all VCs share a single FIFO (first-in-first-out) buffer, or allow each VC to have its own FIFO buffer. In this section, we are going to introduce four packet discard schemes that employ per-VC FIFO buffering. We set three threshold values for each of the four schemes: the lower Periodic Threshold ($PRED_{th}$), the Per-VC Periodic Threshold ($PRED_{vc}$), and the higher EPD Threshold (EPD_{th}). The Per-VC Periodic Threshold is essentially equal to $PRED_{th}$ divided by the number of active VCs ($PRED_{th}/VCN$).

2.2.1 Scheme A

Cells from different VCs will be put into different queues at each switch. Denote Q_j as the queue length for VC_j. We maintain a state variable ($Count_i$) for each VCs. When VC_j has a new packet coming, we will increase $Count_j$ by 1 if both the aggregate queue size (all Q_j combined) exceeds $PRED_{th}$ and Q_j is above the $PRED_{vc}$. When the counter exceeds the given Counter Threshold (N), a packet will be dropped. The counter is being reset to zero whenever a packet gets dropped. The new packet will be dropped if the aggregate queue size exceeds (EPD_{th}).

2.2.2 Scheme B

For this scheme, we maintain two counters: Per-VC counters ($Count_j$) and Aggregate Counter (TotalCount). When VC_j has a new packet coming, we will increase both the $Count_j$ and TotalCount by 1 if both the aggregate queue size exceeds $PRED_{th}$ and Q_j is above the $PRED_{vc}$. When either $Count_j$ exceeds N or $TotalCount$ exceeds N, a packet will be dropped. Both counters are then reset to zero. The new packet will be dropped if the aggregate queue size exceeds (EPD_{th}).

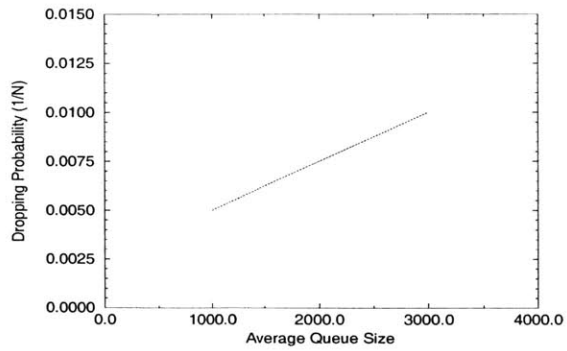


Figure 2-1: Average Queue Size vs. Dropping Probability

2.2.3 Scheme C

For this scheme, we only maintain one counter: the Aggregate Counter (TotalCount). When VC_j has a new packet coming and if the aggregate queue size is greater than $PRED_{th}$, the TotalCount is incremented by 1. A packet is discarded when the TotalCount exceeds N and the arriving VC's queue length is greater than $PRED_{vc}$. The new packet will be dropped if the aggregate queue size exceeds (EPD_{th}).

2.2.4 Scheme D

Again, we maintain one counter: the Aggregate Counter (TotalCount). When VC_j has a new packet coming, if the arriving VC's queue length is greater than $PRED_{vc}$ and the aggregate queue size is greater than $PRED_{th}$, the TotalCount is incremented by 1. A packet is discarded when the TotalCount exceeds N and the arriving VC's queue length is greater than $PRED_{vc}$. The TotalCount is reset to zero whenever a packet is dropped. The new packet will be dropped if the aggregate queue size exceeds (EPD_{th}).

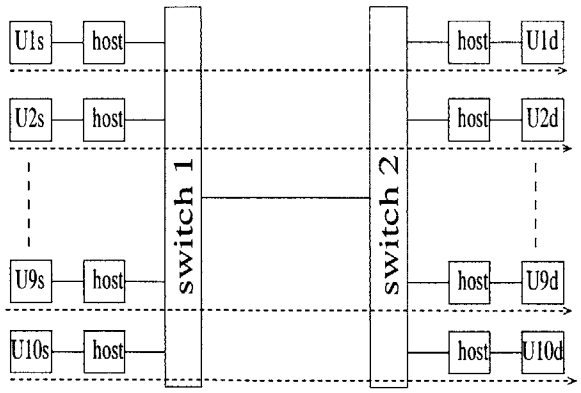


Figure 2-2: A 10-VC peer-to-peer configuration

2.3 Simulation Model and Parameters

2.3.1 Schemes: PRED, SPD, SPDI (Bursty Sources)

This section describes the simulation results of the various aforementioned schemes. The configuration we considered for our simulations is ten peer-to-peer connections based on our simulation components. It is illustrated in Figure 2-2. The following simulation parameters are employed in our simulations:

TCP:

Mean Packet Processing Delay = 100 μsec,

Packet Processing Delay Variation = 10 μsec,

Packet Size = 1024 Bytes,

Maximum Receiver Window Size = 64 KB,

Default Timeout = 500 $m$sec.

Switch:

Packet Processing Delay = 4 μsec,

Buffer Size (Q_{max}) = infinity,

Minimum threshold (min_{th}) = 1000 cells,

Maximum threshold (max_{th}) = 3000 cells.

For RED:

Maximum value for p_b (max_p) = 0.02,

Queue weight (w_q) = 0.002.

For PRED:

Counter Threshold (N) = 200,

Queue weight (w_q) = 0.02.

For Periodic Discard with Sampling (SPD):

Counter Threshold:

Queue weight (w_q) = 0.2.

$N = 200$ when $Q_{avg} < 2000$;

$N = 100$ when $Q_{avg} >= 2000$;

For Period Discard with Improved Sampling (SPDI):

Counter Threshold:

$N = (\text{int})\ 1/P_b$,

$P_b = 1/200 \cdot (Q_{avg} - min_{th})/(max_{th} - min_{th}) + 1/200$.

Input Link:

10 Mbps.

Output Link:

10 Mbps.

The TCP source behavior we simulated incorporate both the fast retransmit and fast recovery mechanisms.

In our simulations, we consider a 10-VC peer-to-peer configuration illustrated in Figure 2-2 where VC 1-10 are bursty sources, each having 102 KB (104448 bytes) data to transmit during each 1-second period.

Let us denote the number of VCs as N, the data burst size for each connection as S KByte ($1KByte$=$1024 \times 8bits$), and the congested link bandwidth as B Mbps. An ATM cell is of 53 bytes in which 48 bytes are payload. Then, the minimum time for each TCP source to complete transmission of each burst can be computed by assuming the link to be fully utilized:

$$\frac{N \times S \times 1024 \times 8}{B \times 10^6 \times \frac{48}{53}}. \tag{2.4}$$

For our particular scenario of Figure 2-2, we have: N = 10, S = 102 KByte, B =

10 Mbps.

Therefore, the shortest time for each of the bursty VCs to complete its transmission of each burst is:

$$\frac{10 \times 102 \times 1024 \times 8}{10 \times 10^6 \times \frac{48}{53}} \text{sec} = 0.92(\text{sec}) \tag{2.5}$$

However, because of the TCP flow control mechanism, it will take longer to transmit during periods of congestion. In our simulations, each burst is generated 1 second apart.

We have also included simulation results for output link equals to 150 Mbps. We use 30VC peer-to-peer configuration with each input link equals to 150 Mbps. We set the burst size to 508 KByte. Using equation 4, the shortest time for each of the bursty VCs to complete its transmission of each burst is:

$$\frac{30 \times 508 \times 1024 \times 8}{150 \times 10^6 \times \frac{48}{53}} \text{sec} = 0.92(\text{sec}) \tag{2.6}$$

2.3.2 Schemes: A, B, C and D (Greedy Sources)

TCP:

Mean Packet Processing Delay = 100 μsec,

Packet Processing Delay Variation = 10 μsec,

Packet Size = 1024 Bytes,

Maximum Receiver Window Size = 64 KB,

Default Timeout = 500 $m$sec.

Switch:

Packet Processing Delay = 4 μsec,

Buffer Size (Q_{max}) = infinity,

Aggregate periodic threshold ($PRED_{th}$) = 1000 cells,

Per-VC periodic threshold ($PRED_{vc}$) = 100 cells,

EPD threshold (EPD_{th}) = 3000 cells.

For Scheme A:

Counter Threshold = 200.

For Scheme B:

Counter Threshold = 200.

TotalCount Threshold = 200.

For Scheme C:

TotalCount Threshold = 200.

For Scheme D:

TotalCount Threshold = 200.

Input Link:

150 Mbps.

Output Link:

150 Mbps.

2.4 Simulation Results

2.4.1 Simulation Results for Periodic Discard Schemes with FIFO queues

Table 1 summarized the results for the simulations described under section 2.1.

As we can see, SPD with $SI = 50ms$ yields the highest throughput. With two different Counter Thresholds, SPD controls output queue size better than PRED. In fact, when we observe the loss of throughput among the above schemes, we realized that SPD yields the fewest repeated packets. SPD drops the most packets when Q_{avg} is greater than 3000. A large amount of dropped packets serves as a signal for TCPs to control their sources. TCPs control their sources by entering slow start stages which can lead to packets retransmission. Based on the number of repeated packets of the above schemes, we can conclude that SPD is able to control its output queue size below max_{th} better than other variations of Periodic Discard Scheme.

The result of SPDI with SI = 50ms is very similar to the result of SPDI with

	Throughput (Mbps)		Duplicated Received Pkts		Throughput Loss (Mbps)	
Time	0-5s	2-5s	0-5s	2-5s	0-5s	2-5s
N_{low}=50, N_{high}=100	8.70	9.34	350	155	0.65	0.48
N_{low}=100, N_{high}=200	8.53	9.24	406	230	0.76	0.72
N_{low}=200, N_{high}=300	8.43	9.11	514	279	0.95	0.86

Table 2.1: SPD for various different values of N_{low} and N_{high}

SI=100ms. SPDI yields a better throughput than PRED, but slightly less than SPD. The packets dropping probability (P_b) of SPDI is calculated as a linear function of the average queue size. Essentially, P_b equals the inverse of Counter Threshold. As we mentioned above, Counter Threshold is used when average queue size is between min_{th} and max_{th}. For our simulations, N is between 100 and 200. The relationship between average queue size and Counter Threshold is best illustrated in Figure 2-1. Intuitively, we would predict SPDI will yield the best results because the dropping probability of SPDI reflects its average output queue size the best. In our simulation results, however, the throughput of SPDI is less than SPD. This can be explained by the fact that SPD drops packets more aggressively than SPDI. With more dropped packets, SPD is able to signal TCPs of congestion earlier.

The Periodic Discard Schemes and its variation yield results that are comparable to RED. The results of RED is as follows:

RED:

Throughput:

$2-5\ sec = 9.35$ Mbps

$0-5\ sec = 8.79$ Mbps

RED drops packets more aggressively than any of the above schemes. The output queue size of RED is kept below $max_{th} = 3000$ after 2 seconds. Keeping the output queue size below max_{th} not only reduces the number of packets dropped when Q_{avg} exceeds max_{th}, repeated transmission is also reduced because of TCPs enter slow start phase less frequently.

	Throughput (Mbps)		Duplicated Received Packets		Throughput Loss (Mbps)	
Time	0-5s	2-5s	0-5s	2-5s	0-5s	2-5s
PRED	8.45	9.02	519	303	0.97	0.94
SPD (50ms)	8.53	9.24	406	230	0.76	0.72
SPDI (50ms)	8.48	9.14	479	267	0.79	0.83
SPDI (100ms)	8.46	9.10	424	267	0.79	0.83
RED	8.79	9.35	269	133	0.50	0.41

Table 2.2: Periodic Discard Scheme and its variations (BW = 10 Mbps)

As we adjust the parameter values for N_{low} and N_{high}, we discover that SPD yields the best performance when N_{low}=50 and N_{high}=100. With threshold values set to 50 and 100, SPD drops packets more aggressively and hence is able to detect congestion at an earlier stage. Though the frequency of dropping is higher when the counter threshold is smaller, by preventing the queue from reaching the higher queue threshold, the source can be throttled without going to the slow start phase. When the queue is too high, a bursty of packets will be discarded, forcing the TCP flow to enter the slow start phase, which in turn causes retransmission of packets that have been received by the destination.

Also note that RED yields the highest throughput despite the fact that the output queue size becomes empty sometimes. RED discards 3 times as many packets as SPD. Even though the queue remains nonempty from 2 to 5 seconds in the SPD simulations, the effective throughput is under 10 Mbps. The loss of throughput is contributed by the duplicated received packets. The effect of duplicated received packets is much more significant than other factors for contributing the loss of throughput.

While these simulations were based on a link bandwidth of 10 Mbps, we have also simulated a 150 Mbps link and compare the throughput of RED and SPD. The results are summarized in Table 2.3, which indicates that the performance of SPD scheme also scales to bandwidth of 150Mbps.

	Throughput (Mbps)	
Time	0-5s	2-5
SPD (50ms)	109	146.15
RED	107.62	142.43

Table 2.3: A throughput comparison of SPD and RED (BW = 150 Mbps)

2.4.2 Simulation Results for Periodic Discard Schemes with Per-VC Technique

Table 2.4 summarizes the results for Schemes A, B, C and D. Scheme D yields the highest throughput among the four schemes. The output queue is never empty from 2 to 5 seconds. As shown in the discard sequences plot of Scheme D, packets are evenly dropped during the last three seconds. We see a full bandwidth utilization during the time period between 2 second and 5 second. This implies that there is no loss of throughput due to repeated packets. Scheme D is able to identify aggressive VCs in a much earlier stage and more accurately than the other schemes.

In Scheme D, TotalCount is used to keep track of the number of packets entered when the arriving VC's queue length exceeds the $PRED_{vc}$ threshold, and the aggregate queue is greater than $PRED_{th}$. The scheme that uses TotalCount is able to detect congestion earlier than the one that uses per-VC counter only, namely Scheme A. When one per-VC counter is greater than N, it is likely that other per-VC counters are close to or greater than N at the same time. Hence packets tend to be dropped in burst as shown in the discard sequences plot for Scheme A. When packets are dropped from different VCs at the same time, it is highly likely that many TCPs will go through slow start stage all at once. This causes global synchronization. This can explain why Scheme A has the worst performance. Scheme C is similar to Scheme D, except that the TotalCount is incremented when the aggregate queue is greater than $PRED_{th}$. Scheme C is unable to identify aggressive VCs accurately since it does not incorporate the individual queue size in its decision of dropping packets. Finally, Scheme B drops packets when TotalCount or per-VC Count is greater than N. As

	Throughput (Mbps)	
Time	0-5s	2-5s
Scheme A	108.8	142.6
Scheme B	115.4	147.6
Scheme C	116.4	149.1
Scheme D	120.5	150.0

Table 2.4: Periodic Discard Scheme with Per-VC technique (BW = 150 Mbps)

we examine the results of the four schemes, we realize that Scheme B dropped the largest number of packets. The less-than-ideal performance of Scheme B is probably due to the fact that it drops more packets than necessary.

2.5 Concluding Remarks

In this chapter, we have presented a simulation study on some variations of packet discarding schemes that aim to emulate the RED schemes. The schemes based on FIFO queuing are simpler to implement in hardware than the original version of RED. Among them, the periodic discard schemes with sampling (SPD) exhibit performance comparable to RED in our simulations.

We have also studied periodic discard schemes that employ per-VC queuing. Our simulation results show that these per-VC queuing schemes exhibit good performance in terms of TCP goodput as well as fairness.

Since the performance of these schemes will depend on the traffic and network parameters such as link bandwidth, link delay, and queue size, there is no single set of scheme parameters that will yield good performance under all traffic conditions. Although the issue of parameter tuning will require a more thorough simulation and theoretical study, we believe that the parameters we chose in our simulations are quite robust and should yield good performance under a variety of traffic conditions.

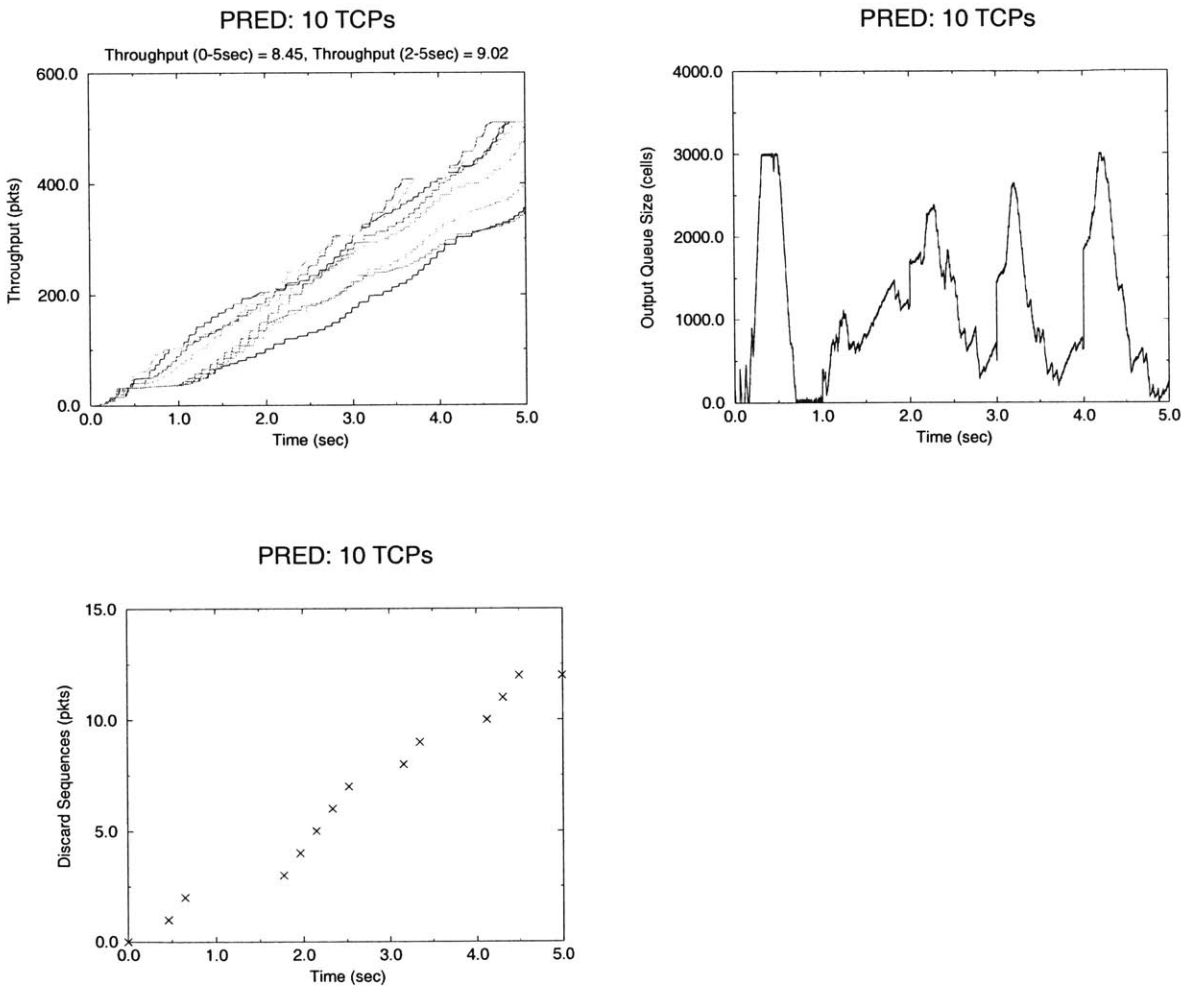


Figure 2-3: Simple Periodic Discard Scheme

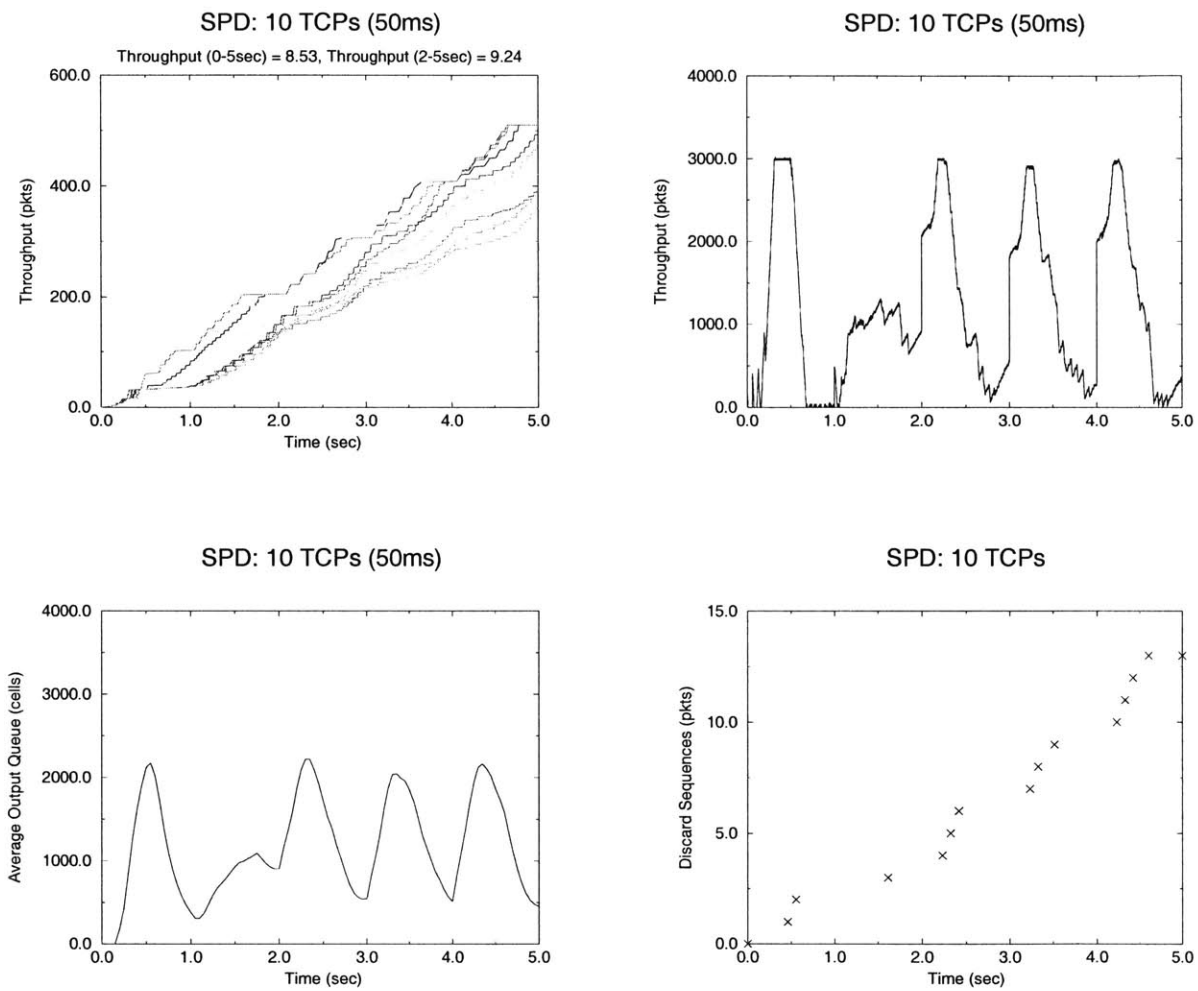


Figure 2-4: Periodic Discard with Sampling (Interval = 50ms)

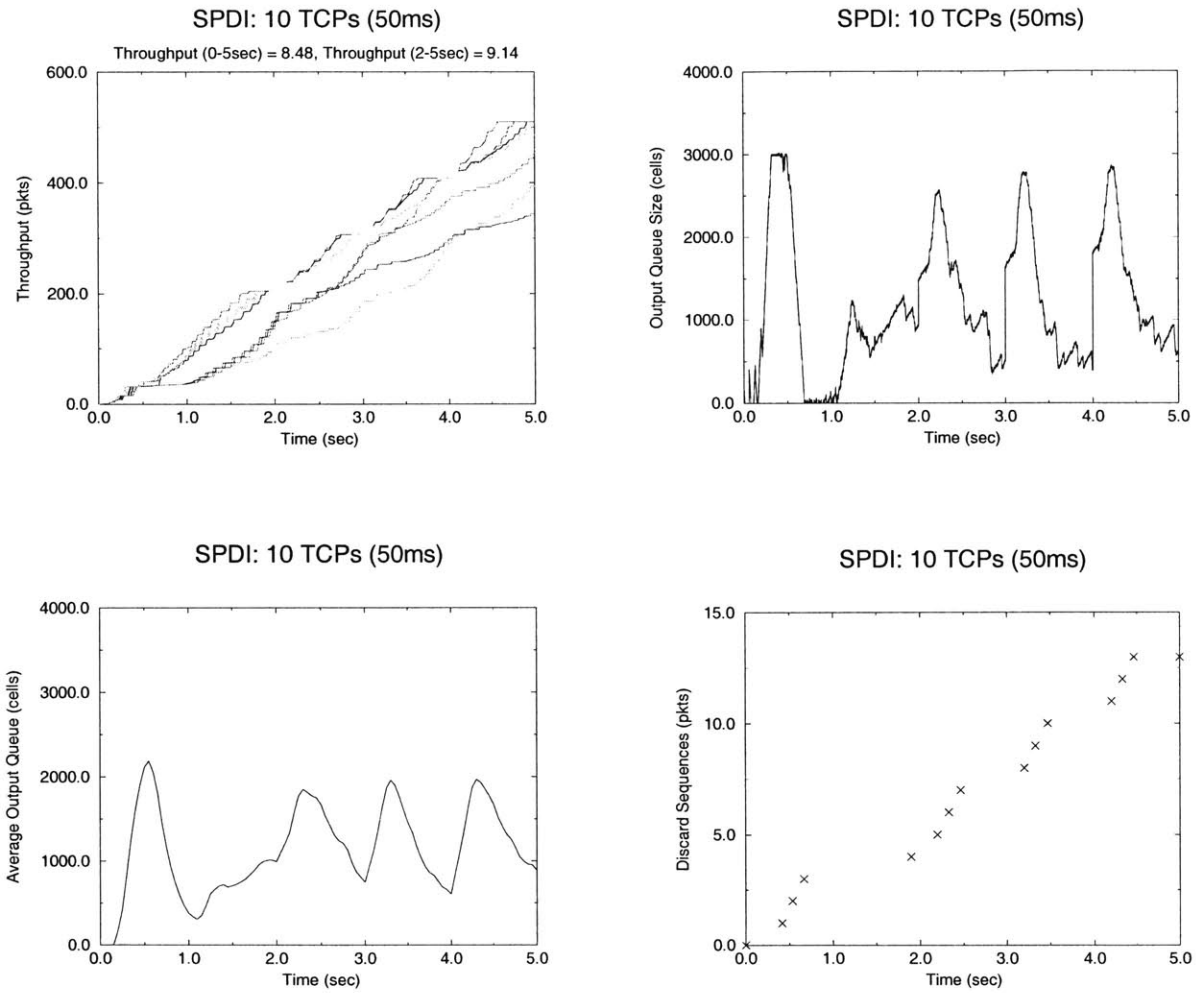


Figure 2-5: Improved Periodic Discard with Sampling (Interval = 50ms)

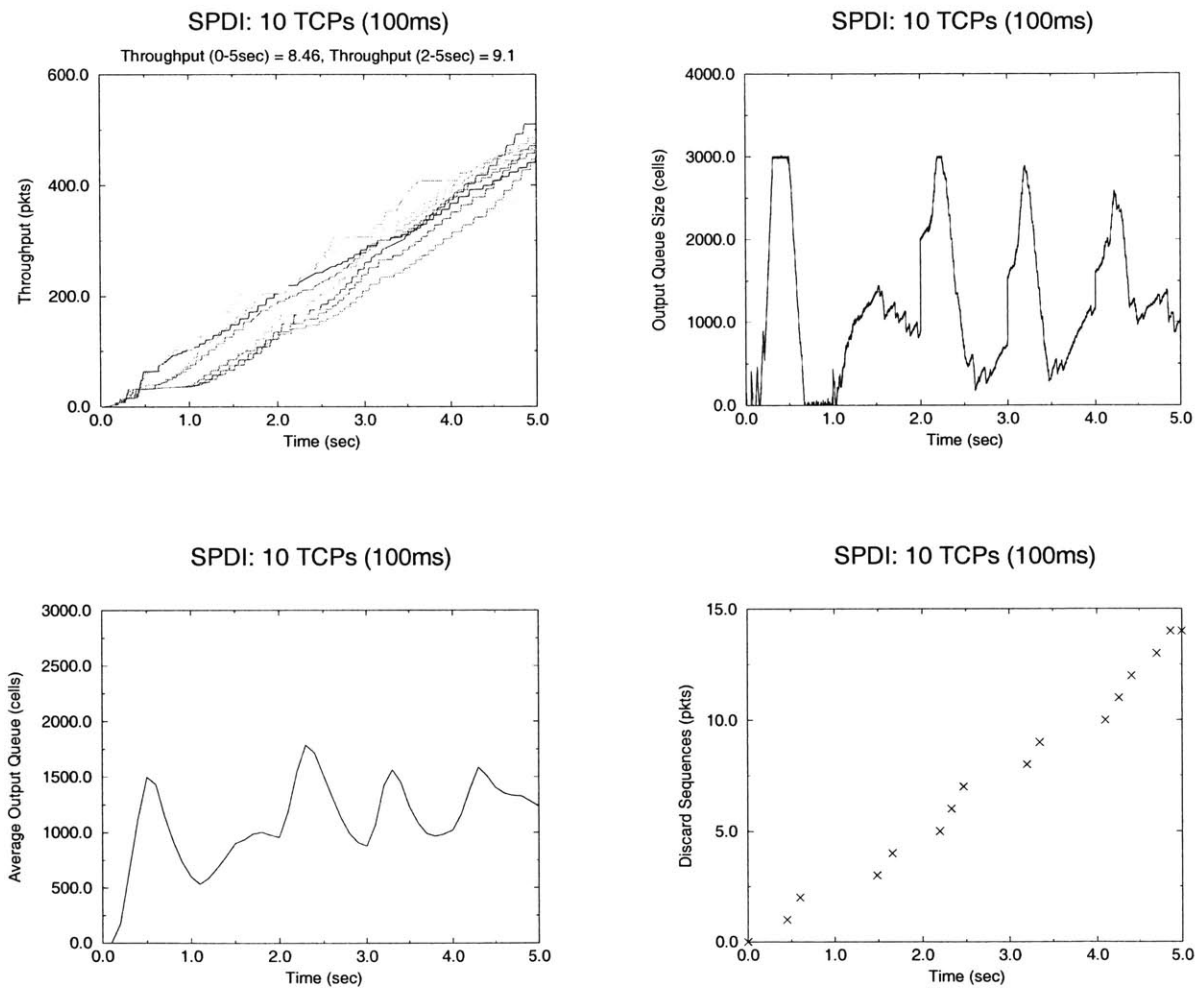


Figure 2-6: Improved Periodic Discard with Sampling (Interval = 100ms)

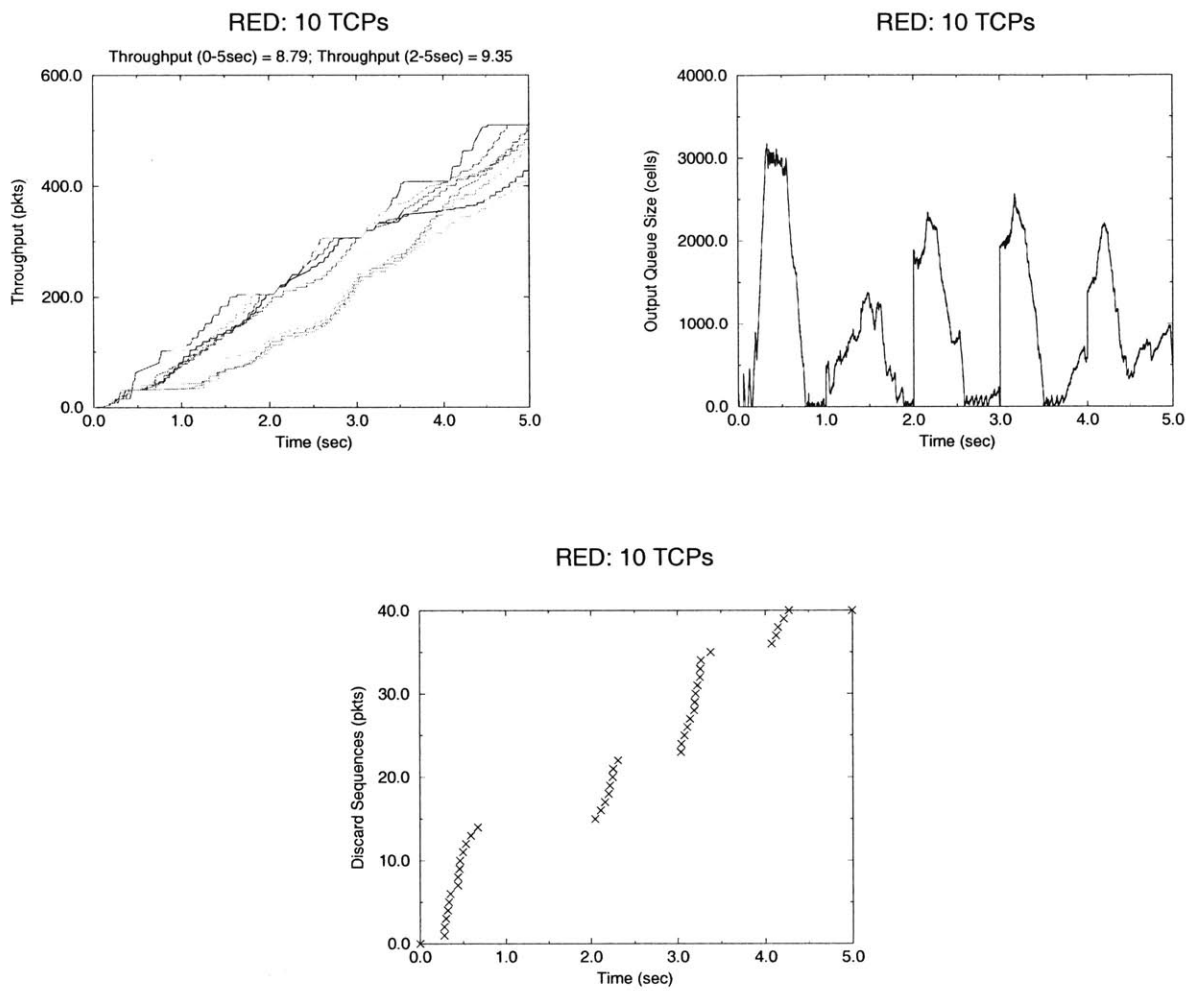


Figure 2-7: RED: 10 TCPs, BW = 10Mbps

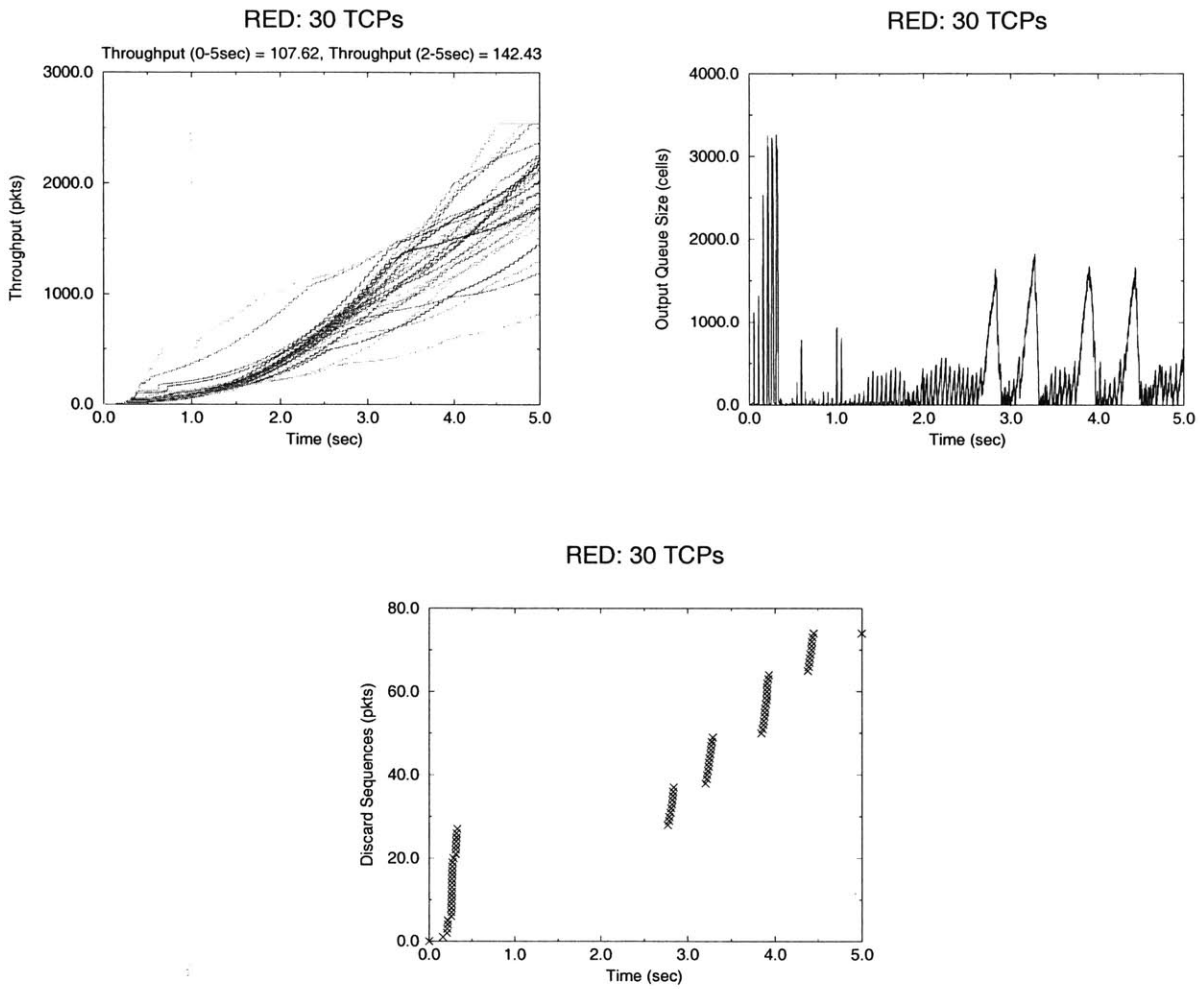


Figure 2-8: RED: 30 TCPs, BW = 150Mbps

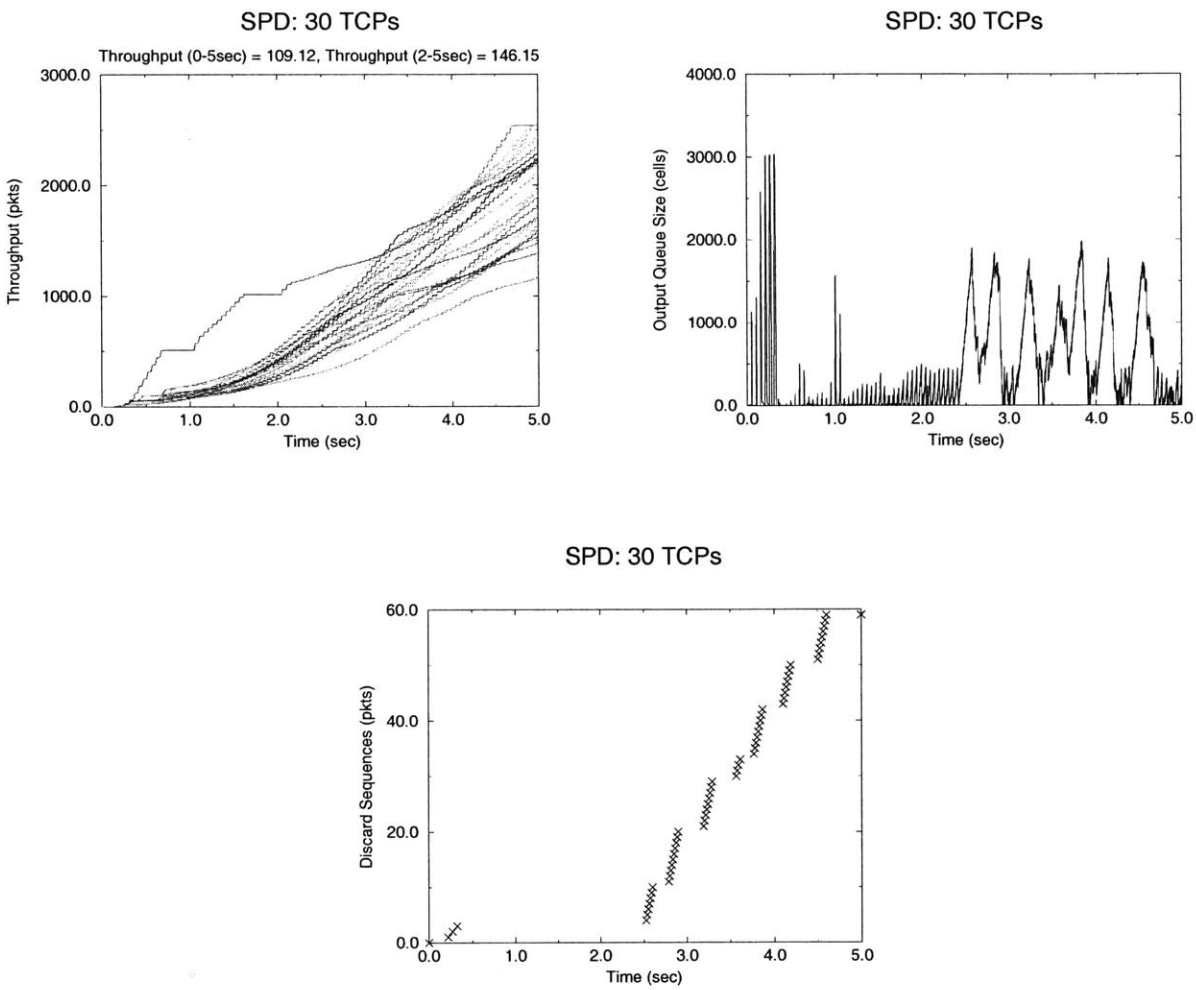


Figure 2-9: SPD: 30 TCPs, BW = 150Mbps

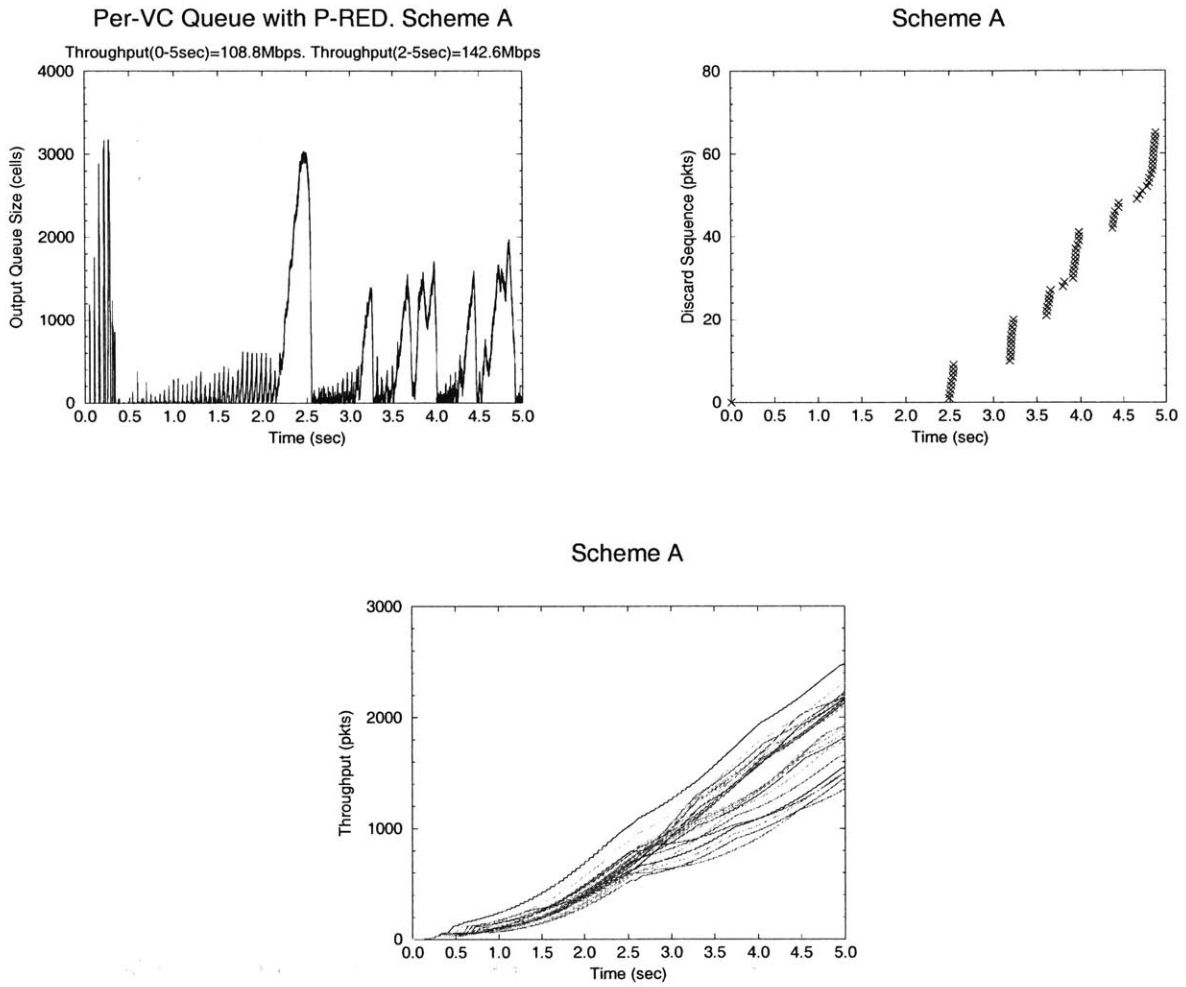


Figure 2-10: Per VC Queue with PRED: Scheme A

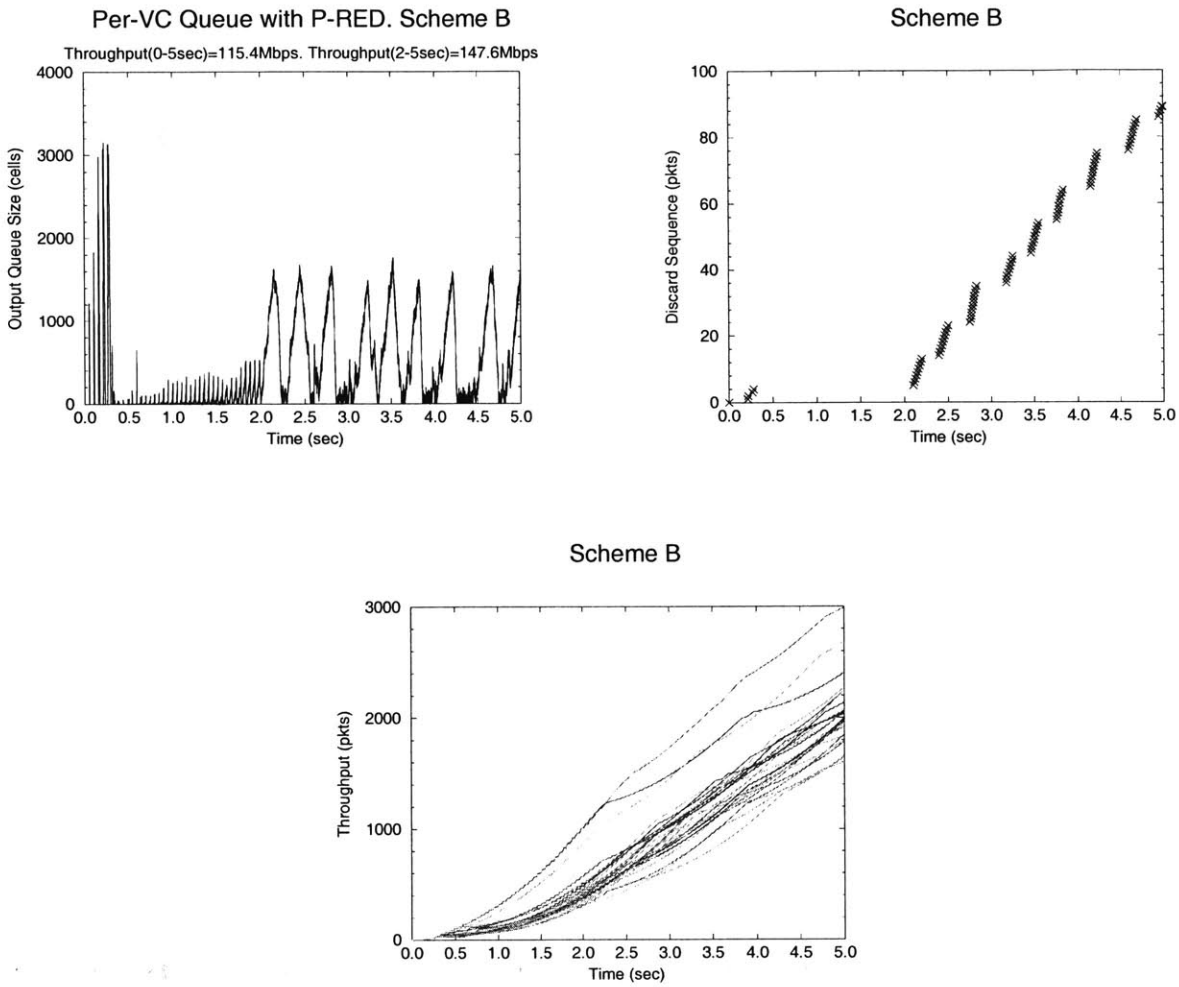


Figure 2-11: Per VC Queue with PRED: Scheme B

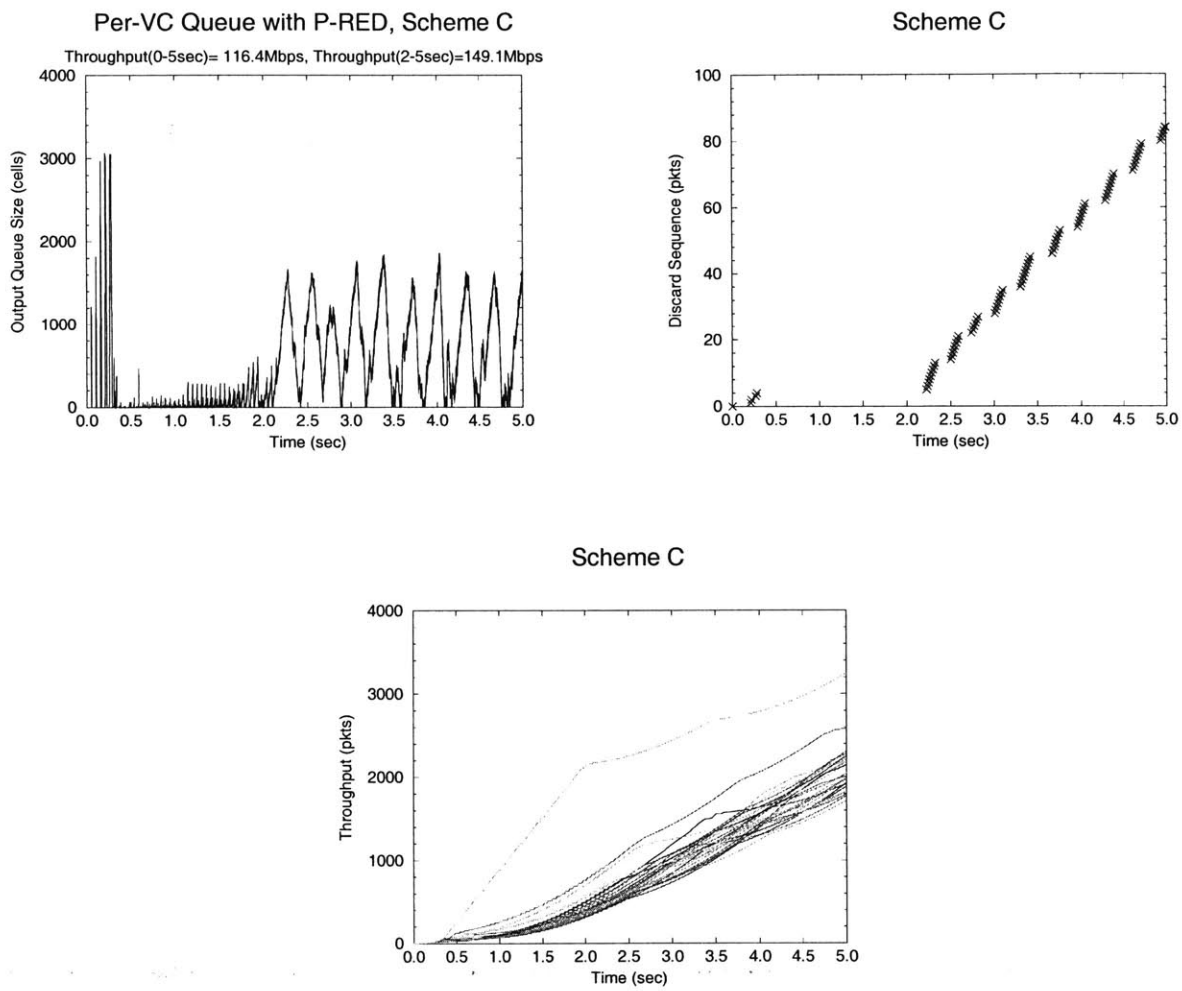


Figure 2-12: Per VC Queue with PRED: Scheme C

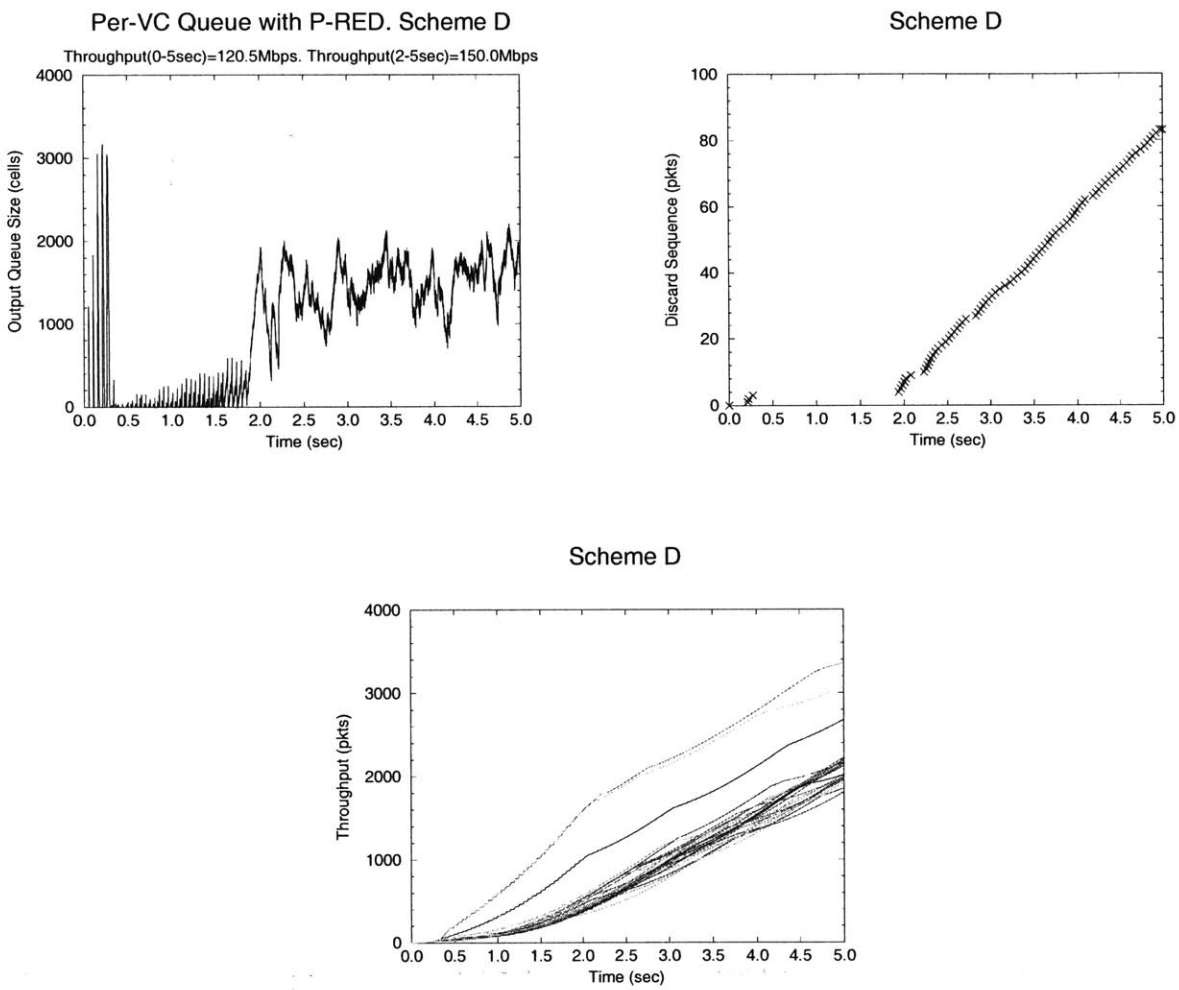


Figure 2-13: Per VC Queue with PRED: Scheme D

Chapter 3

Differentiated Services

In response to the demand of a robust service classification system in the Internet, the Internet Engineering Task Force (IETF) is drafting an architecture and definitions for a Differentiated Services (DS) mechanism [4]. The goal of the architecture framework is to provide differentiated services that are highly scalable and relatively simple to implement. In this chapter, we are going to study and compare two differentiated services network models that are based on the framework proposed by IETF. In this chapter, we are going to study and compare the End-to-End QoS and the Hop-by-Hop CoS network models that are based on the framework proposed by IETF.

The DS architecture proposed by IETF is based on a simple model where traffic entering a network is classified and optionally shaped at the network boundaries, and then is assigned to different behavior aggregates. Specifically, each packet entering DS capable network is marked with a specific value in the DS-field that indicates its per-hop behavior (PHB) within the network. Packets are forwarded according to the PHB specified in the DS-field within the core of the network [4]. Network resources are allocated to the traffic streams based on the PHB. This architecture achieves scalability by pushing complexity out of the core of the network into edge devices which process lower volumes of traffic and fewer numbers of flows, and offering services for aggregated traffic rather than on a per-application flow basis [4]. The DS architecture offers building blocks for supporting differentiated services. Provisioning strategies, traffic conditioners, and billing models can be implemented on top of these

building blocks to provide a variety of services [3].

3.1 End-to-End QoS

The End-to-End QoS model provides quantitative services using provisioning traffic conditioners at the edge routers. The model assumes packets are pre-marked in the DS-field with a specific value. This value is used to classify individual application flows into per-class flows. Each class specifies service performance parameters in the form of traffic conditioning agreement (TCA). The TCAs are static; the variations of the network load will not affect the negotiated performance. The traffic conditioning components used at the edge routers are meters and shapers. Meters measure traffic streams for conformance to TCA and provide input to policing components such as shapers. Shapers delay packets in a traffic stream such that it does not exceed the rate specified in the TCA. The meters and shapers are based on the Generic Cell Rate Algorithms (GCRA), or equivalently, the leaky bucket controller. Using leaky bucket controller as traffic conditioner has significant implementation and cost advantage, because most commercial ATM switches already have hardware implementation of GCRA, which is used for example in VBR service. In our study, the edge is assumed to have a finite-sized buffer and packets are discarded using the RED algorithm. Connection admission control is also added to complement the traffic conditioning at the edge routers.

3.1.1 GCRA

Since our model is based on GCRA, we will first review the basic ideas of GCRA. Figure 3-1 illustrates the Leaky Bucket Algorithm. The algorithm maintains two state variables which are the leaky bucket counter X, and the variable LCT (last conformance time) which stores the arrival time of the last conforming cell. The GCRA is defined with two parameters: I (increment) and L (limit). I is the ideal cell interarrival time (i.e., the inverse of the ideal cell rate). L is the maximum bucket level determined by the allowed burst tolerance. The notation "GCRA (I,L)" means

GCRA with increment I and limit L. When the first cell arrives at time $t_a(1)$, X is set to zero and LCT is set to $t_a(1)$. Upon the arrival time of the kth cell, $t_a(k)$, the bucket is temporarily updated to the value X', which equals the bucket level X, last updated after the arrival of the previous conforming cell, minus the amount the bucket has been drained since that arrival. The level of the bucket is constrained to be non-negative. If X' is less than or equal to the bucket limit L, then the cell is declared conforming, and the bucket variables X and LCT are updated. Otherwise, the cell is declared non-conforming and the values of X and LCT are not changed. We can observe from Figure 3-1 that for each conforming cell, the bucket counter is updated by:

$$X_{new} = X_{old} - (t_a(k) - LCT) + I.$$

Thus, the leaky bucket counter X increases if the actual cell interarrival time $(t_a(k) - LCT)$ is less than the ideal cell interarrival time I (i.e., if the cell arrives too early), and decreases otherwise. If the cells arrive at the ideal rate, the bucket counter will stay constant. This also illustrates the fact that the leaky bucket can be viewed as being drained out at the ideal rate (this is true independent of the input traffic) [11].

As mentioned above, switches with GCRA are normally used to support non-data services such as VBR. ATM Forum defines the VBR parameters in relation to two instances of GCRA. In the notation GCRA (I,L), one instance is GCRA (1/PCR, CDVT), which defines the CDVT (cell delay variation tolerance) in relation to PCR (peak cell rate). The second instance GCRA (1/SCR, BT+CDVT) defines the sum of BT (burst tolerance) and CDVT in relation to SCR (sustainable cell rate). PCR specifies an upper bound of the rate at which traffic can be submitted on an ATM connection. SCR is an upper bound of the average rate of the conforming cells of an ATM connection, over time scales which are long relative to those for which the PCR is defined. CDVT and BT specify the upper bound on the "clumping" of cells resulting from the variation in queuing delay, processing delay, etc. [11] We will refer to the GCRAs associated with PCR and SCR as GCRA(1) and GCRA(2), respectively.

3.1.2 Boundary Provisioning

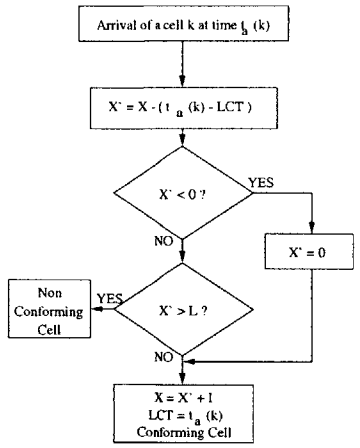
GCRA is used in our model as a traffic conditioner at the edge routers. GCRA combines the functions of a meter and a shaper. Using service performance parameters, PCR and SCR, which are specified in TCAs as parameters for GCRA(1) and GCRA(2), they are able to measure the traffic streams and mark the non-conforming packets. Traffics are shaped by delaying non-conforming cells until they are compliant to the TCAs. The delayed non-conforming cells are placed in finite-sized buffers according to their classes. The cells in these buffers are dropped using RED algorithm.

3.1.3 Interior Provisioning

The core network routers simply use FIFO queuing with EPD. They are also responsible for connection admission control (CAC) of the incoming traffic streams. The CAC we employed is to only accept individual application flow when the total requested SCRs is less than the bottleneck link capacity. In other words, when an application flow of a specific class wants to establish a connection with the DS capable network; if the total requested SCRs of all its flows are less than or equal to the link capacity, the connection is accepted. Otherwise, it is rejected.

3.2 Hop-by-Hop CoS

The Hop-by-Hop CoS model provides qualitative provisioning using priority queue scheduling at the core of the network. Packets are pre-marked in the DS-field with specific values that determine the PHB within the network. Classifiers at the edge routers map individual flows into per-class flows. Service differentiation is done at the core of the network using class-based queuing. Packets are served using weighted round robin scheduling at the core router.



CONTINUOUS-STATE LEAKY BUCKET ALGORITHM

X	Value of the Leaky Bucket Counter
X'	auxiliary variable
LCT	Last Compliance Time
I	Increment (set to the reciprocal of cell rate)
L	Limit

Figure 3-1: A flow chart description of the leaky-bucket algorithm

3.2.1 Boundary Provisioning

The edge routers classify the packets according to their DS-field values into per-class queues.

3.2.2 Interior Provisioning

Weighted round-robin Scheduling is employed at the core network routers. RED ais used to control congestion.

3.3 Simulation Model and Parameters

The network configuration we used for our simulations is illustrated in Figure 3-2. In the figure, the senders are represented by `us0` ... `us35`, the receivers (destinations) are represented by `ud0` ... `ud35`, the edge routers are represented by `ER#1` ... `ER#6`, and the core routers or switches are represented by `sw1` and `sw2`. Each flow (sender-receiver pair) is classified into one of the four classes. The sender transmits data to the receiver through the edge routers and core routers. As far as performance

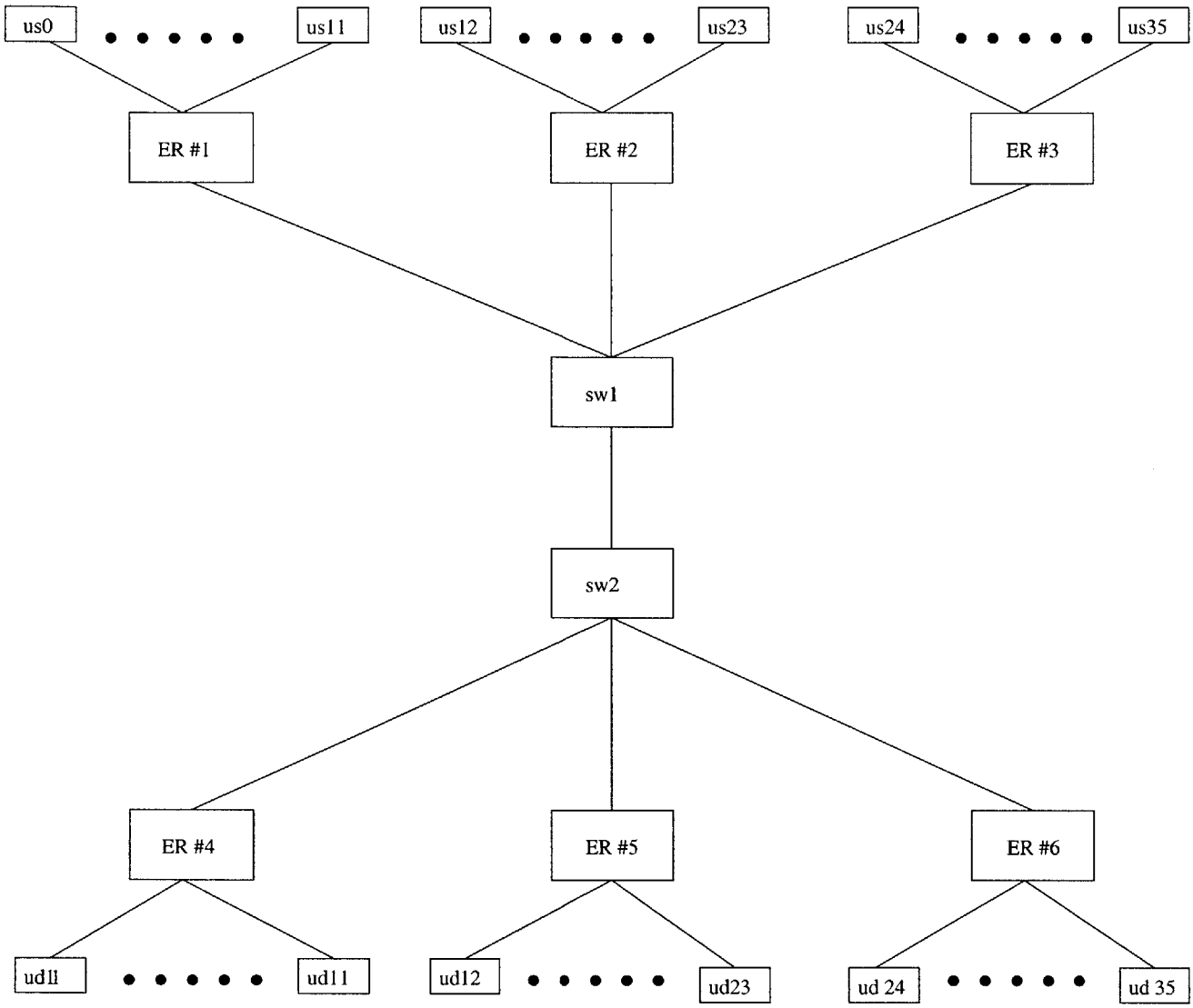


Figure 3-2: A Network Configuration for Differentiated Services

is concerned in our setup, the location of the unique corresponding receiver to each sender is immaterial.

3.3.1 End-to-End QoS without CAC

End-to-End QoS without CAC has traffic shapers at the edge routers, but there is no connection admission control. Hence, the total requested SCRs can be greater than the link capacity. In our simulations, we set the QoS parameters such that the aggregate SCR of each router is 150 Mbps ideally. The traffic sources we used in this

experiment are greedy. Since we have three routers and the bottleneck link capacity is only 150 Mbps, we created a congested link, with an overload factor of three, between the core network routers. The following parameters are employed in our experiments:

TCP:

Mean Packet Processing Delay = 100 μsec,

Packet Processing Delay Variation = 10 μsec,

Packet Size = 1024 Bytes,

Maximum Receiver Window Size = 64 KB,

Default Timeout = 500 $m$sec.

Edge Router QoS parameters:

Class 0: SCR = 80 Mbps, PCR = 90 Mbps.

Class 1: SCR = 40 Mbps, PCR = 50 Mbps.

Class 2: SCR = 20 Mbps, PCR = 30 Mbps.

Class 3: SCR = 10 Mbps, PCR = 20 Mbps.

Edge Router RED parameters:

Maximum Threshold = 50 packets (1100 cells)

Minimum Threshold = 100 packets (2200 cells)

Core Router:

Packet Processing Delay = 4 μsec,

Buffer Size (Q_{max}) = infinity,

EPD threshold (max_{th}) = 3000 cells.

Input Link:

150 Mbps.

Output Link:

150 Mbps.

Round-trip Time:

50 ms.

3.3.2 End-to-End QoS with CAC

The End-to-End QoS with CAC requires the total requested SCRs to be less than or equal to the link capacity. We perform experiments under this model with bursty sources and greedy sources. For the experiment with bursty sources, each VC has 414 KB (423936 bytes) data to transmit every second. Again, if we denote the number of VCs as N, the data burst size for each connection as S KByte ($1KByte$=$1024 \times 8bits$), and the congested link bandwidth as B Mbps. An ATM cell is of 53 bytes in which 48 bytes are payload. Then, the minimum time for each TCP source to complete transmission of each burst can be computed by assuming the link to be fully utilized:

$$\frac{N \times S \times 1024 \times 8}{B \times 10^6 \times \frac{48}{53}}. \tag{3.1}$$

For our particular scenario of Figure 3-2, we have: N = 36, S = 414 KByte, B = 150 Mbps. Therefore, the shortest time for each of the bursty VCs to complete its transmission of each burst is:

$$\frac{36 \times 414 \times 1024 \times 8}{150 \times 10^6 \times \frac{48}{53}} \text{sec} = 0.89(\text{sec}) \tag{3.2}$$

However, because of the TCP flow control mechanism, it will take longer to transmit during periods of congestion. In our simulations, each burst is generated 1 second apart.

For the experiment with greedy sources, each source send as much data and as fast as it can. The following parameters are employed for both of our experiments:

TCP:

Mean Packet Processing Delay = 100 μsec,

Packet Processing Delay Variation = 10 μsec,

Packet Size = 1024 Bytes,

Maximum Receiver Window Size = 64 KB,

Default Timeout = 500 $m$sec.

Edge Router QoS parameters:

Class 0: SCR = 27 Mbps, PCR = 54 Mbps.

Class 1: SCR = 13 Mbps, PCR = 26 Mbps.

Class 2: SCR = 7 Mbps, PCR = 14 Mbps.

Class 3: SCR = 3 Mbps, PCR = 6 Mbps.

Edge Router RED parameters:

Maximum Threshold = 50 packets (1100 cells)

Minimum Threshold = 100 packets (2200 cells)

Core Router:

Packet Processing Delay = 4 μsec,

Buffer Size (Q_{max}) = infinity,

EPD threshold (max_{th}) = 3000 cells.

Input Link:

150 Mbps.

Output Link:

150 Mbps.

Round-trip Time:

50 ms.

3.3.3 Hop-by-Hop CoS

Again, we perform experiments under this model with bursty sources and greedy sources. For the experiment with bursty sources, each VC has 414 KB (423936 bytes) data to transmit every second. Experiments using greedy sources are also performed.

The following parameters are used for both experiments:

TCP:

Mean Packet Processing Delay = 100 μsec,

Packet Processing Delay Variation = 10 μsec,

Packet Size = 1024 Bytes,

Maximum Receiver Window Size = 64 KB,

Default Timeout = 500 $msec$.

Core Router RED parameters:

Maximum Threshold = 2000 cells

Minimum Threshold = 4000 cells

Input Link:

150 Mbps.

Output Link:

150 Mbps.

Round-trip Time:

50 ms.

3.4 Simulation Results

3.4.1 End-to-End QoS without CAC

The throughput of individual VC is plotted by class in Figure 3-3. The numerical values of the throughput are shown in Table 3.1. The results do not exhibit service differentiation among VCs of different classes. The lack of service differentiation is caused by the over-booking of bandwidth reservation. The total requested SCRs is equal to 450 Mbps, while the bottleneck link capacity is equal to 150 Mbps. Over-booking of bandwidth reservation results in congestion at the link between the central switches. Even though traffic streams are shaped at the edge routers, we did not employ any mechanism to differentiate packets from different classes at the core network routers. Since there is no traffic conditioner for congestion at the bottleneck link, service differentiation is not achievable. We conclude from this experiment that connection admission control must be added to complement the traffic conditioning at the edge routers to provide service differentiation.

3.4.2 End-to-End QoS with CAC

In this set of experiments, we added the connection admission control such that over-booking will not occur. As we can see from Table 3.2 and Table 3.4, service differentiation is exhibited in both simulations - with bursty sources and with greedy sources.

With greedy sources, the total throughput is slightly less than the ideal throughput. Since the traffic streams are regulated at the edge routers, all traffic streams coming to the core routers are cooperative. These conservative traffic streams cause a slight under utilization of the link bandwidth. If we examine the output queue size at the central router closely in Figure 3-5, the queue stays low and remains non-empty most of the time. The output queue size reflects the stringent traffic control at the edge routers. Addition degradation is caused by the retransmitted packets that have been received. These duplicated received packets are triggered by miscalculations of round-trip time.

Table 3.6 summarized the results for both bursty and greedy sources. With bursty sources, the small number of dropped packets reflects that the traffic shapers at the edge routers control the sending rates at an early stage. In the case with bursty sources, the major factor contributing to the loss of throughput is because of the less efficient statistical multiplexing of the link bandwidth.

3.4.3 Hop-by-Hop CoS

With greedy sources, the performance under this model is worse than the End-to-End QoS model. At first glance, we would expect traffic regulation based on weighted round-robin scheduling of per-class queues at the core routers should yield better results than the End-to-End QoS model. This is because the Hop-by-Hop CoS model enables more efficient statistical multiplexing of the bottleneck link. With a closer examination, we find that the degradation of performance is caused by a large number of timeouts. Timeouts trigger TCP flows to go to slow start phase. When TCP flows reduce their sending rates together, the bottleneck link becomes underutilized.

Class	0	1	2	3
$\sum_i SCR_i$ (Mbps)	240	120	60	30
Actual BW (Mbps)	35.86	25.52	40.07	26.18

Table 3.1: Throughput for TCP traffic under End-to-End QoS without CAC (Greedy Sources)

Class	0	1	2	3
$\sum_i SCR_i$ (Mbps)	81	39	21	9
Actual BW (Mbps)	75.05	37.03	19.89	8.41

Table 3.2: Throughput for TCP traffic under End-to-End QoS with CAC (Greedy Sources)

There are a total of 689 retransmitted packets which are indications of miscalculation of round-trip time. This miscalculations of round-trip time is caused by the large fluctuation of queuing delay as the queue grows.

With bursty sources, the Hop-by-Hop CoS model performs better than the End-to-End QoS model. As mentioned before, global synchronization can be a major cause for performance degradation. Global synchronization does not have as big an impact on the Hop-to-Hop QoS model because of the statistical multiplexing at the bottleneck link. Weighted round-robin scheduling allows better link utilization since link bandwidth will not be wasted as long as one of the per-class queues is non-empty. In the End-to-End QoS model, there is only one queue at the central switch. Hence, global synchronization causes a tremendous degradation of the performance.

Table 3.5 and Table 3.3 shows service differentiation among classes under the Hop-by-Hop CoS model.

Class	0	1	2	3
Actual BW (Mbps)	46.19	32.020	21.95	16.37

Table 3.3: Throughput for TCP traffic under Hop-by-Hop CoS (Greedy Sources)

Class	0	1	2	3
$\sum_i SCR_i$ (Mbps)	81	39	21	9
Actual BW (Mbps)	33.89	29.99	19.82	8.97

Table 3.4: Throughput for TCP traffic under End-to-End QoS (Bursty Sources)

Class	0	1	2	3
Actual BW (Mbps)	31.84	28.15	23.86	17.65

Table 3.5: Throughput for TCP traffic under Hop-by-Hop CoS (Bursty Sources)

	Total Throughput	Duplicated Received Pkts	Total Dropped Pkts	Total Timeouts
QoS Greedy	140.38 Mbps	600	788	26
CoS Greedy	116.55 Mbps	689	781	76
QoS Bursty	92.22 Mbps	1452	61	54
CoS Bursty	101.50 Mbps	1577	686	96

Table 3.6: Total Throughput in the Two DS Models

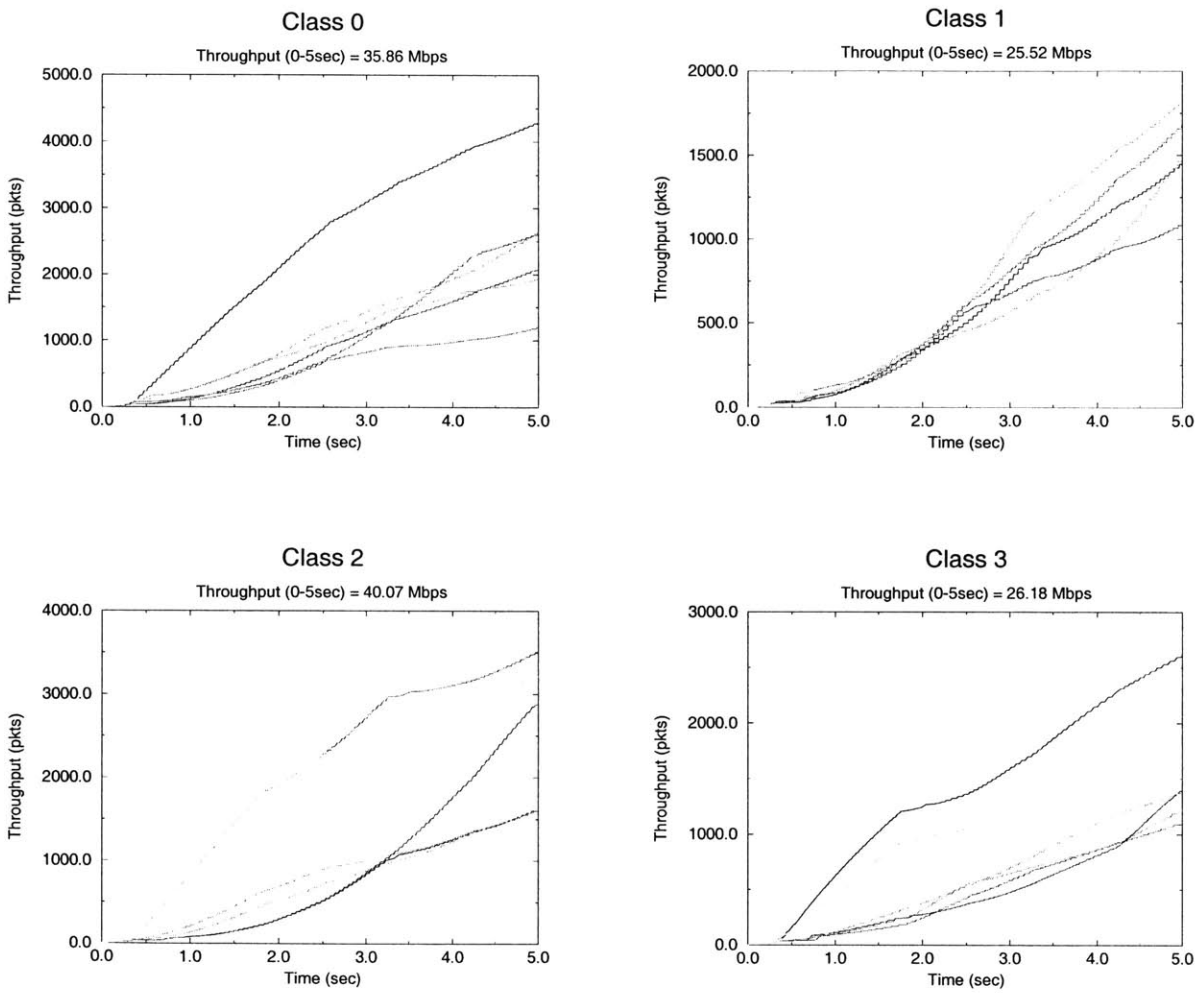


Figure 3-3: End-to-End QoS without CAC: Greedy Sources

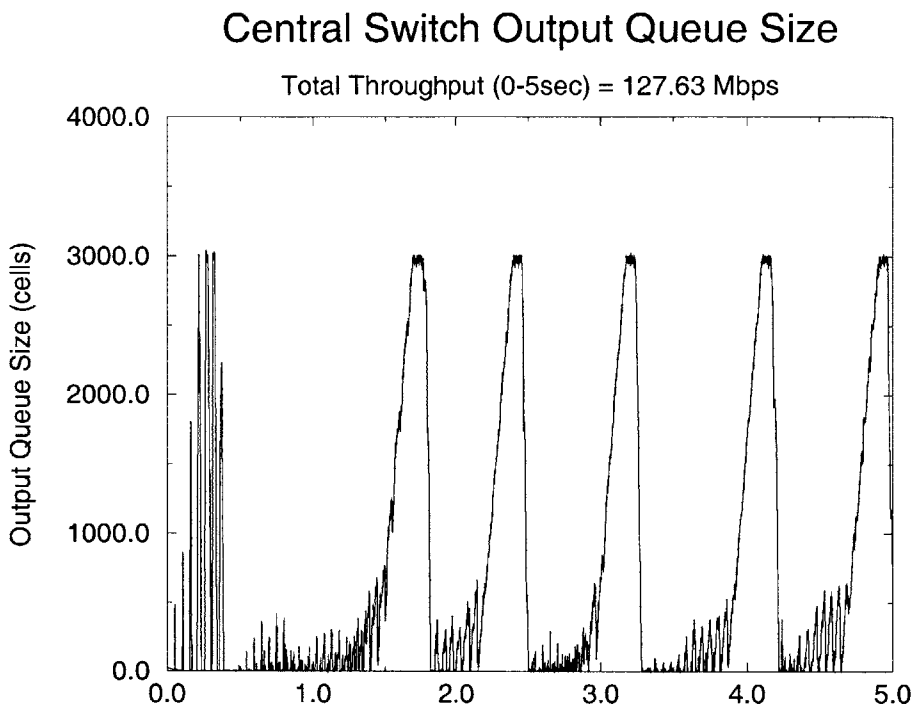


Figure 3-4: End-to-End QoS without CAC: Greedy Sources

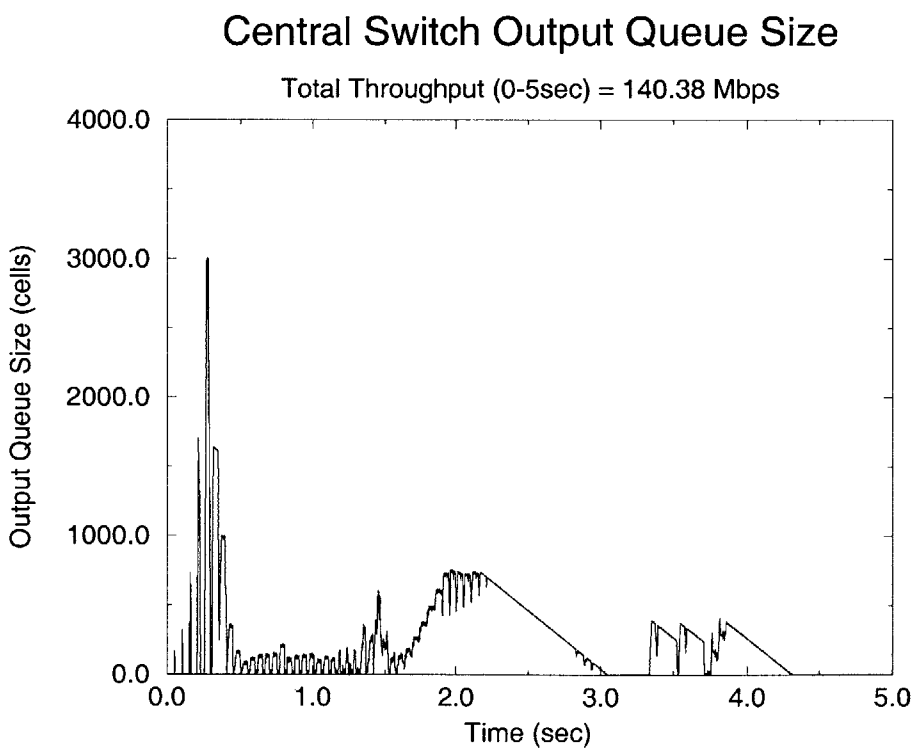


Figure 3-5: End-to-End QoS with CAC: Greedy Sources

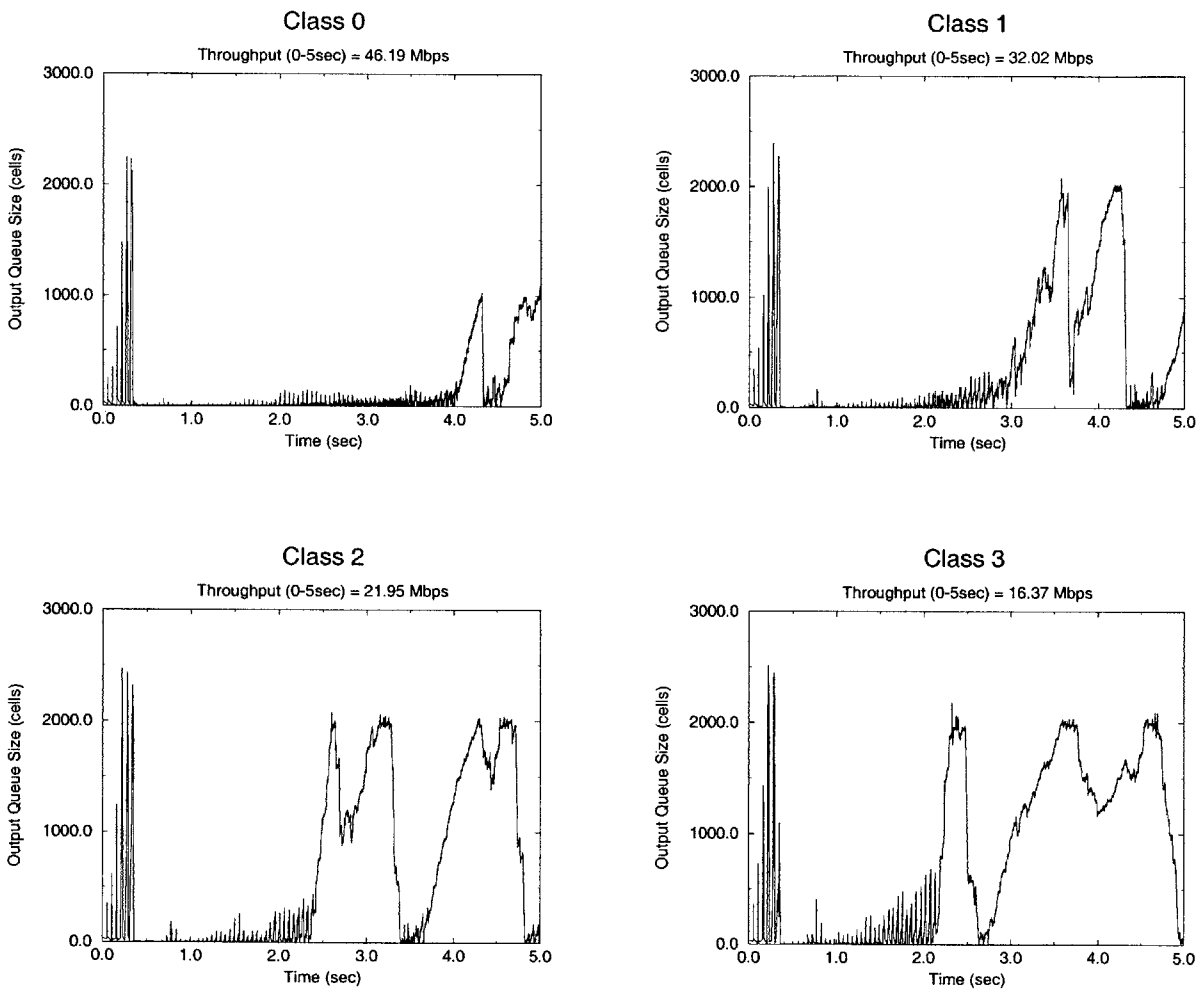


Figure 3-6: Hop-by-Hop CoS: Greedy Sources

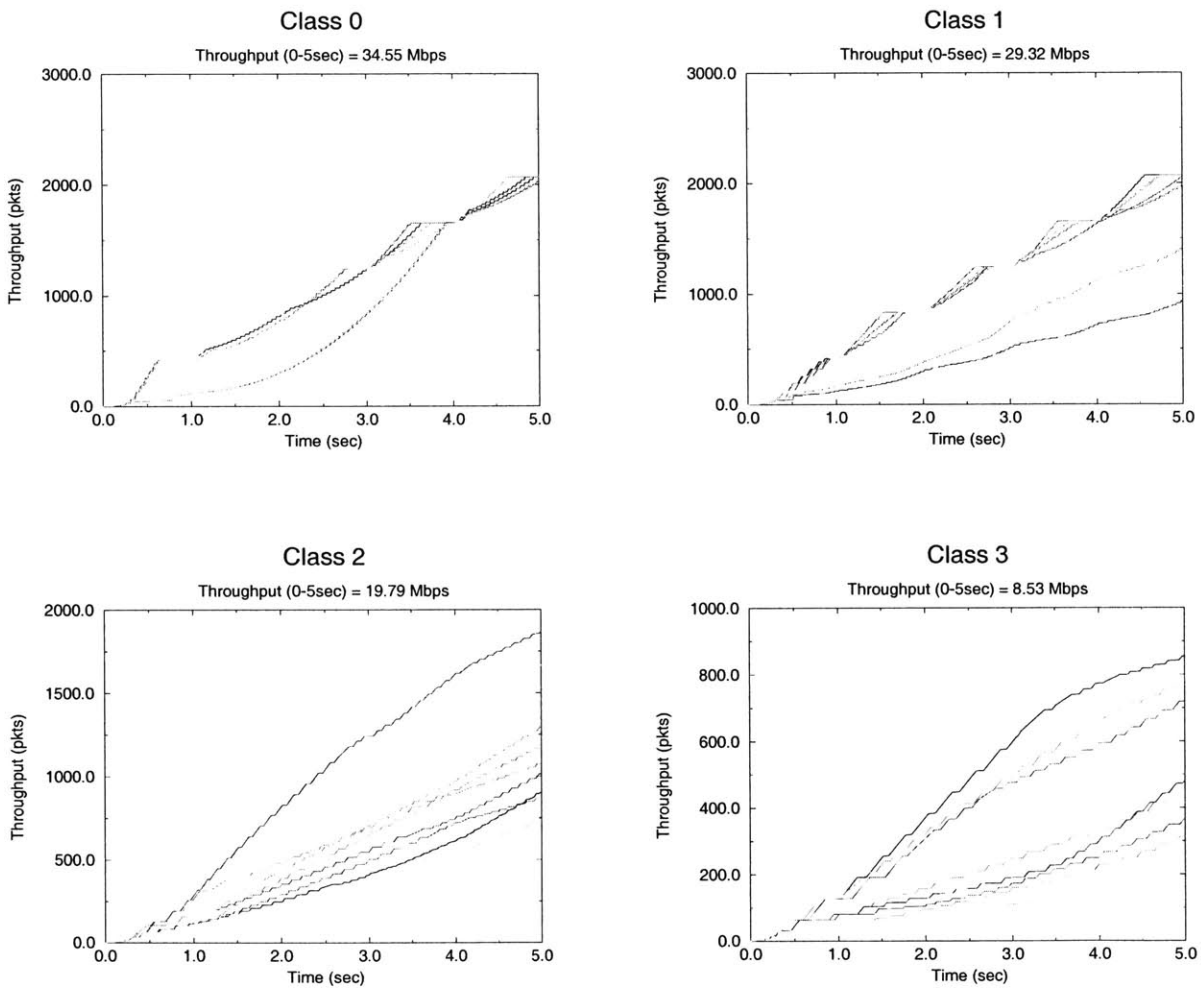


Figure 3-7: End-to-End QoS with CAC: Bursty Sources

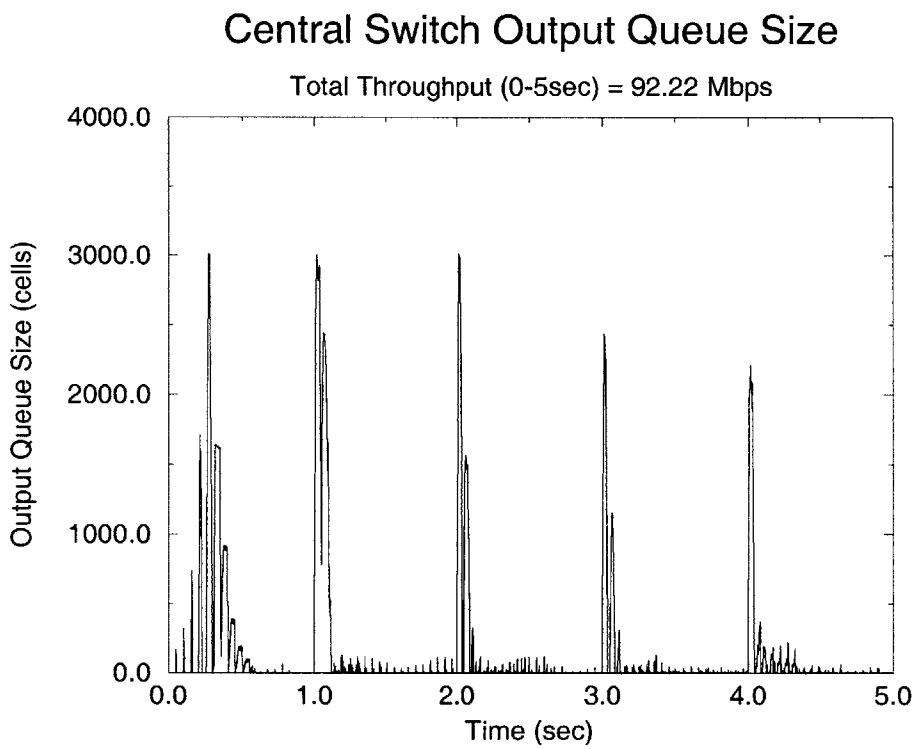


Figure 3-8: End-to-End QoS with CAC: Bursty Sources

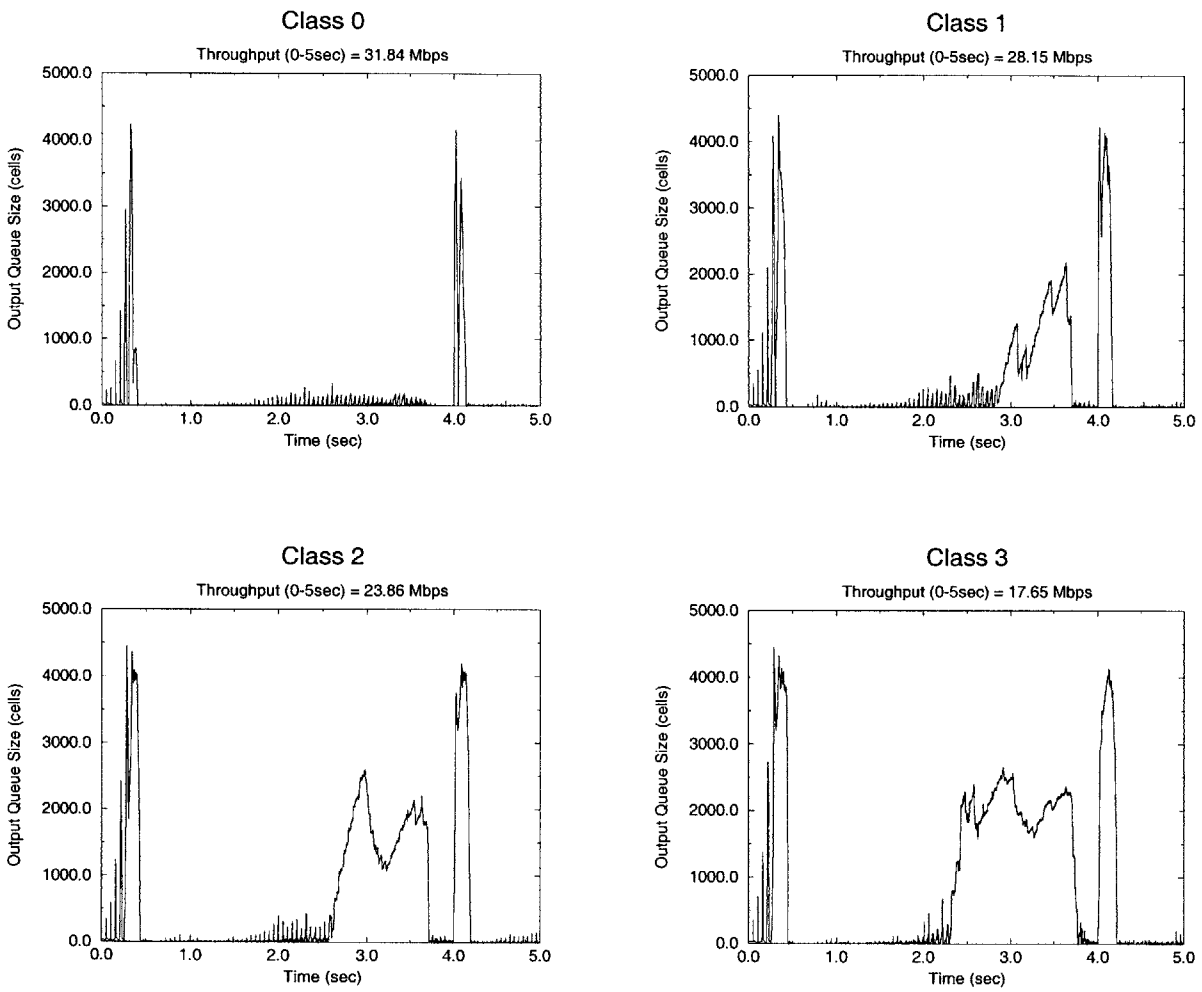


Figure 3-9: Hop-by-Hop CoS: Bursty Sources

Chapter 4

Conclusions

We have introduced and studied several efficient packet discarding schemes as congestion avoidance mechanisms for ATM. These schemes are aimed to emulate RED, but have simpler hardware implementation. Among the schemes we have proposed, the Periodic Discard Scheme with Sampling (SPD) shows performance comparable to RED. We have also investigated how packet discarding schemes can be combined with traffic shaping and priority queue scheduling to provide differentiated services. We have presented a simulation study on two differentiated services models that are based on IETF framework. Our results show that with greedy sources, the End-to-End QoS model yields better performance than the Hop-by-Hop CoS model. On the other hand, with bursty traffic, the Hop-by-Hop CoS model performs better than the End-to-End QoS model.

Bibliography

[1] M. Allman, D. Glover, and L. Sanchez. Enhancing TCP over satellite channels using standard mechanisms, September 1998.

[2] M. Allman, V. Paxson, and W. Stevens. TCP congestion control, December 1998.

[3] Y. Bernet, J. Binder, S. Blake, M. Carlson, S. Keshav, E. Davies, B. Ohlman, D. Verma, Z. Wang, and W. Weiss. A framework for differentiated services, October 1998.

[4] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services, December 1998.

[5] O. Elloumi and H. Afifi. RED algorithm for ATM. *IEEE ATM Workshop'97*, May 1997.

[6] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, August 1993.

[7] E. Hashem. Anaylsis of random drop for gateway congestion control. Thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, USA, November 1989.

[8] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared systems. Technical Report DEC-TR-301, Digital Equipment Coporation, September 1984.

[9] H. Li, K.-Y. Siu, H.-Y. Tzeng, C. Ikeda, and H. Suzuki. A simulation study of TCP performance in ATM networks with ABR and UBR services. In *Proceedings of IEEE INFOCOM'96*, March 1996.

[10] L. Peterson and B. Davie. *Computer Networks.* Morgan Kaufmann, San Francisco, USA, 1996.

[11] Shirish Sathaye, Editor. Traffic management specification version 4.0, April 1996.

[12] W. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols.* Addison-Wesley, San Francisco, USA, 1995.

[13] A. S. Tanenbaum. *Computer networks.* Prentice-Hall International, Englewood Cliffs, 1981.

Appendix

Pseudocode for PRED:

```
Initialization:
    Receiving_i = 0
    Discarding_i = 0
When a cell is coming to an ATM switch:
    if Q > Qmax
        discard the cell
        Discarding_i = 1
        Receiving_i = 1
    else if Discarding_i = 1
        discard the cell
    else if Receiving_i = 0
        Receiving_i = 1
        if Q > max_th
            Discarding_i = 1
            discard the cell
        else if Q > min_th
            if X => N
                discard the cell
                Discarding_i = 1
                X = 0
            else
                accept the cell
                X = X + 1
```

else if $Q <= min_{th}$

accept the cell

if incoming cell is an EOM cell

$Receiving_i = 0$

$Discarding_i = 0$ $\square$

Pseudocode for SPD:

Initialization:

$Receiving_i = 0$

$Discarding_i = 0$

$X = 0$

$N_{low} = 100$

$N_{high} = 200$

For every sampling time interval (SI):

Calculate the new average queue size avg

$avg = (1 - w_q)avg + w_q \cdot q$

if $avg >= mid(min_{th}, max_{th})$

$N = N_{low}$

else if $avg < mid(min_{th}, max_{th})$

$N = N_{high}$

When a cell is coming to an ATM switch:

if $Q > Qmax$

discard the cell

$Discarding_i = 1$

$Receiving_i = 1$

else if $Discarding_i = 1$

discard the cell

else if $Receiving_i = 0$

$Receiving_i = 1$

if $Q > max_{th}$

$Discarding_i = 1$

discard the cell

else if $Q > min_{th}$

if $X => N$

discard the cell

$Discarding_i = 1$

$X = 0$

else

accept the cell

$X = X + 1$

else if $Q <= min_{th}$

accept the cell

if incoming cell is an EOM cell

$Receiving_i = 0$

$Discarding_i = 0$ □

Pseudocode for SPDI:

```
Initialization:
    Receiving_i = 0
    Discarding_i = 0
    max_pb = 1/200
    X = 0
For every sampling time interval (SI):
    Calculate the new average queue size avg and Counter Threshold (N)
        avg = (1 - w_q)avg + w_q · q
        if min_th < avg <= max_th
            P_b = max_pb · (avg - min_th)/(max_th - min_th) + max_pb
            N = (int)P_b
When a cell is coming to an ATM switch:
    Calculate the new average queue size avg
    if Q > Qmax
        discard the cell
        Discarding_i = 1
        Receiving_i = 1
    else if Discarding_i = 1
        discard the cell
    else if Receiving_i = 0
        Receiving_i = 1
        if Q > max_th
            Discarding_i = 1
            discard the cell
        else if Q > min_th
            if X > N
                    discard the cell
                    X = 0
                else
                    accept the cell
```

$X = X + 1$

else

accept the cell

$X = X + 1$

if incoming cell is an EOM cell

$Receiving_i = 0$

$Discarding_i = 0$ □

Pseudocode of RED

```
Initialization:
  Receiving_i = 0
  Discarding_i = 0
  avg = 0
  count = -1
for each packet arrival
  calculate the new average queue size avg:
    if the queue is nonempty
      avg = (1 - w_q)avg + w_q q
    else
      m = (time - q_time)/s
      avg = (1 - w_q)^m avg
if Q ≥ Qmax
  discard the cell
  Discarding_i = 1
  Receiving_i = 1
else if Receiving_i = 0 /* first cell */
  Receiving_i = 1
  if min_th ≤ avg < max_th
    increment count
    calculate probability p_a:
      p_b = max_p(avg - min_th)/(max_th - min_th)
      p_a = p_b/(1 - count · p_b)
    generate a random number between 0 and 1
    if random_number < p_a
      discard the cell
      Discarding_i = 1
      count = 0
    else
      accept the cell
```

```
  else if max_th < avg
    discard the cell
    Discarding_i = 1
    count = 0
  else /* avg < min_th */
    accept the cell
    count = -1
else /* not first cell */
  if Discarding_i = 1
    discard the cell
    count = 0
  else
    accept the cell
if incoming cell is an EOM cell
  Receiving_i = 0
  Discarding_i = 0

when queue becomes empty
  q_time = time
```

□

Pseudocode for Per-VC queueing with PRED:

Q_{max} is maximum queue size.

$PRED_{th}$ is the lower periodic discard threshold.

Q_i is the Queue length for VC_i.

VCN is the total number of VCs.

N: one out of every N packets is discarded.

Pseudocode for Scheme A:

```
Initialization:
    Receiving_i = 0
    Discarding_i = 0
    Count_i = 0
When a cell is coming to an ATM switch:
    if Q >= Qmax
        Discarding_i = 1
        Receiving_i = 1
        discard the cell
    else if Discarding_i = 1
        discard the cell
    else if Receiving_i = 0
        Receiving_i = 1
        if Q > EPDth
            Discarding_i = 1
            discard the cell
        else if Q > PREDth
            if Q_i > PREDth/VCN
                if Count_i > N
                    discard the cell
                    Count_i = 0
                else
```

accept the cell

$Count_i = Count_i + 1$

else

accept the cell

$Count_i = Count_i + 1$

else

accept the cell

if incoming cell is an EOM cell

$Receiving_i = 0$

$Discarding_i = 0$ □

Pseudocode for Scheme B:

```
Initialization:
    Receiving_i = 0
    Discarding_i = 0
    Count_i = 0
    TotalCount = 0
When a cell is coming to an ATM switch:
    if Q >= Qmax
        Discarding_i = 1
        Receiving_i = 1
        discard the cell
    else if Discarding_i = 1
        discard the cell
    else if Receiving_i = 0
        Receiving_i = 1
        if Q > EPDth
            Discarding_i = 1
            discard the cell
        else if Q > PREDth
            if Q_i > PREDth/VCN
                if Count_i > N or TotalCount > N
                    discard the cell
                    Count_i = 0
                    TotalCount = 0
                else
                    accept the cell
                    Count_i = Count_i + 1
                    TotalCount = TotalCount + 1
            else
                accept the cell
```

$Count_i = Count_i + 1$

$TotalCount = TotalCount + 1$

else accept the cell

else

accept the cell

if incoming cell is an EOM cell

$Receiving_i = 0$

$Discarding_i = 0$ □

Pseudocode for Scheme C:

```
Initialization:
    Receiving_i = 0
    Discarding_i = 0
    TotalCount = 0
When a cell is coming to an ATM switch:
    if Q >= Qmax
        Discarding_i = 1
        Receiving_i = 1
        discard the cell
    else if Discarding_i = 1
        discard the cell
    else if Receiving_i = 0
        Receiving_i = 1
        if Q > EPDth
            Discarding_i = 1
            discard the cell
        else if Q > PREDth
            if Q_i > PREDth/VCN
                if TotalCount > N
                    discard the cell
                    TotalCount = 0
                else
                    accept the cell
                    TotalCount = TotalCount + 1
            else
                accept the cell
                TotalCount = TotalCount + 1
        else
            accept the cell
    else    accept the cell
```

if incoming cell is an EOM cell

$Receiving_i = 0$

$Discarding_i = 0$ $\square$

Pseudocode for Scheme D:

```
Initialization:
    Receiving_i = 0
    Discarding_i = 0
    TotalCount = 0
When a cell is coming to an ATM switch:
    if Q >= Qmax
        Discarding_i = 1
        Receiving_i = 1
        discard the cell
    else if Discarding_i = 1
        discard the cell
    else if Receiving_i = 0
        Receiving_i = 1
        if Q > EPDth
            Discarding_i = 1
            discard the cell
        else if Q > PREDth and Q_i > PRED_th/VCN
            if TotalCount > N
                discard the cell
                TotalCount = 0
            else
                accept the cell
                TotalCount = TotalCount + 1
        else
            accept the cell
    if incoming cell is an EOM cell
        Receiving_i = 0
        Discarding_i = 0
```

□