The Virtual Frame Algorithm: A New Scheduling Technique for Integrated Services in Packet Switching Networks

by

Tarek Mahmoud El-Said El-Sayed Nasser
B.Sc., Ain-Shams University, Cairo, Egypt, 1990
M.Sc., Al-Azhar University, Cairo, Egypt, 1997

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

We accept this dissertation as conforming
to the required standard

Dr. F. Gebali, Supervisor (Department of Electrical and Computer Engineering)

Dr. P. Agathoklis, Departmental Member (Department of Electrical and Computer Engineering)

Dr. R. L. Kirlin, Departmental Member (Department of Electrical and Computer Engineering)

Dr. D. Olesky, Outside Member (Department of Computer Science)

Dr. H. Alnuweiri, External Examiner (Department of Electrical and Computer Engineering, University of British Columbia)

Supervisor: Dr. Fayez Gebali

Abstract

In this work, we propose a new scheduling algorithm (the virtual frame algorithm) for handling different types of traffic using an optimization technique based on the transportation problem with upper bounds. The proposed algorithm has independent control for bandwidth and delay performance, by which we are able to support both real and non-real time applications simultaneously. This algorithm has many attractive features such as protection, fairness, scalability, and minimum bandwidth guarantees. The proposed algorithm can work as a traffic shaper at the output of each switch, which reduces network congestion and data loss in the down stream.

The new algorithm was simulated and its performance was tested using MATLAB. Four types of traffic were simulated at the aggregation level and were used as input for testing the performance of the proposed algorithm. Each input traffic had its own quality of service requirements that expressed either a real or non-real time application.

A comparison was done between the proposed algorithm and some of the other well-known algorithms. It indicated that the proposed algorithm and those other algorithms under consideration were almost equal in providing the same quality of services but the proposed technique has the advantages of simple controllability as well as ease of implementation.

The main contribution of this work is introducing a totally new approach to model the scheduling problem in communication networks. The new algorithm provides a great amount of controllability embedded into the different degrees of freedom it possesses. The simplified model developed provides a very simple technique in the sense of complexity. It can be implemented either in software or hardware.

Examiners:

Dr. F. Gebali, Supervisor (Department of Electrical and Computer Engineering)

Dr. P. Agathoklis, Departmental Member (Department of Electrical and Computer Engineering)

Dr. R. L. Kirlin, Departmental Member (Department of Electrical and Computer Engineering)

Dr. D. Olesky, Outside Member (Department of Computer Science)

Dr. H. Alnuweiri, External Examiner (Department of Electrical and Computer Engineering, University of British Columbia)

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AF	Assured Forwarding
ATM	Asynchronous Transfer Mode
ATS	Asynchronous Time-Sharing
CBR	Constant Bit Rate
DRR	Deficit Round Robin
DS	Distributed Service
DSHB	Delay-Sensitive High Bandwidth
EDD-D	Earliest Due-Date for Delay
EDD-J	Earliest Due-Date for Jitter
FCFS	First Come First Serve
FDDI	Fiber Distributed Data Interface
FFQ	Frame Based Fair Queuing
FIFO	First-In First-Out
HRR	Hierarchical Round Robin
LAN	Local Area Network
LBAP	Linear Bounded Arrival Processor
LR	Latency-Rate
LRD	Long-Range Dependence
MAC	Medium Access Control
OR	Operations Research
PCT	Preemptive Cut Through
PGPS	Packet-by-Packet General Processor Sharing
PRR	Packet-by-Packet Round robin
QoS	Quality of Service
RCSP	Rate-Controlled Static Priority
RT-HBS	Real-Time High Bandwidth Services
SCFQ	Self-Clocked Fair queuing
SMS	Statistically Multiplexed services
SPFQ	Starting Potential Based Fair queuing

SRP	Resource Reservation Protocol
SRT	Smallest Response Time
TCP	Transportation Control Protocol
TDMA	Time Division Multiple-Access
TOS	Traffic Class Octet
TTRT	Target Token Rotation Time
URR	Urgency Round Robin
VBR	Variable Bit Rate
VC	Virtual Clock
WAN	Wide Area Network
WFQ	Weighted Fair Queuing
WRR	Weighted Round robin
WSS	Wide-Sense Stationary

Acknowledgments

I would like to express my deepest appreciation to my supervisor, Dr. Fayez Gebali for his thoughtful ideas that really shaped my work. I am very thankful for his precious time, valuable guidance, encouragement, patience, and his huge kindness from the first day I arrived to Victoria.

I am very grateful to the members of my committee, Dr. Agathoklis, Dr. Kirlin, and Dr. Olesky for their support. Their time and effort are highly appreciated. I wish to express my deepest appreciation to Dr. Alnuweiri for agreeing to be the external examiner in my oral examination.

I am greatly indebted to my parents not only now but for the whole of my life. They have always supported me morally and financially and never asked for anything in return.

I would like to express my deep thanks to my fiancé for her support, encourage, and true love that helped me a lot to pursue a Ph.D.

I would like to thank the department's system and office staff for their enormous support and their continuous cooperation. I am very thankful to Vicky Smith, Lynne Barrett, and Monica Bracken.

Special thanks to my friends who were always with me Dr. Ayman Tawfik, Mohamed Watheq El-Kharashi, Mohamed Yasein, Rafik Mikhael, and Yousry Abd-El-Hamied.

I dedicate this dissertation to my beloved Parents, Mahmoud Nasser, and Laila Lotfy.

Chapter 1

Introduction

1.1 Overview

Computer networks are in a transient stage, turning from relatively slow communication links and data-oriented services to high-speed networks and a diverse set of services. Many of these services, such as voice, video, and other applications, will have very restricted real-time constraints and will require not only high bandwidth, but a predictable "quality of service" (QoS) not offered by current best-effort services.

The large amounts of bandwidth promised by future high-speed networks may also give the possibility of integrating such real-time applications together with more traditional data-oriented services in a single unified network. Thus while the scaling of bandwidth to more than tens of gigabits per second in next generation networks will certainly have a profound effect on all aspects of networking, the need to support mixed services by accommodating the performance requirements of real-time applications raises essential issues beyond just bandwidth or bandwidth-delay product scaling [1].

Traditional communication network applications such as file transfer, electronic mail, and remote log-in are examples of non-real time applications, for which the performance metrics of interest are typically average message/packet delay and throughput [2].

The characteristics of real-time communication applications are different from those of the non-real time. The distinctive feature of real-time communication is the fact that the value of the communication depends upon the times at which messages are successfully delivered to the recipient. Typically the desired delivery time for each message across the network is bound by maximum delay or latency, resulting in a deadline being associated with each

message. This delay is an application-layer end-to-end delay constraint. If a message arrives at the destination after its deadline has expired, its value to the application may be greatly reduced. In some circumstances messages are considered "perishable", that is, are useless to the application if delayed beyond the deadline. In effect, the messages are discarded and considered lost. However, some real-time applications do not require early arrival that may be even considered harmful, as it requires buffering at the receiver to achieve constant end-to-end delay [3].

Real-time applications are usually classified as either soft or hard real-time. Soft-real-time applications can tolerate some number of lost messages, while hard-real-time applications have zero loss tolerance. The protocols needed to handle traffic for these two kinds of applications may differ dramatically. In general, soft-real time applications require less stringent services and allow for maximum network utilization.

Another important performance metric for real–time traffic is delay jitter, commonly defined as the maximum variation in delay experienced by packets in a single connection. For example, if the minimum end-to-end delay is 1ms and the maximum delay is 6 ms, then the delay jitter is 5 ms [4]. Many real-time applications, particularly interactive ones, require a bound on jitter, in addition to bound delay. As we will see, some methods of real-time communication specifically manage the jitter, while other does not. Some applications such non-interactive television and audio broadcasting, demand bounds on delay jitter but not delay. The different performance metrics and reliability requirements of real-time traffic suggest that network protocols and architectures previously developed for data-oriented communication applications may not be suitable for supporting real-time and integrated real-time/non-real-time applications.

1.2 General goals for real-time communication techniques

All methods of real-time communication aim to provide real-time message delivery with either low or zero loss rates (soft or hard real-time, respectively) [5]. The following are some of the main required features for real-time communication:

- Efficient utilization of the network resources

- Achieving quality of service requirements for large networks and large numbers of connections
- Low jitter
- Low delay
- Integration with non-real-time services
- Adaptation to network and traffic conditions
- Moderate buffer requirements within the network
- Low processing overhead per packet or cell within the network

1.3 Characteristics of real time traffic

A large number of real-time communication applications are expected to be served in an integrated network. Some of these applications are multimedia conferencing, shared workspace, remote medical diagnosis, telephony, distributed interactive simulation, audio and video broadcasts, and games [6], [7].

The description of sources is made easier by the fact that in many real-time applications, the source of the data is a sensor, which samples a physical quantity to produce a digital signal. The sensor samples the physical quantity at regular intervals T and the data generated by the sensor are fed into the networks as a real-time stream [8], [9]. Many such sources can be approximated by one of the following three source models, as shown in Figure 1.1:
Constant bit rate (CBR): Fixed size packets arrive at deterministic intervals. Certain real-time applications, such as air traffic control, generate data which have few redundancies and which are too important to be compressed in a lossy way.

Variable bit rate (VBR): On/Off: The source alternates between a period in which fixed-size packets arrive with deterministic spacing between idle periods.

Periodic with variable packet sizes: In each period, the source submits a single packet of variable length to the network. An example is video; different frames may experience varying compression ratios for the same quality of service.

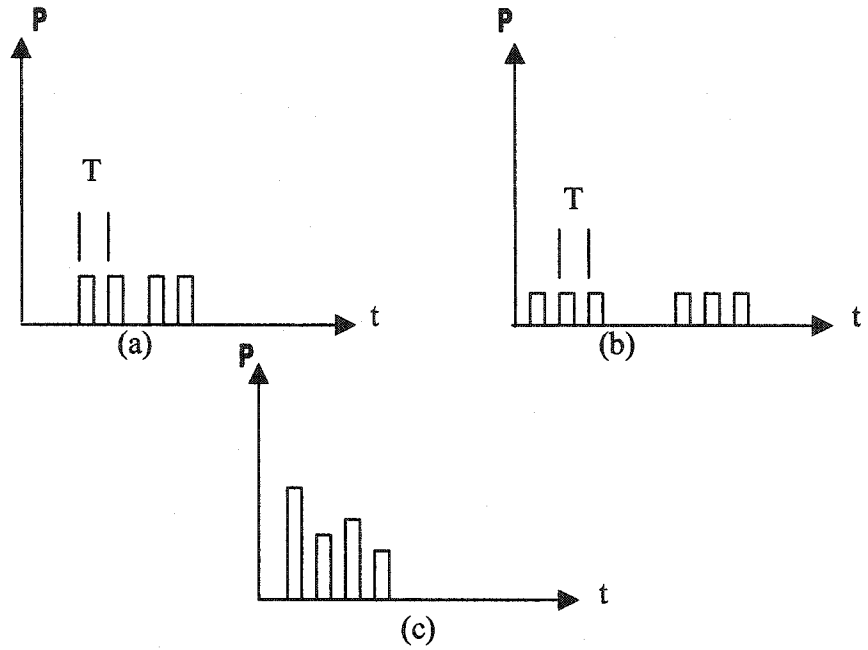

Figure 1.1 Packet arrivals from (a) continuous data sources, (b) voice sources with silence interval detection and (c) compressed video source (P = packet size, t = time, T = packet interval time).

1.4 Quality of service requirements for different applications

In this section, we describe a suitable way to classify traffic types in high-speed networks from the quality of service (QoS) point of view. A variety of applications with different requirements will be supported in an integrated fashion in broadband networks [10], [11]. For this reason it is very helpful to categorize traffic at the application layer as follows:

Type 1: This traffic is divided into two types (1A and 1B) of real-time high bandwidth services (RT-HBS).

Type 1A: Delay-sensitive isochronous high bandwidth services (DSHB). Isochronous means that this call requires a fixed high bandwidth for the duration of the call. Examples of this traffic are constant bit rate (CBR) conference video and other real-time high bandwidth CBR service.

Type 1B: Delay-sensitive non-isochronous high bandwidth service. This traffic is also RT-HBS type but it is non-isochronous in that each call alternates between active and inactive periods, and generates a high-bandwidth data stream during the active periods. Examples of this type of traffic are variable bit rate (VBR) video and LAN-LAN interconnects. This

traffic is further divided into sub-classes. For example, VBR video and LAN-LAN calls will be separately grouped for admission control, bandwidth allocation, and buffer management.

Type 2: Delay-insensitive high bandwidth services. These are essentially non real-time bulk information transport services. The user may specify the extent of delay that is tolerable; specified in the units of minutes, hours, or as over night delivery. There are different alternative overlay methods for appropriately serving this type of traffic. Essentially the network transports this type of traffic during slack periods. Hence, it calls for suitable ways of storing the bulk information, identifying network slack periods, and scheduling of the transmission. Examples of this traffic type are document images, video delivery services.

Type 3: Low-bandwidth statistically multiplexed services (SMS). Calls of this type are delay sensitive with end-to-end delay requirements ranging from a few ten milliseconds to a few hundreds of milliseconds. Examples of such traffic are packetized voice, inter-active data, and inquiry-response messages. All the previous traffic types requirements are shown in Figure 1.2 [11].

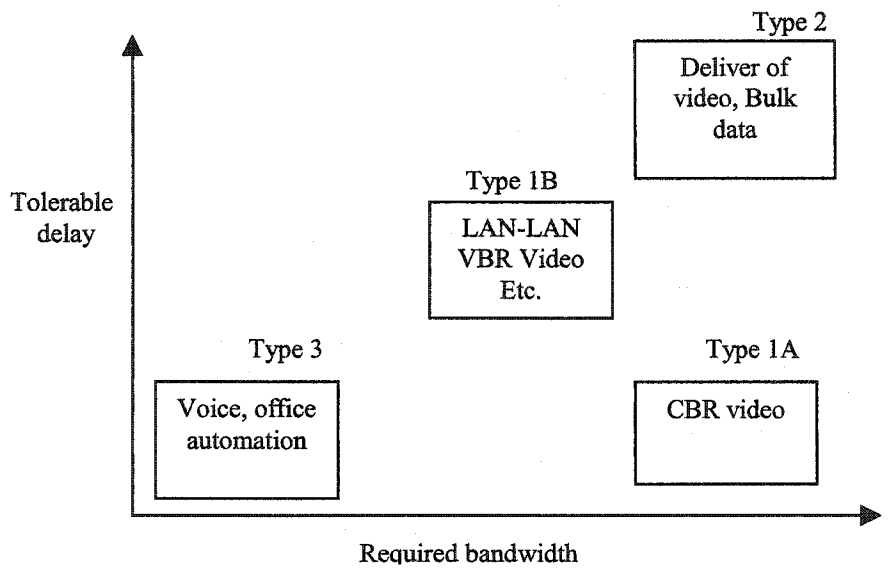

Figure 1.2 Traffic types and their delay-bandwidth requirements [11].

1.5 Thesis Objectives

The overall objectives of this research are: to investigate the techniques used for scheduling in communication networks, to develop a new scheduling technique that is efficient and simpler to implement, to evaluate the performance of the new technique, and to compare it to the well-known schedulers.

1.6 Motivations and scope of the thesis

After surveying most of the previous work related to scheduling in communication networks, we classified most algorithms based on one of the following: (1) priority-based schemes, (2) delay-based schemes, (3) bandwidth-based schemes, and (4) bandwidth delay product schemes. The above-mentioned schemes are not capable of providing the required quality of service for all different traffic classes due to conflicting requirements. An optimization model should be used to develop a scheduler that can handle the conflicting requirements of delay, bandwidth, and loss probability.

In this thesis we choose the transportation problem model to develop a new scheduler for the following reasons: (1) The word "transportation" itself implicitly has the meaning of transferring or the meaning of switching which absolutely suites our case, (2) It has the required priority scheme in an optimized way, and (3) The simplicity of the solution methodology of such model. It is mentioned in the literature that the solution of this problem for small-scale problems (10-15 variables) can be done by a pencil and paper.

1.7 Thesis outline

Chapter two gives the basic information about scheduling in communication networks. The main principles of design as well as the important requirements needed from the scheduler are presented.

Chapter three provides a comprehensive survey about the most well known schedulers used in the area of communication. This chapter is divided into four main parts. The first part gives the common ways of classifying the schedulers. Part two is a general overview about

the different approaches and directions used in this area. A detailed discussion about the well-known schedulers used in WANs is presented in part three. Schedulers used in LANs are introduced brief by in part four.

Chapter four presents the basic background about the transportation problem model and its mathematical terminologies used in the literature. The transportation problem with upper bounds is discussed. The analogy between the transportation problem model and the scheduling problem in the communication networks is introduced. At the end of this chapter, the proposed algorithm as well as its final form is presented.

Chapter five discusses the simulation results obtained from the simulator and gives insight about the new technique in the sense that it can treat different situations very efficiently. Also, it explains all the other nice features of the new proposed algorithm. Comparisons between some of the well-known schedulers and the proposed algorithm are done at the end of the chapter.

Chapter six gives conclusion and insight into the main contributions of the thesis. Also, some of the ideas about future work are discussed.

Appendix A gives some details about the stochastic processes and how the traffic presented in communication networks can be modeled using these stochastic processes. At the end of this appendix we give details about the model used for traffic generation in our simulation. Appendix B presents the MATLAB source code used for building the simulator.

Chapter 2

Schedulers design considerations

2.1 Introduction

Scheduling problems in general describe the phenomena of having a large number of entities requesting the same resource. Schedulers are responsible for handling these requests. This large number of entities is most likely forming a queue just ahead waiting to fulfill its request. The formation of waiting lines is, of course, a common phenomenon that occurs whenever the current demand for a service exceeds the current capacity to provide that service. In the following sections we discuss the scheduling problem in communication networks as well as the necessary designing considerations.

2.1.1 Computer communication networks and scheduling

Computer networks allow users to share (or multiplex) resources such as printers, file systems, long-distance trunks, and sites on the World Wide Web. Sharing, however, automatically introduces the problem of accessing the shared resources. Given a set of resource requests in the service queue, a server uses a scheduling discipline to decide which request to serve next. Scheduling disciplines are important because they are the key to fairly sharing network resources and to supporting performance of different applications, such as telephony and interactive games.

Some real-time applications require a guaranteed maximum delay and cannot tolerate any packet loss. As an example consider a distributed process control system. In such a system, a message that indicates a reactor vessel is about to exceed its pressure limit must be received in time. Likewise, a response message that indicates the appropriate safety method to be taken must be guaranteed a successful and timely delivery. A lost or late message in either

case could be catastrophic. Hard real-time applications are thus intolerant to packet loss or delay [12].

The theory of real-time scheduling has been developed and applied primarily to scheduling of jobs on a main frame as a shared resource. For real-time communication, the link corresponds to the main frame as the shared resource, while packets are the units of work requiring these resources, just as jobs must compete for use of the main frame. With this analogy, most real-time scheduling methods are immediately applicable to the scheduling of packets on a link. This discipline may be optimized for uniformity as in Round Robin (RR) [13], for simplicity as in first come first serve (FCFS), or for several other criteria as in priority-based schedulers [14].

To complete the scope of scheduling in communication networks, we define the schedulable region of a system as the set of points in the space of possible loads (traffic in the case of communication networks) for which the quality of service is guaranteed [15], [16], [17]. As such, this concept is a generalization of the concept of a stability calls region, that is, of the general concept of stability calls for finding the region in the space of loads for which the average time delay is finite [18], [19]. In our case, the set of constraints that determines the schedulable region is defined by the QoS constraints. Examples of such constraints include hard time delay constraints, loss probability, average throughput, and delay jitter. A schematic diagram for the schedulable region is given in Figure 2.1, assuming that the scheduler is serving three different classes of traffic.

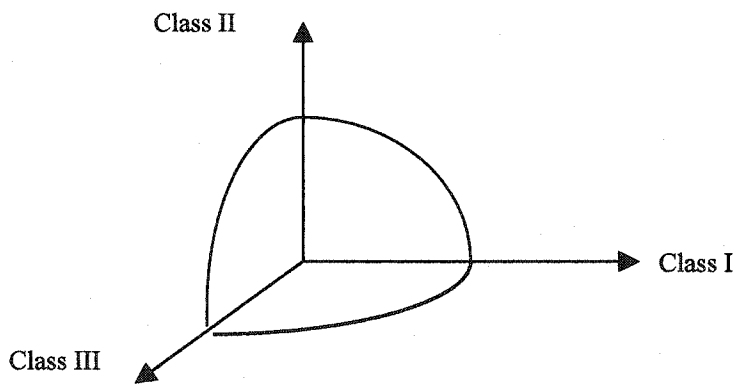

Figure 2.1 A schematic diagram for a three class schedulable region.

2.2 Design principles of schedulers

The main design principles that have to be considered in designing a scheduling policy [20] are:

- Priority levels.
- Work-conserving and non-work-conserving.
- Aggregation level of modeling.
- Service discipline for each priority level.

2.2.1 Priority scheme levels

In a priority-scheduling scheme, each connection is given a priority level. If there are n priority levels, and a higher-numbered priority level corresponds to a connection with higher priority, the scheduler serves a packet from priority level k only if there are no packets awaiting service in levels k+1,k+2, ...n. Priorities allow a scheduler to favor packets in a higher priority level, giving them quicker service than those packets with lower priority level. These priority levels can be set in a fixed manner or dynamic manner.

2.2.2 Work-conserving and non-work-conserving

A scheduler is a work-conserving type if it is idle only when there is no packet awaiting service. In contrast, a scheduler is termed a non-work-conserving if it is idle sometimes even if it has packets to serve. Different applications require different quality of service. According to the nature of the application, the designer decides which type is to be used. Work-conserving schedulers are mainly used to control the delay. On the other hand the non-work-conserving schedulers are used to control the delay jitter. The word controlling here simply means sometimes holding the packet or delaying the packet which is of course totally the opposite of quickly servicing the packet [21], [22].

2.2.3 The aggregation level of modeling

The third principle in the design of scheduling disciplines is the degree of aggregation to which individual connections are aggregated in setting their appropriate service order. At one extreme, the scheduler uses a single state variable to describe all connections, which must all, therefore, share the same quality of service. At the other extreme, the scheduler maintains a

per-connection state, and can give different connections different bandwidth and delay bounds. With an intermediate degree of aggregation, the scheduler treats all packets from connections in the same class the same way (that is, maintains per-class state). In this scheme, the scheduler provides different qualities of service to different classes, while connections within a class share the same service quality. Classes of service, therefore, are an intermediate way between per-connection service guarantees and single shared service guarantees [23], [24].

2.2.4 Service discipline for each priority level

The fourth and the final principle in designing scheduling disciplines is the order in which the scheduler serves packets from connections (or aggregates of connection) at the same priority level. There are two fundamental choices: serving packets in the order they arrive or serving them out of order according to a per-packet service tag. With the second option, the properties of the scheduler depend heavily upon how the scheduler computes the tag, and for example, whether or not each packet is serviced in an optimized way. Nevertheless, all the scheduling disciplines that serve packets in non-FCFS order need to sort packets implicitly or explicitly, which results in an implementation overhead [25], [26].

The main problem with FCFS is that it services the packets according to their buffering order in most cases. This policy does not allow packets that require a lower delay to skip to the head of the queue. In contrast, if we serve in order of service tags, we can give packets that require a low delay a lower tag value than others in the queue, which allows them to jump to the head of the queue. The service tags are implemented in many ways as in potential function trend or the other stamping mechanisms. A second problem with the FCFS service is regarding fairness. Fairness in this case for individual connections is not provided in its max-min sense. Connections receive service roughly in proportion to the speed at which they send data into the network. FCFS service encourages an endpoint to behave greedily, because greediness in this case is rewarded to the endpoint by simply allowing it to send more and more data in a faster rate than the other users. That is very oppressive for the other connections.

2.3 Scheduler requirements

An ideal scheduling discipline has to fulfill four important requirements. These four requirements are contradictory with each other. It is very difficult to achieve all of them simultaneously. The four-scheduler requirements are:

- Ease of implementation
- Fairness and protection
- Performance bounds.
- Help providing an efficient policy for admission control.

2.3.1 Ease of implementation

In high-speed networks, a scheduler must make a decision step in a very short time (in order of ms) to choose the next packet eligible for transmission. It is desired for the number of steps (operations) required for such decision is as few as possible. The discipline should be implementable inexpensively in hardware. If a switch is serving N simultaneous connections, a scheduler that takes O(N) time does not scale, and we prefer a scheduler that takes O(1) time. Wide area switches can serve up to 100,000 simultaneous connections. Thus, scaling is particularly important for wide area switches [27].

2.3.2 Fairness and protection

A scheduling discipline allocates a share of the link capacity and output queue buffers to each connection it serves. To give a solid definition for fairness, we call an allocation at a switch fair if the allocation satisfies a pre-set criterion. For example a well-known criterion called max-min is used for many protocols. This criterion can be stated in a simple form as follows: (a) resources are allocated in order of increasing demand (b) no connection gets a resource share larger than its demand (c) connections with unsatisfied demands get an equal share of the resource [28].

Protection means how the scheduler protects all connections from any misbehaving connection/connections so as not to affect the performance they are supposed to receive. To put the definition in a more precise way, we say that the scheduler provides protection if it

guarantees a minimum bandwidth to every individual connection (in fact it depends on the degree of aggregation) regardless of the behavior of any other connections. For example, a first come first serve (FCFS) discipline does not provide protection, because the mean delay of a source may increase if the sum of the arrival rates over all sources increases. Thus, a misbehaving source, by sending too fast, increases the mean delay of all other connections. In contrast, with Round Robin scheduling, a misbehaving source overflows its own queue, and the other sources are unaffected [29].

Fairness and protection are linked together in some sense. The relationship between fairness and protection is that a fair scheduler automatically provides protection, because it limits a misbehaving connection to its fair share. However, the converse needs not to be true. For example, if connections are policed at the entrance to the network (that is, the network forces them to conform to a pre-declared traffic pattern) they are protected from each other, but their resource shares may not be fair.

2.3.3 Performance bounds

The third main requirement of a scheduling discipline is that it would permit a network operator to guarantee arbitrary per-connection performance bounds. An operator can guarantee performance bounds for a connection only by reserving some network resources, either on-the fly, during the call establishment phase of the connection or in advance. Since the amount of resources reserved for a connection depends on its traffic intensity, guaranteed-performance connections must agree to limit their usage. The relation between the network operator and the user can be seen as a legal contract: the user agrees that its traffic will remain within certain limits, and, consequently the operator guarantees that the network will offer the connection's demands. Note that for the operator to meet the connection's demands it has to have full control through many schedulers in tandem. In heterogeneous networks this assumption is yet impossible, since different parts of the network may have different scheduling disciplines. Thus up to date guaranteeing end-to-end performance bounds is a hard problem, and an area of active research [27], [30].

There are main four performance bounds used in the literature. They may also be called the quality of service (QoS). These four bounds are the: bandwidth (throughput), delay, delay-jitter, and packet loss probability.

2.3.4 Help providing an efficient policy for admission control

A scheduling discipline should help in permitting easy admission control. A switch controller should be able to decide, given the current set of connections and the descriptor for new connections, whether it is possible to meet the new connection's performance bounds without disturbing any of the existing connections. Moreover, the scheduling discipline should not lead by any means to underutilizing the network resources. For example, with first come first serve (FCFS) discipline, we can give all connections a worst-case-delay bound by restricting the number of connections and the largest burst size that each connection may send. However, this typically results in an underutilizing the system resources [29], [31], [32].

Chapter 3

Survey of previous work

3.1 Introduction

In the scheduling area of research there are many definitions and acronyms, some of them sometimes have very close meaning. We try to make clear what is meant by these terms on the spot. And, it should not be confusing for the reader that sometimes the analysis of the subject is taken from certain point of view, while in some other places the analysis of the same subject is made from a different perspective.

This chapter is divided into four main parts. Part one presents the important ways of classifying schedulers in general (section 3.2). Part two gives some of the main directions and approaches that have been proposed in the scheduling area (sections 3.3 to 3.5). Part three focuses on the mathematical and implementation details of some of the well-known schedulers (sections 3.6 and 3.7), to compare and contrast these disciplines since they are closely related. In part four, a brief discussion of some of the techniques that have been implemented for the local area (LANs) networks is presented (sections 3.8 to 3.11). This final part is presented basically because the WAN that is our main concern is a large number of LANs connected together through the whole network. So we need some basic information about the LANs.

3.2 The most common scheduler classifications in communication networks

The most common ways used to classify schedulers in the literature are:

1) Prioritized and fair queuing schedulers
2) Latency-rate schedulers, rate-proportional schedulers, and traffic-reshaper schedulers
3) *Rate-based* schedulers and the *scheduler-based* schedulers.

3.2.1 Prioritized and fair queuing schedulers

In prioritized and fair queuing schedulers [21]-[23], [25], [30], [31], [33], [34] priorities may be designated by the end-user, or may be assigned according to some properties of the packet, such as the arrival period or deadline. In addition, priorities may be statically assigned for all packets in a connection, or may be assigned dynamically at the time of arrival of a packet. The scheduler may enforce priorities at the completion of the current transmission, or may elect to preempt an active transmission in favor of a newly arrived packet.

Prioritized queuing is generally done by sorting packets into different service classes and assigning delivery priorities, drop priorities, and so forth, to each service class. For instance, the assured forwarding (AF) specification is defined as a set of 12 service classes; four AF classes with three packet drop precedence each [21], [25]. A packet's class is indicated by the value found in the IP packet distributed service (DS) field (also known as the Type-of-Service [TOS] or Traffic Class Octet) in the IP header. An AF class based queuing system maps a packet's DS field onto AF class, which it uses as the key to order IP packet delivery.

The alternative, known as fair queuing, is designed to ensure that all users of a router are treated approximately equally. Without appropriate safeguards, an abnormally aggressive stream of packets coming from a single source can drown out more conservative sources, which will thus suffer abnormally large packet delivery latencies, packet drops, and low effective bandwidth. In one form, the router identifies each packet flow and uses packet-by-packet Round Robin (PRR) scheduling to deliver one packet from each flow in turn, provided that a packet is available. Given the difficulty of maintaining information for all flows through a router, many variations of this basic algorithm have been devised.

3.2.2 Latency-rate schedulers, rate-proportional schedulers, and traffic-reshaper schedulers

The second way of classifying schedulers is by dividing them into three main sub-categories that are Latency-rate schedulers (LR), rate proportional schedulers, and traffic-reshaper schedulers [24], [26]-[28], [35]-[37].

The analysis of *LR* schedulers provides a means to describe the worst-case behavior of a broad range of scheduling algorithms in a simple and elegant manner. For a scheduling algorithm to belong to this class, it is only required that the average rate of service offered by the scheduler to a busy period, is at least equal to its reserved rate. Most of the known work–conserving schedulers, including Weighted Fair Queuing (WFQ) also known as Packet-by-packet General Processor Sharing (PGPS), Virtual Clock (VC), Self-Clock Fair Queuing (SCFQ), Weighted Round Robin (WRR), and Deficit Round Robin (DRR), exhibit this property and can therefore be modeled as *LR*-schedulers.

The behavior of an *LR* scheduler is determined by two parameters, Θ the latency and ρ the allocated rate. The latency of an *LR*-switch is the worst-case delay seen by the first packet of the busy period of a session, that is, a packet arriving when the session's queue is empty. The latency of a particular scheduling algorithm may depend on its internal parameters, its transmission rate on the outgoing link, and the allocated rates of various sessions.

The rate proportional switch includes Starting Potential Based Fair Queuing (SPFQ) [27], and Frame Based Fair Queuing (FFQ) [28]. These severs are sharing one common philosophy that is based on isolating the users that gives rise to ideal delay and disturb the fairness behavior of the scheduler.

The traffic-reshaper schedulers are those that have a reshaper for the flows entering the switch and these reshapers are considered themselves as a part of the scheduling algorithm itself. One well-known reshaper is token bucket. A token-bucket reshaper (regulator) regulates linear bounded arrival processes (LBAP) that can be characterized by a linear function that includes two parameters,σ and ρ, so that the number of bits transmitted in any interval of length $t \leq \rho t + \sigma$. Here ρ corresponds roughly to the long-term average rate allocated by the network to the source (which may be substantially larger than source's true average rate), and σ the longest burst a source may send, given the choice of ρ, while still obeying the LBAP descriptor.

Intuitively, the regulator collects tokens in a bucket, which fills up at a steady drip rate. Each token is effectively permission for the source to send a certain number of bits into the network. When a packet arrives at the regulator, the regulator sends the packet if the bucket has enough tokens. Otherwise, the packet waits either until the bucket has enough tokens or until the packet is discarded. If the bucket is already full of tokens, incoming tokens overflow are not available to future packets. Thus, at any time, the largest burst a source can send into the networks is roughly proportional to the size of the token bucket.

In other words, a token bucket collects fixed size tokens in a token bucket and permits transmission of a packet only if the sum of the token sizes in the bucket adds up to the packet's size. Figure 3.1 shows a schematic diagram for the token bucket construction. Upon departure, the regulator gets rid of tokens corresponding to the packet size from the token bucket. The regulator periodically adds tokens to the bucket (at a rate ρ). However, the bucket overflows if the number of tokens crosses some threshold, called its depth, σ. A token bucket limits the size of a transmitted burst to a little more than the bucket's depth (since tokens may arrive while the bucket's worth of packets are being transmitted), and over the long term, the rate at which packets depart the regulator is limited by the rate at which tokens are added to the bucket. The regulator delays a packet if it does not have sufficient token for transmission. Typically, we initialize the bucket to be full. We can simply consider the token bucket reshaper as a traffic light signal for the traffic to flow from certain sources to the switch. It is a very powerful tool in controlling the traffic flow through the whole network.

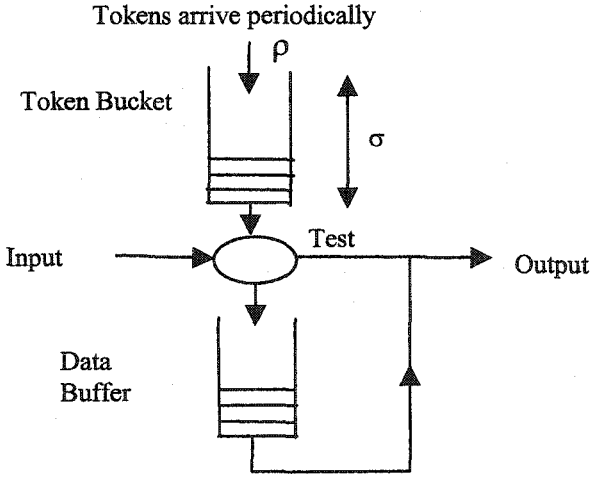

Figure 3.1 Token bucket regulator [20].

3.2.3 Rate-based schedulers and the scheduler-based schedulers

The third way of classifying schedulers (the most convenient one) is by dividing the algorithms into two main categories, the *rate-based* schedulers and the *scheduler-based* schedulers [29], [32], [38].

Rate-based schedulers include Hierarchical Round Robin (HRR), Weighted Fair Queuing (WFQ), named also as packet Generalized Processor Sharing (PGPS), and Rate-Controlled Static Priority (RCSP). In these types of schedulers the quality of service requested by a connection is translated into a transmission rate or bandwidth. There is a predefined set of allowable rates, which are assigned static priorities. The designated bandwidth guarantees a fixed maximum delay for each packet in that rate class, i.e., this policy gives an upper bound on the delay that may be experienced by the specified packet (connection). We should notify the reader that some algorithms have been mentioned in both types of classification (2) and (3). For instance, fair queuing is mentioned in both types depending on the particular feature/features under consideration. Sometimes there is some kind of overlap between some of the feature/features.

Schedule-based schedulers include Earlier Due Date for Delay (EDD-D), Earlier Due Date for Jitter (EDD-J), Smallest Response Time (SRT), and Preemptive Cut Through (PCT). These types of schedulers, instead, analyze the potential interaction between packets of different connections, and determine if there is any possibility of a deadline being missed. Thus, priorities are assigned dynamically based on a deadline. Rate-based schedulers have the advantage of simple implementation, while scheduler-based schedulers allow bandwidth, delay, and jitter to be independently allocated.

3.3 Scheduling trends and traffic classes

A number of proposals have been put forth to provide network support for diverse application requirements. Generally, a priority mechanism gives priority to traffic with deterministic delay bounds, followed by traffic with statistical bounds, and finally best effort traffic. Priorities also simplify the decision of whether to accept a new connection or not.

Tenet approach [19], [39] aims to provide connection-specific QoS, divided into three classes of guarantees: deterministic with delay and delay jitter bounds, statistical (delay bounds with acceptable delay-loss probability), and best effort. At each node, traffic is processed by multi-class earliest due-date scheduling, with class priority decreasing from deterministic to best effort traffic. Admission control is based on peak rates and average rates for connections with statistical guarantees. Connections are set up through the real-time channel administration protocol (RCAP) [40], while data are transported in IP-like packets with added channel identifier and jitter correction factor.

Sriram [11] mixed the notions of traffic classes and per-connection guarantees in a round-robin scheduler. High-bandwidth CBR connections with stringent performance requirements use their own queue with individual time slice assignments, while other connections may be combined into a single assignment. Connections are admitted if a model based on on/off sources approximated by the first two moments predicts sufficient QoS.

Clark et al. [14] proposed a three-level hierarchy: guaranteed, predicted, and best effort service. A connection requests a particular rate based on the worst-case queuing it can accept. The connection will be accepted if there is sufficient remaining capacity at every link along its path to accommodate its assigned rate. Predicted service uses the bandwidth not allocated to guarantee service; admission control for predicted service is not precisely defined. Predicted service uses FIFO+ scheduling with several priority classes to reduce delay variance for multi-hop connections. Best effort traffic is assigned the lowest priority, isolating all other classes of traffic from it. Clark advocates reserving a fixed bandwidth for this class.

3.4 Different approaches for statistical guarantees on delay and loss for soft real-time applications

For soft real-time applications, a number of researchers have advocated statistical guarantees on packet loss and on servicing these applications regarding connection-level. These methods are based on three main techniques, source-based, bounding, and observation-based approaches.

3.4.1 Source-based approaches

In the source-based approach to providing QoS guarantees [41], [42]; traffic sources at the network's edge and within the network are characterized by relatively "simple" models. An example of such a source is the on/off voice source [41], [42], [43]. So as to determine whether or not the multiplexed sources will receive their required QoS, the queuing behavior of the multiplexed traffic sources is then analyzed. In [41], the QoS parameter of concern is the packet loss; in [40] the parameter of concern is the delay.

One advantage of this approach is its simplicity that makes it well suited for real-time, on-line implementation. For example, the connection admission control mechanism based on the approximate QoS scheme described in [41] executes the algorithm used with a very small number of computation operations.

3.4.2 Bounding approaches

The bounding approach explicitly considers the effects of multiplexing on a connection traffic characteristic, and hence its performance, as the traffic passes through various multiplexers. In [17], no special assumptions are made about the actual cell interarrival times, as is done in traditional queuing analysis. Instead, for each connection, a random bound on the number of arrivals in any interval of time of length k is specified, typically for a set of values for k. Given these stochastic bounds on traffic at the edge of the network, bounds could be computed for each connection after it passes through each multiplexer along its path in the network. In that work, performance bounds on the per connection distribution of delay are computed for a sample 27-connection 13-node network, and are shown to be tight for some traffic parameter values but quite loose for others.

The most important note about this approach is its reliance on the ability to bound the maximum length of each queue's busy period for a given set of traffic specifications. If this condition is not satisfied, no bound can be computed, even though it may be known using classic queuing analysis that the queues themselves are stable.

3.4.3 Observation-based approaches

This approach depends on knowing previous history of the network. The measurements of certain types of traffic sources are used to identify an arriving connection and in determining the connection acceptance decision. The main advantage of this approach is that the connection does not have to specify its traffic parameters. However, the connection must belong to one of the known classes, and its traffic must, presumably, correspond to the characteristics of that class if the guarantees are to be satisfied [14], [15], [44].

In the on-line approach described in [14] the bandwidth requirements of already admitted token bucket controlled connections are determined from the current measured behavior of these connections rather than the traffic parameters declared by these connections when they arrived to the network. This measured behavior, together with the declared parameters of an arriving connection, are then used in making a decision for the connection acceptance/rejection decision for the incoming connection. The main point that has to be recognized for this approach is that the QoS requirements are not guaranteed, but instead they are a kind of predicted guarantees. The great advantage for these approaches is that the network is often fully utilized. The main disadvantage of these approaches is that it is based on estimation, which can be inaccurate.

3.5 The Effect of different policies on the performance

In this section, we discuss how the different policies affect the performance of the scheduling process with respect to the classes of services being serviced. Priority policies (static or dynamic): These policies affect the order of service and determine which packets get discarded when the buffer fills up [45], [46].

In [1] Awater and Schoute investigated the combined low-delay low-loss effect through dynamic programming. When the buffer fills, the oldest low-delay packet is replaced. For a slotted system with Bernoulli arrivals, they found that a threshold policy performs best, with service priority given to the low-delay packets as long as the number of low-loss packets is below a given threshold, where the threshold depends on the desired tradeoff between loss

and delay. Several different scheduling algorithms for providing varying degrees of priority to real-time were examined in [47].

Laxity-based policies such as the asynchronous time-sharing system (ATS) [15] also offer a traffic class suitable for soft-real time sources. In the MARS scheduling algorithms, the so-called class II traffic gets the slots in a round-robin cycle not needed by class I (delay bounded) traffic. Within the class II traffic, the scheduler again delays packets as long as possible without violating their delay constraint.

Deadline-based policies divide the end-to-end deadline into per-node deadlines. In many real networks, however, only a subset of nodes is congested and has trouble meeting packet deadlines. Thus to increase node utilization and to reduce the delay variance, it has been suggested [48], [49] to use the laxity divided by the number of hops left to travel as the scheduling criterion. The metric also has the advantage that it readily gives the same delay performance to connections with large and small number of hops. The method has the disadvantage that the laxity measure for all packets in the queue (or at least the first few) has to be recomputed at every scheduling instant.

3.6 Review of the most common schedulers

The key point to understand the majority of schedulers lies on understanding the hypothetical fluid-flow system known as the Generalized Processor Sharing (GPS) [50]. GPS is characterized by positive real numbers, L_1, L_2, ..., L_n. The switch serves all of the nonempty queues simultaneously at a rate proportional to their reserved weights. A flow is called backlogged whenever it has data in its queue. Let $S_i(\tau,t)$ be the amount of flow i traffic served in interval $(\tau,t]$. Then, for any two backlogged flows i,j in the interval $(\tau,t]$, the following relation holds:

$$\frac{S_i(\tau,t)}{S_j(\tau,t)} \geq \frac{L_i}{L_j} \quad , j = 1,2,....n \tag{3.1}$$

Where n is the total number of flows sharing the GPS server. GPS is hypothetical because it can serve n flows simultaneously treating each one as infinitely divisible fluid. The most

natural way of approximating this discipline in the *digital world* would be bit-by-bit Weighted Round Robin (WRR), or more practically packet-by-packet WRR for some very small uniform packets.

Another way of approximating GPS in the real world is simulating it in parallel, that is, computing the time a packet would complete service had it been served by GPS, then serving packets in order of computed completion times. This is the well-known Weighted Fair Queuing (WFQ) algorithm [30], and other time stamping algorithms do for emulating GPS.

The GPS scheduler has a main drawback: the scheduler has to keep track of each connection and update a variable according to the reservation and traffic pattern history of each connection for each packet and later timestamp each packet with an index indicating their virtual finish times. The complexity of this process for emulating GPS makes Weighted Fair Queuing WFQ impractical to implement on high-speed networks. Figure 3.2 shows some of the different algorithms that were inspired by the GPS. They can be divided into clusters. Each cluster started with one basic algorithm or one basic philosophy. The rest of the algorithms in the same cluster are derivatives after making a slight modification to the basic algorithm [51]. In the following sections we review some of the common schedulers that are based on the hypothetical GPS model.

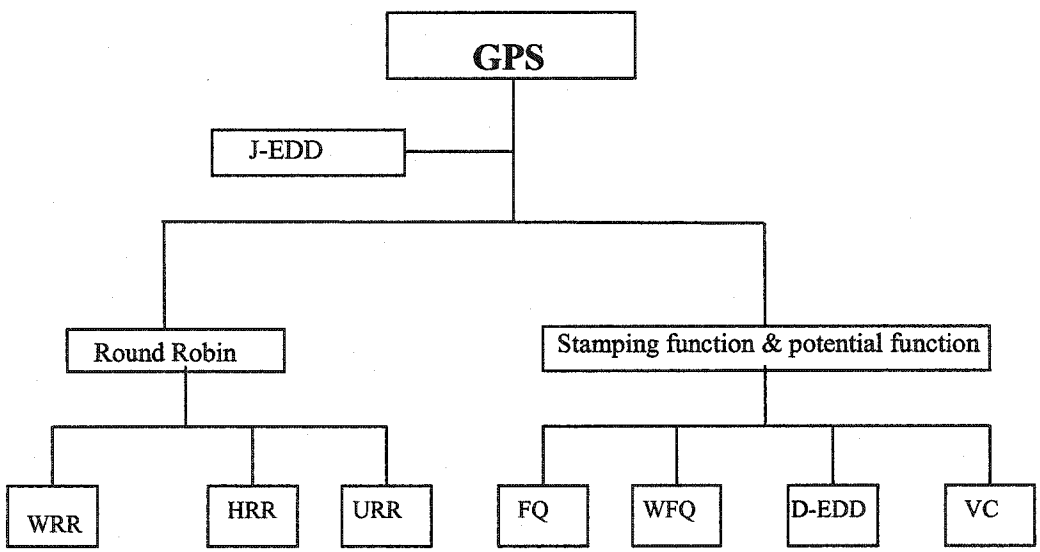

Figure 3.2 Emulation of the most well-known schedulers in communication networks.

We will compare six algorithms, three work-conserving which are Fair Queuing (FQ) [30], Virtual Clock (VC) [31], and Delay Earliest-Due Date (Delay-EDD or EDD-D) [40], and three non work-conserving which are Jitter Earliest-Due-Date (Jitter-EDD or EDD-J) [12], Stop-and-Go (S&G) [27], and Hierarchical Round Robin (HRR) [36]. For the sake of comparison, a description of how each algorithm works is presented, followed by the traffic models used for studying each algorithm. Then, intensive mathematical comparison is done between the different algorithms. Finally, we discuss the implementation issues for these algorithms.

3.6.1 Fair Queuing, Virtual Clock and Delay-EDD

The aim of Fair Queuing is simple: if N channels share the output link, then each channel should get 1/N of the bandwidth, with the provision that if any channel uses less than its share, the slack is equally distributed among the rest. This can be achieved by doing a bit-by-bit round robin (BR) service among the channels. Channels can be given different fractions of the bandwidth by giving them weights; a weight corresponds to the number of bits the channel receives per round of BR service. In Virtual Clock, each packet is allocated a virtual transmission time, which is the time at which the packet would have been transmitted were the server actually doing time division multiplexing. In Delay Earliest-Due-Date, each packet is assigned a deadline, and the packets are sent in order of increasing deadlines.

3.6.2 Jitter Earliest-Due-Date, Stop-and-Go and Hierarchical Round Robin

The Jitter-EDD scheduler extends Delay Earliest-Due-Date to provide delay-jitter bounds. In this scheduler, a packet is stamped with the difference between its pre-calculated deadline and the actual finishing service time. A regulator located at the entrance of each switch holds the packet for the period calculated in the previous step before the packet is made eligible to transmit. A Stop and Go scheduler target is mainly to maintain the 'smoothness' property of the traffic as it traverses the network. Time is divided into frames. In each frame time, only packets that arrived at the switch in the previous frame are sent. The Hierarchical Round Robin scheduler has several levels. Each level offers round robin service for a fixed number of slots. Each channel is allocated a specified number of slots.

3.6.3 Traffic models used for the schedulers under consideration

The scheduling problem requires that the clients should specify their traffic parameters, so that each switch can reserve sufficient resources. In Virtual Clock, Stop-and-Go, and Hierarchical Round Robin a transmission rate AR averaged over an interval AI is given. In the following analysis, Fair Queuing is assumed to have the same traffic parameters just for comparison.

Delay Earliest-Due-Date (Delay-EDD) and Jitter Earliest-Due-Date (Jitter-EDD) have a different model than the one used above which uses three parameters. These three parameters are X_{min}, which is the minimal packet interarrival time, X_{ave}, which is an average packet interarrival time, and I, which is the interval over which X_{ave} is computed. The two parameters $X_{\min}$ and X_{ave} are used to characterize bursty traffic and to offer statistical guarantees [52], [53].

We should relate the two traffic models mentioned together. In EDD disciplines a switch uses X_{ave} to do bandwidth allocation, and $X_{\min}$ to ensure that delay bounds can be met. We can assume that $X_{\min}$ is equal to X_{ave}. Then the following correspondence holds:

$$AR \Leftrightarrow \frac{1}{X_{\min}} \tag{3.2}$$

3.6.4 Mathematical analysis and comparison among work conserving schedulers

In each of the Fair Queuing, Virtual Clock, and Delay-EDD, at each switch, there is a state variable associated with each channel to monitor and enforce the rate for that channel. In Fair Queuing, it is called Finish Number (F); in Virtual Clock the variable is called auxiliary Virtual clock ($auxVC$); and in Delay-EDD it is called Expected Deadline (ExD).

In all three cases, $auxVC$, F, and ExD are used as priority indices of packets; packets are served in the order of increasing priority index. The computation of these indices is shown in Table 3.1. The subscript i is the channel number, the superscript k is the switch number and d_i^k is the local delay bound assigned to the channel at channel establishment time. The delay bound is the maximum allowed difference between the arrival time and departure time of a

packet from the k^{th} switch for the i^{th} channel. In Virtual clock Delay-EDD, AT is the packet arrival time; in Fair Queuing, R^k is the number of rounds that have been completed for a hypothetical bit-by-bit round robin server at switch k, n_i is the weighting factor, and P_i is the packet length measured in number of bits. In Virtual Clock, $Vtick_i$ is the average packet interarrival time.

Virtual Clock	**Fair Queuing**	**Delay-EDD**
$auxVC_i^k \leftarrow \max(AT, auxVC_i^k)$	$F_i^k \leftarrow \max(R^k, F_i^k)$	$ExD_i^k \leftarrow ExD_i^k + X\min_i$
$auxVC_i^k \leftarrow auxVC_i^k + Vtick_i$	$F_i^k \leftarrow F_i^k + \frac{P_i}{n_i}$	$ExD_i^k \leftarrow \max(AT + d_i^k, ExD_i^k)$
Stamp packet with $auxVC_i^k$	Stamp packet with F_i^k	Stamp packet with ExD_i^k

Table 3.1 Comparison of Virtual Clock, Fair Queuing and Delay-EDD [53].

Virtual clock and Fair Queuing are completely equivalent

$$AT \Leftrightarrow R^K \tag{3.3}$$

$$auxVC_i^k \Leftrightarrow F_i^k \tag{3.4}$$

$$Vtick_i \Leftrightarrow \frac{P_i}{n_i} \tag{3.5}$$

If we combine and compare the two state equations for both Virtual Clock and Delay-EDD, we have

- In Virtual Clock,

$$auxVC_i^k \leftarrow \max(AT + Vtick_i, auxVC_i^k + Vtick_i) \tag{3.6}$$

- In Delay-EDD,

$$ExD_i^k \leftarrow \max(AT + d_i^k, ExD_i^k + X_{\min_i}) \tag{3.7}$$

We have the following mapping

$$auxVC_i^k \Leftrightarrow ExD_i^k \tag{3.8}$$

$$Vtick_i \Leftrightarrow X_{\min_i} \ \ or \ \ d_i^k \tag{3.9}$$

In Delay-EDD, $X_{\min_i}$ is the minimum packet interarrival time and d_i^k is the local delay bound. In Virtual Clock, $Vtick_i$ is the average packet interarrival time. It is clear that we have two major differences:
1-Delay-EDD imposes the restriction of minimum spacing between packets, while Virtual Clock does not. 2-Delay-EDD decouples the delay and bandwidth requirements by using both $X_{\min_i}$ and d_i^k, while Virtual Clock has counterpart $Vtick_i$.

The two differences are the reason why Delay-EDD, in conjunction with the establishment and admission control scheme described in [40], can provide both delay and bandwidth guarantees, while Virtual Clock can provide only bandwidth guarantees.

Virtual Clock has a mechanism to handle priorities. In this case channels with higher priority will get lower delay. However, without imposing minimum spacing between packets and a priority allocation scheme similar to the delay bound allocation scheme in [40], Virtual Clock cannot provide deterministic delay bounds.

3.6.5 Delay-EDD and Jitter-EDD

The comparison between Delay-EDD and Jitter-EDD is shown in Table 3.2. It should be mentioned here that we are using different notation than the one used in Table 3.1 for the state equations, but it can be shown that they are equivalent. The variable ExA_i^k is the expected arrival time of the coming packet. It is related to ExD_i^k of Table 3.1 by:

$$ExD_i^k \Leftrightarrow ExA_i^k + X_{\min_i} \tag{3.10}$$

The variable *Ahead* is the amount to time the packet arrives ahead of schedule at the current switch.

There are two major differences between Delay-EDD and Jitter-EDD:
1-There is one more term in calculation of *Ahead* in Jitter-EDD: *PreAhead*. *PreAhead* is a value carried from the previous switch; it tells the switch how much packet is ahead of schedule with respect to the previous switch, i.e., the difference between the deadline and the actual finishing time of the packet in the previous switch.

2- Jitter-EDD holds the packet for *Ahead* units of time, and then calculates its deadline and hands it to the scheduler, while Delay-EDD extends the deadline by *Ahead* time units and hands the packet immediately to the scheduler.

Delay-EDD	Jitter-EDD
$ExA_i^k \leftarrow \max(ExA_i^k + X_{\min_i}, AT)$	$ExA_i^k \leftarrow \max(ExA_i^k + X_{\min_i}, AT)$
$Ahead \leftarrow ExA_i^k - AT$	$Ahead \leftarrow \max(ExA_i^k - AT, \Pr eAhead)$
	Hold the packet for *Aheadtime*
$Deadline \leftarrow AT + Ahead + d_i^k$	$Deadline \leftarrow AT + Ahead + d_i^k$
Stamp packet with *Deadline*	Stamp packet with *Deadline*

Table 3.2 Comparisons between Delay-EDD and Jitter-EDD [53].

The Jitter-EDD has the following properties:

1-The traffic pattern of a channel is preserved at each switch in spite of network load fluctuation; if the traffic from a channel obeys the (X_{min}, I) specification at the entrance of the network, then this traffic will also obey (X_{min}, I) at each switch.

2- Let switch 0 be the source, d_i^h be the local delay bound assigned to channel i at switch h and t_i^h be the service time of the packet at switch h. If a packet from channel i enters the network at time *EnterTime*, the earliest time it can arrive at switch k is $EnterTime + \sum_{h=0}^{k-2} d_i^h + t_i^{k-1}$; the earliest time and the latest time the packet can leave the switch are $EnterTime + \sum_{h=0}^{k-1} d_i^h + t_i^k$ and $EnterTime + \sum_{h=0}^{k} d_i^h$ respectively.

3- Property 2 implies that the maximum residence time of a packet on channel i at switch k is $d_i^{k-1} + d_i^k - t_i^{k-1}$.

The packet service in Jitter-EDD is described in Figure 3.3.

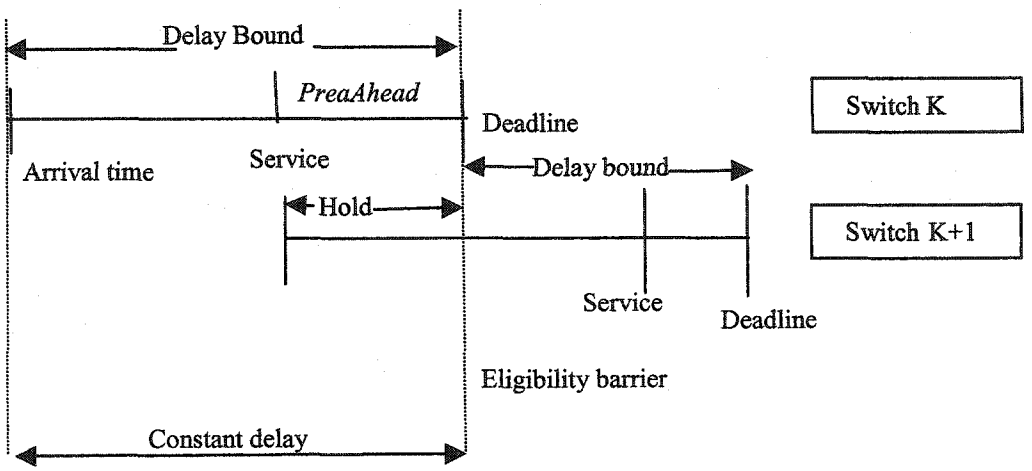

Figure 3.3 Packet Service in Jitter-EDD [53].

3.6.6 Jitter-EDD compared with the three work conserving schedulers

It should be noticed that the three work conserving disciplines: Fair Queuing, Virtual clock, and Delay-EDD, do not have explicit properties similar to the three properties of the Jitter-EDD. But, in some sense they have parallel properties to the previous three properties of the Jitter-EDD. These parallel three properties are:

1- For all three-service disciplines, even if the traffic pattern of a channel obeys an average rate bound at the entrance of the network, a switch may face a higher instantaneous input rate over that channel due to network load fluctuations.

2- For Delay-EDD, if a packet from channel i enters the network at time *EnterTime*, the earliest time it can arrive at switch k is $EnterTime + \sum_{h=0}^{k-1} t_i^h$; the earliest time and the latest time it can leave the switch are $EnterTime + \sum_{h=0}^{k} t_i^h$ and $EnterTime + \sum_{h=0}^{k} d_i^h$, respectively. Since Virtual clock and Fair Queuing do not provide a worst-case delay bound, they do not have a similar property.

3- For Delay-EDD, the maximum residence time of a packet on channel i in which k

$$\sum_{h=0}^{k} d_i^h - \sum_{h=0}^{k-1} t_i^h$$

3.6.7 Stop-and-Go and Hierarchical Round Robin

Both Stop-and-GO and HRR use a multi-level framing strategy. In both Stop-and-Go and HRR, the time axis is divided into periods of length T, each called a frame. Bandwidth is allocated to each channel as a function of the frame time.

Stop-and-Go defines departing and arriving frames for each link. At each switch, the arriving frame of each incoming link is mapped to the departing frame of the output link by introducing a constant delay θ, where $0 < \theta < T$. All the packets from one arriving frame of an incoming link and going to output link l are delayed by θ and put into the corresponding departing frame of l. According to the Stop-and Go discipline, the transmission of a packet that has arrived on any link l during frame f should always be postponed until the beginning of the next frame. Since packets arriving during frame f of the output link are not eligible for transmission until the next frame, Stop-and Go is a non-work conserving service discipline. Within each frame, the service order of packets is arbitrary.

One level HRR is equivalent to a non-work conserving round-robin service discipline. Each channel is assigned a fraction of the total available bandwidth, and receives that bandwidth in each frame, if it has sufficient packets available for service. The switch ensures that the assigned bandwidth is also the designated service rate for that channel in each frame. This means that, in a frame, after providing a channel's bandwidth allocation, even if the switch is available and more packets from that channel are queued for transmission, the packets will not be served until the next frame. Since these extra packets are not eligible for transmission until the next frame, HRR is a non-work conserving service discipline. Within each frame, the service order of packets is arbitrary.

Stop-and-Go has the following properties that are parallel to the properties given before for the Jitter-EDD which are as follows:

1- If at the entrance of the network the traffic for a certain channel obeys the average rate constraint, the traffic will obey the same constraint throughout the network.

2- If a packet from channel i enters the network at time *EnterTime*, the earliest time it can arrive at switch k is $EnterTime + \sum_{h=0}^{k-1} \theta_h + (k-2)T$; the earliest time and the latest time it can leave switch k are $EnterTime + \sum_{h=0}^{k} \theta_h + (k-1)T$ and $EnterTime + \sum_{h=0}^{k} \theta_h + kT$ respectively, where switch 0 represents the source and θ_h is the synchronizing time between the input link and the out link at switch h, $0 \le \theta_h \le T$.

3- Property 2 indicates that the maximum packet residence time at switch k is: $2T+\theta_k$.

HRR posseses the following properties:

1- If the traffic from a channel obeys, at the entrance of the network, the average rate constraint, the traffic will obey the same constraint at each switch. If the traffic rate is higher at the entrance, the first switch will smooth out the traffic, and the traffic of the channel will obey the average rate constraint at downstream switches.

2- If a packet over a channel i enters the network at time *EnterTime*, the earliest time it can arrive at switch k is $EnterTime + \sum_{h=0}^{k-1} t_i^h$; the earliest time and the latest time it can leave the switch are $EnterTime + \sum_{h=0}^{k} t_i^h$ and $EnterTime + (K+!)T$, respectively. Here, switch 0 is considered to be the source, and t_i^h is the service time of a channel i packet at switch h.

3- The maximum packet residence time in a switch is $2T$ (for a three level HRR).

It should be noted that property 2 for HRR does not give a tight bound on the minimum delay that a packet can incur as in Jitter-EDD and Stop-and-GO. However, HRR does provide an upper bound on delay. Thus HRR provides delay bounds but not tight delay jitter bounds. Also, it is important to note that property 3 is derived from property 2 for both Jitter-EDD and Stop-and Go, while property 3 holds by itself for HRR due to the nature of a non-work conserving service discipline.

3.7 Implementation considerations

We have compared the service disciplines from the mathematical point of view. Implementation can be done in hardware or software. The choice of data structure and algorithms determines the complexity of implementation. Thus, it is hard to compare among them due to different ways of implementations. We will briefly give the outlines of the necessary steps to implement the algorithms.

3.7.1 Virtual Clock, Fair Queuing, and Delay-EDD

The three algorithms have been shown to be quite similar, and their implementation is identical. In any of them, when a packet arrives, per-channel state has to be retrieved and updated. Then, the packet is stamped with a priority index and placed in a priority queue. When the output link becomes free, the server removes the packet at the head of the priority queue and sends it.

The efficient implementation of a priority queue for Fair Queuing in software has been investigated in [54], and the results indicated that a simple linked list is a good priority queue implementation. The same result of course applies to the other two disciplines.

3.7.2 Jitter-EDD

Jitter-EDD can be implemented in the same way as Delay-EDD with the addition of a regulator. A regulator has to delay a packet an additional amount of time in order to reconstruct the input traffic stream. It can do so by constructing lists of packets for each delay value and placing them in a calendar queue [40].

3.7.3 Hierarchical Round Robin

HRR is mainly designed to be implemented in hardware three are counters associated with each service level, and these are loaded at the beginning of each frame time. Transfer of control between levels is based on simple combinational logic operating on the values of the counters. The choice of the next packet to send involves only one step in the hardware, and can be done at very high speeds. Per-channel queuing of the data is done using custom VLSI [55].

3.7.4 Stop-and-Go

Stop-and-Go can be implemented by a number of FIFO queues, a service controller, a transmission queue and a transmission switch [56]. Each queue corresponds to a frame time and stores packets for channels that are allocated at that frame time. At the beginning of each frame time, the service controller is interrupted and it places the packets present in the corresponding queue (which have been collected in the previous frame time) at the head of the transmission queue. The switch services packets from the head of the transmission queue. In this way, the service of the packets from a frame of larger frame time may be interrupted, on a non-preemptive basis, by starting a new frame of smaller frame time.

The periodic interruption of the service controller requires a set of timers. The major cost associated with the implementation is in the design of timers. Another important issue regarding the implementation is the synchronization of the output frames at all the output links and introducing a fixed delay at each input link. To summarize all the previous discussion regarding the above-mentioned techniques that have been used to service the real-time applications, we compare them in Table 3.3.

Property	VC	FQ	D-EDD	SG	HRR	J-EDD
Throughput Guarantee	Yes	Yes	Yes	Yes	Yes	Yes
Delay Guarantee	No	Yes	Yes	Yes	Yes	Yes
Jitter Guarantee	No	No	No	Yes	No	Yes
Work Conserving	Yes	Yes	Yes	No	No	No

Table 3.3 Over-all Comparison among the different algorithms.

3.8 Scheduling in local area networks (LANs)

Scheduling messages for transmission over a local area network is the function of the Medium Access Control (MAC). Much progress has been made in scheduling hard real-time messages for transmission over a local area network (LAN). The basic idea has been to emulate existing centralized scheduling algorithms. This is because there is already a well-developed theory for centralized scheduling. For example the rate monotonic scheduling algorithm [57] was originally proposed for uniprocessor scheduling of a set of hard real-time

tasks, but has now received attention as a basis for scheduling messages in local area networks. In the following sections we will give a brief description of the traffic modeling used in handling the scheduling problem in LANs, followed by a short discussion about the algorithms used for scheduling in LANs.

3.9 Traffic modeling

Researchers working with LANs divided the hard real-time messages into two main categories, synchronous messages and asynchronous messages. A distributed hard real-time system will typically contain both types of messages [58], [59]. There are many other classifications, but we are limiting our discussion to these two types in particular.

3.9.1 Synchronous messages

It is assumed that there are n streams of synchronous messages S_1,....., S_n in the network. A synchronous message stream S_i consists of a sequence of messages with constant interarrival times. Messages in a synchronous message stream are referred to as synchronous messages. The period P_i of stream S_i is the interarrival time of messages in stream S_i. For a message M to be a synchronous message it should have the following properties:

- An arrival time, which is the time at which the message becomes available for transmission. The arrival time of a synchronous message is the beginning for its synchronous message stream.
- A length, which is the number of time units required to successfully transmit the entire message.
- A deadline, D_M, which is the time by which the message must be successfully transmitted. The deadline of a synchronous message is often taken to be the end of the period in which it arrived, i.e., the arrival time of the next message in the stream.

Given a set of synchronous message streams with periods P_1, P_2, P_3,.....,P_n and message lengths L_1, L_2, L_3,.....,L_n, the utilization of the synchronous messages streams is defined as

$$U_s = \sum_{i=1}^{n} \frac{L_i}{P_i} \tag{3.11}$$

The utilization U_s represents the proportion of time that will be needed for transmitting this set of synchronous messages streams.

3.9.2 Asynchronous messages

In contrast to synchronous messages, asynchronous messages have stochastic interarrival times and prior to their arrival the system may not have any knowledge of their timing characteristics [60]. Like synchronous messages, asynchronous messages may be associated with an arrival time, and a length, a deadline D_M. Another time constraint that has often been used for an asynchronous message M is the laxity of the message at time t, denoted $LA_M(t)$, The laxity of the message M is defined as the amount of time remaining before transmission of the message must begin if the message is to meet its deadline. That is,

$$LA_M(t) = D_M - L_M - t - \tau \tag{3.12}$$

where L_M is the length of the message and τ is the propagation delay of the network.

If the average arrival rate of asynchronous messages to a network is λ, and the average message length is L, then the demand utilization U_A of the asynchronous messages is defined as

$$U_A = \lambda * L \tag{3.13}$$

If the deadlines of some asynchronous messages are missed, the actual utilization may be lower than the demand utilization.

3.10 Protocols for synchronous messages

Three main approaches that were used to handle synchronous messages are:

1- Priority driven protocols, each message is assigned a priority (the priority is preferred to be based on the relative urgency of the message). The protocol then employs global priority arbitration to determine which message should be transmitted next. As is clear, this approach is equivalent to the scheduling approach used in WAN's because it uses priority concept in its processing methodology.

2- Bounded access protocols, the amount of time during which a node may not have access to the channel is bounded. The protocol then employs this information to guarantee synchronous messages.
3- Global reservation protocols, synchronous messages are guaranteed by reserving future access time to the channel.

Using a priority driven protocol to transmit synchronous messages is similar to scheduling periodic tasks on a processor; both cases involve scheduling access to a serially used resource for a set of real-time processors. In [57], the rate monotonic algorithm is introduced, for scheduling independent periodic tasks on a uniprocessor. The rate monotonic algorithm assigns priorities to tasks in inverse proportion to their periods. Tasks with a smaller period are assigned a higher priority.

An important parameter that is used frequently to measure the efficiency of these algorithms is called the worst-case achievable utilization. A protocol is said to have an achievable utilization U if it can guarantee all synchronous messages from streams with total utilization no more than U. For the rate monotonic algorithm it has been been proved that the worst-case achievable utilization for a set of n periodic tasks is $n(2^{1/n} -1)$. For large n, this approaches $ln2$ which is approximately 0.69. Provided that the total utilization of a set of tasks is less than 0.69, then the set of tasks can always be scheduled by the rate monotonic algorithm. The rate monotonic algorithm can easily handle asynchronous tasks. It has a relatively low runtime overhead, because task priorities are assigned during system design time, and at run time it only remains to preempt low priority tasks upon arrival of a high priority task [61], [62].

In a protocol with a bounded access time, the amount of time that elapses before a node gains exclusive access to the channel is bounded. Therefore, a node with a message to send can begin transmission of the message within a bounded amount of time. Because of this, the worst-case proportion of time available to a given node to transmit its message can be calculated. This information can then be used to guarantee synchronous messages.

Another way to guarantee synchronous messages, without using rate monotonic scheduling or a bounded access time, is to reserve future access to the channel [63]. One approach is to maintain a global list of the future channel reservation times. When a new synchronous message stream is initiated at a node, the node first uses its local copy of the reservation list to determine whether the messages in the new stream can be guaranteed. If so, a control packet must be broadcast to all nodes to inform them of the change in the schedule. Great care must be taken to guard against inconsistency between the local copies of the reservation list [63]. Inconsistency can arise due to transmission errors or failures on individual nodes.

Also an approach to reserving future channel access can be employed with a slotted ring protocol such as FDDI-II [60], [64]. In a slotted ring, the ring is divided into a number of fixed sized slots, each capable of holding one packet [65]. Synchronous messages may be guaranteed by dedicating a slot in the ring to messages in a given stream. This slot will be reserved for exclusive use of the given synchronous message stream until such time that the stream is terminated. For example, this method is used by FDDI-II to handle voice traffic.

3.11 Protocols for asynchronous messages

Asynchronous messages arrive randomly during runtime and must therefore be scheduled dynamically. Because there are usually insufficient resources to handle all contingencies with respect to asynchronous messages, the goal in designing a network protocol for hard real-time asynchronous messages is to minimize the proportion of messages that miss their deadline. The ability to handle asynchronous messages well makes a system more responsive to changing conditions and will be essential in the next generation of highly distributed and dynamic real-time systems [64], [66].

Several approaches may be adopted by a protocol designed for the transmission of asynchronous messages, which are:

1- Optimal transmission policy. The protocol may attempt to emulate an "optimal" transmission policy in order to reduce the proportion of messages that are lost. A protocol is optimal if no other protocol can achieve a lower message loss.

2- Multi-version messages. The protocol may use multi-version messages in order to reduce the proportion of messages that are lost. Multi-version messages have several versions of different lengths. In order to reduce message loss, a shorter version of a message may be transmitted when the network traffic is high (burst) or when message deadlines are tight.

3- Message guarantee. The protocol can attempt to guarantee ahead of transmission time whether a message will be successfully sent. This gives more time for recovery action in the case that a message is lost.

Chapter 4

The transportation problem and the virtual frame algorithm

4.1 The classical transportation problem

A wide class of optimization problems can be modeled as a transportation problem. The transportation problem received its name because it arises very naturally in the context of determining optimal shipping patterns. However, many problems having no relation with transportation may fit the mathematical model for the transportation problem and thus be solved by one of its simple and efficient solution procedures [67].

To illustrate a prototype of the transportation problem, suppose that m factories supply n warehouses with a certain product. Factory i (i=1,2,.....,m) produces a_i units (total or per unit time), and warehouse j (j=1,2,.....,n) requires b_j units. Suppose that the cost of shipping from factory i to warehouse j is directly proportional to the amount shipped, and that the unit cost is c_{ij}. Let the decision variables x_{ij} be the amount shipped from factory i to warehouse j.

The corresponding mathematical model for the transportation problem is the following. Find x_{ij} (i=1,2,...., m; j=1,2,....., n) in order to minimize

$$f = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{4.1}$$

subject to the constraints

$$\sum_{j=1}^{n} x_{ij} = a_i \qquad for\ i = 1,2,...,m \tag{4.2}$$

$$\sum_{i=1}^{m} x_{ij} = b_j \qquad for\ j = 1,2,...,n \tag{4.3}$$

$$x_{ij} \geq 0, \quad \textit{for all } i \textit{ and } j \tag{4.4}$$

It is the simple structure of this model, particularly that all of the constraints coefficients equal either zero or one, which permits the use of a much more efficient solution procedure than the complete simplex method. The basic procedure of examining a sequence of constantly improving, basic feasible solutions until an optimal one is reached is the same as with the simplex method, but it can now be done more simply [67].

It should be noted that the model has feasible solutions only if

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j \tag{4.5}$$

This may be verified by observing that the restrictions require that both

$$\sum_{i=1}^{m} a_i \text{ and } \sum_{j=1}^{n} b_j \text{ equal } \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}$$

This condition implies that the total supply must equal the total demand when the system is in balance. If the problem has a physical significance and the condition is not fulfilled, it usually means that either a_i or b_j actually represents a bound rather than an exact requirement. If this is the case, a fictitious "factory" or "warehouse" can be introduced to take up the slack in order to convert the inequalities into equalities and satisfy the feasible condition.

If x_{ij} represents a total amount to be shipped rather than a shipping rate, it may have physical significance only for integer values. Fortunately, because of the simple structure of the model, it can be shown that there must exist an optimal solution involving only integers if all a_i and b_j are integers [67], [68].

Rather than writing all of the equations involved, the model for a transportation problem is usually written in a concise form, as illustrated below in Table 4.1. This cost and requirements table is sufficient to completely specify a transportation problem preparatory to its solution. The actual calculations are made directly on the "transportation array" which is

similar to the one shown in Table 4.1 except that all the c_{ij} terms are replaced with the decision variables x_{ij} from each source to each destination.

Source	Destination & cost per unit shipped				Supply
	1	2	...	n	
1	c_{11}	c_{12}		c_{1n}	a_1
2	c_{21}	c_{22}		c_{2n}	a_2
⋮	...	...	...	...	⋮
m	c_{m1}	c_{m2}		c_{mn}	a_m
Demand	b_1	b_2	...	b_n	

Table 4.1 Parameter table for the transportation problem [68].

4.1.1 Decision variables bounds

The formulation of the upper bounded transportation problem is just similar to the standard transportation problem described by the set of Equations (4.1) to (4.5) except for a new added set of bounds on the values of x_{ij}, given by

$$l_{ij} \le x_{ij} \le u_{ij} \qquad \text{for all } i \text{ and } j \tag{4.6}$$

l_{ij} is the designated lower value for the specified x_{ij} and u_{ij} is the upper designated value for the same variable. The upper bounds are introduced so as not to exceed a certain maximum shipping amount for each specified route between source i and destination j [68]. Also, it has to be mentioned that the lower bounds on the variables we have are not necessary to be zero as in the basic formulation of the Transportation Problem. The lower bounds can be any value. These lower bounds will be used in our work to ensure minimum bandwidth as will be shown later.

4.2 The scheduler model

The scheduler is modeled as a single server fed from multiple finite-sized queues. Time is divided into frames of certain duration (in our case the frame duration is set to be 1ms). The scheduler we propose here decides on the amount of traffic to be sent from each service class in one frame period. The traffics entering the buffers are simply the superposition of each specific call (voice, video, isochronouse, and best-effort) after identifying its content in a previous stage. The description of the server with the queues is shown in Figure 4.1.

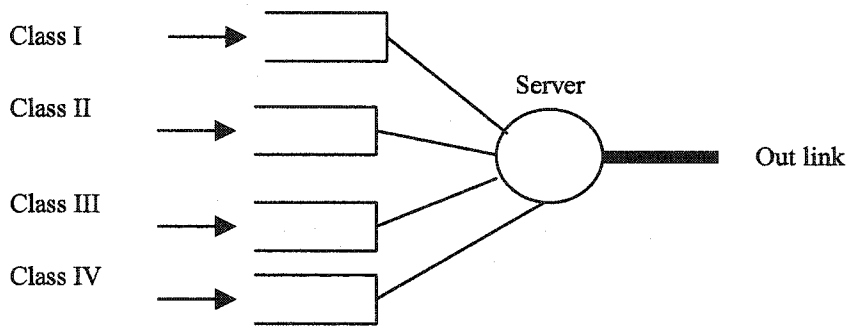

Figure 4.1 The scheduler model.

4.3 The analogy between the transportation problem and the scheduling problem

There are similarities between the transportation problem and the traffic-scheduling problem. The *factories* in the Transportation Problem are analogous to the *number* of bits that can be transmitted within a specified set of time frames. The *warehouses* in the Transportation Problem are analogues of the *data in the buffers* of the different classes. The challenge of the modeling in our case is how to construct the cost function that will implicitly include some of the system requirements that interprets the QoS requested by each service class. Table 4.2 summarizes the analogy between the transportation problem model and the scheduling problem model. It should be noted that the bounds shown in Equation (4.6) would be used with specified lower limits so as to ensure minimum bandwidth allocations for the

different classes. This is done to ensure protection. Also, the upper bound may be set at a certain value to ensure fairness for all classes, and to reshape the output traffic.

Two major differences have to be noticed about the scheduling problem model with respect to the transportation model: 1) The demands which represent the packet storage in the buffers are simply no longer constant and depend mainly on the number of the packets arriving and departing. 2) The values of c_{ij} that represent the cost in the transportation problem for shipping one item or product to the destination are no longer constant for the scheduling problem, since the price that has to be paid for transmitting one packet for the specified class depends mainly on the situations of the buffers. It will be shown in Equation (4.7) how the cost coefficients are built in a dynamic way. So two parameters are changing dynamically in our scheduling problem, the number of packets to be serviced as well as the influence of these packets on each other with respect to the different QoS classes. The cost coefficients have to reflect three main things: (1) The general system cost per transmitting one packet which simply expresses the speed of the output link or in other words the service rate of system link. (2) The mutual coupling between each class and the other classes being serviced regarding the maximum delay that can be withstood by each class. (3) The buffer states from the queue size point of view. Hence, c_{ij} is formulated as follows:

$$c_{ij} = t_0 \times \frac{\tau_j}{\sum_{j=1}^{n} \tau_j} \times \left\lfloor \frac{\sum_{j=1}^{n} b_j}{\sum_{j=1}^{n} b_j - i \times a} \right\rfloor \tag{4.7}$$

where t_o is the time required for the link to transmit one byte. τ_j is the maximum delay tolerated by class j. b_j is the number of packets in the buffer for class j. i is the frame number (i = 0,1,2.....,m-1). m is calculated with the help of Equation (4.5). a is the number of packets that can be transmitted within one frame. There are actually many ways for which we can construct the values of the cost function, and the one shown in Equation (4.7) is one way of doing that.

Name and/or Symbol	Transportation Problem		Scheduling Problem	
	Representation	Value	Representation	Value
Source i	Factory	Integer	Frame i	Integer
Destination j	Warehouse		Class j	N/A
Supply a_i	Products of factory i	Integer	Capacity of frame	Bytes
Demand b_j	Amount required by each consumer	Integer	Packets waiting for service in each buffer	Bytes
x_{ij}	Products from source i to destination j	Integer	Packets transmitted from class j in frame i	Bytes
c_{ij}	Cost of product unit from source i to destination j	Unit cost	Time delay due to transmitting one packet from class j	Sec/Byte
u_{ij}	Upper bounds	Integer	Upper bounds	Bytes
l_{ij}	Lower bounds	Integer	Lower bounds	Bytes

Table 4.2 The analogy between the transportation problem and the scheduling problem.

4.4 The proposed algorithm

A brief description of the algorithm is written in steps below.

Begin

1- *Count* the buffer occupancy of each class at the start of each time frame.

2- *Calculate* the number of frames to satisfy Equation (4.5).

3- *Evaluate* the cost coefficients according to Equation (4.7).

4- *Perform* an optimization step using any linear programming method to evaluate x_{ij}.

5- *Transport* x_{ij} for all service classes for the first frame.

6- *Repeat* by starting from step (1) if $t < T_{simu.}$.

End

Where T_{simu} is a preset simulation time. The algorithm is described in brief in its tabulated form in Table 4.3. It has to be mentioned that w in Table 4.3 represents any number of classes just for the sake of illustration. In our case we are dealing with four classes. Also, we should mention that the final form of the algorithm is achieved mathematically with slight modification of the algorithm described above as will be shown in section 4.5.

Class / Frame	Class 1		Class 2		.....	Class w		Frame Cap.
1	c_{11}	x_{11}	c_{12}	x_{12}	.....	c_{1w}	x_{1w}	a_1
2	c_{21}	x_{21}	c_{22}	x_{22}	.....	c_{2w}	x_{2w}	a_2
...	....		....		.....	....		
....	....		....		....	....		
....	....		....		....	....		....
m	c_{m1}	x_{m1}	c_{m2}	x_{m2}	.....	c_{mw}	x_{mw}	a_m
packet	b_1		b_2		.....	b_w		

Frames

Table 4.3 The tabulated form of the proposed scheduling algorithm.

4.5 The virtual frame algorithm

The scheduling algorithm described in the previous section will just transmit data from the first frame. The rest of the m-1 frames are just calculated for the mathematical validity. Also, the number of variables to be optimized is $m \times n$. Hence, we proposed that we could combine

the m-1 frames (following the first frame) into one frame called the *virtual frame*. That frame will virtually carry all the other packets in replacement of the other m-1 frames. This step was mainly done to reduce the number of variables in a dramatic way from $m \times n$ variables to $2 \times n = 2n$ variables (in our case n = 4). So, the only modification that will be done to the algorithm shown in the previous section is basically step (2). That is done by mathematically combining the *m-1* frames into one frame.

Chapter 5

Numerical simulation

5.1 Introduction

Numerical simulations were performed in order to get an idea about the performance of the scheduler. The performance criterion is measured using four main statistics. These four statistics are relative share, average delay time, average queue length, and probability of packet loss (loss ratio).

The transient performance of the system was briefly studied. The steady state performance of the system was considered to have an estimate of the schedulable region on a rate base concept. The output link utilization was measured under different loading conditions in many different runs. The output link utilization is defined by the ratio between the actual data that has been transmitted and the maximum data that can be transmitted in a certain period of time. Finally, we investigated the effect of frame duration on the performance of the scheduler.

5.2 Numerical simulation setup

The simulation code for the scheduler was written in MATLAB (see Appendix B). The simulator was run for 40 frames for the first set of results. The simulation was run for 15 seconds real time for the second, third, and fourth set of results. We repeated the runs several times and found that the results were close to each other.

Four types of traffic were simulated at the aggregation level; voice, video, isochronous, and best-effort. The traffic sources used were defined using three parameters: the conforming rate ρ_c, the burst rate σ, and the conforming probability p. Appendix A discusses this three-parameter model in detail. All sources had a value of $p = 0.5$ and $\sigma = 3\,\rho_c$. The packet size

was simulated by a normal distribution with average size A equal to 60 bytes and a standard deviation of 10 bytes (for all classes). The output speed link is assumed to be 155 Mbits/s.

The results obtained for steady state performance, output link utilization, and effect of frame duration are entirely based on setting *three* of the class traffic rates to their maximum acceptable rate. That maximum acceptable rate was determined for each traffic stream according to the packet length (average and standard deviation) so as not to exceed a certain rate from the link speed point of view. Otherwise the packets become overlapped which is not practically accepted. Incrementing the *fourth* traffic stream in steps until reaching its maximum acceptable permissible rate, we measured the four main performance criteria described in the previous section.

We set the cost coefficient values (cost function coefficients) at 1, 1, 1, 1 for the basic frame and 6, 5, 4, 3 for the virtual frame. It was found from the numerical simulation that the scheduler performance depends on the differences between the cost function values for the basic frame and the corresponding values in the virtual frame. The choice of the buffer size was 60, 120, 120, and 200 packets respectively for the four different classes introduced before.

The lower limits for the different classes are 10%, 25%, 20% and 0% of the frame capacity for voice, video, isochronous, and best-effort respectively. The upper limits are set equal to the frame capacity for all classes. Although the upper limit of each service class is equal to the full frame capacity, traffic from each class will never reach this upper limit because the scheduler guarantees certain minimum limits for all classes. Thus, protection is assured in our proposed algorithm.

5.3 Transient performance

From Figure 5.1 we see that each service class has a different time constant. We concluded that the transient performance depends on two parameters: the buffer size and the traffic rates. Thus, we can see four different transient responses for the four different classes we have.

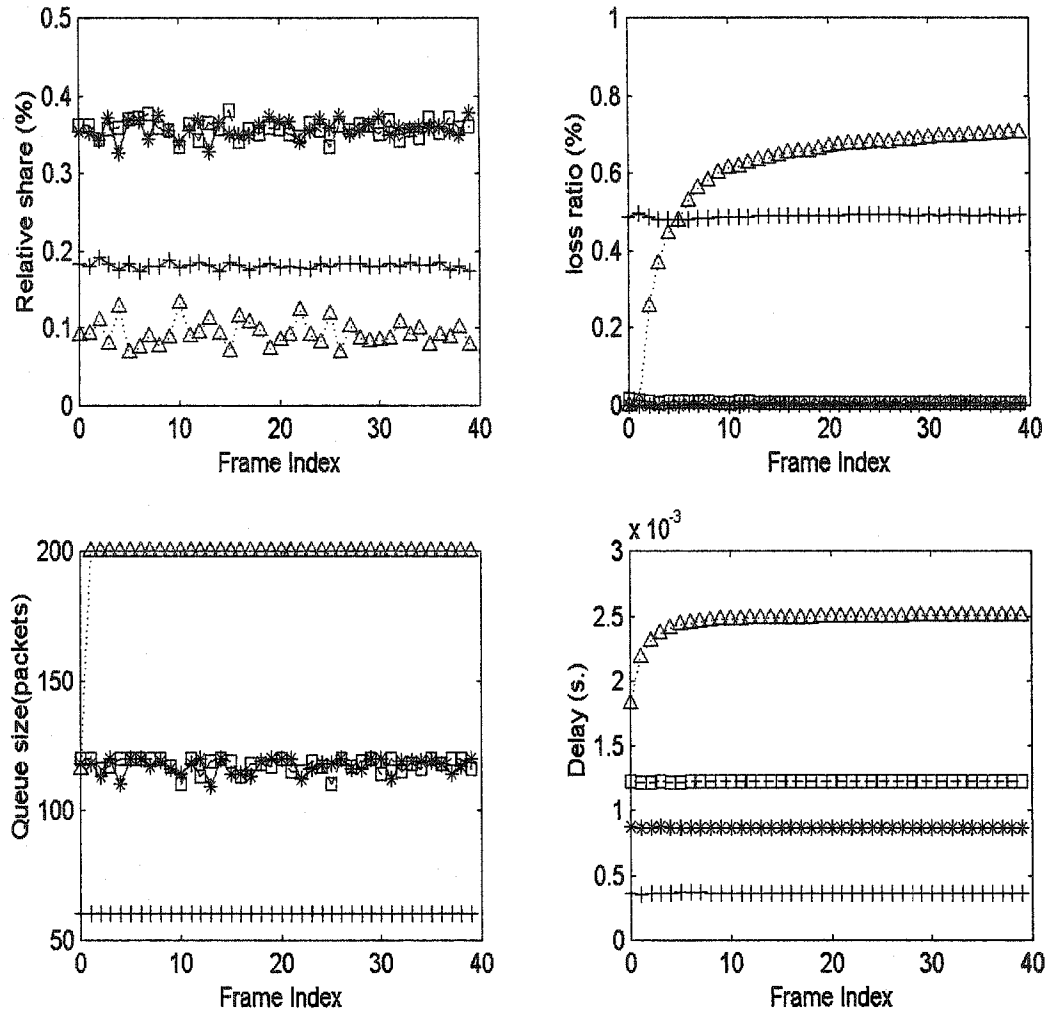

Figure 5.1 The transient behavior of the scheduler. Rates are kept maximum for all classes. Voice : +, Video : * , Isochronous : □ , and Best-effort : Δ.

5.4 Steady state performance

To give more insights on the way we measure the steady state performance, we introduce what is known as the schedulable region in a little different perspective than the one mentioned in chapter 2. The schedulable region is a four-dimensional space (we have four classes) that represents the maximum available space (maximum number of users) at which the scheduling algorithm can serve all classes fulfilling their quality of service requirements. But instead of dealing with the definition as maximum users we deal with it or redefine it as

maximum rates. It has to be mentioned here that measuring the schedulable region is based on two main methods. The first method is to insert the required quality of services parameters as constraints within the simulator code. Then, change the input traffic rates (i.e., number of users) for the different classes up till violating any of the constraints. The second method of measuring the performance is to change the rates without putting any constraints. Then, measure the provided quality of service from which we can deduce the permitted number of users that can be served within certain offered quality of service. In this work we used the second method of measuring the performance. The output link utilization simply expresses the percentage of the system capability we use. In other words it measures how the scheduler uses the system resources efficiently.

Based on the above discussion, we have measured the four main performance criteria of the system but with slight modification. Throughput measures the amount of traffic (bytes or packets) transmitted through the switch from each class versus time. In our case measuring the throughput in absolute terms will not give a true picture about the relative participation of each class with respect to the frame capacity and the relative share of each class. And, actually what we are more concerned about is the relative contribution of each class compared to the total traffic outgoing from the switch.

Figure 5.2 shows the relative share of each class (as defined before) when the rate of each class is varied. The rate of each class was scaled by the factor C/A, where C/A represents the maximum number of packets that could be transmitted on a link with speed C given that the average packet size is A. In Figure 5.2 (a), voice has the variable rate; the share is alternating between the best-effort and the voice traffic. The scheduler gave the lower limit share for the voice, video and isochronous, and distributed the empty portion of the frame capacity in an optimized way for the four classes. During the alternating process between the voice and the best-effort, the video and the isochronous traffics were not affected. It is clear that protection is fully achieved for the voice, video, and isochronous according to their designated shares (upper and lower limits) that were set from the beginning. Also, it is clear how the scheduler is fairly distributing the available frame capacity between the high priority traffics and the

low priority traffics in an optimized way. Figure 5.2 (a) insures fairness and protection for the different classes.

In Figure 5.2 (b), video has the variable rate; the share is alternating between the video and the best-effort. The share of the best-effort is relatively high at the beginning of this alternating process, since the video traffic (with variable rate) has the highest lower limit among the four classes (25%). During this alternating process the voice and the isochronous have got their lower limit shares and in addition they have got an extra share for the empty frame capacity. Figure 5.2 (b) guarantees fairness and protection for all the four classes.

In Figure 5.2 (c), isochronous has the variable rate; the share is alternating between the isochronous and the best-effort. The best-effort has relatively high share at the beginning of this alternating process, since the isochronous traffic has the second highest lower limit among the four classes (20%). Video and voice were fully protected by taking their designated shares and even extra shares. That implies that the protection and fairness are accomplished for all classes.

In Figure 5.2 (d), best-effort has the variable rate; the scheduler performed as required in the sense that the share of the three traffics voice, video, and isochronous are not affected by the growth of the best-effort traffic, which was exactly demanded. Again, the fairness and protection are fully achieved for the four classes according to pre-set configuration of the scheduler.

Figure 5.3 (a), (b), (c), and (d) shows the output utilization associated with each separate run of Figure 5.2 (a), (b), (c), and (d) respectively (point-by-point). It is obvious that the output link utilization ranges between 97% and 99%. Thus, we can say that the scheduler is working perfectly in the sense of efficient use of system resources.

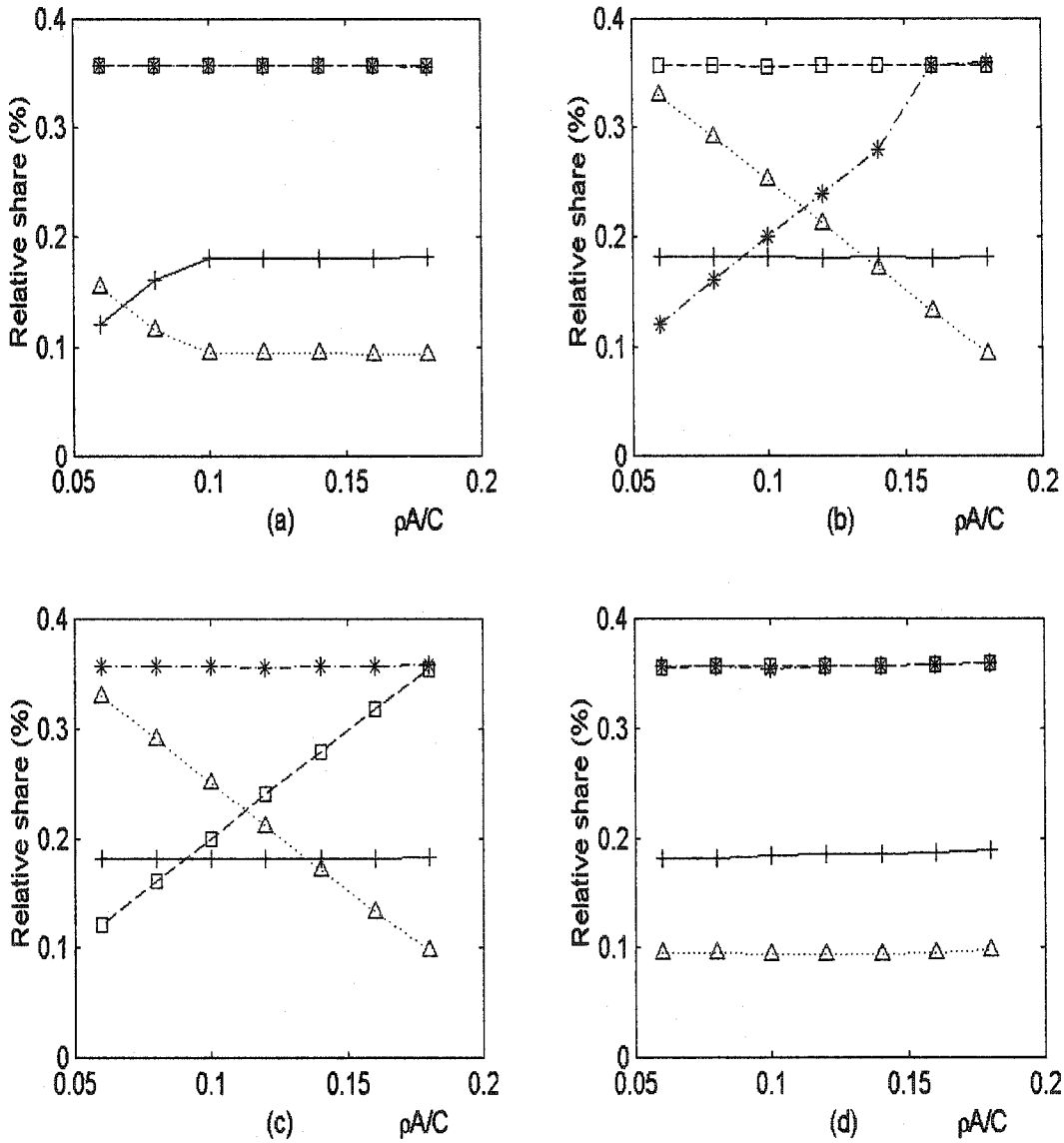

Figure 5.2 Relative share for each class when the rate of one of the service classes is varied; (a) voice traffic (b) video traffic (c) isochronous traffic (d) best-effort traffic. Voice : +, Video : * , Isochronous : □ , and Best-effort : Δ.

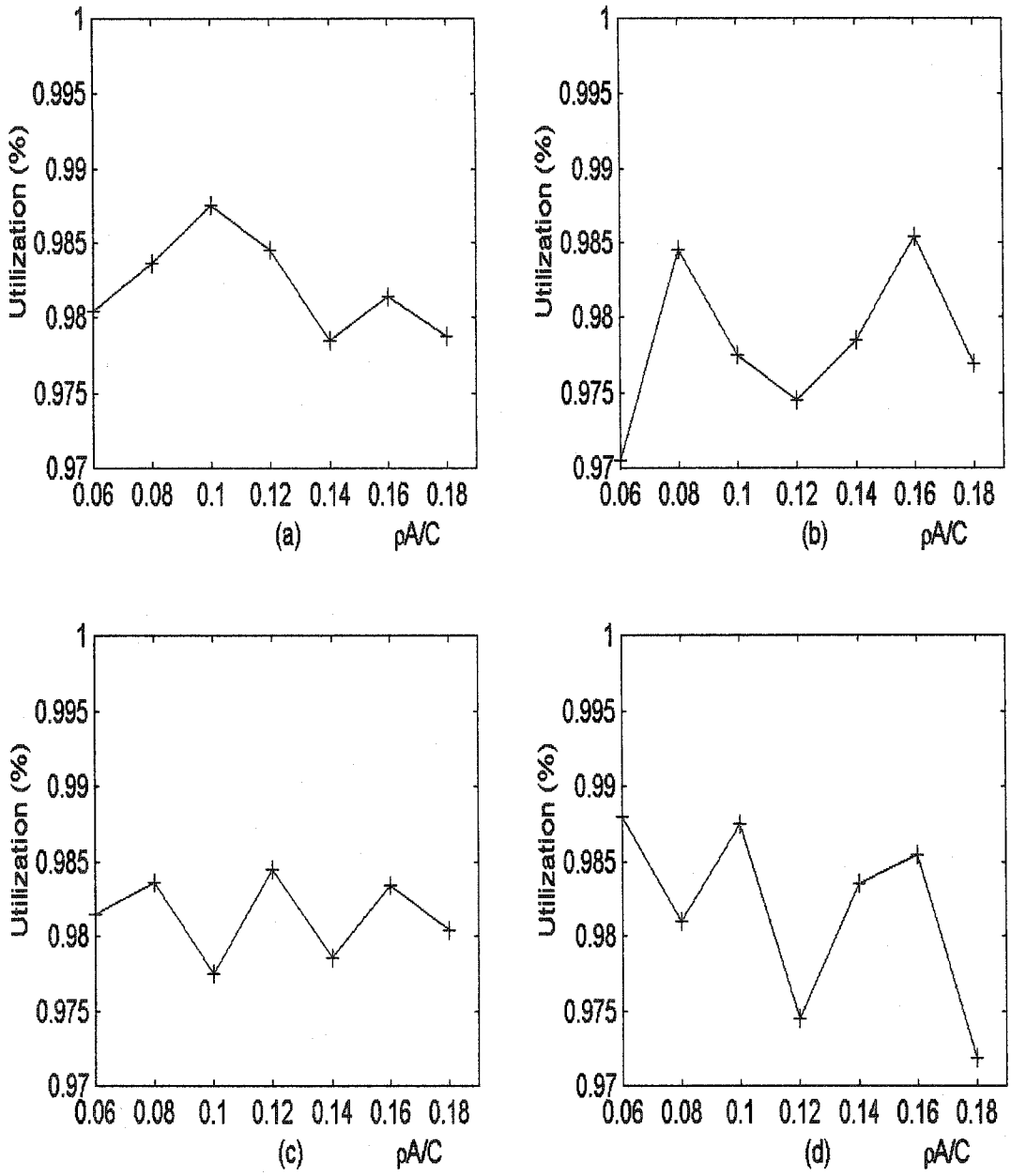

Figure 5.3 Output link utilization associated with the variable rate of different classes corresponding to Figure 5.2 (a) voice traffic (b) video traffic (c) isochronous traffic (d) best-effort traffic.

Figure 5.4 shows average queue size of each class when the rate of each class is varied. Figure 5.4 (a), voice has the variable rate; shows the average queue size for voice traffic is below its maximum buffer capacity limit (60 packets) at the low rate of voice traffic. The other three classes have average queue size approximately equal to the maximum buffer limit

for each class respectively (120 packets for both video and isochronous and 200 packets for best-effort). These three classes have very high traffic rates and consequently they have full buffer occupation due to the limited buffer capacity.

In Figure 5.4 (b), video has the variable rate; the curve for average queue size of video traffic can be divided into three main portions. The first and third portions have a relatively high rate of increase compared to the second portion. In the first portion the scheduler gave a relatively low share to the video traffic so the queue size was increasing at a higher rate. In the second portion of the curve when the video traffic started to have high traffic rate, the scheduler gave it a high share. Hence, the average queue size was not increasing rapidly. In the third portion of the curve although the scheduler gave the video traffic extra share, its queue size still has a high rate of increase. This is due to the burst traffic rates of all traffics at this moment, i.e., the extra high share was not enough.

In Figure 5.4 (c), isochronous has the variable rate; the average queue size of the isochronous traffic is linearly increasing almost with a constant rate. The scheduler gave the isochronous traffic extra share to cope with its increasing demand in all time frames. But, the video and voice traffics have already very high traffic rates and they have high priorities so there was not enough frame capacity for the isochronous. The priority scheme implicitly depends on cost function coefficients given to the different traffics as well as the upper and lower limits.

Figure 5.4 (d), best-effort has the variable rate; shows that all the four class buffers are full even for the one with the increasing rate (best-effort in this case). This is mainly because the best-effort traffic has the least priority among all traffics; therefore even at the low traffic rate it still has full buffer occupancy. Especially that all the other three classes have high traffic rates from the beginning.

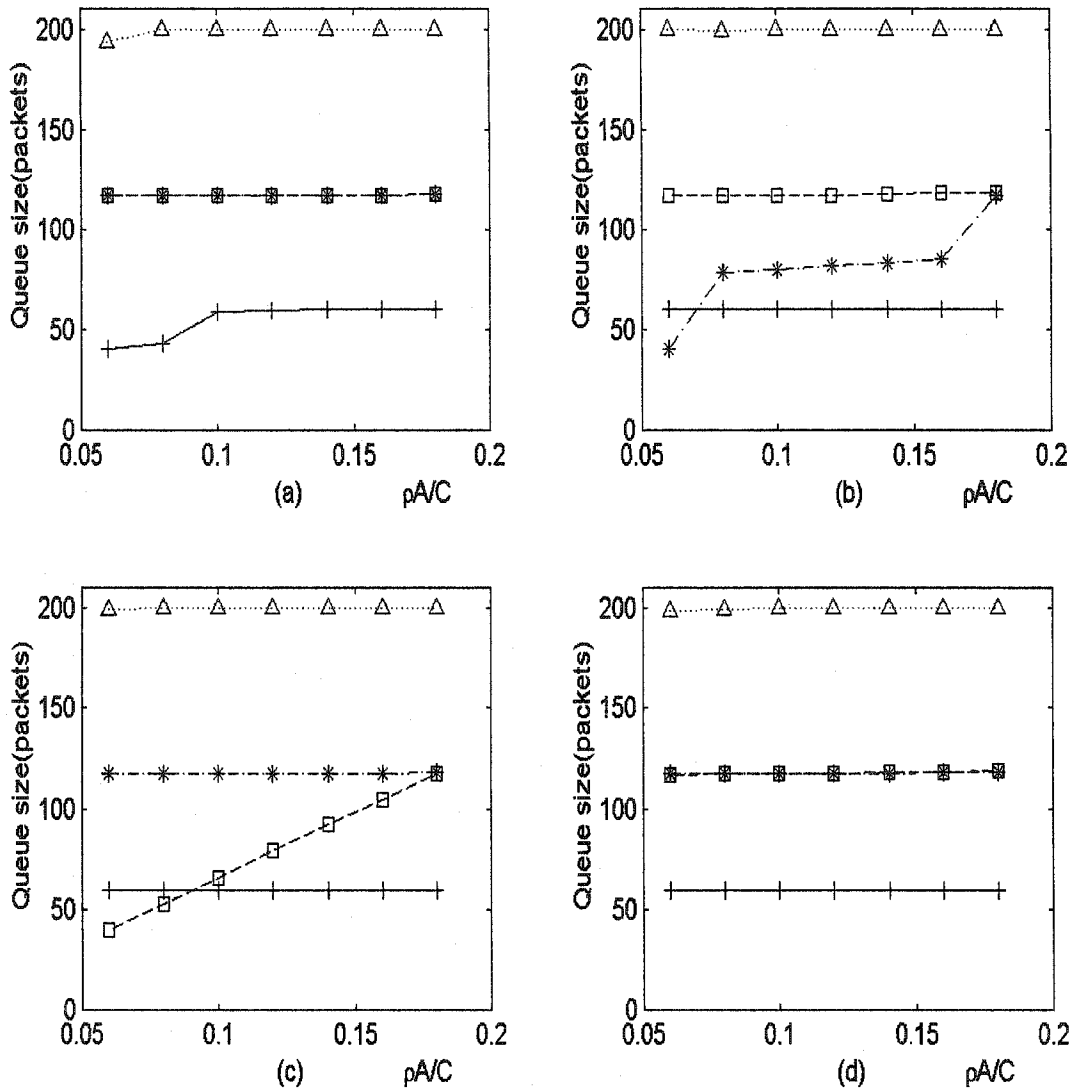

Figure 5.4 Average queue size of each class when the rate of each class is varied; (a) voice traffic (b) video traffic(c) isochronous traffic (d) best-effort traffic. Voice : +, Video : * , Isochronous : □, and Best-effort : Δ.

Figure 5.5 (a), (b), (c), and (d) shows the output link utilization associated with each separate run of Figure 5.4 (a), (b), (c), and (d) respectively (point-by-point). The output link utilization varies between 97% and 99%. It has to be mentioned that due to the statistical nature of the input traffic, the utilization has the variations shown in Figure 5.5. But in general we can say that the overall output link utilization is approximately 98%, which is quite appropriate.

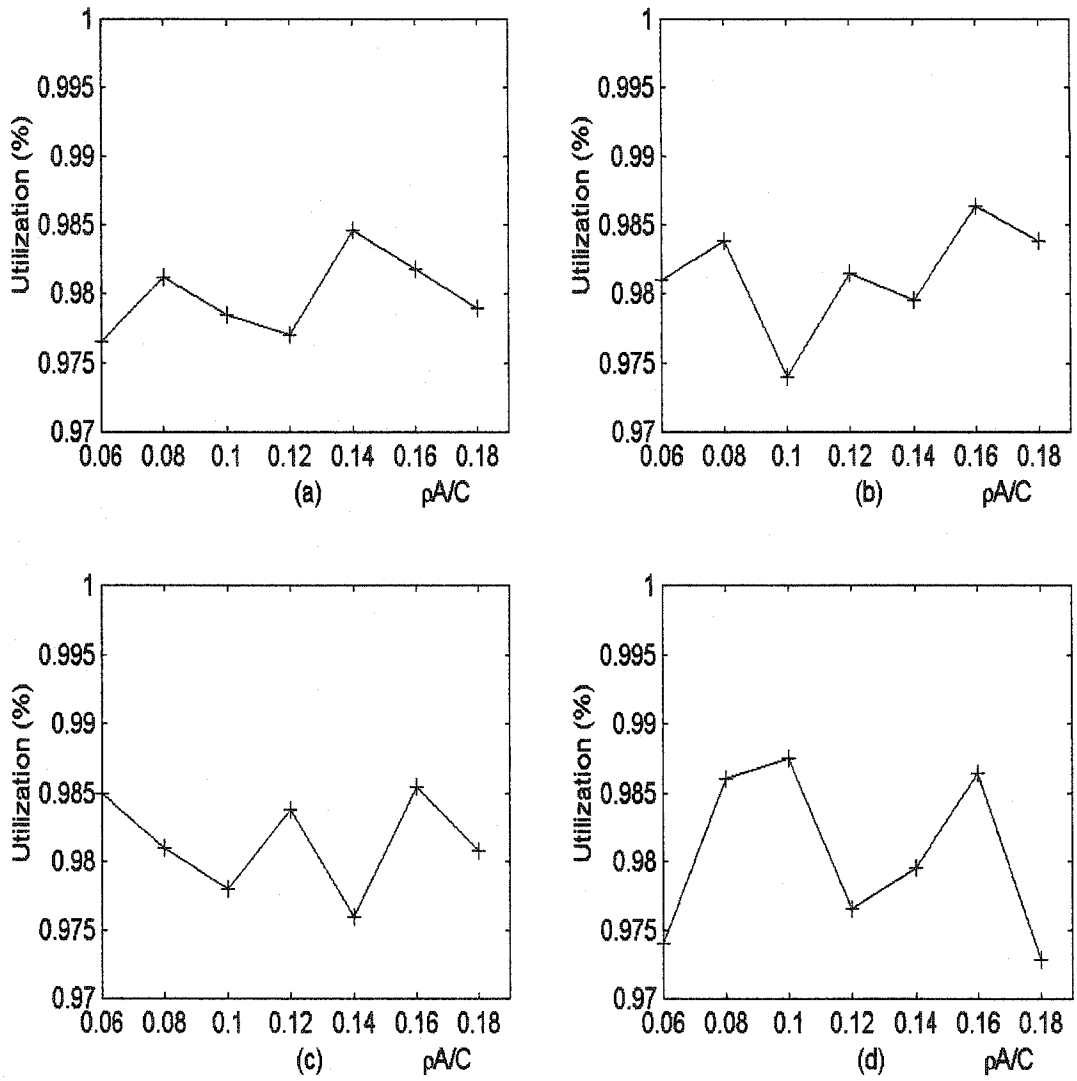

Figure 5.5 Output link utilization associated with the variable rate of different classes corresponding to Figure 5.4 (a) voice traffic (b) video traffic (c) isochronous traffic (d) best-effort traffic.

Figure 5.6 shows the average loss ratio of each class when the rate of each class is varied. Figure 5.6 (a), voice has the variable rate; shows that the average loss ratio for the voice is increasing with the increasing rate of the voice traffic. This is expected since the voice has limited buffer capacity (60 packets). The average loss ratio of the video and isochronous traffics is almost zero. They have the highest reserved minimum share among all the four

classes (0.25% and 0.2% respectively). Best-effort has very high average loss ratio, because it has the least priority with minimum reserved share of 0%.

In Figure 5.6 (b), video has the variable rate; the average loss ratio of the voice is increasing at a lower rate than that of Figure 5.6 (a) but it started already at a high value. The reason for that is that the video that has the highest minimum share has variable rate. Consequently, the voice could have more share in (b) rather than (a) for the empty frame capacity during the portion of the curve at which the video has a low traffic rate. The voice loss ratio started with a high value because it has already a high initial input traffic rate and limited buffer. The video and isochronous have almost negligible average loss ratio. The best effort has the highest average loss ratio. It must be noticed that the average loss ratio for the best-effort started with relatively low value compared to the starting value of average loss ratio of the one shown in Figure 5.6 (a). This is because the traffic with the variable rate in (b) is the video. The video has the highest minimum designated share so there was more available frame capacity for the best-effort in Figure 5.6 (b) rather than the available frame capacity in Figure 5.6 (a).

In Figure 5.6 (c), isochronous has the variable rate; the voice has an increasing rate of loss ratio slightly higher than the increasing rate of loss ratio compared to Figure 5.6 (b). The video traffic has a high rate while the isochronous has the variable rate. Because the reserved minimum share for the video is higher than reserved minimum reserved share for the isochronuse, the voice did not have enough frame capacity as in Figure 5.6 (b). Hence, the loss ratio of the voice has increased. Video and isochronous have low loss ratio, while the best-effort has high loss ratio. The loss ratio of the best-effort has started with low value due to the same reason mentioned above regarding Figure 5.6 (b).

In Figure 5.6 (d), best-effort has the variable rate; the average loss ratio is very high for the best-effort traffic since it has the lowest priority among all classes. The voice traffic has the second highest loss ratio, this is mainly due to two reasons. The first that it has low buffer capacity, and the second because the voice has relatively low designated minimum share.

Taking into account that the video and isochronuse have high traffic rates, it was expected to have a high voice loss ratio according to the pre-set configuration.

Figure 5.7 (a), (b), (c), and (d) show the output link utilization associated with each separate run of Figure 5.6 (a), (b), (c), and (d) respectively (point-by-point). It shows that the output link utilization is ranging again between 97% and 99%.

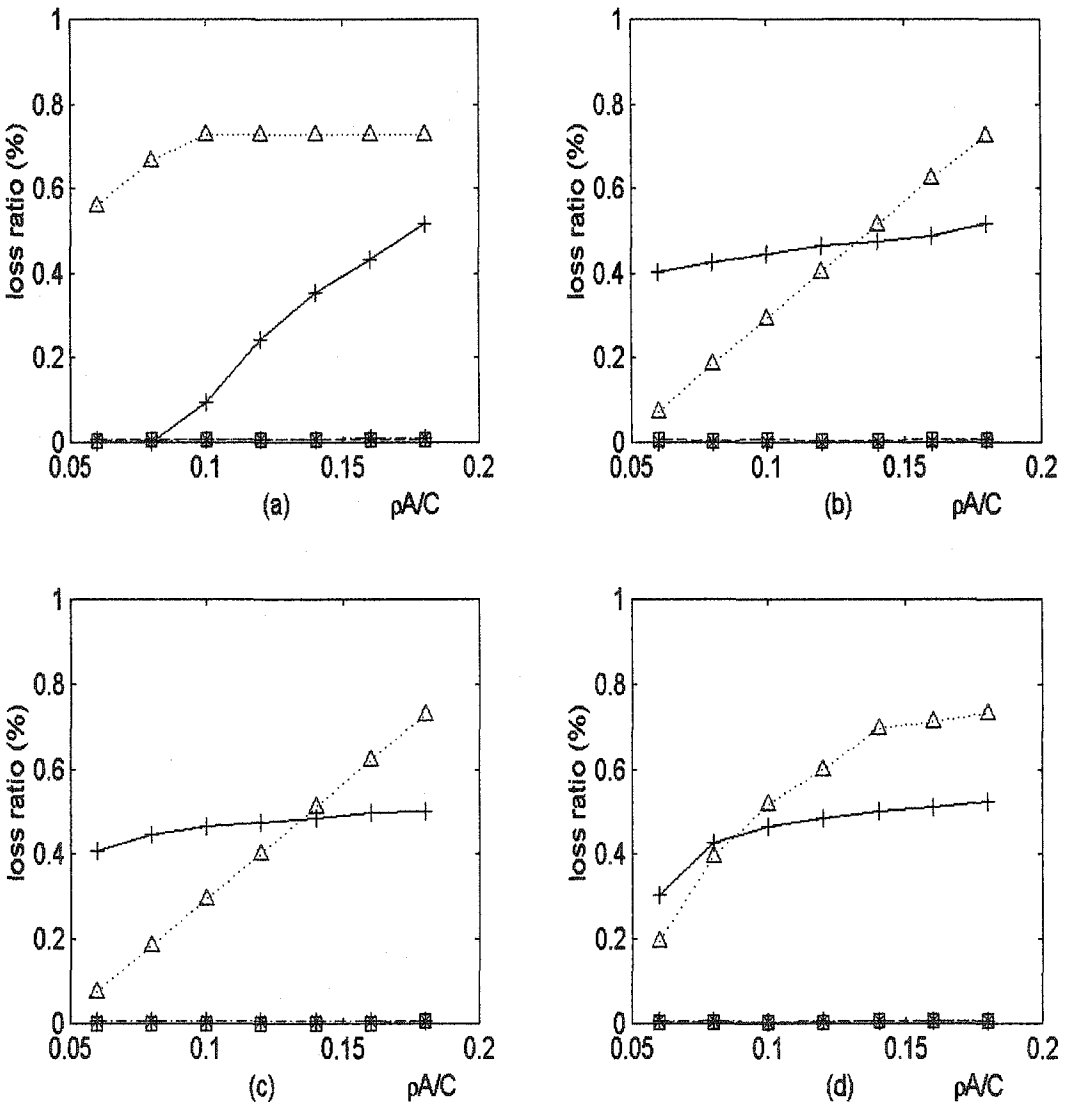

Figure 5.6 Average packet loss ratio for each class when the rate of one of the service classes is varied; (a) voice traffic (b) video traffic (c) isochronous traffic (d) best-effort traffic. Voice : +, Video : * , Isochronous : □, and Best-effort : Δ.

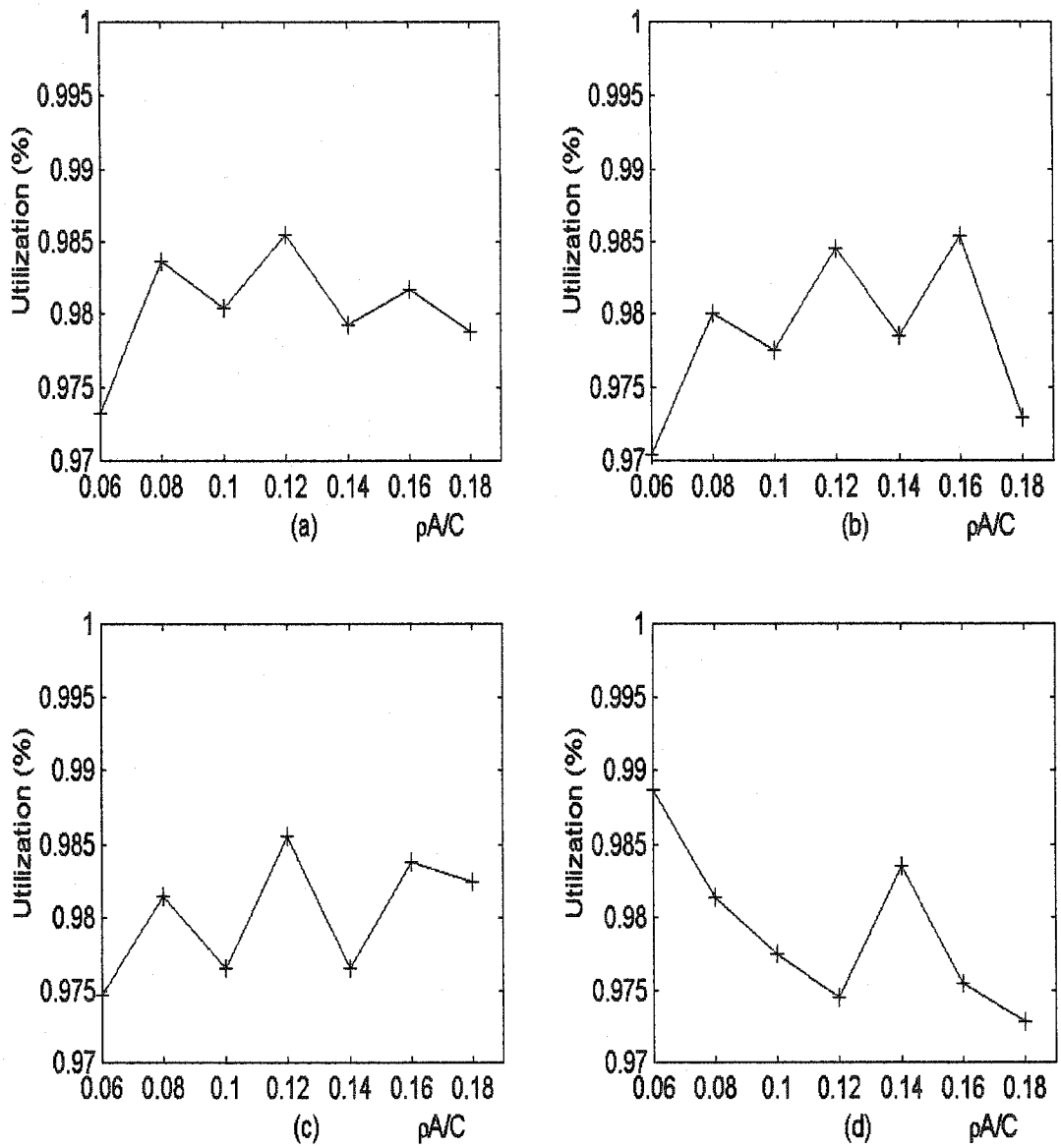

Figure 5.7 Output link utilization associated with the variable rate of different classes corresponding to Figure 5.6 (a) voice traffic (b) video traffic (c) isochronous traffic (d) best-effort traffic.

Figure 5.8 represents the average delay for each class when the rate of each class is varied. In Figure 5.8 (a), voice has the variable rate; the average delay for the voice traffic is quite low in the neighborhood of 1ms. This is due to the high priority of the voice. The average delay of the video and isochronuseare is in the neighborhood of 2-3ms. This is also quite appropriate for both traffics. The average delay of best-effort traffic is in the range of 3-4ms.

The average delay for best-effort traffic is nonlinearly increasing because the voice traffic with low traffic rate has a relatively reserved low minimum share. At the same time the video and isochronous traffics have very high traffic rates and their designated minimum shares are higher than the voice.

In Figure 5.8 (b), video has the variable rate; the average delay for the voice is slightly higher than the average delay for the voice in Figure 5.8 (c). That is mainly due to the higher reserved minimum share for the video (with the variable rate in Figure 5.8 (b)) compared to the reserved minimum share of the isochronous (with variable rate in Figure 5.8 (c)). The average delays for the video and isochronous in Figure 5.8 (b) and (c) are low and almost identical. The average delays for the best-effort in Figure 5.8 (b) and (c) is almost linearly increasing because the traffic with variable rate in both figures has reserved high minimum share (video in Figure 5.8 (b) and isochronous in Figure 5.8 (c)).

In Figure 5.8 (d), best-effort has the variable rate; the average delays for all classes are increasing in a relatively higher rates at the beginning of curve. This happened because the three traffics with the highest reserved minimum shares have very high traffic rates, while the traffic with the lowest designated minimum share has the variable rate in this case. The average delay for the best-effort has a nonlinear rate of increasing due to the bursty case. But, in general the delay for all classes is in the range of few milliseconds, which is quite reasonable for a switch.

Figure 5.9 (a), (b), (c), and (d) shows the output link utilization associated with each separate run of Figure 5.8 (a), (b), (c), and (d) respectively (point-by-point). The results indicate that the output link utilization varies between 97% and 99%. The results of the output link utilization for all of the simulation runs proved that the system resources are very efficiently used under the proposed scheduler. The output link utilization has been always one of the main issues under consideration for any scheduler performance.

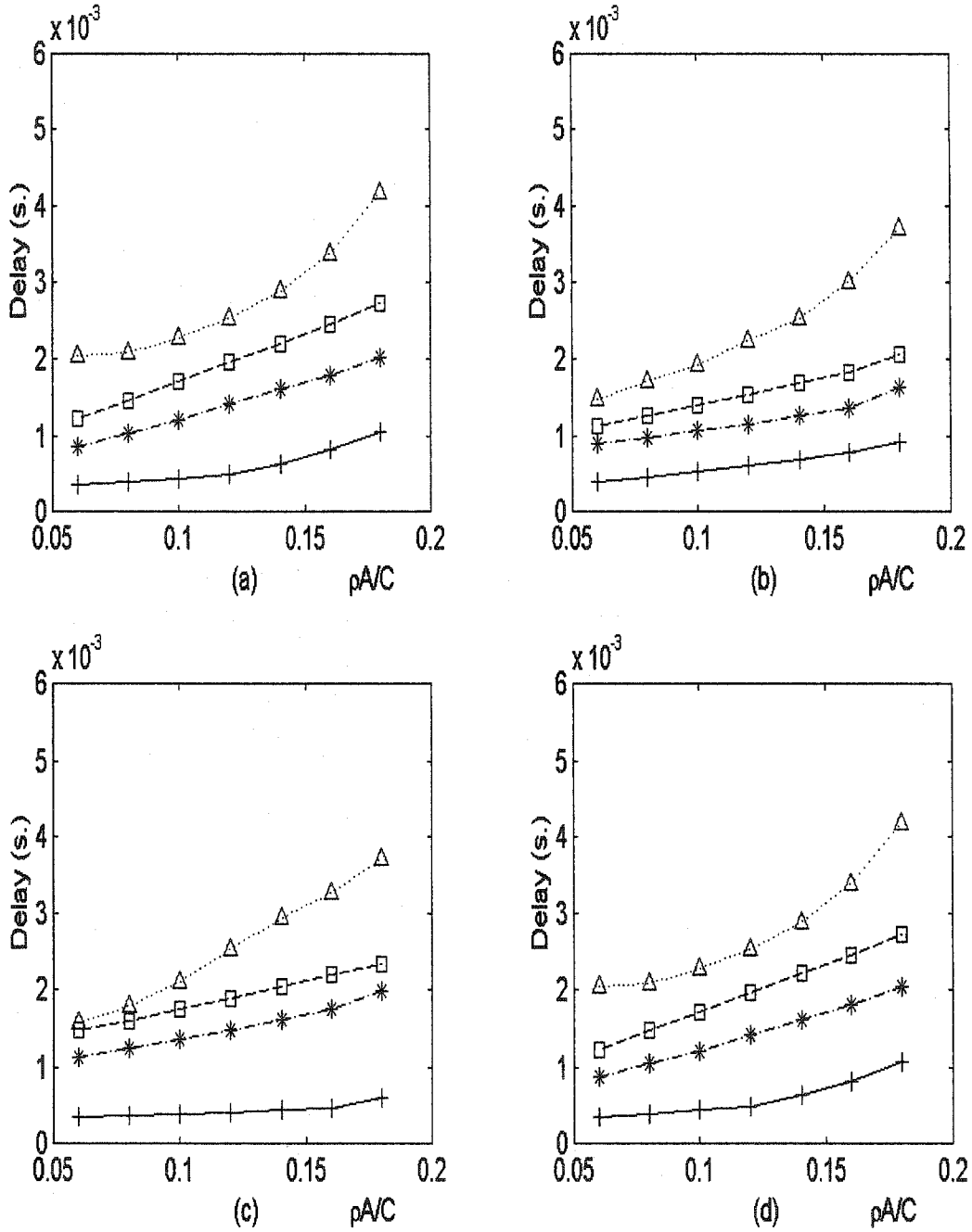

Figure 5.8 Average delay time for each class when the rate of one of the service classes is varied; (a) voice traffic (b) video traffic (c) isochronous traffic (d) best-effort traffic. Voice : +, Video : * , Isochronous : □, and Best-effort : Δ.

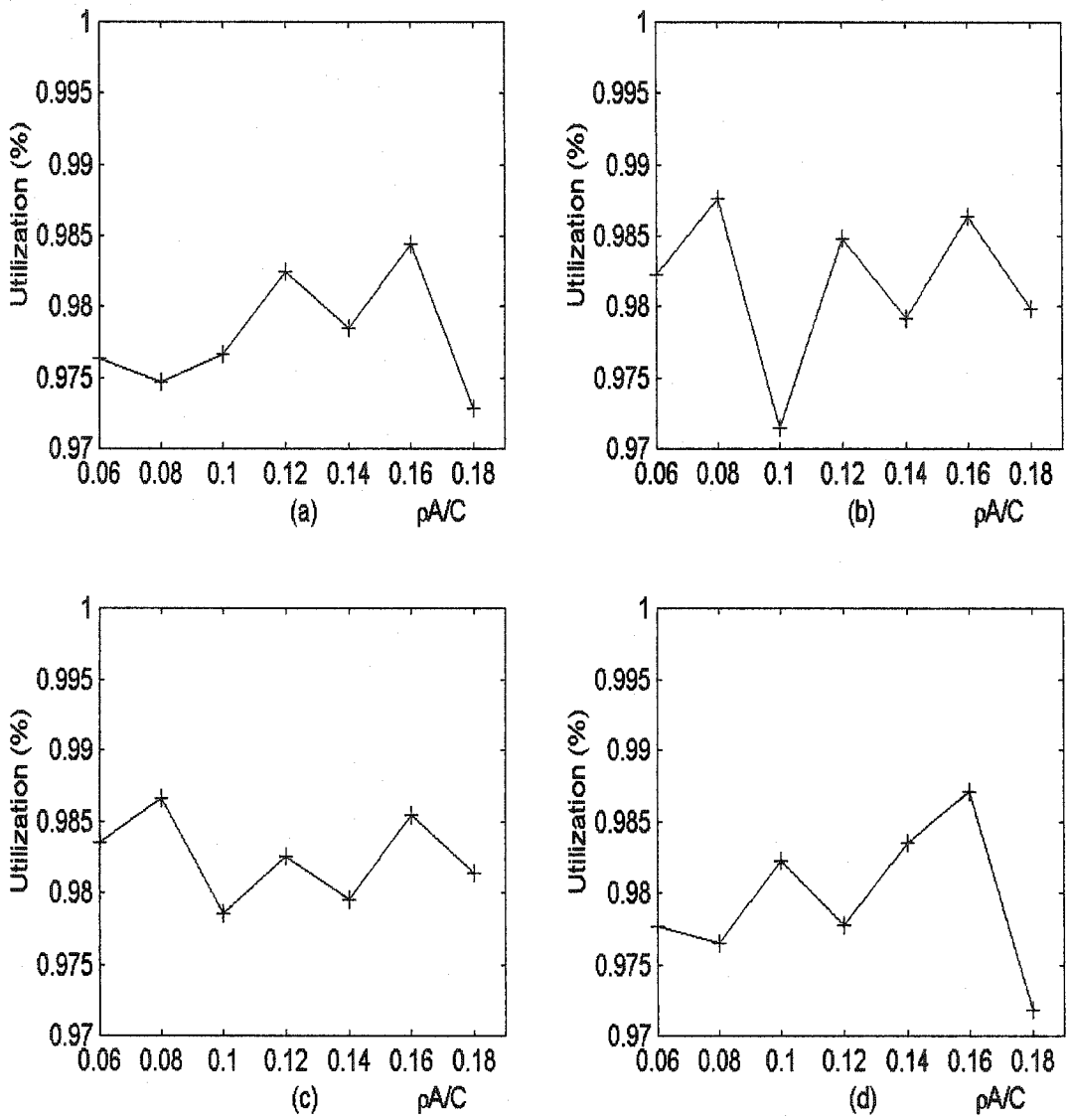

Figure 5.9 Output link utilization associated with the variable rate of different classes corresponding to Figure 5.8 (a) voice traffic (b) video traffic (c) isochronous traffic (d) best-effort traffic.

5.5 The effect of frame duration

This set of results is related to the effect of the frame length (duration in seconds) on the performance of the scheduler. We varied the value of the frame duration from 0.4ms to 1.8ms in a 0.2ms steps. The rates for the four classes were set at their maximum possible values for this test then the four main performance criteria were measured. It is recommended to have a frame size between 1 and 2ms, mainly to minimize the delay for all classes in general.

In Figure 5.10 the four different performance criteria are measured against the variable frame duration. For very low values of frame duration (0.4 and 0.6 ms) the scheduler favored voice and video traffics in the sense that their average share is maximum with respect to the other classes. Also, the average loss ratio and the average delay for both traffics is minimum. On the other hand, isochronous has relatively low share and high loss ratio. Best effort is almost neglected, which is very oppressive for this traffic.

For high values of frame duration (above 1.6 ms) the scheduler starts to act much better for best-effort while the other traffics are highly affected due to the fact that they suffer from a very high loss ratio and relatively high delay.

For moderate values of the frame duration (0.8, 1 and 1.2 ms) the scheduler acts balanced for all traffics. These results strongly lead us to recommend that the frame duration be any of the above-mentioned moderate values. This is recommended mainly to have high flexibility as well as high controllability for driving the scheduler to act as requested with respect to all quality of service requirements regarding all classes. This conclusion is only valid for the system parameters being chosen for the simulation. Figure 5.10 shows that the problem is a trade-off between some contradicting objectives and many constraints.

The implementation of the algorithm can be done either in software or hardware. It is essentially required that the scheduler takes its decision within the minimum possible time. We might try to have parallel processes for both the counter and the scheduler decision to select the right amount of data to be transmitted from each different class in each individual frame. All the above-mentioned considerations imply that the duration of the frame is bounded.

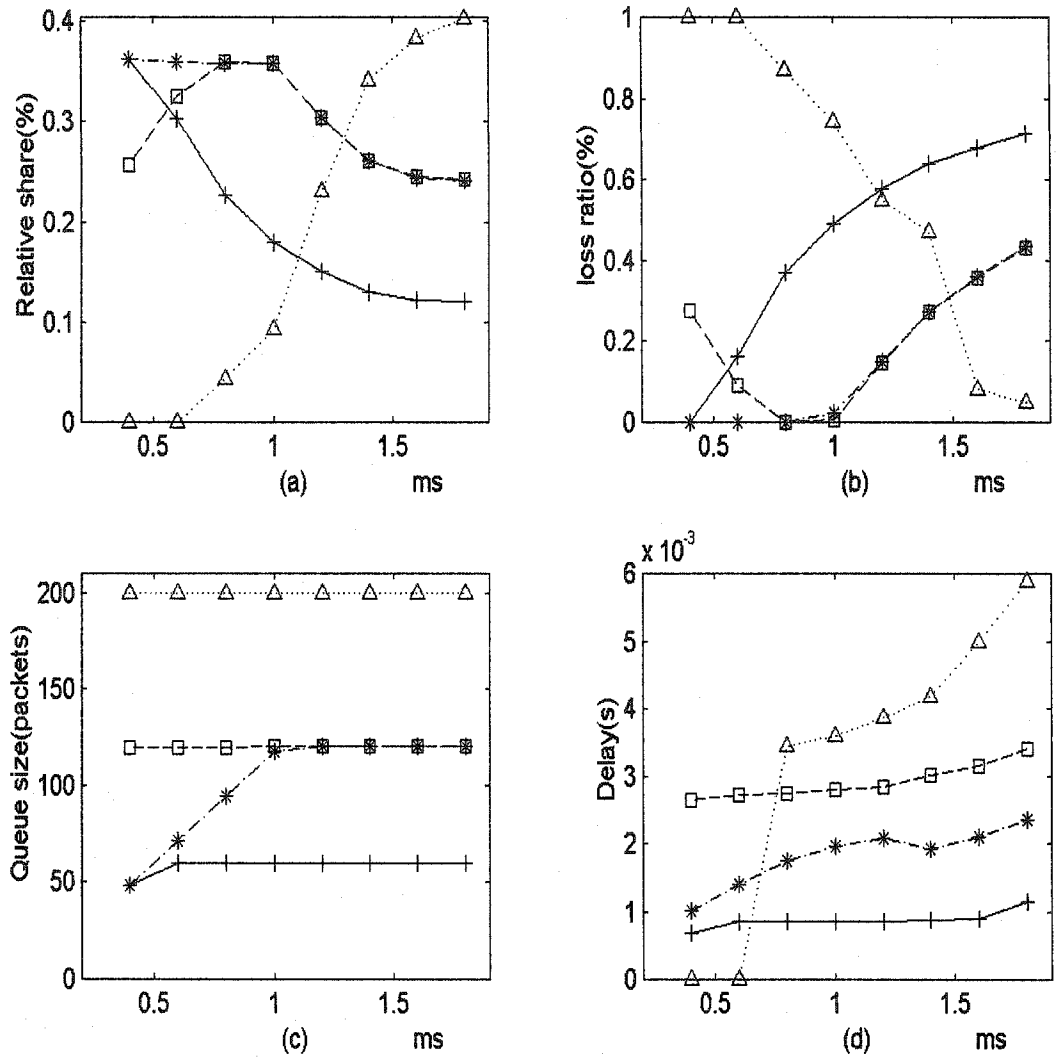

Figure 5.10 The different four performance parameters as measured versus the variable time duration of the frame. (a) relative share (b) average loss ratio (c) average queue size (d) average packet delay time. Voice : +, Video : * , Isochronous : □, and Best-effort : Δ.

5.6 Comparison of the virtual frame algorithm with some other algorithms

In chapter 3 we made three kinds of comparisons among the different well-known schedulers. The first comparison dealt with the mathematical structure for each scheduler, and we made a mapping between them whenever possible. The second comparison focused on the implementation issues from the practical point of view that implicitly discussed the complexity of each algorithm. The third comparison was done regarding the performance

metric (i.e. controllability, scalability, fairness, protection, and quality of service parameters) achieved by each scheduler. As mentioned previously, most of the scheduling techniques are based on the hypothetical model that is known as the general processing sharing (GPS) model. These different algorithms are based on different priority schemes that are implicitly or explicitly derived from stamping or virtual functions. The proposed technique is based on an optimization model that has a totally different nature than all the other techniques. Hence it is meaningless to map them from the mathematical structure point of view. However the comparison can be held for the other two perspectives, namely the performance metric and the complexity of the system.

The controllability accompanied with the proposed algorithm showed great flexibility in forcing the scheduler to act almost as pre-required. Compared to most of the other algorithms' tools of controlling the performance, it seems that the proposed algorithm is the easiest one for doing that. For the required protection and fairness, the proposed algorithm has almost equal capability compared with the other algorithms in providing these two main requirements. Regarding the scalability issue the proposed algorithm can serve many classes, since the price of adding x more classes is only twice x variables which is very reasonable.

One of the most important issues regarding comparison among different algorithms is the complexity. This is quite an essential feature from the practical point of view. The modified model that inspired the virtual frame algorithm is very simple regarding implementation. Although it seems to be a little bit complex due to the optimization nature of the algorithm, in fact it is not since the modified algorithm has reduced the number of variables to double the classes it serves. This step makes the algorithm quite simple to be implemented. Indeed, the other algorithms and the proposed one have many nice features, but what really makes the proposed algorithm quite distinctive with respect to the other algorithms is its complexity. Table 5.1 gives the comparison between the proposed algorithm (VF) and some other algorithms.

The proposed technique has the disadvantage of working at the aggregated level of traffic. The other techniques work at the user level, which has the disadvantage of increasing

complexity thus limiting the possibility of implementation. For the time being the aggregated level quality of service guarantee seems to be quite satisfactory for the Internet.

Property	VF	VCL	FQ	D-EDD	SG	HRR	J-EDD
Throughput Guarantee	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Delay Guarantee	Yes	No	Yes	Yes	Yes	Yes	Yes
Jitter Guarantee	No	No	No	No	Yes	No	Yes
Work Conserving	Yes	Yes	Yes	Yes	No	No	No

Table 5.1 Comparison among the VF algorithm and some other algorithms.

Chapter 6

Conclusions and future work

6.1 Conclusions

The scheduling problem was handled using different algorithms that were based on static and/or dynamic priority levels. In this work, we proposed a totally new scheduler using an optimization technique based on the transportation problem with upper bounds.

Numerical simulations were performed using MATLAB. The simulation results indicated that the proposed algorithm has five degrees of freedom for controlling the performance: (1) cost coefficients, (2) upper bounds, (3) lower bounds, (4) service order within each frame, and (5) frame duration. The proposed algorithm shares many good features with other scheduling algorithms, such as protection, fairness, scalability and controllability of dealing with all scenarios. As well, this technique is simple and can be implemented using software or hardware. The reduction of the number of variables to twice the number of classes made the number of iterations less.

Some recommendations for the network operator can be drawn out from the simulation results. (1) The frame duration should be set within a certain range so as to have a considerable amount of controllability of the scheduler performance. (2) The upper bounds on each service class can be used to manage the output traffic.

6.2 Future work

In future work, we would like to extend the scope of the algorithm to handle the delay jitter problem, to explore choice of the cost coefficients, and to guarantee a per-user quality of service instead of guaranteeing it just for the aggregated level.

The frame capacity for each class share in the proposed algorithm was determined according to an optimized scheme. As well, the order of service within each frame was fixed such as voice packets at the beginning of the frame, followed by video packets, isochronous packets, and finally best-effort packets. That trend in filling the capacity of the frame is not of course the optimum one. In future work we want to use the proposed algorithm in its optimized fashion to have a dynamic order of service within the frame, i.e., according to the different possible scenarios this could be a very effective tool of having more controllability for leading the scheduler to the pre-required performance.

Bibliography

[1] G. A. Awater and F. C. Schoute, "Performance improvement of fast packet switching by LDOLL queuing," in *Proc. Conf. On Computer Communication* (IEEE Infocom), Florence, Italy, vol. 2, pp. 562-568, May 1992.

[2] C. S. Chang, "Stability, queue length and delay deterministic queuing networks," *Res. Rep. RC* 17708 (77962), IBM Res. Div., Yorktown Heights, NY, February 1992.

[3] M.V. Hegde, O. A. Schmid, H. Saidi, P. S. Min, " A new input-output scheduling and flow control protocol for cell-based switches," Tech. Report, Department of Electrical Engineering, Washington University, St. Louis, MO 63130, 1998.

[4] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," *RFC 3031, IETF*, Jan. 2001.

[5] Z. Wang, and J. Crowcroft, "Analysis of Burstiness and Jitter in Real-Time Communications," *In Symp. on Computer Communications Architecture and Protocos (SIGCOM)*, pp.13-18, June 1993.

[6] A. Arora and J. S. Baras, "Performance monitoring in ATM networks," Tech. Rep., CSHCN and ISR, University of Maryland, April 1998.

[7] C. Courcoubetis, V. A.Siris and G. D. Stamoulis, " Integration of pricing and flow control for ABR services in ATM networks". *Proceeding of Globecom'96*, November 1996.

[8] N. Malcolm and W. Zhao, "Advances in hard real-time communications with local area networks," in 17th *IEEE Conf. On Local Computer Networks*, Minneapolis, MN, Sept. 13-16 1990.

[9] L. Breslau, E. Knight, S. Shenker, I. Stoica, and H. Zhang, "Endpoint admission control: architecture issues and performance," *SIGCOMM* 2000.

[10] S. Shakkottai and R. Sirkant, "Scheduling real-time traffic with deadlines over a wireless channel," *Wireless Networks*, pp. 13-26, vol. 8, 2002.

[11] K. Sriram, "Methodologies for bandwidth allocation, transmission scheduling, and congestion avoidance in broadband ATM network," *in Proc. Conf. On global Communications (GLOBECOM)*IEEE, Orlando, Fl, pp. 1545-1551, Dec. 1992.

[12] D.C. Verma, H. Zhang, and D. Ferrari, "Delay jitter control for real-time communication in a packet switching network," *in Proc.Tricomm '91* IEEE, Chapel Hill, NC, Apr. 1991.

[13] M. Shereedhar and G. Varghese, "Efficient fair queuing using deficit round robin," *IEEE/ACM Trans. Networking*, vol. 4, no. 3, pp.375-385, June 1996.

[14] D. D. Clark, S. Shenker, and L. Zhang, "Supporting real-time applications in an integrated services packet network: architecture and mechanism," *In SIGCOMM Symp. On Communications Architecture and Protocols* ACM, Baltimore, MD, pp.14-26, Aug. 1992.

[15] J. M. Hyman, A. A. lazar, and G. Pacific, "Real-time scheduling with quality of service constraints," *IEEE J. Selected Areas in Communications*, vol. 9, no. 2, pp-214-226, 1997.

[16] O. C. Imer, S. Comoans, T. Baser, and R. Sirkten, "ABR congestion control in ATM networks," *IEEE Control Systems Magazine*, Feb. 2001.

[17] J. Kurose, "On computing per-session performance bounds in high-speed multi-hop computer networks ," *In ACM*, New Port, RI, pp.128-139, June 1992.

[18] W. Chen, R. Lee, and H. Lin, "A QoS-Guaranteed and delay-minimized cell multiplexing method in ATM network," *IEEE J. Selected Areas in Communication Networks*, 1998.

[19] D. Ferrari, A. Banerjea, and H. Zhang, "Network support for multimedia: A discussion of the tenet approach," *In Computer Networks and ISDN Systems* 26, pp.1267-1280, 1994.

[20] S. Keshav, "An Engineering Approach to Computer Networking," Addison-Wesley, Aug. 1999.

[21] J. Bennett, and H. Zhang, "Hierarchical Packet Fair Queuing algorithm," *In SIGCOMM Symp. on Computer Communications Architecture and Protocols*, pp.143-155, May 1996.

[22] J.C.R. Bennett and H. Zhang, "WF^2Q: Worst-Case Fair Weighted Fair Queuing," *Proc. IEEE INFOCOM'96*, San Francisco, USA, pp. 120-128, Apr. 1996.

[23] A. Greenberg and N. Madras, "How fair is Fair Queuing?," *In Proceeding Performance' 90*, 1990.

[24] A. Weinrib and L. T. Wu, "Virtual Clocks and Leaky Buckets: flow control protocols for high-speed networks," *In Second IFIP International Workshop on Protocols for High-Speed Networks*, Palo Alto, California November 1990.

[25] Y. Ohba, "QLWFQ: A queue length based weighted fair queuing algorithm in ATM networks," *Proc. IEEE INFOCOM'97*, Kobe, Japan, pp.567-576, Apr. 1997.

[26] T. Kelly, "An ECN probe-based connection acceptance control," *Computer Communication Review*, vol. 31(3), July 2001.

[27] S. J. Golestani, "Congestion-free transmission of real-time traffic in packet networks," *In Proceedings of IEEE INFOCOM'90*, pp.527-542, San Francisco, California, *IEEE Computer and Communication Societies,* June 1990.

[28] D. Stiliadis, and A. Verma, "Latency-rate servers: a general model for analysis of traffic scheduling slgorithms," *In IEEE/ACM Transactions on Networking*, vol. 6, no. 5, 611-624, October 1998.

[29] O. Altintas, Y. Atsumi, and T. Yoshida, "Urgency-Based Round Robin: a new scheduling discipline for packet switching networks," *IEEE Journal on Selected areas in Communications*, vol. 2, pp.1179-1184, 1998.

[30] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a Fair Queuing algorithm, " *In Journal of Internetworking Research and Experience*, pp. 3-26, October 1990. Also I*n Proceeding ACM SIGCOMM'89*, pp. 3-12.

[31] S.J. Golestani, "A self-clocked fair queuing scheme for broadband applications," *Proc. IEEE INFOCOM'94*, Toronto, Canada, pp. 636-646, June 1994.

[32] J. kurose, "Open issues and challenges in providing quality of service guarantees in high-speed networks," *ACM Computer Communications Rev.,* vol. 23, no. 1, pp.6-15. January 1993.

[33] N. Matsufuru and R. Aibara, "Efficient Fair Queuing for ATM networks using uniform round robin," *Proceeding of Infocom '99* (New York), March 1999.

[34] H. Fattah, "Frame based Fair Queuing: Analysis and design," M.Sc. thesis, electrical and computer engineering department, *University of Victoria*, 2000.

[35] M. Gerla, C. Casetti, S. S. Lee, and G. Reali, "Resource allocation and admission control style in QoS diffserv networks," *QoS-IP 2001*, Rome, Italy, Jan. 2001.

[36] C. R. Kalmanek, H. Kanakia, and S. Keshav, "Rate controlled servers for very high-speed networks," *In IEEE Global Telecommunications Conference*, San Diego, California, December 1990.

[37] F. Kelly, P. Key, and S. Zachary, "Distributed admission control," *IEEE Journal on Selected Areas in Communications*, pp. 2617-2628, vol. 18, 2000.

[38] L. Zhang, "A comparison of traffic control algorithms for high-Speed Networks," *In the 2nd annual Workshop on Very High-Speed Networks*, Proceedings Supplement, Greenbelt, Maryland, March 1991.

[39] C. Parris, H. Zhang D, and Ferrari, "A mechanism for dynamic re-routing of real-time channels," Tech. Rep. Tr-92-053, Int. Computer Sci. Inst., Berkeley, Ca, 1992.

[40] D. Ferrari and D. C. Verma, "Scheme for real-time channel establishment in wide-area networks," *IEEE J. Selected Areas Commun.*, vol. 10, no. 6, pp. 368-379, Apr. 1990.

[41] O. Yaron and M. Sidi , "Calculating performance bounds in communication networks," *in Proc. Conf. On Computer Communications* (IEEE Infocom), San Francisco, CA, Mar. 1993.

[42] R. Guerin, H. Ahmadi, and M. Naghshineh, "Equivalent capacity and its application to bandwidth allocation in high-speed networks," *IEEE J. Selected Areas Commun.*, vol. 9, no. 7, pp.968-981, Sept. 1991.

[43] G. M. Woodruff and R. Kositpaiboon, "Multimedia traffic management principles for guaranteed ATM network performance," *IEEE J. Selected Areas Commun.*, vol. 8, no. 3, pp. 437-446, Apr. 1991.

[44] J. Hyman, A. A. Lazar, and J. Pacifici, "Joint scheduling and admission control for ATS-based switching nodes," *in SIGCOMM Symp. on Communications Architectures and Protocol* ACM and IEEE, Baltimore, MD, pp.223-234, Aug. 1994.

[45] H. Kroner, G. Hebuterne. P. Boyer, and A. Gravey, "Priority management in ATM switching nodes," *IEEE J. Selected areas in Comun.*, vol. 9, no. 3, pp.79-89, Apr. 1992.

[46] B. Teitelbaum, "Future priorities for Internet2 QoS," Working Group: papers, http://www.internet2.edu/qoswg/documents.shtml, Oct. 2nd 2001.

[47] R. Chipalkatti, J. F. Kurose, and T. Towsley, "Scheduling policies for real-time and non real-time traffic in statistical multiplexers," *in Proc. Conf. On Computer Communications* (IEEE Infocom) IEEE, Ottawa, Ont., Canada, pp. 774-783, Apr. 1989.

[48] C. Dou and S. Jiang, "Fair scheduling of integrated IP and ATM based traffic mobile communication networks," *Vehicular Technology conference*, VTC spring 2001.

[49] H. T. Kung, "Gigabit local area networks: a system perspective," *IEEE Commun. Mag.*, vol 30, no. 4, pp. 79-89, Apr. 1992.

[50] R. J. Gibbens and F. P. Kelly, "Distributed connection Acceptance control for a connectionless network," *ITC16, Edinburgh*, 1999.

[51] Lixia zhang, "A New Architecture for Packet Switched Network Protocols" Ph.D. dissertation, Massachusetts Institute of Technology, July 1989.

[52] D. Ferrari, "Client requirements for real-time communication services,". *IEEE Communications Magazine*, 28(11), November 1990. Also RFC 1193.

[53] H. Zhang and S. Keshav, "Comparison of rate-based service disciplines," *in SIGCOMM Symp. On Communications Architectures and Protocols* ACM, Zurich, Switzerland, pp. 113-121, Sept. 1991.

[54] S. Keshav, "On the efficient implementation of Fair Queuing," *J. of Internet Working Research and Working Experience*, 1993.

[55] I. S. Rejin, M. Stanojevic, and B. D. Reljin, "Modified Round-Robin scheduler for pareto traffic streams," Telecommunications in Modern Satellite, cable and broadcasting service, TELSIKS, 5th *International IEEE Conference*, vol. 1, pp.25-28, 2001.

[56] S. J. Golestani, "A Stop-and-Go queuing framework for congestion Management," In SIGCOMM'90 Symposium, *Communications Architecture and Protocols*, pp.8-18, Philadelphia Pennsylvania, ACM SIGCOMM, Sept. 1990.

[57] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in hard-real-time environment," *J. of the Association for Computing Machinery*, 20(1):46-61, Jan. 1973.

[58] P. Dharawadker, H. J. Siegel, and E. P. Chong, "A heuristic for dynamic bandwidth allocation with preemption and degradation for prioritized requests," Distributed Computing Systems, 21st *International IEEE Conference*, pp.547-556, 2001.

[59] M.T. Liu, "Distributed loop computer networks," *Advances in Computer*, 17:163-221,1978.

[60] M. de Prycker, "Asynchronous transfer mode: solution for broadband ISDN," Ellis Horwood, 1991.

[61] E. Trevor and F. Vasques, "Non pre-emptive scheduling of messages on SMTV token-passing networks," real-Time Systems, Euromicro RTS 2000, 12th Euromicro Conference, pp.531-538, 2000.

[62] K. Ramamritham, "Channel characteristics in local-area hard real-time systems," *Computer Networks and ISDN Systems*, 13:3-13,1987

[63] K. Arvind, K. Ramamritham, and J. A. Stankovic, "A local area network architecture for communication in distributed real-time systems," *The Journal of Real-Time Systems*, 3(2), May 1991.

[64] M. Teener and R. Gvozdanovic, "FDDI-II operation and architecture," *in Proc. Of the 14th IEEE conference on Local Computer Networks*, pp. 49-61, 1991.

[65] J. K. Y. Ng and J. W. S. Liu, "Performance of local area network protocols for hard real-time applications," *in Proc. Of the 11th International Conference on Distributed Computing Systems*, pp. 318-326, Arlington, Texas, May 1991.

[66] S. Bhulai and G. Koole, "Scheduling time-constrained jobs in the presence of background traffic," Decision and control, *In Proc. Of the* 39th *IEEE conference*, vol. 2, pp. 1421-1426, 2000.

[67] F. S. Hillier and G. J. Lieberman, "Introduction to Operation Research," Seventh edition, McGraw-Hill, 2001.

[68] Hillier, F. S. Hillier, and G. J. Lieberman, "Introduction to Management Science: A Modeling and Case Studies Approach with Spreadsheets," Irwin/McGraw-Hill, Burr Ridge, IL, 2000.

[69] B. Melamed, "TES: a class of methods for generating autocorrelation uniform variates," *ORSA J. Comp.*, vol. 3, no. 4, pp. 317-329, 1991.

[70] A. Veres, and M. Boda, " The chaotic nature of TCP congestion control," *In Proc INFOCOMM*, 2000.

[71] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of ethernet traffic (Extended Version)," *IEEE/ACM Transactions on Networking*, 2(1), pp.1-15, February 1994.

[72] W. S. Cleveland, D. Lin, and D. X. Sun, "IP packet generation: statistical models for TCP start times based on connection-rate superposition," *In Proc. ACM SIGMETRICS 2000*, pp.166-177, 2000.

[73] D. R. Figueiredo, B. Liu, V. Misra, and D. Towsley, "On the autocorrelation structure of TCP traffic," Tech. Rep., *Computer Science Technical report* 00-55, University of Massachusetts, 2000.

[74] A. Papoulis, "*Probability Random Variables, and Stochastic Processes*," 3rd ed., Mcgraw Hill, 1991.

[75] L. Klienrock, "*Queuing Systems*," vol. 1, John Wiley and sons, 1975.

[76] J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun, "On the non-stationary of Internet traffic," *In Proc. ACM SIGMETRICS* 2001, pp. 102-112, 2001.

[77] V. Frost and B. Melamed, "Traffic modeling for telecommunications networks," *IEEE Commununications Mag.*, Mar. 1994.

[78] M. Taqqu, V. Teverovsky and W. Willinger, "Estimators for long-range dependence : an empirical study," *Fractals*, 3(40, pp.785-798, 1995.

[79] R. H. Riedi, M. S. Crouse, V. J. Ribeiro, and R. G. Baraniuk, "A multifractal wavelet model with application," *IEEE Transactions on Information Theory*, vol. 45, no. 3, pp.992-1019, 1999.

[80] B. Madelbort, "The fractal geometry of nature," *Freeman Co.*, San Francisco, 1982.

[81] W. willinger, M. Taqqu, W. Leland, and D. Wilson, "Self-similarity in high-speed packet traffic: analysis and modeling of Ethernet traffic measurements," *Statistical science*, 10(1), pp.76-85, February 1995.

[82] K. Sriram, and W. Whitt, "Characterizing superposition arrival processes in packet multiplexers for voice and data," *In IEEE Journal in Selected Areas in Communications*, vol.-SAC-4, no.6 , Sept. 1986.

[83] J. B. Gao and I rubin, "Superposition of multiplicative multifractal traffic streams," *In Proceedings ICC2000*, 2001.

[84] I. Norros, J. W. Roberts, A. Simonian, J. T. Virtamo, "The superposition of variable bit rate sources in an ATM multiplexer," *IEEE J. Selected Areas in Communications*, vol. 9, no. 3, pp.378-387, April 1991.

[85] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson, "Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level," *IEEE/ACM Transactions on Networking*, 5(1), pp.71-86, January 1997.

[86] J. b. Gao and I. Rubin, "Multiplicative multifractal modeling of long-range-dependent network traffic," *Int. Journal Communication Systems*, vol. 14, pp. 783-801, 2001.

[87] N. Likhanov, B. Tsybakov, and N. Georgans, "Analysis of an ATM buffer with self-similar (Fractal) input traffic," *IN Proc. IEEE INFOCOM'95,* Apr. 1995.

[88] F. Gebali, "*Computer Communication Networks*," North-Star Digital Design, Inc., Victoria, B.C., 2002.

[89] F. Gebali, and E. A. Raheem, "A queuing model for self-similar traffic", *IEEE symposium on signal processing and information technology*, Cairo, Egypt, 28-31 December 2001.

Appendix A

Stochastic processes and traffic modeling

A.1 Introduction

Typically the traffic modeling for any kind of data stream (audio, video,..etc.) is a stochastic process [69], [70]. In 1994, a vital event in the field of network traffic modeling occurred with the publication of the paper titled "On the Self-Similar Nature of Ethernet Traffic" [71]. The authors reported the results of a comprehensive study of Ethernet traffic and demonstrated that it had a self-similar (i.e., fractal) characteristic. This meant the traffic had similar statistical properties at a range of time scales: milliseconds, seconds, minutes, hours, even days and weeks. Another consequence is that the merging of traffic streams, as in a statistical multiplexer (switch), does not result in smoothing of traffic. Again, bursty data streams that are multiplexed tend to produce a bursty aggregate stream. This first paper ignited a huge amount of researches all around the globe.

The results showed the self-similarity in the ATM traffic, compressed digital video streams, and web traffic between browser and servers. Although a number of researchers had observed over the years that network traffic didn't always obey Poisson assumptions used in queuing analysis, this paper for the first time provided an explanation and a systematic approach to modeling realistic data traffic patterns [71].

To date researchers are divided into two camps; one advocated that the entire network engineering has to be rewritten coping with the new perspective of traffic modeling, and the other disagreed. Traditionally, generalized Markovian models have described networks. The main property that is very distinctive for these models that they have limited memory of the past (they are called memoryless). And consequently they reflect short-range dependence. In

a Markovian model, smoothing of bursty data is possible. Averaging of bursty traffic over a long period of time give rise to a smooth data stream. A network based on fractal nature will have very different parameters and congestion control techniques.

On any way the problem has not been solved yet. There is still big argument about the accurate modeling of traffic. We will give more insight on both types of modeling, the traditional way (non-self-similar) and the new self-similar modeling from the mathematical perspective.

From these two main categories mentioned above many traffic models had been inspired [72], [73]. It is not our intention to favor one model or another, but we try to give clear picture about these processes used for traffic modeling. We will discuss the main differences between what is called non self-similar stochastic processes and the self-similar stochastic processes especially those confusing definitions corresponding to both of them.

In our simulation we used one of these processes, which is the renewal process. The parameters used to simulate the renewal process are based on what is called the three-parameter model. At the end of the chapter we show how to derive these parameters for simulation.

A.2 Non self-similar stochastic processes

The main definition of stochastic process states that it is a family of random variables $X(t)$ where the random variables are "indexed" by the time parameter t. For example number of people sitting in a movie theater as a function of time is a stochastic process, as is also the atmospheric pressure in that movie theater as a function of time. It is always referred to a random process as a stochastic process [74], [75]. The classification of random process depends upon three quantities: the state space; the index (time) parameter; and the statistical dependencies among the random variables $X(t)$ for different values of the index parameter t.

The set of possible values (or states) that $X(t)$ may take on is called its state space. If these states or values are discrete, then we say we have a discrete-state process, often referred to as

a chain. On the other hand, if the set values are continuous in nature rather than discrete, then we say that we have a continuous-state space.

For the index (time) parameter, if the permitted times which may take place between successive events are finite or discrete, then we say we have a *discrete-time parameter* process; if the permitted times can take any value (finite or infinite) on the time axis, then we have a continuous-time parameter process.

The crucial distinguishing property of a stochastic process is the relationship of the random variables $X(t)$ or X_n to other members of the same family. One must specify the complete join distribution function among the random variables so as to characterize the full model for the stochastic process under consideration.

There are many ways to classify the stochastic processes. One of these ways is the following [75]:

(a) **Stationary processes.** A stochastic process $X(t)$ is said to be stationary if the cumulative distribution function (cdf) $F_x(x;t)$ is invariant to shifts in time for all values of its arguments; that is, given any constant τ the following must hold:

$$F_x(x;t+\tau)=F_x(x;t) \tag{A.1}$$

Where $F_x(x;t)=P\left[X(t_1)\le x_1,.....,X(t_n)\le x_n\right]$, and the notation $t+\tau$ is defined as the vector $(t_1+\tau,t_2+\tau,.....,t_n+\tau)$.

As associated notation, that of wide-sense-stationary, is defined with the random process $X(t)$ if merely both the first and the second moments are independent of the location on the time axis, that is, $E[X(t)]$ is independent of t and if $E[X(t)X(t+\tau)]$ depends only upon τ and not upon t. Observe that all stationary process are wide-sense stationary, but not conversely.

(b) **Independent processes.** The easiest stochastic process to consider is the random sequence in which $\{X_n\}$ forms a set of independent random variables, that is, the joint

probability density function (pdf) defined for our stochastic process must factor into the product, thusly

$$f_x(x;t) = f_{x_1.....x_n}\left(x_1,.....,x_n;t_1,.....,t_n\right) = f_{x_1}\left(x_1;t_1\right).....f_{x_n}\left(x_n;t_n\right) \qquad (A.2)$$

In this case we are stretching things somewhat by calling such sequence a random process since there is no structure or dependence among the random variables. In the case of a continuous random process, such an independent may be defined, and commonly referred to as *"white noise"*.

(c) **Markov processes.** In the year 1907 A.A. Markov published a paper in which he defined and investigated the properties of what are now known as Markov processes. In fact, what he created was simple and highly useful form of dependency among the random variables forming a stochastic process.

A Markov process with discrete state space is referred to as a Markov chain. The discrete-time Markov chain is the easiest to conceptualize and understand. A set of random variables $\{X_n\}$ forms a Markov chain if the probability that the next values (state) is X_{n+1} depends only upon the current value (state) X_n and not upon any previous values. Thus we have a random sequence in which the dependency extends backwards one unit in time. That is, the way in which the entire past history affects the future of the process is completely summarized in the current value of the process.

In the case of discrete-time Markov chain the instants when the state changes may occur are preordained to be the integers 0,1,2,....*n*,..... In the case of the continuous-time Markov chain, however, the transition between states may take place at any instant in time. Thus there is a new concept of new random variable must be considered which describes how long the process remains in its current state (discrete) before making a transition to some other state (indeed the time variable plays a significant rule in all stochastic processes not only for Markov processes, and in fact there are a bunch of definitions for many time parameters with any stochastic process). Because the Markov property insists that the past history be

completely summarized in the specification of the current state, then we are not free to require that a specification also be given as to how long the process has been in its current state! This imposes a heavy constraint on the distribution of time that the process may remain in a given state. In fact, this state time has been proved to be *exponentially* distributed. In a real sense, then the exponential distribution is continuous distribution, which is "memoryless".

Similarly, in the discrete-time Markov chain, the process may remain in the given state for a time that must be *geometrically* distributed; this is the only discrete probability mass function that is memoryless. This memoryless property is required of all Markov chains and restricts the generality of the processes one would like to consider.

Expressed analytically the Markov property may be written as

$$P\left[X\left(t_{n+1}\right)=x_{n+1}\middle|X\left(t_n\right)=x_n, X\left(t_{n-1}\right)=x_{n-1},\ldots\ldots,X\left(t_1\right)=x_1\right]=P\left[X\left(t_{n+1}\right)=x_{n+1}\middle|X\left(t_n\right)=x_n\right] \quad \text{(A.3)}$$

where $t_1 < t_2 < \ldots\ldots < t_n < t_{n-1}$ and x_i is included in some discrete state space.

(d) **Semi-Markov processes.** The discrete-time Markov chain has the property that at every unit interval on the time axis the process is required to make a transition from the current state to some other state. The transition probabilities are completely arbitrary; however the requirement that a transition be made at every unit time leads to the fact that the time spent in a state is geometrically distributed. This imposes a strong restriction on the kinds of processes we may consider. Now if we wish to *relax* that constraint, namely, to permit an arbitrary distribution of time the process may remain in a state, then we are led directly to the notation of a discrete time semi-Markov process; specifically, we now permit the times state transitions to be an arbitrary probability distribution. Note, however, that at the instants of state transitions, the processes behaves just like an ordinary Markov chain and, in fact, at those instants we say we have an imbedded Markov process.

The same definition of a continuous-time semi-Markov process follows directly. Here we permit state transitions at any instant in time. However, as opposed to the Markov process

that required an exponentially distributed time in state, we now relax this condition. This affords us much great generality. Here, again, the imbedded Markov process is defined at those instants of state transition. Of course, the class of Markov processes is contained within the class of semi-Markov processes.

(e) Special cases of semi-Markov processes.

There are many well-known stochastic processes that can be classified as a part of the Markovian family. In fact, they can be classified under the big title "semi-Markov processes". In the following discussion we will be focusing on some of these processes, such as Random Walks, Renewal process, Birth-Death process, and Poisson process.

(1) Random Walks

A random walk may be thought of as a particle moving among states in some (say, discrete) state space. What is of interest is to identify the location of the particle in that state space. The salient feature of a random walk is that the next position the process occupies is equal to the previous position plus a random variable whose value is drawn independently from an arbitrary distribution; this distribution, however, does not change with the state of the process (except perhaps at the boundary states). That is, a sequence of random variables $\{S_n\}$ is referred to as a random walk (starting at the origin) if

$$S_n = X_1 + X_2 + X_3 + ... + X_n \qquad \text{n} = 1,2,.... \qquad \text{(A.4)}$$

where $S_0 = 0$ *and* $X_1, X_2, ...$ is a sequence of independent random variables with common distribution. The index n merely counts the number of state transitions the process goes through; of course, if the instants of these transitions are taken from discrete set, then we have a discrete-time random walk, whereas if they are taken from a continuum, then we have a continuous-time random walk. It is clear that the name random walk came mainly because it describe the "walk" of a particle in some medium. In the case when the common distribution for X_n is a discrete distribution, then we have a discrete-state random walk; in this case the transition probability p_{ij} of going from state i to state j will depend only upon the difference in indices j-i (which is denoted q_{ij}). An example of random walk is that of

Brownian motion. A random walk is occasionally referred to as a process with "independent increments".

(2) Renewal process

A renewal process is related to a random walk. However, the interest is not following a particle among many states but rather in counting transitions that took place as a function of time. That is, we consider the real time axis on which is laid out a sequence of points; the distribution of time between adjacent points in an arbitrary common distribution and each point corresponds to an instant of a state transition. We assume that the process begins at state 0 [i.e., $X(0)=0$] and increases by unity at each transition epoch; that is; $X(t)$equals the number of state transition that have taken place by *t*. In this sense it is a special case of a random walk in which $q_1=1$ and $q_i=0$ for $i\neq 1$. We may think of Equation (A.4) as describing a renewal process in which S_n is the random variable denoting the time at which the *nth* transition takes place. As earlier, the sequence $\{X_n\}$ is a set of independent identically distributed random variables where X_n now represents the time between the *(n-1)th* and *nth* transition. One should be careful to distinguish the interpretation of Equation (A.4) when it applies to renewal processes as here and when it applies to random walk as earlier. Simply in the random walk we care about the state in which the particle is found but in the renewal process we care about the time elapsed until reaching some other state.

(3) Birth-Death process

A very important special case of Markov chains has come to be known as the birth-death process. These may be either discrete- or continuous-time processes in which the defining condition is that transition take place between neighboring states only. That is, one may choose the set of integers as the discrete state space (with no loss of generality) and then the birth-death process requires that if $X_n=i$, then $X_{n+1}=i-1, i \text{ or } i+1$ and no other.

(4) Poisson process

The heart of all the Markovian processes is the Poisson Process. Simply it enjoys all the nice properties of all Markovian processes in the sense that it has a lot of simple relations

between both the states and the time. It has been used in many applications so far because of its simplicity, and specially the queuing analysis. It is the easiest among all of them to be modeled and simulated.

A.3 Self-similar stochastic processes and network traffic

The meaning of self-similar is kind of ambiguous one in the sense of the stochastic processes. And so as to clarify this terminology regarding the stochastic processes we will have to go backward to the origin of that terminology as well as some other related words which may be confusing. The expressions self-similar as well as those other terms are not necessary to have the same meaning exactly but by some mean they are closely related. In the following we will be concerned about these terminologies; fractal, scaling, long-range dependence, and heavy tailed distribution [76]- [79]

A.3.1 Fractal network traffic

The notion of Fractal, a word invented by Mandelbrot [80], has its origin in geometry. Fractals are mathematical objects that posses an order of irregularity and fragmentation inherent in many patterns of nature such as clouds, mountains, and coastlines. In a more formal manner, fractals are objects that posses a form of self-similarity; parts of the whole can be made to fit to the whole in some way by scaling. Therefore, scaling and fractal are closely related. The close relation between scaling and fractal in geometry also holds in one-dimensional stochastic processes. One definition of a fractal stochastic process is one in which the sample paths of the stochastic process have non-integer dimensions; the expected measure of the sample path included within some radius scales with the size of the radius. Since this is but one statistic of the process, we call a stochastic process fractal if several of the relevant statistics exhibit scaling.

At the same time, we may call a stochastic point process fractal when a number of the relevant statistics exhibit scaling with related scaling exponents, indicating that the represented phenomenon contains clusters of points overall (or a relatively large set of) time or length scales. Since scaling leads mathematically to the power-low relations in the scaled

quantities, a fractal process often has its statistics (more than one) mathematically expressed by power-low functions.

It is natural to view network traffic as a realization of a stochastic point process; each packet (or cell) arrival is associated with an arrival epoch. Then it follows that the network traffic subject to study may be called fractal when several estimated statistics exhibit power-law behavior over a wide range of time and frequency scales. Indeed, many recent high-quality measurement studies on local area network (LAN) [81], and wide area network (WAN) [82], [83], provide prolific evidence that this is the case regardless of where and when the measurement took place, the type of traces, and coding methods.

A.3.2 Self-similarity

A continuous-time stochastic process $Z(t)$ is self-similar with a self similarity parameter H ($0<H<1$) if $Z(at)$ and $a^H Z(t)$ have identical finite-dimensional distribution, i.e.,

$$\left\{Z\left(a_{t_1}\right), Z\left(a_{t_2}\right),\ldots., Z\left(a_{t_n}\right)\right\} \approx \left\{a^H Z\left(t_1\right), a^H Z\left(t_2\right),\ldots, a^H Z\left(t_n\right)\right\} \quad \text{(A.5)}$$

for all finite positive integer n. Note that this definition of self-similarity is in a distributional sense, quite different from that of a fractal process which is defined in a statistical sense. When this self-similar process $Z(t)$ has the property of stationary increments, i.e., the finite-dimensional distribution of $Z(t_0+t) - Z(t_0)$ do not depend on t_0 we can construct a stationary increments process $X=\{X_1, X_2, \ldots\}$ defined as

$$X_n \equiv Z(nT_s) - Z[(n-1)T_s] \quad \text{(A.6)}$$

for some positive constant T_s . Then, this discrete-time increment process X will have an autocorrelation function of the form of Equation (A.11) for $0.5<H<1$, making it a long-range dependent process. Therefore, the self-similar process $Z(t)$ with stationary increments serves as an underlying process yielding a fractal process. For example, Fractional Brownian Motion process with $0.5<H<1$ has been used as an underlying process to yield a long-rang dependent process (Fractional Gaussian Noise process) by Norros [84].

A.3.3 Long-range dependence

We provide three definitions of long-range dependence LRD in the literature, in order of decreasing generality. Let $X=\{X_k: k=0,1,2,.....\}$ be a wide-sense stationary (WSS) process in the discrete-time domain with mean $\mu \equiv E[X_k]$, variance $\sigma^2 \equiv E\left[(X_k-\mu)^2\right]$ and (normalized) autocorrelation function

$$r(k;\ T_s) \equiv \frac{E\left[(X_n-\mu)(X_{n+k}-\mu)\right]}{\sigma^2} \tag{A.7}$$

Here X_k might represent the number of packets, cells or bytes that have arrived during the k-th time interval of size T_s sec. Note that X_k is obtained by

$$X_k = N[kT_s]-N[(k-1)T_s] \tag{A.8}$$

when constructed from an underlying counting process $N(t)$ [78], [85]. The process X is said to be a long-range dependent (LRD) process if for fixed T_s its autocorrelation function $r(k;\ T_s)$ is non-summable [84], [104], i.e.,

$$\sum_{k=0}^{\infty} r(k;\ T_s) = \infty \tag{A.9}$$

This broad definition depends only on the sum of the autocorrelation function, and not on its specific form; thus it reveals little information about the process X.

Because the behavior of the tail of $r(k;\ T_s)$ completely determines its summability, the details of how $r(k;\ T_s)$ decays with k engender much interest. The second and most widely employed definition of LRD is given by

$$r(k;\ T_s) \sim f(T_s)k^{-(2-2H)}, \qquad \text{as } k \to \infty \tag{A.10}$$

with $0.5<H<1$ and $f(T_s)$ a positive function independent of k. The quantity H is called the Hurst parameter and completely characterizes the relation (A.10). We introduce *f(.)* in (A.10)

to emphasize the dependence of the autocorrelation function on the time scale T_s in addition to the lag k. The time scale T_s appears to impart specific properties to X depending in turn on the nature of the underlying physical process from which X was constructed.

Equation (A.10) specifies the form of the tail of the autocorrelation function, and is a special case of (A.9). However, the behavior of $r(k;\ T_s)$ for small k remains arbitrary.

A third definition specifies $r(k;\ T_s)$ explicitly for all $k \geq 0$:

$$r(k;\ T_s) = \begin{cases} 1 & for\ \mathrm{k} = 0 \\ \mathrm{g}(T_s).\frac{1}{2}\nabla^2\left(k^{2H}\right) & for\ \mathrm{k} > 0 \end{cases} \qquad \text{(A.11)}$$

with $\nabla^2(h(k)) \equiv h(k+1) - 2h(k) + h(k-1)$ the second central difference operator, and $g(T_s)$ a positive function independent of k. It follows immediately that $f(T_s) = H(2H-1)g(T_s)$ due to the asymptotic equivalence between difference and differentiation for large k. Thus (A.11) is a special case of (A.10) and therefore of (A.9) as well [86].

In order to distinguish (A.10) and (A.11) we call X an asymptotic LRD process when it satisfies (A.10) and an exact LRD process when it satisfies (A.11). For example the discrete-time Fractional Gaussian Noise (FGN) process is an exact LRD process with $g(T_s) = 1$.

A.3.4 Heavy-tailed distributions

When self-similarity concerns scaling properties of time-dependent such as the autocorrelation function, the heavy-tailed property concerns the marginal amplitude distributions of X_k defined in (A.8) inter-arrival times, or ON/OFF periods. A random variable U is said to be heavy-tailed if its survivor or complementary distribution has the following form

$$Pr\{U \geq u\} \sim u^{-\alpha}, \qquad \text{as u} \to \infty \qquad \text{(A.12)}$$

for some $\alpha > 0$. Note that for $0 < \alpha \leq 1$ all moments of U are infinite, and in general the *n-th* moment will be infinite for $n \geq \alpha$.

The heavy-tailed property is widely conceived to be closely associated with self-similarity but it is not. The heavy-tailed property is not necessary to be associated with the self-similarity [87].

A.4 The model used for input traffics

As stated previously, typically a traffic model is a stochastic process that is based on a set of parameters. In this work, we model four types of traffic. These four types of traffic are video, voice, isochronous and best effort traffic. There are many approaches for traffic modeling as mentioned before. And, there is a debate about the proper model that best matches the actual real life situation.

In this work, we will be using what is called the three-parameter model (ρ_c,σ , *p)* for describing the traffic generation sources [88], [89]. ρ_c represents the conforming rate of the source, σ represents the peak rate of the source, and finally the conforming probability p which represents the percentage of the time the incoming traffic rate is below a pre-specified rate ρ_c. There are direct relations between these three parameters and the main parameters of the probability density function (pdf) describing the inter-arrival time for the different streams.

Figure A.1 shows how these three parameters are related to the pdf of the inter-arrival time. The area under the curve to the right of the shaded line is equal to the conforming probability.

The pdf for the inter-arrival time can be written in the form of

$$f_T(t) = b \exp\text{-}b(t-a) \tag{A.13}$$

Where a is called the position parameter, and b is called the shape parameter.

The position parameter a can be written as [88], [89]:

$$a = 1/\sigma \tag{A.14}$$

The average rate ρ_a can be written in terms of the given three parameters as:

$$\rho_a = p\ \rho_c + (1-p)\sigma \tag{A.15}$$

The shape parameter b is given by:

$$b = \sigma\rho_a / (\sigma - \rho_a) \tag{A.16}$$

The packets have different sizes. In fact, any distribution can be applied for the packet size and the one used in this study is the normal distribution with certain average as well as pre-given standard deviation.

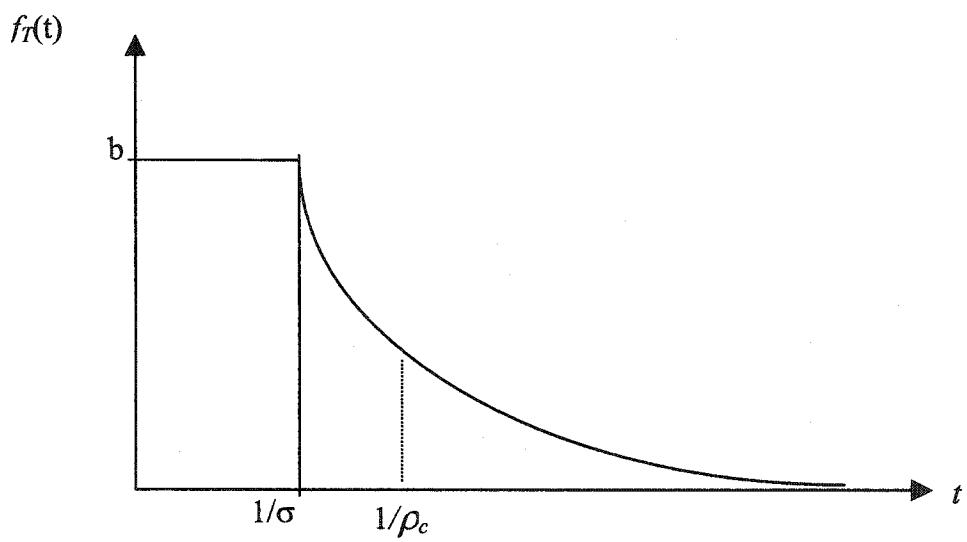

Figure A.1 The Pdf for inter-arrival time.

Appendix B

Simulation code

B.1 The virtual frame algorithm. MATLAB file

```
% MAIN_LP.M : scheduler implementation using linear programming
% We work by the buffers of each service class.
% We assume four service classes.
% Our solution vector has 8 components and is partitioned into two
%equal parts. First part is the transmitted data
% and second part is deferred data (virtual frame).
% class 1 is assumed to be voice
% class 2 is assumed video
% class 3 is assumed isochronous
% class 4 is assumed best-effort
% define format for displaying numbers:
format short e

% Define the number of supported service classes
classes = 4;
dimension = 2*classes;
% Define objective function coefficients
c = zeros(dimension,1);

% Define the maximum number of frames to consider
max_frame_index = 10;

% Define output link and f rame parameters
C = 155*10^6/8; % output line capacity bytes/s
duration = 0.5*10^(-3); % duration of frame in seconds
F = floor(C*duration); % bytes carried in transport frame


% class 1 black class 2 blue class 3 green class 4 red

length_mu =[200, 200, 200, 200]; % Average packet length
length_var = [1*10^2 1*10^2 1*10^2 1*10^2] ; %variance in bytes
alpha =[3, 3, 3, 3];% source burstiness
% data rate in packets/s when source is conforming
rho = C*[(18/100) (18/100) (18/100) (18/100)]./length_mu;
sigma=rho.*alpha; % burst data rate in packets/s
```

```
% ensure that interarrival time is larger than duration of longest
packet
for j=1:classes
    if sigma(j) > C/(length_mu(j)+5*sqrt(length_var(j)))
        j
        error('burst rate for class is too high')
    end
end
p = [0.7 0.7 0.7 0.7]; %Probability that the source is conforming
rho_a = p.*rho + (ones(1,classes)-p).*sigma;

a = ones(1,classes)./sigma; % Poisson position parameter, units of
seconds/packet
for j=1:classes
    b(j)=sigma(j)*rho_a(j)/(sigma(j)-rho_a(j)); % Poisson rate
parameter, units packets/second
 end %This is the main modification done for the formula and it is
dramatic change
 %%%%%%%%%%%%%%%%%%%%%%%%%khos balak%%%%%%%%%%%%%%%%%%%%%%%%%%%%
B = [40 40 40 40]; % maximum buffer size in packets

%Define lower and upper limits
U = F*[1 1 1 1];
L = F*[0.1 0.25 0.2 0.0];
c1 = [1   1   1  1]; % transmitted data cost
c2 = [6   5   4  3]; %deferred data cost


% Define the size of the buffer status locations
status = 15;
% Initialize the arrays for each buffer
% buffer_i = [
%  arrival time,              packet size  frame index
%  arival time,              packet size  frame index
%       :                         :          :
%  arrival time,              packet size   frame index
%              ----------------------------------------
 % buffer_size+1     start index                 end index
% buffer_size+2     occupancy (packets)         occupancy(bytes)
% buffer_size+3     bytes_rcv_f                 bytes_rcv_t
% buffer_size+4     packets_rcv_f               packet_rcv_t
% buffer_size+5     bytes_xmt_f                 bytes_xmt_t
% buffer_size+6     packets_xmt_f               packets_xmt_t
% buffer_size+7     bytes_lost_f                bytes_lost_t
% buffer_size+8     packets_lost_f              packets_lost_t
% buffer_size+9     eta_b_f                       eta_b_t
% buffer_size+10    eta_p_f                     eta_p_t
% buffer_size+11     loss_b_f                     loss_b_t
% buffer_size+12    loss_p_f                    loss_p_t
% buffer_size+13     delay_f                     delay_t
% buffer_size+14    delay_a_f                     delay_a_t
% buffer_size+15     empty_flag                  full_flag
%               ]
%
% Start index indicates location of first packet that arrived in
buffer.
% End index indicates location of first empty location in buffer.
```

```
% s = e ==> array is empty or completely full
% s < e or s > e ==> array has data in it
% number of bytes in buffer is given by
% mod(e-s, B_max)
%
buffer_1 = zeros (B(1)+status, 3);
buffer_1(B(1)+1,1) = 1; % start index at first element of matrix
buffer_1(B(1)+1,2) = 1; % end index at first element of matrix
buffer_1(B(1)+status,1) = 1; % initially buffer is empty
for i=1:status
    buffer_1(B(1)+i, 3) = i;
end

buffer_2 = zeros (B(2)+status, 2);
buffer_2(B(2)+1,1) = 1;
buffer_2(B(2)+1,2) = 1;
buffer_2(B(2)+status,1) = 1; % initially buffer is empty
for i=1:status
    buffer_2(B(2)+i, 3) = i;
end

buffer_3 = zeros (B(3)+status, 2);
buffer_3(B(3)+1,1) = 1;
buffer_3(B(3)+1,2) = 1;
buffer_3(B(3)+status,1) = 1; % initially buffer is empty
for i=1:status
    buffer_3(B(3)+i, 3) = i;
end

buffer_4 = zeros (B(4)+status, 2);
buffer_4(B(4)+1,1) = 1;
buffer_4(B(4)+1,2) = 1;
buffer_4(B(4)+status,1) = 1; % initially buffer is empty
for i=1:status
    buffer_4(B(4)+i, 3) = i;
end

for k=1:max_frame_index


    % calcualte the starting time of each frame
    frame_start_time = (k-1)*duration;
    frame_end_time = k*duration;



    % update the buffers at frame k
    %disp('input to buffer 1')
    buffer_1 = traffic_in(k,frame_start_time,duration, buffer_1,
status, a(1),b(1),length_var(1), length_mu(1));
    %disp('input to buffer 2')
    buffer_2= traffic_in(k,frame_start_time,duration, buffer_2, status,
a(2),b(2),length_var(2), length_mu(2));
    %disp('input to buffer 3')
    buffer_3 = traffic_in(k,frame_start_time,duration, buffer_3,
status, a(3),b(3),length_var(3), length_mu(3));
    %disp('input to buffer 4')
```

```
    buffer_4 = traffic_in(k,frame_start_time,duration, buffer_4,
status, a(4),b(4),length_var(4), length_mu(4));

    % calculate the buffer occupancies in bytes
    size_b(1,k) = buffer_1(B(1)+2,2); % buffer size in bytes
    size_b(2,k) = buffer_2(B(2)+2,2); % buffer size in bytes
    size_b(3,k) = buffer_3(B(3)+2,2); % buffer size in bytes
    size_b(4,k) = buffer_4(B(4)+2,2); % buffer size in bytes

    % Define upper and lower limits on decision variables.
    sum_B = 0;
    sum_upper = 0;
    for i=1:classes
        Upper(2*i-1) = min([size_b(i,k),U(i)]);
        Lower(2*i-1) = min([size_b(i,k),L(i)]);
        Upper(2*i) = max([size_b(i,k)-Lower(2*i-1), 0]);
        Lower(2*i) = max([size_b(i,k)-Upper(2*i-1), 0]);
        sum_B = sum_B+size_b(i,k);
        sum_upper = sum_upper+Upper(2*i-1);

    end

    E = min(sum_upper, F);

    % Define matrices A and b
    A_ineq = []; b_ineq = []; % we do not have inequality conditions.

    Aeq = zeros(classes+2, dimension);
    beq = zeros(classes+2,1);
    beq(1) = min(sum_upper,F);
    beq(2) = sum_B-beq(1);
    for j=1:classes
        Aeq(1,2*j-1)=1; % first row
        Aeq(2,2*j) = 1; % second row
        Aeq(j+2,2*j-1)=1;
        Aeq(j+2,2*j)=1;
        beq(j+2) = size_b(j,k);

    end

    % construct the cost function matrix out of c1 and c2
    for j=1:classes
        c(2*j-1) = c1(j);
        c(2*j) = c2(j);
    end
    % Apply scheduler to first frame
    [x,fval] = linprog(c,A_ineq,b_ineq,Aeq,beq,Lower,Upper);

    % disp('output traffic for first frame')
    % Start transmitting data out of buffers
    % disp('buffers after transmission')

    % Now we service the buffers in random order at each
    % frame to equalize performance
    %This was mainly done to check about having optimumsolution

    if size(x,1) > 0
```

```
        %disp('buffers after output')

        %disp('output from buffer 1')
        [buffer_1, exit_time_1] =
traffic_out(C,x(1),frame_end_time,buffer_1, status);
        %disp('output from buffer 2')
        [buffer_2, exit_time_2]  =
traffic_out(C,x(3),exit_time_1,buffer_2, status);
        %disp('output from buffer 3')
        [buffer_3, exit_time_3]  =
traffic_out(C,x(5),exit_time_2,buffer_3, status);
        %disp('output from buffer 4')
        [buffer_4, exit_time_4]  =
traffic_out(C,x(7),exit_time_3,buffer_4, status);
    end

 % Define performance data arrays, and collection of the statistics
 % transmitted bytes/frame
    Th_b (1,k) = buffer_1(B(1)+5,1);
    Th_b (2,k) = buffer_2(B(2)+5,1);
    Th_b (3,k) = buffer_3(B(3)+5,1);
    Th_b (4,k) = buffer_4(B(4)+5,1);

    % transmitted packets/frame
    Th_p (1,k) = buffer_1(B(1)+6,1);
    Th_p (2,k) = buffer_2(B(2)+6,1);
    Th_p (3,k) = buffer_3(B(3)+6,1);
    Th_p (4,k) = buffer_4(B(4)+6,1);

    % estimate of total eta in bytes
    eta_b_f(1,k) = buffer_1(B(1)+9,1);
    eta_b_f(2,k) = buffer_2(B(2)+9,1);
    eta_b_f(3,k) = buffer_3(B(3)+9,1);
    eta_b_f(4,k) = buffer_4(B(4)+9,1);

    % estimate of total eta in bytes
    eta_b_t(1,k) = buffer_1(B(1)+9,2);
    eta_b_t(2,k) = buffer_2(B(2)+9,2);
    eta_b_t(3,k) = buffer_3(B(3)+9,2);
    eta_b_t(4,k) = buffer_4(B(4)+9,2);

    % estimate of total eta in packets
    eta_p_f(1,k) = buffer_1(B(1)+10,1);
    eta_p_f(2,k) = buffer_2(B(2)+10,1);
    eta_p_f(3,k) = buffer_3(B(3)+10,1);
    eta_p_f(4,k) = buffer_4(B(4)+10,1);

    % estimate of total eta in packets
    eta_p_t(1,k) = buffer_1(B(1)+10,2);
    eta_p_t(2,k) = buffer_2(B(2)+10,2);
    eta_p_t(3,k) = buffer_3(B(3)+10,2);
    eta_p_t(4,k) = buffer_4(B(4)+10,2);


    % estimate of total loss ratio in bytes
    loss_b_f(1,k) = buffer_1(B(1)+11,1);
    loss_b_f(2,k) = buffer_2(B(2)+11,1);
```

```
    loss_b_f(3,k) = buffer_3(B(3)+11,1);
    loss_b_f(4,k) = buffer_4(B(4)+11,1);

    % estimate of total loss ratio in bytes
    loss_b_t(1,k) = buffer_1(B(1)+11,2);
    loss_b_t(2,k) = buffer_2(B(2)+11,2);
    loss_b_t(3,k) = buffer_3(B(3)+11,2);
    loss_b_t(4,k) = buffer_4(B(4)+11,2);

    % estimate of total loss ratio in packets
    loss_p_f(1,k) = buffer_1(B(1)+12,1);
    loss_p_f(2,k) = buffer_2(B(2)+12,1);
    loss_p_f(3,k) = buffer_3(B(3)+12,1);
    loss_p_f(4,k) = buffer_4(B(4)+12,1);

    % estimate of total loss ratio in packets
    loss_p_t(1,k) = buffer_1(B(1)+12,2);
    loss_p_t(2,k) = buffer_2(B(2)+12,2);
    loss_p_t(3,k) = buffer_3(B(3)+12,2);
    loss_p_t(4,k) = buffer_4(B(4)+12,2);

    % average accumulated delay
    delay_a_f(1,k) = buffer_1(B(1)+14,1);
    delay_a_f(2,k) = buffer_2(B(2)+14,1);
    delay_a_f(3,k) = buffer_3(B(3)+14,1);
    delay_a_f(4,k) = buffer_4(B(4)+14,1);

    % average accumulated delay
    delay_a_t(1,k) = buffer_1(B(1)+14,2);
    delay_a_t(2,k) = buffer_2(B(2)+14,2);
    delay_a_t(3,k) = buffer_3(B(3)+14,2);
    delay_a_t(4,k) = buffer_4(B(4)+14,2);

    % average queue size in packets

    queue_p(1,k) = buffer_1(B(1)+2,1);
    queue_p(2,k) = buffer_2(B(2)+2,1);
    queue_p(3,k) = buffer_3(B(3)+2,1);
    queue_p(4,k) = buffer_4(B(4)+2,1);

    % average queue size in bytes
    queue_b(1,k) = buffer_1(B(1)+2,2);
    queue_b(2,k) = buffer_2(B(2)+2,2);
    queue_b(3,k) = buffer_3(B(3)+2,2);
    queue_b(4,k) = buffer_4(B(4)+2,2);

end % end for loop for k

% Define performance data arrays
frames = (0:max_frame_index-1);


for k=1:max_frame_index
    sum_traffic= 0;
    for j=1:classes
        sum_traffic=sum_traffic+Th_b(j,k);
    end
```

```
    traffic(k) = sum_traffic;
    relative_share_1 = Th_b(1,:)/sum_traffic;
    relative_share_2 = Th_b(2,:)/sum_traffic;
    relative_share_3 = Th_b(3,:)/sum_traffic;
    relative_share_4 = Th_b(4,:)/sum_traffic;
end

subplot(531)
plot(frames,relative_share_1,'k-', frames, relative_share_2,'k:',
frames, relative_share_3, 'ko', frames, relative_share_4, 'k+');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('Relative Share ','Fontsize', 12)
axis([0 max_frame_index 0 1])

frame_share_1 = Th_b(1,:)/E;
frame_share_2 = Th_b(2,:)/E;
frame_share_3 = Th_b(3,:)/E;
frame_share_4 = Th_b(4,:)/E;

subplot(532)
plot(frames,frame_share_1,'k-', frames, frame_share_2,'k:', frames,
frame_share_3, 'ko', frames, frame_share_4, 'k+');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('Link Share ','Fontsize', 12)
axis([0 max_frame_index 0 1])

link_utilization = traffic/F;

%subplot(533)
%plot(frames,link_utilization,'k-');
plot(frames,link_utilization,'k-');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('Link Utilization', 'Fontsize', 12)
axis([0 max_frame_index 0 1])

subplot(534)
semilogy(frames,eta_b_f(1,:),'k-', frames, eta_b_f(2,:),'k:', frames,
eta_b_f(3,:), 'ko', frames, eta_b_f(4,:), 'k+');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('\eta_{bf}', 'Fontsize', 12)
axis([0 max_frame_index 10^-1 1])

subplot(535)
semilogy(frames,eta_b_t(1,:),'k-', frames, eta_b_t(2,:),'k:', frames,
eta_b_t(3,:), 'ko', frames, eta_b_t(4,:), 'k+');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('\eta_{bt}', 'Fontsize', 12)
axis([0 max_frame_index 10^-1 1])

subplot(536)
semilogy(frames,eta_p_f(1,:),'k-', frames, eta_p_f(2,:),'k:', frames,
eta_p_f(3,:), 'ko', frames, eta_p_f(4,:), 'k+');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('\eta_{pf}', 'Fontsize', 12)
axis([0 max_frame_index 10^-1 1])
```

```
subplot(537)
semilogy(frames,eta_p_t(1,:),'k-', frames, eta_p_t(2,:),'k:', frames,
eta_p_t(3,:), 'ko', frames, eta_p_t(4,:), 'k+');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('\eta_{pt}', 'Fontsize', 12)
axis([0 max_frame_index 10^-1 1])

subplot(538)
plot(frames,loss_b_f(1,:),'k-', frames, loss_b_f(2,:),'k:', frames,
loss_b_f(3,:), 'ko', frames, loss_b_f(4,:), 'k+');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('Loss ratio_{bf}', 'Fontsize', 12)
axis([0 max_frame_index 0 1])

subplot(539)
plot(frames,loss_b_t(1,:),'k-', frames, loss_b_t(2,:),'k:', frames,
loss_b_t(3,:), 'ko', frames, loss_b_t(4,:), 'k+');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('Loss ratio_{bt}', 'Fontsize', 12)
axis([0 max_frame_index 0 1])

subplot(5,3,10)
plot(frames,loss_p_f(1,:),'k-', frames, loss_p_f(2,:),'k:', frames,
loss_p_f(3,:), 'ko', frames, loss_p_f(4,:), 'k+');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('Loss ratio_{pf}', 'Fontsize', 12)
axis([0 max_frame_index 0 1])
pause

subplot(5,3,11)
plot(frames,loss_p_t(1,:),'k-', frames, loss_p_t(2,:),'k:', frames,
loss_p_t(3,:), 'ko', frames, loss_p_t(4,:), 'k+');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('Loss ratio_{pt}', 'Fontsize', 12)
axis([0 max_frame_index 0 1])
pause

%subplot(5,3,12)
%plot(frames,queue_b(1,:),'k-', frames, queue_b(2,:),'k:', frames,
queue_b(3,:), 'g', frames, queue_b(4,:), 'r');
plot(frames,queue_b(1,:),'k-', frames, queue_b(2,:),'k:', frames,
queue_b(3,:), 'ko', frames, queue_b(4,:), 'k+');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('Length_b ', 'Fontsize', 12)
pause

%subplot(5,3,13)
%plot(frames,queue_p(1,:),'k-', frames, queue_p(2,:),'k:', frames,
queue_p(3,:), 'g', frames, queue_p(4,:), 'r');
plot(frames,queue_p(1,:),'k-', frames, queue_p(2,:),'k:', frames,
queue_p(3,:), 'ko', frames, queue_p(4,:), 'k+');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('Length_p ', 'Fontsize', 12)
pause
```

```
subplot(5,3,14)
plot(frames,delay_a_f(1,:),'k-', frames, delay_a_f(2,:),'k:', frames,
delay_a_f(3,:), 'ko', frames, delay_a_f(4,:), 'k+');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('Delay_f', 'Fontsize', 12)

subplot(5,3,15)
plot(frames,delay_a_t(1,:),'k-', frames, delay_a_t(2,:),'k:', frames,
delay_a_t(3,:), 'ko', frames, delay_a_t(4,:), 'k+');
xlabel('Frame Index', 'Fontsize', 12)
ylabel('Delay_t', 'Fontsize', 12)
```

B.2 Input traffic characteristics. MATLAB file

```
function output_buffer=traffic_in(k,frame_time,duration,input_buffer,
status, a,b,length_var, length_mu)


% traffic_in(frame_time,duration,input_buffer,a,b,length_var,
length_mu)
% Update the input buffer to produce the
% output buffer after generating traffic during
% a certain frame period.
% Generate packets with
% random interarrival times and
% radnom lengths within a specified time duration.
% Random interarrival times are generated

% according to the inversion method.
% k is frame index
% frame_time: time of current frame
% duration: time duration of one frame
% input_buffer: structure for buffer at input
% status: size of buffer status word
% a: position parameter (in seconds)
% b: shape parameter (in seconds^-1)
% length_var: variance in length
% length_mu: average packet length


% get all the info you can about the input buffer
% since we have 10 extra fields at tail, we subtract 10
% to get actual storage space for packets.

buffer_size= size(input_buffer, 1)-status; %size in bytes

output_buffer= input_buffer; % initialize updated buffer

s= input_buffer(buffer_size+1,1); % start index
e= input_buffer(buffer_size+1,2); % end index

occupancy_p = output_buffer(buffer_size+2,1);
occupancy_b = output_buffer(buffer_size+2,2);

bytes_rcv_t = output_buffer(buffer_size+3,2);
packets_rcv_t = output_buffer(buffer_size+4,2);

bytes_lost_t = output_buffer(buffer_size+7,2);
packets_lost_t = output_buffer(buffer_size+8,2);

full_flag = output_buffer(buffer_size+status,2);

local_time=0; % Reset the counter indicating relative time
bytes_rcv_f=0; % initialize number of arriving bytes/frame
packets_rcv_f = 0;
bytes_lost_f=0; % initialize number of lost packets/frame
packets_lost_f = 0;
```

```
loss_b_f= 0; % loss ratio per frame
loss_p_f = 0;

while local_time < duration % check that packet arrived during current
frame
    % We want to make sure that the incoming packet has room in the
buffer
    % we are sure the packet will be saved, so we make empty flag zero
    % packet will be saved because we could drop the HOL packet
    % to accomodate incoming packet.
    % when empty_flag =0, buffer is not empty.

    empty_flag = 0;
    output_buffer(buffer_size+status,1) = empty_flag;

    x = rand(1); % source random number

    % packet size in integer bytes
    packet_length = floor(sqrt(length_var)*randn(1)+ length_mu);

    % evaluate absolute packet arrival time and local time within the
frame
    inter_arrival_time=a - (1/b)*log(1-x); % interarrival time of
packet

    % packet arrival time relative to start time of current frame
    packet_time = frame_time + local_time; % absolute packet arrival
time

    % check that buffer is not full. We do not care if buffer is empty
or not.
    if full_flag == 0 % buffer is not completely full
        % Thus there is an empty location in buffer as indicated with
the end index e.
        output_buffer(e,1) = packet_time; % insert arrival time of new
packet in free location
        output_buffer(e,2) = packet_length; % insert length of new
packet in free location
        output_buffer(e,3) = k; % indicate frame index
        e = e+1; % increment end index to indicate effect of adding a
packet
        if e == buffer_size+1 % index overflow
            e = 1;
        end % end if that checks index overflow
        output_buffer(buffer_size+1,2)  = e; % end index

        if e == s
            % buffer was not completely full but is now full since
location
            % of empty cell equals start of head of line packet
            full_flag = 1; % now we indicate buffer is full (which is
not same as overflowed)
            output_buffer(buffer_size+status,2) = full_flag;
        end % end if that compares s and e indices

        % update buffer data
        occupancy_p = occupancy_p+1;
```

```
        output_buffer(buffer_size+2,1)  = occupancy_p; % size of buffer
in packets

        occupancy_b = occupancy_b + packet_length;
        output_buffer(buffer_size+2,2)  = occupancy_b; % size of buffer
in bytes

        bytes_rcv_f = bytes_rcv_f+packet_length;
        bytes_rcv_t = bytes_rcv_t+packet_length;
        output_buffer(buffer_size+3,1)  = bytes_rcv_f; % input number
of bytes per frame
        output_buffer(buffer_size+3,2)  = bytes_rcv_t; % input size of
buffer in bytes per frame

        packets_rcv_f = packets_rcv_f+1;
        packets_rcv_t = packets_rcv_t+1;
        output_buffer(buffer_size+4,1)  = packets_rcv_f; % input number
of packets per frame
        output_buffer(buffer_size+4,2)  = packets_rcv_t; % input size
of buffer in packets per frame

        %output_buffer
        %bytes_buffer = sum(output_buffer(1:buffer_size,2))
  else % i.e. full_flag ==1
        % buffer is full we must delete a packet to insert the new one.
        % a full buffer is characterized by
        %                 e==s
        % Now drop head of line packet to make room for new packet
        % 1. We correct the buffer size in bytes since HOL packet is
dropped
        % and the arriving packet is inserted at end of line EOL.
        % 2. We write new packet at location of starting packet (index
s).
        % This effectively drops the head of the FIFO
        % 3. We increment both s and e to move to next HOL packet which
        % is now the oldest packet in FIFO.

        % First we gather statistics of packet to be dropped.
        bytes_lost_f = bytes_lost_f+output_buffer(s,2); % increase
number of lost bytes/frame
        output_buffer(buffer_size+7,1) = bytes_lost_f ; % number of
transmitted packets/frame

        bytes_lost_t = bytes_lost_t+output_buffer(s,2); % increase
total number of lost bytes
        output_buffer(buffer_size+7,2) = bytes_lost_t ; % number of
lost packets/frame

        if bytes_lost_t > bytes_rcv_t
            disp('error: lost bytes > received bytes');
        end

        packets_lost_f = packets_lost_f+1; % increase number of lost
packets/frame
        output_buffer(buffer_size+8,1) = packets_lost_f ; % number of
transmitted packets/frame
```

```
        packets_lost_t = packets_lost_t+1; % increase total number of
lost packets
        output_buffer(buffer_size+8,2) = packets_lost_t ; % number of
lost packets/frame

        if packets_lost_t > packets_rcv_t
            disp('error: lost packets > received packets');
        end

        % Update the number of bytes in the buffer
        % by deleting the HOL packet and replacing it with the new
packet
        occupancy_b = occupancy_b - output_buffer(s,2)+ packet_length;
%delete HOL packet
        output_buffer(buffer_size+2,2)  = occupancy_b; % size of buffer
in bytes

        occupancy_p = occupancy_p; %delete one packet and add one
packet
        output_buffer(buffer_size+2,1)  = occupancy_p; % size of buffer
in bytes

        % update buffer size
        bytes_rcv_f = bytes_rcv_f+packet_length;
        output_buffer(buffer_size+3,1)  = bytes_rcv_f; % input number
of packets per frame

        bytes_rcv_t = bytes_rcv_t+packet_length;
        output_buffer(buffer_size+3,2)  = bytes_rcv_t; % input size of
buffer in bytes per frame

        packets_rcv_f = packets_rcv_f+1;
        output_buffer(buffer_size+4,1)  = packets_rcv_f; % input number
of packets per frame

        packets_rcv_t = packets_rcv_t+1;
        output_buffer(buffer_size+4,2)  = packets_rcv_t; % input size
of buffer in bytes per frame

        % Now insert the new packet at location s
        output_buffer(s,1) = packet_time; % replace HOL packet arrival
time
        output_buffer(s,2) = packet_length; % replace HOL packet packet
length
        output_buffer(s,3) = k; % indicate frame index

        s = s + 1; % now move HOL packet one spot into the FIFO
        if s == buffer_size+1
            s = 1;
        end
        e = s; % buffer is still full as indicated  by e==s.
        output_buffer(buffer_size+1,1)  = s; % start index
        output_buffer(buffer_size+1,2)  = e; % end index

        %output_buffer
        %bytes_buffer = sum(output_buffer(1:buffer_size,2))
```

```
        %end % end if that checks for buffer full
local_time=local_time+inter_arrival_time; % update local time

end %End of while


% Now that we are finished with our frame, we update our loss ratio
if bytes_rcv_f == 0
    loss_b_f = 0;
else
    loss_b_f = bytes_lost_f/bytes_rcv_f;
end
output_buffer(buffer_size+11,1)  = loss_b_f;


if bytes_rcv_t == 0
    loss_b_t = 0;
else
    loss_b_t = bytes_lost_t/bytes_rcv_t;
end
output_buffer(buffer_size+11,2)  = loss_b_t;

if packets_rcv_f == 0
    loss_p_f = 0;
else
    loss_p_f = packets_lost_f/packets_rcv_f;
end
output_buffer(buffer_size+12,1)  = loss_p_f;


if packets_rcv_t == 0
    loss_p_t = 0;
else
    loss_p_t = packets_lost_t/packets_rcv_t;
end
output_buffer(buffer_size+12,2)  = loss_p_t;
```

B.3 Output traffic characteristics. MATLAB file

```
function [output_buffer, exit_time] =
traffic_out(C,x,entry_time,input_buffer, status)

%TRAFFIC_OUT transmit buffer packets
%
% C: output line rate (bytes/s)
% x : maximum amount of bytes to send
% time: time at which transmission is taking place
% input_buffer: buffer to pick pakcets from
% stauts: size of buffer status word
% traffic_out updates the time value to account for the delay
% required to transmit a packet.

% get all the info you can about the input buffer
buffer_size = size(input_buffer, 1)-status;

output_buffer = input_buffer;

s = input_buffer(buffer_size+1,1); % start index
e = input_buffer(buffer_size+1,2); % end index

time = entry_time; % initialize starting time

occupancy_p = output_buffer(buffer_size+2,1);
occupancy_b = output_buffer(buffer_size+2,2);

bytes_rcv_f = output_buffer(buffer_size+3,1);
bytes_rcv_t = output_buffer(buffer_size+3,2);

packets_rcv_f = output_buffer(buffer_size+4,1);
packets_rcv_t = output_buffer(buffer_size+4,2);

bytes_xmt_f=0; % initialize number of transmitted bytes/frame
bytes_xmt_t = output_buffer(buffer_size+5,2);

packets_xmt_f=0; % initialize number of transmitted bytes/frame
packets_xmt_t = output_buffer(buffer_size+6,2);

eta_b_f= 0; % loss ratio per frame
eta_p_f = 0;

delay_f= 0; % initialize total delay in one frame
delay_t = output_buffer(buffer_size+13,2);

delay_a_f = 0; % initialize average delay per frame

empty_flag = output_buffer(buffer_size+status,1);
full_flag = output_buffer(buffer_size+status,2);

while x > 0 % our buffer has some credit left to transmit
    if empty_flag == 1 % buffer is empty, leave the transmit mode
        x = 0; % set x=0 to exit while loop and stop transmission
    else % buffer is not empty and we can potentially send a packet
```

```
        packet_size = input_buffer(s,2); % pick HOL packet and check
its size

        if x > packet_size % we can send the HOL packet and houskeep
buffer status word

            if full_flag == 1 % buffer could be full at start
                full_flag = 0; % buffer is not full since we are able
to send one packet
                output_buffer(buffer_size+status,2)=full_flag;
            end

            % now do some statistics gathering since we know a packet
will be sent

            % extract the packet at HOL location s for transmission
            packet_time = input_buffer(s,1);

            % Clear the location of the packet so we can see the action
when we list buffe_i
            output_buffer(s,1) = 0; output_buffer(s,2) = 0;
output_buffer(s,3)=0;

            % decrease the size of the buffer by one packet
            occupancy_p = occupancy_p - 1;
            output_buffer(buffer_size+2,1)= occupancy_p;

            % decrease the size of the buffer by size of packet that
left
            occupancy_b = occupancy_b - packet_size;
            output_buffer(buffer_size+2,2)= occupancy_b;

            % indicate that one more packet left in the current frame
            bytes_xmt_f = bytes_xmt_f + packet_size;
            output_buffer(buffer_size+5,1)= bytes_xmt_f;

            bytes_xmt_t = bytes_xmt_t + packet_size;
            output_buffer(buffer_size+5,2)= bytes_xmt_t;


            packets_xmt_f = packets_xmt_f + 1;
            output_buffer(buffer_size+6,1)= packets_xmt_f;

            packets_xmt_t = packets_xmt_t + 1;
            output_buffer(buffer_size+6,2)= packets_xmt_t;

            % update the bytes efficiency
            if bytes_rcv_f == 0
                eta_b_f = 0;
            else
                eta_b_f = bytes_xmt_f/bytes_rcv_f;
            end
            output_buffer(buffer_size+9,1) = eta_b_f;

            if bytes_rcv_t == 0
                eta_b_t = 0;
            else
```

```
            eta_b_t = bytes_xmt_t/bytes_rcv_t;
        end
        output_buffer(buffer_size+9,2) = eta_b_t;
        %update the packets efficiency
        if packets_rcv_f == 0
            eta_p_f = 0;
        else
            eta_p_f = packets_xmt_f/packets_rcv_f;
        end
        output_buffer(buffer_size+10,1) = eta_p_f;

        if packets_rcv_t == 0
            eta_p_t = 0;
        else
            eta_p_t = packets_xmt_t/packets_rcv_t;
        end

        output_buffer(buffer_size+10,2) = eta_p_t;

        % calculate the total delay within that frame
        packet_delay = abs(time - packet_time);

        % update local time since our packet needs time to be
transmitted
        time = time + packet_size/C;

        delay_f = delay_f+packet_delay;
        output_buffer(buffer_size+13,1) = delay_f;

        delay_t = delay_t+packet_delay;
        output_buffer(buffer_size+13,2) = delay_t;


        % update average delay

        if packets_xmt_f == 0
            delay_a_f = 0;
        else
            delay_a_f = delay_f/packets_xmt_f;
        end
        output_buffer(buffer_size+14,1) = delay_a_f;

        if packets_xmt_t == 0
            delay_a_t = 0;
        else
            delay_a_t = delay_t/packets_xmt_t;
        end
        output_buffer(buffer_size+14,2) = delay_a_t;

        % now we move to the next packet in the buffer

        s = s+1; % increment start index to move to next packet at
the next iteration

        if s > buffer_size % index overflow, which does not mean
buffer overflow!
            s = 1;
```

```
            end % end if that checks index overflow
            output_buffer(buffer_size+1,1)= s;

            if s == e % buffer has just been emptied since s=e
                empty_flag = 1;
                output_buffer(buffer_size+status,1)=empty_flag;
            end

            % update allotted quota
            x = x - packet_size;

        else % packet is bigger than allotted quota x < packet_size
            x = 0; % exit the wile loop by making zero credit

        end % end if that checks that allotted data is more than packet
size

    end % end if that checks that buffer is empty or not

    %output_buffer
    %bytes_buffer = sum(output_buffer(1:buffer_size,2))


end % end while for allocated output data

exit_time = time;
```