A NEW PRIORITY BASED PACKET SCHEDULER WITH DEADLINE CONSIDERATIONS

ORAL GÖKGÖL

IŞIK UNIVERSITY

2006

A NEW PRIORITY BASED PACKET SCHEDULER WITH DEADLINE CONSIDERATIONS

A Thesis
Presented To The Institute Of Science And Engineering
of IŞIK University
In Partial Fulfillment Of The Requirements
For The Degree Of
Master Of Science
In
The Department Of Computer Engineering

by
ORAL GÖKGÖL

IŞIK UNIVERSITY
2006

Approved for the University Committee on Graduate Studies.

.....
Prof. Dr. Hüsnü A. Erbay
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

.....
Prof. Dr. Selahattin Kuru
Head of the
Computer Engineering
Department

This is to certify that I have read this thesis and that, in my opinion, it is fully adequate in scope and quality as a thesis for the degree of Master of Science.

.....
Dr. Tamer Dağ
Supervisor

Examining Committee Members

.....	_____
.....	_____
.....	_____

ABSTRACT

A NEW PRIORITY BASED PACKET SCHEDULER WITH DEADLINE CONSIDERATIONS

ORAL GÖKGÖL

Quality of Services (QoS) issues have become a focus point of research on Next Generation Networks (NGNs). In order to supply the various QoS requirements for different kinds of applications, new packet scheduling policies need to be developed. This thesis focuses on the packet scheduling policies in computer networks. An effort to develop a packet scheduling algorithm that supplies QoS in computer networks is an interesting topic. This thesis introduces two new packet schedulers which try to integrate an important QoS parameter (the delay) with the classical schedulers. The two sets of algorithms introduced; Static Priority with Deadline Considerations (SPD) and Dynamic Priority with Deadline Considerations (DPD); not only simplify the complexity and overhead of the classical Earliest Deadline First (EDF) or Static Priority (SP) algorithms, but also provide a better level of QoS based on the simulations conducted.

Key words: packet scheduling, QoS, Static Priority, Earliest Deadline First, packet loss, deadline

ÖZET

PAKETLERİN ANLAMSIZ HALE GELME ZAMANLARI GÖZ ÖNÜNDE BULUNDURULARAK DİZAYN EDİLEN YENİ BİR ÖNCELİĞE DAYALI PAKET GÖNDERİM ALGORİTMASI

ORAL GÖKGÖL

Servis kalitesi (QoS) konuları Next Generation networklerde araştırma yapmak için önemli konulardır. QoS gereksinimlerini farlı tiplerdeki uygulamalarda sağlamak için yeni Paket Gönderimi Algoritmaları geliştirilmesi gerekmektedir. Bu tezde bilgisayar ağlarındaki kuyruklarda paket gönderimi algoritma uygulamaları üzerine odaklanılmıştır. Paket gönderimi algoritmaları bilgisayar ağlarındaki performans açısından büyük önem arz eder. Ağlardaki Servis Kalitesini (QOS) garanti etmek için üretilen paket gönderimi algoritmaları günümüzde ilginçliğini kaybetmeyen bir konudur. Bu tez kuyruklarda paket gönderimini kontrol eden yeni iki algoritma üzerinedir. Bu algoritmalar bilinen paket kontrol algoritmalarına yeni bir QOS parametresi olan gecikmeyi (delay) eklemektedir. Bu tezde iki yeni algoritma tanıtılacak; Sabit öncelikli algoritma - paketlerin anlamsız hale gelmeden önce gönderilmeleri düşünülerek (SPD) ve Değişken Öncelikli Algoritma - paketlerin anlamsız hale gelmeden önce gönderilmeleri düşünülerek (DPD); bu algoritmalar sadece algoritmaların karmaşıklığını azaltmakla kalmayıp ayrıca klasik algoritmlardan; paketlerin anlamsız hale gelme zamanlarına göre gönderilmesi (EDF) ve Sabit Öncelikli paketler (SP) algoritmalarına göre daha iyi sonuç veriyor.

Key words: kuyruklarda paket gönderimi sırası, QoS, Servis Kalitesi, Sabit Öncelikli Paketler Algoritması, paketlerin son gönderilme zamanlarına göre işlenmesi algoritması, paket kaybı

ACKNOWLEDGEMENTS

I would like to express my thanks to Dr. Tamer Dağ for his comments, help and supervision on the topic, and for understanding my faults; and also I would like to thank all people that support me by giving intelligent ideas, and physcological support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

With the increase of the Internet's popularity, attention and research areas are concentrating on emerging integrated services packet-switched networks to simultaneously support applications with diverse performance and QoS requirements and traffic characteristics. There has been a lot of research on designing new scheduling algorithms which would support the requirements of such applications by providing them different QoS levels [1, 2, and 3].

The scheduling algorithms can be classified according to their complexity and how they behave to different kinds of applications. Networks in which QoS requirements are thoroughly satisfied can be best described as best-effort networks [4]. Best effort network designs treat all packets coming from different applications as equally important. Such networks work well if there is enough CPU, memory, and bandwidth, as packets traversing through the network can be handled immediately after their arrival. However, this is not always achievable due to the lack of resources.

In this thesis, two sets of priority based scheduling algorithms which consider the remaining deadlines of the packets are introduced. The first algorithm, Static Priority with Deadline Considerations (SPD) schedules the packets based on their assigned priorities and reduces the overhead of sorting that can be seen in SP and EDF algorithms. The second algorithm, Dynamic Priority with Deadline Considerations (DPD) schedules the packets based on their assigned priorities, but at the same time modifies the priorities based on the remaining deadlines. DPD also has a reduced complexity like SPD by introducing the concept of degree sorting as will be described in the following sections.

The rest of the thesis is organized as follows: Chapter two gives background material on QoS Networks and QoS parameters. In this chapter, QoS definition and why we need it, UDP and TCP protocols, QoS differentiation on multimedia applications, QoS requirements and parameters for different applications are described.

Chapter three explains what a packet scheduler is and introduces different packet scheduling algorithms used on networks. It gives background information on packet scheduling and algorithms design, and some widely used packet scheduling algorithms.

Chapter four introduces the first proposed packet scheduling algorithm, Static Priority with Deadline Considerations (SPD). In this chapter, SPD packet scheduling algorithm is introduced with the experimental environment. SPD simulation design, simulation results and various SPD algorithms' simulation results comparisons are presented.

Chapter five introduces the next proposed packet scheduling algorithm, Dynamic Priority with Deadline Considerations (DPD). In this chapter, DPD packet scheduling algorithm is introduced with the experimental environment. DPD simulation design, simulation results and various DPD algorithms' simulation result comparisons are presented.

Chapter six compares the SPD and DPD algorithms in a simulation environment. In this chapter, the DPD and SPD algorithms are compared with each other and with classical algorithms based on their loss ratios and complexities. Finally, the thesis concludes with the conclusions made in Chapter seven.

CHAPTER 2

QOS AND QOS PARAMETERS

The capabilities of a network to provide resource guarantees and service differentiation are defined as the Quality of Service (QoS). In this chapter, QoS overview, parameters and some approaches on how to provide QoS in the networks will be discussed.

2.1 QoS Overview

Initially, Internet has been designed by using the datagram model which is about dividing the data into packets and transferring them over the network one by one. In Figure 1.1, datagram model is introduced with its objectives. Datagram model is good for file transfers, remote connections and e-mail packets. However, with the increase of internet's popularity and network needs such as multimedia application flows, this model needs to be improved.

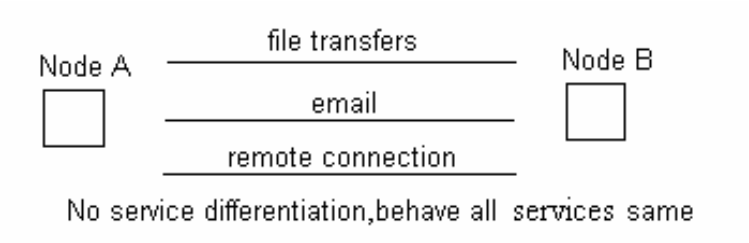

Figure 2.1 Datagram Model Approach

TCP protocol supplies the packets the possibility of retransmit them, when they are lost on the way. The important data flows use the TCP protocol. For example, email and file transfer packets are transmitted using the TCP protocol because if some of the packets are lost on the flow, it makes the information useless. UDP

protocol is somewhat less secure and is used for multimedia applications. In UDP protocol, there is no guarantee if a packet is successfully transmitted. Thus, if many multimedia packets are lost on the flow, the sound or video will be useless.

File transfer and e-mail packets require only arrival to destination without any error. They are not sensitive to delay or jitter parameters. If the data that is flowing on the network arrives without error, there would be no problem. However, web sites now provide their users online videos and music by using the internet. Also, users now can make phone calls by using the internet. With these developments on the internet technology, a new topic has evolved which is service differentiation; to differentiate the packets according to their special characteristics.

For example, if someone is listening music from a web site, and if the arriving music packets' delay becomes larger, then the quality of the music will be low. In other words, the music cannot be perceived correctly by the user. These difficulties have evolved the definition of Quality of Service (QoS), which is controlling the network flow by considering each application characteristics.

QoS brings many advantages to both users and network administrators. By using QoS, users can get more quality services according to their demands. Network administrators can attract many users by supplying good quality services. QoS allows network administrators to use their existing resources in efficient manner. By considering QoS parameters, critical applications may receive high-quality service. As a result, network providers can have better control over their networks and improve customer satisfaction.

There are no infinite network resources. If there are infinite resources on networks, then we do not need to define the Quality of Services terms. Every application may get the best desired resource within infinite resources. QoS allows real-time programs such as VoIP applications to make the most efficient use of network bandwidth. Because QoS provides some level of guarantee for available

network resources, it gives a shared network a level of service similar to that of a dedicated network[1].

2.2 QoS Advantages

As can be seen in the previous subsection, with the increase of demand on networks, the QoS consideration is required for the applications.

QoS has a key role to control the application flows on a network. Multimedia applications require more bandwidth and less delay requirements than data applications. Applying QoS requirements on a network can supply the required bandwidth and worst-case delay for multimedia or other key applications. Thus, QoS can help network administrators to manage the traffic flow.

The decision which applications can get the better QoS is a critical problem. Therefore every network needs changes from the other. If a network is using mostly the multimedia applications, then QoS parameters will be controlled by packet schedulers to get the best effort on multimedia flows on the network. Also with the QoS, there are equipments on the network that are need to be developed and some equipment that are newly designed and integrated into the networks such as some components on routers to measure loss ratios or delays.

QoS gives administrators control over their networks, improves the user experience, and reduces costs by the efficient use of existing resources, which can delay the need for expansion or make it less expensive.

2.3 QoS Parameters

The QoS parameters are the QoS requirements of applications and are important for understanding the QoS term. The bandwidth, delay, jitter and loss rate constitute the QoS parameters.

[1] Dedicated network is a private network that is used only by a single user. Shared network is the network that is used by more than one users.

2.3.1 Minimum Bandwidth

The bandwidth is the required transfer rate for an application. Minimum bandwidth is the minimum amount of bandwidth required by an application flow. For example voice packets require at least 8 kbps, and video packets require at least 128 kbps to be transferred over the network.

Bandwidth allocation is enforced by packet scheduling algorithms on the end nodes. The Weighted Fair Queuing (WFQ) scheduler is able to provide minimum bandwidth guarantee over small time intervals as can be seen in Chapter three.

2.3.2 Latency (Delay)

Latency is the delay of a packet on the networks. In other words, it is the time to transfer the packet across a network. Packets may be held up in queues, on slow links, or because of congestion. In the networks that are congested, the delay will be higher for packets. Although delays that are over 100 ms are disruptive to voice packets, email packets are not sensitive to delay.

The delay requirement can be speci ed as the average delay or worst-case delay. Average delay is the mean value of delays that the packets on the flow experience. Worst-case delay is the highest delay that a packet experiences in the flow. The delay that a packet experiences has three components: propagation delay, transmission delay and queuing delay.

Propagation delay is caused because of the distance of the nodes. It is the time required for data to travel from transmission point to destination. Transmission delay is the time to send a packet into the link that is caused by the node that sends the packets. Queuing delay is the delay for packets while waiting on the queue to be served. It will be also called as the *average waiting time on queue* for packets. Transmission and queuing delay can be converted to a bandwidth requirement because they can be controlled on a network. Propagation delay cannot be controlled or decreased because the speed of light is constant.

2.3.3 Jitter

Jitter is simply defined as the variance of delay. In other words, jitter is the variance of delay between the same kinds of packets that flows on a network. A delay-jitter requirement is the maximum difference between the largest and smallest delays that packets experience.

Higher levels of jitter are more likely to occur on slow or heavily congested networks. Increasing use of QoS control mechanisms on higher speed links such as 100 Mbit Ethernet will decrease the jitter and reduce the jitter related problems.

2.3.4 Loss rate

Loss rate is the percentage of lost packets to the total number of transmitted packets. Packet losses on a network are often caused by congestion which can happen when many different flows want to use network resources. These losses can be prevented by allocating sufficient bandwidth to packets and queues for traffic ows. When a shared network resource is busy then the queues will be filled quickly with the incoming packets on routers. This will increase the delays of packets on queue and cause packet losses. Packet scheduling algorithms can guarantee a minimum packet loss ratio by controlling the queues.

2.4 QoS Requirements

Packets that are traveling on a network can be simply categorized in three groups as data, voice and video packets. Table 2.1 shows the QoS requirements of these packet types; email as a data application, VoIP as a real time voice application and videoconference as a real time video application.

In Table 2.1, the word high or low shows how much the application is sensitive to the corresponding QoS parameter. For example, from the table we see that VoIP

packets are susceptible to low loss ratios and their bandwidth usage is low. And, email packets' sensitiveness to loss ratios is high.

Table 2.1 The stringent QoS requirements for different types

Application	Loss Rate	Delay	Jitter	Bandwidth
Email	High	Low	Low	Low
VoIP	Low	High	High	Low
Videoconference	Low	High	High	High

Data, Voice and Video are the some type of application flows on QoS networks. Data applications like email or web access applications are smooth and burst. In real life, data application flows on networks are much higher than video and voice application flows. In other words, data application flow density is higher than other applications because of the people needs. Data packets are sensitive to losses. However, data packets use TCP protocol, so if they are lost on the way, then they can be retransmitted. The delay is not an important parameter for data packets compared to video and voice packets. The data packets in queues can be served with different times and so jitter parameter may be high for data packets.

Real time voice and video applications are much sensitive to the change on QoS parameters compared to data applications. Video packets require more bandwidth compared to voice packets. Both video and voice packets are jitter and delay sensitive. Because of these requirements, the UDP protocol is used for video and voice packets. UDP protocol is the fastest way to pass application flows across a network. However, the disadvantage of this, if many packets are lost on the way, then they could not be retransmitted and this brings lower quality voice or videos on the network.

2.5 QoS Architectures and Models

There are some QoS technologies that are used in networks, Integrated Services, Differentiated Services and Multiprotocol Label Switching architectures. These models will be discussed in this section.

2.5.1 Integrated Services

Integrated services model is based on the resource reservation system that is used to provide QoS on the small networks and somehow on the Internet. Resource reservation system is a protocol to request a reservation of a resource before using it from the admission control. Admission control checks if sufficient resources are available and assigns the resource to the application. Integrated services model is designed by the Integrated Service Working Group (ISWG-IETF) of IETF (Internet Engineering Task Force). To control the resource reservation step, IETF has standardized the RSVP (Resource Reservation Setup Protocol).

In this model, the applications which want to send their data over the network must reserve the required resources and provide the required information such as destination and needed resources. Consider an example in Figure 2.2. An ambulance that will be sent from the hospital will take an injured patient and bring him back to the hospital. Here the hospital is the application. The ambulance is the packet that will be transferred on the network which is the road. The hospital first calls the police to ask them clear the road for the ambulance. In other words, the hospital wants to *reserve the road*. The police then reserve the road for the hospital's ambulance and then the ambulance is sent by the hospital to arrive to node that is the injured man.

Integrated Services model guarantee worst-case delay and jitter needs and thus is applicable for video packets or real time applications that require high bandwidth and small delay requirements. However it is not applicable for web access and e-mail packets which do not require low delay.

ISWG-IETF defines two services for Integrated Services Model; guaranteed service and controlled load service. The guaranteed service guarantees the worst case

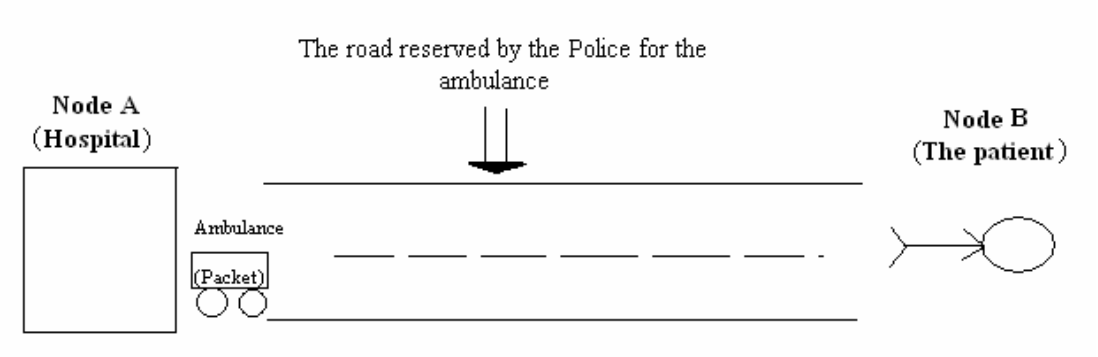

Figure 2.2 Integrated Services Model Example

delay for the applications that define their exact delay bounds. Guaranteed Service makes the reservation of resources for the worst case delay. However, it leads to a less efficient utilization of the available bandwidth. Controlled load service agrees only to carry a certain traffic volume in the lightly loaded network.

2.5.2 Differentiated Services

While Integrated Services try to guarantee QoS requirements on small networks, differentiated service is a method of trying to guarantee QoS requirements on larger networks such as the Internet. Differentiated Services model allows network providers to allocate their different levels of service to different users. Traffic management or bandwidth control mechanisms that treat different users differently range from simple Weighted Fair Queuing (WFQ) to RSVP.

Differentiated Services Code Point (DSCP) is an integer value encoded in the IP header to define the packet priorities. The packets' priorities (DSCP) are attached by the edge of the network service providers according to the service level agreement between service provider and the customer. Therefore, more money given to the network providers will result in better quality on the service.

Before packets enter one of the Differentiated Services routers, they are initially classified by the sender. The sender sets the packet's type of service field (DSCP), in the IP header according to the class of the data, in such a way that the better classes get higher priorities.

The advantage of Differentiated Services is that all the policing and classifying is done at the edges of the Differentiated Service routers. This means routers only deal with their job of queuing and serving packets, and do not care about the complexities of collecting payment or enforcing agreements.

When there is not enough network resources Differentiated Services model decides which packets to delay and which packets to drop. Therefore, the packets that are sent by the users who give less money to the service provider will always be lost, if there are higher priority packets on the network.

2.5.3 MPLS (Multiprotocol Label Switching)

MPLS simply specifies mechanisms to manage traffic flows between different machines, and flows between different applications. It is designed by the Internet Engineering Task Force (IETF). It provides bandwidth management and QoS for various protocols such as IP, ATM, and Frame Relay by increasing the speed of network traffic flow by inserting information on the packets that is about a specific destination path. This decreases the heavy tasks of routers. Because routers don't need to find the address for the next node that the packet is sent to.

The most important benefit of MPLS is that it allows service providers to deliver new services that cannot be supported by standard IP routing techniques. MPLS maps IP addresses to simple, fixed-length labels used by different packet-forwarding technologies. MPLS model is based on building label switched paths (LSPs) across networks and then forwarding IP packets across the network by these paths. By adding LSP labels to the packets' structures, it is possible to remove the overhead of checking packets at every network device on the link.

CHAPTER 3

PACKET SCHEDULING

In this chapter, packet scheduling algorithms and their purpose will be introduced in detail.

3.1 Introduction

A packet scheduler is a running algorithm on a router which decides which packet will be served next. Consider a router that is receiving packets from the network link every t1 seconds, and routes the packets to their destinations every t2 seconds. The situation when t1 < t2 results in some difficulties. In this situation, the packets arrive at the router faster than the service time. As a result, many packets that arrive at the router will be dropped or discarded. This problem has brought the idea *queuing the packets* on routers.

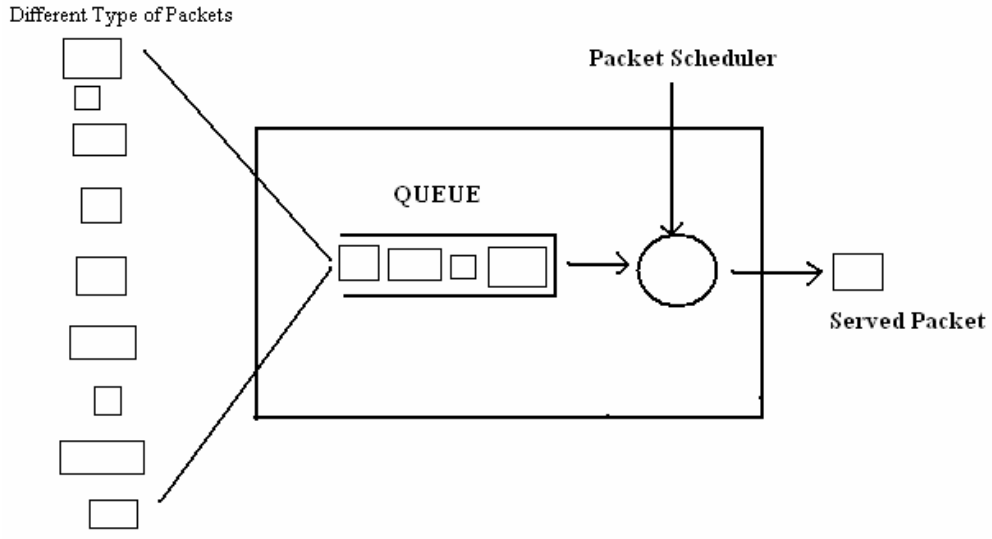

Figure 3.1 Queue and Packet Scheduler Definition

In Figure 3.1, a queue and packet scheduler definition is introduced. A queue or buffer is a part of the router storage that keeps the all incoming packets from the network link temporarily not to loose them. Then, how can the packets are served from the queue and in which order? A packet scheduler or a packet scheduling algorithm manages the queue and sends the packets to their destinations by looking up the packets' characteristics and decides which packet will be served first.

All Packet Schedulers should run by considering of QoS requirements of the packets. However, the simplest packet scheduling algorithm FIFO (First in First Out) only serves the packets according to their order of arrival into the queue. The FIFO is the first algorithm and doesn't supply QoS requirements for the network flow. The types of packet scheduling algorithms will be discussed in section 3.3.

Packet Scheduling is important step because it may guarantee resource reservations in networks. With the definition of QoS, packet scheduling has become a favorite topic in networks.

3.2 Design of Packet Schedulers

There are many possibilities how to design scheduling algorithms. Before describing packet scheduler algorithms' characteristics, we will examine some key issues that are important to design a packet scheduler in the following subsections.

3.2.1 Work Conserving Schedulers

A work conserving scheduler is a simple approach to design a packet scheduling algorithm. A scheduler is work conserving if it is idle only if the queue is empty. In a detailed way, in work conserving schedulers, if there are no packets in the queue to process, then there will be no job to do. Some of the simple packet scheduling algorithms such as EDF and FIFO run based on this scheme.

Work Conserving Schedulers run in two states. If there is a packet on the queue then the scheduler enters to “busy” state, and serves the packets on the queue one by one. If there is no packet to process on the queue then the scheduler enters “idle” state, and waits for a packet to come to the queue. Work Conserving Schedulers use the network bandwidth with an efficient way and do not waste bandwidth.

3.2.2 Non-Work Conserving Schedulers

The non-work conserving scheduling algorithms may be idle even though there are packets to be served in the queue. The goal of non-work conserving schedulers is to decrease the jitter for packets which is a parameter of QoS.

For example, when there is only one packet (e.g. an email packet) in the queue, non-work conserving scheduler waits for other packets to arrive the queue. It means it changes its state to idle. It delays the email packet in the queue and doesn’t serve it. When a high delay sensitive packet (e.g. a videoconference packet) arrives to the queue the scheduler goes from idle state to busy state and processes that packet. Lower jitter with non-work conserving schedulers is achieved using the fact that packet becomes eligible for transmission only after a short period from the departure of the previous packet from the same ow.

The average waiting times (delay) on the queue will be increased in non-work conserving schedulers because of the delayed packets on the queue. However, non-work conserving schedulers reduce the jitter of packets. The other disadvantage is bandwidth is wasted in non-work conserving schedulers.

3.2.3 Priority

Priority is a number that will define the precedence of packets. The small numbers like priority 1 or 2 will show the high precedence packets, and large numbers like priority 9 or 10 will show the low precedence packets. Higher priority packets are always more important than the lower priority packets.

The priority is an important parameter when designing a packet scheduler. Without a priority definition, the service of packets will be a cumbersome. Many scheduling algorithms are based on the priority scheme and they differ in how priorities are assigned to the packets. A scheduler based on priority scheme always serves packets with highest priorities from the queue. The lower priority packets will be left waiting in the queue and this causes the starvation of packets. The delay bounds for higher priority packets will be lower while for others it will be higher.

3.3 Packet Scheduling Algorithms

A packet scheduler is the most important QoS functional component. A packet scheduler takes packets from a queue and serves them in order and packet schedulers differ in that way.

There are many trade-offs, when a packet scheduling algorithm is designed. For example, if the loss ratio is wanted to restrain with the low values, then the complexity of the algorithm increases. If designer wants to decrease the delays of some packets in the queue then the casualties for other packets will increase. This causes the different packet scheduling algorithm designs that is specialized for different kinds of networks.

The efficiency of a packet scheduling algorithm can be calculated with many formulas. We can define a simple formula as the following;

$$\text{The efficiency of an alg.} = \frac{\text{Throughput}}{\text{Avg. Waiting Times in the queue} * \text{Complexity of the Algorithm}}$$

In this formula, *throughput* is the amount of data successfully transferred in a specific amount of time, *average waiting time* is the average delay of packets on the queue, and *complexity of the algorithm* is the time and designing complexities of the algorithm. When the throughput increases, the efficiency of the algorithm also increases. If delay times of packets on the queue increases, then the efficiency of the

algorithm decreases. Furthermore, if the complexity of the algorithm increases, this will also lead to less efficient algorithm.

Scheduling algorithms can be roughly divided into three categories: fair queuing, deadline based and rate based scheduling. In the fair queuing approach, the share of bandwidth by a packets' ow is represented by a weight which is a real number. In the fair queuing approach the bandwidth is allocated according to the weights of flows. If a flow cannot use all of its allocated bandwidth, then the remaining bandwidth is shared between other flows according to their weights. With fair queuing, it is guaranteed for a ow to get its entitled bandwidth and maybe more if there is any unused bandwidth left. Fair queuing is able to provide a delay bound and is commonly used in QoS capable networks.

The deadline based scheduling is based on the Earliest Deadline First algorithm (EDF). In an EDF scheduler, each packet has a deadline time which denotes the duration of time in which a packet needs to be transmitted to its destination. The scheduler simply transmits the packets based on their deadline times. A packet with the smallest deadline will be served first. The advantage of the algorithm is that delay and bandwidth parameters are decoupled (delay bounds can be independent on bandwidth allocation). For example, a ow reserving a small amount of bandwidth can still obtain a small delay bound. However, the admission control is much more complex. In general, two tests must be performed. First the total allocated bandwidth must not exceed the link capacity. Second a *schedulabili*ty test must be performed to ensure that deadlines will not be exceeded.

Rate based scheduling is a principle that is used for constructing different work conserving and non-work conserving scheduling disciplines. This type of scheduler has two components which are a regulator and a scheduler. The regulator determines the eligibility time for each packet. Once a packet becomes eligible, the scheduler may select this packet for transmission. Arriving traffic is shaped by the regulator before coming to the scheduler. A packet may be delayed at the regulator. Different number of regulators may be used such as token bucket regulators, peak rate

regulators and jitter regulators. The scheduler can also be FCFS (FIFO), fair queuing or EDF.

The following subsection discusses various packet scheduling algorithms in detail.

3.3.1 EDF (Earliest Deadline First) Packet Scheduling Algorithm

In EDF packet scheduling algorithm, the packets leave the queue based on their deadline times. In this approach, the packet that has the earliest deadline has the highest priority in the queue. This algorithm can also be considered a static priority algorithm with the deadline times determine the packet priorities.

In Figure 3.2, EDF scheduling algorithm is introduced in details. In the figure, the packets that came to the queue are sorted according to their deadline times and the packet which has the least deadline time is served first. Because EDF requires a sorting algorithm in queue according to packet deadlines, this function brings some complexity on the algorithm which decreases the algorithm efficiency. EDF algorithm can supply some of the QoS requirements. For example, because the voice packets' deadline times are usually lower than data packets, they will be processed first, which as a result supplies lower delay and jitter for voice packets. However, it also results in some disadvantages on data packets, that is data packets will always be lost if there are some voice packets on the queue.

3.3.2 SP (Static Priority)

In the SP algorithm, the packets in the queue have priorities that determine which packet will be processed next. The higher priority packets always will be processed first. As a result, higher priority packets' average waiting times in the queue will be smaller than lower priority packets.

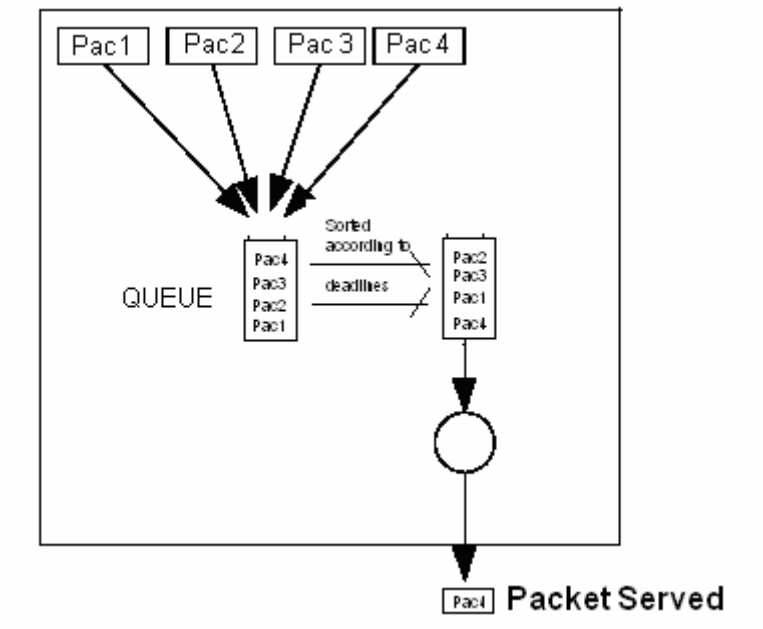

Figure 3.2 EDF Scheduling Algorithm

In Figure 3.3, there are four packets and they arrive to the queue in the order 3,1,4,5 by assigned priority values. Then, the packets are sorted based on their priorities and the highest priority packet which is 1 will be served first.

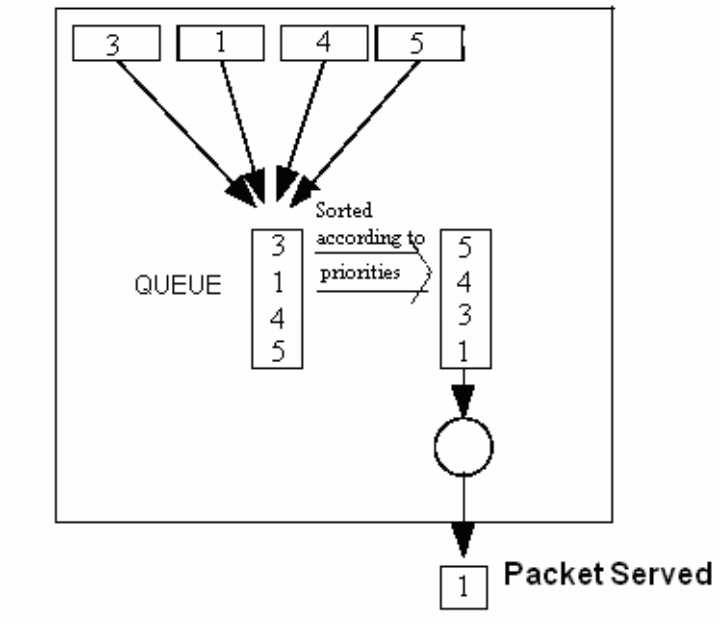

Figure 3.3 SP Scheduling Algorithm

Because EDF and SP algorithms sort the packets on the queue, the complexity of SP will be same as EDF. However, SP algorithm somewhat can supply QOS requirements. For example, we can assign higher priorities to voice packets, so the voice packets will wait for a short amount of time in the queue. As a result, lower delay values will be supplied for voice packets in the queue.

3.3.3 FIFO (First in First out)

FIFO is the simplest algorithm. There is no need to sort packets in the queue and this cause low complexity on the algorithm. The only process is that a packet which comes into the queue first will be processed and transmitted first. Consider an example in Figure 3.4. Packets are settled in the queue based on their arrival times and the first arrived packet *pac1* is served first.

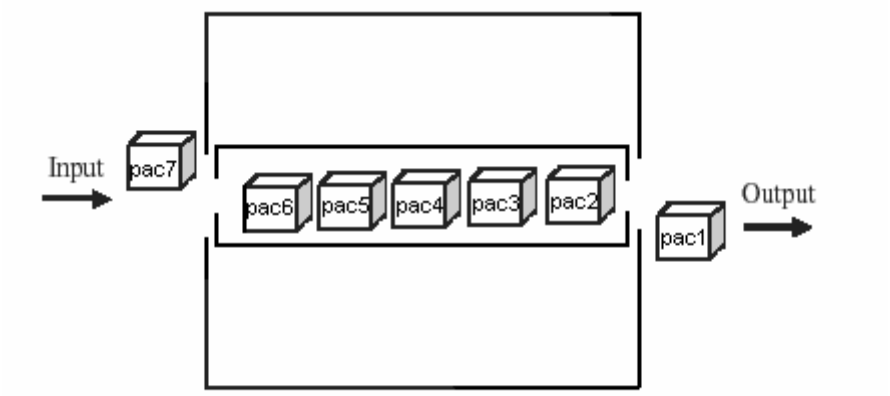

Figure 3.4 FIFO Scheduling Algorithm Flowchart

However, the most important disadvantage of this algorithm is that it cannot provide the QoS requirements. The algorithm cannot differentiate the packets and it cannot decide if a packet is a voice, video or data packet. The algorithm only serves the first packet in the queue. The algorithm cannot control jitter or delay for video packets or real time applications.

3.3.4 Rotating Priority Queues (RPQ+)

As the complexity of the packet scheduling algorithm EDF is high, a new packet scheduling algorithm RPQ+ has been designed which approximates to EDF algorithm. In RPQ+ scheduling algorithm, there is not only one queue but some set of queues that run in the FIFO scheme. Initially, queues are assigned with priorities. And these priorities periodically change to reduce the number of waiting packets in the queues. The goal of RPQ+ algorithm is to supply worst-case delay guarantees.

RPQ scheduler has the following key characteristics:

- The operations of RPQ+ algorithm are independent from waiting packets on the queues.
- This scheduler supplies worst-case delay guarantees like EDF.
- RPQ+ leads to the higher network utilization.

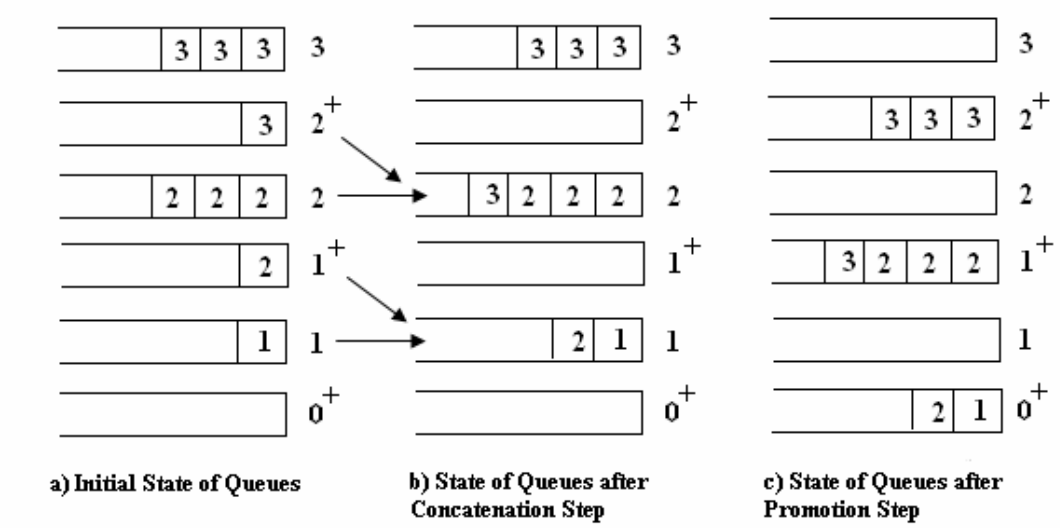

Figure 3.5 RPQ^{+} Scheduling Algorithm Overview

In the RPQ+ scheduler there are 2P queues that are sorted according to their priorities where P is the number of priorities assigned to each queue. Every packet that comes to a router is stored at the end of Pth queue. In Figure 3.5, there are 6 queues ordered according to their priorities. For P = 3, the packets that come to queue will be added to the end of the queue with priority 3. Every Δt time interval,

the queues will be processed in two steps; concatenation and promotion steps. In the concatenation step, all P^+ priority queue elements will be added at the end of P priority queues ($2^+ \rightarrow 2$ and $1^+ \rightarrow 1$). Then, in the promotion step all queues priorities will be promoted, such as the priority queue 3 will be 2^+, 2^+ will be 2, and so on. The packets are always served from the highest priority queue which is 0^+.

As a result, RPQ+ scheduling algorithm does not use the sorting of packets technique as EDF does. This brings RPQ+ algorithm less complexity than EDF algorithm.

3.3.5 Weight Fair Queue (WFQ)

Weight Fair Queue (WFQ) packet scheduler is based on a set of queues that have a weight ratio. WFQ scheduling algorithm is based on the QoS parameter bandwidth. All queues are assigned a weight w_i according to the used network policy. As can be seen in Figure 3.6, there are three queues A, B, C having the weights w_1, w_2, and w_3. Queues A, B, and C receive the following ratios of available bandwidth: w_1/TW, w_2/TW, and w_3/TW respectively where TW is the sum of the queue weights.

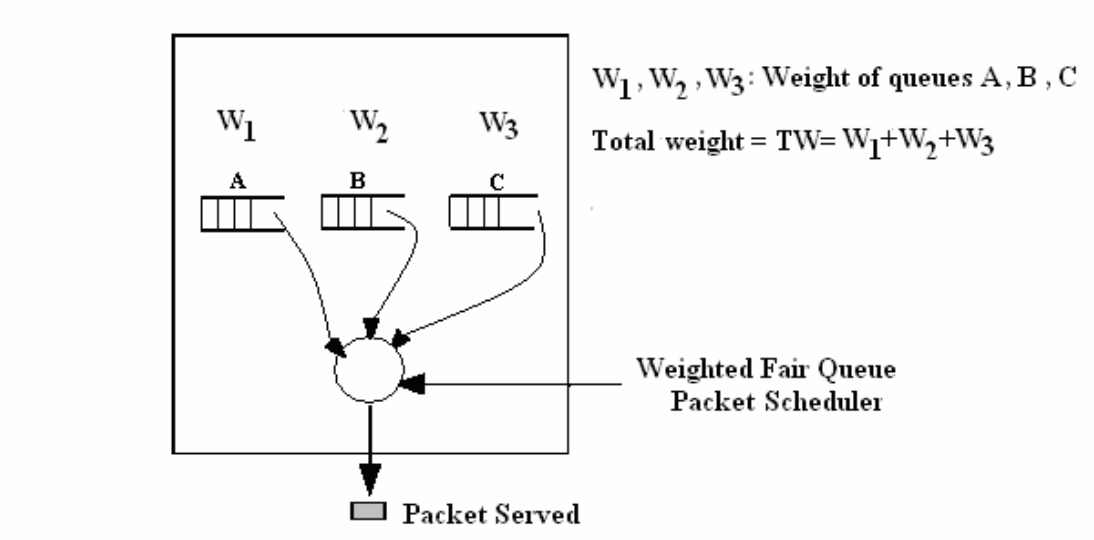

Figure 3.6 WFQ Packet Scheduling Algorithm

If some queues waste their reserved bandwidth, this will not affect the other queues. The queue that reserves the highest bandwidth will result in the best delay bound for its packets. However, the bandwidth and delay parameters are not directly related for packet types. Some applications may require low bandwidth and low delay and some require higher bandwidth and does not require low delay. In the first case WFQ will allocate high bandwidth to these applications in order to guarantee the low delay bound. In the second case, WFQ still has to allocate high bandwidth to supply application needs. In WFQ, applications will satisfy their delay needs but sometimes they will get more than their needs. This mismatch can lead to low bandwidth utilization. In real life, the goal of WFQ is to provide network link sharing among the groups instead of concerning individual flows. It schedules packets which belong to aggregated flows, groups, and classes. Delay is the less considered parameter in this situation.

3.3.6 Round-Robin

In this algorithm, it is considered there are N queues. Packets are classified according to their priorities or types and sent to the corresponding queues that are assigned a priority 0 to (n-1). Then, packets in higher priority queues get serviced first. Then queues are serviced in order from priority 0 to (n-1). This algorithm can't offer bandwidth or delay guarantees. Packets can wait in a queue, while empty queues are checked for servicing. Also, the algorithm is insensitive to packet size (inherently unfair).

3.4 Summary

Packet scheduling is a very important functionality which can provide the QoS requirements for the networks. The algorithms differ according their complexity, and supplying QoS parameters. There is no optimum algorithm for QoS networks. The success of the algorithms changes according to the requirements and needs of the networks.

In the remaining chapters, the proposed scheduling algorithms will be discussed.

CHAPTER 4

STATIC PRIORITY WITH DEADLINE CONSIDERATIONS (SPD)

In this chapter, our first proposed packet scheduling algorithm the Static Priority with Deadline Considerations will be introduced.

4.1 SPD Scheduling Algorithm Overview

The Static Priority with Deadline Considerations (SPD) resembles to the SP algorithm. The packets are sorted according to their priorities. However, before a packet is served, the remaining deadline of the packet is checked and the packet is discarded if the packet has no remaining deadline line which is different from classical SP algorithm. By this way, a packet without any use when it reaches its destination is eliminated when its deadline expires. Discarding a packet with an expired deadline also helps to reduce the unnecessary network traffic and allows other applications to use the limited network resources.

The major difference of SPD from SP is that, SPD introduces a new topic which is *degree sorting*. Instead of sorting every element in the buffer, partial sorting is done and the processing overhead of SP is reduced. The simulation results show that by partial sorting, we can achieve the similar levels of packet losses.

Consider a network buffer of size N and buffer occupancy of Q packets. Assume that every packet is assigned a priority. The classical SP algorithm sorts all the packets in the buffer and transmits the packet with the highest priority.

Figure 4.1 illustrates an example of serving a packet under the classical SP packet scheduler. The packet whose priority is the highest, thus the packet with priority 1 is taken in front of the buffer and then served.

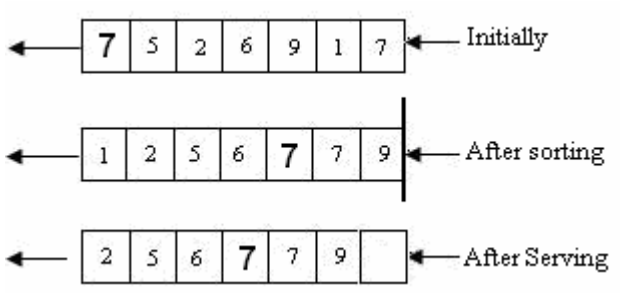

Figure 4.1 SPD-FULL packet scheduling example

In a degree k, SPD packet scheduler which will be denoted as SPD-k, only the first k packets in the buffer are sorted, and the packet with the highest priority among the first k packets is transmitted.

Figure 4.2 illustrates an example of serving a packet under SPD-2, thus order 2 packet scheduling. The scheduler sorts only the first 2 packets in the buffer and serves the packet whose priority is 5, as that packet has the highest priority among the first 2 packets in the buffer.

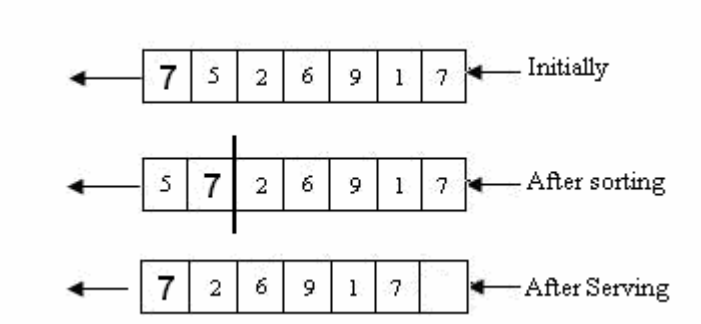

Figure 4.2 SPD-2 packet scheduling example

In the same manner, Figure 4.3 illustrates an example of serving a packet under SPD-4, thus order 4 packet scheduling. The scheduler sorts only the first 4 packets in

the buffer and serves the packet whose priority is 2, as that packet has the highest priority among the first 4 packets in the buffer.

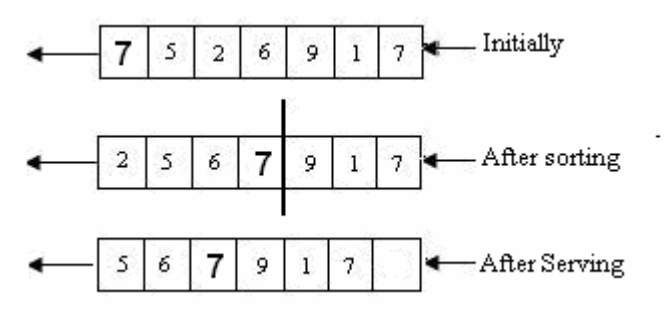

Figure 4.3 SPD-4 packet scheduling example

As a result, we can understand the differences between the classical SP algorithm and SPD-k algorithms. The classical SP algorithm sorts all the queue elements according to their priorities and serves the packet with the highest priority. However SPD-k algorithms sort the first k elements of queue and serve the highest priority packets between first k elements. This can result in many advantages which will be described in section 4.3 in the simulation results.

4.2. Experimental environment and SPD simulation design

In this section SPD simulation design, experimental environment and program variables will be introduced.

4.2.1 SPD simulation design

The UML diagram of algorithm in Figure 4.4 shows the designed simulation that simulate SPD algorithm in MATLAB. The programming language MATLAB is used for its easiness and graphical interface.

Figure 4.4 shows the running simulation algorithm step by step. In the initial stage, the program creates packets for different applications with different priorities such as data, voice, video and other types of application packets.The program assigns

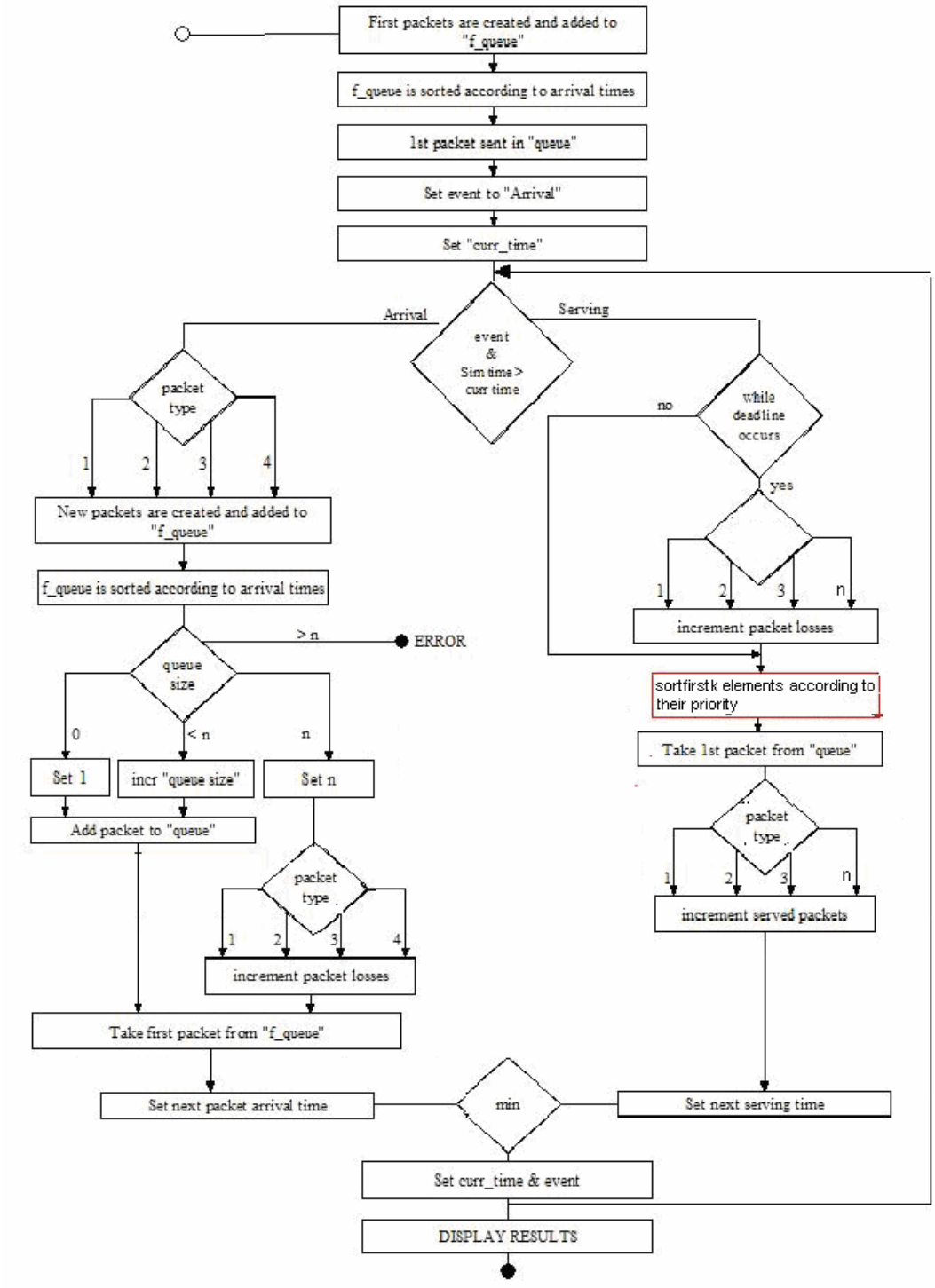

Figure 4.4: The UML Diagram of SPD Simulation

their *properties* and adds them to the *f_queue*. *f_queue* is the queue to store the created packets. P*roperties* are the priorities of packets- which is randomly selected between 1-10, the arrival and deadline times - which are selected by using Poisson

Distribution values and packet types which is the initial priority of the packet. Packet types are selected as the priorities of the packet. For example, if we create a packet which has the priority 1, then the packet type is also 1 to differentiate it from the other packets. Thus, because priorities are changing in the range 1-10 for a packet, there are 10 types of packets in the simulation. After sorting the f_queue according to arrival times, the first packet is selected from the *f_queue*. Then, the program selects *packet arrival* as an event at the beginning and starts to the simulation.

In packet arrival event, since the first packet is selected, the next incoming packet is created and added to f_queue. Then, f_queue is sorted again. This operation is done for next packet arrival steps which will be calculated further. After that, the first selected packet from f_queue is added to the end of the *real queue*. f_queue is only for keeping the generated packets; it is not a real queue on switch. It can be thought as a source that produces packets that are needed in the simulation. It is required to simulate the environment. Then, if there is enough space in the queue, queue size will be incremented by one and the new packet will be added to the end of the queue. Queue will not be sorted in the arrival process. And, the time for a new packet arrival time is decided in arrival event of the simulation. Sorting the f_queue as explained above, the second packet appears at the head of the f_queue array. By looking its arrival time, the next packet arrival time is updated.

When the program exits from the packet arrival event, it compares the next packet arrival time and next packet serving time to decide which event will be done next. The smallest value of next packet arrival time and next packet serving time is selected as the next event. Consequently, the virtual clock is set to choose the next event and program continues to process the selected event. The other type of event is *packet serving*.

In packet serving event; initially, the simulation takes the first packet in the queue and checks if its deadline time is expired or not. If the packet's deadline time is expired, simulation discards it from the queue and increases the total lost packet number. This is required not to deal with the expired packets. Then, if there is still a

packet in queue, the queue will be sorted according to packet priorities. Here, an important strategy is used about sorting the queue. The selected sorting degree here which we call it "k" results in different algorithms like SPD-2, SPD-4, SPD 7, and SPD-FULL. For example, if SPD-k algorithm will be used in which k=2, that means SPD-2 algorithm will be simulated, the sorting is done between the first and second packets on the queue only. Other packets are not considered. In the same manner for k=4, in SPD-4 algorithm, only the first 4 packets are sorted on the queue based on their priorities. Also in SPD-FULL algorithm, all of the queue elements are sorted based on their priorities. Then, the first packet is taken from the queue and simulation serves this packet by changing the simulation statistics. At the end, the program checks other packet arrival times and adds the transmission time to calculate next packet service time. By comparing next packet arrival time and service time, the simulation continues. When the simulation is finished, a statistical view of the program shows the results.

4.2.2 Experimental environment and program variables

For the simulations, the buffer sizes capacities are selected as 10, 20, 30, 40 and 50 packets. That means that the queue can accept 10,20,30,40 or 50 packets by the defined value *queue_size*. The arrival process is defined to be exponential and the departure process is deterministic. It is assumed that all the packets have equal length and 100 bits for simplicity. The packets are assigned priorities between 1 and 10 taken from a uniform distribution. The remaining deadlines of the packets are assigned randomly taken from an exponential distribution with the average value 8.

The deadline times' exponential mean value that is the difference between service and arrival times of packets is taken as *8 unit seconds(BND_DELAY)*. The time in the simulation is not a real time. *Unit seconds* expresses the time according to the simulation. The value *8 unit seconds* is found experimentally. If the mean deadline time is selected smaller than that value then the packet losses based on deadline will extremely increase. If the mean deadline time is selected larger than that value then the packet losses according to packets deadline will approach to zero.

In the simulation, the transmission rate is taken as 10 kbps and the packet size for all packets is taken as 100 bits as expressed before. The packets in the simulation are arriving to the queue every 0.0102 seconds on the average because the arrival rate λ is selected 98 in the simulation and packets are served every 0.01 seconds on the average because departure rate $\mu = 100$ is selected on the simulation. That means every second, 48 packets may arrive to the queue and 50 packets may be served from the queue in average (not all of them served some of them lost).

Simulation time is selected as 20000 seconds. That is not a real time that is the time in the simulation in other words it is a relative time. In 20000 seconds, approximately 2 million packets are processed in the simulation.

In the following section, the simulation results will be described according to these environments and defined variables.

4.3 SPD Simulation Results

In SPD algorithm simulations, the processing overhead of SPD-2 is observed lower than the processing overhead of SPD-FULL as SPD-2 only checks and sorts the first two packets in the router queue. Also, SPD-k algorithms can be thought in the same manner.

Figure 4.5 and Table 4.1 presents the total packet loss ratios for different order SPD packet schedulers as a function of buffer sizes for SPD-2, SPD-4, SPD-7 and SPD-FULL. It is observed that, the total induced packet losses for SPD-2 is the best. As the order for SPD-k increases, the packet loss ratios increase as well. The reason why SPD-2 has a better performance can be explained, when we differentiate the packets losses into its two components; Packet losses due to buffer overflows and packet losses due to deadline violations. While the packet losses due to buffer overflows decrease as the buffer size increases, the packet losses due to deadline violations might be inversely affected for increased buffer sizes.

Figure 4.6 and Table 4.2 presents the packet loss ratios due to buffer overflows as a function of buffer size. As the buffer size increases, the packet losses decrease significantly. For this kind of losses, full sort, thus SPD-FULL has the best performance as expected and as the order of SPD decreases the induced packet losses increase.

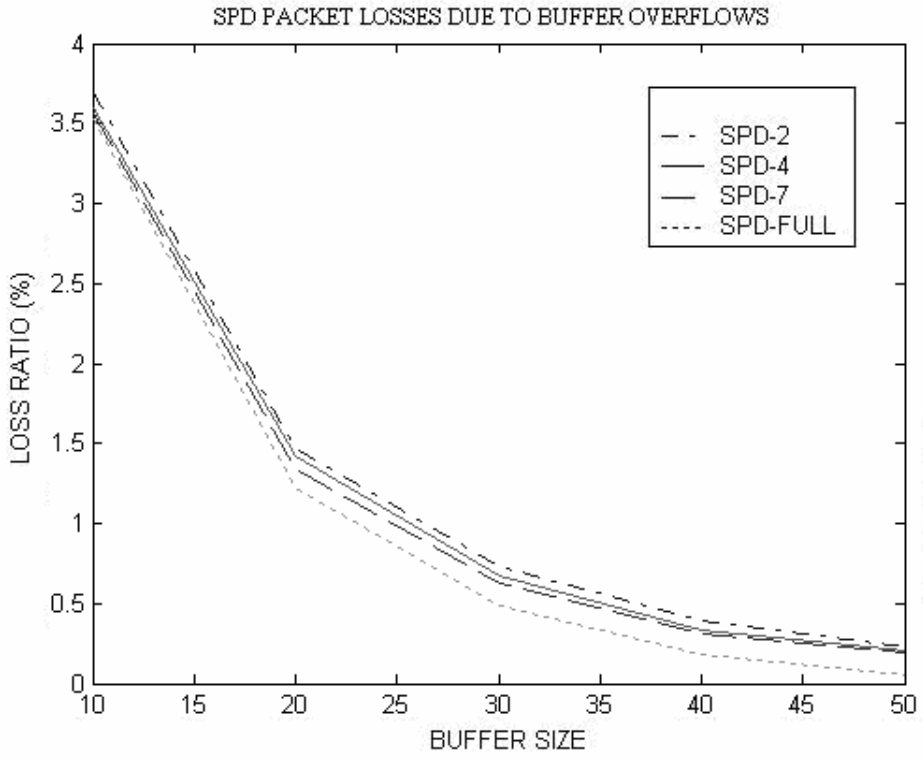

Figure 4.5 Total Packet Loss Ratio Graph in SPD algorithms

Table 4.1 Total Packet Loss Ratios in SPD algorithms

ALGORITHM \ QUEUE SIZE	10	20	30	40	50
SPD-2	3.8537	1.6804	0.9937	0.6768	0.5602
SPD-4	3.9064	1.8097	1.1140	0.8077	0.7134
SPD-7	4.0102	1.9400	1.2797	1.0130	0.9063
SPD-FULL	4.0211	2.1091	1.6020	1.4272	1.3613

When the buffer size is small, the majority of the packet losses depend on buffer overflows. However, as the buffer size increases, the packet losses mainly depend on losses due to deadline violations as can be seen in Figure 4.7 and Table 4.3. SPD-2

has the best performance for cell losses due to deadline violations. Because SPD-2 only allows a packet at the head of the buffer move one step back, if its priority is low. Thus, the reason of having better packet loss ratios for the overall performance of SPD-2 is due to the fact that it performs better in terms of deadline violations.

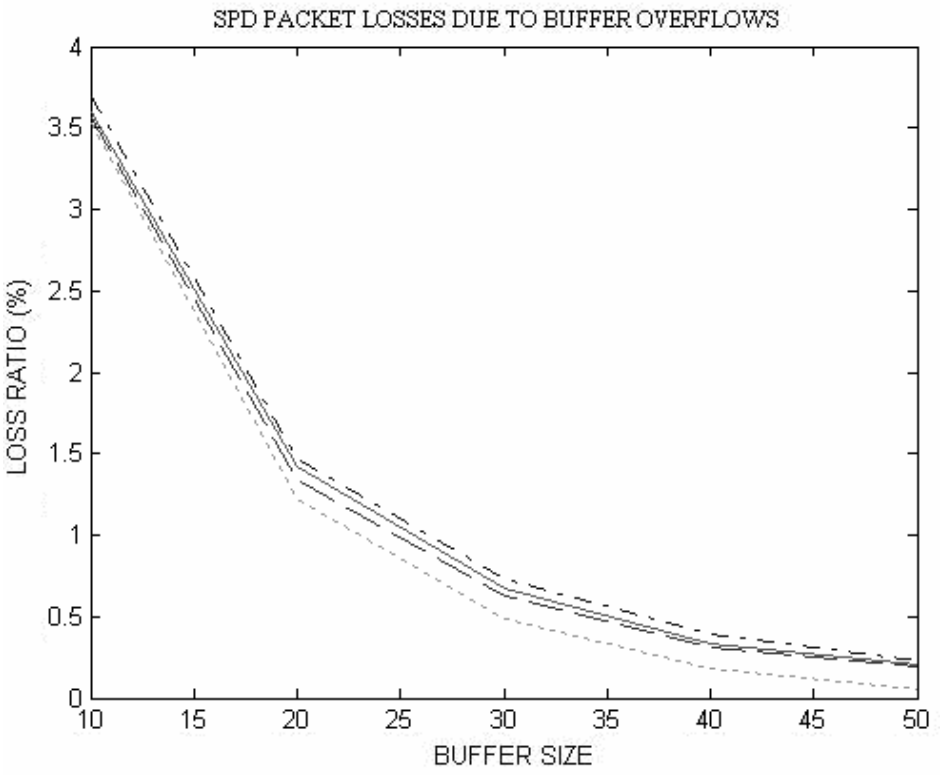

Figure 4.6 Packet Losses Graph due to Buffer Overflows for SPD

Table 4.2 Packet Losses due to Buffer Overflows for SPD

ALGORITHM \ QUEUE SIZE	10	20	30	40	50
SPD-2	3.6933	1.4635	0.7338	0.3879	0.2317
SPD-4	3.5947	1.4169	0.6717	0.3318	0.2004
SPD-7	3.5586	1.3445	0.6213	0.3123	0.1918
SPD-FULL	3.5223	1.2185	0.4797	0.1854	0.0475

Figure 4.8 illustrates how SPD packet scheduler behaves to packets with different priorities. For example for SPD-2 packet scheduler, the total packet loss ratio for

packets with priority of 1 is 0.02%, and for SPD-FULL, it is about 0.007%. For SPD-2 packet scheduler, the total loss ratio of packets with priority 10 is 0.31%, and for SPD-FULL, it is about 0.91%. For SPD-FULL packet scheduler, 65% of the total packets lost have priority 10. Thus, as the degree increases in an SPD algorithm, the lower priority packets are neglected more.

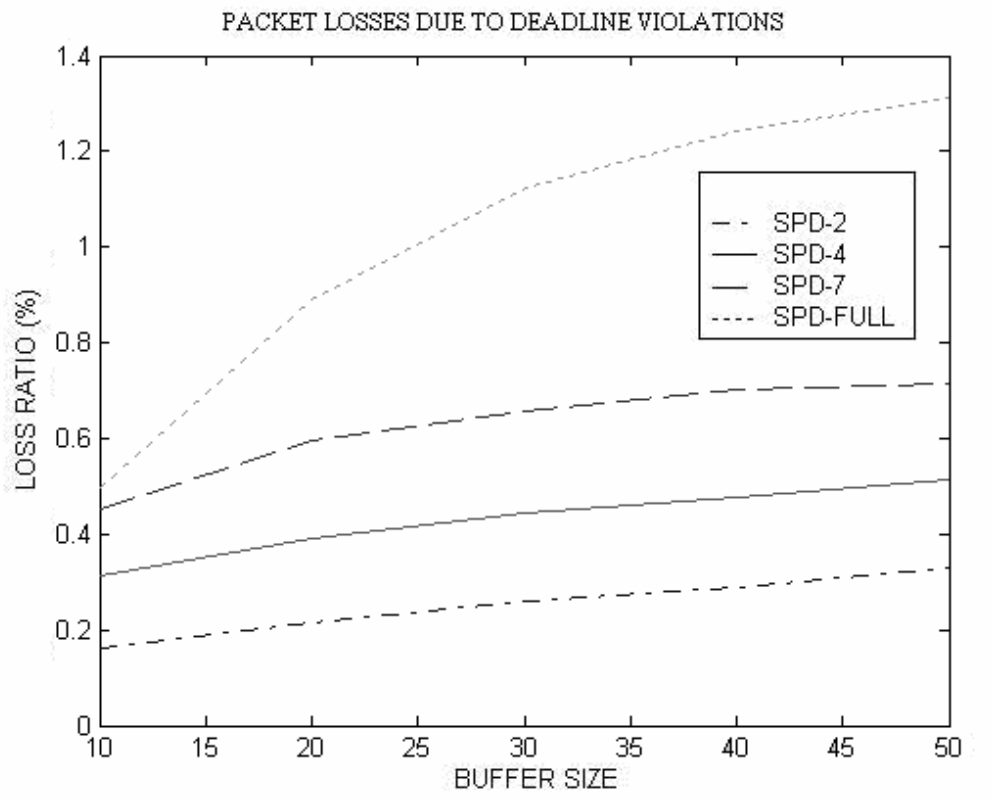

Figure 4.7 Packet Losses Graph due to Deadline Violations for SPD

Table 4.3 Packet Losses due to Deadline Violations for SPD

ALGORITHM \ QUEUE SIZE	10	20	30	40	50
SPD-2	0.1604	0.2169	0.2599	0.2890	0.3286
SPD-4	0.3117	0.3928	0.4423	0.4759	0.5130
SPD-7	0.4517	0.5955	0.6584	0.7008	0.7145
SPD-FULL	0.4988	0.8906	1.1224	1.2418	1.3139

Figure 4.9 represents the average waiting times (delay) of packets in the buffer as a function of buffer size for various SPD-k packet schedulers. It is observed that SPD-FULL has lower average waiting times, as SPD-FULL schedules high priority packets as fast as possible.

Figure 4.10 and Table 4.4 presents the average waiting times for packets for different priorities. As the degree of the SPD packet scheduler increases, the waiting times for higher priority packets decrease, and lower priority packets increase.

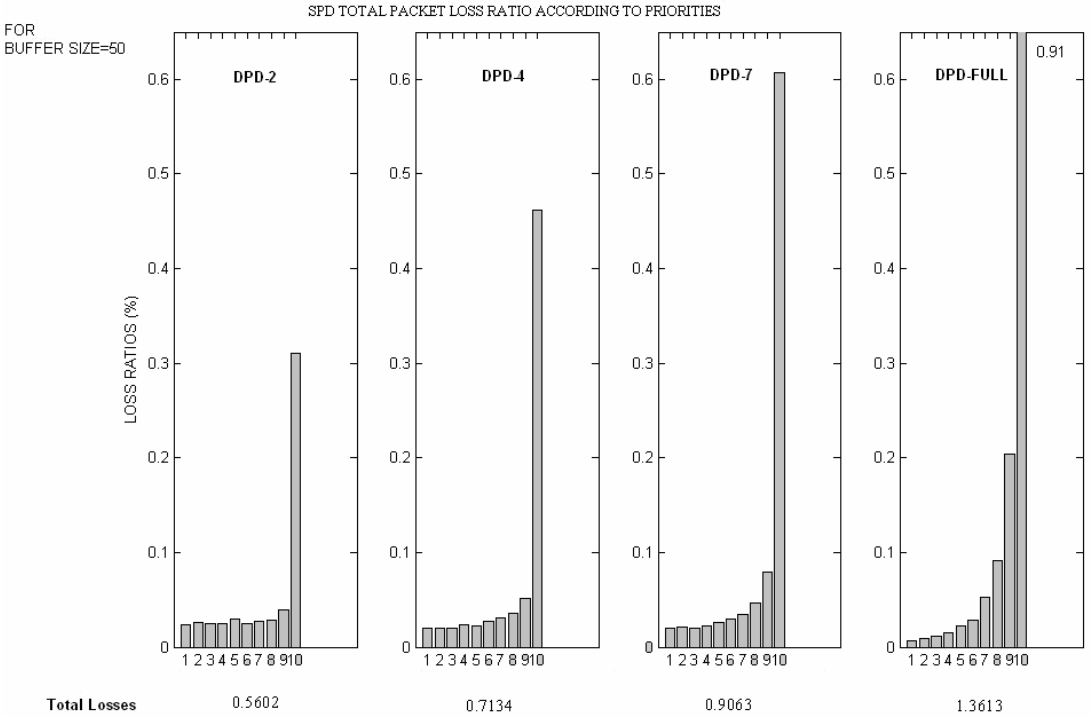

Figure 4.8 Total Packet Losses According to Priorities in SPD algorithms

Table 4.4 Average Waiting Times on the queue for SPD

ALGORITHM \ QUEUE SIZE	10	20	30	40	50
SPD-2	0.0551	0.0970	0.1460	0.1877	0.2325
SPD-4	0.0548	0.0956	0.1388	0.1809	0.2235
SPD-7	0.0543	0.0937	0.1361	0.1768	0.2131
SPD-FULL	0.0543	0.0899	0.1132	0.1359	0.1425

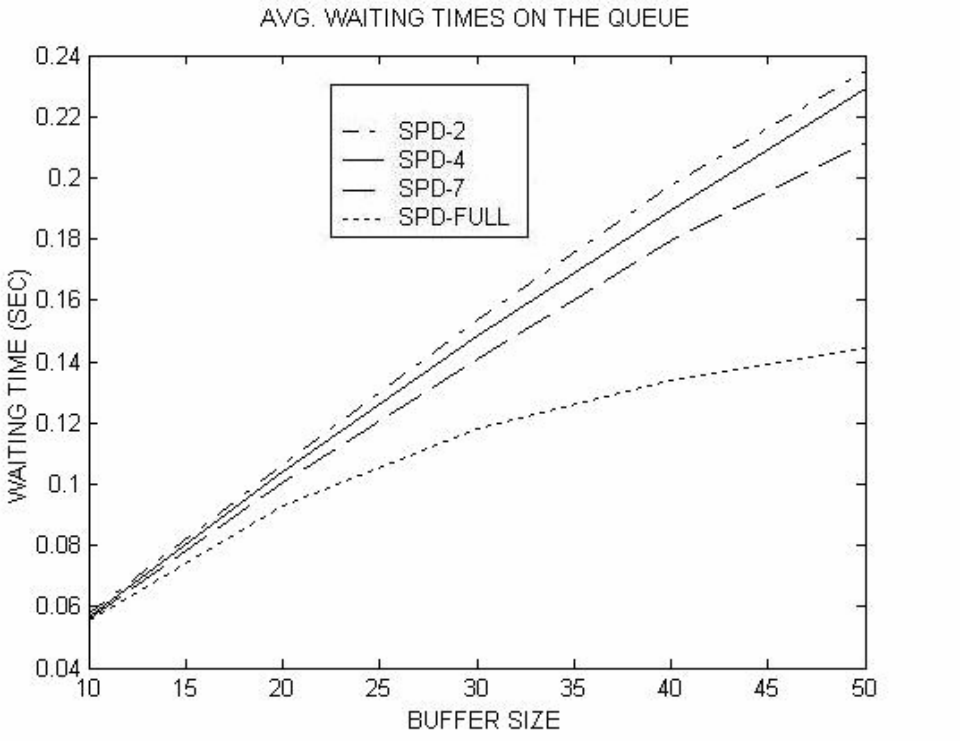

Figure 4.9 Avg.Waiting Times for SPD

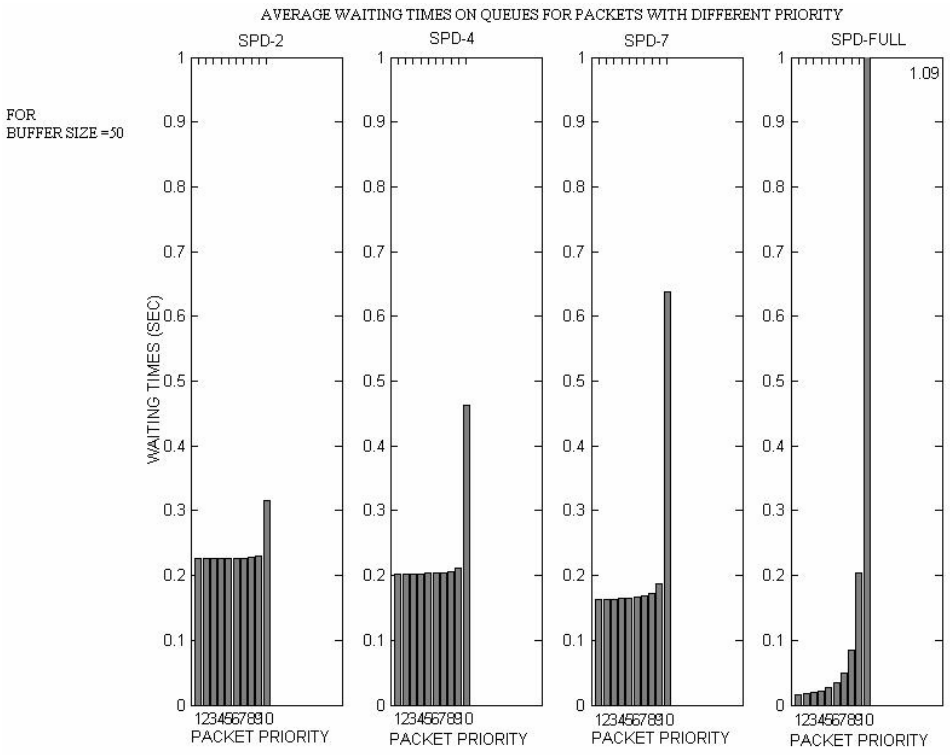

Figure 4.10 Avg.Waiting Times for SPD for each priority

How does the algorithms' complexity change? If the router makes more operations to serve a packet that means if the running time of algorithm increases, there may be more lost packets.

Figure 4.11 shows which algorithms run how many minutes in the same conditions for queue_size equal to 50. In the figure we see that SPD-FULL algorithm finishes in 300 minutes while SPD-2 algorithm finishes in only 21 minutes. We can see the how complexity of the full sort algorithm is. Because of for loops to sort the queue, full sort algorithm gives worst time complexity. This can also lead packet losses in SDP-FULL algorithm because of buffer overflow on real life which is not considered in the simulations.

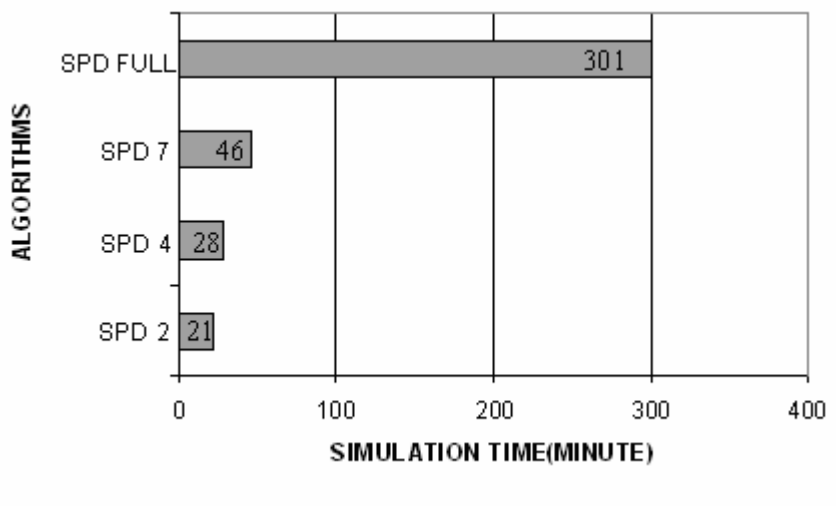

Figure 4.11 Simulation times for SPD algorithms

As a result it is obviously seen that SPD-k algorithms give lower loss ratios according to SP algorithm. Also, the time complexity of SPD-k algorithms is much lower than SPD-FULL algorithm.

CHAPTER 5

DYNAMIC PRIORITY WITH DEADLINE CONSIDERATIONS (DPD)

In this chapter the Dynamic Priority with Deadline Considerations (DPD) packet scheduling algorithm is discussed in detail.

5.1 DPD Scheduling Algorithm Overview

The Dynamic Priority with Deadline Considerations (DPD) explicitly takes into account the deadlines and merges this information with the priorities. It is not always correct to schedule the packets based entirely on their priorities or deadlines. The DPD algorithm tries to combine both of these properties in order to provide diversification of QoS requirements.

Under the DPD packet scheduler, the packets at the buffer are also sorted based on their priorities. However, these priorities do not stay constant. By applying a two-level threshold on their remaining deadlines, the DPD algorithm can modify the priority of the packets waiting in the buffer. The DPD packet scheduler also does partial sorting as the SPD packet scheduler.

In a degree k, DPD packet scheduler (DPD-k), only the first k packets in the buffer are sorted, and the packet with the highest priority among the first k packets is either transmitted or kept waiting.

In a DPD-k algorithm, the packet at the head of the buffer is decided to be served or not, based on its remaining deadline. Assume that the packet at the head of the

buffer has a remaining deadline of t units. The deadline of this packet is compared to two threshold levels (T1 and T2 where T1<T2) and the following actions are taken.

- If $t \le T1$, the packet is immediately served without considering its priority. As the packet's remaining deadline is at a very critical level, it needs to be served immediately. Otherwise, the packet will have will have no remaining deadline and be considered as lost.

- If $T1 < t \le T2$, the packet is served based on the corresponding SPD-k scheduling. Thus, the first k packets will be sorted and the packet with the highest priority will be served. The packet which was at the head of the buffer will be placed in its appropriate position by increasing its priority one level as a way of compensation for removing it from the head of the buffer.

- If $t > T2$, the packet is served based on the corresponding SPD-k scheduling. However, the priority of the packet at the head of the buffer will not be changed as it already has a sufficient amount of remaining deadline.

Figure 5.1 illustrates a DPD-4 packet scheduling example where the packet at the head of the queue with priority 7 has a remaining deadline $t \le T1$. Since the packet's remaining deadline is at a critical level, it is immediately served although there are higher priority packets.

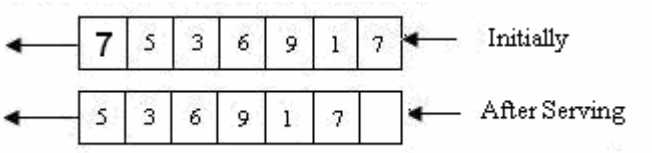

Figure 5.1 DPD-4 packet scheduling with $t \le T1$

Figure 5.2 illustrates a DPD-4 packet scheduling example where the packet at the head of the queue with priority 7 has a remaining deadline $T1 < t \le T2$. In this case DPD packet scheduler works like SPD packet scheduler. However, as the packet

which was scheduled to be transmitted might go back in the queue after the sorting, its priority is increased one level.

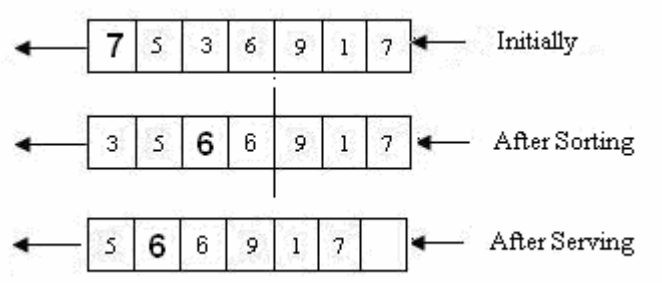

Figure 5.2 DPD-4 packet scheduling with $T1 < t \leq T2$

Figure 5.3 illustrates the last case for a DPD-4 packet scheduler. Since the remaining deadline for the packet at the head of the queue is large enough, the packet is sorted based entirely on SPD-k packet scheduler. As, the packet is not in danger of a deadline violation its priority is not changed as well.

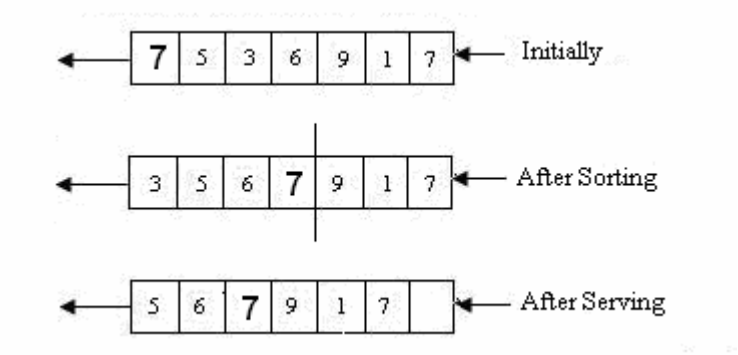

Figure 5.3 DPD-4 packet scheduling example $t > T2$

An important question is how to determine the levels of thresholds for the DPD packet scheduler. Initially, we have determined some static values for these thresholds based on the buffer occupancy. However, ongoing research proposes to have dynamic values for the thresholds based on 3 parameters: the current queue occupancy, the packet's priority and the packet's remaining deadline. If the queue occupancy is high, then the packets wait more in the buffer which leads to choose lower threshold values. Consider a packet with a low priority and has a small

remaining deadline time. If the threshold levels are selected small values, then this packet may be served immediately. If the threshold levels are selected large values, then this packet will be sent to the somewhere else in the queue after sorting and probably will be lost. These considerations will be discussed detailed in section 5.2.2.

5.2 Experimental environment and DPD simulation design

In this section DPD simulation design, experimental environment and program variables will be introduced.

5.2.1 DPD simulation design

The UML diagram of DPD algorithm in Figure 5.4 shows the designed simulation in MATLAB.

Figure 5.4 shows the running simulation algorithm step by step. Again as we see in section 4.2.1, f_queue is used for the same purposes and packet producing scheme is the same as SPD simulation.

In packet arrival event, since the first packet is selected, the next incoming packet is created and added to f_queue. Then, f_queue is sorted. After that, the first selected packet from f_queue is sent to the *real queue*. Then, if queue size allows that transaction, queue size will be incremented by one and packet will be added to the end of the queue. Queue will not be sorted in the arrival event again. It will be sorted in serving event. Next, sorting the f_queue above, the second packet appears at the head of the f_queue array. By looking its arrival time, the next packet arrival time is updated.

In packet serving event; initially, the simulation takes the first packet in the queue and checks if its deadline time is expired or not. If the packet's deadline time is expired, simulation discards it from the queue and increases total lost packet number.

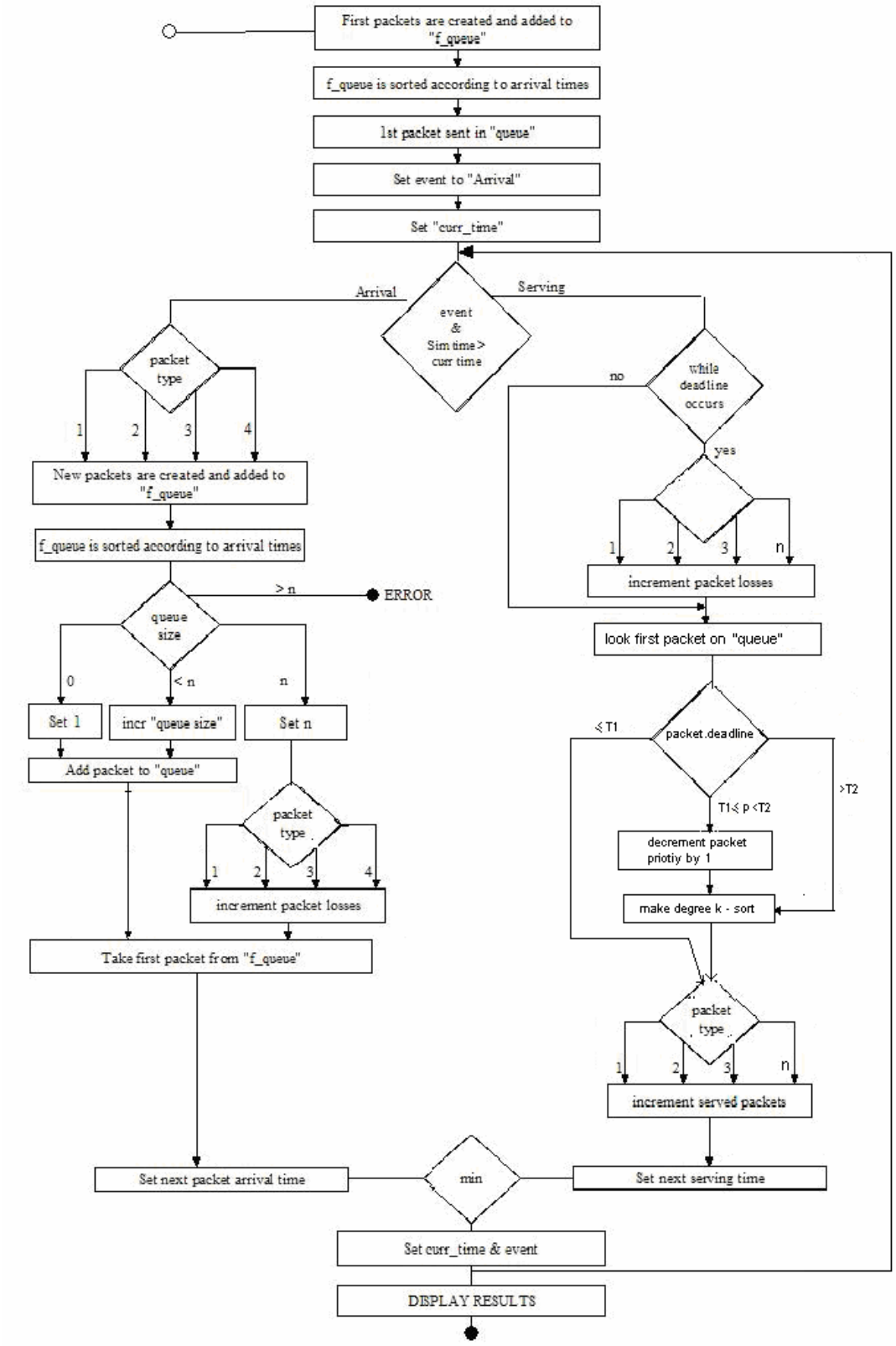

Figure 5.4 UML Diagram of DPD-k algorithms

It is the same as in SPD simulation. Then, the queue will be sorted according to *packet types* in other words *packet base priorities*. Here, our same strategy we used in SPD simulation comes about sorting the queue. The chosen sorting degree here which we call it "k" brings different algorithms like DPD-2, DPD-4, DPD 7, and DPD-FULL. For example, if DPD-k algorithm is used in which k=4, that means DPD-4 algorithm will be simulated, only the first 4 packets are sorted in the queue based on their priorities. Also in DPD-FULL algorithm, all of the queue elements are sorted based on their priorities.

Before sorting there are some jobs to do in DPD algorithms. We defined two threshold levels at the beginning of program which are T1 and T2. Initially, the first packet is taken from queue. If the remaining deadline of the packet is smaller than T1 then the packet will be served immediately similar to FIFO algorithm. If deadline of the packet is between T1 and T2 then the packet's priority will be decreased by 1 and the queue will be sorted by a degree k. If the packet's deadline is larger than T2 then the queue will be sorted by not changing the priority of the packet. Then DPD simulation serves this packet by changing the simulation statistics. At the end, the program checks other packet arrival times and adds the transmission time to calculate current time.

In order to evaluate the performance of DPD packet scheduler in terms of the induced packet losses and waiting times, several simulations are conducted. The following sections describe the experimental environment and present the simulation results observed in DPD algorithm.

5.2.2 Experimental environment and program variables

All of the simulation variables are same with SPD algorithm defined in Section 4.2.2 except two new threshold variables.

T1 value is chosen in the simulation as 0.2 seconds. Considering that queue size is equal to 50 and one packet serving time is 0.01 seconds, it may be selected as a

critical value. T2 value is chosen in the simulation as 1 second. Then, it is considered that for different queue sizes, two threshold values T1 and T2 should also change. When the queue size increases, the T1 and T2 value should decrease and when the average deadline times for packets (bnd_delay) increases, T1 and T2 values also increases. The simulation results for determining the best T1 and T2 values are included in Appendix A.

In the following section, the DPD algorithms' simulation results will be discussed based on these environment and defined threshold levels.

5.3.3. DPD Simulation Results

DPD packet scheduler is based on selecting the two threshold levels which would determine, when and how the priorities of the packets will be modified. Figure 5.5 shows the impact of the lower threshold T1 on the total packet loss ratios when queue size is equal to 50 (Table 5.1). As the value of the lower threshold increases, the total packet loss ratios also increase. Because increasing the lower threshold too much might cause high priority packets which are not at the head of the buffer wait more and thus become more likely candidates for losses. As this threshold increases, the algorithm behaves more like a FIFO algorithm and as this threshold decreases, the algorithm behaves more like the SPD algorithm.

Table 5.1 The effect of T1 on DPD-7 algorithm

T1	0	0.5	1	1.5	2	2.5	3	3.5
LOSS RATIO	0,9091	0,8084	0,8261	0,8454	0,8891	0,8980	0,9342	0,9371

Another parameter is the higher threshold value T2 for the DPD algorithm. Figure 5.6 presents the total packet losses when queue size is 50 for various values of the upper threshold level, T2.

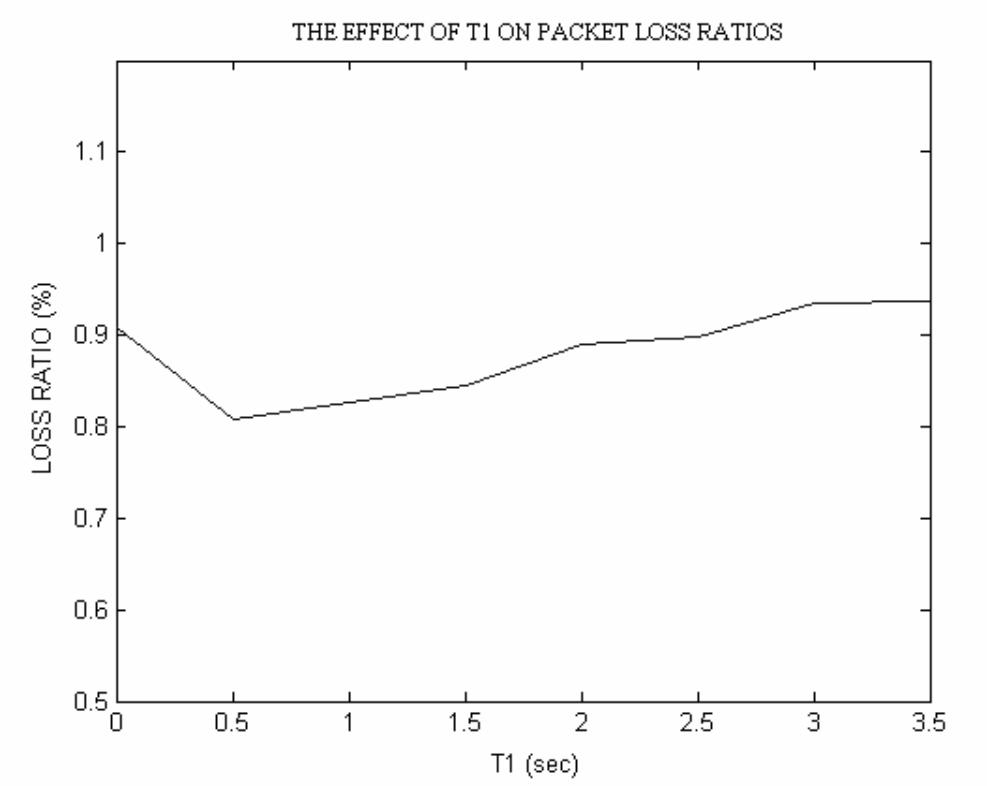

Figure 5.5 The effect of T1 on DPD-7 algorithm

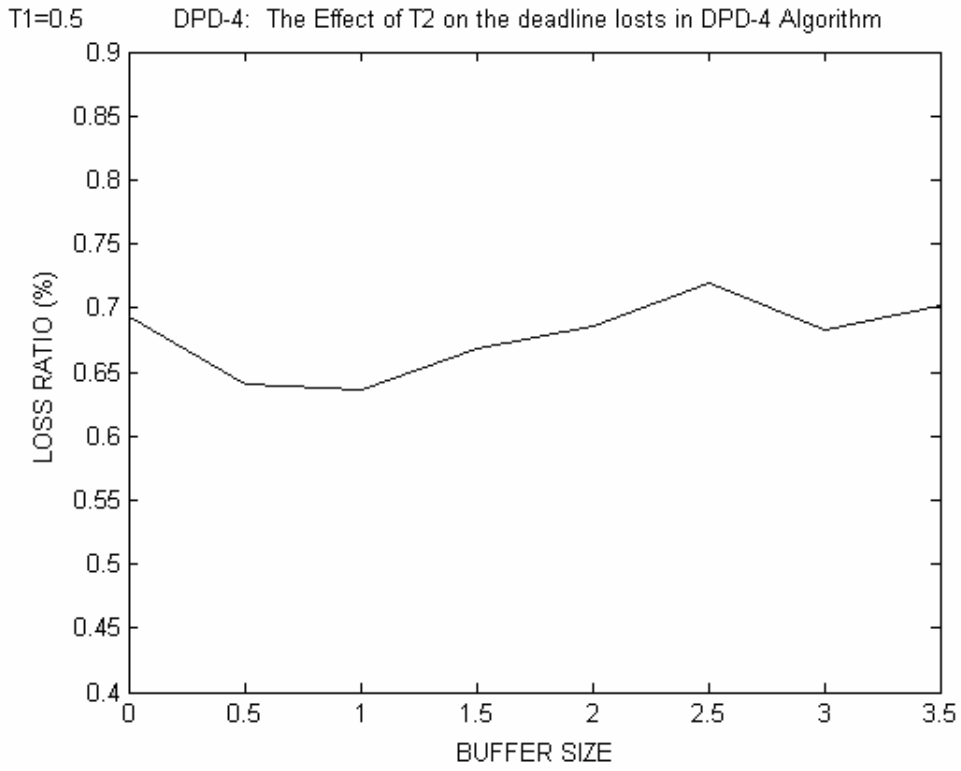

Figure 5.6 The effect of T2 on DPD-4

Figure 5.7 and Table 5.2 present the total packet losses induced for various DPD-k packet schedulers as a function of the buffer size. Similar to SPD packet scheduler, as the order of the packet scheduler increases, the packet loss ratios increase, due to the packet losses induced from deadline violations.

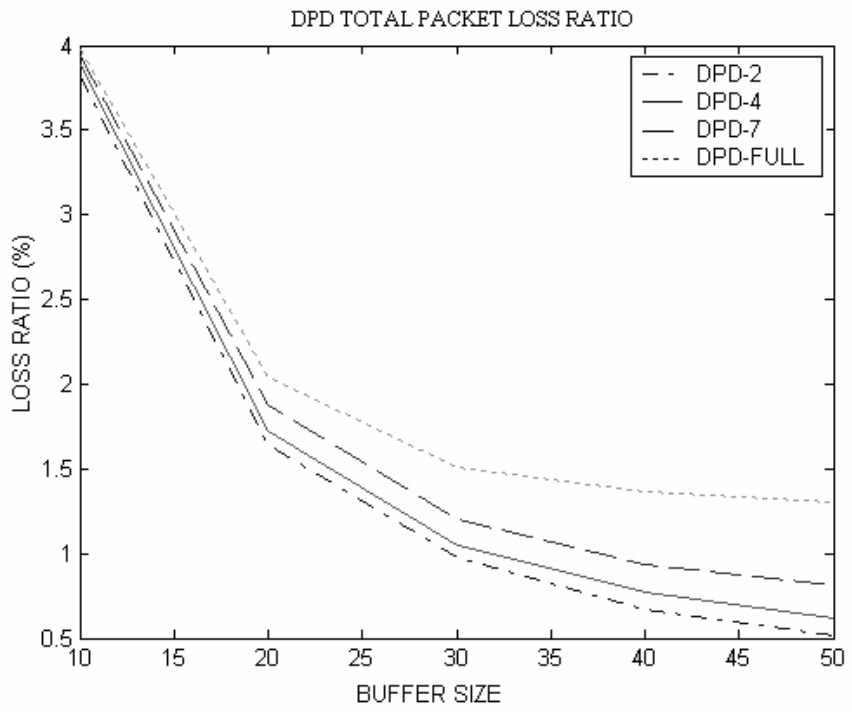

Figure 5.7 Total loss ratio for DPD for T1=0.2 and T2=1

Table 5.2 Total loss ratio for DPD for T1=0.2 and T2=1

ALGORITHM \ QUEUE SIZE	10	20	30	40	50
DPD-2	3.8083	1.6403	0.9727	0.6655	0.5146
DPD-4	3.8832	1.7270	1.0524	0.7663	0.6177
DPD-7	3.9362	1.8792	1.2046	0.9362	0.8084
DPD-FULL	3.9717	2.0355	1.5071	1.3618	1.3072

As seen in Figure 5.8 and Table 5.3, DPD-2 has also the best performance for cell losses due to deadline violations. Because DPD-2 only allows a packet at the head of the buffer move one step back, if its priority is low. Thus, the reason of having better

packet loss ratios for the overall performance of DPD-2 is due to the fact that it performs better in terms of deadline violations.

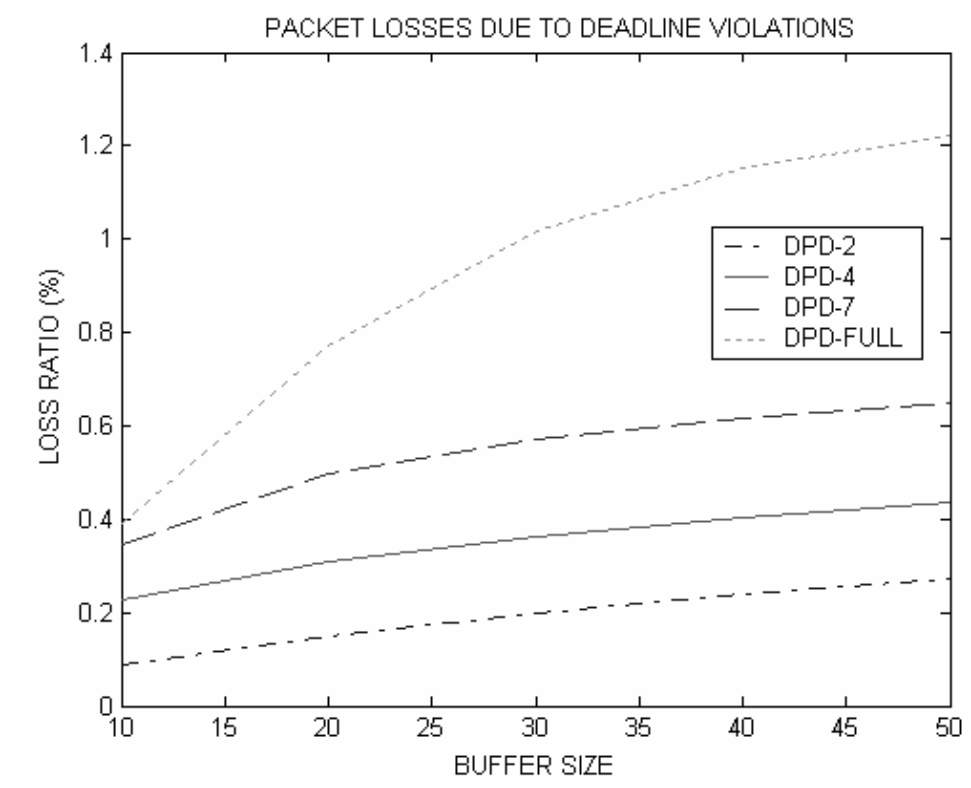

Figure 5.8 Loss ratios due to Deadline in DPD for T1=0.2 and T2=1

Table 5.3 Loss ratios due to Deadline in DPD for T1=0.2 and T2=1

ALGORITHM \ QUEUE SIZE	10	20	30	40	50
DPD-2	0.0866	0.1489	0.1990	0.2384	0.2722
DPD-4	0.2285	0.3093	0.3610	0.4045	0.4345
DPD-7	0.3471	0.4986	0.5695	0.6173	0.6477
DPD-FULL	0.3913	0.7708	1.0155	1.1506	1.2231

In Figure 5.9 and Table 5.4, it is seen that loss ratios due to buffer overflows are higher in DPD-2, DPD-4 and DPD-7 algorithms compared to DPD-FULL algorithm. However, the effect of buffer overflow losses on the total loss ratios is smaller than the effect of deadline based losses. The efficiency of DPD-k algorithms according to their loss ratios are affected largely with the deadline based losses.

Table 5.4 Loss ratios due to Overflow in DPD for T1=0.2 and T2=1

ALGORITHM \ QUEUE SIZE	10	20	30	40	50
DPD-2	3.7217	1.4915	0.7737	0.4270	0.2423
DPD-4	3.6547	1.4177	0.6914	0.3619	0.1832
DPD-7	3.5891	1.3806	0.6351	0.3189	0.1606
DPD-FULL	3.5804	1.2647	0.4916	0.2113	0.0841

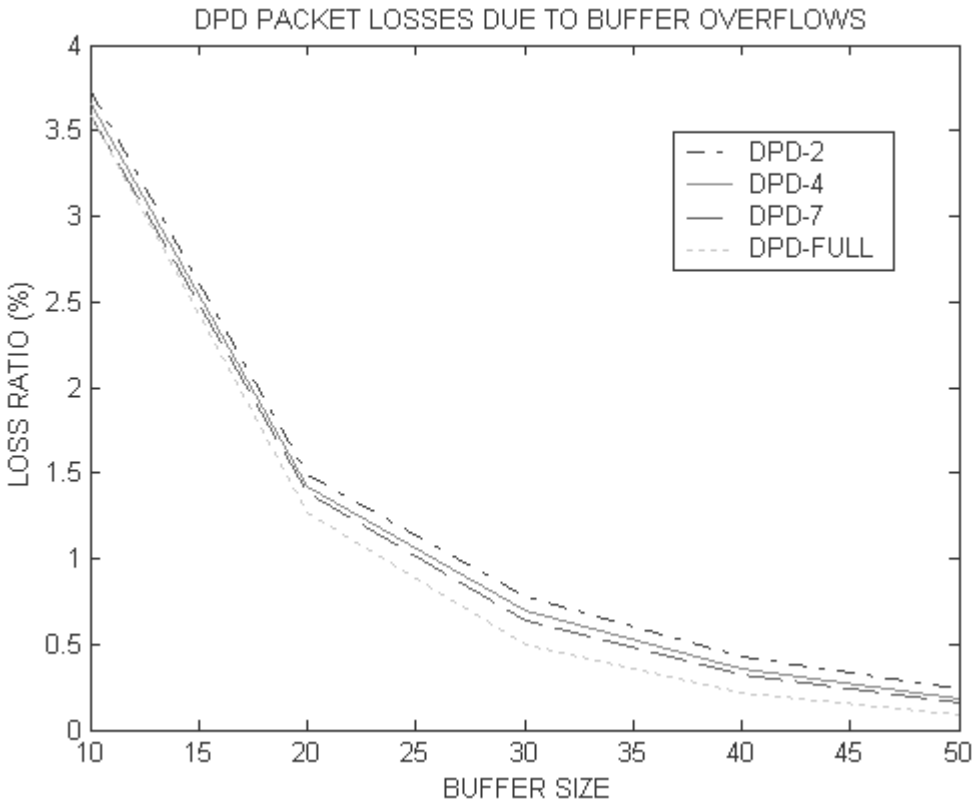

Figure 5.9 Loss ratios due to Overflow in DPD for T1=0.2 and T2=1

DPD-k algorithms' complexity change based on the degree k. Figure 5.16 shows which algorithms run how many minutes in the same conditions for queue_size equal to 50. In the figure we see that full sort algorithm finishes in 300 minutes while degree-2 algorithm finishes in only 21 minutes. It is obviously seen that how complexity of the DPD-FULL algorithm is in the figure.

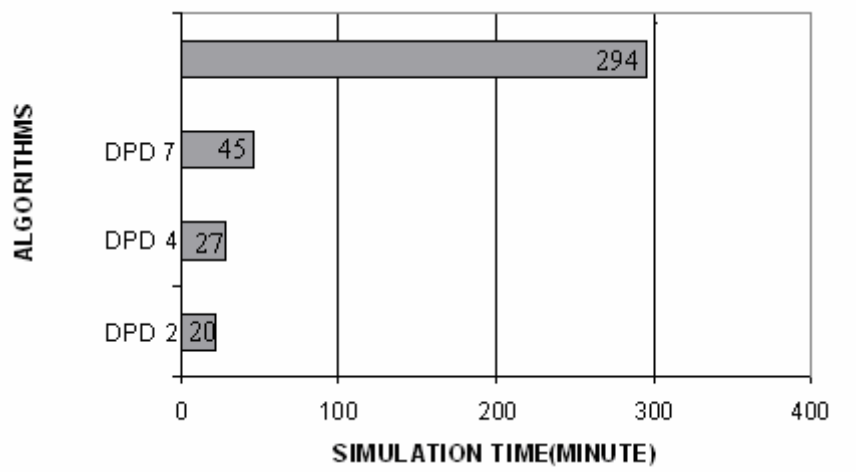

Figure 5.10 Simulation time (complexity) comparisons between DPD algorithms

CHAPTER 6

COMPARISON OF SPD, DPD AND RPQ ALGORITHMS

6.1 Comparison of SPD and DPD Algorithms

In this section, the observed results for SPD and DPD algorithms will be compared based on packet loss ratios and delay in the queue.

Figure 6.1 compares SPD and DPD packet loss ratios for deadline violations as a function of buffer size for SPD-7 and DPD-7. As expected, with a chosen good T1 and T2 values, DPD-7 has a better performance compared to SPD-7. The impact of modifying packet priorities can be best seen from deadline violation losses.

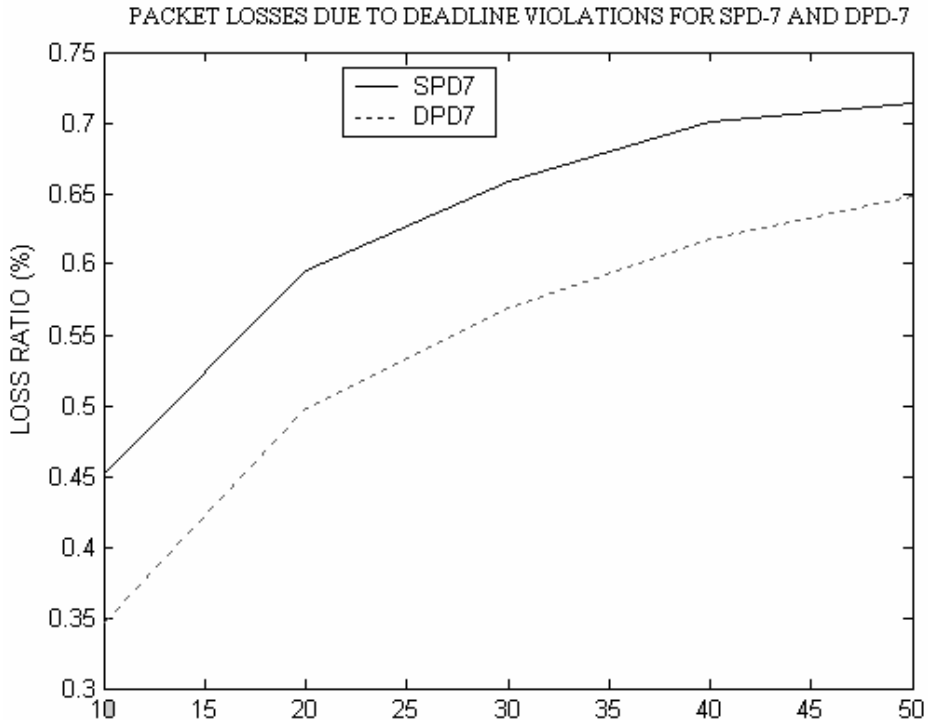

Figure 6.1 Comparing SPD-7 and DPD-7 algorithms according to their loss ratios due to deadline violations

Figure 6.2 compares SPD-7 and DPD-7 algorithms based on their lost ratios as a function of buffer size. Although, DPD-7 and SPD-7 have similar results for packet losses due to buffer overflows. However, considering better performance in terms of deadline violations, DPD total packet loss ratio is less than its corresponding SPD. The value of the loss ratios can be seen from the previous two sections.

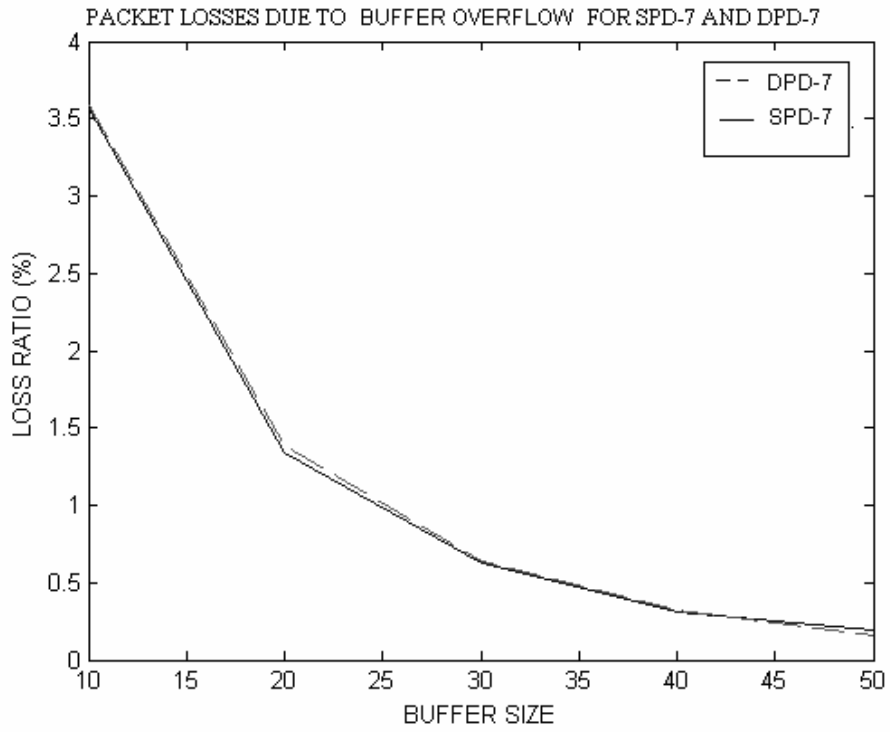

Figure 6.2 Comparing SPD-7 and DPD-7 algorithms according to their loss ratios due to buffer overflow

Based on the loss ratios, DPD-FULL algorithm approximates to EDF scheduling algorithm because DPD-FULL changes packet priorities according to remaining deadline times of packets and sorts all packets on the queue as EDF does. Also, the DPD-FULL algorithm has the best effort according to its match SPD-FULL algorithm based on their loss ratios.

Figure 6.3 shows the average waiting times (delays) of different priority packets in the queue. The packets for SPD algorithm is showed with gray bar, and the packets for DPD algorithm is showed with black bar in the figure. In DPD algorithm

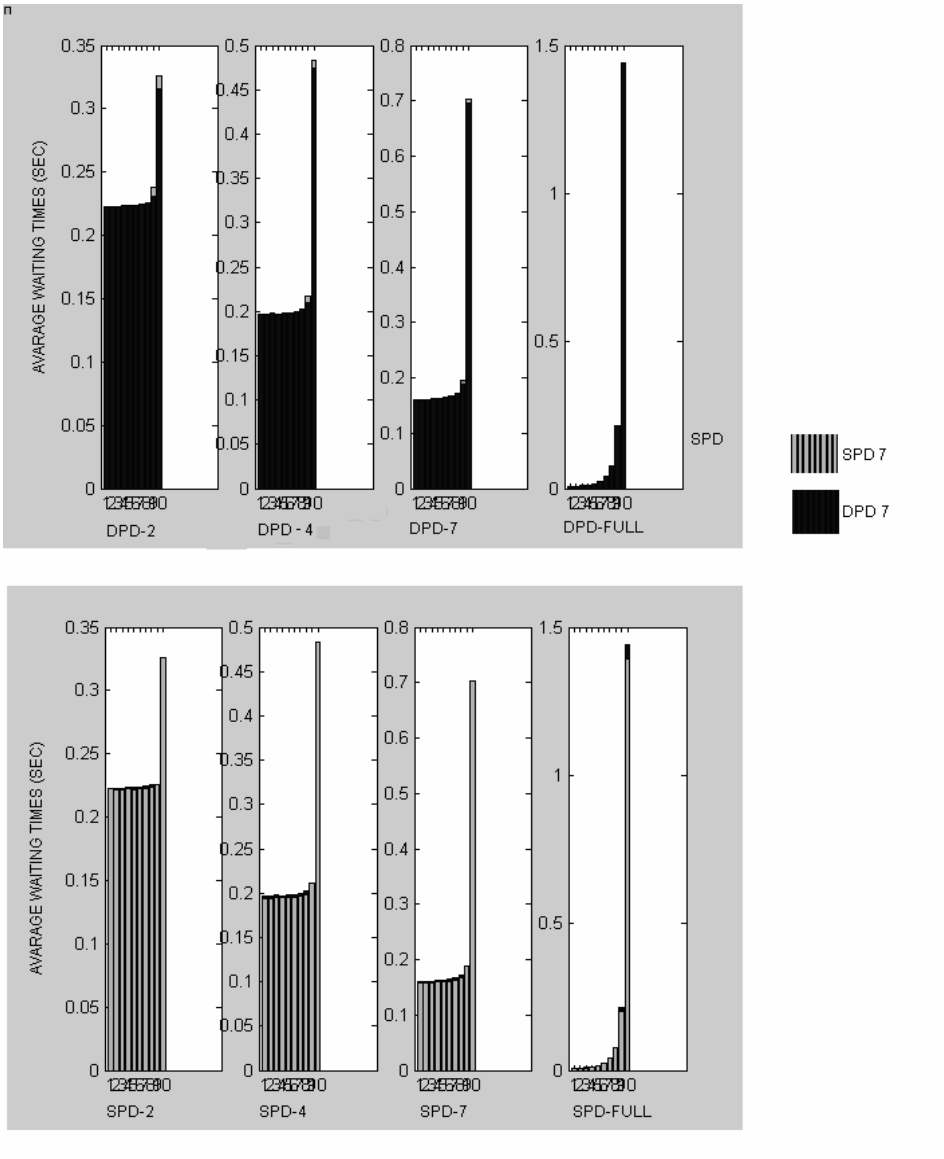

Figure 6.3 Comparing SPD-7 and DPD-7 algorithms according to their average waiting times on the queue

the priorities of packets which have small remaining deadline times are increased. This process permits low priority packets to increase their priorities. Because of that, low priority packets (e.g. priority-10 packets) generally wait less in the queue in DPD algorithm compared to SPD algorithm. This can be seen in the figure. In the above part of the graph, it is seen that e.g. priority-10 packets wait in the queue more for SPD because gray bars exceed the black bars on the graph. As opposite, in the same manner, higher priority packets (e.g. priority-2 packets) wait more in the queue

for DPD algorithms. This can be seen from bottom part of the graph. In this time, the black bars are exceeding the gray bars for higher priority packets.

The time complexity of DPD-k algorithms is lower than their corresponding SPD-k algorithms. Figure 6.4 shows which algorithms run how many minutes in the same conditions for queue_size equal to 50. In the figure the DPD algorithms end shortly. The simulation runs for 2 million packets. If the algorithms could be run 2 billion packets, we can imagine how the difference is big. As a result, we can say that DPD-k algorithms are less complex according to their corresponding SPD-k algorithms, because they do not sort the queue every time as SPD-k algorithms do.

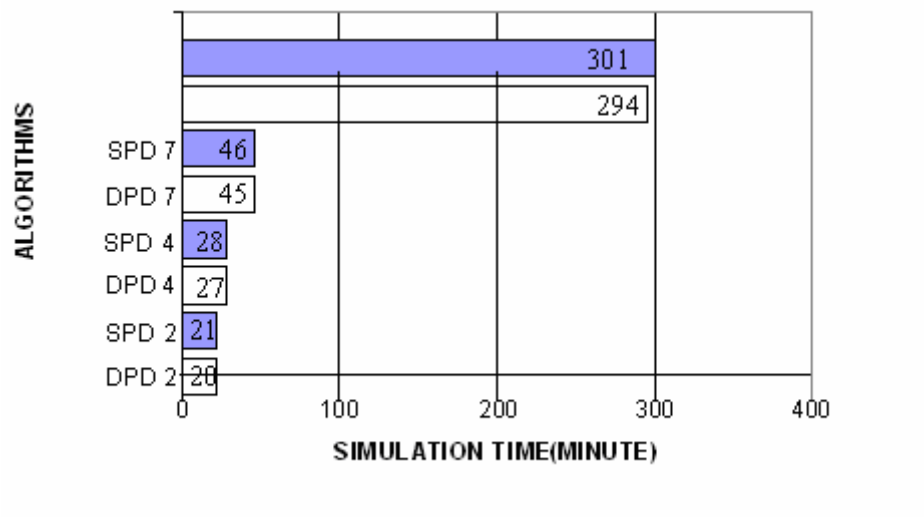

Figure 6.4 Simulation time comparisons between SPD and DPD (Time complexity of the algorithms)

In summary, it is obvious that DPD algorithms give better results in all conditions. DPD algorithms are less complex than SPD algorithms, they guarantee lower packet losses with a defined good T1 and T2 threshold values.

6.2 Comparison of DPD and RPQ Algorithms

In this section, the observed results for DPD and RPQ algorithms will be compared based on packet loss ratios and delay in the queue.

In the RPQ simulation design, there are 10 queues for 10 types of priority packets. The other simulation environment variables are same within the DPD simulation design.

In Figure 6.5, the total packet loss ratios are seen for different queue sizes for each used priority queue in RPQ algorithm. From the figure, it is observed that when one queue size is equal to 21, that mean the total buffer capacity is 210, the loss ratio will be 1.3104%. That is higher than the DPD-k algorithms which use queue size 50. However, as seen in Figure 6.6, average waiting times for packets, observed in RPQ packet scheduling algorithm, are lower than the DPD and SPD algorithms. When the queue size for RPQ is 5 (totally 50), the observed average deadline time is 0.0439 seconds. That value is lower than the DPD algorithms. DPD-FULL algorithm gives the best average waiting time between the DPD-k algorithms and it is 0.1220 seconds. Low delay parameters are the goal of RPQ that is also seen in the simulation.

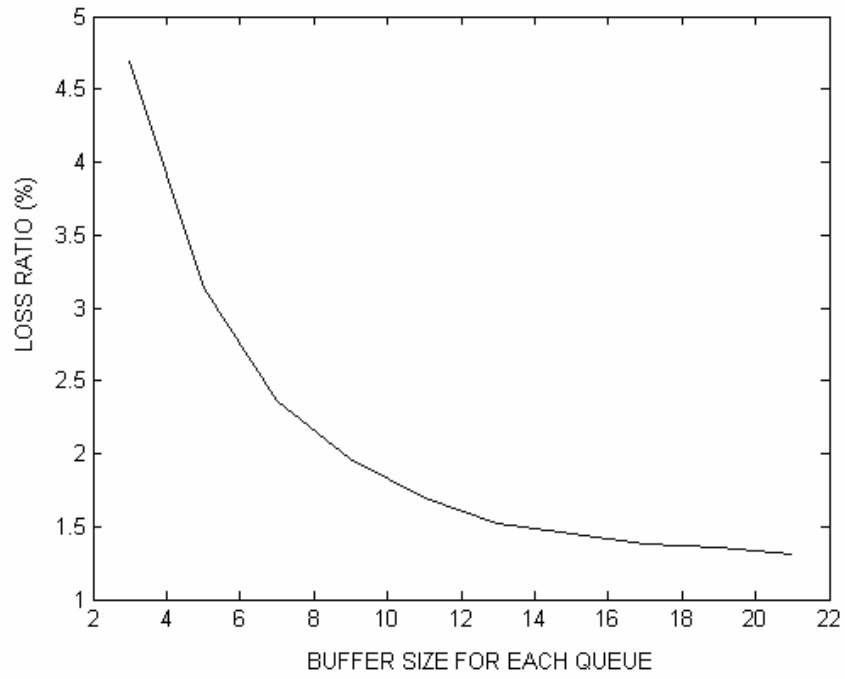

Figure 6.5 Total packet loss ratios versus buffer size for RPQ

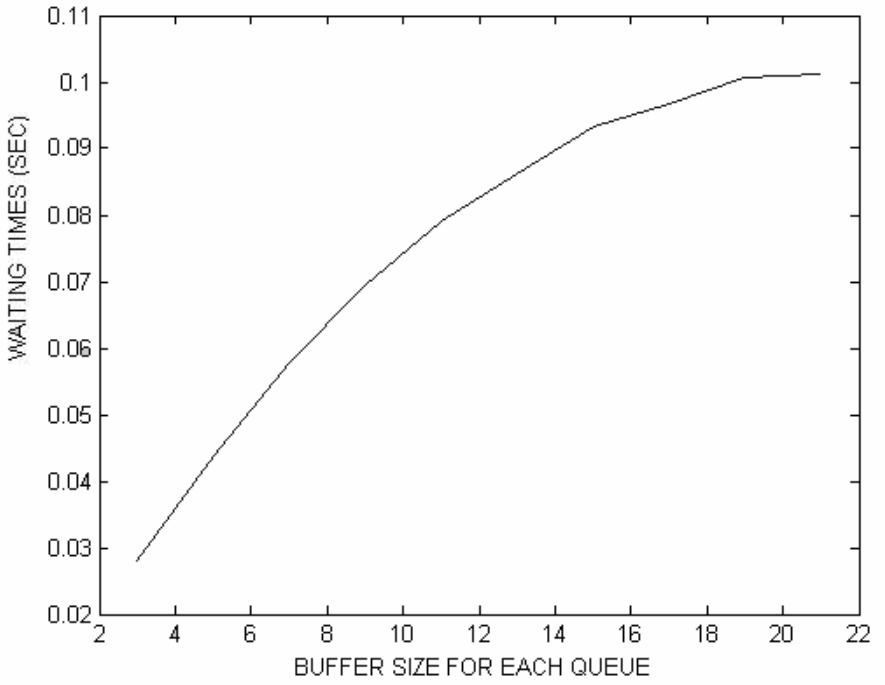

Figure 6.6 Average waiting times for packets versus buffer size in RPQ

In RPQ algorithm, if queue size for each queue is taken 5 (totally 50), then packet lost due to deadline violations can be examined. In Figure 6.7, it is observed that the deadline violation loss value for RPQ algorithm is between the DPD-4 and DPD-7.

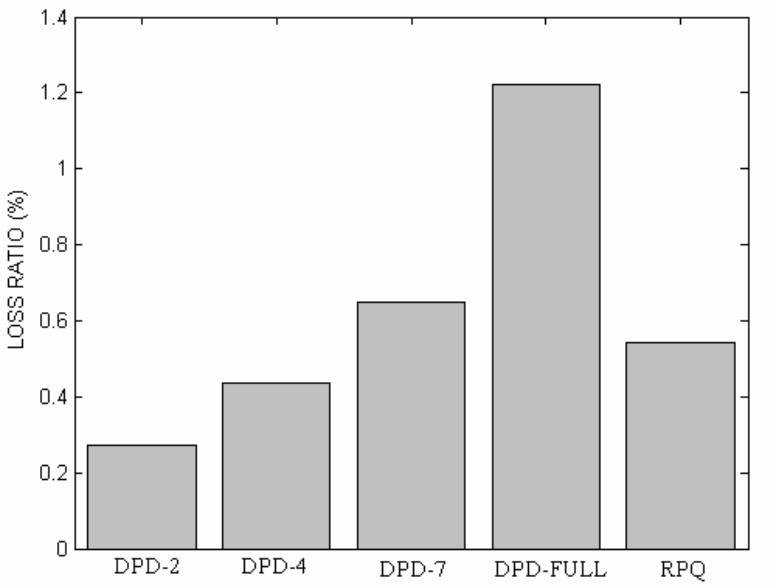

Figure 6.7 Packet lost values due to deadline violations

CHAPTER 7

CONCLUSIONS

Packet schedulers are used as a technique to provide the QoS requirement on various kinds of applications. The induced delay over a connection is an important QoS parameter as well as the ratio of packet losses. In this thesis, a new packet scheduler with two different versions is introduced; The Static Priority with Deadline Considerations (SPD) and Dynamic Priority with Deadline Considerations (DPD). Different from classical packet schedulers, both of these packet schedulers try to integrate the delay and loss parameters, since NGNs will be heavily dependent on supplying QoS requirements of various applications.

The SPD and DPD packet schedulers try to reduce the processing overhead of sorting at a network buffer by introducing, order-k sorting, thus SPD-k and DPD-k. Sorting a few elements at the buffer gives similar results of sorting all the elements at the buffer. This results less complex algorithm, low loss ratios and low delay times compared to classical full sort algorithms like EDF and SP.

SPD packet scheduler differs from classical SP algorithm in a way that it considers the packet deadline times and includes the expired deadline packets as lost packets. Also the SPD-k algorithms have lower complexity compared to classical SP algorithm because of the overhead of sorting all packets in SP. The SPD-k algorithms give low total lost ratios in the order of SPD-2, SPD-4, SPD-7, and SPD-FULL. SPD-2 algorithm has given the lowest total loss ratios in the simulation with the lowest packet losses due to deadline violations. On the other hand, the delay of packets in the queue is the highest in SPD-2 algorithm and lowest in SPD-FULL algorithm. This causes also the highest buffer overflow losses in SPD-2 algorithm and lowest buffer overflow losses in SPD-FULL algorithm.

On the other hand, DPD packet scheduler is based on the classical SP algorithm with the improvement on changing priorities of packets in the queue. In the simulation, two threshold levels selected; the lower one is T1 and higher one is T2. When T1 and T2 are selected as too small values, the DPD algorithm approximates to SPD algorithm. If the difference between T1 and T2 values is high then the algorithm complexity increases because of the overhead of sorting. If T1 and T2 value is selected too high then the algorithm runs in FIFO scheme. It is observed from the simulation that the algorithm gives the best results when T1=0.2 seconds and T2=1 second.

Again, in DPD-k algorithms the lowest loss ratios and highest delay times are observed in DPD-2 simulation. The losses due to buffer overflow are similar for all DPD-k algorithms, but the losses due to deadline expiration are lowest in DPD-2 algorithm.

While the priorities are kept fixed for an SPD packet scheduler, DPD packet scheduler modifies the priorities based on the remaining deadline. The simulation results have shown that, DPD packet scheduler furthermore decreases the packet losses due to deadline violations and gives low loss ratios compared to SPD algorithm.

The current ongoing and future work of SPD and DPD packet schedulers include to increase the threshold levels for the DPD packet scheduler, with a possibility of introducing dynamic thresholds based on the QoS requirements of the current packet to be served. If the threshold levels are increased, e.g. not only T1 and T2 but also a new threshold value T3, DPD algorithm complexity may be decreased more. Also, the impact on jitter will be studied since jitter is another important QoS parameter.

REFERENCES

[1] H. Zhang, “Service disciplines for guaranteed performance service in packet-switching networks”, proceedings of the IEEE, 83(10):1374-1399, October 1995.

[2] Mehdi Karhagi and Ali Movaghar, “A Method for Performance Analysis of Earliest-Deadline-First Scheduling Policy”, Proceedings of DSN ’04, Sep 2004.

[3] Nichael Menth and Ruediger Martin, “Service Differentiation with MEDF scheduling in TCP/IP networks”, Journal of Computer Communications (article in press), Sep. 2005.

[4] Yong Jiang, Jianping Wu, Xiaoxia Sun, “A Packet Scheduling Algorithm in High Performance Routers”, proceedings of IEEE ICATM2001, Korea, pp 168-172.

[5] Cisco, “Internetworking Technologies Handbook”, Introduction to QOS, Chapter 49 URL http://www.cisco.com,

[6] Tim Szigeti, “End-to-End Qos Network Design”, Cisco Press, 2004.

[7] P.Goyal, H.M.Vi, and H.Chen, “Start-time Fair Queuing: A scheduling algorithm for integrated services”, proceedings of the ACM-SIGCOMM '96, Palo Alto, CA, August 1996, pp 157-168.

[8] Dallas E. Wrege, Jörg Liebeherr, “A Near-Optimal Packet Scheduler for QoS Networks”, proceedings of IEEE Infocom '97, Kobe, Japan, April 1997, pp 577-583.

[9] J.C.R Bennett and H.Zhang, “WF'Q: Worst-case fair weighted fair queuing”, proceedings of ACM SIGCOMM '96, Palo Alto, CA, August 1996, pp143-156.

[10] N.McKeown; V. Anantharam; J.Walrand, “Achieving 100% Throughput in an input-queued switch”, IEEE Infocom '96.

[11] Andrew S. Tanenbaum, “Computer Networks, Fourth Edition”, 2003, Pearson Education.

[12] Floyd, S., “Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic”. Computer Communications Review, Vol.21, No.5, October 1991, pp. 30-47. URL http://ftp.ee.lbl.gov/floyd/.

[13] Gaynor, M., “Proactive Packet Dropping Methods for TCP Gateways”, October 1996, URL http://www.eecs.harvard.edu/~gaynor/final.ps.

[14] T. V. Lakshman, Arnie Neidhardt, Teunis Ott, “The Drop From Front Strategy in TCP Over ATM and Its Interworking with Other Control Features”, Infocom 96, MA28.1.

[15] ® 2001 Tom Sheldon and Big Sur Multimedia - “Queuing”
URL www.linktionary.com/q/queuing.html

[16] Anton´ n Kr´al, “Scheduling Strategies in Routers”, MS Thesis, 2003

[17] Tom Sheldons Linktionary.com, “Tom Sheldon and Big Sur Multimedia” URL http://www.linktionary.com/q/qos.html, 2001

[18] WIKIPEDIA,”Packet Scheduling Types”, The Free Encyclopedia, URL http://en.wikipedia.org/

[19] Prof. Dr.-Ing. Adam Wolisz, “Packet Scheduling for Bandwidth Sharing and Quality of Service Support in Wireless Local Area Networks”, Technische Universität Berlin, URL www.et2.tu-harburg.de/Mitarbeiter/Wischhof/thesis.pdf

APPENDIX A

FINDING BEST T1 AND T2 VALUES FOR DPD ALGORITHMS

In this section, the effect of average deadline times for packets which is denoted as bnd_delay, and the effect of buffer size which is denoted as queue_size on the threshold values T1 and T2 will be examined.

A.1 The Effect of Average Deadline Times of Packets (BND_DELAY) on the threshold values

Bnd_delay is the average deadline times for the packets. If the selected value of bnd_delay is small, then there will be more packet losses as expected. Because the losses due to deadline violations will increase.

In bnd_delay simulations, the queue_size is taken as 50 for different bnd_delay values. Also the used algorithm to examine the effect of bnd_delay on T1 and T2 is DPD-4.

In Tables A.1, A.2, A.3, and A.4, the loss ratios table is presented when bnd_delay is equal to 4,8,16 and 24 seconds in order. Also, Figures A.1, A.2, A.3, and A.4 show the corresponding loss ratios for different T1 and T2 values with a specific bnd_delay value with 3-d graphs. With the results of simulations, it is seen that when bnd_delay value increases, the range for selecting T1 and T2 values also increases.

To explain the simulation results, a critical loss value, called *loss ratio limit,* is defined for different simulations with different bnd_delay values. And then, the loss

ratios that are lower than the loss ratio limit are circled on the table. Lastly, the tables are commented with circled values.

As observed from the figures and tables, if T1 and T2 value is selected as 0, then there will be more packet losses. This proves that good selected T1 and T2 values in DPD simulations decrease the total loss ratio.

Table A.1 Total loss ratios for different T1 and T2 values when bnd_delay = 4 secs

lossratio =

BND_DELAY=4

T1 \ T2	0	0.5	1.0	1.5	2.0	2.5	3.0
0	1.1079	1.0090	1.0321	1.1053	1.1519	1.2020	1.2300
0.1	0	0.9550	1.0019	1.0319	1.0759	1.1344	1.1556
0.2	0	0.9719	1.0103	1.0421	1.0620	1.1066	1.1435
0.3	0	0.9529	0.9935	1.0544	1.0766	1.1018	1.1750
0.4	0	0.9809	0.9992	1.0178	1.0855	1.1293	1.1715
0.5	0	0.9536	1.0031	1.0504	1.0780	1.1208	1.1709
0.6	0	0	1.0282	1.0474	1.0650	1.1288	1.1699
0.7	0	0	1.0140	1.0412	1.0941	1.1386	1.1657
0.8	0	0	1.0017	1.0638	1.0953	1.1537	1.1834
0.9	0	0	1.0080	1.0343	1.0825	1.1202	1.1733
1.0	0	0	1.0072	1.0662	1.0911	1.1202	1.1935

Loss ratio limit	T1 and T2 range	The best case
1.00 %	0.1 < T1 < 0.5 0.5 < T2 < 1.0	T1=0.3 T2=0.5

In the simulations, it is observed that if T1 and T2 value increase, generally the loss ratios also increase without the effect of bnd_delay value. Bnd_delay value only determines the lower and upper limit of T1 and T2 value for low losses. Therefore, after that limits, if T1 and T2 value increase, the loss ratio will also increase. If bnd_delay increases the range to select best T1 and T2 values will be bigger.

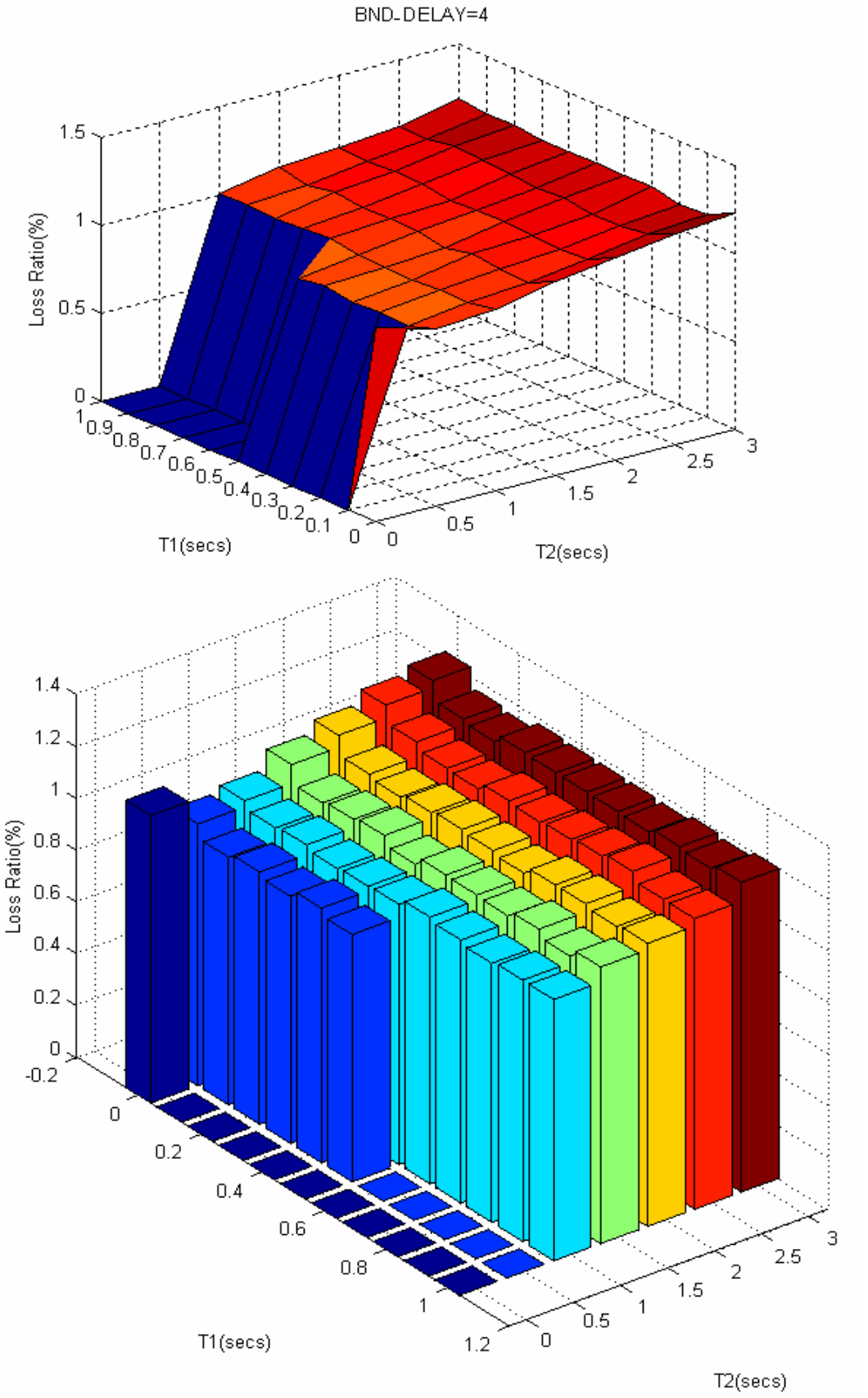

Figure A.1 Total loss ratios for different T1 and T2 values when bnd_delay = 4 secs

Table A.2 Total loss ratios for different T1 and T2 values when bnd_delay = 8 secs

lossratio = **BND_DELAY = 8 SECONDS**

T1 \ T2	0	0.5	1.0	1.5	2.0	2.5	3.0	3.5
0	0.6930	0.6403	0.6354	0.6689	0.6852	0.7202	0.6829	0.7015
0.1	0	0.6019	0.6574	0.6315	0.6964	0.6448	0.6774	0.6726
0.2	0	0.6046	0.6093	0.6457	0.6804	0.6314	0.6677	0.6999
0.3	0	0.6448	0.6124	0.6252	0.6581	0.6768	0.6740	0.6856
0.4	0	0.5891	0.6278	0.6640	0.6604	0.6492	0.7005	0.6867
0.5	0	0.6322	0.6372	0.6178	0.6157	0.6499	0.6443	0.7046
0.6	0	0	0.6260	0.6613	0.6761	0.6651	0.7015	0.7014
0.7	0	0	0.6273	0.6318	0.6807	0.7000	0.6443	0.6769
0.8	0	0	0.6373	0.6502	0.6575	0.6602	0.6542	0.7085
0.9	0	0	0.6346	0.6232	0.6371	0.6405	0.6637	0.6728
1.0	0	0	0.6446	0.6394	0.6415	0.6635	0.6606	0.6695
1.1	0	0	0	0.6311	0.6564	0.6708	0.6877	0.6967
1.2	0	0	0	0.6342	0.6755	0.6872	0.6700	0.6834
1.3	0	0	0	0.6732	0.6479	0.6549	0.6857	0.7287
1.4	0	0	0	0.6504	0.6902	0.6633	0.6814	0.7177
1.5	0	0	0	0.6438	0.6899	0.6702	0.7103	0.6729
1.6	0	0	0	0	0.6683	0.6689	0.6856	0.6796
1.7	0	0	0	0	0.6519	0.6599	0.6532	0.7012
1.8	0	0	0	0	0.6571	0.6594	0.7004	0.7226
1.9	0	0	0	0	0.6617	0.6708	0.6879	0.6814
2.0	0	0	0	0	0.6516	0.6687	0.6836	0.6848
2.1	0	0	0	0	0	0.6661	0.7205	0.7223

Loss ratio limit	T1 and T2 range	The best case
0.62%	0.1 < T1 < 0.5 0.5 < T2 < 2.0	T1=0.4 T2=0.5

Table A.3 Total loss ratios for different T1 and T2 values when bnd_delay= 16 secs

lossratio = **BND_DELAY = 16 SECONDS**

T1 \ T2	0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
0	0.5115	0.5044	0.4522	0.4763	0.4847	0.4789	0.5034	0.4956	0.5059	0.5236	0.4764
0.1	0	0.4646	0.4621	0.4917	0.4645	0.4761	0.5007	0.5010	0.4844	0.4736	0.4780
0.2	0	0.4694	0.4405	0.4694	0.4582	0.4843	0.4786	0.4555	0.4720	0.4913	0.5010
0.3	0	0.4707	0.4675	0.4601	0.4885	0.4637	0.4410	0.4588	0.4770	0.4662	0.4856
0.4	0	0.4968	0.4415	0.4536	0.4855	0.4584	0.4859	0.4850	0.4890	0.5070	0.4619
0.5	0	0.4722	0.4954	0.4437	0.4667	0.4738	0.4688	0.4918	0.4841	0.4920	0.5106
0.6	0	0	0.4348	0.4715	0.4675	0.4986	0.4710	0.4821	0.5080	0.5075	0.4917
0.7	0	0	0.4506	0.4558	0.4787	0.4498	0.4792	0.4679	0.4945	0.4877	0.4648
0.8	0	0	0.4532	0.4728	0.4827	0.4831	0.4804	0.4731	0.5013	0.4758	0.4547
0.9	0	0	0.4417	0.4614	0.4505	0.4700	0.4815	0.4413	0.4961	0.4727	0.4971
1.0	0	0	0.4765	0.4987	0.4473	0.4714	0.4776	0.4719	0.4955	0.4880	0.4955
1.1	0	0	0	0.4884	0.4421	0.4799	0.4748	0.5054	0.4769	0.4887	0.5145
1.2	0	0	0	0.4799	0.5018	0.4504	0.4744	0.4810	0.4752	0.4995	0.4914
1.3	0	0	0	0.4799	0.5018	0.4505	0.4744	0.4811	0.4751	0.4992	0.4918

Loss ratio limit	T1 and T2 range	The best case
0.45%	0.1 < T1 < 1.1 0.5 < T2 < 3.5	T1=0.6 T2=1.0

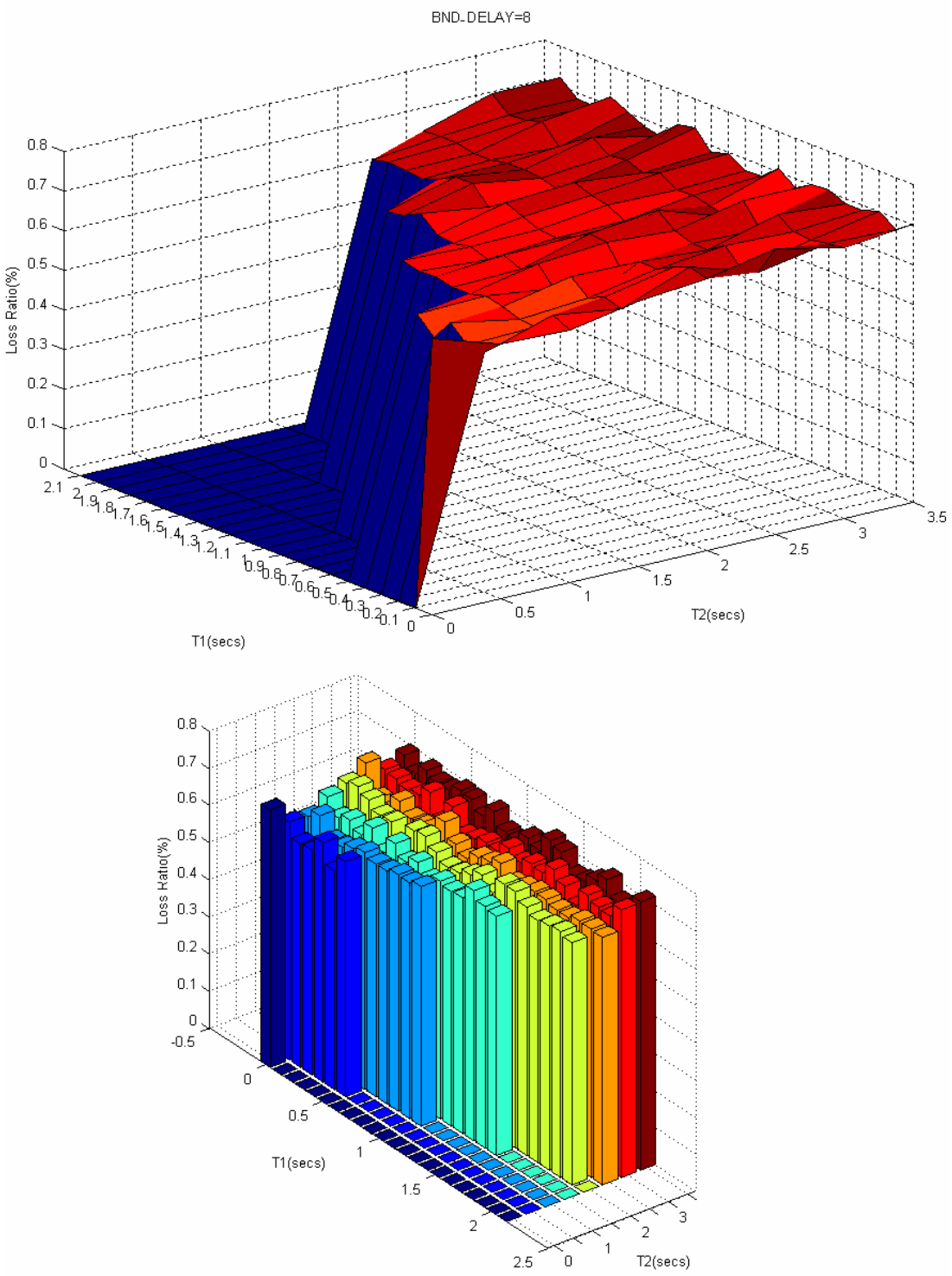

Figure A.2 Total loss ratios for different T1 and T2 values when bnd_delay = 8 secs

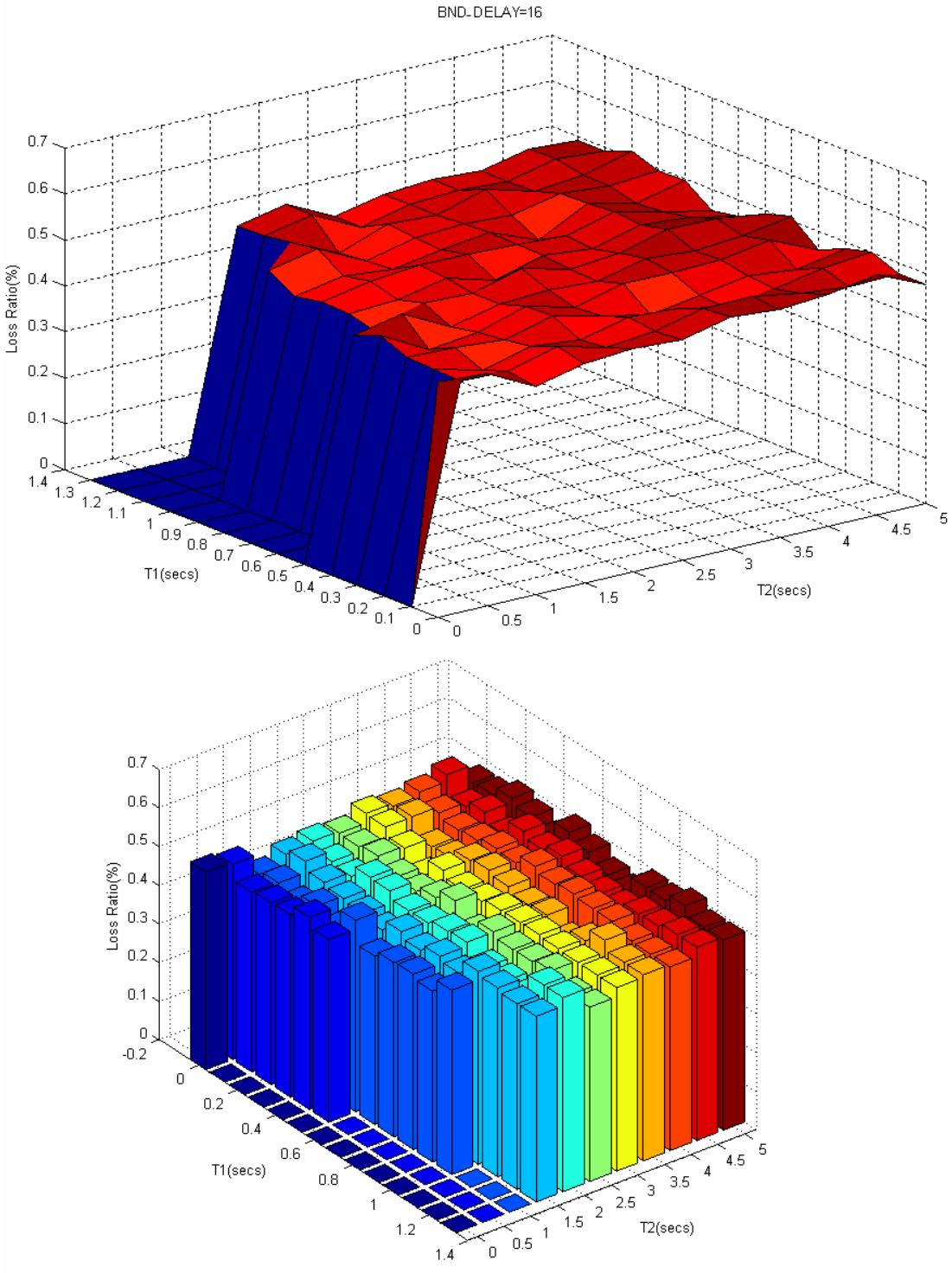

Figure A.3 Total loss ratios for different T1 and T2 values when bnd_delay= 16 secs

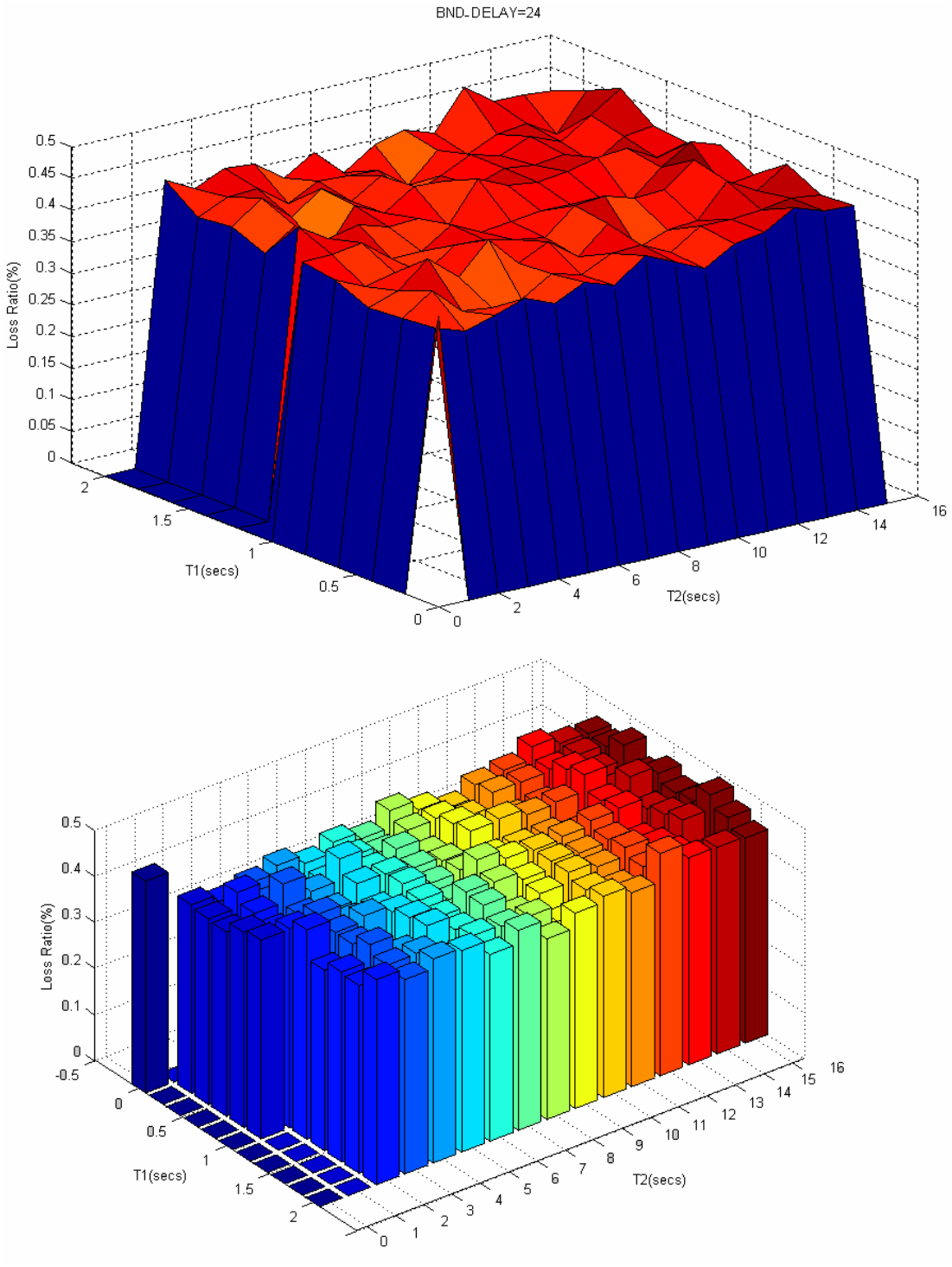

Figure A.4 Total loss ratios for different T1 and T2 values when bnd_delay= 24 secs

Table A.4 Total loss ratios for different T1 and T2 values when bnd_delay= 24 secs

lossratio =

BND DELAY = 24 SECONDS

T1 \ T2	0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0
0	0.4604	0	0	0	0	0	0	0	0	0	0
0.2	0	0.4097	0.3946	0.4079	0.4240	0.4050	0.4272	0.4118	0.4488	0.4270	0.4063
0.4	0	0.4042	0.4377	0.3940	0.3891	0.3919	0.4207	0.4073	0.4307	0.4330	0.4110
0.6	0	0.4005	0.4169	0.4568	0.4175	0.4645	0.4174	0.4388	0.4053	0.4458	0.3966
0.8	0	0.4240	0.4033	0.4128	0.4005	0.4350	0.4324	0.4222	0.4140	0.4544	0.4573
1.0	0	0.4320	0.4330	0.4406	0.4036	0.4040	0.4356	0.4138	0.4185	0.4248	0.4231
1.2	0	0	0.4555	0.4261	0.4377	0.4232	0.4251	0.4024	0.4654	0.4444	0.4391
1.4	0	0	0.3924	0.4254	0.3697	0.4357	0.4122	0.4441	0.4419	0.4234	0.4399
1.6	0	0	0.4131	0.4488	0.4078	0.4423	0.4102	0.4396	0.3976	0.4176	0.4400
1.8	0	0	0.4084	0.4292	0.4309	0.3967	0.4215	0.4184	0.4172	0.3821	0.4267
2.0	0	0	0.4458	0.4224	0.4433	0.4379	0.4057	0.4345	0.3925	0.4246	0.4376

Loss ratio limit	T1 and T2 range	The best case
0.40%	0.2 < T1 < 2.0 0.5 < T2 < 10.0	T1=1.4 T2=4.0

A.2 The Effect of Buffer Size (QUEUE_SIZE) on the threshold values

Queue_size is the total capacity of the buffer. If the selected value of queue_size is small, then there will be more packet losses as expected. Because the losses due to buffer overflow will increase.

In queue_size simulations, the bnd_delay is taken as 8 seconds for different queue_size values. Also the used algorithm to examine the effect of queue_size on T1 and T2 is DPD-4.

In Tables A.5, A.6, and A.7, the loss ratios table is given when queue_size is equal to 25, 50 and 75 in order. In each table, the best T1 and T2 values and range of T1 and T2 values for best loss ratios are defined. With the results of simulations, it is seen that when the queue_size increases, the range for selecting T1 and T2 values decreases.

The Figures A.5, A.6 and A.7 show the corresponding loss ratios for different T1 and T2 values with a specific queue_size value with 3-d graphs.

Table A.5 Total loss ratios for different T1 and T2 values when queue_size = 25

lossratio =

QUEUE SIZE = 25

T1 \ T2	0	0.5	1.0	1.5	2.0	2.5	3.0	3.5
0	1.3915	1.3901	1.3469	1.3790	1.3767	1.3561	1.4031	1.3920
0.1	0	1.3349	1.3378	1.2924	1.3566	1.3427	1.3916	1.3577
0.2	0	1.3245	1.3492	1.3565	1.3360	1.3197	1.3505	1.3548
0.3	0	1.3040	1.3122	1.3109	1.3505	1.3353	1.3504	1.3349
0.4	0	1.3304	1.3386	1.3338	1.3651	1.3250	1.3635	1.3571
0.5	0	1.2674	1.2990	1.3497	1.3076	1.3231	1.3804	1.3105
0.6	0	0	1.3323	1.3766	1.3338	1.3654	1.3604	1.3401
0.7	0	0	1.3529	1.3402	1.3530	1.3547	1.3106	1.3746
0.8	0	0	1.3233	1.3719	1.3393	1.3482	1.3733	1.3794
0.9	0	0	1.3328	1.3771	1.3328	1.3650	1.3609	1.3414
1.0	0	0	1.3516	1.3390	1.3527	1.3541	1.3101	1.3745
1.1	0	0	0	1.3289	1.3782	1.3455	1.3535	1.3798
1.2	0	0	0	1.3566	1.3363	1.3196	1.3510	1.3573
1.3	0	0	0	1.3147	1.3251	1.3230	1.3642	1.3477
1.4	0	0	0	1.3346	1.3181	1.3529	1.3610	1.3571
1.5	0	0	0	1.3591	1.3201	1.3575	1.3519	1.3060
1.6	0	0	0	0	1.3109	1.3644	1.3197	1.3352
1.7	0	0	0	0	1.3681	1.2981	1.3657	1.3516
1.8	0	0	0	0	1.3585	1.3496	1.3453	1.3596
1.9	0	0	0	0	1.3489	1.4025	1.3732	1.4135
2.0	0	0	0	0	1.3444	1.3905	1.3463	1.3773
2.1	0	0	0	0	0	1.3580	1.3379	1.3814
2.2	0	0	0	0	0	1.3538	1.3651	1.3661
2.3	0	0	0	0	0	1.3059	1.3686	1.3549
2.4	0	0	0	0	0	1.3843	1.3546	1.3623

Loss ratio limit	T1 and T2 range	The best case
1.31 %	0.1 < T1 < 2.4 0.5 < T2 < 3.5	T1=0.5 T2=0.5

In the simulations, it is observed that queue_size value only determines the lower and upper limit of T1 and T2 value for low losses. Therefore, after that limits, if T1 and T2 value increase, the loss ratio will also increase. Another important result is; if queue_size increases the range to select best T1 and T2 values will be smaller.

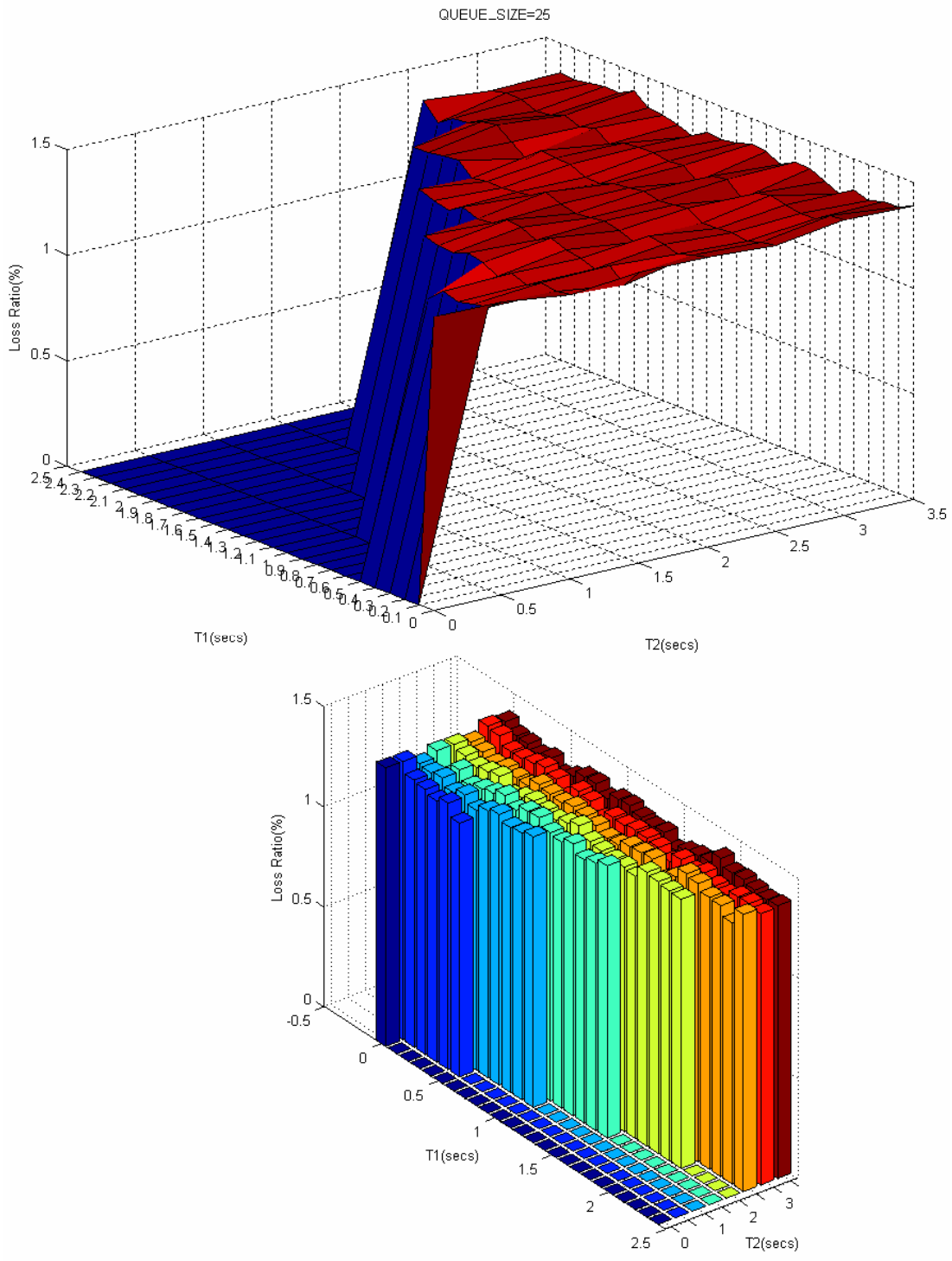

Figure A.5 Total loss ratios for different T1 and T2 values when queue_size = 25

Table A.6 Total loss ratios for different T1 and T2 values when queue_size = 50

lossratio =

QUEUE SIZE=50

T1 \ T2	0	0.5	1.0	1.5	2.0	2.5	3.0	3.5
0	0.6930	0.6403	0.6354	0.6689	0.6852	0.7202	0.6829	0.7015
0.1	0	0.6019	0.6574	0.6315	0.6964	0.6448	0.6774	0.6726
0.2	0	0.6046	0.6093	0.6457	0.6804	0.6314	0.6677	0.6999
0.3	0	0.6448	0.6124	0.6252	0.6581	0.6768	0.6740	0.6856
0.4	0	0.5891	0.6278	0.6640	0.6604	0.6492	0.7005	0.6867
0.5	0	0.6322	0.6372	0.6178	0.6157	0.6499	0.6443	0.7046
0.6	0	0	0.6260	0.6613	0.6761	0.6651	0.7015	0.7014
0.7	0	0	0.6273	0.6318	0.6807	0.7000	0.6443	0.6769
0.8	0	0	0.6373	0.6502	0.6575	0.6602	0.6542	0.7085
0.9	0	0	0.6346	0.6232	0.6371	0.6405	0.6637	0.6728
1.0	0	0	0.6446	0.6394	0.6415	0.6635	0.6606	0.6695
1.1	0	0	0	0.6311	0.6564	0.6708	0.6877	0.6967
1.2	0	0	0	0.6342	0.6755	0.6872	0.6700	0.6834
1.3	0	0	0	0.6732	0.6479	0.6549	0.6857	0.7287
1.4	0	0	0	0.6504	0.6902	0.6633	0.6814	0.7177
1.5	0	0	0	0.6438	0.6899	0.6702	0.7103	0.6729
1.6	0	0	0	0	0.6683	0.6689	0.6856	0.6796
1.7	0	0	0	0	0.6519	0.6599	0.6532	0.7012
1.8	0	0	0	0	0.6571	0.6594	0.7004	0.7226
1.9	0	0	0	0	0.6617	0.6708	0.6879	0.6814
2.0	0	0	0	0	0.6516	0.6687	0.6836	0.6848
2.1	0	0	0	0	0	0.6661	0.7205	0.7223

Loss ratio limit	T1 and T2 range	The best case
0.62%	0.1 < T1 < 0.5 0.5 < T2 < 2.0	T1=0.4 T2=0.5

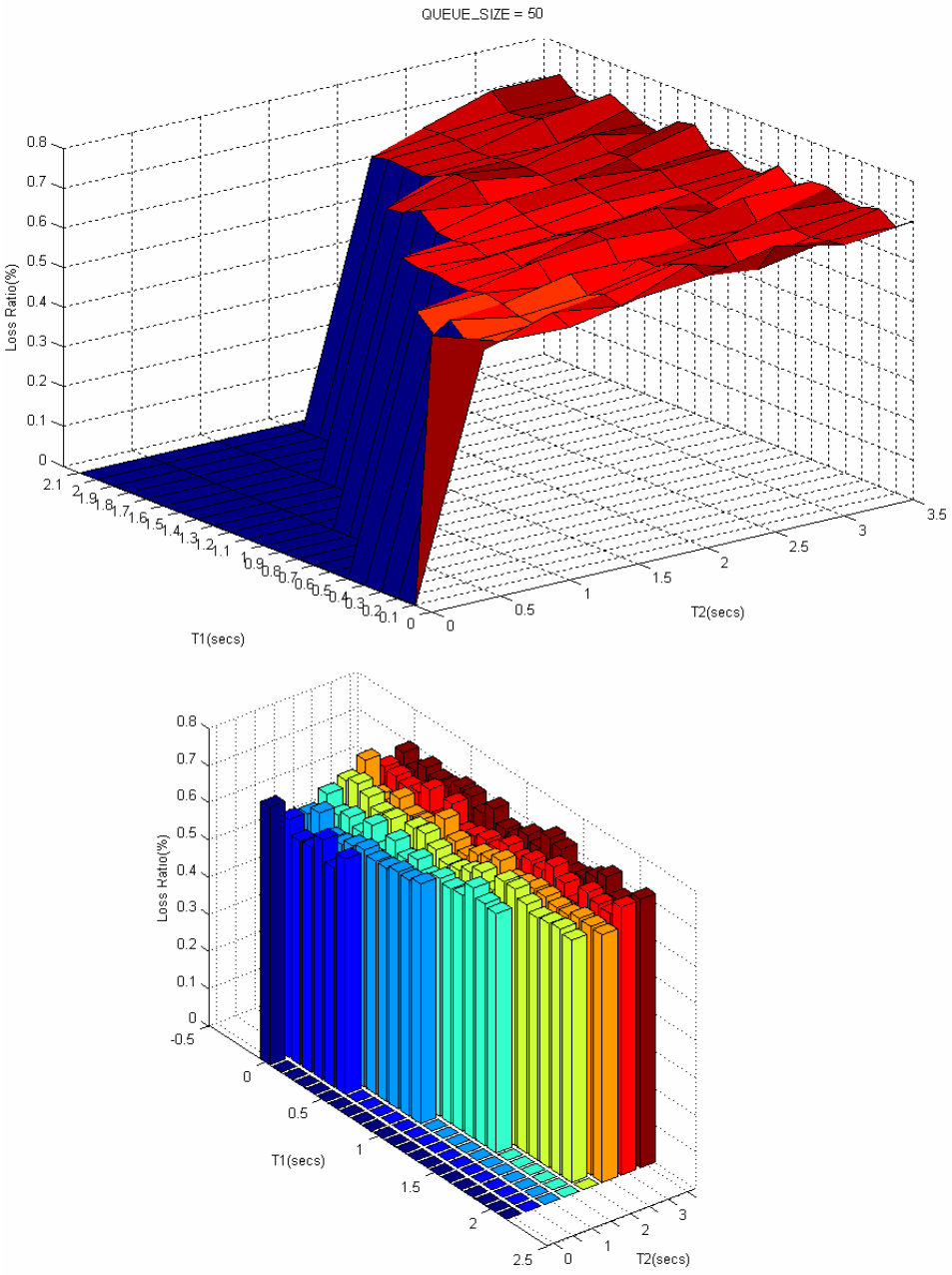

Figure A.6 Total loss ratios for different T1 and T2 values when queue_size = 50

Table A.7 Total loss ratios for different T1 and T2 values when queue_size = 75

lossratio =

QUEUE_SIZE=75

T1 \ T2	0	0.5	1.0	1.5	2.0	2.5	3.0	3.5
0	0.5835	0.5456	0.5227	0.5583	0.5814	0.5868	0.6166	0.6230
0.1	0	0.5018	0.5225	0.5143	0.5499	0.5569	0.5890	0.5841
0.2	0	0.4967	0.5196	0.5454	0.5411	0.5547	0.5618	0.5964
0.3	0	0.4777	0.5159	0.5249	0.5533	0.5742	0.5613	0.5882
0.4	0	0.5082	0.5226	0.5300	0.5470	0.5556	0.5930	0.5860
0.5	0	0.5193	0.4944	0.5292	0.5520	0.5563	0.5858	0.5919
0.6	0	0	0.5188	0.5405	0.5336	0.5688	0.5744	0.6071
0.7	0	0	0.5087	0.5329	0.5571	0.5792	0.5767	0.6080
0.8	0	0	0.5289	0.5524	0.5294	0.5643	0.5861	0.5901
0.9	0	0	0.5244	0.5296	0.5516	0.5751	0.5678	0.6006
1.0	0	0	0.5135	0.5417	0.5431	0.5659	0.5896	0.6136
1.1	0	0	0	0.5309	0.5438	0.5548	0.5865	0.5798
1.2	0	0	0	0.5363	0.5478	0.5748	0.5936	0.5846
1.3	0	0	0	0.5311	0.5620	0.5764	0.5846	0.6007
1.4	0	0	0	0.5301	0.5663	0.5613	0.5651	0.5852
1.5	0	0	0	0.5380	0.5366	0.5640	0.5947	0.5782
1.6	0	0	0	0	0.5551	0.5588	0.5938	0.6026
1.7	0	0	0	0	0.5490	0.5698	0.5883	0.6284
1.8	0	0	0	0	0.5701	0.5937	0.5703	0.6070
1.9	0	0	0	0	0.5625	0.5665	0.5945	0.6018
2.0	0	0	0	0	0.5620	0.5817	0.5759	0.6097
2.1	0	0	0	0	0	0.5710	0.6043	0.6015

Loss ratio limit	T1 and T2 range	The best case
0.50%	0.2 < T1 < 0.5 0.5 < T2 < 1.0	T1=0.3 T2=0.5

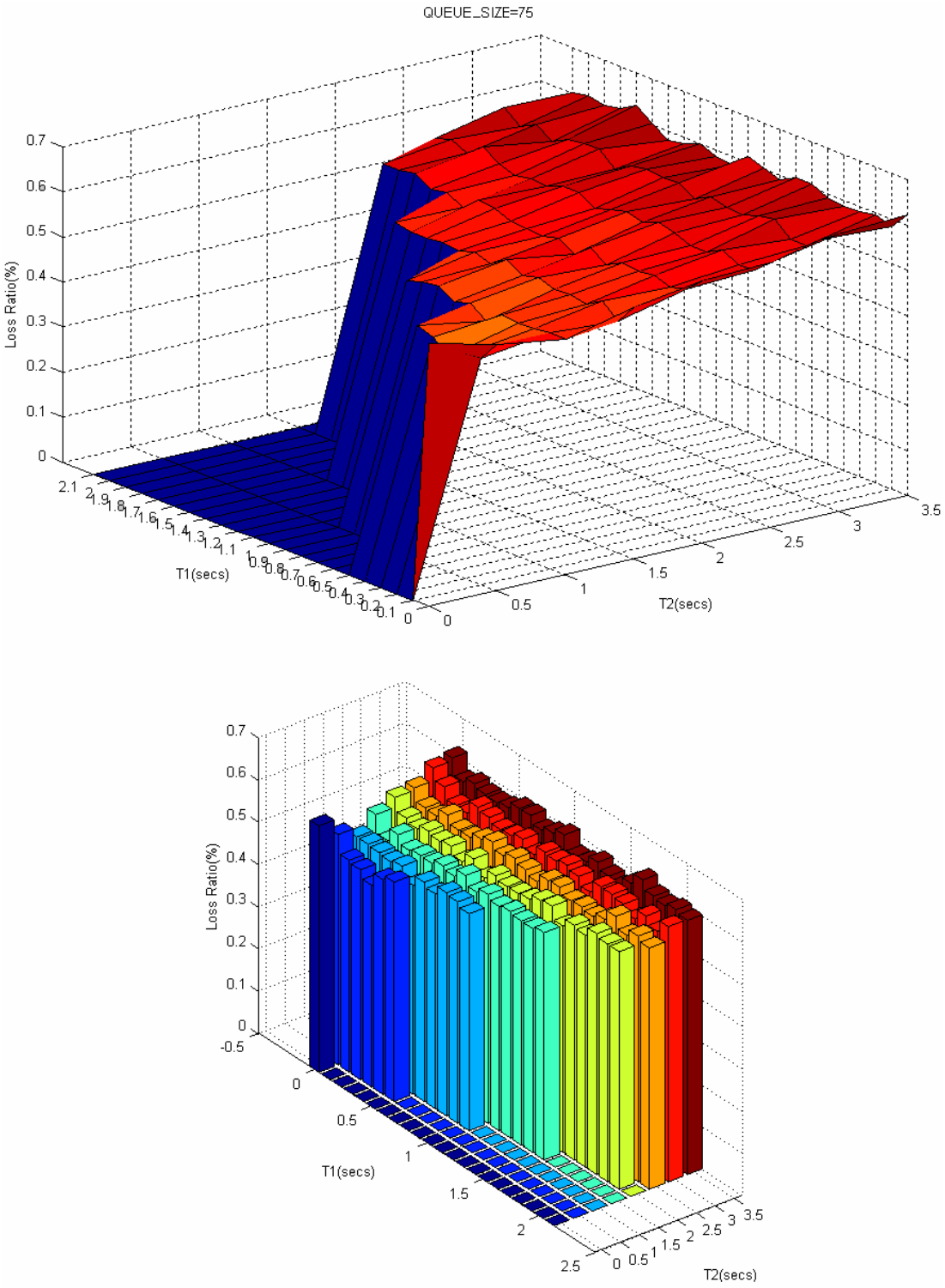

Figure A.7 Total loss ratios for different T1 and T2 values when queue_size = 75

APPENDIX B

SPD AND DPD SIMULATIONS' MATLAB CODES

All other programming codes that are not included here can be found in the CD that is included with the thesis.

SPD.m

```
%50000 sim time
% SP SIMULATION
clear all;
 TC = 10000;
 Avg_Pckt_Size = 100;
 ad=1;
 bnd_delay = 8;
 maxx=0;
 avgx=0;
 k=0;
 lambda = 98;
 mu = TC / Avg_Pckt_Size;
 Sim_time =50000;
 t1=clock;
 for ii=1:1:5
       n = 0 + ii*10 ; %Bu queue size limit
       for jj=1:4
        klmn=[2 4 7 n];
        sort2=klmn(jj);
        t0=clock;


% SIMULATION PARAMETERS
% Lambda -> packet arrival rate
% Mu -> Packet Service Rate
% n -> The Buffer Size
% The transmisson Capacity is 10 kbps
```

```
        if(mu < lambda)
           disp('Packet Arrival Rate should not be greater than Packet Service Rate')
           return;
        end

% The information about the next packet will be kept in these parameters

        next_pckt_arrvl_time = 0;
        next_pckt_serv_time = 0;

% The current time is set to 0 @ the beginning

        curr_time = 0;

        queue_size = 0;    %MAX n is the limit of the Buffer

        nbr_type_arr = zeros(10);
        nbr_type_loss = zeros(10);
        nbr_type=zeros(10);
        nbr_type_waiting=zeros(10);


        outofloss=0;
        deadlineloss=0;

        nbr_type1_waiting=0;
        nbr_type11=0;
        packetsize= 100;

           randvar=floor(rand(10,1)*10)+1;
            priorityofpacket=randvar(1);
            packet1 = struct('Size', packetsize, 'Arrival_Time', curr_time + exprnd(1 / (lambda/ad)), 'deadline', -1, 'type',
priorityofpacket, 'priority',priorityofpacket);
            packet1.deadline = packet1.Arrival_Time + exprnd(bnd_delay);

            f_queue(1) = packet1;
            f_queue_size=1;


           packet = f_queue(1);
           f_queue_size = f_queue_size - 1;



           next_pckt_arrvl_time = f_queue(1).Arrival_Time;
           next_pckt_serv_time = packet.Arrival_Time;

           curr_time = packet.Arrival_Time;
```

```
        event = 'Packet_Arrival';

% Until the simulation finishes

while (curr_time < Sim_time)

   switch event

% The Packet Arrival event. When a packet arrives, enter to this case

      case 'Packet_Arrival'

                        randvar=floor(rand(10,1)*10)+1;
                        priorityofpacket=randvar(1);
                        packet1 = struct('Size', packetsize, 'Arrival_Time', curr_time + exprnd(1 / (lambda/ad)), 'deadline', -1,
'type', priorityofpacket, 'priority', priorityofpacket);
                        packet1.deadline = packet1.Arrival_Time + exprnd(bnd_delay);
                        f_queue_size = f_queue_size + 1;
                        f_queue(f_queue_size) = packet1;
                        nbr_type_arr(packet1.type) = nbr_type_arr(packet1.type) + 1;

              %--------SORTING ALGORITHM according to arrival times
              %for f_queue
                    if (f_queue_size > 2)
                     for i=1:f_queue_size
                        for j=1:f_queue_size-1
                          if (f_queue(j).Arrival_Time > f_queue(j+1).Arrival_Time)
                              hold = f_queue(j);
                              f_queue(j) = f_queue(j+1);
                              f_queue(j+1) = hold;
                          end
                        end
                     end

                    elseif (f_queue_size == 2)
                        j=1;

                        if(f_queue(j).Arrival_Time > f_queue(j+1).Arrival_Time)
                              hold = f_queue(j);
                              f_queue(j) = f_queue(j+1);
                              f_queue(j+1) = hold;
                        end
                    end
              %----------------------------------------------------



            if (queue_size == 0)
                  queue_size = 1;
```

```
                queue(queue_size) = packet;

                if (packet.Arrival_Time > next_pckt_serv_time)
                    next_pckt_serv_time = packet.Arrival_Time;
                end

            elseif (queue_size < n)
                queue_size = queue_size + 1;
                queue(queue_size) = packet;

            elseif (queue_size == n)
                queue_size = n;
                outofloss=outofloss+1;
                nbr_type_loss(packet.type)=nbr_type_loss(packet.type)+1;

            elseif (queue_size > n)
                disp('Error#1 -> Exceed the Buffer Size')
                return;
            end

% Set the next packet arrival times

        packet = f_queue(1);
        f_queue = f_queue(2:f_queue_size);
        f_queue_size = f_queue_size - 1;

        next_pckt_arrvl_time = packet.Arrival_Time;

% The Packet Serving event. When a packet will be served, enter to this case

    case 'Packet_Serving'

        i=1;

    % This section begins from the first element and goes up to the element
    % which is greater than the current time

        if queue_size > 0
            while(i <= queue_size & ((queue(i).deadline) < curr_time))
                deadlineloss=deadlineloss+1;
                nbr_type_loss(queue(i).type) = nbr_type_loss(queue(i).type) + 1;
                nbr_type_waiting(queue(i).type) = nbr_type_waiting(queue(i).type) + curr_time - queue(i).Arrival_Time;
                i = i + 1;
            end
        end

    % The queue will be shorten by eliminating the loss packets due to their deadlines
```

```
            queue = queue(i:queue_size);
            queue_size = queue_size - (i - 1);

  if (  queue_size >= 2 )

              %--------SORTING ALGORITHM according to priorities
                        if(sort2<=queue_size)
                            kgt=sort2;
                        else
                            kgt=queue_size;
                        end

                          for i=1:kgt-1
                                for j=1:kgt-1
                                    if (queue(j).priority > queue(j+1).priority)
                                        hold=queue(j);
                                        queue(j)=queue(j+1);
                                        queue(j+1)=hold;
                                    end
                                end
                          end
            %Dynamic Priority Changing

      pckt_served = queue(1); % Sort ettikten sonra ilk paketi al yoruz

end


 if (queue_size==1)
    pckt_served = queue(1);
 end

       % The control. It is may be impossible, but control is a must

            if (queue_size < 0)
               queue_size = 0;
            end

       % Number of lost elements has to be added to the queue

          if (queue_size > 0)
            queue = queue(2:queue_size);
            next_pckt_serv_time = curr_time + pckt_served.Size / TC;
            nbr_type_waiting(pckt_served.type) = nbr_type_waiting(pckt_served.type) + curr_time - pckt_served.Arrival_Time;
            nbr_type(pckt_served.type)=nbr_type(pckt_served.type)+1;

            queue_size = queue_size - 1;

                  if (queue_size < 0)
```

```
                    queue_size = 0;
                end

        else
            next_pckt_serv_time = next_pckt_arrvl_time;
        end

    end    %End of Switch

% To decide which Event will occur

curr_time = min([next_pckt_arrvl_time, next_pckt_serv_time]);

    if(curr_time == next_pckt_arrvl_time)
        event = 'Packet_Arrival';
    else
        event = 'Packet_Serving';
    end

end  %End of While

%If any packet remains in f_queue, these are considered as loss packets%
while (f_queue_size > 0)
    i = 1;
            nbr_type_loss(f_queue(i).type) = nbr_type_loss(f_queue(i).type) + 1;
    i = i + 1;
    f_queue=f_queue(i:f_queue_size);
    f_queue_size = f_queue_size - 1;
end

disp(sprintf('------------SPD--------------'))
disp(sprintf('------------ SORT DEGREE= %d----------BUFFER SIZE= %d\n',sort2,n))
totalpacket1=sum(nbr_type_arr);
totalpacket=totalpacket1(1);
totallost1=sum(nbr_type_loss);
totallost=totallost1(1);
lost1=sum(nbr_type_loss);
lost=lost1(1);
waiting1=sum(nbr_type_waiting);
waiting=waiting1(1);
totalwaiting1=sum(nbr_type);
totalwaiting=totalwaiting1(1);

disp(sprintf('   Number Of All Sent Packets  : %d', totalpacket))
disp(sprintf('   Number Of All Lost Packets  : %d - %f \n', lost,(totallost/totalpacket)*100))
disp(sprintf('   Because of Queue Limit  : %d ----  %f\n', outofloss,(outofloss/totallost)*100))
disp(sprintf('   Because of Deadline Limit  : %d ----  %f\n', deadlineloss,(deadlineloss/totallost)*100))
disp(sprintf('   Percentage Of Loss ==>      : %.2f\n',(totallost/totalpacket)*100))
lossratio(ii,jj)=(totallost/totalpacket)*100;
```

```
deadlost(ii,jj)=(deadlineloss/totallost)*lossratio(ii,jj);
outlost(ii,jj)=(outofloss/totallost)*lossratio(ii,jj);
avgwaiting(ii,jj)=waiting/totalwaiting;

disp(sprintf('   Duration of Simulation : %d mins %.1f secs\n', floor(etime(clock, t0)/60), mod(etime(clock,t0),60) ) )
disp(sprintf(' Avg waiting for a packet in the queue: %f\n' ,waiting/totalwaiting))


for i=1:10
  l(ii,jj,i)=(nbr_type_loss(i)*100)/totalpacket;
  t(ii,jj,i)=nbr_type(i);
  w(ii,jj,i)=nbr_type_waiting(i)/nbr_type(i);
end
save 'SPD.mat'
end
end
disp(sprintf('   Duration of Simulation : %d mins %.1f secs\n', floor(etime(clock, t1)/60), mod(etime(clock,t1),60) ) )
```

DPD.m

```
%50000 sim time, T1=0.2
% DPD SIMULATION
clear all;
 TC = 10000;
 Avg_Pckt_Size = 100;
 ad=1;
 bnd_delay = 8;
 maxx=0;
 avgx=0;
 k=0;
 lambda = 98;
 mu = TC / Avg_Pckt_Size;
 Sim_time =50000;
 t1=clock;
%T1 and T2 are chosen as this.
 T2=1;
  for ii=1:1:5
      n = 0 + ii*10 ; %Bu queue size limit
      for jj=1:4
       T1=0.2;
       klmn=[2 4 7 n];
       sort2=klmn(jj);
       t0=clock;
% SIMULATION PARAMETERS
% Lambda -> packet arrival rate
% Mu -> Packet Service Rate
```

```
% n -> The Buffer Size
% The transmisson Capacity is 10 kbps

    if(mu < lambda)
        disp('Packet Arrival Rate should not be greater than Packet Service Rate')
        return;
    end

% The information about the next packet will be kept in these parameters

    next_pckt_arrvl_time = 0;
    next_pckt_serv_time = 0;

% The current time is set to 0 @ the beginning

    curr_time = 0;

    queue_size = 0;    %MAX n is the limit of the Buffer

    nbr_type_arr = zeros(10);
    nbr_type_loss = zeros(10);
    nbr_type=zeros(10);
    nbr_type_waiting=zeros(10);


    outofloss=0;
    deadlineloss=0;

    nbr_type1_waiting=0;
    nbr_type11=0;
    packetsize= 100;


        randvar=floor(rand(10,1)*10)+1;
        priorityofpacket=randvar(1);
        packet1 = struct('Size', packetsize, 'Arrival_Time', curr_time + exprnd(1 / (lambda/ad)), 'deadline', -1, 'type',
priorityofpacket, 'priority',priorityofpacket);
        packet1.deadline = packet1.Arrival_Time + exprnd(bnd_delay);

        f_queue(1) = packet1;
        f_queue_size=1;

       packet = f_queue(1);
       f_queue_size = f_queue_size - 1;

       next_pckt_arrvl_time = f_queue(1).Arrival_Time;
       next_pckt_serv_time = packet.Arrival_Time;

       curr_time = packet.Arrival_Time;
```

```
        event = 'Packet_Arrival';

% Until the simulation finishes

while (curr_time < Sim_time)

   switch event

% The Packet Arrival event. When a packet arrives, enter to this case

      case 'Packet_Arrival'

                        randvar=floor(rand(10,1)*10)+1;
                        priorityofpacket=randvar(1);
                        packet1 = struct('Size', packetsize, 'Arrival_Time', curr_time + exprnd(1 / (lambda/ad)), 'deadline', -1,
'type', priorityofpacket, 'priority', priorityofpacket);
                        packet1.deadline = packet1.Arrival_Time + exprnd(bnd_delay);
                        f_queue_size = f_queue_size + 1;
                        f_queue(f_queue_size) = packet1;
                        nbr_type_arr(packet1.type) = nbr_type_arr(packet1.type) + 1;

              %--------SORTING ALGORITHM according to arrival times
              %for f_queue
                    if (f_queue_size > 2)
                        for i=1:f_queue_size
                            for j=1:f_queue_size-1
                              if (f_queue(j).Arrival_Time > f_queue(j+1).Arrival_Time)
                                  hold = f_queue(j);
                                  f_queue(j) = f_queue(j+1);
                                  f_queue(j+1) = hold;
                              end
                            end
                        end

                    elseif (f_queue_size == 2)
                        j=1;

                        if(f_queue(j).Arrival_Time > f_queue(j+1).Arrival_Time)
                              hold = f_queue(j);
                              f_queue(j) = f_queue(j+1);
                              f_queue(j+1) = hold;
                        end
                    end
              %----------------------------------------------------



            if (queue_size == 0)
```

```
                queue_size = 1;
                queue(queue_size) = packet;

                  if (packet.Arrival_Time > next_pckt_serv_time)
                     next_pckt_serv_time = packet.Arrival_Time;
                  end

            elseif (queue_size < n)
                queue_size = queue_size + 1;
                queue(queue_size) = packet;

           elseif (queue_size == n)
              queue_size = n;
              outofloss=outofloss+1;
              nbr_type_loss(packet.type)=nbr_type_loss(packet.type)+1;

            elseif (queue_size > n)
              disp('Error#1 -> Exceed the Buffer Size')
              return;
            end

% Set the next packet arrival times


        packet = f_queue(1);
        f_queue = f_queue(2:f_queue_size);
        f_queue_size = f_queue_size - 1;

        next_pckt_arrvl_time = packet.Arrival_Time;

% The Packet Serving event. When a packet will be served, enter to this case

     case 'Packet_Serving'

         i=1;

      % This section begins from the first element and goes up to the element
      % which is greater than the current time

         if queue_size > 0
            while(i <= queue_size & ((queue(i).deadline) < curr_time))
               deadlineloss=deadlineloss+1;
               nbr_type_loss(queue(i).type) = nbr_type_loss(queue(i).type) + 1;
               nbr_type_waiting(queue(i).type) = nbr_type_waiting(queue(i).type) + curr_time - queue(i).Arrival_Time;
                i = i + 1;
            end
         end

      % The queue will be shorten by eliminating the loss packets due to their deadlines
```

```
            queue = queue(i:queue_size);
            queue_size = queue_size - (i - 1);

  if ( (queue_size > 0) && (queue(1).deadline-curr_time <= T1))
          pckt_served = queue(1); % Sort etmeden direk paketi al yoruz
  elseif ((queue_size > 0) && (queue(1).deadline-curr_time <= T2))
        queue(1).priority=queue(1).priority-1;
  end

  if (  (queue_size >= 2) && ( queue(1).deadline-curr_time > T1) )

              %--------SORTING ALGORITHM according to priorities
                        if(sort2<=queue_size)
                            kgt=sort2;
                        else
                            kgt=queue_size;
                        end

                          for i=1:kgt-1
                                for j=1:kgt-1
                                    if (queue(j).priority > queue(j+1).priority)
                                        hold=queue(j);
                                        queue(j)=queue(j+1);
                                        queue(j+1)=hold;
                                    end
                                end
                          end
            %Dynamic Priority Changing

     pckt_served = queue(1); % Sort ettikten sonra ilk paketi al yoruz

end

 if (queue_size==1)
    pckt_served = queue(1);
 end

      % The control. It is may be impossible, but control is a must

            if (queue_size < 0)
               queue_size = 0;
            end

      % Number of lost elements has to be added to the queue

     if (queue_size > 0)
            queue = queue(2:queue_size);
            next_pckt_serv_time = curr_time + pckt_served.Size / TC;
            nbr_type_waiting(pckt_served.type) = nbr_type_waiting(pckt_served.type) + curr_time - pckt_served.Arrival_Time;
```

```
            nbr_type(pckt_served.type)=nbr_type(pckt_served.type)+1;

            queue_size = queue_size - 1;

                if (queue_size < 0)
                   queue_size = 0;
                end

         else
            next_pckt_serv_time = next_pckt_arrvl_time;
         end
   end    %End of Switch

% To decide which Event will occur

curr_time = min([next_pckt_arrvl_time, next_pckt_serv_time]);

   if(curr_time == next_pckt_arrvl_time)
      event = 'Packet_Arrival';
   else
      event = 'Packet_Serving';
   end

end  %End of While

%If any packet remains in f_queue, these are considered as loss packets%
while (f_queue_size > 0)
   i = 1;
         nbr_type_loss(f_queue(i).type) = nbr_type_loss(f_queue(i).type) + 1;
   i = i + 1;
   f_queue=f_queue(i:f_queue_size);
   f_queue_size = f_queue_size - 1;
end

disp(sprintf('------------DPD modified DYNAMIC--------------'))
disp(sprintf('--T1=%3.2f--------T2=%3.2f------- SORT DEGREE= %d----------BUFFER SIZE= %d\n',T1,T2,sort2,n))
totalpacket1=sum(nbr_type_arr);
totalpacket=totalpacket1(1);
totallost1=sum(nbr_type_loss);
totallost=totallost1(1);
lost1=sum(nbr_type_loss);
lost=lost1(1);
waiting1=sum(nbr_type_waiting);
waiting=waiting1(1);
totalwaiting1=sum(nbr_type);
totalwaiting=totalwaiting1(1);

disp(sprintf('   Number Of All Sent Packets  : %d', totalpacket))
disp(sprintf('   Number Of All Lost Packets  : %d - %f \n', lost,(totallost/totalpacket)*100))
```

```
disp(sprintf('   Because of Queue Limit  : %d ----  %f\n', outofloss,(outofloss/totallost)*100))
disp(sprintf('   Because of Deadline Limit  : %d ----  %f\n', deadlineloss,(deadlineloss/totallost)*100))
disp(sprintf('   Percentage Of Loss ==>      : %.2f\n',(totallost/totalpacket)*100))
lossratio(ii,jj)=(totallost/totalpacket)*100;
deadlost(ii,jj)=(deadlineloss/totallost)*lossratio(ii,jj);
outlost(ii,jj)=(outofloss/totallost)*lossratio(ii,jj);
avgwaiting(ii,jj)=waiting/totalwaiting;

disp(sprintf('   Duration of Simulation : %d mins %.1f secs\n', floor(etime(clock, t0)/60), mod(etime(clock,t0),60) ) )
disp(sprintf(' Avg waiting for a packet in the queue: %f\n' ,waiting/totalwaiting))



for i=1:10
  l(ii,jj,i)=(nbr_type_loss(i)*100)/totalpacket;
  t(ii,jj,i)=nbr_type(i);
  w(ii,jj,i)=nbr_type_waiting(i)/nbr_type(i);
end
save 'DPD.mat'
end
end
disp(sprintf('   Duration of Simulation : %d mins %.1f secs\n', floor(etime(clock, t1)/60), mod(etime(clock,t1),60) ) )
```