A Performance Evaluation of Multiplexer Scheduling Algorithms

Zhao Chen

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montreal, Quebec, Canada

November 2003

Your file Votre référence
ISBN: 0-612-91011-3
Our file Notre référence
ISBN: 0-612-91011-3

ABSTRACT

A Performance Evaluation of Multiplexer Scheduling Algorithms

Zhao Chen

In the future, Broadband Integrated Services Digital Network (B-ISDN) is expected to serve voice, video, data and other signals in a single network. Asynchronous Transfer Mode (ATM) and Internet have emerged as two competing networking architectures for the realization of B-ISDN. ATM and Internet are connection-oriented and connectionless packet-switching technologies respectively. A main roadblock in this integration is how to meet Quality of Service (QoS) requirements of different services. Real-time traffic such as voice and video are delay sensitive and loss tolerant while data is usually loss sensitive and delay tolerant. Thus there is a need to provide support for service differentiation in both network architectures. An important network device for service differentiation is the scheduling algorithms implemented at the switch and router queues.

The objective of this thesis has been to provide a comprehensive performance study of these scheduling algorithms. The main scheduling algorithms are First In First Out (FIFO), Priority Queueing (PQ), Fair Queueing (FQ) and Weighted Round Robin (WRR) service disciplines. Several derivatives of these algorithms were introduced with varying efficiency and complexity. We compare these algorithms and their derivatives with respect to mean message delay, probability distribution of delay and call blocking probability performance measures. It is known that mean delay is independent of service discipline if service doesn't depend on message size. We observe that service in both PQ and FQ may be made message size aware and we are able to obtain mean message delay results very different than FIFO service discipline. The WFQ algorithm has the desired property of isolating the service given to each class of traffic from the others. However, implementation of WFQ is complex and it doesn't scale well with the number of traffic flows. WRR is much simpler and it is known to behave similar to WFQ. Our study shows that the probability distribution of message delay under WRR is very close to that of WFQ. Thus WRR may be preferred to WFQ in many situations. We also study call admission control under different scheduling algorithms and present call blocking probabilities.

As the Internet backbone speed increases, the users are demanding higher throughput from the network. However, TCP congestion control algorithm stands in the way of meeting these demands. Among the proposed solutions are to modify the congestion control algorithm and to increase the allowed message size in the network. The first solution results in very large window sizes that will further increase the already heavy processing load of the routers. However, both solutions have the common drawback that they fail to protect low-throughput users from the high-throughput users. In this work, we propose that low and high throughput traffic queues up separately and then use a WFQ or WRR server to protect the bandwidth share of low-throughput users. The simulation study shows satisfactory results for both types of users.

Acknowledgments

I would like to express my sincere gratitude to my thesis supervisor for his invaluable guidance, encouragement and help throughout the entire course of this work.

I also would like to appreciate all my friends for their encouragement and moral support.

Finally, I wish to thank my family for their continuous support and encouragement through every step of my life.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ABR	Available Bit Rate
AF	Assured Forwarding
ATM	Asynchronous Transfer Mode
BRFQ	Bit Round Fair Queueing
B-ISDN	Broadband Integrated Services Digital Network
Bps	Bit per second
CBR	Constant Bit Rate
CLR	Cell Loss Rate
DS	Differentiated Service
DWRR	Deficit Weighted Round Robin
EF	Expedited Forwarding
FIFO	First In First Out
FQ	Fair Queueing
FR	Frame Relay
GPS	General Processor Sharing
Gbps	Gigabit per second
GFR	Guaranteed Frame Rate
IPP	Interrupted Poisson Process
ITU	International Telecommunications Union
IETF	Internet Engineering Task Force
IS	Integrated Services

ISA	Integrated Services Architecture
Kbps	Kilobits per second
LAN	Local Area Network
LB	Leaky Bucket
Mbps	Megabits per second
MMPP	Modulated Poisson Process
PHBs	Per Hop Behaviors
PQ	Priority Queueing
QoS	Quality of Service
RT-VBR	Real-time Variable Bit Rate
RSVP	Resource Reservation Protocol
SLA	Service Level Agreement
SCFQ	Self-clocked Fair Queueing
UBR	Unspecified Bit Rate
WAN	Wide Area Network
WFQ	Weighted Fair Queueing
WRR	Weighted Round Robin

Chapter 1

Introduction

In this decade, dramatic changes have been taking place in communication networking techniques. High-speed networks now dominate both the wide-area (WAN) and local area network (LAN) markets. The advent of high-speed networking has introduced opportunities for new applications such as video conferencing, scientific visualization and media imaging and it is in turn driven by the popularity of those applications.

1.1 Evolution of Computer Networks

Computer networking involves the two fields of the communication and computer. On the one hand, communication networks provide the necessary means for the data transmission and exchange between computers; on the other hand, digital computation technology is used in the telecommunication technology and improves the performance of communication networks.

Circuit switching and packet switching are two different technologies that evolved over a long period of time. Public telephone system is based on the circuit switching, which involves three phases: circuit establishment, data transfer and circuit disconnection. In circuit switching, channel capacity is dedicated for the duration of the connection, even when no information is being sent. Circuit switching is a good solution for voice, which has a relatively continuous flow of information. However, it is not a good solution if the information is bursty as the case of data. That intermittent type of transmission causes the low utilization of the circuit-switched connection. In addition, because the data transmission rates in the computers and terminals are different, it is difficult to communicate between different types of terminals.

To overcome the shortcomings of circuit switching, packet switching technology has been developed for data exchange. Information is sent in packets, and each packet has a header with the destination address. A packet is passed through the network from node to node until it reaches its destination. Compared with circuit switching, packet switching has the following advantages:

- Bandwidth efficiency is greater. A single node-to-node link can be dynamically shared by many packets over time. The packets are queued up and transmitted over the link.
- Two stations of different data rates can exchange packets because each station connects to its node at its proper data rate.
- On a packet-switched network, when traffic becomes heavy, packets are still accepted with increased delivery delay.
- The traffic may be prioritized, so that the higher-priority packets will experience less delay than lower-priority packets.

Packet switching also leads to some new problems. Packets experience a random delay due to processing and queueing in the node. In addition, control information in each packet results in extra overload. The packet switching network needs specified management and control mechanisms.

In packet switching, two different techniques can be used, virtual circuits and datagrams. A virtual circuit imitates circuit switching, which involves the same three phases: call set-up, transfer of packets, and call termination. Before any packet is allowed to be sent, a logical connection is established between the sender and the receiver; as a result, all packets follow the same path. However, in packet switching, channel capacity on each transmission link is not dedicated to a virtual circuit. Rather, the transmission link is shared by all the virtual circuits that pass through it. For example, ATM networks are packet-switched networks with virtual circuits.

In datagrams, no call set-up is required, and each packet is routed through the network individually. The IP network, used in the Internet, is a packet-switched network based on datagrams.

1.2 The Need for Quality of Service

As the network capacity is increased, network users have more demands on real-time, multimedia and multicasting applications. They are beginning to combine real-time applications such as voice and video, which have a limited tolerance for network latency, with non-real time data traffic. These different service classes have different performance requirements in terms of throughput, end-to-end delay, delay jitter and packet loss rate. Current IP networks only offer a best-effort service, which treats all packets equally. Under a best-effort scheme, the performance of each flow can be degraded significantly when the network is overloaded. There is an urgent need to provide network services with performance guarantee and to develop algorithms supporting these services.

The new approach for network design is to carry large volume of traffic with different quality of service requirements over networks operating at very high data rate. IP network and ATM network are two types of network facilities that dominate high-speed scene and have many common features.

1.2.1 The Concept of QoS

Quality of Service (QoS) refers to the capability of a network to meet user traffic requirements over various network technologies including Ethernet and 802.1 networks, wireless networks, IP-based networks, Asynchronous Transfer Mode (ATM), and Frame Relay (FR). It also can be interpreted as a method for providing preferential treatment to some arbitrary amount of network traffic, as opposed to all traffic being treated as "best effort".

1.2.2 Quality of Service Parameters

A number of different parameters can be used to express the service quality of a connection.

- Throughput: Throughput defines how much data is delivered to the end system in a fixed time interval. A minimum throughput value may be required for many applications. It needs to be determined to ensure that components and links are sized correctly.
- Delay: End-to-end delay is the time that takes to transfer a packet end-to-end, that is from the source to the destination. The end-to-end delay is made up of a fixed and a variable component. The fixed delay is the sum of all fixed delays that a packet encounters from the source to the destination, such as transmission times and propagation delays. The variable delay is the sum of all variable delays that a packet encounters between the source and the destination. These delays are primarily due to queueing delays in the switches along a packet's path.

- Jitter (Delay Variation): This is the variation in the inter-packet arrival time introduced at the destination as a result of the variable component of the end-to-end delay. If a packet arrives at a switch when it has a large queue length, and another packet arrives when the switch has a small queue length, the difference between the queue sizes in these two cases cause a packet's total end-to-end jitter. Removing jitter requires storing packets in buffers at the receiver and holding them long enough to allow the slowest packets to arrive in time to be placed in correct sequence.
- Packet loss: IP networks don't guarantee delivery. Packets will be dropped under heavy load and during periods of congestion. Because there is no flow control in ATM networks, Cell Loss Rate (CLR) is also a popular QoS parameter.

1.3 ATM networks and QoS Mechanisms

As mentioned earlier, ATM network and IP network dominate the high-speed network scene. Next we will present the main characteristics of both of these networks.

1.3.1 The Asynchronous Transfer Mode (ATM)

ATM is a technology that provides a single platform for the transmission of voice, video and data at specified quality of service and at speeds varying from fractional T1 (i.e. nX64 Kbps) to Gbps [2].

ATM was standardized by ITU-T (ITU Telecommunications Standardization Sector) in 1987. It is based on packet switching technology and it is connection- oriented. An ATM packet is known as a cell. It has a fixed-size of 53 bytes. It consists of a payload of 48 bytes and a header of 5-bytes. Unlike IP networks, ATM has built-in mechanisms that permit it to provide different quality of service to different types of traffic. The small, fixed-length cells allow fast and efficient multiplexing and switching of traffic with different QoS constraints. Due to its low queueing delay and delay variance, ATM technology networks are well suited for multimedia applications, and can handle any kind of traffic from voice to bursty video stream at any speed [3]. Until now, ATM is still the only networking technology that provides quality of service guarantees to different services.

ATM was originally defined to run over high-speed links and is the preferred architecture for Broadband Integrated Service Digital Network (B-ISDN). There is neither error control nor flow control between two adjacent ATM nodes. Error control is not necessary because of the high reliability of fiber-based transmission links. Congestion control schemes permit the ATM network operator to carry as much traffic as possible without affecting the QoS requirements of the users.

1.3.2 Quality of Service in ATM networks

Next, we briefly describe the QoS features of ATM networks. An ATM service category is in simple terms of a QoS class. Each service category is associated with a set of traffic parameters and a set of QoS parameters. The traffic parameters are used to characterize the traffic transmitted over a connection, and the QoS parameters are used to specify the performance requirements of each class: the cell loss rate, the end-to-end delay, and delay jitter. Functions such as call admission control and bandwidth allocation are applied differently to each service category. The switch and multiplexer scheduling algorithms provide mechanisms to meet QoS requirements of different services.

There are six different categories provided by the ATM networks: Constant Bit Rate (CBR), Real-time Variable Bit Rate (RT-VBR), Non-Real-Time Variable Bit Rate (NRT-VBR), Available Bit Rate (ABR), Unspecified Bit Rate (UBR), and Guaranteed Frame Rate (GFR). CBR and RT-VBR are for real-time applications. The remaining service categories are for non-real-time applications.

The CBR service is intended for real-time applications which transmit at constant bit rate, such as circuit emulation service and constant-bit rate video. The RT-VBR service is intended for real-time applications that transmit at a variable bit rate, such as encoded video and voice. The NRT-VBR service is intended for non-real-time applications that transmit at a variable bit rate. The UBR service is a best-effort type of service for non-real-time applications with variable bit rate. It is intended for applications that involve the transfer of data, such as file transfer, web browsing and email. The ABR service is intended for non-real-time applications that can vary their transmission rate according to the congestion level in the network. The GFR Service is for non-real-time applications that may require a minimum guaranteed rate.

1.4 IP Networks and QoS Mechanisms

Next, we describe the main features of IP networks.

1.4.1 IP Networks

The web growth in the Internet is the dominating factor in the development of new protocols and mechanisms for data communications and computer networking. The TCP/IP protocol suit running on the Internet combines logic for routing through an IP network with end-to-end control.

IP networks were designed to provide a best-effort, fair delivery service that treats all packets equally. It was intended for applications that are relatively delay insensitive, can tolerate variations in throughput, and can tolerate packet loss. With the tremendous increase in traffic volume and the types of service, IP networks are being asked to support new applications such as video conferencing, scientific visualization and media imaging over high-capacity links. These different service classes have different performance requirements in terms of throughput, end-to-end delay, delay jitter and packet loss rate. There is a strong need to provide network services with suitable QoS and to develop algorithms supporting these services.

To provide different QoS commitments, two different traffic management frameworks have been defined by the IETF (Internet Engineering Task Force): Integrated Services (IS) and Differentiated Services (DS).

1.4.2 Integrated Services

The Integrated Services Architecture (ISA) was proposed in RFC 1633 [4] to support real-time traffic as well as "best-effort" traffic. It requires resources such as bandwidth and buffers to be explicitly reserved for a given flow to ensure that the application receives its requested QoS.

Network nodes classify incoming packets and use reservations to provide differentiated services. It performs resource reservation using a dynamic signaling protocol and employs admission control, packet classifier, and packet scheduler to achieve desired QoS. Packet classifier identifies flows that are to receive a certain level of service. Packet scheduler handles the forwarding of different packet flows in a manner that ensures that QoS commitments are met. Admission control determines whether a router has the necessary resources to accept a new flow. This model is relatively complex because it provides per flow service, therefore it has difficulties in scaling to large backbone networks.

The Resource Reservation Protocol (RSVP) is used by the Integrated Services model to provide the reservation messages required to set up a flow with a requested QoS across the network. RSVP is used to inform each router of the requested QoS. If the flow is found admissible, each router in turn adjusts its packet classifier and scheduler to handle the given packet flow.

The integrated services model defines three categories of service: guaranteed service, controlled-load service, and best effort service. The guaranteed service can be used for applications that require real-time service delivery. It provides a firm bound on the end-to-end delay for a flow. The controlled-load service is intended for adaptive applications that can tolerate some delay but are sensitive to traffic overload conditions.

1.4.3 Differentiated Services

Because of the scalability and complexity issues associated with the integrated services model, IETF has introduced another service model called the differentiated services (DS) model [5], which is designed to provide a simple, easy-to-implement, low-overhead tool to support a range of network services that are differentiated on the basis of performance.

The efficiency and easy-deployment of DS model is based on several key characteristics:

- ToS field in IPv4 header or the IPv6 Traffic Class field is used for differing QoS treatment, these fields mark a packet to receive a particular per-hop behavior at each network node. No change is required to IP.
- A service level agreement (SLA) is established between the service provider and the customer. Existing applications need not be modified to use DS.
- DS provides a built-in aggregation mechanism. Per flow service in IS model is replaced with per aggregate service. All traffic with the same DS field is treated the same way by the network, this provides good scalability to larger networks and traffic loads.
- DS is implemented in individual router by queueing and forwarding packets based on the DS field. Routers do not have to save state information on packet flows.
- Complex processings such as classification, marking and policing are moved from the core of a network to the edge. Only packet handling requirements need to be provided in the core of the network.

Two types of per hop behaviors (PHBs) are defined in the DS model, which can be associated with a specific differentiated service. The expedited forwarding (EF) PHB provides a low-loss, low-latency, low-jitter, assured bandwidth, end-to-end service through DS domains. Such a service is referred to as a premium service. The assured forwarding (AF) PHB provides a service superior to best-effort but one that does not require the reservation of resources within an internet and does not require the use of detailed discrimination among flows from different users.

1.5 Packet Scheduler

The actions of the packet scheduling algorithms at the ATM switches or IP routers are one of the most important issues in providing service performance guarantees. This is used in both integrated services and differentiated services mechanisms.

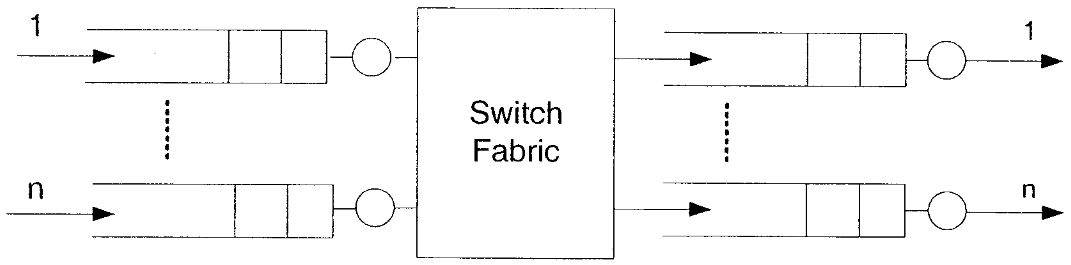

Figure 1.1: A General Structure Inside a Router

In a packet-switched network, packets from different flows interact with each other at each switching node and share a limited network resource; without proper control, these interactions may adversely affect the performance seen by different services. The packet scheduler at the switching node, which controls the order in which packets are served, determines how packets from different flows interact with each other and fairly allocates the limited shared resources. From Figure 1.2, the basic model of a switching node is a single server with n FIFO queues each being fed by an independent traffic flow. The scheduling algorithm determines the service order of head-of-line (HOL) packets in the n queues.

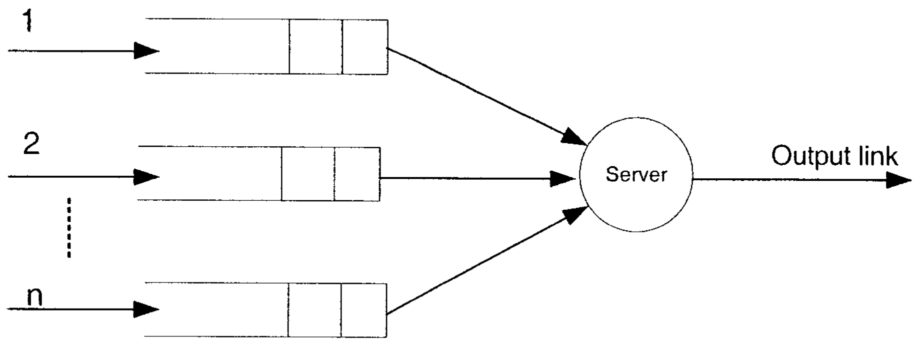

Figure 1.2: Packet Scheduler Model

Generally, there are several tasks that a scheduling algorithm should accomplish:

- Support the fair distribution of bandwidth to each of the different service classes competing for bandwidth at the output port.
- Furnish protection (firewalls) between the different service classes at an output port, so that a poorly behaved service class in one queue cannot impact the bandwidth and delay delivery to other service classes assigned to other queues at the same output port.
- Allow other service classes to access bandwidth that is assigned to a given service class if the given service class is not using all of its allocated bandwidth.
- Provide an algorithm that can be implemented in hardware, so it can arbitrate access to bandwidth on the high-speed router interfaces without negatively impacting system forwarding performance. If the scheduling algorithm cannot be implemented in hardware, then it can be used only at the low-speed router or switch interfaces.

Scheduling algorithms are the main mechanisms available to networks to ensure that QoS requirements of different services are met. The scheduling algorithms may be classified as follows:

- FIFO algorithm

This is the simplest scheduling algorithm that serves the packets on the order of their arrivals.

- Priority Queueing (PQ)

The queues are assigned priorities and served according to the non-preemptive priority service discipline.

- Fair Queueing (FQ)

This scheduling algorithm simulates the Processor Sharing (PS) service discipline. Many variations of PS have been introduced for delivering different amount of service to different queues.

- Round Robin Queueing (RRQ)

This scheduling algorithm serves the queues in round-robin order. The server is able to give different amount of service to different queues either by serving different number of packets at each queue or visiting a queue multiple times during a cycle.

The above scheduling algorithms are applicable in both ATM and IP networks.

1.6 The Objectives of the Work

The objectives of this work have been to give a comprehensive performance comparison of the packet scheduling algorithms introduced in the previous section. The performance comparisons have been made with respect to the following measures,

- average message delay
- standard deviation of delay
- probability distribution of delay

- call blocking probabilities

We have chosen simulation as the main tool for the performance study because of the large number of scheduling algorithms under consideration. As far as we know, there is no such detailed comparison of these algorithms in the literature. Most previous works compared the average message delay of the algorithms in the same class among two or three algorithms. These comparisons are made when a new variant of a class of algorithms is introduced in order to demonstrate that it performs better than the other algorithms in the same class [6][7][8][9][10][11][12]. Thus, we provide performance comparison of all the classes simultaneously at the probability distribution of message delay and call blocking probability level.

Recently, there has been a growing interest in the performance of TCP congestion control algorithm in the high-speed links. It has been shown that the additive increase/multiplicative decrease adjustment of congestion window by the TCP congestion control algorithm will limit the throughput of TCP users in the high-speed links. Two solutions have been proposed to this problem, faster increase and slower decrease of congestion window size [13][14] and increasing the maximum packet size [15][16]. Neither of these two algorithms protects the low-throughput users from the high-throughput users. Further the first solution results in large window sizes that increase the already heavy router processing load. In this work, we prefer the second solution and propose to store the low and high-throughput user traffic at two different queues in order to protect the bandwidth share of the low-throughput users.

1.7 Outline of the Rest of the Thesis

The outline of the rest of the thesis is as follows.

Chapter 2 describes the different scheduling algorithms and discusses their advantages and disadvantages.

Chapter 3 introduces the simulation program and then presents the simulation results for the different scheduling algorithms.

Chapter 4 first presents the performance of TCP in the presence of real-time traffic under the scheduling algorithms. Then it studies the behavior of TCP in the high-speed links with large message sizes.

Chapter 5 concludes this thesis and gives some suggestions for future work.

Chapter 2

Scheduling Algorithms

In this chapter, we present a survey of several well-known traffic scheduling algorithms for serving multiple queues. In a later chapter, we will study the performance of all these algorithms.

2.1 Background

Providing QoS guarantees in a packet-switched network requires the use of traffic scheduling algorithms at the switches (or routers) and multiplexers. The function of a scheduling algorithm is to select, for each outgoing link of the switch, the packet to be transmitted in the next slot from the available packets belonging to the flows sharing that output link.

Some notions are given before further description:

- A flow is defined as a sequence of packets that have one or more characteristics in common. For example, each traffic class such as voice, video may be considered as a flow; similarly, all packets with identical destination may be considered as a flow. The packets belonging to different flows are usually queued separately while they await transmission.
- A scheduler dequeues packets from these queues and forwards them for transmission.

- Fairness in the allocation of a resource among multiple requesting entities means every entity receives equal right to the resource for the equal demand; for unequal demands access to the resource, is as follows [17]:
  - The resource is allocated in the order of increasing demand.
  - No entity receives a share of the resource larger than its demand.
  - Requesting entities with unsatisfied demands get equal shares of the resource.

There are many different scheduling algorithms, each attempting to find the correct balance between complexity, control and fairness.

A packet scheduler is generally classified as either work-conserving or non-work-conserving. With work-conserving scheduler, the server is never idle when a packet is buffered in the system. FIFO, PQ [1], FQ [7] [18], PS-based Queueing [6] [19] [20] [21], WRR [22] and DRR [9] are all work-conserving schedulers. With a non-work-conserving scheduler, the server could be idle even when there are buffered packets in the system. This kind of discipline is proposed to control the traffic distortions in the network such as Stop-and-Go [23], Hierarchical Round Robin (HRR) [24], and Rate-Controlled Static Priority [25].

Furthermore, based on its internal architecture a packet scheduler is also classified as either sorted-priority or frame-based. With sorted-priority scheduler, a system potential (a global variable) is updated each time a packet arrives or departs, and a timestamp is computed as a function of the system potential for each packet, then packets are sorted and transmitted based on their timestamps. Priority Queueing (PQ), Weighted Fair Queueing (WFQ) belongs to this kind of scheduler. With frame-based scheduler, time is split into fixed or variable length frames. Each flow makes a reservation in terms of maximum traffic that it is allowed to transmit during a frame period. Weighted Round Robin, Deficit Round Robin belongs to this category of scheduler. Next, we will describe the main scheduling algorithms one by one.

2.2 First In and First Out (FIFO)

FIFO queueing is the most basic scheduling algorithm. In FIFO queueing, all packets are treated equally by placing them into a single queue according to their arrival time, and then servicing them in the order of their arrival times.

FIFO queueing may be modeled as shown in the Figure 2.1:

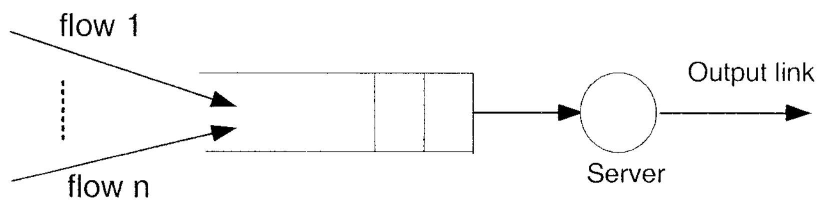

Figure 2.1: FIFO model

FIFO queueing offers the following benefits:

- For software-based routers, FIFO queueing places an extremely low computational load on the system compared with more elaborate queue scheduling algorithms.
- The behavior of a FIFO queue is very predictable. Packets are not reordered and the maximum delay is determined by the maximum depth of the queue.

- As long as the queue depth remains short, FIFO queueing provides simple contention solution for network resources without adding significantly to the queueing delay experienced at each hop.

FIFO queueing also has the following drawbacks:

- A single FIFO queue does not allow routers to provide service differentiation.
- A single FIFO queue impacts all flows equally during congestion, and the mean queueing delay for all flows increases as congestion increases. As a result, FIFO queueing can result in increased delay, delay jitter, and packet loss for real-time applications.
- If a number of smaller packets are queued behind longer packets, FIFO queueing results in a larger average delay per packet than if the shorter packets were transmitted before the longer packet. In general, flows of larger packets get better service.
- A bursty flow can consume the entire buffer space of a FIFO queue, and that causes all other flows to be denied service until the burst is serviced. This can result in increased delay, jitter, and loss for the other well-behaved flows traversing the network.

2.3 Priority Queueing (PQ)

Priority queueing is the basis for a class of scheduling algorithms that are designed to provide a relatively simple method for supporting different service classes. In classic PQ, packets are first classified by the system and then placed into different priority queues.

Packets are scheduled from the head of a given queue only if all higher priority queues are empty. Within each of the priority queues, packets are scheduled in FIFO order.

The model of PQ is shown in Figure 2.2:

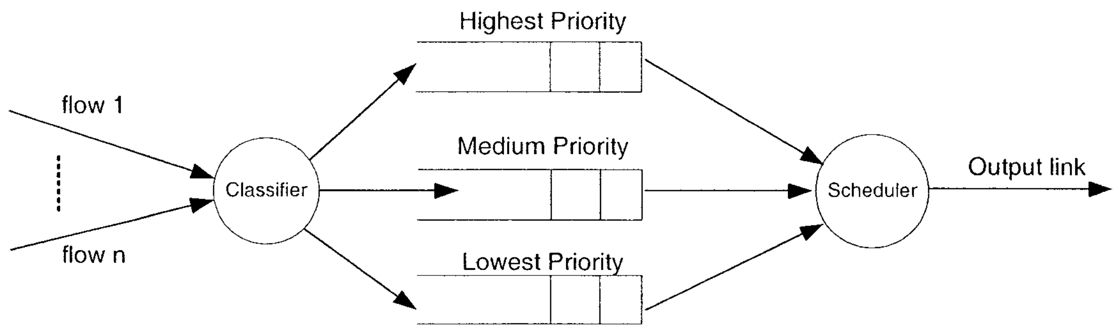

Figure 2.2: PQ model

PQ scheduling offers the following benefits:

- For software-based routers, PQ places a relatively low computational load on the system compared with more elaborate scheduling algorithms.
- PQ allows routers to provide service differentiation to meet QoS requirements. For example, real-time applications get priority over applications that do not operate in real time through the priorities set.

PQ also poses the following limitations:

- If the amount of high-priority traffic is not policed or conditioned at the edges of the network, lower-priority traffic may experience excessive delay as it waits for unbounded higher-priority traffic to be served.

- If the volume of higher-priority traffic becomes excessive, the lower-priority traffic can be dropped as the buffer space allocated to low-priority queues starts to overflow. This ultimately leads to complete resource starvation for lower-priority traffic.
- A misbehaving high-priority flow can add significantly to the amount of delay and jitter experienced by other lower-priority flows sharing the same queue.

PQ can be configured in two modes: strict priority queueing and rate-controlled priority queueing. Strict PQ ensures that packets in a high-priority queue are always scheduled before packets in lower-priority queues. Rate-controlled PQ allows packets in a high-priority queue to be scheduled before packets in lower-priority queues only if the amount of traffic in the high-priority queues stays below a user-configured threshold.

2.4 Fair Queueing (FQ)

The primary goal of fair queueing is to serve flows in proportion to some pre-specified service shares, independent of the traffic load presented by the flows. Recent years, many fair queueing algorithms have been proposed.

2.4.1 Fair Queueing (FQ)

To overcome the drawbacks of FIFO and PQ scheduling, Nagle proposed a scheme called fair queueing [18]. Nagle' FQ is the foundation for a class of scheduling algorithms that are designed to ensure that each flow has a fair access to network resources and to prevent a bursty flow from consuming more than its fair share of output port bandwidth.

In FQ, a router maintains multiple queues at each output port (Figure 2.3) and each incoming packet is placed in the appropriate queue. The queues are serviced in a round-robin order, taking one packet from each non-empty queue in turn and skipping empty queues.

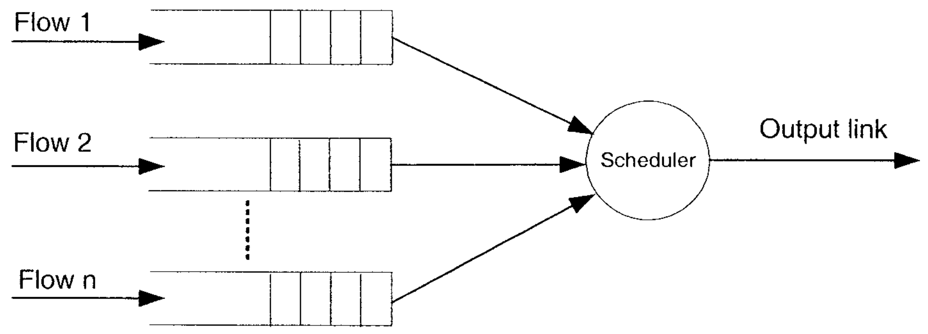

Figure 2.3: Fair Queueing model

The primary benefit of FQ is that an extremely bursty or misbehaving flow does not degrade the quality of service delivered to the other flows because flows are isolated from each other. If a flow attempts to consume more than its fair share of bandwidth, only its own queue is affected. So there is no impact on the performance of the other queues on the shared output port.

FQ also has several limitations:

- The objective of FQ is to allocate the same amount of bandwidth to each flow over time. FQ is not designed to support a number of flows with different bandwidth requirements.

- FQ provides equal amounts of bandwidth to each flow only if all of the packets in all of the queues are the same size. Flows containing larger packets get a larger share of output port bandwidth than those containing smaller packets.
- FQ is sensitive to the order of packet arrivals. If a packet arrives in an empty queue immediately after the queue is visited by the round-robin scheduler, the packet has to wait in the queue until all of the other queues have been serviced before it can be transmitted.
- FQ does not provide a mechanism to support real-time services easily.

2.4.2 Processor Sharing – based Fair Queueing (PS-based FQ)

2.4.2.1 Bit Round Fair Queueing (BRFQ)

A serious drawback to the FQ is that queues containing shorter packets are penalized. This disadvantage is overcome by bit-round fair queueing (BRFQ) [7], which uses packet size as well as flow identification to schedule packets. This discipline is based on an ideal policy Processor Sharing (PS) [26] that is not practical to implement.

In PS, only one bit from each queue is transmitted on each round. The queues containing longer packets no longer receive an advantage, and each busy queue receives exactly the same amount of service. In particular, if there are n queues and each of the queues is always active, each queue receives exactly $1/n$ of the available capacity. Next, we define some terms to describe PS:

$R(t)$ = the number of rounds made in the PS service discipline up to time t

$N(t)$ = the number of active queues at time t

R(*t*) can be thought as a virtual time, which records the rate of service seen by the packet at the head of a queue. An equivalent definition is

$$R'(t) = \frac{d}{dt} R(t) = \frac{1}{\max[1, N(t)]} \tag{2.1}$$

A packet of size L whose first bit gets service at time t_0 will have its last bit serviced L rounds later, at time t, then,

$$R(t) = R(t_0) + L \tag{2.2}$$

Then, we set

τ_i^{α} = arrival time of packet i to queue α

S_i^{α} = the values of R(t) when the packet i in queue α starts transmission

F_i^{α} = the values of R(t) when the packet i in queue α finishes transmission

L_i^{α} = the size of the packet i in queue α

Then, the following relations hold,

$$F_i^{\alpha} = S_i^{\alpha} + L_i^{\alpha} \tag{2.3}$$

$$S_i^{\alpha} = \max[F_{i-1}^{\alpha}, R(\tau_i^{\alpha})] \tag{2.4}$$

Since *R*(*t*) is a strictly monotonically increasing function whenever there are bits at the node, the ordering of the F_i^{α} values is the same as the ordering of the finishing times of various packets in the PS discipline.

Sending packets in a bit-by-bit round robin fashion in order to satisfy the requirements for an adequate queueing algorithm, is obviously unrealistic. BRFQ emulates this impractical algorithm in a practical packet-by-packet transmission scheme.

BRFQ is implemented by computing virtual starting and finishing times on the fly as if PS is running.

The BRFQ rule is this: Whenever a packet finishes transmission, the next packet to be transmitted is the one with the smallest value of F_i^{α}.

The order of transmission of packets, based on either real start time or real finish time, is not exactly the same for BRFQ and PS. Nevertheless, BRFQ gives a good approximation to the performance of PS. In fact, it is demonstrated that the throughput and average delay experienced by each flow under BRFQ converges to that under PS as time increases.

BRFQ is an improvement over FQ and FIFO in that it fairly allocates the available capacity among all active flows even with different packet sizes. However, it is not able to provide different amounts of capacity to different flows. If we set $W_i(\tau,t)/r_i$ as the normalized service provided to the flow i by the scheduler during the time interval $(\tau\text{-}t)$, where

r_i = the service rate allocated to flow i

r_j = the service rate allocated to flow j

$W_i(\tau,t)$ = the amount of service received by flow i during the time interval (τ,t)

$W_j(\tau,t)$ = the amount of service received by flow j during the time interval (τ,t)

A scheduler is called perfectly fair if $| W_i(\tau,t)/r_i - W_j(\tau,t)/r_j | = 0$ such as PS scheduler. For a packetized scheduler like BRFQ, there is a definition to describe its fairness: a scheduler is defined to be close to fair if $| W_i(\tau,t)/r_i - W_j(\tau,t)/r_j | \leq F^s$, here

F^s is a constant that does not dependent on the time interval (τ,t), and it is named as fairness of the scheduler [4].

2.4.2.2 Weighted Fair Queueing (WFQ)

WFQ is an enhancement of BRFQ to support unequal capacity allocation to different flows. WFQ is based on the ideal Generalized Processor Sharing (GPS) policy [19].

GPS is generated from the PS discipline to allow for arbitrary capacity allocations. With GPS, each flow α is assigned a weight ϕ_α that determines how many bits are transmitted from that queue during each round. Let us define each arrival and departure to/from the GPS queues as events. Let

τ = a busy period of the GPS system

t_j = the occurrence time of the j^{th} event

B_j = the set of flows that are busy in the time interval (t_{j-1}, t_j)

Virtual time $R(t)$ is defined as

$$R(0) = 0$$

$$R(t_{j-1} + \tau) = R(t_{j-1}) + \frac{\tau}{\sum_{i \in B_j} \phi_i}, \tau \leq t_j - t_{j-1}, j = 2,3 \ldots \quad (2.5)$$

The following modified equation from PS holds:

$$F_i^{\alpha} = S_i^{\alpha} + \frac{L_i^{\alpha}}{\phi_\alpha} \quad (2.6)$$

$$S_i^{\alpha} = \max[F_{i-1}^{\alpha}, R(\tau_i^{\alpha})] \quad (2.7)$$

So a nonempty flow i has a guaranteed rate,

$$r_i = \frac{\phi_i}{\sum_j \phi_j} r \tag{2.8}$$

where r is the output link data rate, and the sum is taken over all active queues.

GPS is an attractive scheme for a number of reasons:

- If a user requests a given service rate r_i for a flow, then the node can grant the request if sufficient capacity is available and can assign the proper weight to guarantee the service.
- The delay experienced by flow α can be bounded as a function of its queue length, independent of the other queues. Schemes such as FIFO and FQ do not have this property; by varying the ϕ_α, we have the flexibility of treating the flows differently from each other. For example, when the all ϕ_α are equal, the system reduces to PS.
- Most importantly, it is possible to make worst-case network queueing delay guarantees when the sources are constrained by leaky buckets. Thus, GPS is particularly attractive for flows sending real-time traffic such as voice and video.

WFQ is a packetized version of GPS and tries to emulate GPS on a packet-by-packet basis. WFQ approximates GPS by calculating and assigning a virtual finish time to each packet. Given the bit rate of the output port, the number of active queues, the relative weight assigned to each of the queues, and the length of each of the packets in each of the queues, it is for the scheduling algorithm to calculate and assign a virtual finish time to each arriving packet. The scheduler then selects and forwards the packet that has the earliest finish time F_i^{α} from all of the queued packets.

However, weighted fair queueing comes with several limitations:

- WFQ implements a complex algorithm that requires the maintenance of a significant amount of per-service class state and iterative scans of state on each packet arrival and departure.
- Computational complexity impacts the scalability of WFQ when attempting to support a large number of service classes on high-speed interfaces. Furthermore, on high-speed interfaces, minimizing delay to a single packet transmission may not be worth the computational expense if the insignificant serialization delay introduced by high-speed links and the lower computational requirements of the other scheduling algorithms are taken into account.

2.4.2.3 Worst-case Weighted Fair Queueing (WF^2Q)

Worst-case Fair Weighted Fair Queueing (WF^2Q) [20] is an enhancement to WFQ that uses both the virtual start and finish times of packets in the corresponding GPS to achieve a more accurate simulation of a GPS.

Recall that in WFQ system, when the server chooses the next packet for transmission at time t, it selects among all the packets that are backlogged in the queues, the first packet that would complete service (packet with smallest value of F_i^{α}) in the corresponding GPS system. In WF^2Q system, when the next packet is chosen for service at time t, rather than selecting it from among all the packets in the queues as in WFQ, the server only considers the set of packets that have started (and possibly finished) receiving service in the corresponding GPS system at time t and selects the packet from them that would complete service first. Formally, the packets set p_i^{α} (the packet i in queue α) considered by the server at time t is,

$\{ p_i^{\alpha} \mid S_i^{\alpha} \leq R(t), \alpha \in B_j \}$ (2.9)

where B_j is the set of flows that are busy in the time interval (t_{j-1}, t_j)

WF^{2}Q provides almost identical service to GPS system, differing by no more than one maximum size packet.

2.4.2.4 Worst-case Weighted Fair Queueing+ (WF^{2}Q+)

Worst-case Fair Weighted Fair Queueing+ (WF^{2}Q+) [6] is an enhancement to WF^{2}Q which implements a new virtual time function that results in lower complexity and higher accuracy. WF^{2}Q+ provides the same delay bound and worst-case index as WF^{2}Q.

Let's define the following notations:

$W(t, t+\tau)$ = total amount of service provided by the server during the period $(t, t+\tau)$

$B(t)$	=	the set of flows backlogged in the WF^{2}Q+ system at the time t
$h_\alpha(t)$	=	the sequence number of the packet at the head of the queue α at the time t
$S_\alpha^{h_\alpha(t)}$	=	the virtual start time of the packet $h_\alpha(t)$ in the queue α
$F_\alpha^{h_\alpha(t)}$	=	the virtual finish time of the packet $h_\alpha(t)$ in the queue α

The new virtual time function $R(t)$ is defined as:

$$R(t + \tau) = \max(R(t) + W(t, t+\tau), \min_{i \in B(t)} (S_\alpha^{h_\alpha(t)})) \qquad (2.10)$$

To simplify the implementation, this algorithm also modifies the definition of virtual start and finish times. With the old definition in WFQ and WF^{2}Q, virtual start and finish times need to be maintained on a per packet basis. In WF^{2}Q+, there is only one pair of S_α and F_α that needs to be maintained for each queue α. Whenever a packet $h_\alpha(t)$

reaches the head of the queue α at time t, S_α and F_α are updated according to the following:

$$S_\alpha = \begin{cases} F_\alpha & Q_\alpha(t) \neq 0 \\ \max(F_\alpha, R(t)) & Q_\alpha(t) = 0 \end{cases} \tag{2.11}$$

$$F_\alpha = S_\alpha + \frac{L_i^\alpha}{\phi_\alpha} \tag{2.12}$$

Where $Q_\alpha(t)$ is the size of queue α just before time t. With this definition, S_α and F_α of queue α are also the virtual start and finish times of the packets at the head of the queue.

2.4.2.5 Self-clocked Fair Queueing (SCFQ)

WFQ and its enhancements are computationally expensive since system needs to emulate a reference GPS system and keep track for the number of active connections at any moment in GPS system. To reduce the complexity of calculating the virtual times, Self-clocked Fair Queueing (SCFQ) introduces an approximation [21]. However, the decrease in complexity results in a larger worst-case delay and the delay increases with the number of service classes.

In the SCFQ algorithm, the system's virtual time at any moment t may be estimated from the virtual time of the packet currently being serviced.

Let's define:

p	= the packet receiving service at time t
S^p	= the virtual time that packet p starts service
F^p	= the virtual time that packet p finishes service

The approximate virtual time function $R(t)$ is defined to be F^p where

$S^p < t < F^p$

Each arriving packet i in the queue α is tagged with a virtual time F_i^{α} before it is placed in the queue. The packets in the queue are picked up for service in the increasing order of their associated virtual time.

$$F_i^{\alpha} = \frac{L_i^{\alpha}}{\phi_\alpha} + \max(F_{i-1}^{\alpha}, R(\tau_i^{\alpha})), \text{ where } F_0^{\alpha} = 0 \tag{2.13}$$

2.4.3 Round Robin – based Fair Queueing (RR-based FQ)

2.4.3.1 Weighted Round Robin (WRR)

WRR is the foundation for a class of queue scheduling algorithms that are designed to address the limitations of the FQ and PQ models. WRR also addresses the limitations of the FQ model by supporting flows with significantly different bandwidth requirements. With WRR queueing, each queue can be assigned a different percentage of the output port's bandwidth.

In WRR queueing, the queues are served in a round-robin order, and as in FQ, empty queues are skipped. WRR is also referred to class-based queueing or custom queueing. WRR queueing supports the allocation of different amount of bandwidth to different service classes by either visiting each queue once during a service round and allowing higher-bandwidth queues to send more than a single packet during a visit or allowing each queue to send only a single packet during a visit, but visiting higher-bandwidth queues multiple times in a single service round.

WRR includes the following benefits:

- WRR can be applied to high-speed interfaces because of its lower complexity implementation.
- WRR queueing ensures that all service classes have access to at least some configured amount of network bandwidth to avoid bandwidth starvation.
- Classification of traffic to service classes provides more equitable management and more stability for network applications than the use of priorities or preferences.

The primary limitation of WRR queueing is that it provides the correct percentage of bandwidth to each service class, only if all of the packets in all of the queues are the same size or when the mean packet size is known in advance.

2.4.3.2 Deficit Weighted Round Robin (DWRR)

DWRR queueing [9] is designed to address the limitations of the WRR and WFQ models. DWRR addresses the limitations of the WRR model by accurately supporting the weighted fair distribution of bandwidth when it serves queues containing variable-length packets. DWRR addresses the limitations of the WFQ model by defining a scheduling algorithm that has lower computational complexity, and it can be implemented on high-speed interfaces.

A number of parameters are configured in DWRR queueing.

- A Weight defines the percentage of the output port bandwidth allocated to each queue.
- A DeficitCounter specifies the total number of bytes that the queue is permitted to transmit each time when it is visited by the scheduler. The DeficitCounter allows a queue that was not permitted to transmit in the previous round because the

packet at the head of the queue was larger than the value of the DeficitCounter to save transmission "credits" and use them during the next service round.

- A quantum of service is proportional to the weight of the queue and is expressed in terms of bytes. The DeficitCounter for a queue is incremented by the quantum each time that the queue is visited by the scheduler.

Packets arriving from different flows are stored in different queues. Defining:

byte(i,k) = number of bytes sent out from queue i in round k.

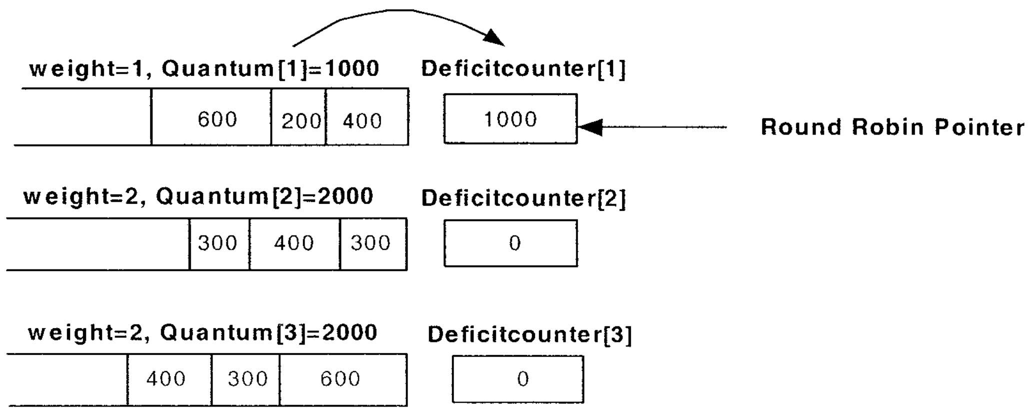

Figure 2.4: Deficit Round Robin Step 1

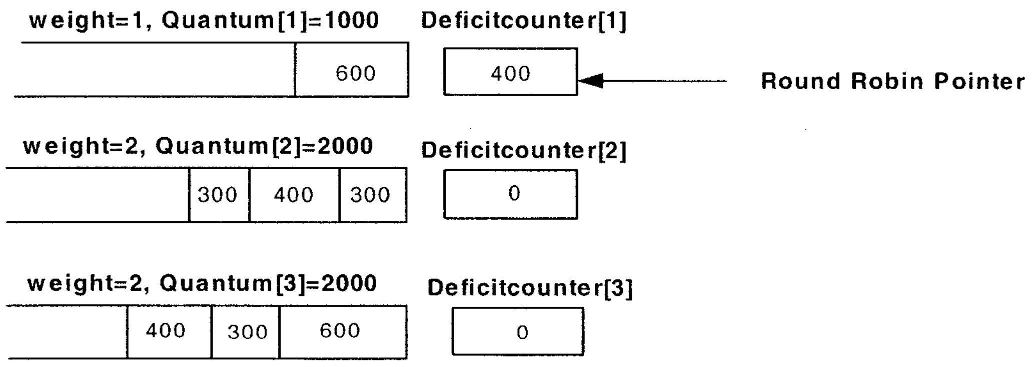

Figure 2.5: Deficit Round Robin Step 2

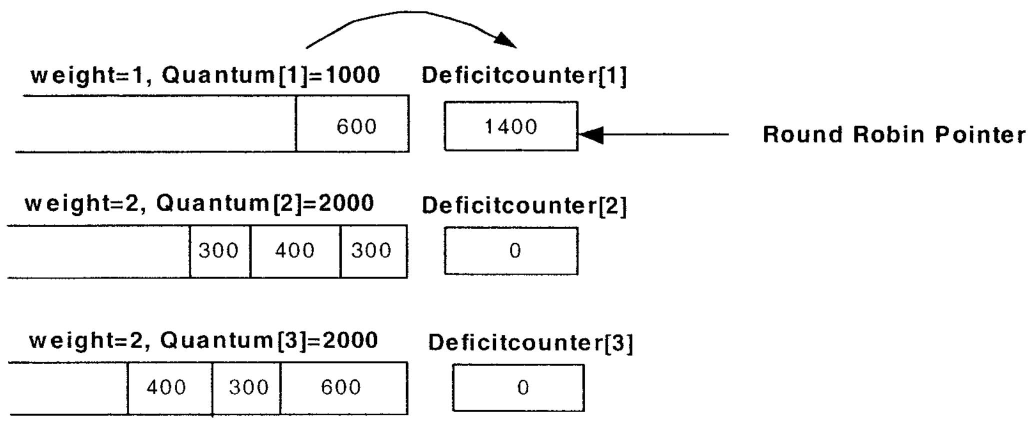

Figure 2.6: Deficit Round Robin Step 3

Each queue *i* is allowed to send out packets in the first round subject to the restriction: byte(*i*,*k*) ≤ Quantum (Figure 2.4). If there are no more packets in queue *i* after the queue has been served, DeficitCounter is reset to 0. Otherwise, the remaining amount, Quantum(*i*) - byte(*i*,*k*), is stored in the DeficitCounter (Figure 2.5). In subsequent rounds, the amount of bandwidth usable by this flow is the sum of DeficitCounter of the previous round added to the Quantum (Figure 2.6). The ActiveList, which is a list of indexes of queues that contain at least one packet, is kept to avoid examining empty queues. Whenever a packet arrives to a previously empty queue *i*, *i* is added to the end of ActiveList. Whenever index *i* is at the head of ActiveList, the algorithm serves up to Quantum(*i*) + DeficitCounter, worth of bytes from queue *i*. If at the end of this service opportunity, queue *i* still has packets to send, the index *i* is moved to the end of ActiveList; otherwise, DeficitCounter is set to 0 and index *i* is removed from ActiveList.

The benefits of DWRR queueing are that it:

- Provides protection among different flows, so that a poorly behaved service class in one queue cannot impact the performance provided to other service classes assigned to other queues on the same output port.

- Overcomes the limitations of WRR by providing precise controls over the percentage of output port bandwidth allocated to each service class when forwarding variable-length packets.
- Overcomes the limitations of strict PQ by ensuring that all service classes have access to at least some configured amount of output port bandwidth to avoid bandwidth starvation.
- Implements a relatively simple inexpensive algorithm, from a computational perspective, which does not require the maintenance of a significant amount of per-service class state.

As with other models, DWRR queueing has limitations:

- DWRR does not provide end-to-end delay guarantees as precise as other queue scheduling algorithms do.
- DWRR may not be as accurate as other scheduling algorithms. However, over high-speed links, the accuracy of bandwidth allocation is not as critical as over low-speed links.

Chapter 3

Performance Analysis of Scheduling Algorithms

In this chapter, first we describe the system and source models for the simulation. We introduce the simulation software and explain the major classes and their functions in the program. Then, we discuss gathering of statistics for determining the various network performance measures. Finally, we present performance results of the scheduling algorithms introduced in the previous chapter.

3.1 System Model Description

We consider a multiplexer with n queues each with infinite waiting room and a single server (Figure 3.1). The queue i is fed by type-i sources and each type of source consists of a number of mutually independent and identical binary Markov sources. The server according to the scheduling algorithm decides the order of message transmissions on the shared output link.

We assume that each message consists of a variable number of fixed-size packets. When a message is served, all the packets that belong to that message are served consecutively. The time axis is divided into equal duration slots and a packet is transmitted at the slot boundaries. A packet transmission time is equal to one slot. The transmission of a message cannot begin during the slot that it has arrived.

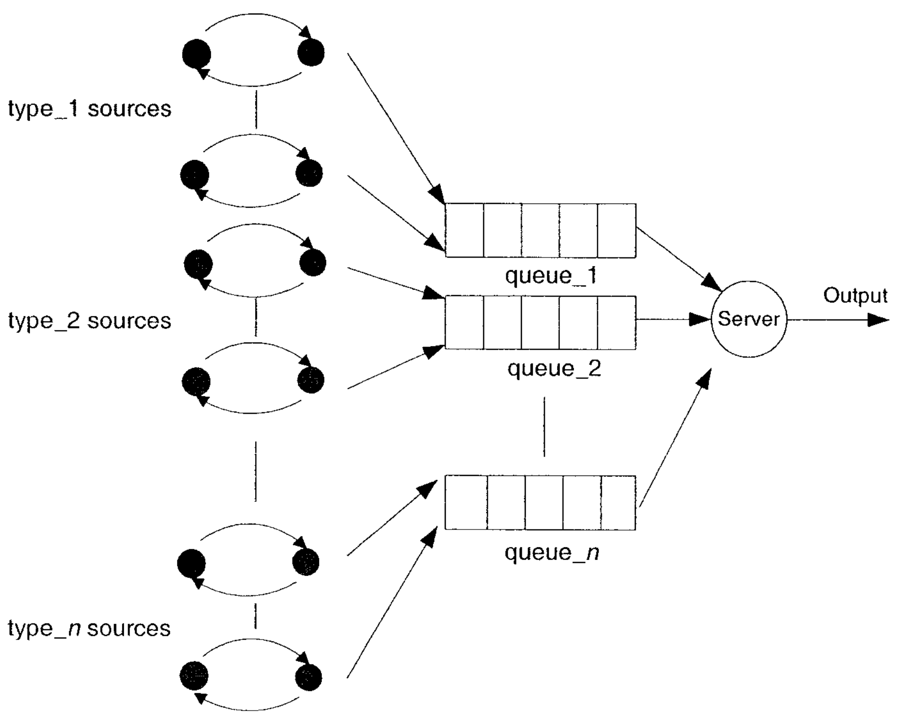

Figure 3.1: A multiplexer with multiple queues and a single server

3.2 Source Model Description

3.2.1 The Binary Markov On/Off Traffic Model

To evaluate the performance of current networks that support various communication services, an appropriate source modeling is required. There have been many traffic models proposed in the literature for characterizing individual data traffic source and superposition of multiple sources. For instance, Poisson arrival process (continuous time case), geometric inter-arrival process (discrete time case) are proposed for data traffic, Interrupted Poisson Process (IPP) for voice traffic and Markov Modulated Poisson Process (MMPP) for data, voice and video traffic. A good survey on traffic modeling can be found in [27].

Among those traffic models that have been used for different types of sources, the most versatile one is the binary Markov On/Off model. In this model, each source is characterized by On (corresponding to active bursts) and Off (corresponding to silent duration) periods, which alternate with each other. During the silent periods, no packets are generated. This model is very popular and has been often used for the modeling of traffic. For instance a binary Markov model has been successfully applied for modeling the voice source ([28][29]). In addition, in [30], a video source is modeled as a birth-death process, which consists of the superposition of a number of independent and identical On/Off mini-sources.

Because of its versatility and flexibility, the binary Markov On/Off model has been chosen as the basic model for the characterization of input traffic sources. Hence input process in this simulation consists of the superposition of many identical independent traffic streams generated by binary Markov sources.

3.2.2 Source Model Parameters

In this section, we describe the parameters of binary Markov sources. We assume that type-*i* sources consist of m_i independent and identical binary Markov sources. Each source alternates between On and Off states (Figure 3.2). During an On slot each source generates a single variable-length message, while during an Off slot no message is generated. State transitions of the sources are synchronized to occur at the slot' boundaries according to a two-state periodic and irreducible Markov chain.

Let us define the probabilities of the following events:

$\alpha_i = Pr$ (that type-*i* source will be active in the next time slot given that it is active in the present slot)

$\beta_i = Pr$ (that type-*i* source will be passive in the next time slot given that it is

idle in the present slot)

From Figure 3.2, the probability of a transition from an On state to an Off state is 1-α_i, while a transition from an Off state to an On state occurs with a probability 1-β_i, where *i* corresponds to the source type. Thus the number of slots that a source spends in On and Off states is geometrically distributed with parameters α_i and β_i respectively.

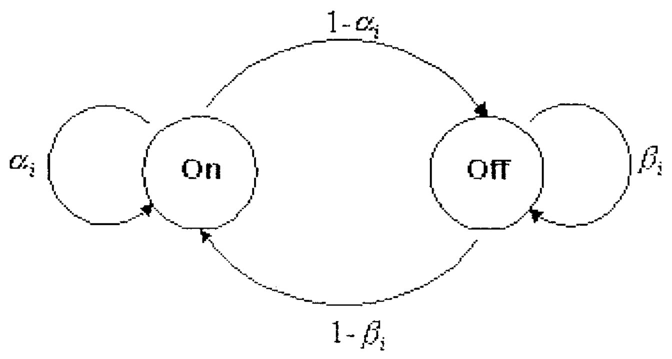

Figure 3.2: On-Off Markov Source Model

When α_i and β_i are high, the generated packets have tendency to arrive in clusters, alternatively when α_i and β_i are low, the packet arrivals are more dispersed in time. Also, the sum of a source' On and Off probabilities, $\alpha_i + \beta_i$, is an index of the correlation of the arrivals. When $\alpha_i + \beta_i = 1$, the probability that a source is On is independent from one slot to the next one and this results in independent Bernouli arrivals; the lower or higher values of the sum $\alpha_i + \beta_i$ cause an increased correlation between the arrivals in consecutive slots.

We assume that each message consists of random number of packets. Let us define:

p_j = the probability that a message consists of *j* packets

$f_i(z)$ = probability generating function (PGF) of the number of packets in a message generated by a type-*i* On source during a slot. $f_i(z)$ is determined by the following equation [31]:

$$f_i(z) = \sum_{j=0}^{\infty} p_j z^j \tag{3.1}$$

$\overline{f_i}$ = the average number of packets in a type-*i* message. $\overline{f_i}$ is determined by the following equations[31]:

$$\overline{f_i} = f_i'(z) \Big|_{z=1} = p_1 + 2p_2 + 3p_3 + 4p_4 + 5p_5 + \ldots \tag{3.2}$$

If the service rate of the system is assumed to be 1 packet/ slot, the load of the system can be determined from the following equation in [31]:

$$\rho = \sum_{i=1}^{n} m_i \, Pr(\text{a type-}i\text{ source is On}) \overline{f_i} = \sum_{i=1}^{n} m_i \frac{(1-\beta_i)\overline{f_i}}{2-\alpha_i-\beta_i} \tag{3.3}$$

and for a stable system we require that $\rho<1$.

3.3 Simulation Program

In this section, we describe the simulation program.

3.3.1 Introduction of the Simulation Program and Parameter Setting

The simulation program has been developed in C++ programming language. The output of the program is the MATLAB M-file type that can be directly run by MATLAB and plots the figures. All the performance figures in this chapter have been produced by this program. Even though the simulation program structure for the different scheduling algorithms is similar, the coding for the different algorithms is very different. Therefore the different algorithms have been simulated with different programs.

We use XML file as our input file. In XML file, we set source type, the number of sources and the weights of different classes of traffic. We can also control the run times of our program in XML file to get more smooth and accurate curves.

The number of queues in the simulation is automatically changed with the number of source types. The messages belonging to the same source type are stored in the same queue. Next we introduce major simulation parameters:

N = number of simulation runs. Simulation has been repeated *N* independent times to ensure that the results are statistically reliable. *N* is chosen on the basis of confidence interval for the mean queue delay that will be explained later on. The value of *N*=10 gives good confidence intervals.

amax = number of messages that has gone through the system during a run. If *amax* is not large enough then simulation results are not reliable, on the other hand, if *amax* is too large, the simulation takes too much time. It has been found that a value of *amax* = 8000 is adequate, larger values of *amax* don't change the results

astart = message number for the beginning of statistics collection. At the beginning of a simulation run, system is idle; therefore the system goes through a transient period before reaching steady-state. During the transient period, statistics are not collected since it is not representative of steady-state system operation. We found that a value of *astart*= 100 is adequate, therefore we start collecting statistics after *astart* messages have gone through the system.

3.3.2 Class Diagram of the Simulation Program

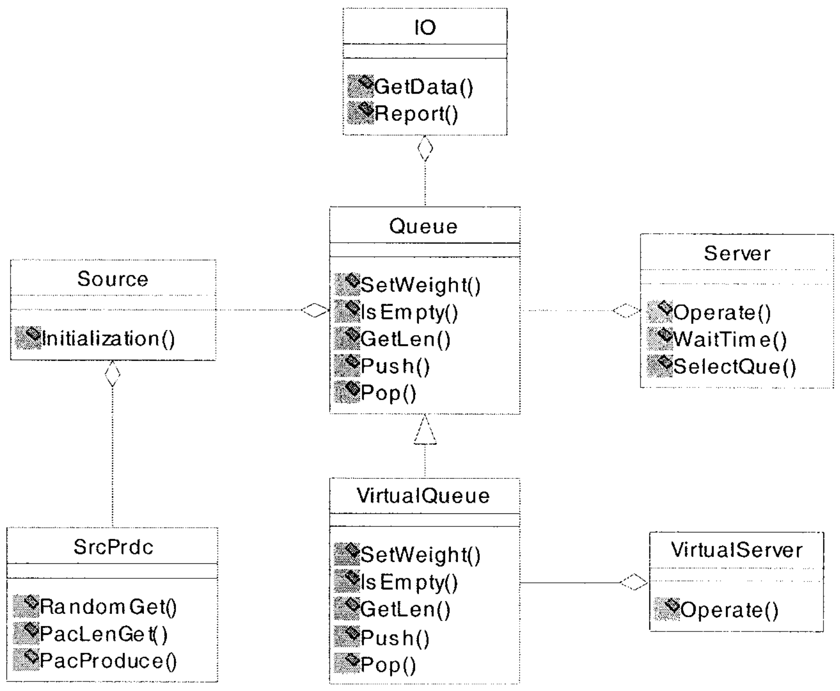

Figure 3.3: Class Diagram of the Simulation

The main classes of the simulation program are shown in Figure 3.3. Next, we briefly describe these classes:

Source class:

This class is used to create source types and sources in the system.

- Initiatialization() function is used to set α_i and β_i for type-i sources from the setting in XML input file.

SrcPrdc class:

This class controls the arrival rate of packets and the length of messages. The main functions of this class are,

- RandomGet() is designed to generate a uniformly distributed random number between 0 and 1.

- PacLenGet() determines the length of a message in number of packets by using the random number generated by the RandomGet().
- PacProduce() generates a variable-length message through the comparison of the random number with α_i or β_i.

Queue class:

This class includes the functions for the maintenance of queues, such as for the storage and removal of the messages from the queues. We use linked lists to simulate queues. The main functions of this class are,

- Push() stores a message at the end of a queue.
- Pop() removes a message from the head of a queue.
- IsEmpty() checks if a queue is empty.
- GetLen() gets the length of a queue.
- SetWeight() is used to set service weights in those applicable scheduling algorithms.

Server Class:

The main function in this class is Operate() which determines the queue to be serviced in the present slot under the current scheduling algorithm.

- SelectQue() determines the queue that the packet at the head of it has minimal virtual finish time, this function is used for those algorithms that need virtual server running on the fly as if PS is running.
- WaitTime() accumulates and calculates the waiting time of a message in the queue.

IO Class:

This class controls the input and output of the program. For the input part, functions GetTimes(), GetDataNo(), GetSrcTypeNo(), GetSrcNum(), GetSrcWgt(), GetSampleA(), GetSampleB() are used to read data from XML file. It includes run times, source type, the service weight of a queue, the parameters of a source type. The function RptFinish() generates Matlab M-file type output file.

VirtualQueue and VirtualServer classes:

They are designed to determine virtual time in the PS-based Fair Queueing algorithms. At the same time when a message enters into a queue, it is pushed into its corresponding virtual queue. The virtual server services the messages in the virtual queue according to the fluid Processor Sharing (PS) discipline. The message's virtual finish time will be used to determine the service order of messages in the real queues.

3.3.3 Simulation Flow Chart

Figure 3.4 presents the flow chart of the simulation system. Next we explain the major steps in the simulation.

At the beginning of the program, the number of source types is set. Then we create source instances according to the number of source types from the source class. After that we set the parameters of sources through the functions in the IO class. These parameters include the number of sources under the certain source type, On, Off parameters of the sources, distribution of the number of packets in messages. The network load is controlled by the setting of these parameters. The queues are created adaptively according to the number of source types. We assume that the messages generated by each source type form a different traffic flow.

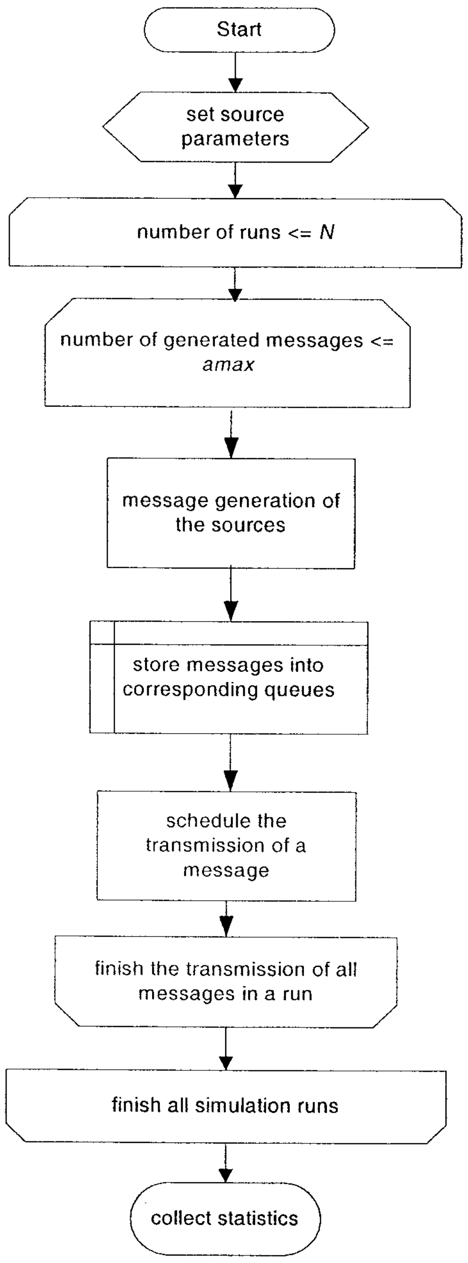
Start
set source parameters
number of runs <= N
number of generated messages <= amax
message generation of the sources
store messages into corresponding queues
schedule the transmission of a message
finish the transmission of all messages in a run
finish all simulation runs
collect statistics

Figure 3.4: Flow Chart of the Simulation

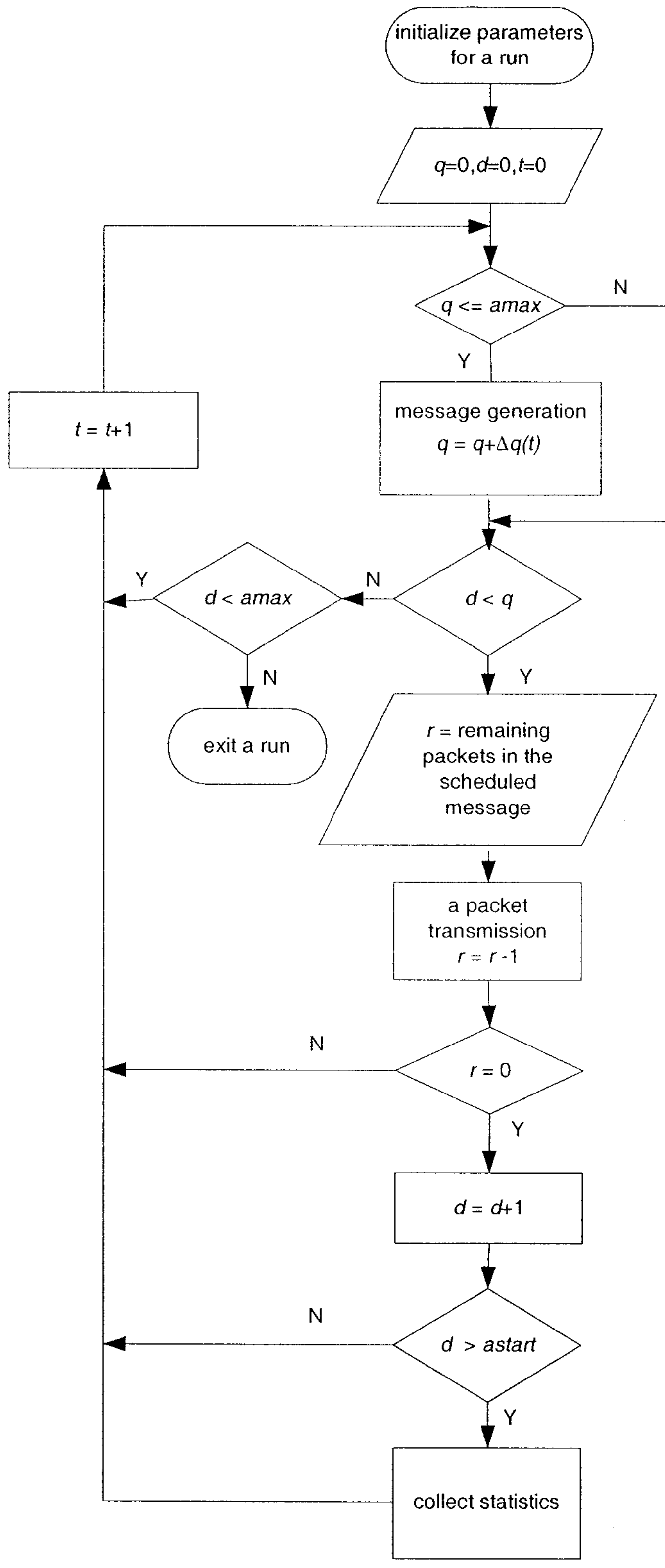

Figure 3.5: Flow Chart of a Simulation Run

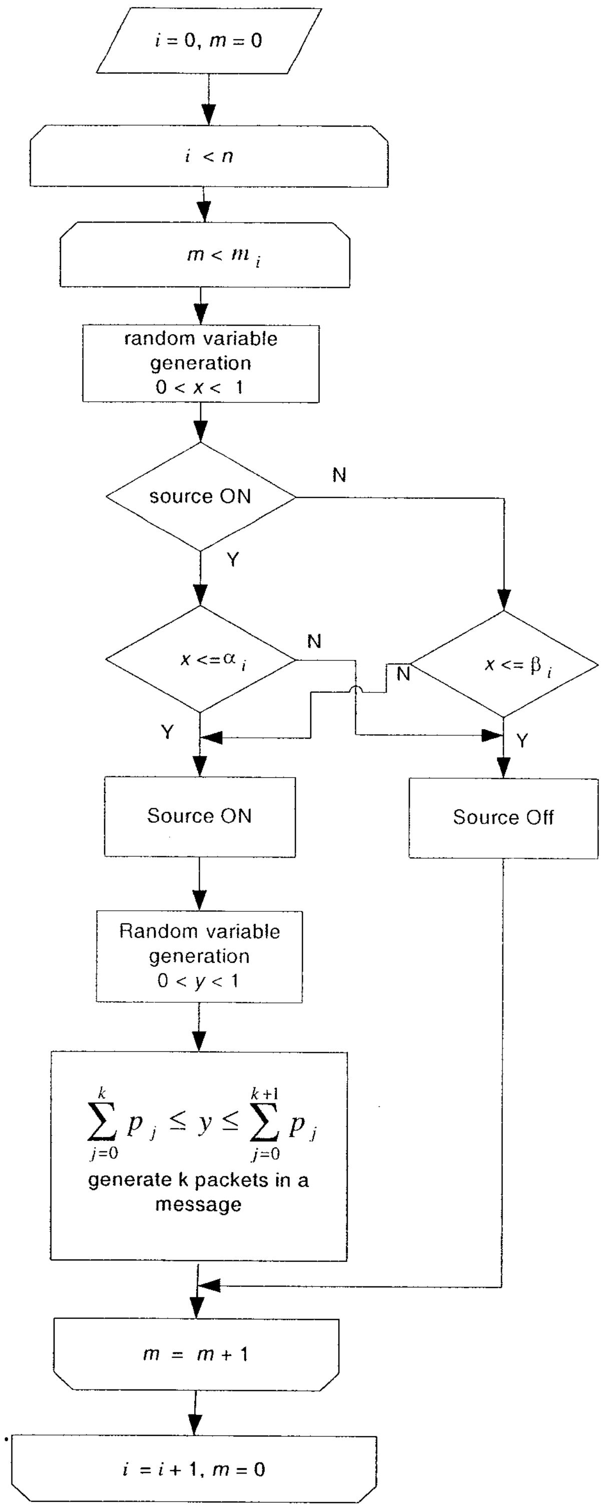
$i = 0, m = 0$
$i < n$
$m < m_i$
random variable generation $0 < x < 1$
source ON
N
Y
$x <= \alpha_i$
N
Y
$x <= \beta_i$
N
Y
Source ON
Source Off
Random variable generation $0 < y < 1$
$\sum_{j=0}^{k} p_j \le y \le \sum_{j=0}^{k+1} p_j$
generate k packets in a message
$m = m + 1$
$i = i + 1, m = 0$

Figure 3.6: Flow Chart for Message Generation in a Slot

We have made several runs to obtain each simulation point for statistical reliability. During each run, we generate *amax* number of messages. When all these messages have been transmitted, then the run is over. Next we present the steps during a simulation run (Figure 3.5). Let us define the following additional notation for this chart:

t = time in number of slots

r = remaining number of packets that needs to be transmitted in a message. At the beginning of a message transmission, it is set to the message length in number of packets

q = variable to store the number of the generated messages until time t

$\Delta q(t)$ = variable to store the number of messages generated during the slot t

d = variable to store the number of the transmitted messages until time t

From Figure 3.5, each run begins with the initialization of q, d and t to zero. During each slot, first we generate the new messages as long as the total number of messages generated is less than the maximum *(amax)* . In each slot, if the system is not empty, a packet is transmitted and the time is incremented by one. When all the packets of a message are transmitted, the message depart from the system, d is incremented by one. If the number of messages that have departed from the system is larger than *astart,* the statistics is collected. When the number of messages departed from the system equals to *amax*, a simulation run is over.

Figure 3.6 is the detailed flow chart of message generation in a slot. Let us define the following notation for this chart:

i = index of source type

m = the variable to store the current source number in *type-i* sources

m_i = the number of type-i sources

x, y = values of the uniformly distributed random variables in the interval (0,1)

In the message generation process, the status of each source is checked by the two layers loop. The outer layer loop controls the current source type (i) and the inner loop controls the source number of the current source type (m). For each source, a random number x is generated. If the source is at On status currently, it remains On if $x \leq \alpha_i$; otherwise, it changes to Off status during the next slot. If the source is Off currently, it remains Off if $x \leq \beta_i$; otherwise, it changes to On status during the next slot. If a source is On during a slot, it generates a message with a random number of packets. The number of packets in a message is determined by the random variable y.

3.3.4 Performance Measures and Statistic Collection

The main performance measures of interest are mean message delay, standard deviation of message delay, probability distribution of message delay and the confidence interval for the mean message delays. Next we explain how these performance measures have been determined.

Let us define:

$t_i(j)$ = arrival time of the jth message during simulation run i

$\tilde{t}_i(j)$ = departure time of the jth message during simulation run i

$d_i(j)$ = message delay in number of slots of the jth message during simulation run i.

d_i = random variable that denotes the message delay during run i

$\overline{d_i}$ = mean message delay during simulation run i.

d = random variable that denotes the message delay during the simulation

$\overline{d}$ = mean message delay in number of slots during the simulation.

s = standard deviation of the message delay

c_i = number of messages during run i that experience delay less than or equal

to x

c = total number of messages under consideration during run i

From the above, we have

$$d_i(j) = \tilde{t}_i(j) - t_i(j) \tag{3.4}$$

$$\overline{d_i} = \frac{\sum_{j=1}^{a\max - astart} d_i(j)}{a\max - astart} \tag{3.5}$$

then,

$$\overline{d} = \frac{\sum_{i=1}^{N} \overline{d_i}}{N} \tag{3.6}$$

s is determined by,

$$s = \sqrt{\left(N\sum_{i=1}^{N} \overline{d_i}^2 - \left(\sum_{i=1}^{N} \overline{d_i}\right)^2\right) \Big/ N(N-1)} \tag{3.7}$$

The probability distribution of the message delay is calculated by the following equation:

$$\text{Prob } (d_i \le x) = \frac{c_i}{c} \tag{3.8}$$

$$\text{Prob } (d \le x) = \frac{\sum_{i=1}^{N} \Pr ob(d_i \le x)}{N} \tag{3.9}$$

Next, we give the (1-α)100% confidence interval μ for mean message delay, $\bar{d}$,

$$\bar{d} - t_{\alpha/2} \frac{s}{\sqrt{N}} < \mu < \bar{d} + t_{\alpha/2} \frac{s}{\sqrt{N}} \tag{3.10}$$

where $t_{\alpha/2}$ is the *t*-value with v=N-1 degrees of freedom [32]. The above result states that we are (1-α)100% confident that the true mean delay lies in the given interval. The tighter is this interval, the more reliable is the simulation results. Clearly, as N increases, the results become more reliable.

3.4 Simulation Results

The objective of this study is to determine the effect of different queueing algorithms on the behavior of ATM and IP multiplexing systems. Next we present the simulation results.

In the simulation, three types of sources have been assumed, each type of sources feeding a different queue. It is assumed that each source type has 5 sources and the traffic load generated by each type is assumed to be 1/3 of the total traffic load. Thus we have n = 3, m_i=5 (i =1,2,3), $\rho_1 = \rho_2 = \rho_3 = \rho/3$ where ρ is the total load. To fully observe the performance of different scheduling algorithms, we set the average message sizes of three classes of traffic to different values, that is $\bar{f_1}$= 1, $\bar{f_2}$ = 2, $\bar{f_3}$ = 3 respectively. Since the loads of different source types are equal, the number of message generated per class decreases with the class number. We set the total number of generated messages (*amax*) to 8000 to control the running time of simulation. The first 100 generated messages (*astart*) are not recorded in the simulation result, as has been explained on earlier. Each simulation was repeated 10 times independently, then the results were averaged to have statistically dependable results. Next we present the simulation results for different performance measures.

3.4.1 Mean Message Delay

In this part, we give the simulation results for overall mean message delay. Then, results have been determined without distinguishing among messages of different classes.

Figures 3.7 – 3.9 present the overall mean message delay in number of time slots versus the system traffic load ρ under different scheduling algorithms. It may be seen that the mean message delay always increases with the traffic load ρ and it approaches to infinity as ρ approaches to 1.

Figure 3.7 presents the mean message delay under FIFO, PQ and FQ service disciplines. FIFO serves the messages in the order of their arrival. FQ serves them in a round robin manner. In the case of PQ, the priority decreases with the increasing queue number, thus class 1 messages have the highest priority. As may be seen, the results are very close under light load, but as the load increases, PQ, FIFO and FQ experience increasing delays in the given order. In general, the mean message delay is independent of the service discipline, therefore, this is an interesting result. In the system under study, the mean message delay depends on the service discipline because the service order depends on the message size. As indicated before, class 1 traffic has the shortest message size but it generates the highest number of messages. Since PQ gives the highest priority to class 1, it results in the least delay. In the FQ service discipline, the messages are transmitted in round robin manner, class1 traffic experiences the highest delay since class 1 forms the largest group. Thus, PQ and FQ result in least and highest overall delay respectively.

Figure 3.8 presents the mean message delay in the PS-based algorithms. In all these algorithms, we assign equal weight to provide the same amount of service for three classes of traffic. We observe that the mean message delay curves are very similar for all algorithms.

Figure 3.9 presents the mean message delay in the RR-based algorithms again with equal weights assigned to three flows. The figure shows that DWRR has smaller mean message delay than WRR. When WRR transmits messages from different queues, it doesn't take into consideration the message size. When the mean message size is different in different classes of traffic, the received service in different classes of traffic is different even though we have assigned the same weight to different classes of traffic. However, in DWRR, algorithm considers message size when it transmits a message. So when we assign the same weight to different classes of traffic, these classes of traffic receive the same amount of service.

In Table 3.1 we present the mean message delay confidence intervals for the simulation results presented in Fig.3.7, 3.8 and 3.9 for $\rho=0.6$. In general, we would like to have confidence intervals as short as possible. It may be seen that these intervals are quite short thus giving confidence to the simulation results. As expected, the corresponding interval gets larger with the increasing confidence level.

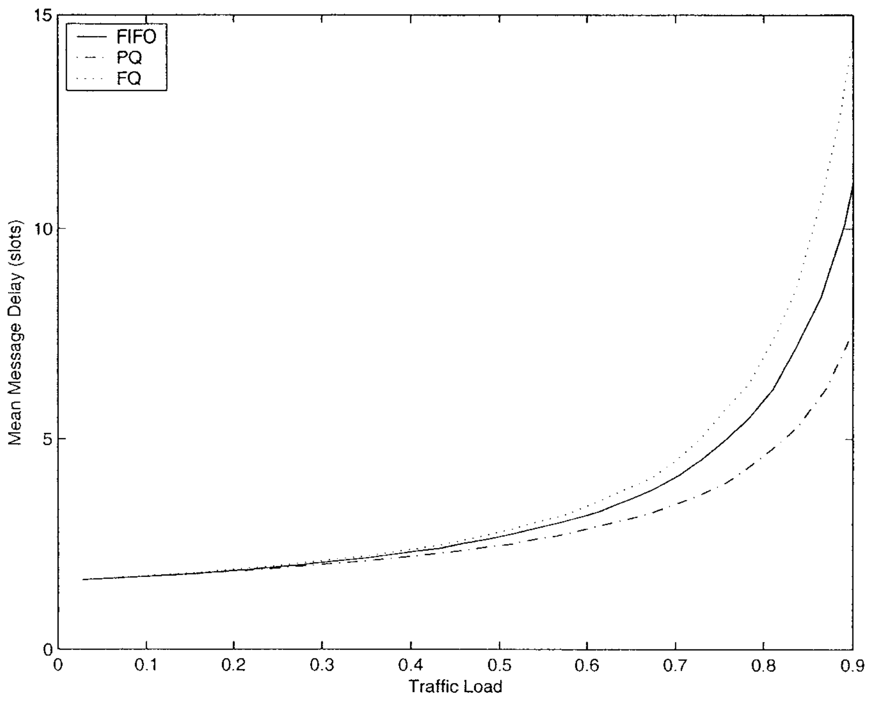

Figure 3.7: Mean message delay comparison of FIFO, PQ and FQ algorithms

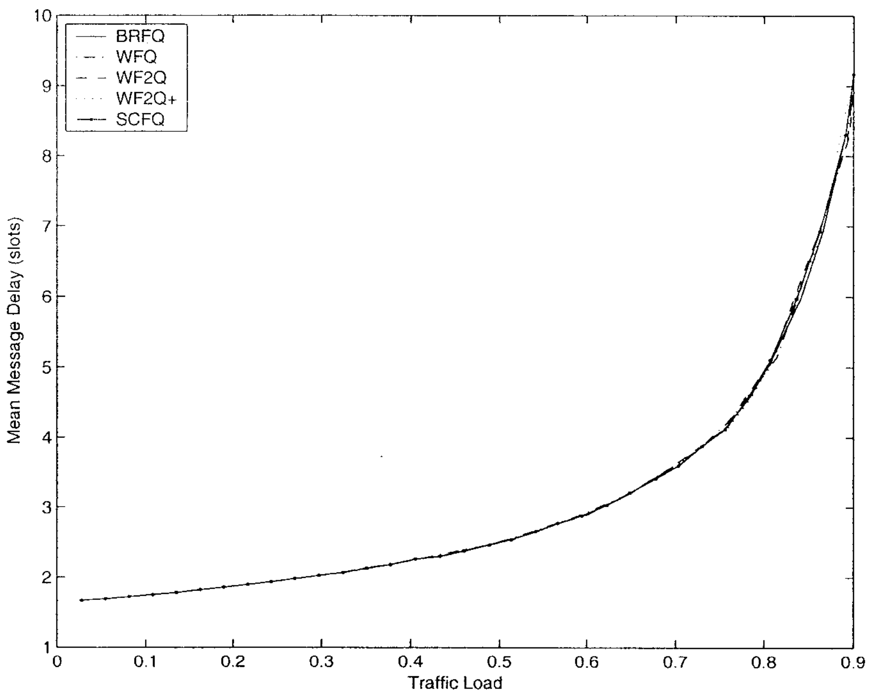

Figure 3.8: Mean message delay comparison of the PS-based queueing algorithms

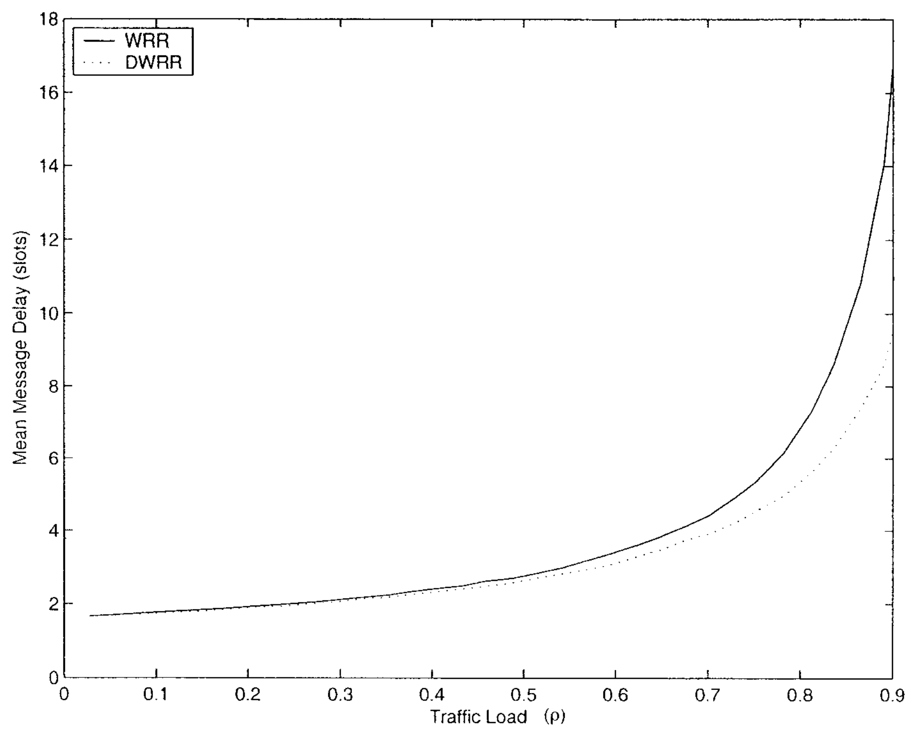

Figure 3.9: Mean message delay comparison of RR-based queueing algorithms

Scheduling Algorithm	Mean Delay (slots)	Confidence Interval		
		90%	95%	99%
FIFO	3.181141	0.0619	0.0762	0.1083
PQ	2.842647	0.0383	0.0471	0.1083
FQ	3.368027	0.0517	0.0635	0.0904
BRFQ	2.878911	0.0464	0.0570	0.0811
WFQ	2.925907	0.0490	0.0603	0.0857
WF^2Q	2.901344	0.0483	0.0594	0.0845
WF^2Q+	2.864524	0.0483	0.0594	0.0844
SCFQ	2.952513	0.0511	0.0628	0.0893
WRR	3.371022	0.0666	0.0819	0.1165
DWRR	2.903531	0.0486	0.0597	0.0849

Table 3. 1 : Confidence intervals for the simulation results presented in Fig. 3.7, 3.8 and 3.9 for ρ=0.6

3.4.2 Mean Message Delay of Different Classes

Figures 3.10– 3.12 present the mean message delay in each class of traffic versus the system traffic load ρ under the FIFO, PQ and FQ algorithms. As we noted earlier, classes 1, 2, 3 have average message sizes from 1 to 3 respectively and the three classes have the same traffic load. Delay of a message consists of its queueing delay plus its transmission time. In FIFO service discipline (Figure 3.10), the message delay is determined by the message transmission time under light load. Therefore, flows with larger message size experience higher delay. Under heavy load, the delay curves approach each other because queueing delay dominates. In PQ (Figure 3.11), the priority is set according to the message size, that is traffic with smaller average message size has higher priority. The class 1 has the highest priority, then classes 2 and 3 respectively. In this case, as expected, we observed that the traffic with highest priority has the lowest delay. Under light load, FQ behaves similar to FIFQ and PQ. However, as the load increases we have the interesting result that the mean message delay of class 1 increases (Figure 3.12). This is because the queue with larger message size receives more service since FQ serves one message per queue per cycle. Therefore the queue with shortest message size is penalized compared to those with longer message size.

Figures 3.13- 3.15 present the mean message delay in each class of traffic against the system traffic load ρ under the PS-based algorithms. The service weights for the three classes of traffic are assigned to be same, thus the service rate of each class is 1/3 since the output link data rate is 1 packet /slot. In BRFQ (Figure 3.13), as the traffic load increases, the delay curves rise with a similar slope. The difference between the curves under light load is due to difference between the transmission times of three classes of messages. Though three classes are given the same amount of service, class 1 messages experience the shortest delay because they have the shortest size. That is expected because BRFQ is trying to provide the same amount of service to each queue. Since the three classes have the same load, the increases in their mean message delay should be similar. In Figure 3.14, we see that WFQ, WF^2Q, WF^2Q+ have similar curves. We note that WFQ in this figure corresponds to BRFQ since all classes have equal service weights. The WFQ-based algorithms can be used to provide different amount of service to different classes through assigning different weights to different queues. Compared with the WFQ-based queueing. BRFQ can only provide same amount of service to different traffic classes. The curves in SCFQ (Figure 3.15) have some differences with those in Figure 3.14, which is due to the approximation in simulating GPS in SCFQ. SCFQ algorithm increases the inaccuracy in simulating GPS while decreasing the computational complexity.

Figures 3.16-3.17 present the mean message delay of each class of traffic versus the system traffic load under the WRR and DWRR algorithms. We can see that the curves of WRR are similar to those of FQ (Figure 3.12) in this case because the same weights have been assigned to different queues. The curves of DWRR are similar to those of PS-based algorithms.

We also present figures about the mean message delay of each class of traffic versus the system traffic load when different service weights are assigned to each class. In this case, the service weights assigned to each class are 1,2,3 respectively. As a result, the service rates received by classes 1,2,3 under WFQ are 1/6, 2/6, and 3/6 respectively (Figure 3.18). The service rate of class 1 decreases, class 2 remains same and class 3 increases with respect to BRFQ. As may be seen in all the figures, the message delay of class 3 decreases at the expense of class 1 traffic.

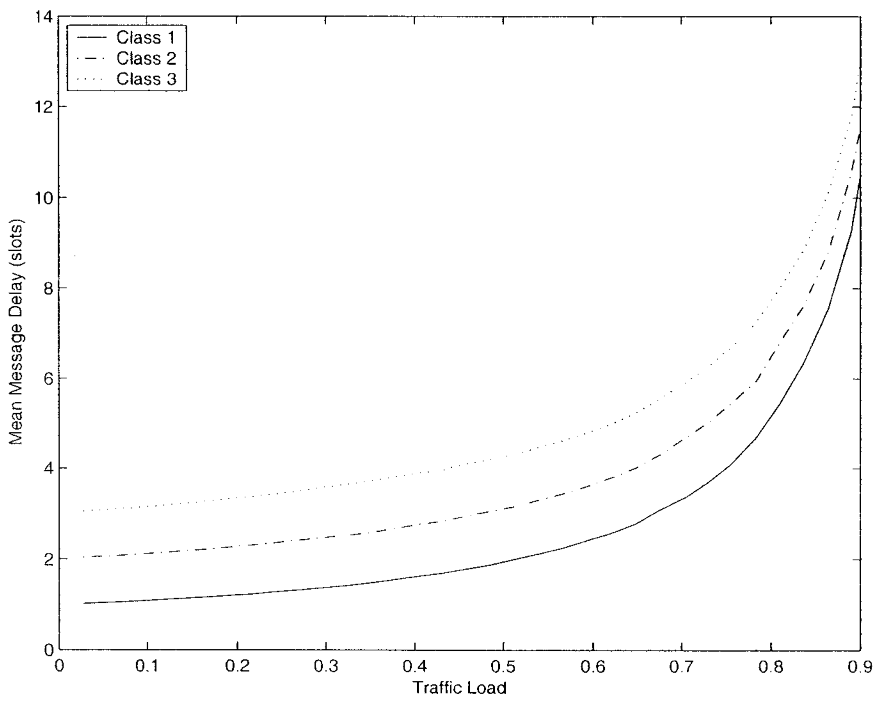

Figure 3.10: Mean message delay for each class of traffic under FIFO

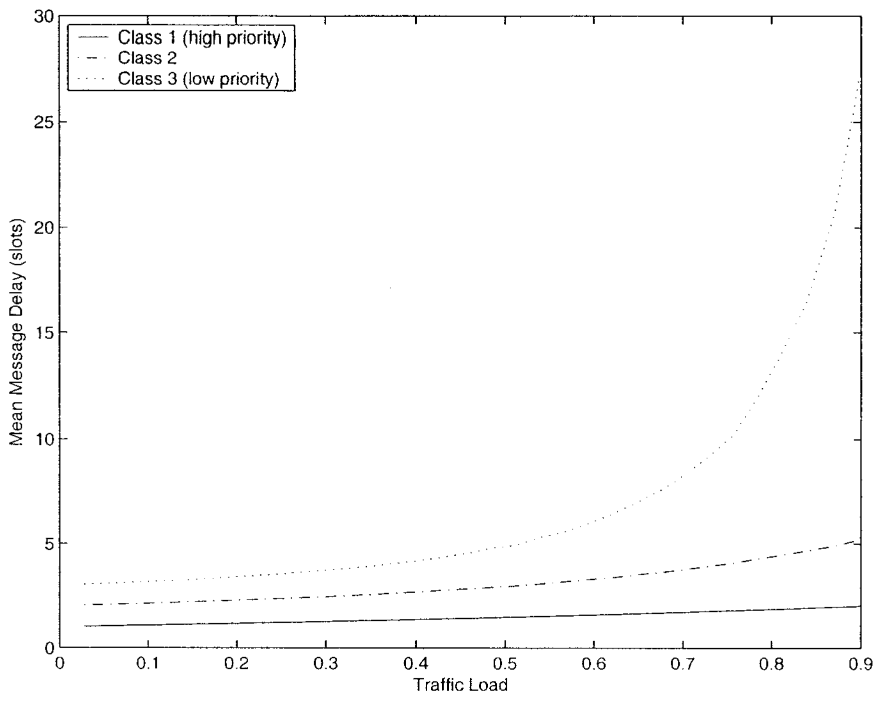

Figure 3.11: Mean message delay for each class of traffic under PQ

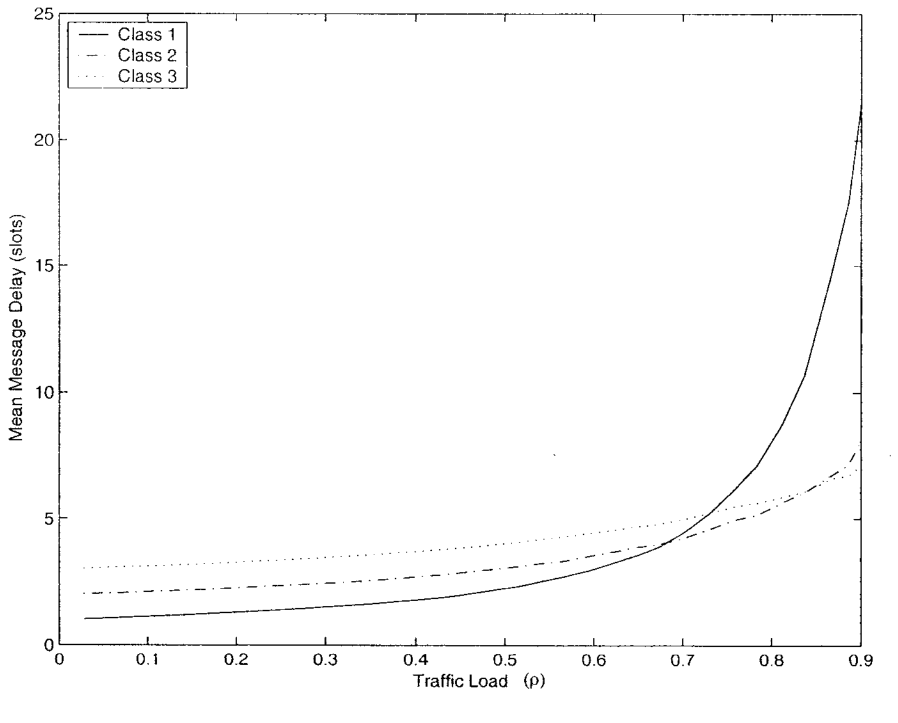

Figure 3.12: Mean message delay for each class of traffic under FQ

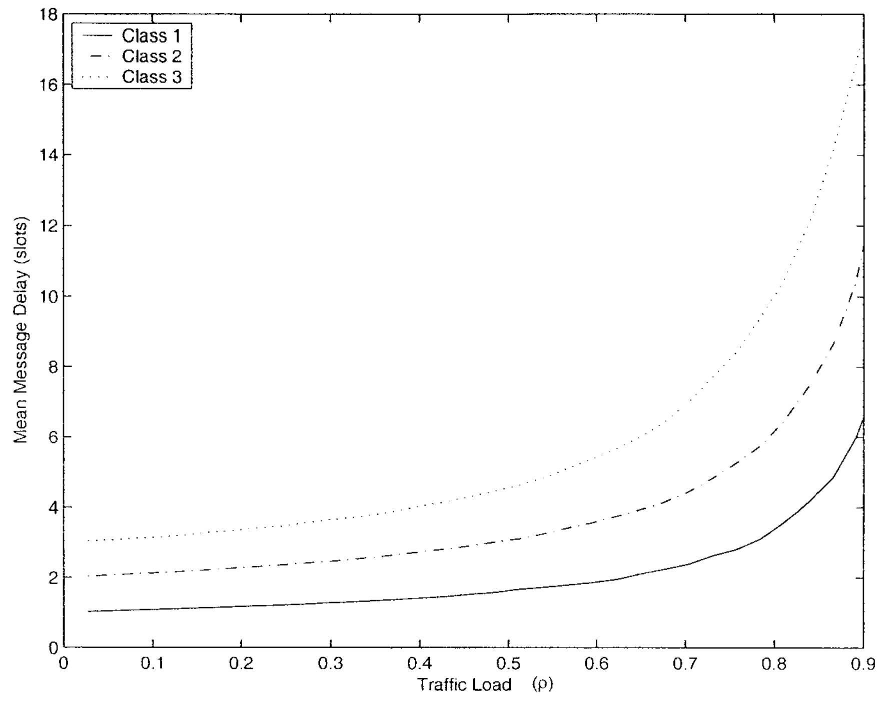

Figure 3.13: Mean message delay for each class of traffic under BRFQ

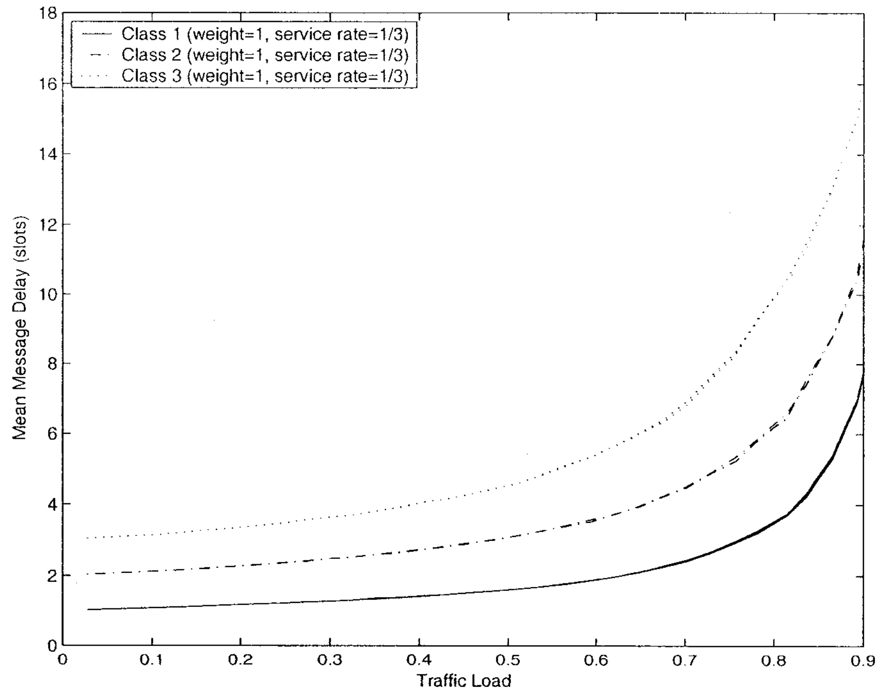

Figure 3.14: Mean message delay for each class of traffic under WFQ, WF^2Q, WF^2Q+

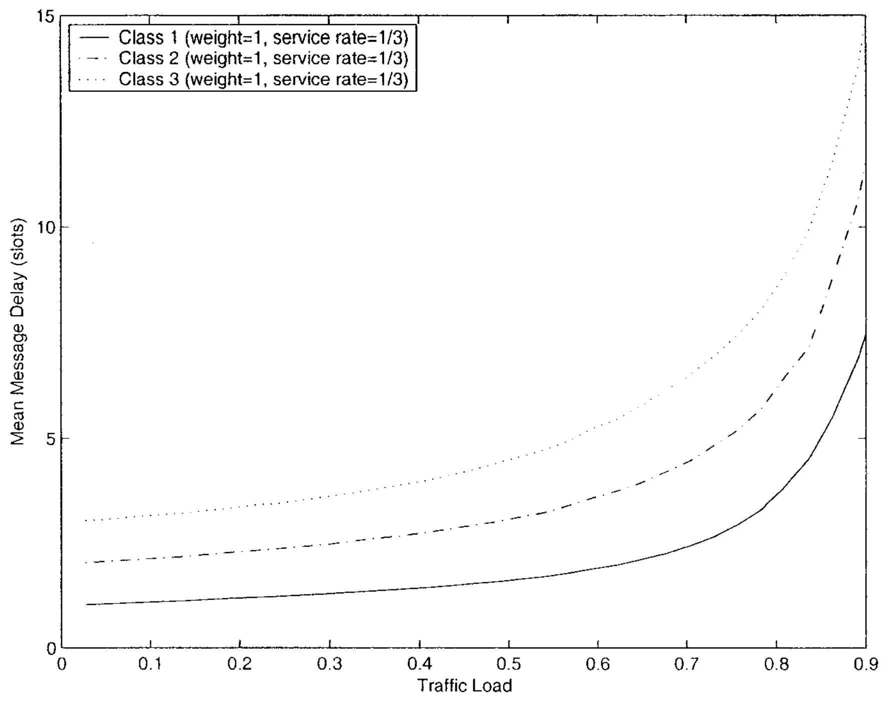

Figure 3.15: Mean message delay for each class of traffic under SCFQ

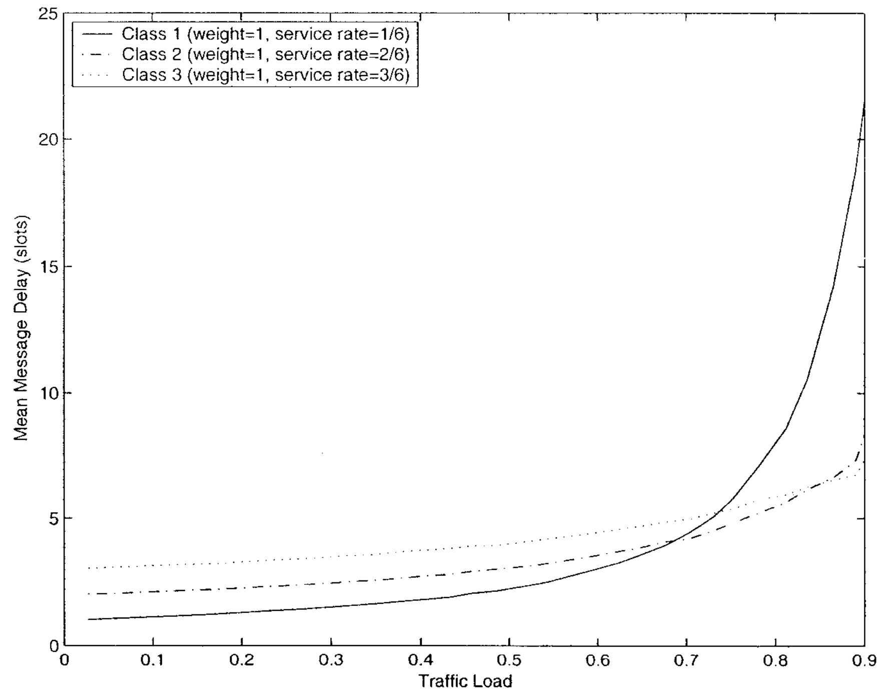

Figure 3.16: Mean message delay for each class of traffic under WRR

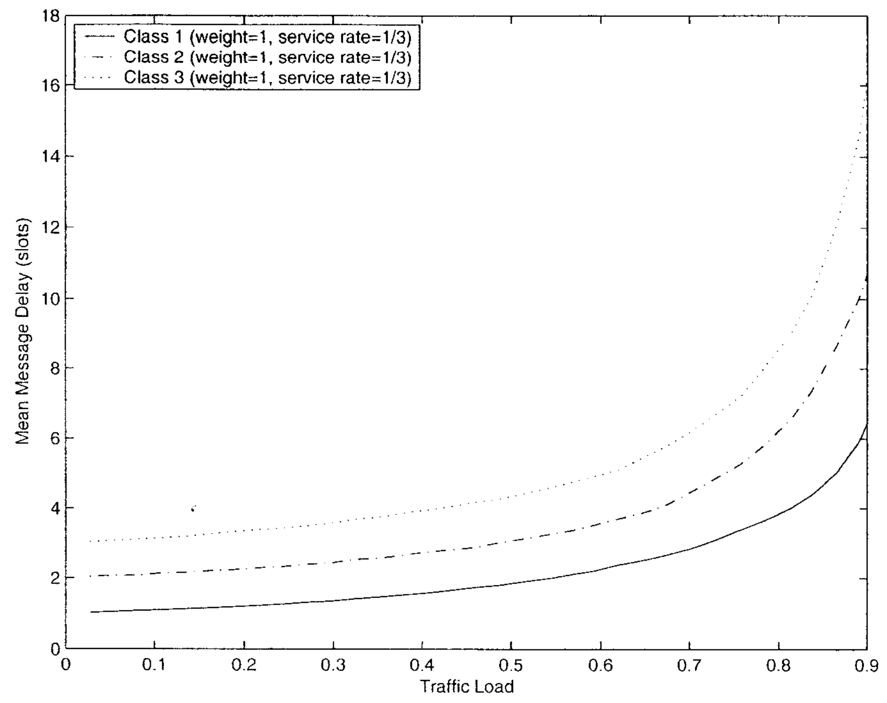

Figure 3.17: Mean message delay for each class of traffic under DWRR

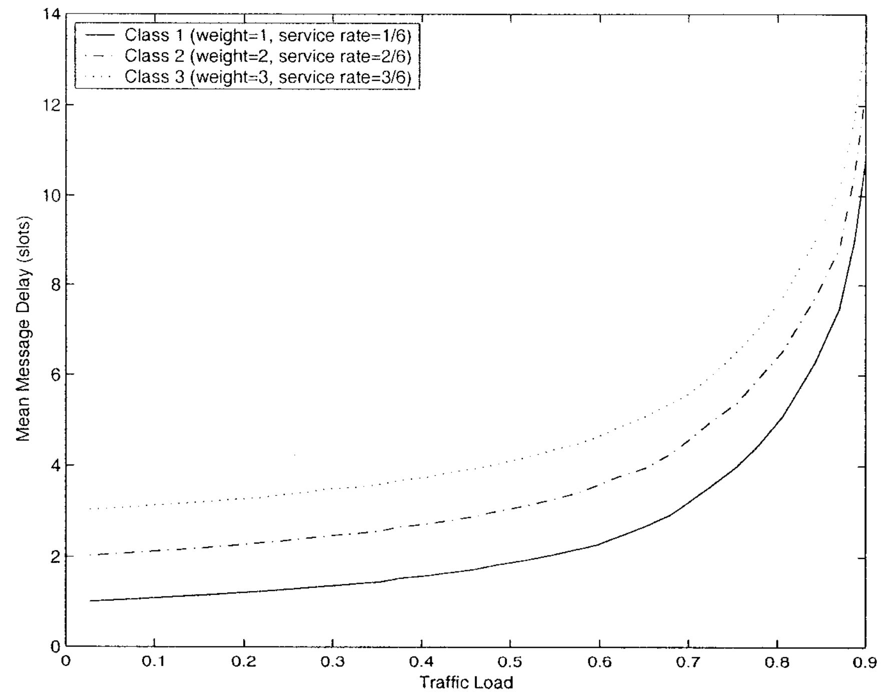

Figure 3.18: Mean message delay for each class of traffic under WFQ

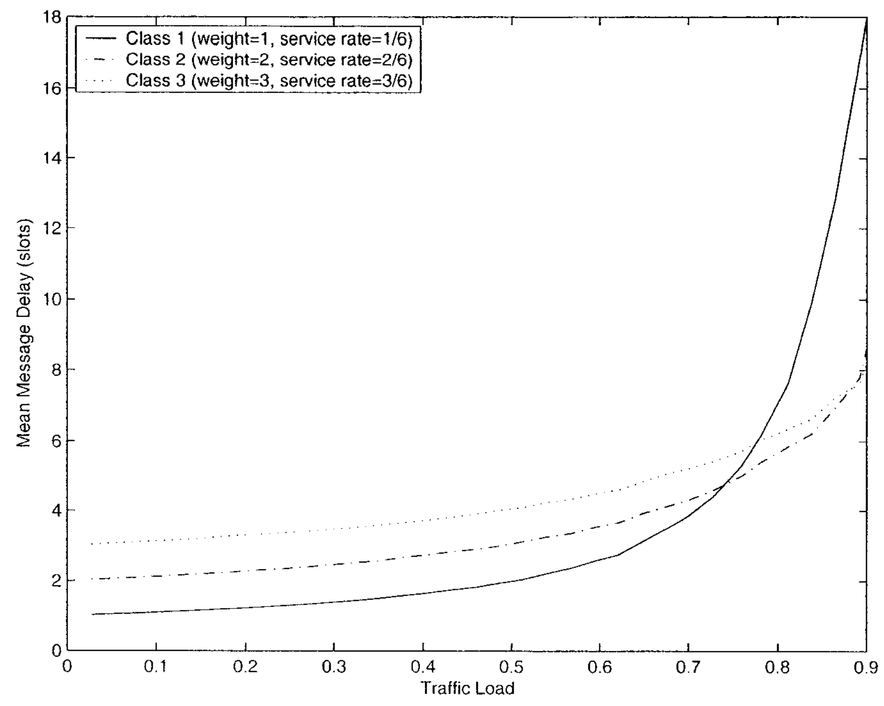

Figure 3.19: Mean message delay for each class of traffic under SCFQ

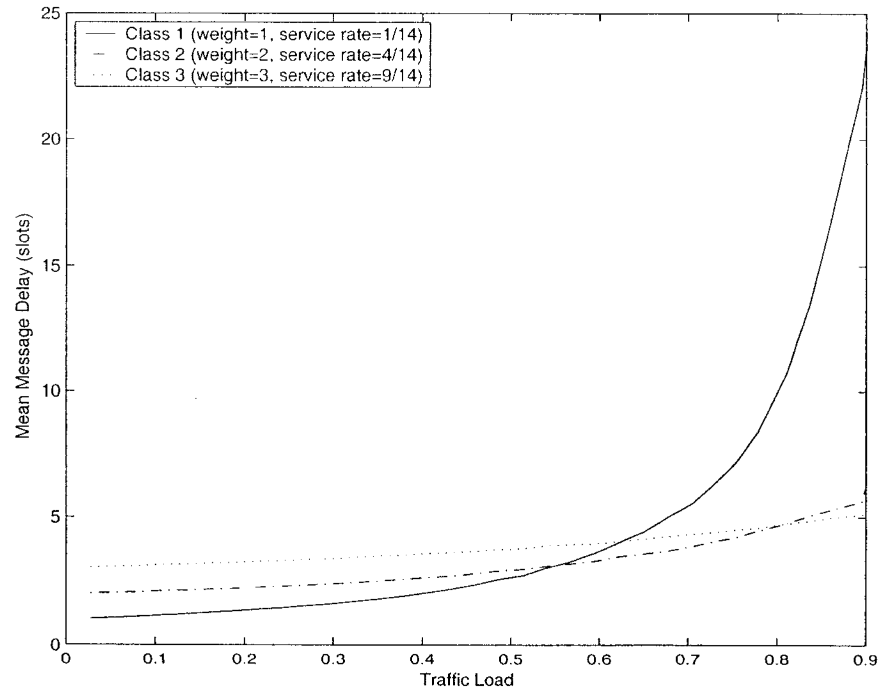

Figure 3.20: Mean message delay for each class of traffic under WRR

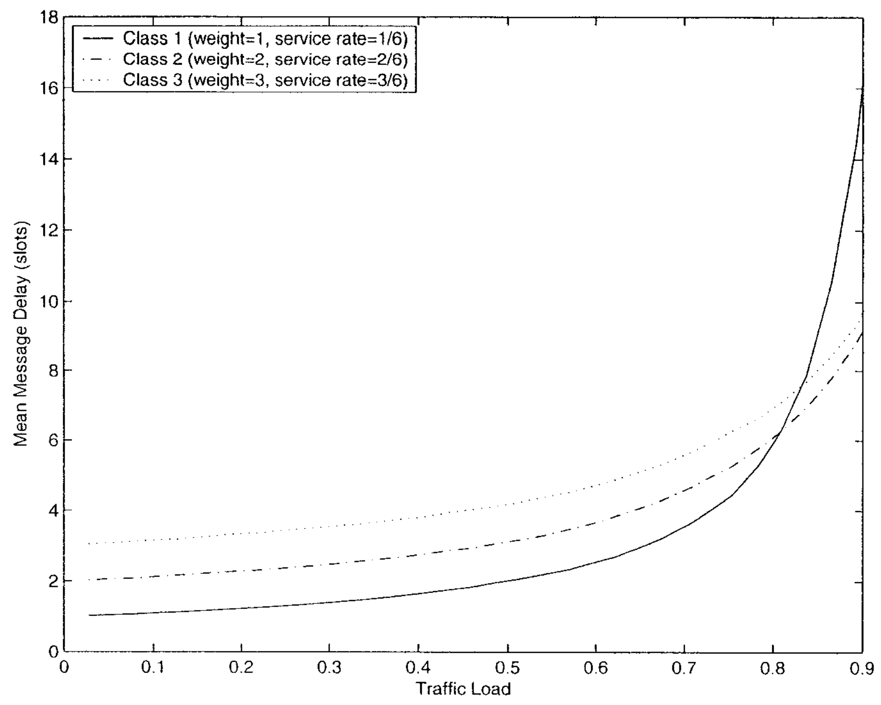

Figure 3.21: Mean message delay for each class of traffic under DWRR

3.4.3 Standard Deviation

Figures 3.22 - 3.24 present the standard deviation of the message delay versus the system traffic load ρ under different scheduling algorithms. The figures show that the mean message delay variation increases as the traffic load increases. The trend in the rise of standard deviation is very similar to that of mean delay curves.

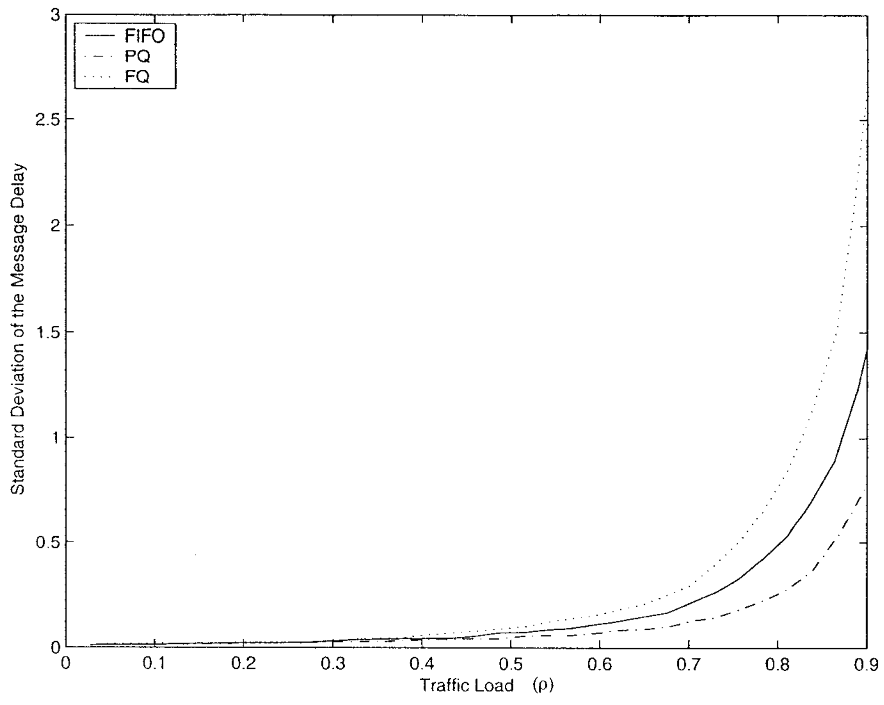

Figure 3.22: Standard deviation comparison of FIFO, PQ and FQ algorithms

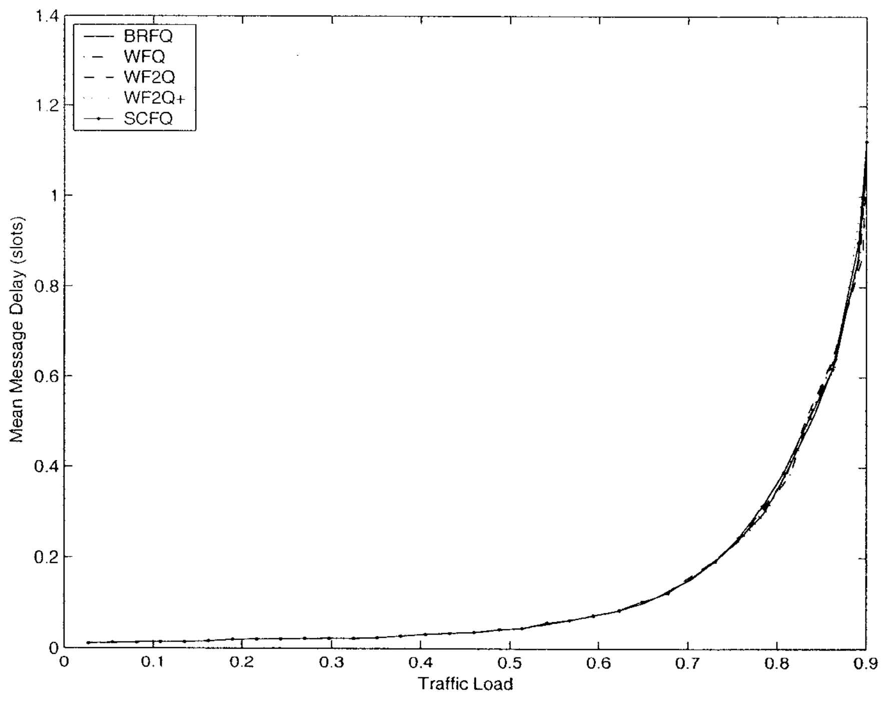

Figure 3.23: Standard deviation comparison of PS-based queueing algorithms

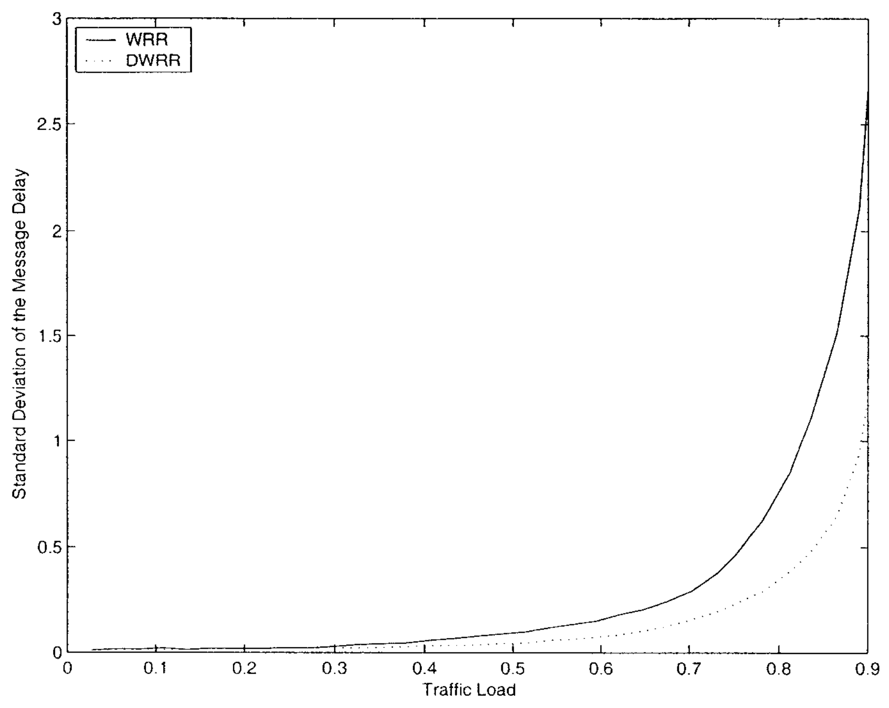

Figure 3.24: Standard deviation comparison of RR-based queueing algorithms

3.4.4 Probability Distribution of Message Delay

In Figures 3.25 - 3.27, we present overall probability distribution of message delay with the traffic load as a parameter under different scheduling algorithms. From left to right, the curves are for ρ=0.2 and ρ=0.8 respectively.

In Figure 3.25, we present the probability distribution of message delay under the FIFO, PQ and FQ algorithms. Under the light traffic load, ρ=0.2, the three probability distribution curves are very close. Under the heavy traffic load, ρ=0.8, the probability distributions diverge from each other. For example, it may be observed that the probability that message delay is less than or equal to 8 slots is lower under PQ than under FIFO and FQ which are close to each other. That means there are more messages that experience shorter delay under PQ than under FIFO and FQ. As may be seen, the curve of FQ approaches to unity slower than that of PQ and FIFO, which means that more messages experience larger delay under FQ. These observations are in agreement with those of mean delay in reference to Figure 3.7.

Figure 3.26 presents the probability distribution of message delay under the PS-based scheduling algorithms. Under light traffic load, the five probability distribution curves are almost same. Again under heavy traffic load, there are differences among the curves. The probability that a message experiences shorter delay is highest under BRFQ and lowest under SCFQ.

Figure 3.27 presents the probability distribution of message delay under the RR-based scheduling algorithms. Again the curves are very close under light load but they are different under heavy load. DWRR results lower delay than WRR under heavy load.

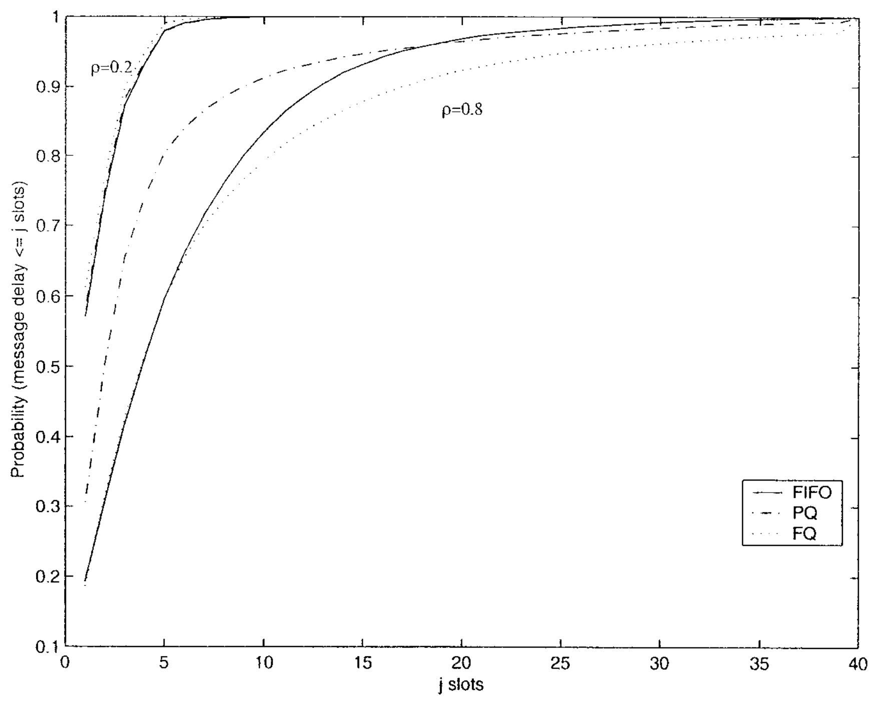

Figure 3.25: Probability distribution comparison of the message delay of FIFO,PQ, FQ algorithms

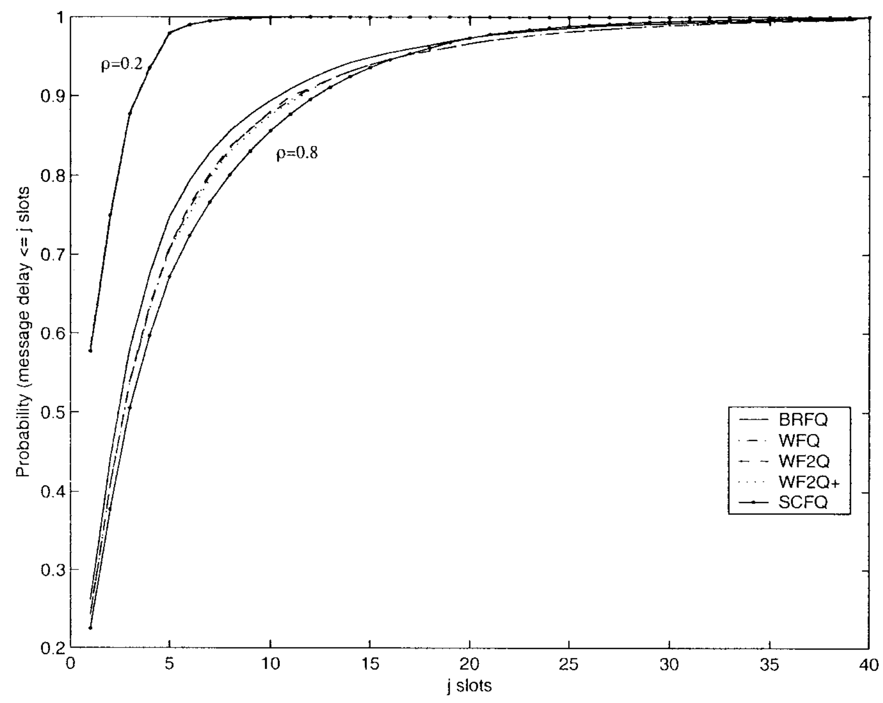

Figure 3.26: Probability distribution comparison of the message delay of PS-based queueing algorithms

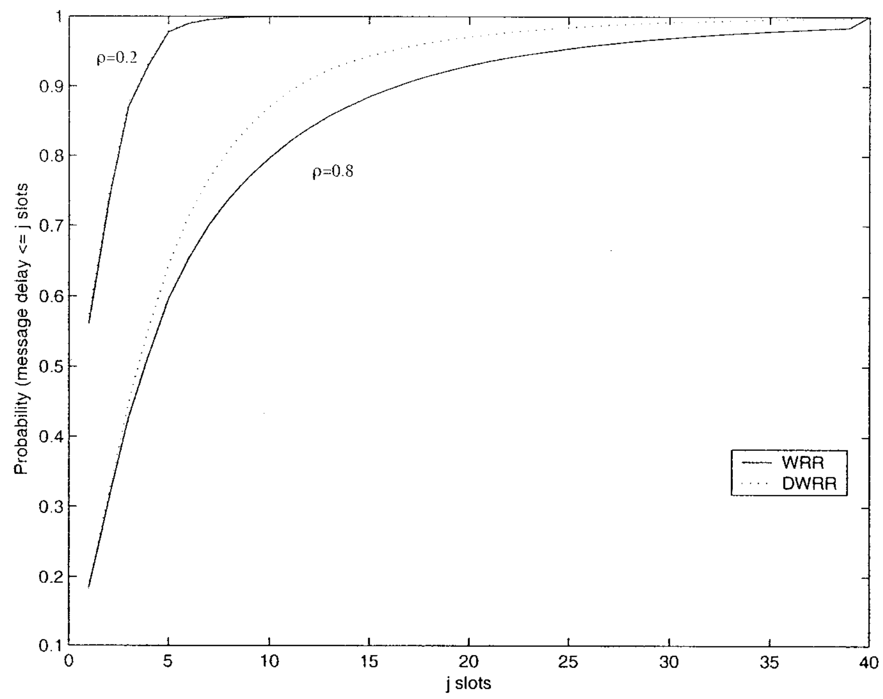

Figure 3.27: Probability distribution comparison of the message delay of RR-based queueing algorithms

3.4.5 The Probability Distribution of Message Delay of Different Classes

Figures 3.28 – 3.30 present the probability distribution of message delay for each class under the FIFO, PQ and FQ scheduling algorithms. The high priority traffic benefits absolutely under the PQ algorithm. The curves for the FIFO algorithm are very close for larger delays. As may be seen, class 1 messages experience the largest delay under FQ.

Figures 3.31-3.33 present the probability distribution of message delay for each class under the PS-based algorithms. We observe that the curves of probability distribution of WFQ, WF^2Q, WF^2Q+ are very similar. We note that since we assumed equal weights, BRFQ and WFQ are identical. From Figure 3.33, SCFQ results in higher delay because it is not as accurate as the other algorithms in tracking GPS.

Figures 3.34-3.35 present the probability distribution of message delay for each class under the RR-based algorithms. The curves of WRR and DWRR have obvious differences. The behavior of WRR is similar to that of FQ, while that of DWRR is close to the curves of PS-based algorithms.

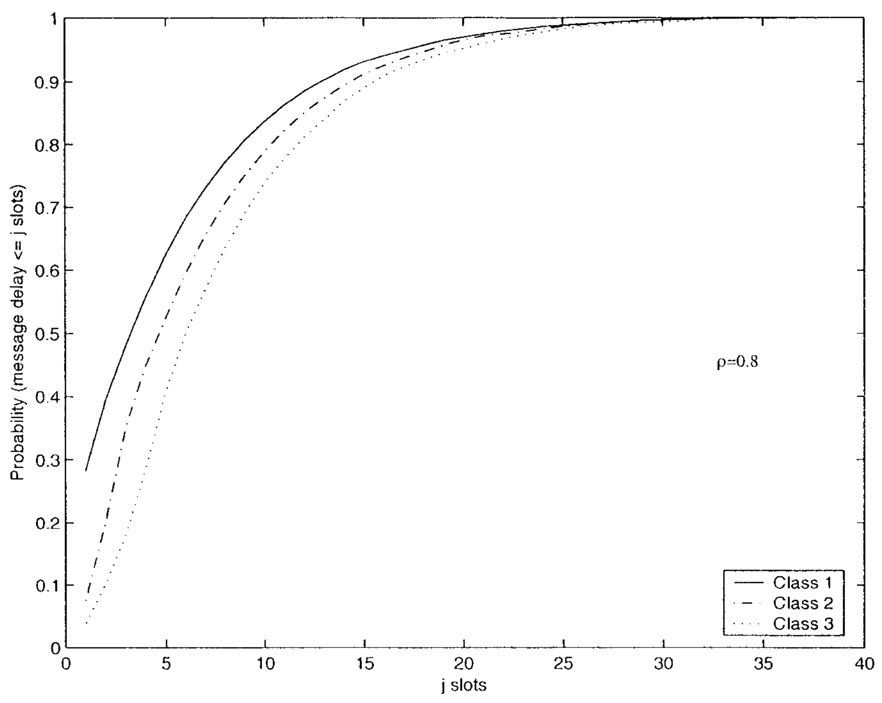

Figure 3.28: Probability distribution of the message delay for each class of traffic under FIFO

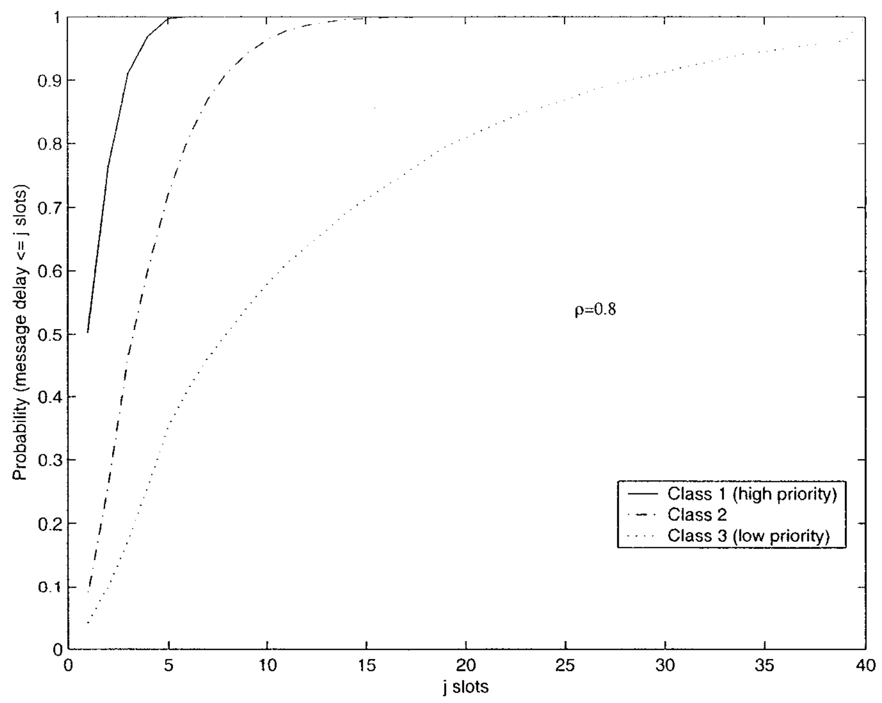

Figure 3.29: Probability distribution of the message delay for each class of traffic under PQ

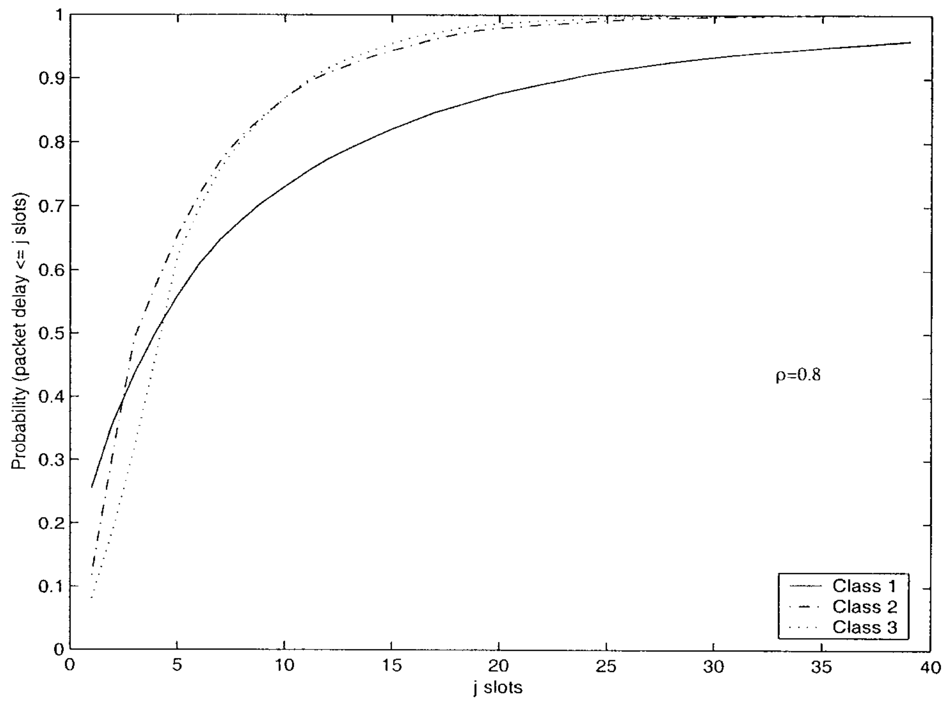

Figure 3.30: Probability distribution of the message delay for each class of traffic under FQ

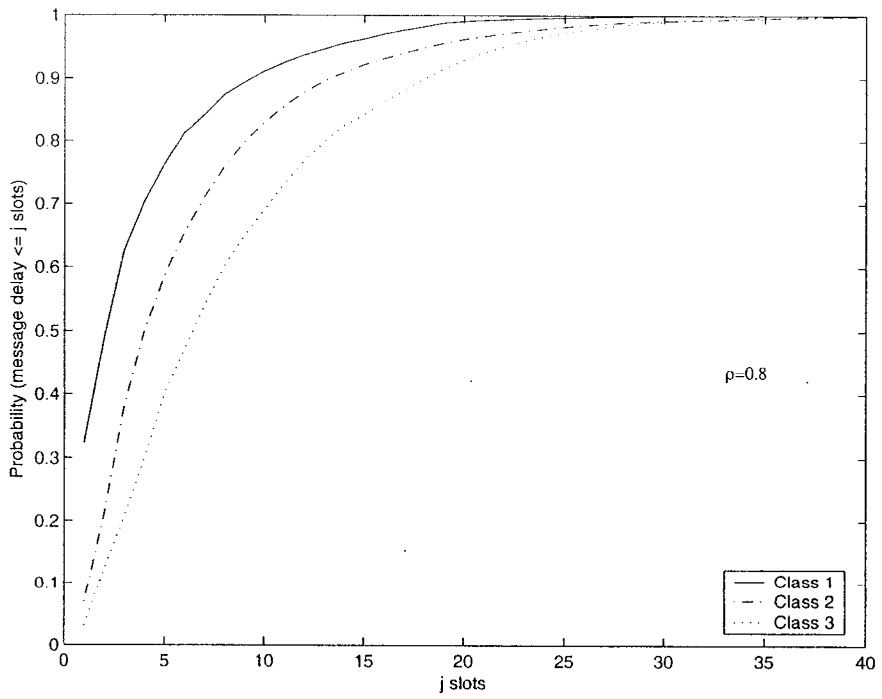

Figure 3.31: Probability distribution of the message delay for each class of traffic under BRFQ

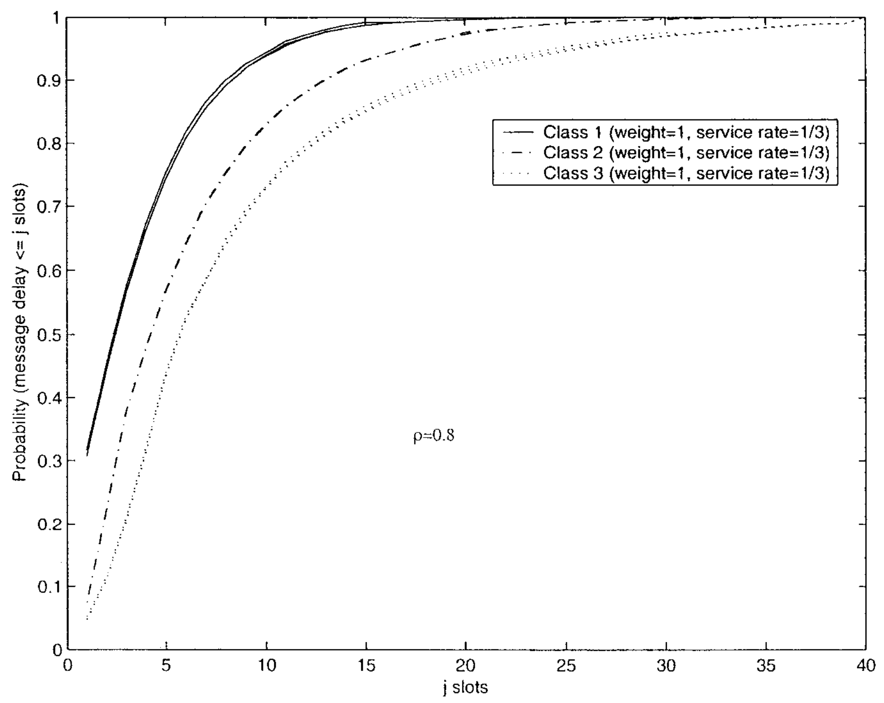

Figure 3.32: Probability distribution of the message delay for each class of traffic under WFQ, WF^2Q, WF^2Q+

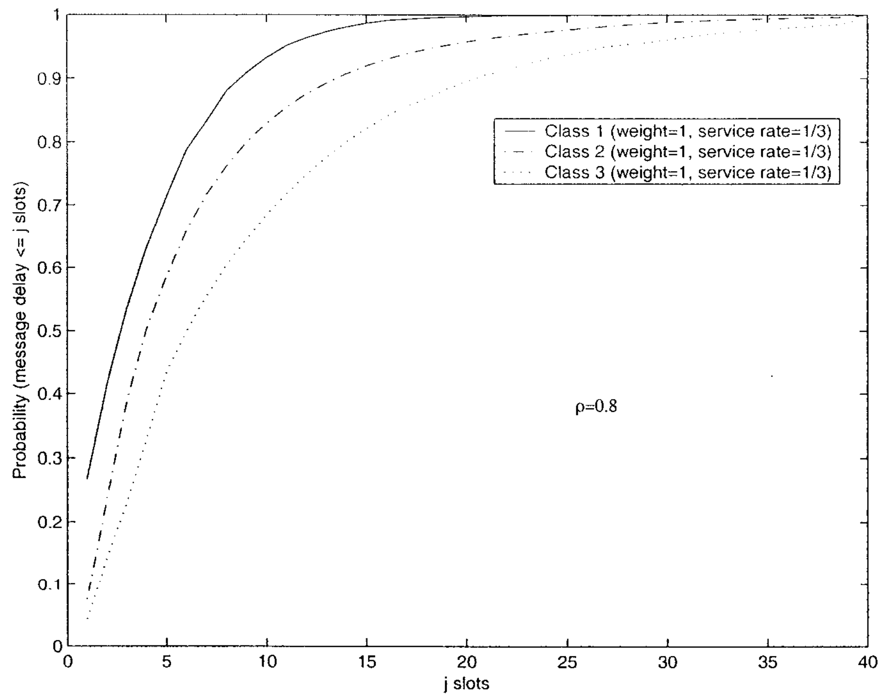

Figure 3.33: Probability distribution of the message delay for each class of traffic under SCFQ

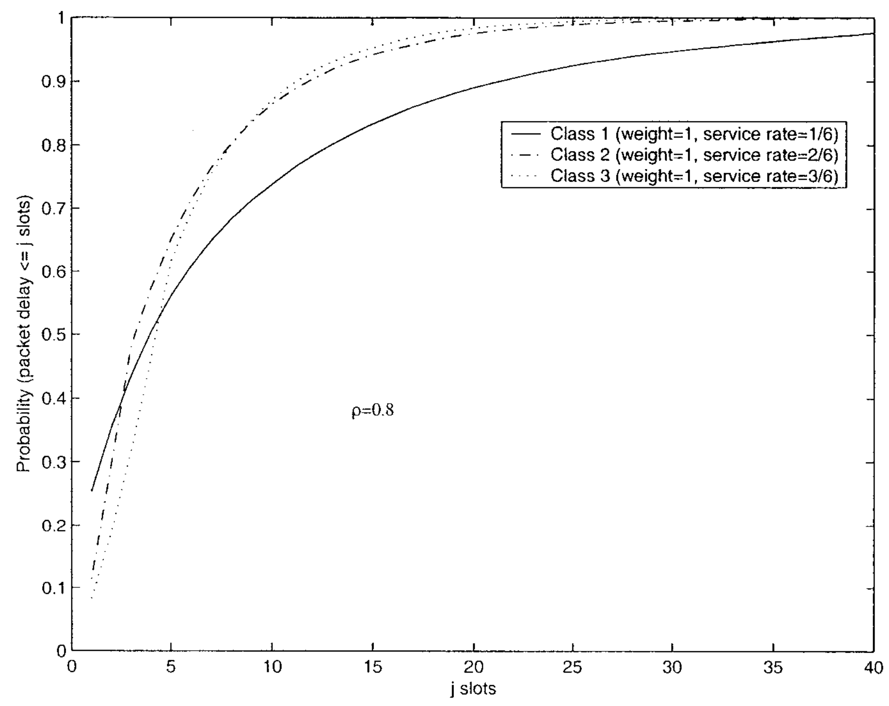

Figure 3.34: Probability distribution of the message delay for each class of traffic under WRR

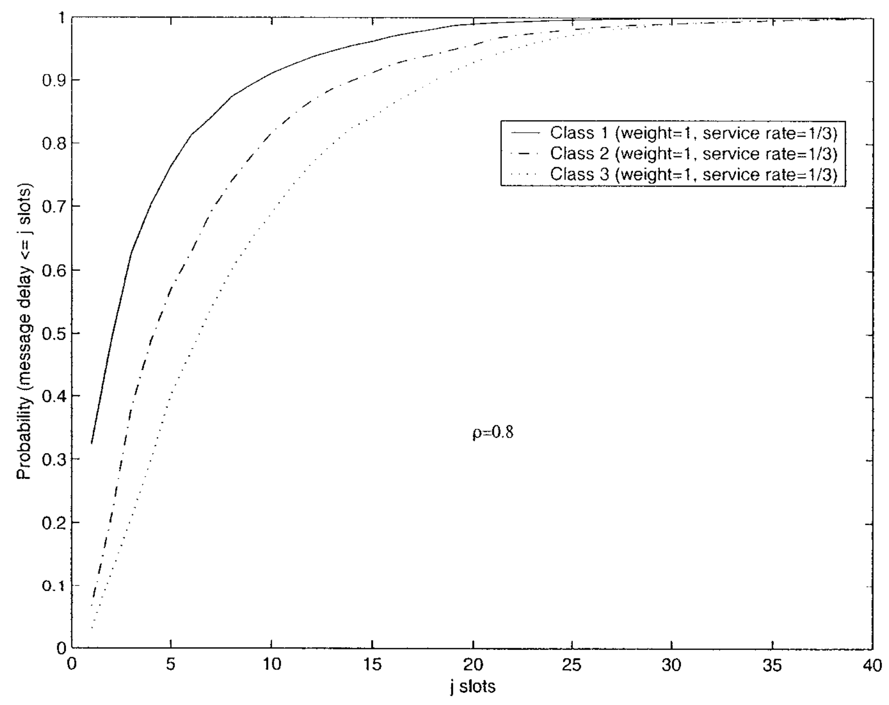

Figure 3.35: Probability distribution of the message delay for each class of traffic under DWRR

3.4.6 Comparison of WFQ and WRR

Next, we would like to show that the WRR may achieve performance similar to WFQ. This is a significant achievement since WRR is a much simpler algorithm than WFQ. Since in the example under study, the three classes have identical traffic loads, WFQ with equal weights provides the same amount of service to each class. As explained earlier on, WRR service discipline doesn't take into account message sizes. We have factored message sizes into WRR service discipline through service weights. Since average message sizes of class 1,2,3 are 1,2,3 packets respectively, we have assigned service weights of 6,3,2 to the corresponding classes. This choice of service weights ensures that each class will receive the same amount of service as in WFQ.

Figures 3.36-3.38 present the simulation results. Figure 3.36 shows the mean message delay under WFQ and WRR. WRR experiences higher delay than WFQ under heavy load. Figure 3.37 shows the mean message delay under WFQ and WRR for each class. As may be seen, WRR and WFQ differ from each other under heavy load. WRR results in higher delay than WFQ except for class 2. Figure 3.38 shows the corresponding probability distribution of message delay for each class, which confirms the observations of mean message delay.

Thus the above results show that WRR may achieve performance similar to WFQ. WRR may be used in place of WFQ in applications that delay requirements are not very strict since it is less complex.

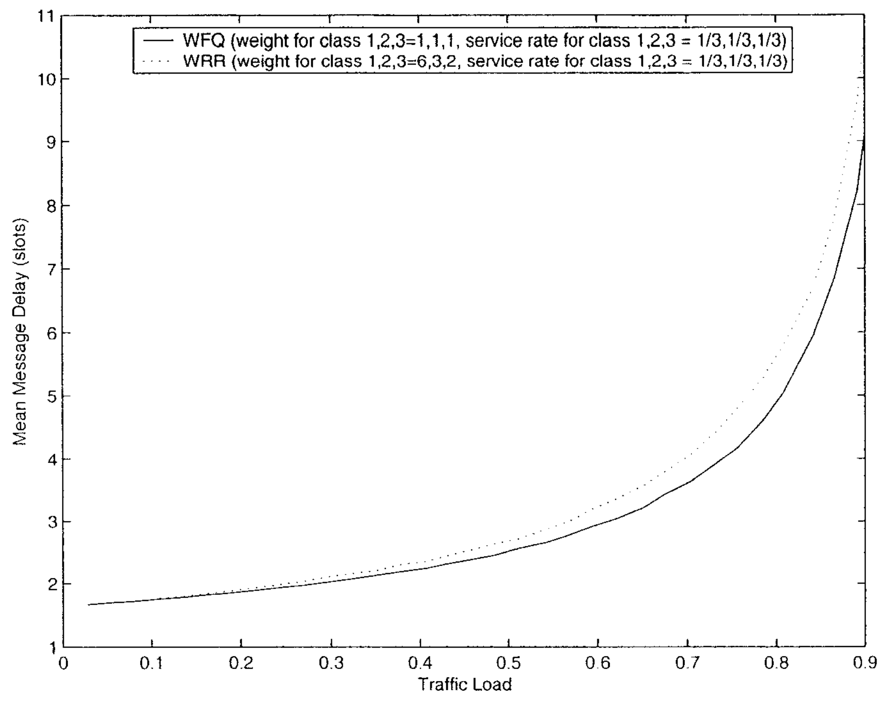

Figure 3.36: Mean message delay under WFQ and WRR

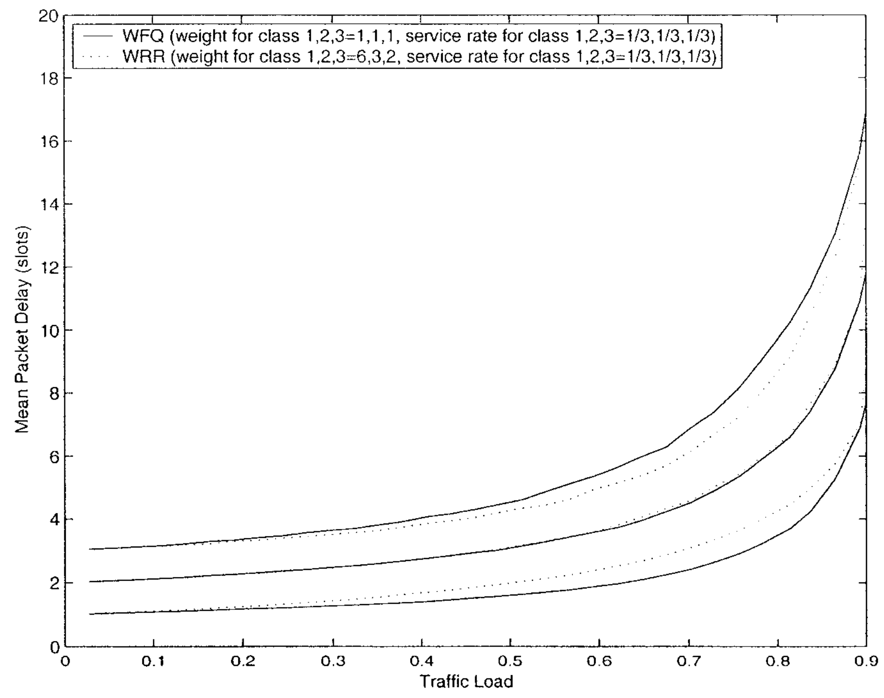

Figure 3.37: Mean message delay for each class of traffic under WFQ and WRR

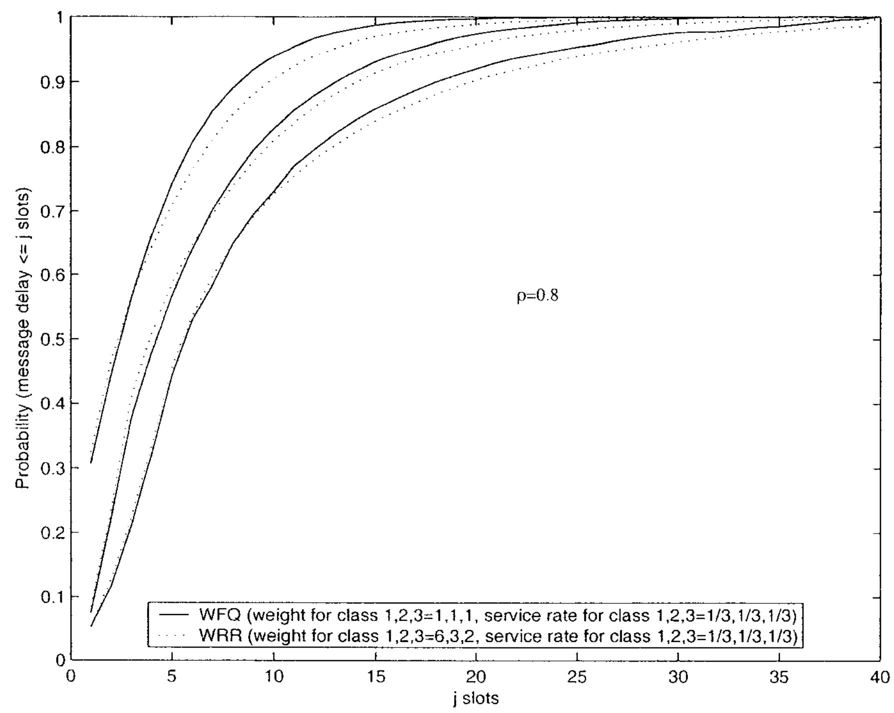

Figure 3.38: Probability distribution of message delay for each class of traffic under WFQ and WRR

3.4.7 Call Admission Control

The objective of this part of the simulation is to study the call admission control in the system under study. We assume that each source represents a call and the number of calls in the system is variable. Thus, the number of sources feeding the multiplexer will be dynamic. If an arriving call to the system is accepted it stays a random amount of time in the system and then it terminates. We note that the call and source blockings are equivalent, therefore we will use them interchangeably. It is assumed that the arrival of a new source during a slot is according to a Bernoulli process, thus during a slot a new source is generated with probability p and no source is generated with probability 1-p. The new source will increase the total load of the system, if the new total load is higher than 1, the new source is blocked. As before a source in the system alternates between On and Off states, however, at the end of an On period a source may leave the system with probability q and remain in the system with probability $1-q$.

Figure 3.39 presents the overall source blocking probability against the probability of new source generation during a slot under FIFO and FQ service disciplines. It may be seen that the results are very close. We also observe the blocking probability of each source class versus the probability of new source generation during a slot under FQ algorithm (Figures 3.40-3.41). In Figure 3.40, three source classes have the same load. The mean message sizes of three source classes are 1,2,3 respectively. We can see that the blocking probabilities are close when the loads from three source classes are same even though their mean message sizes are different. In Figure 3.41, we assume that the traffic load of each class is different and the loads of class 1,2,3 are 1/6,2/6,3/6 of the total load respectively. Since the average message sizes of the three classes are 1,2 and 3, it means that on average equal number of calls from each class will arrive at the system. As expected, class 3 calls experience much higher blocking probability because they demand more resources.

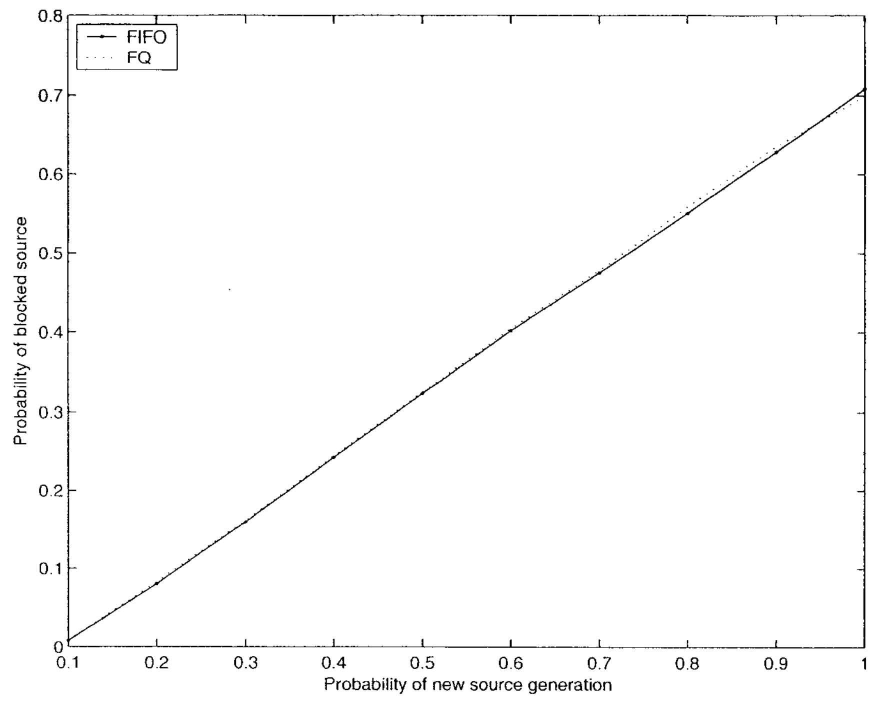

Figure 3.39: Probability of source blocking versus the probability of new source generation

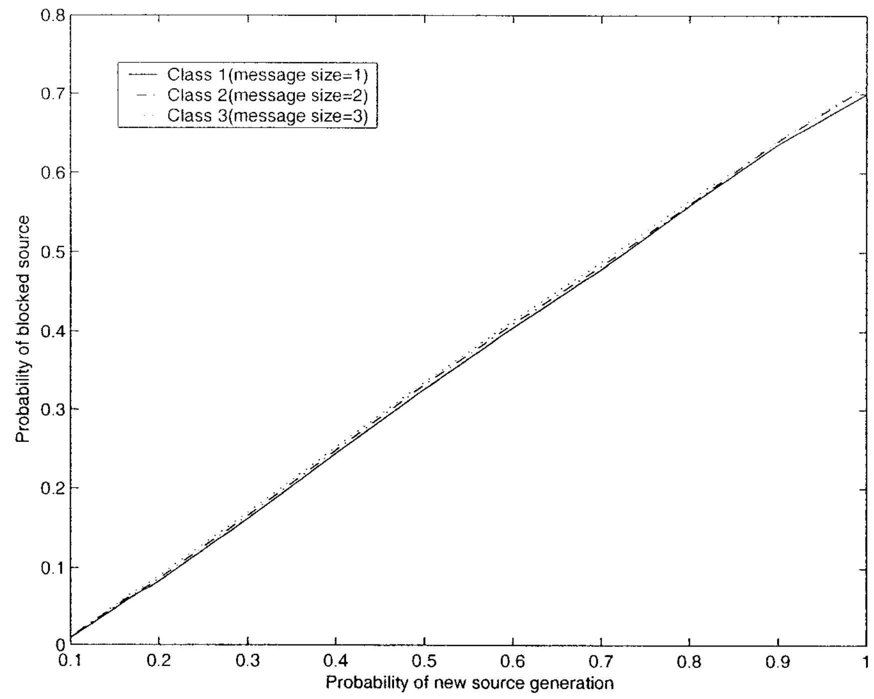

Figure 3.40: Probability of source blocking for each class of traffic with different message sizes

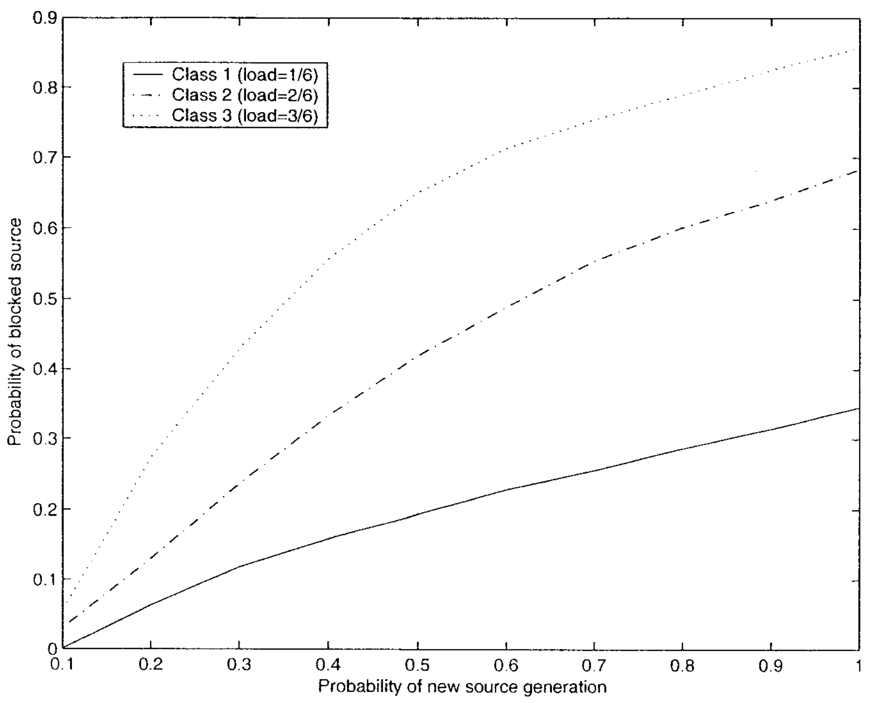

Figure 3.41: Probability of source blocking for each class of traffic with different loads

Chapter 4

A Performance Study of Scheduling Algorithms with TCP Congestion Control

In this chapter, we evaluate the performance of scheduling algorithms with TCP congestion control in the packet-switched networks. First, we investigate the performance of a real-time connection and a TCP connection sharing a single bottleneck link. After describing the congestion control algorithm and the network under consideration, we present the results of the simulation. Then, we study the performance of TCP over high-speed links.

4.1 TCP Congestion Control Algorithm

4.1.1 TCP Flow Control

TCP uses a form of sliding window mechanism to provide flow control. The unit of window size is byte. The size of send window is set through the consulting between sender and receiver when the connection is established. During the communication, receiver can adjust the size of receive window dynamically, and informs sender to adjust its send window size. For this scheme, each individual byte of data transmitted is considered to have a sequence number. When a TCP entity sends a segment, it includes the sequence number of the first byte in the segment data field. A TCP entity acknowledges an incoming segment with a message of the form (A=*i*, W=*j*), with the following interpretation:

- All bytes through sequence number $i-1$ are acknowledged; the next expected byte has the sequence number i.
- Permission is granted to send an additional window (W) of j bytes of data; that is, the j bytes corresponding to sequence numbers i through $i+j-1$.

In our simulation, the send window size is determined according to the following formula [33],

$$wnd = \mathrm{MIN}(cwnd, maxwnd) \qquad (4.1)$$

where,

wnd = the window size used by the sender

maxwnd = the maximum window size that the receiver specifies at TCP connection set-up

cwnd = the congestion window adjusted by the sender in response to network congestion

To simplify the simulation, all window sizes are assumed to be measured in units of maximum message size, instead of bytes. In the original TCP specification [29], the window used by the sender, which we denote by *wnd,* is the receiver advertised window *maxwnd* regardless of the load in the network. In the TCP algorithm in [28], the window size used by the sender is adjusted in response to network congestion. The sender has a variable *cwnd,* which is increased whenever new data is acknowledged and is decreased whenever a message drop is detected. The mechanism used to adjust the size of *cwnd* is shown as the additive increase/multiplicative decrease. The actual window used by the sender is the minimum of the congestion window and the receiver advertised window.

The congestion window adjustment algorithm has two phases, the slow start or congestion recovery phase, during which the window increases exponentially; and the congestion avoidance phase, during which window increases linearly. During the slow start, the sender increases the congestion window by one each time when it receives an acknowledgement from the receiver. During congestion avoidance, the sender increases the congestion window by one after each round-trip time and this is the additive increase. This halving of the control threshold corresponds to the multiplicative decrease. The current phase of the algorithm is determined by a control threshold, *ssthresh.* Whenever a message drop is detected, *ssthresh* is set to half of the current *cwnd* value, *cwnd* is then set to one, and the congestion recovery phase begins. *cwnd* increases rapidly until it passes the threshold *ssthresh.* After *cwnd* passes the *ssthresh,* the algorithm switches into the congestion avoidance phase and from there the *cwnd* increases linearly. The adjustment algorithm is specified below:

When new data is acknowledged, the parameters of the algorithm are set as follows [34],

```
if ($cwnd < ssthresh$)
    $cwnd$ += 1;
else
    $cwnd$ += $1/cwnd$;
```

On the other hand, when a message drop is detected, the parameters are initialized as:

```
$ssthresh = cwnd/2$;
$cwnd = 1$;
```

We define an *epoch* of a TCP connection to be the time period which an entire window of messages have been acknowledged. The amount by which the congestion window increases during an *epoch*, which will be called the acceleration ϕ, is an important measure of how rapidly the window size is changing. Notice that when $cwnd < ssthresh$, *cwnd* doubles during an epoch, so $\phi \approx cwnd$. In contrast, when $cwnd > ssthresh$, *cwnd* increases by approximately 1 during an epoch: $\phi \approx 1$.

4.1.2 TCP Retransmission Strategy

TCP relies on retransmission as error control when an acknowledgement does not arrive within a given timeout duration. There is a timer associated with each segment. When a segment is sent, the timer is set. If the sender doesn't receive acknowledgement before the timer expires, the segment is retransmitted.

The retransmission timer is set according to the following adaptive algorithm specified in RFC 793 [35]. The round-trip time is estimated using exponential averaging:

$$\mathrm{SRTT}(K + 1) = \alpha \times \mathrm{SRTT}(K) + (1 - \alpha) \times \mathrm{RTT}(K + 1) \tag{4.2}$$

where

SRTT(K) = smoothed round-trip time estimate for the last K segments

RTT(K+1) = the round-trip time observed for the (K+1)th transmitted segment

α = constant value ($0 < \alpha < 1$)

α is a constant value that is independent of the number of past observations. We would like to give greater weight to more recent instances because they are more likely to reflect future behavior.

The retransmission timer RTO is set by :

$$\mathrm{RTO}(K+1) = \beta \times \mathrm{SRTT}(K+1) \qquad (4.3)$$

Here, β is also a constant ($\beta>1$). RFC 793 does not recommend specific values for α and β, but does list as "example values" a range of values, $0.8 \leq \alpha \leq 0.9$ and $1.3 \leq \beta \leq 2.0$. In our simulation, we set α to 0.8 and β to 2.

4.2 Simulation Program

In this section, we describe the program used for the system simulation. We will only explain the new features of the program from the one used in the previous chapter.

4.2.1 Simulation Model and Parameters Setting

We will consider a network topology that consists of two tandem links connected by a router. As may be seen from Figure 4.1, two hosts are communicating with each other over this network. We assume that there are two connections between the two hosts, a real-time (RT) and non-real-time (TCP). We assume that RT connection is fed by five On-Off sources and TCP connection always has data to send and the message flow is only controlled by the congestion window. Each connection has its own queue at the source and router. We assume infinite queue except for a finite queue (assumed to be 60 packets size) for TCP traffic at the router. We assume that the link A between the source host and the router has a bandwidth of 100 Mbps and propagation delay of 0.36msec. The link B between the router and the destination is assumed to be the bottleneck, which has a bandwidth of a 90 Mbps and propagation delay of 1.08msec. RT messages have a mean size of three packets and TCP messages are assumed to have a constant size of three packets. We assume that the packet size is 1500 bytes. This results in 120 usec and 133.3

usec packet transmission times over the link A and B respectively. We base the slot duration on the packet transmission time over the link A, 120usec. As a result, the packet transmission times over links A and B are 1 slot and 1.11 slots respectively. The propagation delays over links A and B are 3 slots time and 9 slots time respectively.

To simplify the simulation, we ignore the processing time of the messages at the destination host and assume that the ACK packets on the return channel don't experience any delay and arrive at the sender with a minimum spacing equal to the transmission time of a data message at the bottleneck link. TCP connection is assumed to have a maximum window size of 50 messages. For our network topology the value of *cwnd* never exceeds 50, so that the maximum window size will not be a factor in any of our simulations.

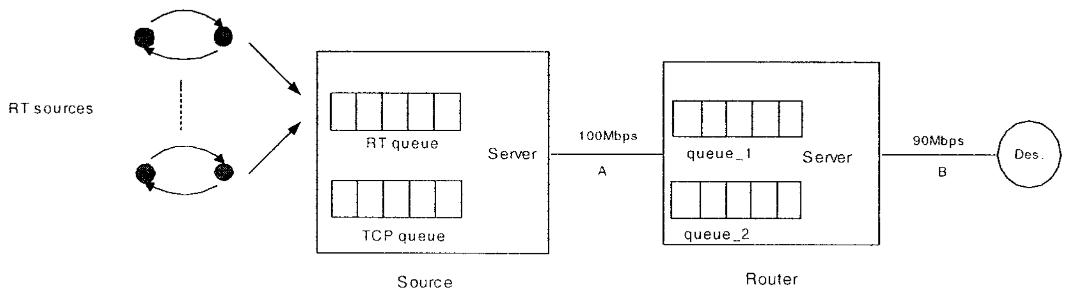

Figure 4.1: Network Model

4.2.2 Class Diagram of the Simulation Program

The main classes of the program are shown in Figure 4.2. In addition to the source, server and queue classes, we have created several new classes.

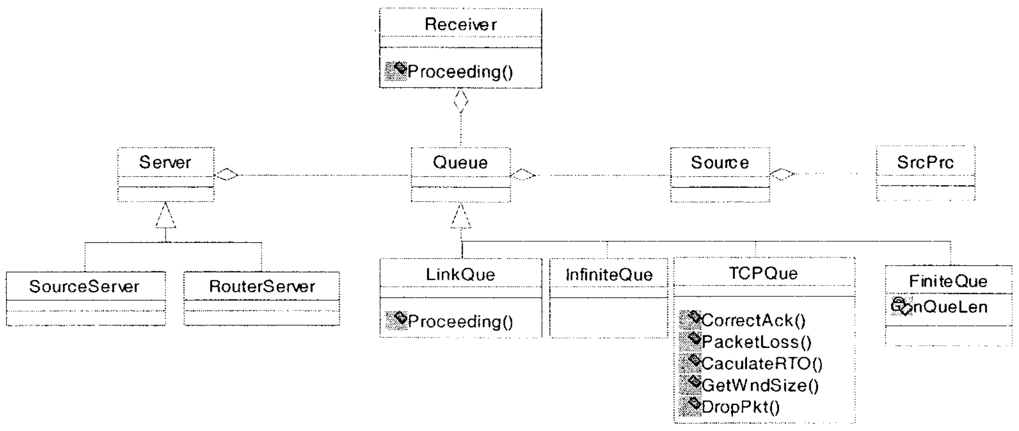

Figure 4.2: Class Diagram of Simulation Program

New **Queue Class** has four subclasses:

- TCPQue class is designed to store TCP messages and send messages within the window size. When a TCP message is send, its corresponding information is still buffered in this queue. When this message is detected to be lost during the transmission time, it will be retransmitted. Message drops are detected by either the receipt of duplicate acknowledgements or the expiration of a timer. This class has more functions than its base class. The function CorrectACK() checks if an ACK from the receiver is correct. It includes the checks of the sequence number and time out duration. The function MessageLoss() checks if a message is lost in the transmission. When the timer expires or the sender receives a wrong sequence number, this determines a message loss. The function CalculateRTO() calculates the time out duration for a new send message from the round-trip times of the previous messages. The function GetWndSize() calculates the window size according to the adjustment algorithm described in chapter 4.1.1. The function

DropPkt() clears the corresponding information of a transmitted message after receiving its correct ACK.

- FiniteQue class is designed to handle the finite queue size and stores TCP messages at the router. When the buffer is full, the new coming messages will be dropped.
- InfiniteQue class is designed to handle the infinite queue size and stores RT messages.
- LinkQue class is designed to simulate the links between the source and router and between the router and receiver. Messages experience constant delays (propagation delays) over the links. The function Processing() provides a constant delay for a message.

SourceServer and **RouterServer classes** are the subclasses of the Server class:

- SourceServer includes all the functions to control message transmission at the source end. The name of functions in this class is same with its base class we described in chapter 3.3.2, but there are differences in the implementation and coding. PQ and WRR algorithms have been used to schedule the messages.
- RouterServer includes all functions to control the message transmissions from a finite queue and infinite queue at the router.

Receiver class includes functions to send an ACK to the sender. The main function Proceeding() sends an ACK including receiving time and the sequence number of a TCP message.

4.2.3 Simulation Flow Chart

Figure 4.3 is the flow chart of the simulation system, for both RT and TCP traffic, infinite and finite queues, scheduling algorithms, TCP flow control and retransmission strategy. All procedures execute within a loop representing a time slot over the link A. In this loop, we simulate all the activities for the transfer of a message from the source to the receiver. As described earlier, the bandwidth on the link B is the 90% bandwidth on the link A. Because the loop in our program is based on the slot time on the link A, the router terminates a packet transmission in one slot after 9 slots to simulate this slower transmission rate on the link B.

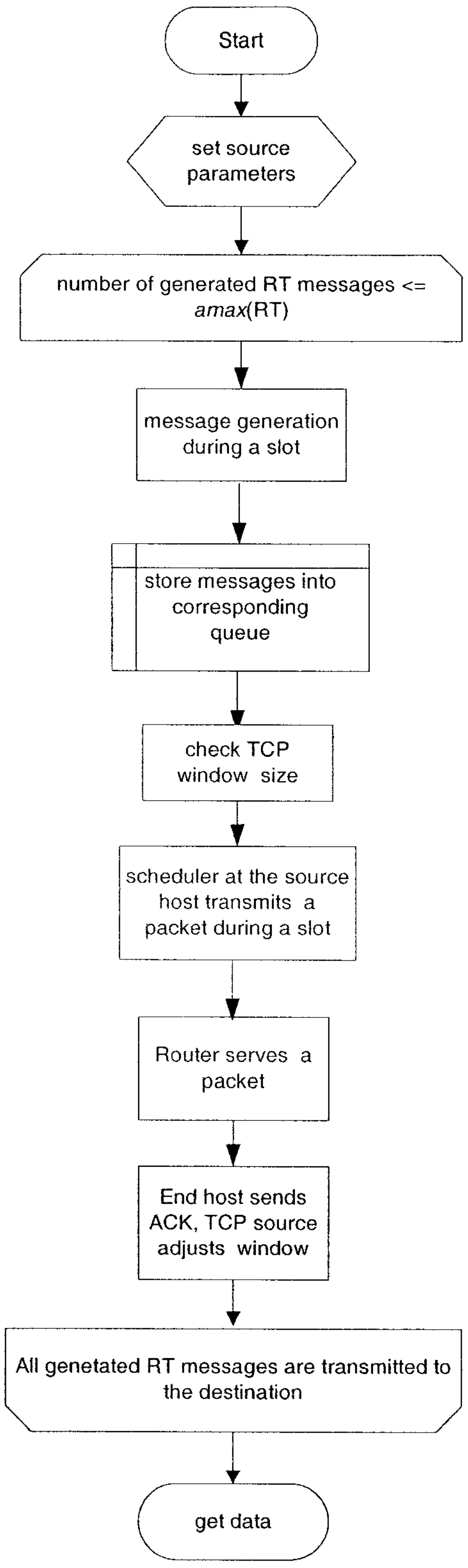
Start
set source parameters
number of generated RT messages <= amax(RT)
message generation during a slot
store messages into corresponding queue
check TCP window size
scheduler at the source host transmits a packet during a slot
Router serves a packet
End host sends ACK, TCP source adjusts window
All genetated RT messages are transmitted to the destination
get data

Figure 4.3: Flow Chart of Simulation

4.3 Simulation Results

In this section, we present simulation results for the system under consideration. The objective is to study the performance of TCP traffic under the multiplexer scheduling algorithms.

We have studied the performance of the system under two scheduling algorithms, Priority Queueing and Weighted Round Robin respectively. The size of finite queue at the router has been set to 60 packets size. Under the PQ (Priority Queueing), the RT traffic is assigned higher priority than the TCP traffic at both source end and the router. TCP messages can only be transmitted when the RT queue is empty. Under the WRR (Weighted Round Robin), RT queues at the source and router are assigned higher service weight than the TCP queues.

First we consider the performance of TCP and RT traffics at the source end (Figure 4.4- Figure 4.9). Figure 4.4 presents the mean message delay of RT traffic versus its load under PQ and WRR algorithms. As expected, under PQ algorithm, the performance of RT traffic is not affected by the TCP traffic. On the other hand, under WRR, the performance of RT traffic depends on its 'service weight' which is proportional to the service rate provided. As the service weight assigned to RT traffic increases, its performance approaches to that of under PQ.

Figure 4.5 presents the standard deviation of the message delay versus the RT traffic load under the PQ and WRR. It shows that the message delay variation increases as the traffic load increases.

In Figure 4.6, we compare the probability distribution of RT message delay under PQ and WRR. We can see that the probability distribution of message delay under PQ is steeper than that under WRR. As the service weight for RT traffic increases, the corresponding probability distribution becomes steeper. This means that the RT messages experience shorter delay under PQ than under WRR.

Figure 4.7 presents the mean transmission time of TCP messages versus the RT traffic load. Since we have assumed that the TCP source always has a message to transmit, only the transmission time of a message may be determined. The transmission time is the time interval between the arrival time of the correct ACK for that message and the first transmission of that message. When a message drop is detected, the source retransmits the message. We observe that the message transmission time increases with the increase of the RT traffic load. When the RT traffic load is light, the mean TCP transmission time under the two scheduling algorithms is similar. When the RT traffic load becomes heavy, the mean TCP transmission time under PQ increases sharply. The reason is that the amount of service provided to the TCP traffic decreases with the increase of the RT traffic load. We see that the increase of mean TCP transmission time under WRR algorithm is less steep than that under PQ algorithm. As the service weight assigned to TCP traffic decreases, the curve of mean transmission time approaches to that under PQ. This shows that the resource distribution to different queues under WRR is fairer than that under PQ if the suitable weights are assigned to different types of traffic. The WRR algorithm not only provides priorities to different queues but also don't entirely deny service to the lower-priority queues. In Figure 4.8, we compare the probability distribution of TCP transmission time under PQ and WRR. We conclude that the result in this figure is in agreement with that of the Figure 4.7.

Figure 4.9 presents the throughput of TCP traffic versus RT traffic load under the two scheduling algorithms. The unit of throughput is packets/slot. It may be seen that the traffic load for the high-priority queue affects the throughput of the lower-priority queue: as the RT traffic load increases, the throughput of TCP traffic decreases, because server has to spend more time serving the RT traffic. It may be seen again that the service given to the TCP traffic is better protected under the WRR than the PQ algorithm. As the weight assigned to TCP traffic increases, the throughput of TCP traffic increases under the WRR algorithm.

Figures 4.10-4.13 present the performance of the two types of traffic at the router. The infinite queue is for RT traffic and the finite queue is for TCP traffic. Note that we always use the same scheduling algorithm both at the source and router, thus both of them are either PQ or WRR.

Figure 4.10 presents the mean message delay of RT traffic as a function of its load at the router. We can see that there is longer delay for RT messages under WRR algorithm than under PQ algorithm. Compared with Figure 4.4, we observe that the curve under WRR increase sharper under heavy load. The reason is that TCP traffic have larger throughput when WRR is used at the source end. Therefore the server at the router has to spend more time in the finite queue under WRR algorithm.

Figure 4.11 presents the TCP message loss probability versus RT traffic load at the router. Message loss includes both losses due to buffer overflow and the timeouts. As the RT traffic load increases, the message loss probability of TCP messages at the router also increases. It shows that the message loss increases sharply under the PQ algorithm when the RT traffic load becomes heavy. On the other hand, the increase of message loss under

WRR algorithm is not sharp and it levels off under heavy traffic because of service protection of WRR.

Finally, Figures 4.12 and 4.13 present the probabilities of TCP message loss versus RT traffic load for different buffer sizes under PQ and WRR algorithms.

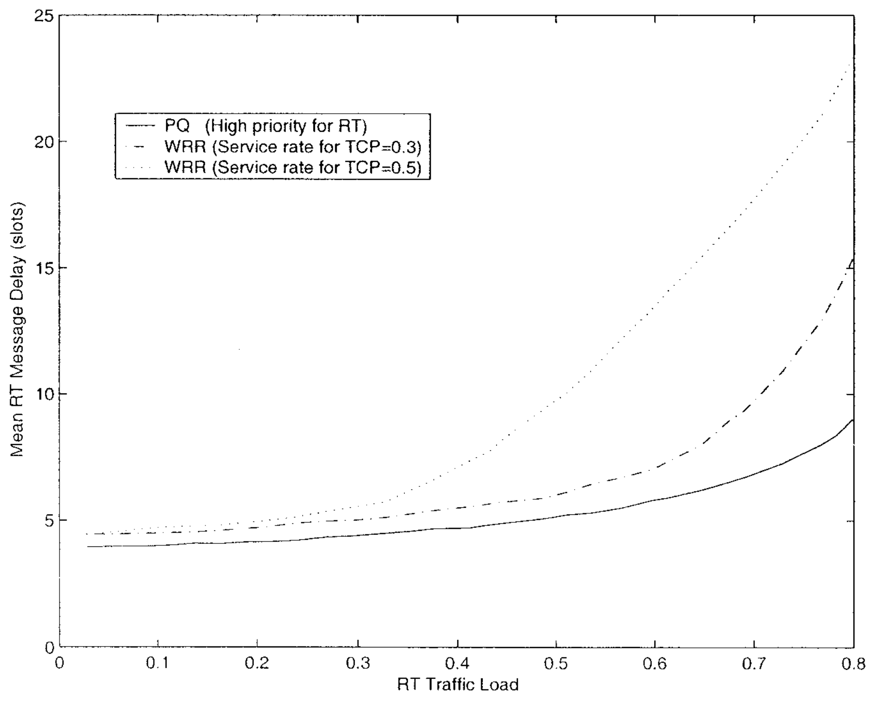

Figure 4.4: Mean message delay of RT traffic versus its load under PQ and WRR

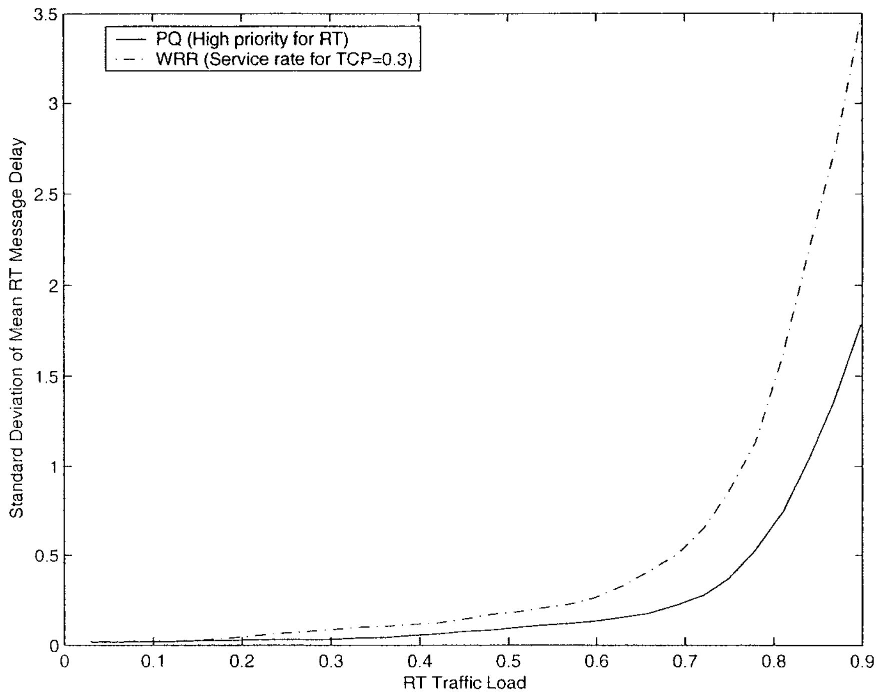

Figure 4.5: Standard deviation of RT message delay versus its load under PQ and WRR

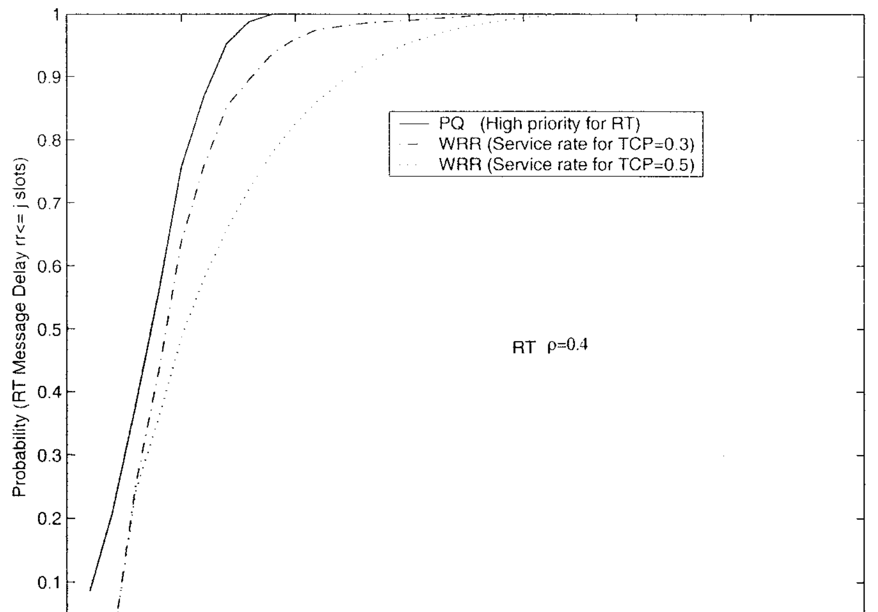

Figure 4.6: Probability distribution of RT message delay under PQ and WRR

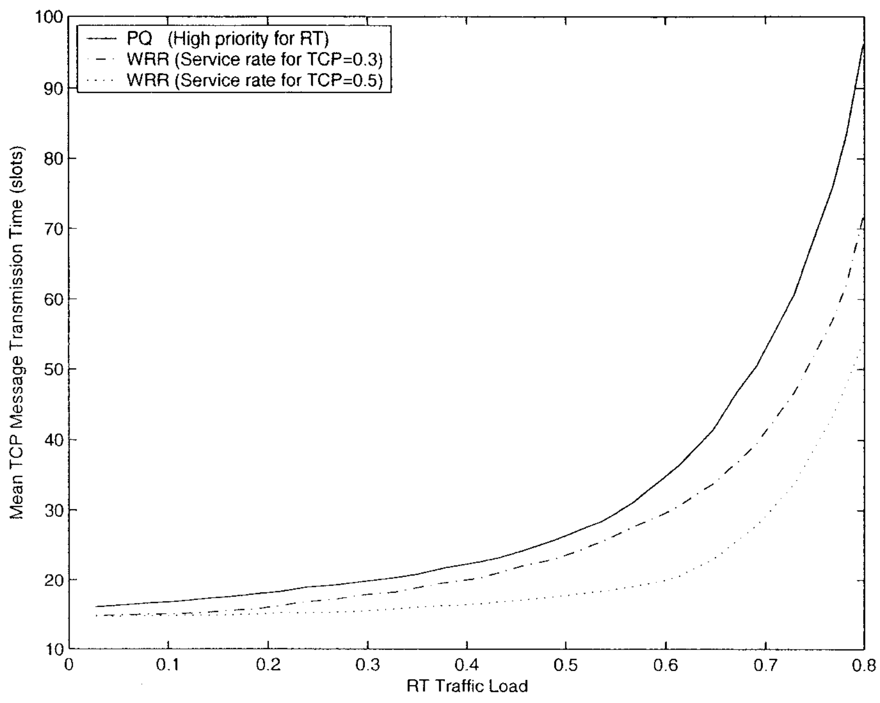

Figure 4.7: Mean transmission time of TCP traffic versus RT traffic load under PQ and WRR

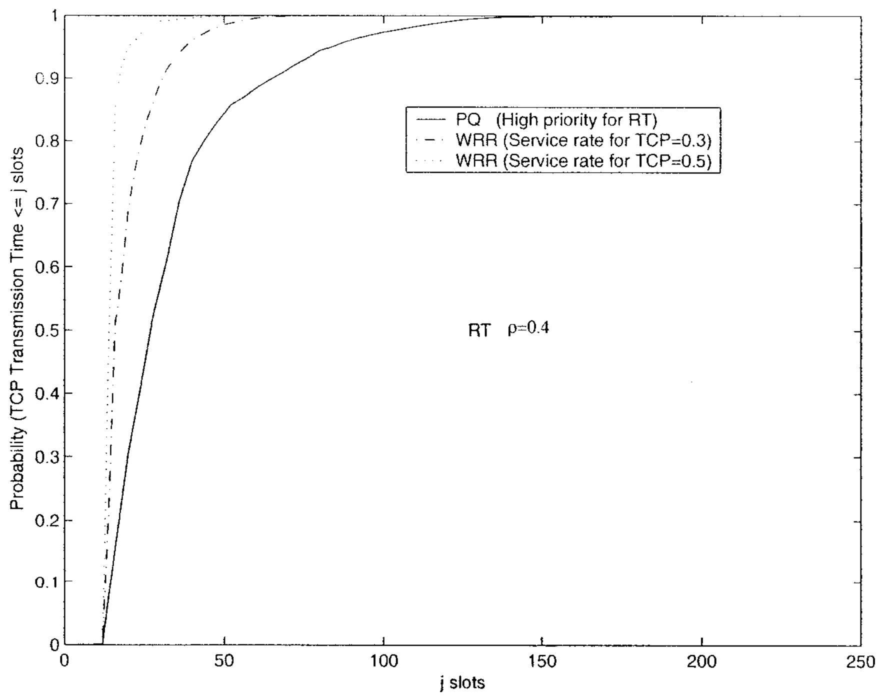

Figure 4.8: Probability distribution of TCP transmission time under PQ and WRR

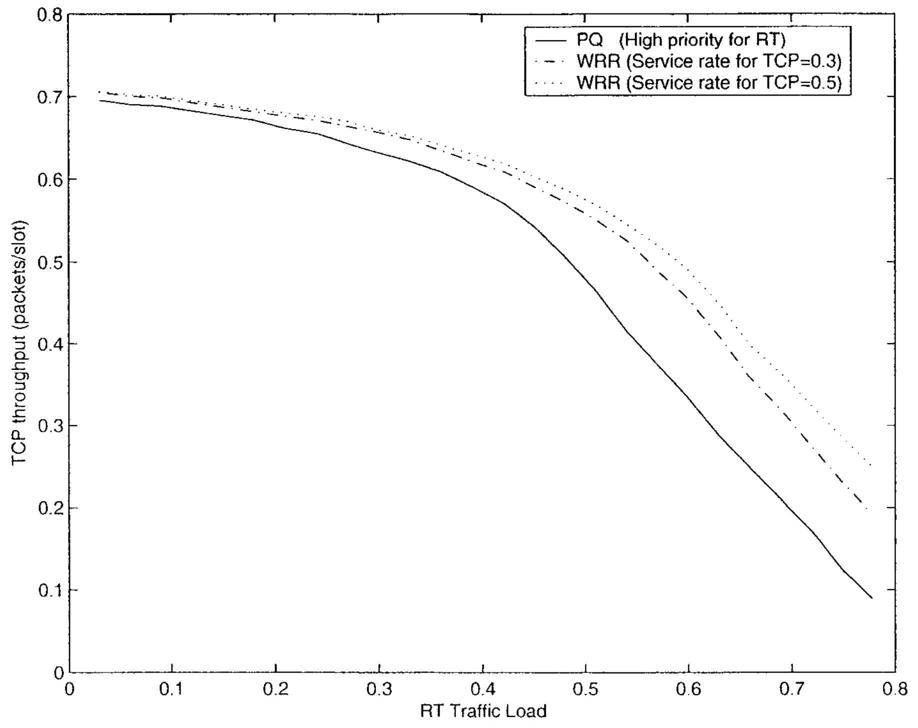

Figure 4.9: Throughput of TCP traffic versus RT traffic load

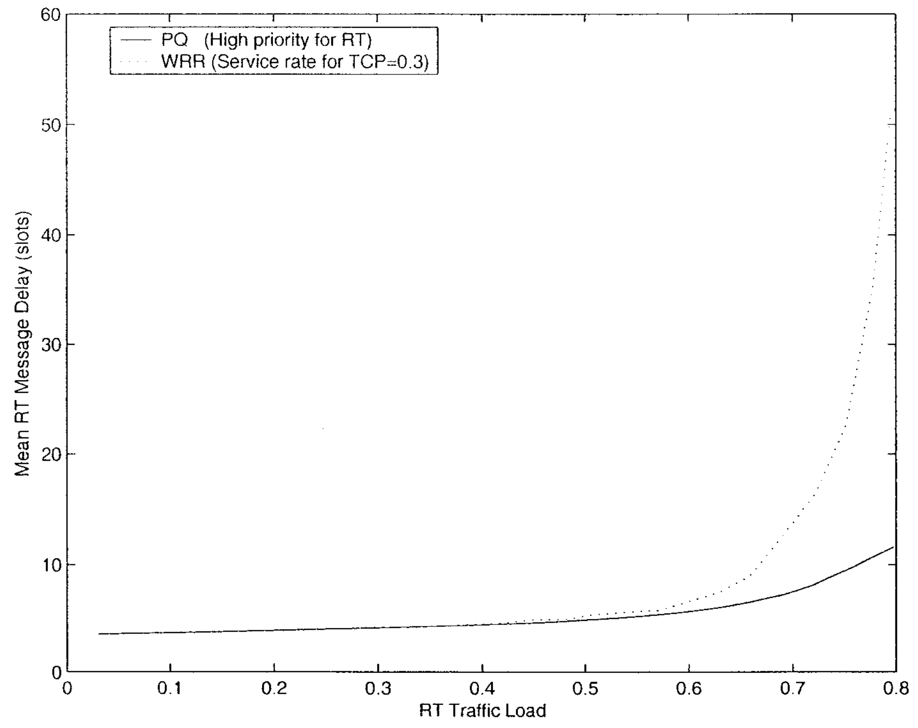

Figure 4.10: Mean RT message delay versus its load at the router

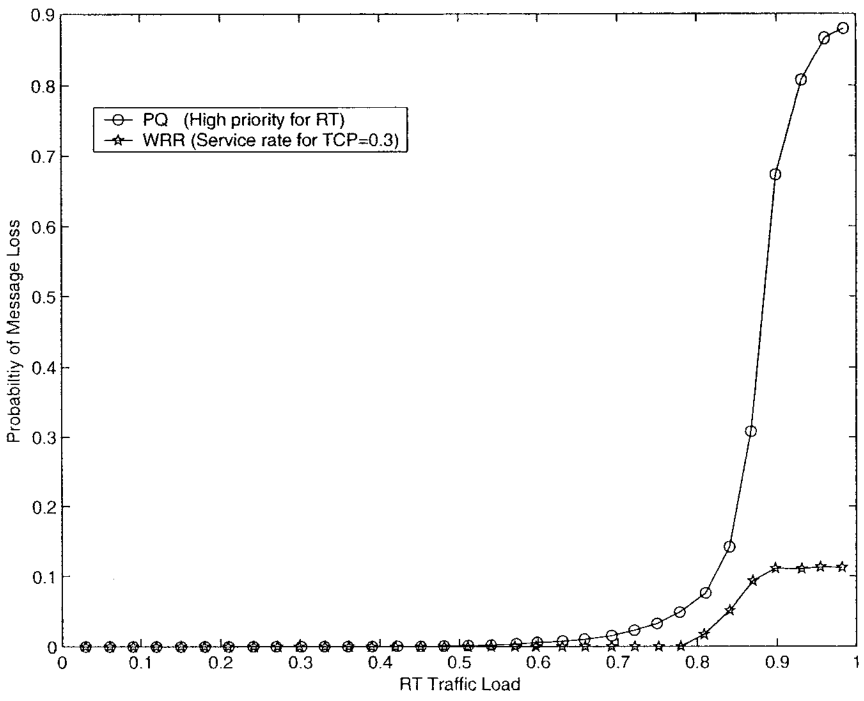

Figure 4.11: Probability of TCP message loss versus RT traffic load at the router under PQ and WRR

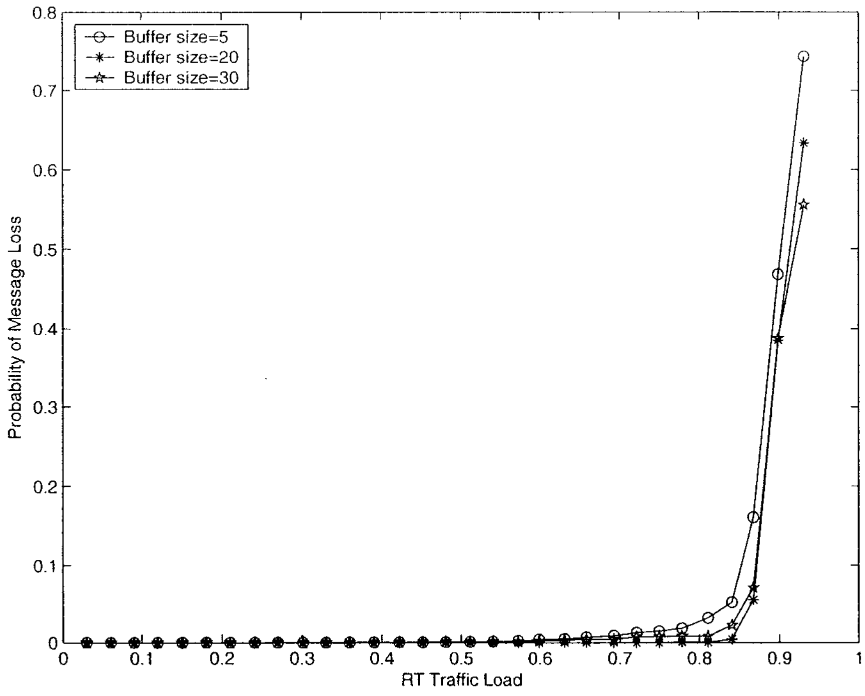

Figure 4.12: Probability of TCP message loss versus RT traffic load with different buffer sizes under PQ

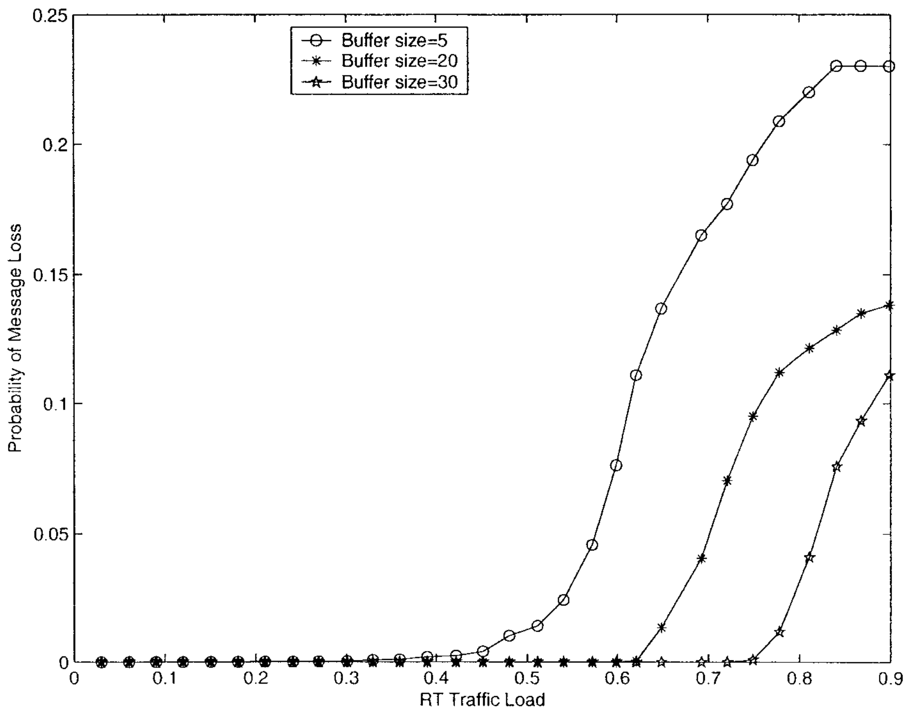

Figure 4.13: Probability of TCP message loss versus RT traffic load with different buffer sizes under WRR

4.4 The Performance of TCP Congestion Control Algorithm over High-speed Transmission Links

Recently, the performance of TCP congestion control algorithm over high-speed transmission links has been receiving growing attention. As explained earlier, the main feature of TCP congestion control algorithm is its additive increase/multiplicative decrease property. This congestion control algorithm is proved to be inadequate as the speed of the transmission links increases and users demand higher throughput [13]. For example, certain future TCP applications may require a throughput of 1-2 Gbps over a 10 Gbps high-speed transmission line. The main drawback of the present TCP congestion control algorithm is its additive increase/multiplication decrease feature. As a result, it may take minutes for a TCP connection to recover from a message loss. The [13] has proposed to modify the congestion control algorithm to remedy this problem. High-speed TCP is designed to have a different response in environments of very low congestion event rate, and to have the standard TCP response in environments with packet loss rates of at most 10^{-3}. In environments with very low packet loss rates, high-speed TCP presents a more aggressive response function. The high-speed TCP response function is specified using three parameters: Low_Window, High_Window and High_P. Low_Window is used to establish a point of transition and ensure compatibility. The high-speed TCP response function uses the same response function as the regular TCP when the current congestion window is at most Low_Window. High_Window and High_P are used to specify the upper end of the high-speed TCP response function. It is set as the specific packet drop rate High_P, needed in the high-speed TCP response function to achieve an average congestion window of High_Window. As described earlier: in congestion avoidance phase, the congestion window (*cwnd*) can be expressed by the following equations:

ACK: $cwnd \leftarrow cwnd + a(cwnd)/cwnd$ (4.3)

DROP: $cwnd \leftarrow cwnd + b(cwnd) * cwnd$ (4.4)

For standard TCP, $a(cwnd) = 1$ and $b(cwnd) = ½$. For $cwnd$ = High_Window, there is following relationship between $a(cwnd)$ and $b(cwnd)$:

$a(cwnd) = \text{High\_window}^2 * \text{High\_P} * 2 * b(cwnd)/(2 - b(cwnd))$. (4.5)

As a result, the high-speed TCP response function will have faster additive increase and slower multiplicative decrease than standard TCP. In [13], it has been shown that this will provide big users with high throughput over high-speed links. Unfortunately, there is another problem related to the transmission over high-speed links. Since TCP limits the maximum message size to 1500 byte, the window size of a big user will be in tens of thousands. Clearly, this will substantially increase the processing load of routers. Another proposal to remedy the problem of high-speed TCP is to change the maximum message size in TCP while keeping the additive increase/multiplicative decrease congestion control algorithm [15]. This solution doesn't result in very large window sizes as the previous solution. However, both solutions have a common problem that they fail to protect the low-throughput users against the high-throughput users. Low-throughput users don't receive their share of bandwidth at the presence of high-throughput users. Next, we demonstrate this through a number of simulation results.

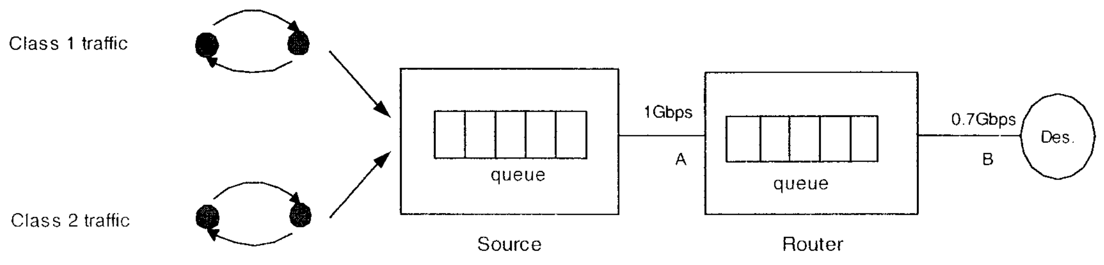

Figure 4.14: Simulation Model

Simulation model is showed in Figure 4.14. Low and high throughput user traffics (class 1 and 2) are stored in a single queue at the source and router. The class 1 traffic load is kept constant value 0.5 while the class 2 traffic load is varied. We assume that the link A between the source host and router has a bandwidth of 1 Gbps and propagation delay of 0.36msec. The link B between the router and the destination is assumed to be the bottleneck. It has a bandwidth of 0.7 Gbps and propagation delay of 0.12msec. We assume that the packet size is 1500 bytes. This results in 12 usec and 17.1 usec packet transmission times over the link A and B respectively. We base the slot duration on the packet transmission time over link A, 12 usec. As a result, the packet transmission times over links A and B are 1 slot and 1.33 slots respectively. The propagation delays over links A and B are 30 slots time and 10 slots time respectively. Figure 4.15-4.17 present the throughput of class 1 versus the traffic load of class 2. In Figure 4.15, the average message sizes of low and high-throughput users are 1 and 10 packets respectively. As may be seen, the throughput of class 1 decreases as the load of class 2 increases. In Figure 4.16, it is assumed that the standard TCP algorithm controls the flow in both queues. As may be seen, the throughput of class 1 decreases as the load of class 2 increases. Next in Figure 4.17, we assume that low and high throughput traffic use standard TCP and high-speed TCP respectively and they have equal message sizes of single packet. As explained earlier on, high-speed TCP uses faster than the additive increase and slower than the multiplicative decrease compared to standard TCP. As may be seen, the throughput of class 1 decreases as the load of class 2 increases. Thus none of the above three solutions is able to protect the bandwidth share of low throughput users in the presence of high throughput users.

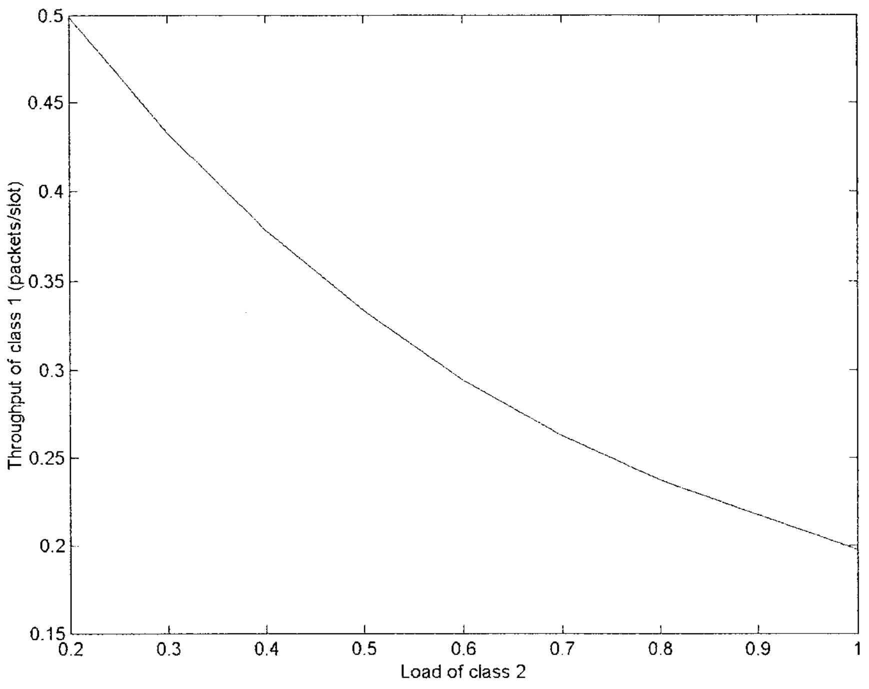

Figure 4.15: Throughput of class 1 versus the traffic load of class 2

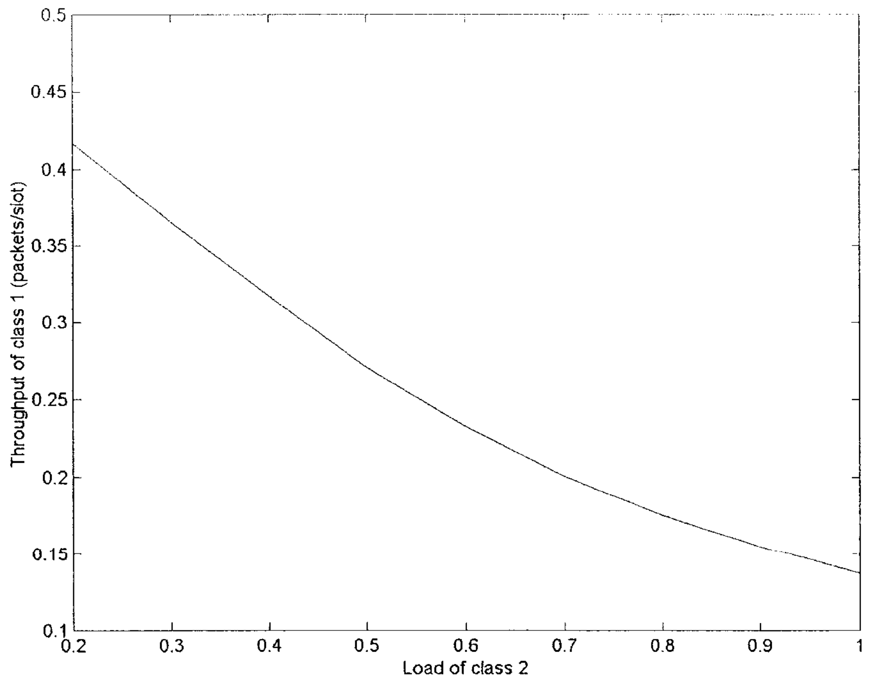

Figure 4.16: Throughput of class 1 versus the traffic load of class 2

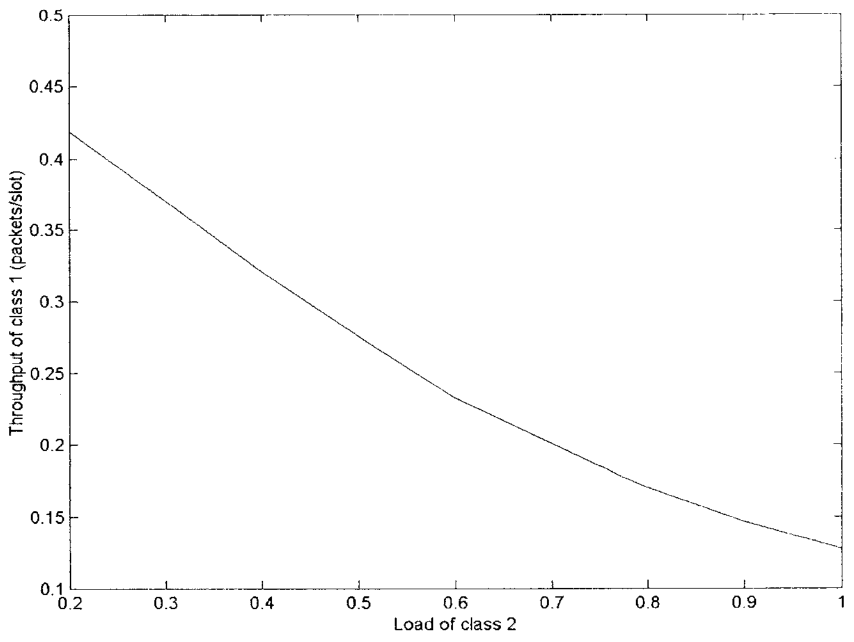

Figure 4.17: Throughput of class 1 versus the traffic load of class 2

Next we consider a system with different maximum message sizes for low-throughput and high-throughput users. Low-throughput users will still have a maximum message sizes of 1500 bytes, while high-throughput users will have a maximum message size of may be 10 times higher. The low and high throughput users traffics will be stored in two separate queues at the source and router. They will be controlled by the standard TCP. The system will provide different throughputs to the two types of users by assigning appropriate service weights to each queue.

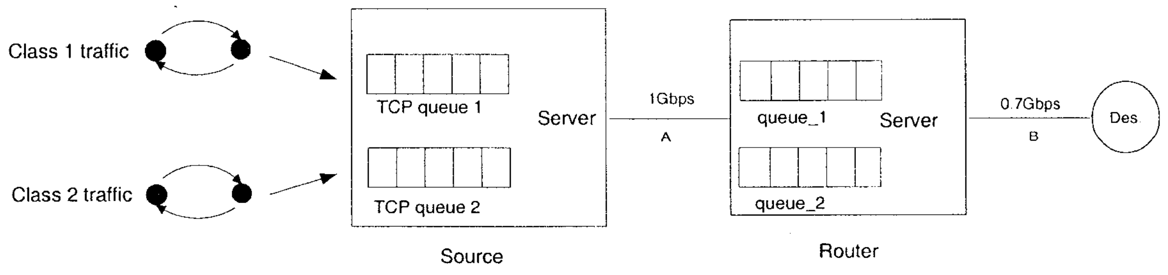

Figure 4.18: Simulation Model

Next, we have studied the performance of the proposed algorithm through simulation. It will be seen that the throughput of the high-throughput users increases with the increase of their service weight while low-throughput users are being protected. We first study the performance in the model of Figure 4.18. Low and high throughput user traffics (class 1 and 2) are stored in separate queues at the source and router. The load of low-throughput user is also kept 0.5 while the load of high-throughput user is varied. The average message sizes of low and high throughput users are 1 and 10 packets respectively. We have used WRR scheduling algorithm in serving to the two queues. Figure 4.19 presents throughput of class 1 versus load of class 2 under different service rate. As may be seen, the throughput of class 1 can be controlled by assigned different service weight. The throughput of class 1 increases as the assigned service weight increases.

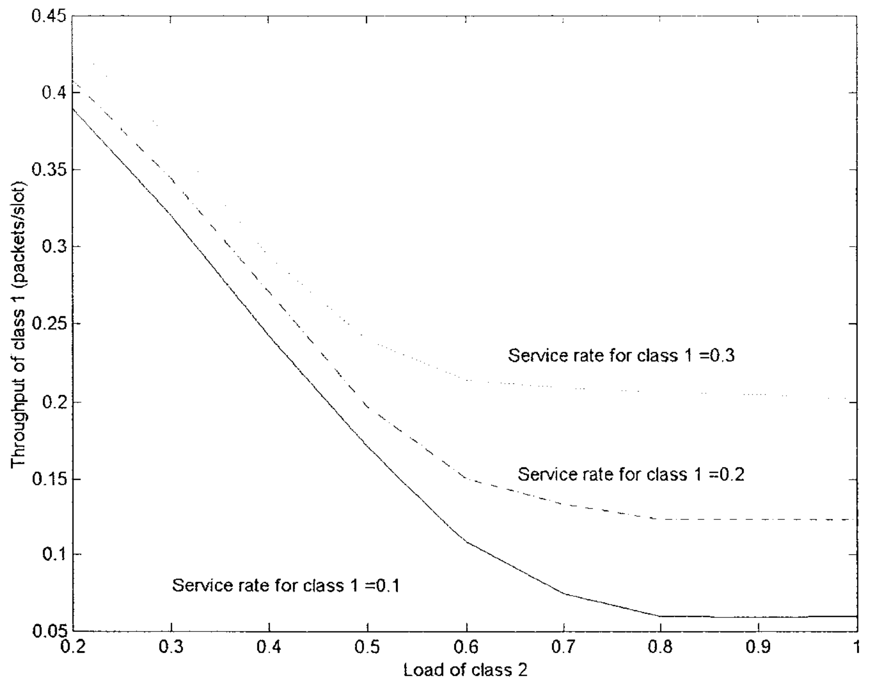

Figure 4.19: Throughput of class 1 versus traffic of class 2 under different service rate

Next we present the simulation results in the model of Figure 4.20. Low and high-throughput user traffics are stored in TCP queues 1 and 2 respectively. We assume that both source queues are saturated, thus they always have messages to transmit. The message size of low-throughput user traffic 1 is assumed to be 1 packet and the message size of large user traffic 2 is assumed to be 10 and 50 packets respectively. We assume finite queue size of 500 packets at the router. Message loss includes message drops at the tail of finite queues as well as message timeouts. We have also used WRR scheduling algorithm in serving to the two queues.

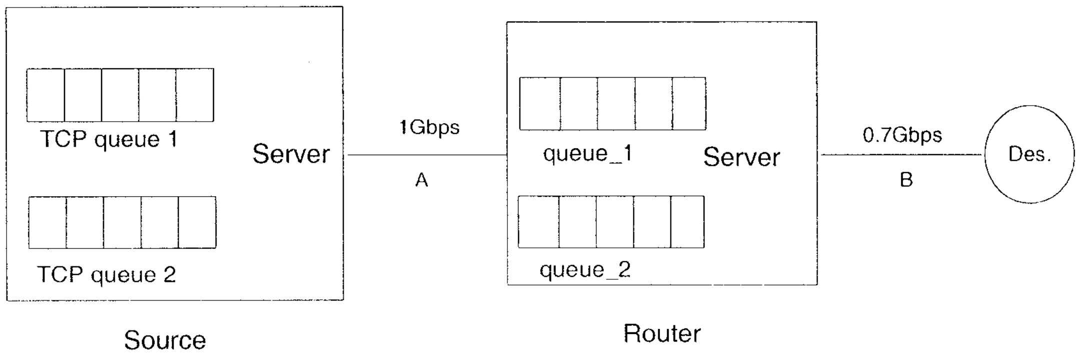

Figure 4.20: Simulation Model

Figures 4.21-4.26 present some simulation results. Figure 4.21 presents the throughput of each class versus service rate of class 2 traffic. We observe that the throughput of class 2 (high-throughput user) increases together with its service rate. It may be seen that the message size doesn't affect the results. Figure 4.22 presents the average round trip time of class 2 versus its service rate. We observe that the messages with larger size experience longer round trip time than those with smaller size. Figures 4.23-4.24 present the probability of message loss for class 2 traffic as a function of its service rate. We observe that the probability of message loss decreases as its service rate increases. Figure 4.24 presents message loss probability of class 2 traffic due to buffer over flow versus its service rate. Comparison of Figure 4.23 and Figure 4.24 shows that most of message losses are due to timeouts. Figure 4.25 presents the probability distributions of window sizes for both classes for a given service rate. Figure 4.26 presents the average window size for class 2 traffic as a function of its message size for constant values of service rate. As may be seen, the average window size drops as the message size increases. This confirms that the large message size keeps the required window size small.

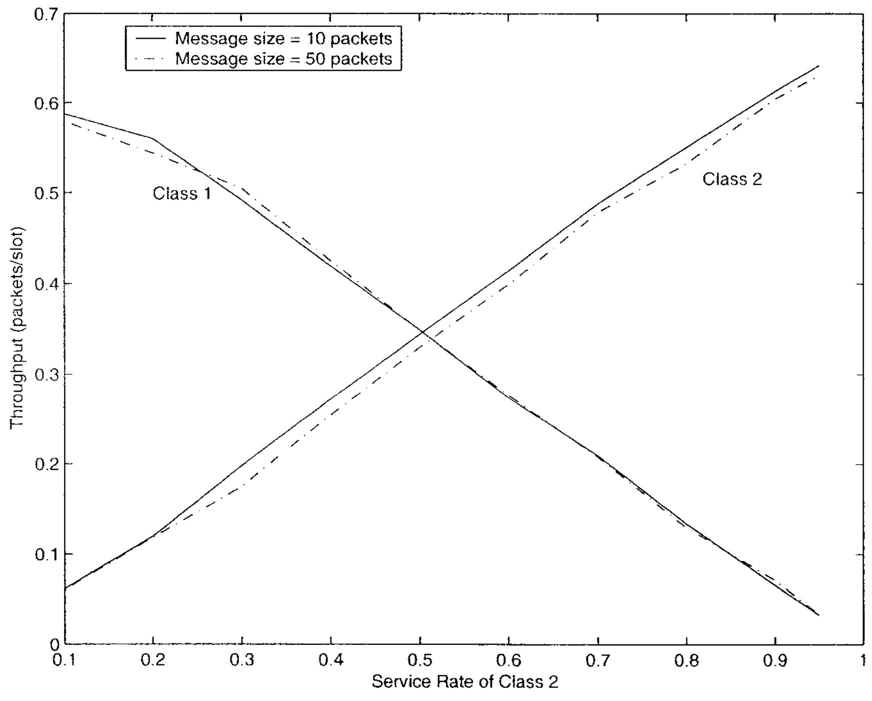

Figure 4.21: Throughtput versus service rate of class 2 traffic

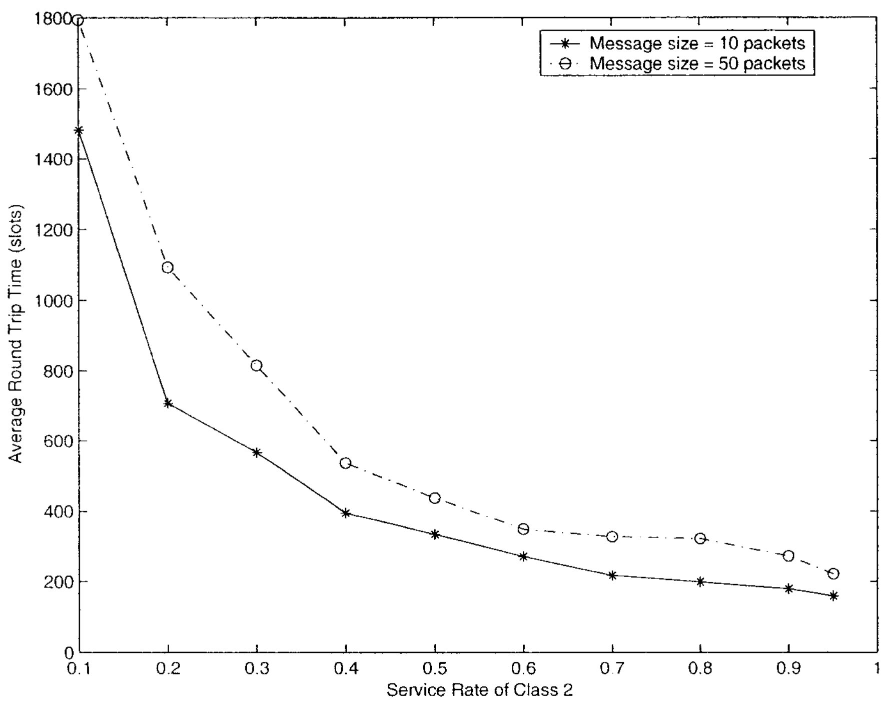

Figure 4.22: Average round trip time of class 2 traffic versus its service rate

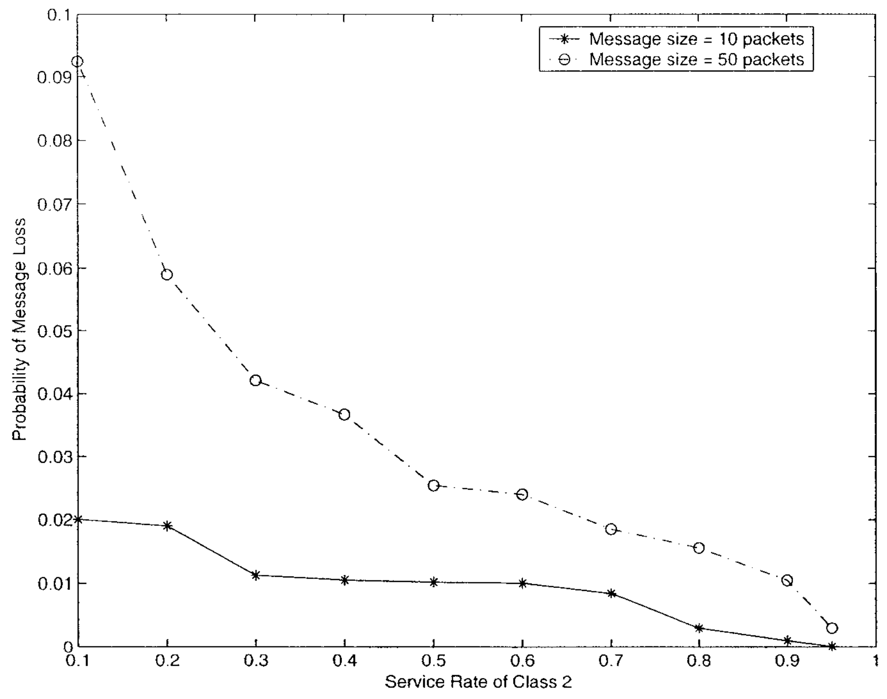

Figure 4.23: Probability of message loss for class 2 traffic versus its service rate

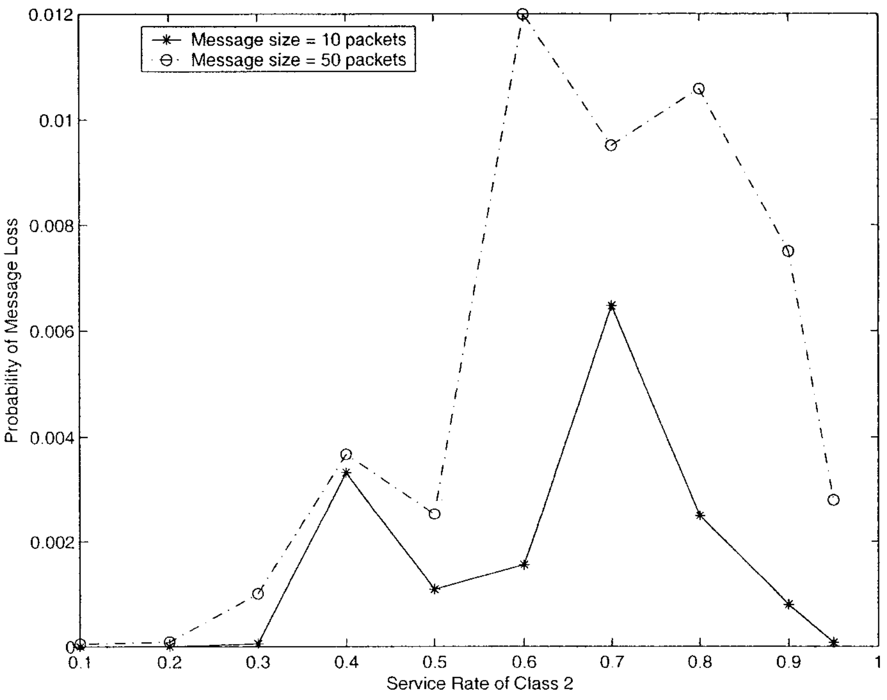

Figure 4.24: Message loss probability of class 2 traffic due to buffer overflow versus its service rate

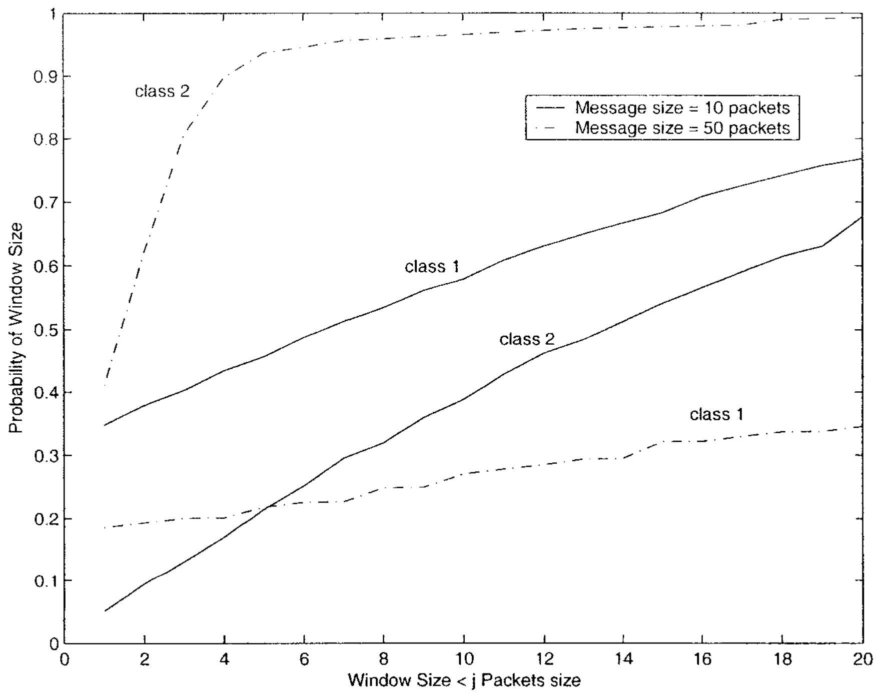

Figure 4.25: Probability distributions of window sizes for class 2 service rate of 0.9

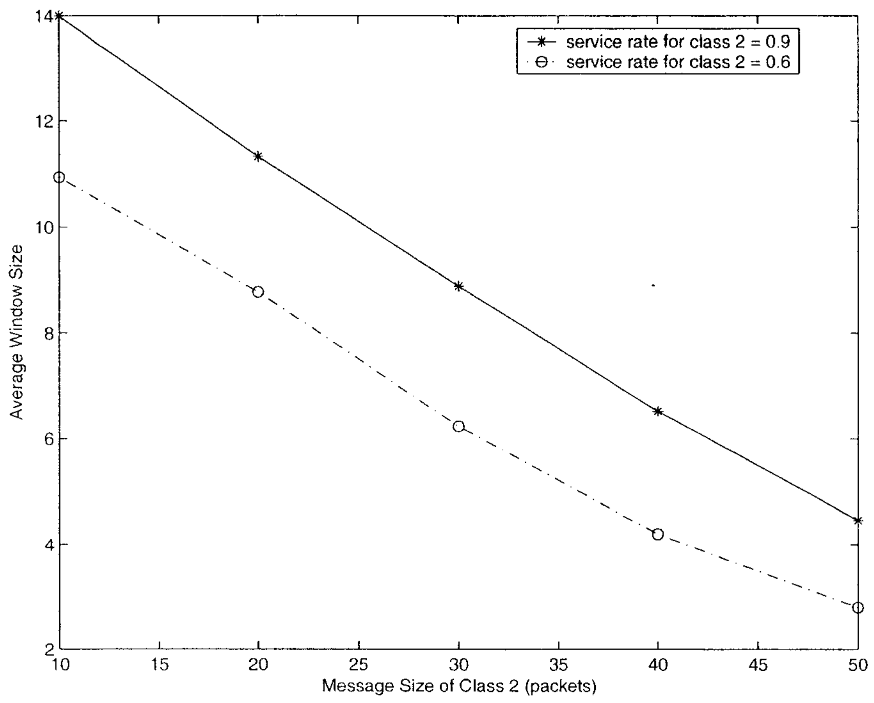

Figure 4.26: Average window size for class 2 traffic versus its message size

Figures 4.27-4.32 present the simulation results for different values of the propagation delay of links A and B. The propagation delays of links A and B have been increased by ten times to 3.6msec and 1.2msec respectively in this simulation.

We observe the simulation results when the class 2 traffic achieves the similar throughput to the previous simulation. Figure 4.27 presents the throughput results for the two classes. Comparison with Figure 4.21 shows that their throughput remains unchanged under higher propagation delay. Comparison of Figure 4.28 and Figure 4.22 shows that the round trip time has increased. Figures 4.29 and 4.30 are similar to Figures 4.23 and 4.24. Figures 4.31 and 4.32 correspond to Figures 4.25 and 4.26 respectively. As may be seen, higher propagation delay results in larger window sizes. However the increase in window sizes is far greater for small message sizes than large message sizes. This again confirms that the large message size provides a better solution to the high-speed TCP than modification of the congestion control algorithm.

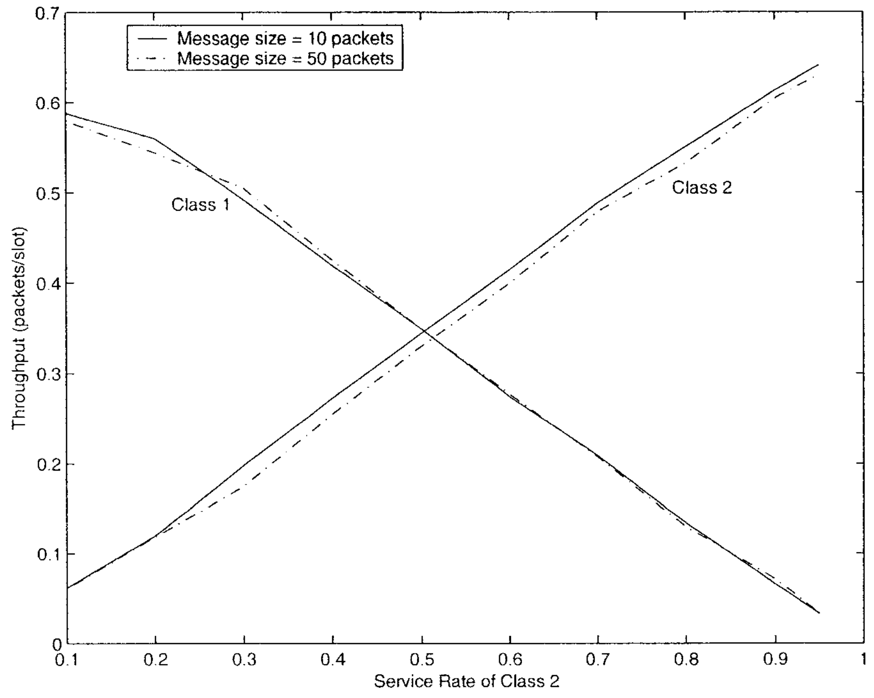

Figure 4.27: Throughtput versus service rate of class 2 traffic

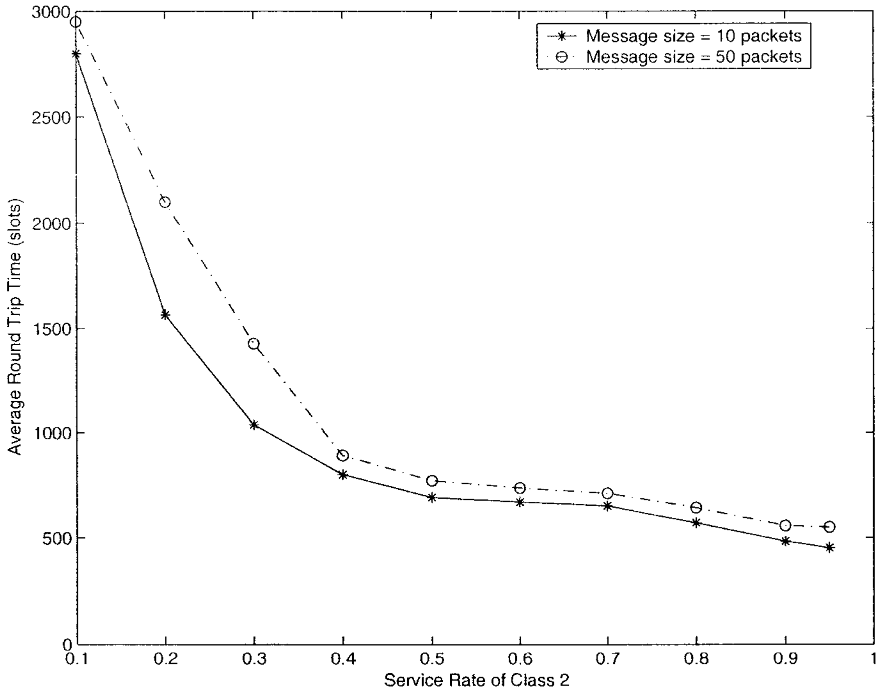

Figure 4.28: Average round trip time of class 2 traffic versus its service rate

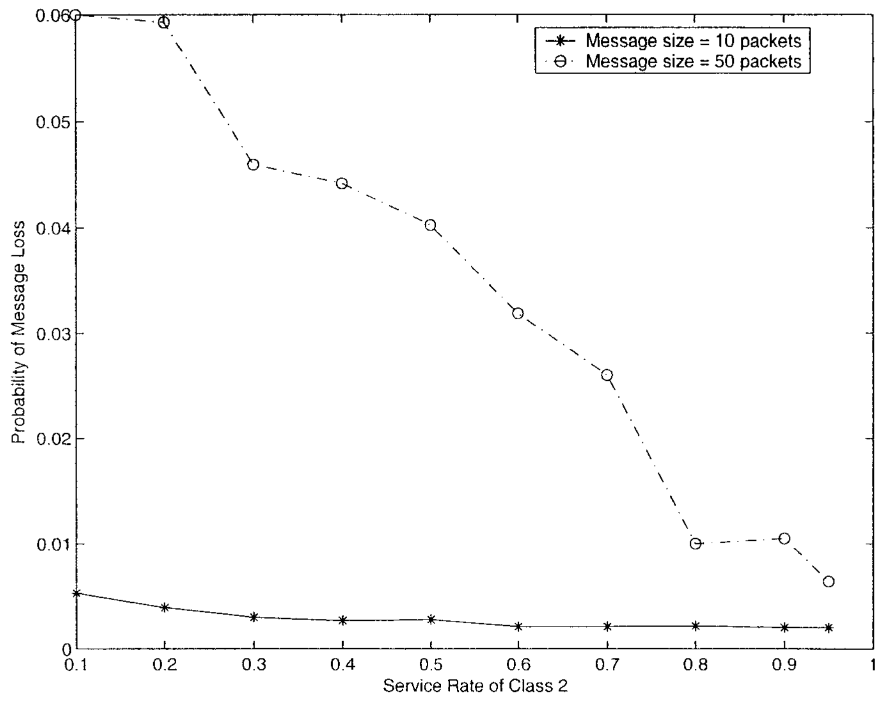

Figure 4.29: Probability of message loss for class 2 traffic versus its service rate

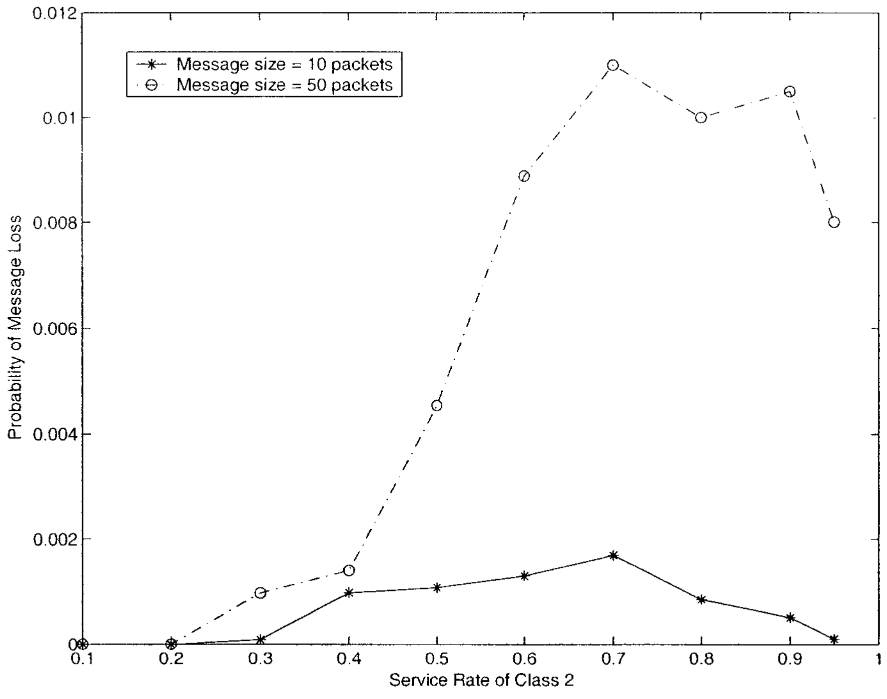

Figure 4.30: Message loss probability of class 2 traffic due to buffer overflow versus its service rate

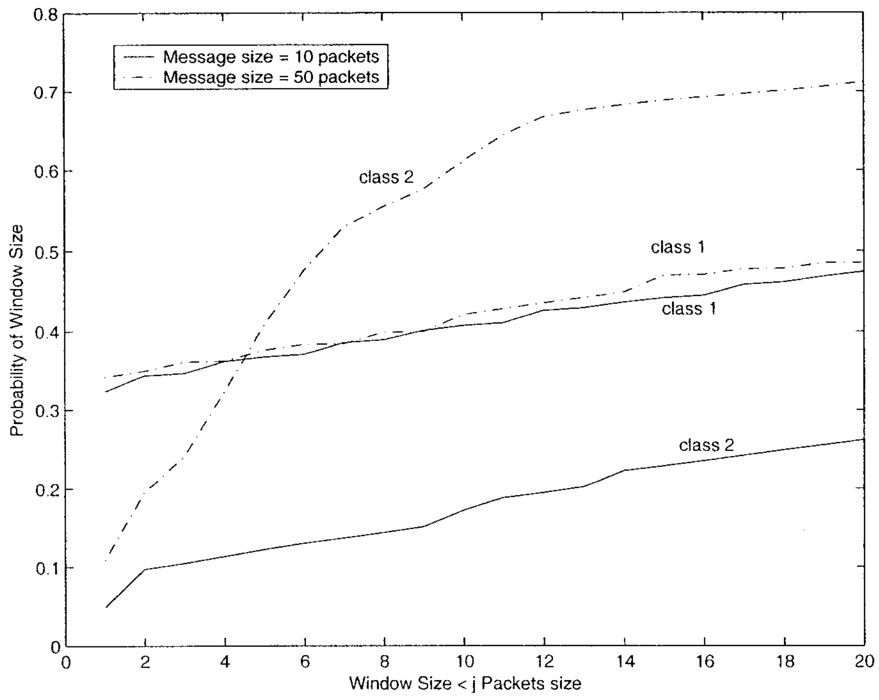

Figure 4.31: Probability distributions of window sizes for class 2 service rate of 0.9

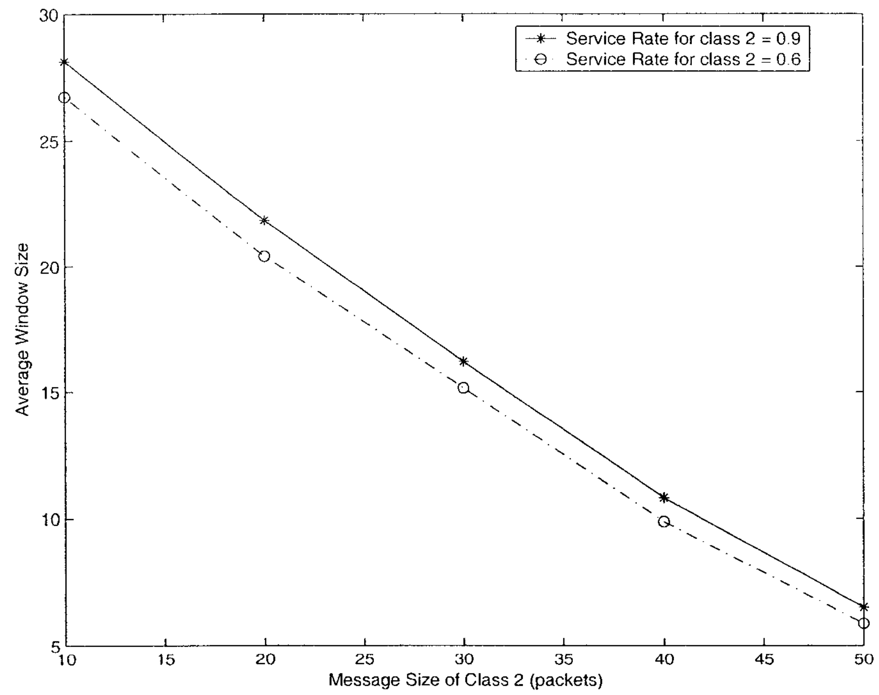

Figure 4.32: Average window size for class 2 traffic versus its message size

Chapter 5

Conclusions and Future work

5.1 Conclusions

Main challenge facing the telecommunication industry is how to integrate the transmission of voice, video, data and other information in a single network. ATM and Internet have emerged as the two competing network architectures for the realization of this integration. In both solutions, traffic will be divided into a few broad service classes according to their characteristics and QoS requirements. The service classes will be treated differently in terms of call admission, routing and bandwidth allocation.

The bandwidth sharing is determined by the scheduling algorithms implemented at the switch and router queues. This thesis has concerned itself with a comprehensive study of different classes of scheduling algorithms. We have studied FIFO, PQ, WFQ and WRR classes of algorithms. The mean and probability distribution of message delay, throughput, message loss probabilities and call blocking probabilities have been performance measures of interest. Simulation has been used as the main tool of the analysis. We have modeled different types of multimedia traffic with Markov On/Off sources. This type of sources is suitable for capturing burstiness and correlation of traffic.

Among the scheduling algorithms studied, FQ and PQ provide the least and the most service differentiation respectively while WFQ and WRR may cover the entire spectrum between the two. WFQ and WRR allow the amount of service differentiation to be controlled through assignment of service weights. This gives an effective way to control the bandwidth allocation to different services in order to meet their QoS requirements.

The implementation of the WFQ class of algorithms is far more complicated than WRR algorithm. This work shows that through appropriate choice of serving weights, the performance of WRR may be close to WFQ not only in mean but as well as in probability distribution of delay. Thus much simpler WRR may be used in place of WFQ in many applications. Our work also shows that the message size may also be used in service differentiation. If the order of service depends on the message size, then mean delay is not independent of service discipline. Thus we suggest that the service classes may also be created based on the message size.

Presently the Internet uses two transport protocols, UDP and TCP, which carry real-time and non-real-time traffic respectively. Voice and video make up real-time traffic while non-real-time traffic consists of data. Up to now the Internet has treated all the TCP traffic in the same manner. However, in the future there will be non-real-time applications that require high throughput. It is proposed to meet this demand either through modification of TCP congestion control algorithm or by increasing the maximum allowed message size. We prefer the latter solution to the former since it keeps the window size and therefore the processing load of routers under control. However, neither of these solutions protects the bandwidth share of low-throughput users. As a solution, we propose to divide the non-real-time traffic into two classes as low and high throughput TCP traffic and queue them separately. The WFQ or WRR scheduling algorithms may be used to protect the bandwidth share of low-throughput users. The simulations show satisfactory results for both types of traffic. It is expected that low-throughput users will use smaller messages compared to high-throughput users. This proposed solution is an example of service differentiation based on message size explained above.

5.2 Future Work

This work may be extended along the following directions,

- Performance of scheduling algorithms in multi-node networks through simulation. This will show if our results hold at a network level.
- Study the performance of scheduling algorithms analytically. The most promising in this aspect will be the analysis of WRR algorithm. Further the results of this analysis will also apply to WFQ algorithm.

References

[1] W. Stallings, "High-Speed Networks and Internets: Performance and Quality of Service", Second Edition, Prentice Hall Inc., 2002.

[2] Harry G. Perros, "An Introduction to ATM Networks", John Wiley & Sons, Ltd. 2002.

[3] X.Song, " Performance Analysis of a Multiplexer with Priority Queues and Correlatedd Arrivals", M.A.Sc. thesis, Dapartmetn of Electrical and Computer Engineering, Concordia university, 2002.

[4] R.Braden, D.Clark, S.Shenker, " Integrated Services in the Internet Architecture: an Overview," RFC 1633, June 1994.

[5] K.Nichols, S.Blake, F.Baker, D.Black, "Definition of the Differentiated Services Field (DS Field) in the Ipv4 and Ipv6 Headers," RFC 2474, December 1998.

[6] J. C. R. Bennett and H. Zhang, "Hierarchical packet fair queueing algorithms," in IEEE/ACM Tran. on Networking, Oct 1997.

[7] A. Demers, S. Keshavt and Scott Shenker, "Analysis and simulation of a fair Queueing Algorithm," in ACM, 1989, pp.1-12.

[8] A.G. Greenberg and N. Madras, " How fair is fair queueing?" in ACM, July 1992.

[9] M. Shreedhar and G. Varghese, " Efficient fair queueing using deficit round robin," in ACM, 1995.

[10] Hyun-Ho Yoon, Hakyong Kim, Changhwan Oh, and Kiseon Kim, " A queue length-based scheduling scheme in ATM networks, " in IEEE, 1999.

[11] Y.Ito, S. Tasaka and Y.Ishibashi, " Variably weighted round robin queueing for core IP routers, " in IEEE, 2002.

[12] M. F. Horng, W. T. Lee, k. R. Lee and Y.H. Kuo, " An Adaptive Approach to Weighted Fair Queue with QoS Enhanced on IP Network, " Proceedings of IEEE Region 10 International Conference on, Aug. 2001.

[13] S. Floyd, S.Ratnasamy and S. Shenker, " Modifying TCP's Congestion Control for High Speeds, " URL http://www.icir.org, May 5, 2002.

[14] S. Floyd, " HighSpeed TCP for Large Congestin Windows," IETF draft, February, 2003.

[15] "Raising the Internet MT, " URL http://www.psc.edu/~mathis/MTU/.

[16] S. Shalunov "TCP Armonk, " URL http://www.internet2.edu/~shalunov/tcpar/ .

[17] S.Keshav, "An engineering approach to computer networks, " Addison-Wesley, 1997.

[18] J. Nagle, "On packet switches with infinite storage, " IEEE Transactions on Communications, April 1987.

[19] A.K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control: the single node case", in Proc. of IEEE INFOCOM, May 1992.

[20] J. C. R. Bennett and H. Zhang, "WF^2Q: worst-case fair weighted fair queueing", in Proceeding of IEEE INFOCOM, Mar, 1996.

[21] S. Golestani, " A self-clocked fair queueing scheme for broadband applications, " in Proceedings of IEEE INFOCOM'94,June 1994, pp. 636-646.

[22] H.M. Chaskar, U. Madhow, " Fair scheduling with tunable latency: A Round Robin approach", IEEE 1999.

[23] S. Golestani, "A stop-and-go queueing framework for congestion management. " In proceedings of ACM SIGCOMM'90, pp. 8-18, Sep. 1990.

[24] C. Kalmanek, H. Kanakia, and S. Keshav, "Rate controlled servers for very high-speed networks, " in IEEE Global Telecommunications Conference , San Diego, California, December 1990, pp. 300.3.1-300.3.9.

[25] H. Zhang and D. Ferrari, " Rate-controlled static priority queueing, " In Proceedings of IEEE INFOCOM'93, Apr. 1993, pp. 227-236.

[26] L. Klenrock, "Queueing Systems, Volume 2: Computer Applications, " Wiley, New York, 1976.

[27] Selvakumaran N. Subramanian, "Traffic Modeling in a Multi-Media Enviroment", M.A.Sc. thesis, Department of Electrical and Computer Engineering, Concordia University, 1996.

[28] K. Sriram and W. Wjott, "Characterizing superposition Arrival Processes in Packet Multiplexer for Voice and Data", IEEE on Selected Areas in Communications, vol.4. No.6, September.1986, pp.833-846.

[29] H. Heffes and D. M. Lucantoni, " A Markov Modulated Characterization of Packetized Voice and Data Traffic and Related Statistical Multiplexer Performance", IEEE on Selected Areas in Communications, vol.4. No.6, September.1986, pp.833-846.

[30] B. Maglaris, et al., "Performance Models of Statistical Multiplexing in Packet Video communications", IEEE Transactions on Communications, vol 36, No.7 1998, pp.834 –844.

[31] M.Mehmet-Ali Asrin, F.Kamoun, " A transient discrete-time queueing analysis of the ATM multiplexer", Elsevier Performance Evaluation, 32, 1998, pp.153 – 183.

[32] R. E. Walpole, P. H. Myers, "Probability and Statistics, " Macmillan, New York, 1985.

[33] V. Jacobson, " Congestion avoidance and control", In proceedings of SiGCOMM' 88, August 1988.

[34] S. Shenker, L. Zhang and D..Clark," Some observation on the Dynamics of a congestion control algorithm", ACM Computer Communication Review, Vol. 20 No.4, Oct. 1990, pp.30-39.

[35] J. Postel, " DoD Standard Transmission Control Protocol", RFC 793.

[36] E. de Souza and D. Algarwal, " A HighSpeed TCP study: Characteristics and Deployment Issues." URL http://www-itg.lbl.gov.

[37] A. Romanow and S. Floyd, " Dynamics of TCP Traffic Over ATM Networks." IEEE Journal on Selected Areas in Communications, May 1995.

[38] D. Stiliadis and A. Varma, "Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms", IEEE/ACM Transactions on Networking, October 1998.

[39] D. Stiliadis and A. Varma, "Rate Proportional Servers: A Design Methodology for Fair Queueing Algorithms", IEEE/ACM Transactions on Networking, April 1998.

Appendix

Pseudo Code for Scheduling Algorithms

In our simulation, the key part is the message processing in the queues. Because the code of total program is big and complex, we only provide the pseudo code of this part in different scheduling algorithms.

1. First In First Out (FIFO)

When a message arrives, it is placed at the end of the queue. The function dequeue() deals with the message at the head of a queue during a slot.

```
Dequeue()
{
    while (( !empty(queue))
    {
            server is idle;
    }
    prclength= the processed message size in one message until past slots
    queue.message.prclength++;
    /*handle a message during a slot  */
    if(queue.message.prclength = message length)
            transmit the whole message;
}
```

2. Priority Queueing (PQ)

- Q= Number of flows(queues)
- *i* = the queue number that is being processed at current slot
- max= the non-empty queue that has max priority in all active queues

```
Dequeue()
{
    if(( !empty(queue[i]))
    {
        if(empty(queue[max] empty)
            server is idle
    }
    else
    {
        if(Queue[i].prclength==0)
        {
            if(empty(queue[max]))
                server is idle
        }
    }
    queue[i].message.prclength++;
    /*processing of  a message during a slot  */
    if(queue[i].message.prclength = message length)
        transmit the whole message;
}
```

3. Fair Queueing (FQ)

- Q= Number of flows(queues)

```
Dequeue()
{
        while (( !empty(queue[i]))
         {
                   if (i<Q)
                        i++;   //point to next queue
                   else
                        server is idle
         }
         queue[i].message.Prclength++;
        /*processing of  a message during a slot  */
        if(queue[i].message.Prclength = message length)
         {
                   transmit the whole message;
                   i++; //point to next queue
         }
}
```

4. Bit Round Fair Queueing (BRFQ)

When a message arrives, it is placed in the current queue as well as the corresponding virtual queue. Caculating the message' virtual finish time from the processing virtual queues. Virtual time is got from the function Operate():

```
Operate()
{
    ActiveVirtualQueue= queue numbers that are not empty at current slot;
    virtual_time= Virtualtime+ 1/ActiveVirtualQueue;
    transmit 1/ActiveVirtualQueue bytes messages from the virtual queue;
}
```

Recording message' virtual finish time when a message is placed in the queue:

```
    queue[i].virtual_finish_time = virtual_time + messagelength;
```

Selecting the queue with minimum virtual finish time message and transmiting the message at the head of this queue:

```
selectqueue()
{
    min.virtual_finish_time = Min(queue[i].virtual_finish time);
    i = queue(min.virtual_finish_time) ;
    queue_selected= i ;
}
Dequeue()
{
    if(( !empty(queue[i]))
  {
        if((queue[i].message.prclength ==0) /* no message has been processed
                                                        message in the past slots */

        {
            if(!selectqueue())
             /* find the queue with minimum virtual time message */
                 server is idle;
        }
    }
    else
    {
        if(!selectqueue())
         /* find the queue that has the message minimum virtual finish time */
            server is idle
    }
    queue(queue_selected).message.prclength++;
    if(queue(queue_selected).message.prclength = message Length)
        transmit the whole message;
}
```

5. Weighted Fair Queueing (WFQ)

The main difference between BRFQ and WFQ is the calculation of the virtual finish time:

```
    queue[i].virtual_finish_time = virtual_time +Messagelength/queue[i].Weight;
```

6. Worst-case Weighted Fair Queueing (WFQ)

The main difference between WFQ and WFQ is in the selectqueue():

```
selectqueue()
{
        for(i=0;i< num_queues: i++)
        if ( !empty(queue[i])) && queue[i].virtual_start_time> virtual_time
           && queue[i].virtual_finish_time < minimum_finish_time)
           {
                minimum_finish_time = queue(i).virtual_finish_time;
                queue_selected = i :
            }
}
```

7. Worst-case Weighted Fair Queueing+ (WFQ+)

The queue selection and message transmission procedure of WFQ+ are same as of WFQ. The different point is the calculation of virtual time. When a message arrive in queue[*i*],

```
    queue[i].virtual_start_time = virtual time;
    queue[i].virtual_finish_time = virtual time + messagelength/queue[i].weight
    min.virtual_start_time = Min(virtual_start_time);
    virtual_time = max(min.virtual_start_time, virtual_time);
```

8. Self_clocked Fair Queueing (SCFQ)

This algorithm doesn't need to simulate virtual queueing system at the same time that it is processing the messages in the queues. When a message arrives at queue[*i*]:

```
queue[i].virtual_finish_time = max(queue[i].lastmessage.virtual_finish_time,virtual
time) + messagelength/queue[i].weight;
virtual time = queue[i].virtual_finish_time;
```

9. Weighted Round Robin (WRR)

```
dequeue()
{
   while (empty(queue[i]))
   {
      if (i< num_queues)
      {
            queue[i].usedweight=0;/* initial the couter recording used weight;
             i++; // check next queue
      }
      else
      {
            server is idle;
      }
      if(queue[i].usedweight<queue[i].weight)  /* the weight allows this queue to
                                                   transmit a message*/
           transmit a packet in the message;
      if(queue(queue_selected).message.prclength = message length)
      {
           {
            transmit a message;
            queue[i].usedweight++; /*record used weight for this queue*/
        }
        if(queue[i].usedweight>=queue[i].weight)/*used up all weight*/
        {
            queue[i].usedweigt=0;//initial used weight for the given queue
             move to next queue;
        }
    }
```

10. Deficit Weighted Round Robin (DWRR)

Initialize the DeficitCounter for each queue:

```
for(i=0;i< num_queues: i++)
{
    queue[i].deficitcounter =0;
}
```

The function Enqueue(*i*) places newly arriving messages into the correct queue and manages what is known as the ActiveList. The ActiveList is maintained to avoid examining empty queues. The ActiveList contains a list of the queue indices that contain at least one message. Whenever a message is placed in a previously empty queue, the index for the queue is added to the end of the ActiveList by the function InsertActiveList(*i*). Similarly, whenever a queue becomes empty, the index for the queue is removed from the ActiveList by the function RemoveFromActiveList(*i*).

Enqueue(*i*)

```
{
    i = the index of the queue that will hold the new message
    if(!ExistsInactiveList(i))    /*if i not in ActiveList*/
        InsertActiveList(i);
        Queue(i).DefictCounter = 0;
    Enqueue message to queue(i); /*place message at end of queue */
}
```

Whenever an index is at the head of the of the ActiveList, the function Dequeue() transmits up to queue[*i*].DeficitCounter + queue[*i*].Quantum worth of bytes from the queue. If queue[*i*] still has messages to send at the end of the service round, the function InsertActive(*i*) moves the index *i* to the end of the ActiveList. However, if queue[*i*] is empty at the end of the service round, the queue[*i*]. DeficitCounter is set to zero and the funtion RemoveActiveList(*i*) removes the index *i* from the ActiveList.

```
Dequeue()
{
  while(true)
  {
     if (ActiveList !empty)
        i= the index at the head of the ActiveList;
        queue[i].DeficitCounter = queue[i].DeficitCounter + queue[i].Quantum;
        while(queue[i].DeficitCounter > 0 and !empty(queue(i)))
        {
           messagelength= length(head(queue(i));
        if (messagelength <= queue[$i$].DeficitCounter)
        {
            transmit message at the head of queue[$i$];
            queue[$i$].DeficitCounter = queue[$i$].DeficitCounter – messagelengh;
        }
        else
            exit;
      if(empty(queue[$i$])
      {
           queue[$i$].DeficitCounter = 0;
           RemoveFromAcitveList($i$);
      }
      else
           InsertActiveList($i$);
}
```