Performance analysis of Virtual Routing and Queuing (VRQ) Switch

by

Deepali Arora

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

M.A.Sc

in the Electrical and Computer Engineering

We accept this thesis as conforming
to the required standard

[redacted]

Dr. F. Gebali, Supervisor, Dept. of Elect. & Comp. Eng.

[redacted]

Dr. P. Agathoklis, Member, Dept. of Elect. & Comp. Eng.

[redacted]

Dr. A. Shoja, Outside Member, Dept. of Computer Science

[redacted]

Dr. Z. Dong, External Examiner

Supervisor: Dr. Fayez Gebali

ABSTRACT

A new generation asynchronous transfer mode (ATM) switch based on the principle of virtual routing and queuing (VRQ) is presented and its performance is compared with input, output, and shared buffered switches. The VRQ switch is improved since 1) it provides dedicated input queues for incoming packets destined for different output ports, and 2) the output buffers store pointers to the packets, rather than the packets themselves.

Simulations are peformed using traffic data generated using both Poisson and Pareto distributions. These data are used to drive the switches and to analyze their peformance. The simulations are performed using C programs. The results show that while principally being an input buffered switch the performance of the VRQ switch is comparable to that of an output buffered switch. Also since VRQ is essentially an input buffered switch it is relatively easy to implement in hardware compared to the output and shared buffered switches. An improved performace and easy implementation in hardware make VRQ an ideal switch.

In regard to the effect of traffic characteristics on a switch's performance, the simulations show that switches perform better, both in terms of cell delay and cell loss, when they are driven with Poisson distributed traffic. This is expected since the Pareto distributed traffic shows more bursty behavior and exhibits larger on-periods than the Poisson traffic.

The design of the multi-ported input buffer of the VRQ switch is obtained using VHDL. This multi-ported buffer is capable of performing multiple read operations per clock cycle and can support high data rates without any speed-up requirements.

Examiners:

[redacted]

Dr. F. Gebali, Supervisor, Dept. of Elect. & Comp. Eng.

[redacted]

Dr. P. Agathoklis, Member, Dept. of Elect. & Comp. Eng.

[redacted]

Dr. A. Shoja, Outside Member, Dept. of Computer Science

[redacted]

Dr. Z. Dong, External Examiner

Table of Contents

List of Figures

List of Tables

Notation

AAL	ATM Adaptation Layer
ACK	Acknowledgement
ATM	Asynchronous Transfer Mode
BISDN	Broadband Integrated Services Digital Network
CBR	Constant Bit Rate
CLP	Cell Loss Priority
DU	Data Unit
FBM	Fractional Brownian Moment
FIFO	First In First Out
FTP	File Transfer Protocol
GFC	Generic Flow Control
HEC	Header Error Check
HOL	Head Of Line
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISO	International Organisation for Standardisation
LAN	Local Area Network
LIFO	Last In First Out
NNI	Network Network interface
OAM	Operation Administration and Management
OC	Optical carrier
OSI	Open Systems Interconnection
PCR	Peak Cell Rate
PCM	Pulse Code Modulation
PDU	Protocol Data Unit
PM	Physical Medium
PT	Payload Type

PTI	Payload Type Identifiers
QOS	Quality Of Service
SAR	Segmentation And Re-assembly
SDH	Synchronous Digital Hierarchy
SCR	Sustainable cell rate
SONET	Synchronous Optical Network
STM	Synchronous Transmission Mode
TC	Transmission Control Protocol
UBR	Unspecified Bit Rate
UNI	User Network Interface
VBR	Variable Bit Rate
VCC	Virtual Channel Connection
VCI	Virtual Channel Identifier
VHDL	Very High speed Integrated circuit Hardware Design Language
VIQ	Virtual Input Queuing
VOQ	Virtual Output Queuing
VPC	Virtual Path Connection
VPI	Virtual Path Identifier
VRQ	Virtual Routing and Queuing
WAN	Wide Area Network
WWW	World Wide Web

Acknowledgement

Writing this acknowledgement was a very emotional process for me because I had to look back on two years which have been full of work but which have also left enough time for some fun. All the small and big things which enabled me to write this thesis would have been much more difficult and sometimes impossible without the help of my supervisor, course instructors, colleagues and friends. Here I want to thank them for their contributions, discussions and useful professional as well as private pieces of advice and hints.

First of all, I want to thank my supervisor Dr. Fayez Gebali for his help and support during the course of this work. I am grateful for his continuous assistance and was impressed with his great ability to recognize situations when to direct me and when to let me go my own way. I would also like to thank Dr. Kin Li and Dr. Lu. who helped and inspired me throughout my course work.

Furthermore, I would like to thank all members of the digital signal processing (DSP) lab for contributing to such an inspiring and pleasant atmosphere. I would also like to thank Eric Laxadal, Mohamed Watheq El-Kharashi and Debasish Sasmal for their considerable support and patience.

Lastly, I would also like to thank my parents and my husband Dr. Vivek Arora, whose financial and moral support made this thesis possible.

Dedication

Dedicated to my Parents and my Husband

Chapter 1

Introduction

1.1 Motivation

The Internet is growing exponentially and is impacting every aspect of modern life. The amount of information exchanged through email sent over the Internet is greater than the amount transmitted over any other traditional medium of communication. The Internet has grown out of its academic roots and now plays a major role in our everyday lives. The Internet has led to an economic boom that has resulted in the record long economic expansion in developed nations. It has enabled business to attain increased productivity, reduced product development time, and extended their reach to a global customer base. Several developing nations are aggressively building their Internet infrastructure to bridge their economic gap from developed nations. But even with the tremendous growth of the Internet and the vast amount of research focused on it, there is more to be accomplished. Though one can compare the Internet to other modern inventions such as television and telephone, it has certain unique characteristics. Its most important characteristic is that it is technology that has been and is likely to evolve continuously, to increasingly support richer set of features and functionality.

With the rapid growth of the Internet, the role of computer networks is becoming extremely important. Currently the drive in the computer industry is for computer hardware and software to achieve higher performance with greater reliability. Computer networks primarily consist of two main components; media to transfer information and network elements, both of which have to be improved to meet the increased bandwidth demands. Though the advances in the fibre technology have made avail-

able huge amounts of transmission bandwidth, the network elements still continue to impose limitations in the attempt to increase the bandwidth of computer networks.

One of the most promising technology for high speed data is asynchronous transfer mode (ATM). ATM is a switching technology used to transfer fixed size cells and combines advantages of both circuit and packet switching technologies. Within ATM networks, ATM switches have been identified to be the primary bottleneck. At present, several architectural design concepts that are used in ATM switches exist[1-7]. The tremendous increase in Internet traffic require that the next generation switches be capable of supporting data rates as high as tera-bits per second. The new generation of switch architectures are expected to satisfy the quality of service (QoS) requirements of the future Internet users, while taking into account both reliability and modularity. This is a major challenge for switch designers.

This thesis attempts to gain insight into the performance of the next generation ATM switch and also focuses on how improvements in buffering stratergy can be used to achieve high performance levels while keeping the switch architecture relatively simple. To this end this thesis compares the performance of the Virtual Routing and Queuing switch with the other basic switches based on three different standard buffering strategies.

1.2 Thesis contributions

To achieve the aim of comparing the performance of the VRQ switch with three other switches based on different buffering strategies, and to investigate the effect of traffic flow characteristics on a switch's performance, the following main tasks are performed.

- Traffic data is generated using Pareto distribution.
- The VRQ, and the input, output and shared buffered switches are simulated using C-programs to evaluate their performances.
- Inter-comparisons are performed to assess the improvement in the performance of VRQ over other switches.

- Multi-ported input buffer of the VRQ switch is designed using hardware design language VHDL.

1.3 Thesis overview

This thesis is organised as follows. Chapter 2 provides a brief introduction to switches and their types. Amongst the various types of switches focus is laid on asynchronous transfer mode (ATM) switches, its classifications and its characteristics. Finally the different buffering strategies used in the ATM switches are also discussed in brief.

Chapter 3 provides an introduction to virtual routing and queuing (VRQ) switch. The switch architecture and its main features are discussed in detail. The modification made to the input buffer of the VRQ switch, in order to support multicast traffic are also presented in this chapter. The new multi-ported input buffer is designed using VHDL.

To assess the performance of the switch via simulation traffic data are required. A brief introduction to traffic generation and its impact on the performance of the switch are discussed in Chapter 4. The methodology adopted to generate traffic used for the simulation of the VRQ switch and other switches is also illustrated.

The performance of the VRQ switch is analysed in terms of cell delay, cell loss, and throughput in Chapter 5. The comparison of the VRQ with input, output and shared buffered switches in terms of its performance is also presented.

Chapter 6 summarizes the work done in thesis and suggestions are made for future work.

Chapter 2

Introduction to Asynchronous transfer mode (ATM) switches

2.1 Introduction

Switches are used to connect several local area networks (LANs) and to facilitate transmission of data between them. Data is transmitted from source to destination by set of intermediate switching nodes. These switching nodes are not concerned with the content of data but their function is to provide the routing facility [8-10]. Switches work in Layer two of the open systems interconnection (OSI) model but the new trend is to implement them at Layer three of the OSI model [11]. Switches serve as a temporary connection between input port and an output port, and make the routing decision for the incoming packet. The two main architectural components of switches are, a controller and a datapath [11], as shown in [11] Fig. 2.1.

The datapath deals with individual packets and is composed of the input port, the output port, and the switching fabric as shown in Fig. 2.1. The datapath performs the following functions.

- It accepts the incoming packets and stores them in the local buffers for processing the header information,
- Establishes a path from the input port to the appropriate outport port via switching fabric, and
- Routes the packet to the output port and stores it till it is routed out.

The controller deals with the packet streams and is composed of the routing table

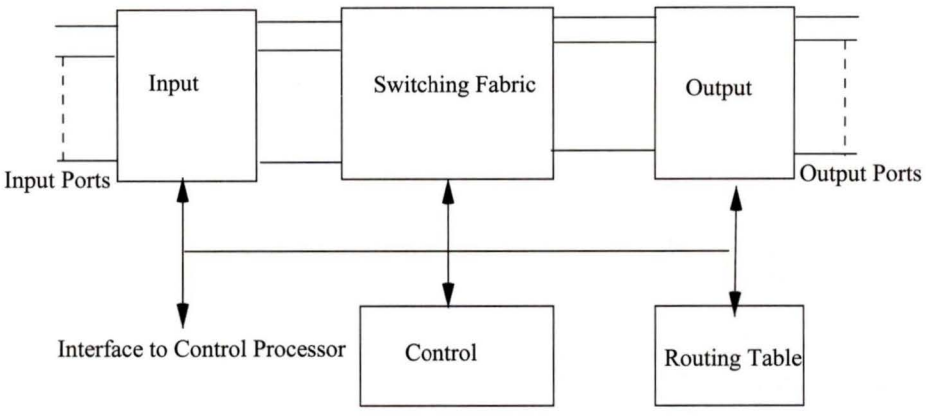

Figure 2.1. *Basic components of a switch*

and the control section (see Fig. 2.1). The control section performs the following functions.

- It maintains the contents of the routing table, which determines the destination output port for the incoming cell,
- Provides congestion control for the switch by continously monitoring the switch resources and taking proper actions, and
- Assigns switch resources to the established connections based on the class of service.

During 70's, the Wide Area Networks (WANs) were implemented using two main switching technologies, circuit switching and packet switching. These switching technologies performed adequately for constant bit rate traffic (CBR) but unfortunately not for variable bit rate traffic (VBR). The need to develop a single network that can support all types of information led to broadband integrated services digital network (B-ISDN). The term transfer refers to both transmission and switching aspects, so a transfer mode is a specific way of transmitting and switching information in a network. In ATM, all information to be transferred is packed into fixed size cells, which are 53 bytes long. The first 5 bytes form the header and the remaining 48 bytes form the information field. The term asynchronous, in the context of multiplexed connections means that the cells allocated to the medium may exhibit an irregular recurrence pattern as they are inserted into the medium according to the actual bandwidth of

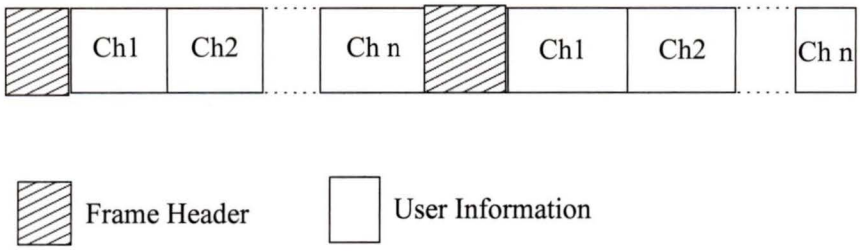

Figure 2.2. *Synchronous Transfer Mode*

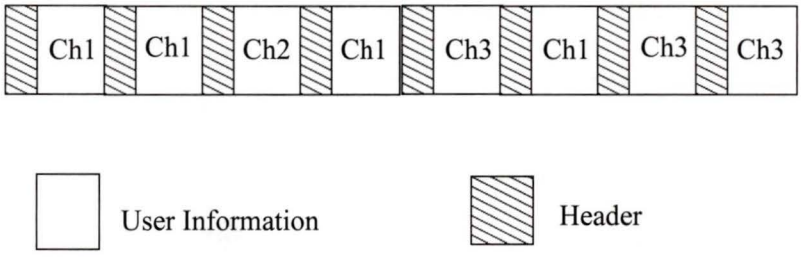

Figure 2.3. *Asynchronous Transfer Mode*

remaining 48 bytes form the information field. The term asynchronous, in the context of multiplexed connections means that the cells allocated to the medium may exhibit an irregular recurrence pattern as they are inserted into the medium according to the actual bandwidth of the connection.

Examples of a synchronous transmission mode (STM) and an ATM transmission pattern are shown in Fig. 2.2 and Fig. 2.3, respectively. In STM, a data unit associated with a given channel is identified by its position in the transmission frame, while in ATM a cell associated with a specific virtual channel may occur at any position. This property provides ATM flexibility in allocation of bit-rate (i.e., bandwidth) to a specific connection or channel. Bit-rates in STM are restricted to a set of predefined channel bit-rates.

Networks are commonly categorized as circuit switched or packet switched networks. In circuit switched networks a connection is established between the source and the destination before any information is transferred. In packet switched networks every packet contains the address of the destination and is routed independently. Cir-

transfer delay of the packets is also not constant. However, packet switched networks can utilize the bandwidth of the network more efficiently, and are much more flexible in terms of the bit-rate assigned to individual connections. ATM combines the advantages of both switching techniques by using the concept of virtual channels, which are connection-oriented channels. A connection-oriented channel is like a telephone call, because it involves a connection setup phase, an information transfer phase and a connection tear down phase. A connection within the ATM layer consists of one or more links, each of which is assigned an identifier. These identifiers remain unchanged for the duration of the connection. Also, cell routing is done in hardware to achieve low transfer delays. ATM specifications guarantee cell sequence integrity under fault free conditions. Maintaining cell sequence integrity means that nowhere in the network can a cell belonging to a specific virtual channel connection overtake another cell belonging to the same virtual channel connection that was sent earlier.

2.2 ATM Reference Model

The 3-dimensional ATM reference model shown in Fig. 2.4 [8] consists of layers and planes. The reference model consists of physical layer, an ATM layer, an ATM Adaptation Layer (AAL) and the higher layers. The planes include the user, control, and management plane.

The functions of these layers are listed below.

- The physical layer provides for transmission of ATM cells over a physical medium that connects two ATM devices. It consists of two sublayers: The physical medium sublayer (PM) and transmission convergence sublayer (TC). The physical medium sub-layer is responsible for the correct transmission and reception of bits on the appropriate physical medium. This sub-layer must guarantee a proper bit timing reconstruction at the receiver. The transmission convergence (TC) sub-layer converts the ATM cell stream into bits to be transported over the physical medium. Other functions performed by TC sublayer are generation of the header error check (HEC) for each cell at the transmitter and its validation at the receiver. During startup, to detect the proper cell boundary, an

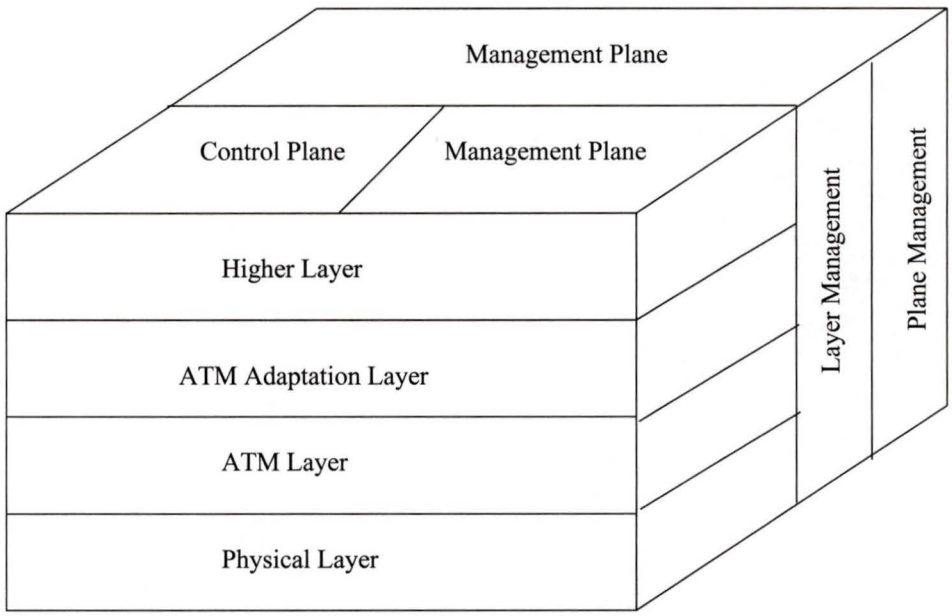

Figure 2.4. *3-Dimensional ATM reference model*

ATM network element needs to validate a number of HEC values for each cell. If it can successfully perform this check on a specific number of cells it means that the correct cell boundary has been detected. Once cell delineation has been found, an adaptive mechanism uses the HEC for correction and detection of cell header errors.

- The ATM layer is common to all services and provides packet transfer capabilities. The ATM layer is independent of the physical medium used to transport the ATM cells. The ATM layer is responsible for performing the following functions.

  1. The multiplexing and de-multiplexing of cells of different connections identified by different virtual channel identifiers (VCI) and virtual path identifier (VPI) values, into a single cell stream on a physical layer.

  2. Translating the cell identifier, which is required in most cases when switching a cell from one physical link to another in an ATM switch. This translation can be performed on the VCI, the VPI, or both.

3. Providing the user of a virtual channel connection (VCC) or virtual path connection (VPC) with one quality of service (QoS) class, out of a number of classes supported by the network. Some services may require a certain QoS for one part of the connection, and a lower QoS for the remainder. The distinction within the connection is made by means of the cell loss priority (CLP) bit in the cell header.

4. Performing management functions, such as setting the payload type identifier (PTI) field in the cell header to indicate congestion. When the PTI does not indicate user information, more information related to layer management is present in the information field of the cell.

5. Extracting of the cell header before the cell is delivered to the adaptation layer and adding the cell header after the cell is delivered from the adaptation layer to be transmitted.

6. Implementing a flow control mechanism on the user network interface using the generic flow control bits in the cell header.

- The AAL layer is service dependent. The ATM adaptation layer enhances the service provided by the ATM layer to a level required by the next higher layer. It performs functions for the user, control, and management planes, and supports the mapping between the ATM layer and the next higher layer. The functions performed by the AAL layer depend on the higher layer requirements. The AAL layer is divided into two sub-layers: the segmentation and re-assembly (SAR) sub-layer and the convergence sub-layer (CS). The main function of the SAR sub-layer is to segment the higher layer information into suitable size payloads for the consecutive ATM cells of a virtual connection on the transmit side. On the receiver side reassembly of the contents of the cells is performed. The convergence sub-layer performs functions like message identification, time/clock recovery, etc. If an application finds the service provided by the ATM layer

sufficient, then the AAL layer may be empty. There are four classes of AAL layers. To obtain these four classes the services are classified according to three basic parameters.

1. Time relation between source and destination: Some services have a time relation between the source and destination, while for others there is no such time relation. For example, for 64 Kbps pulse code modulation (PCM) voice there is a clear time relation between the source and destination. Information transfer between computers has no time relation. Services with time relation are also called real time services.
2. Bit-rate: Some services have a constant bit-rate, while others have a variable bit-rate.
3. Connection mode: Services can be either connectionless or connection oriented. Connectionless mode is similar to packet switching, whereas connection oriented mode is similar to circuit switching. In connection oriented mode, all the information is sent through the same path decided upon during connection setup. In connectionless mode the information may be traversed through different paths.

The functions of the plane include the following.

- User plane is used for user information transfer along with associated controls such as flow and error control.
- Control plane performs call control and connection control functions.
- Management plane consists of plane management and layer management. Plane management performs management functions related to system as a whole and provides coordination between these planes. Layer management performs management functions related to resources and parameters residing in its protocol entities.

2.3 Basic principles of ATM

This section briefly describes the basic principles put forward by the international telecommunication union.

- Information Transfer: ATM is considered a packet oriented transfer mode based on asynchronous time division multiplexing and the use of fixed length cells. Each cell consists of an information field and a header. The header is primarily used to identify cells belonging to the same virtual channel within the asynchronous time division multiplexing and to perform the appropriate routing. Cell sequence integrity is preserved on each virtual channel. The information field of ATM cells is carried transparently through the network. No processing, such as error control, is performed on the information field in the network. To accommodate various services such as voice, video, and data, several types of ATM adaptation layers (AAL) have been defined.

- Routing: ATM is connection oriented. The header values are assigned to each section of a connection for the complete duration of the connection, and translated when switched from one section to another. Signaling information is carried on a separate virtual channel. Two sorts of connections are possible, virtual channel connections (VCC) and virtual path connections (VPC). A VPC can be considered to be an aggregate of VCCs. Switching must first be done based on the VPC and then on the VCC.

- Resources: Since ATM is connection oriented, connections are established either semi-permanently or for the duration of the call in the case of switched services. This establishment includes not only allocation of a virtual channel identifier (VCI) and/or virtual path identifier (VPI), but also the allocation of the required resources for the user access and inside the network. These resources are expressed in terms of throughput (bit-rate) and quality of service (QoS). They may be negotiated between the user and the network for switched connections, during the call setup phase or possibly during the call.

- ATM Cell Identifiers: ATM cell identifiers, which include virtual path identifiers (VPI), virtual channel identifiers (VCI) and payload type identifiers (PTI), support recognition of an ATM cell on a physical transmission medium. Recognition of the cell is the basis for all further operations. VPI and VCI values are unique for cells belonging to the same virtual connection on a shared transmission medium. Within a particular virtual circuit, cells may be further distinguished by their PTI, which cannot be allocated freely, but depends on the type of payload carried by the cell. This field indicates whether the cell is carrying user information to be delivered transparently through the network or the network information. A number of pre-assigned ATM cell identifiers have been chosen in the ATM layer for particular cell streams. They are necessary for enabling communication with the network and performing network management. Other pre-assigned values define meta-signaling cells, point-to-point signaling cells and general-broadcast cells.

- Throughput: Bandwidth has to be reserved in the network for each virtual connection. ATM offers the possibility to realize resource savings in the total bandwidth needed when multiplexing traffic for many variable bit-rate connections. The amount that can be saved, however, depends heavily on the number of multiplexed connections, on the burstiness of the traffic they carry, and on the quality of service (QoS) required. Peak cell rate (PCR) and sustainable cell rate (SCR) are used as throughput parameters. PCR is the maximum cell rate of the connection for the duration of the call. SCR is the maximum, mean cell rate measured over a period shorter than the duration of the call but longer than the cell inter-arrival interval.

- Signaling: The negotiation between the user and the network with respect to the resources (e.g. VPI/VCI, throughput, QoS) is performed over a separate signaling virtual channel.

- Flow Control: There are three ATM layer service classes with respect to the bandwidth usage, constant bit-rate (CBR), variable bit-rate (VBR) and unspec-

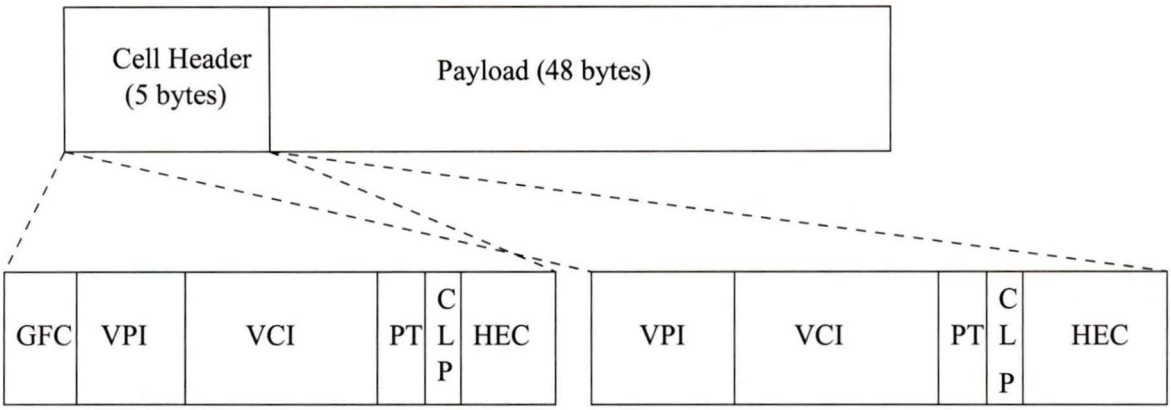

Figure 2.5. *An ATM cell header (left,UNI and right,NNI)*

ified bit-rate (UBR). An application requests for CBR if its bandwidth needs are constant. A VBR application requires a variable amount of bandwidth. The bandwidth available in the network at any given time may not be completely allocated. UBR applications make use of this unallocated bandwidth, and can handle varying throughput and the possibility of increased delay. To effectively manage the network bandwidth resource a fast signaling and control flow mechanism is used. There are two types of flow control mechanisms used in ATMs; generic flow control and resource management cells.

2.4 ATM cells

ATM cells, which are of fixed size, consist of 53 bytes. The first five bytes constitute the header followed by 48 bytes of information field. The header consists of six fields at User Network Interface (UNI), and at Network Network Interface(NNI) it consists of five fields as shown in Fig. 2.5 [8].

The Generic Flow Control (GFC) field is used for the control of cell flow at UNI. It is used to alleviate overload conditions in the network. The virtual path identifier (VPI) constitutes a routing field for the network and is of length 8 bits at UNI and 12 bits at NNI. The virtual channel identifier (VCI) is used for routing to and from

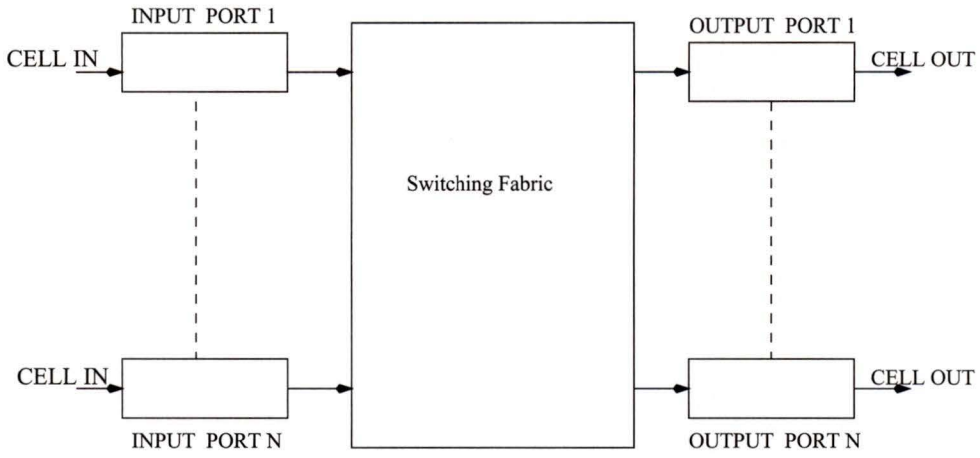

Figure 2.6. *ATM Switch*

the end user. The payload type (PT) field indicates the type of information in the information field i.e., user or management. The cell loss priority provides the guidance to the network in the event of congestion. A value of 0 indicates cell of higher priority and the value of 1 indicates lower priority cells. The header error control (HEC) field is used for error detection and correction in the header.

2.5 ATM switch architecture

An ATM switch of size N can be regarded as a box with N input ports and N output ports as shown in Fig. 2.6 [11]. During each time slot cells arrive at the input ports. The incoming cells are stored in the input buffers and depending upon the information in the cell header, each input port makes a routing decision. The input port controller is also responible for checking the errors using the header error control field. During the next time slot, the cell is routed to the output port. Cells at the output port is then delievered to their destinations. The cells from input port to output port are routed via switching fabric.

ATM switching fabrics can be broadly categorized as space division and time division switching fabrics [1]. In time division switching fabric, the cells flow through a single resource (bus or ring) shared by all input and output ports. The capacity of

the entire switching fabric is limited by the bandwidth of this shared resource. In case of space division switch, multiple concurrent paths can be established between input and output ports, so that multiple cells can be simultaneoulsy transmitted across the switching fabric.

Most of the switches used today are space division ATM switches. The space division switches can be classified into following categories.

1. Single and multistage switches [12]: In single stage switches, input and output ports are interconnected through one stage and a cell is switched in a single phase. In multistage switches, switching is performed by switching elements in consecutive stages.

2. Single and multipath switches [13]: For single path switch only one path exists for any input-output pair, while in the multipath switches there are more than one path for any input-output pair.

3. Blocking and non-blocking switches [14,15]: Blocking switch is one in which cells cannot reach their destined output ports because of internal blocking, while a non-blocking switch does not suffer from internal blocking.

4. Unicast and multicast switches [16]: Unicast switch is one in which a single user participates in connection, while in multicast switches more than two users participate in the connection.

5. Input, output, and shared buffered switches [1,17]: The location of buffers where the cells are stored determines whether an ATM switch is input or output buffered. In shared buffered switches the cells are stored in a common buffer which is shared by both input and output ports.

6. Crossbar, disjoint path, and banyan based switches [18,19]: Based on the internal structure, an ATM switch can be classified as a crossbar switch, a disjoint path based switch or a banyan based switch. A crossbar based (N×N) switch

consists of square array of N^2 individually operated crosspoints, one for each input-output pair. In disjoint based switches, the switching fabric has the capability of establishing all possible N^2 paths. Banyan based switches are those which have self routing capability and have same latency for all input-output pairs.

2.6 Classification of ATM switches based on different buffering stratergies

Buffers are used to store the incoming cells, which cannot be routed to their destined output ports at a given time. When multiple cells are destined to the same output port a temporary storage of cells is desired to prevent cell loss. The incoming cells can be stored at the input port, output port or in the central space which is shared by both input and output ports. Accordingly the ATM switches can be classified as input buffered, output buffered or shared buffered switches. Other buffering stratergies based on combination of these three stratergies have also been used. In this section different buffering stratergies are discussed.

2.6.1 Input buffered ATM switches

In an input buffered switch, as shown in Fig. 2.7 [11], the arriving cells enter in the first in first out (FIFO) buffer, placed at the port of entry. Each buffer can be considered as a module in which cells are queued before being delievered to their destination. The incoming cell is placed at the tail of the queue and is moved only when the cell at the head of the queue is routed through the switching fabric to an output port. If all head of line (HOL) cells are destined to distinct output ports then all of them are admitted and switched to their desired output lines. However, if k HOL cells ($1 < k \leq N$, where N is the number of output ports) are destined to the same output port, then only one cell is chosen according to some selection criteria. Some examples of selection criterion are random selection [20], longest queue selection [21], round robin selection [22], oldest queue selection [21], HOL LIFO(Last In First Out)

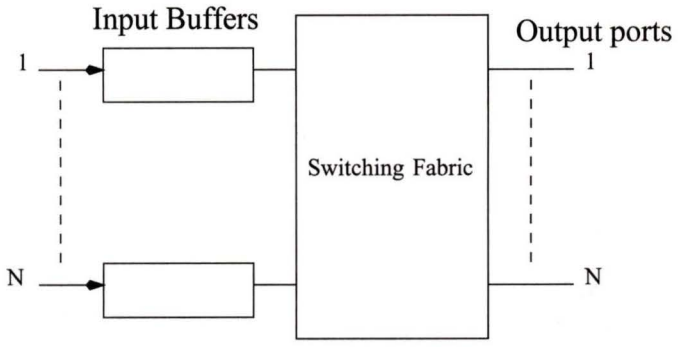

Figure 2.7. *An input buffered ATM switch*

selection [23], and global FIFO selection [21].

The input buffered switches suffer from the head of the line blocking [24-26]. HOL blocking arises when the packet at the head of the queue is blocked from accessing the desired output port. This can happen when some other packet is accessing the output port. With FIFO buffer the maximum throughput that can be achieved theoretically is 58.6% [24-26]. Packet loss in an input buffered switches occur 1) when an input queue is full and the arriving packet has no place to be queued, 2)the packet is lost within the switching fabric due to internal blocking, or 3) when an output buffer is full.

To overcome the problems associated with FIFO input queuing, several modifications has been suggested by various researchers [24-38]. Primary improvements include switch expansion, windowing or a channel grouping.

- Switch Expansion: Liew [29] proposed that the HOL blocking can be reduced by switch expansion. A switch can be expanded by providing a larger number of output ports (O) than the number of input ports (I). In this case the throughput depends on an expansion factor denoted by E, where $E = O/I$. As the value of E increases, the throughput of the switch increases. The switch expansion was adopted to transmit more than one cell to an output port in a single time slot.

- Window mechanism: Hluchyj and Karol [24] introduced the window mechanism in which the input ports provided non-FIFO service. Windowing is a technique conceived to relieve the HOL blocking by also allowing non-HOL cells to contend for the switch output ports. In particular, adopting the window of depth w means that if a cell in position i (HOL cell) is blocked owing to a conflict for switch output port, a chance is then given to the cell in position $i+1$ and so on upto w times. With this approach Hluchyj and Karol achieved the maximum throughput of 88% for N=128 and w=8.

- Pattavina [27] proposed an output channel grouping, in which output ports of a switch are partitioned into disjoint sets, with each port in the set offering same service as other members of the set. It was shown that output channel grouping resulted in an improved throughput of an input buffered switch depending on the group size. Li [28] introduced the concept of input channel grouping. In this technique the input ports are partitioned into disjoint sets each with a multiplexer and each multiplexer skips those input links whose buffers are full. With this approach an incoming cell is lost only when all input buffers within a same input channel group are full. Although both these techniques have a significant impact in improving the performance of input buffered switches, there implementation in hardware is very complex.

2.6.2 Output buffered ATM switches

In output buffered switches as shown in Fig. 2.8 [11], all arriving cells in a given time slot are switched to the output side before the beginning of next time slot, even if all N input ports have cells destined to the same output port. The address of the output queue is determined by the controller of the input buffer. At the output only one cell can be served at a time by the output line and other cells with the same output request have to be buffered. A number of output buffered switches have been proposed [39 - 49].

In output buffered switches each output queue must be able to support one read and N write operations in one slot time. Packet loss occurs 1) when input buffer is full

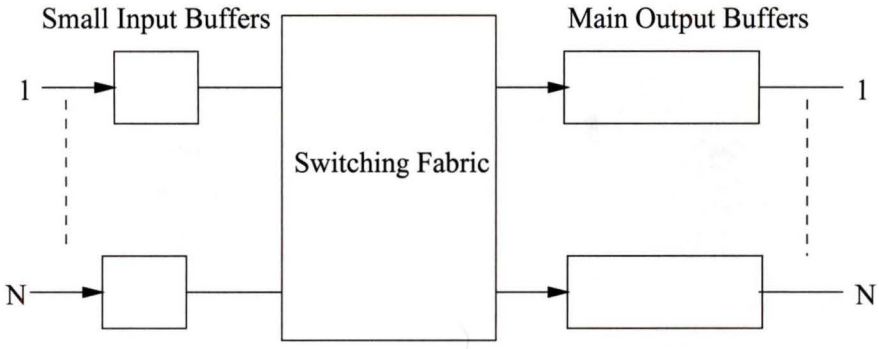

Figure 2.8. *An output buffered ATM switch*

and the arriving packet is discarded, 2) when a packet is blocked inside the switching fabric and is discarded, and 3) when a packet is discarded because the output port is full.

2.6.3 Shared buffered ATM switches

Shared buffer design as shown in Fig. 2.9 [11], employs a single common buffer in which all packets are stored. In shared memory switches all the input and output ports have access to a shared memory module. In a certain time slot upto N cells from the input can be written into the shared memory and upto N cells can be read out of the shared memory. In $N \times N$ shared memory switch, the memory module must be able to perform N number of reads and N number of writes in one time slot. Therefore for $N \times N$ input/output switch operating at line speed of 155 mbps, the shared memory switch requires a line speed of $2N \times 155$ mbps. Similar to output buffering each input port needs a local buffer to store the incoming packet. Depending upon the header information the input port determines a packet's outgoing queue and delievers that packet to the shared buffer. The buffer then queues the packet in the queue containing packets corresponding to the packet's destination address.

Different architectures for shared queues have been proposed to improve the performance of these switches and to minimize the complexity [50-61]. Cell loss in shared buffered switches occur when the input buffer is full and the incoming packet is discarded, and when the shared buffer is full and the arriving packet is discarded.

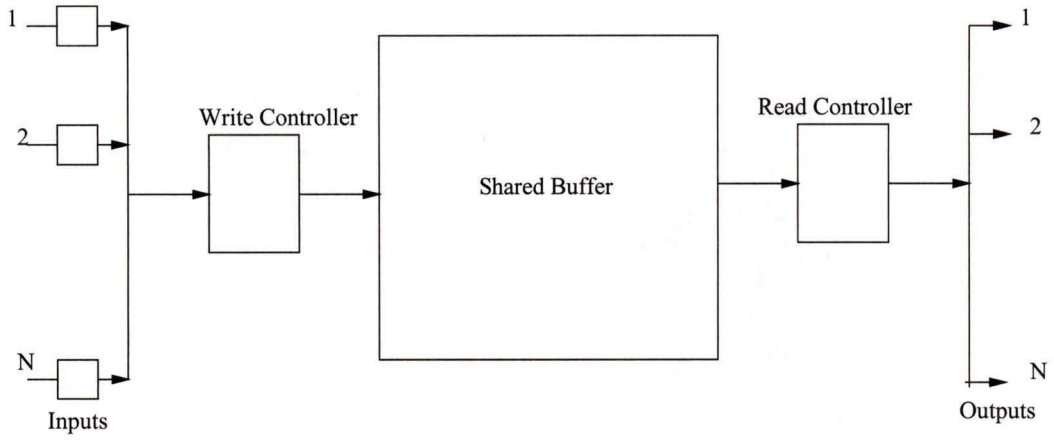

Figure 2.9. *A Shared buffered ATM switch*

2.6.4 Virtual output queued ATM switches

Virtual output queuing (VOQ) switch is an input buffered switch(VOQ) switch that overcomes the HOL blocking associated with FIFO input queuing while keeping the scalability advantage [62]. In this technique each input port maintains a seperate queue for each output port as shown in Fig. 2.10 [11]. The input port controller places each incoming packet in the FIFO queue dedicated to the destined output port. With virtual output queueing 100% throughput can be attained by using suitable scheduling algorithms. In VOQ switches packets are lost when either the input buffer is full or internal and output blocking occurs. The main drawback of VOQ switches is that the packets at the head of queue contend for the switching fabric. Also scheduling packets for a certain output port becomes a problem as each output port must select a packet from N virtual queues located at N input ports.

2.6.5 Virtual input queued ATM Switches

In virtual input queuing (VIQ) (Fig. 2.11 [11]) the output buffer at each output port consists of N queues that are dedicated to each input port. The output controller places the packet in the queue dedicated to the input port that sends it. The main advantage of such approach is that the FIFO speed may not be N times the line rate unlike other output queued switches. In VIQ switches the packets are lost when either the input or output queue is full or when internal and output blocking occurs.

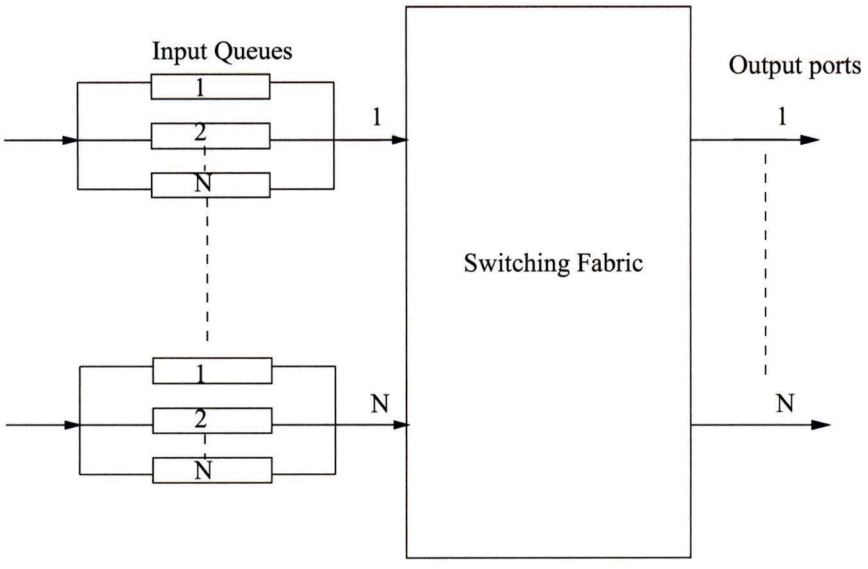

Figure 2.10. *A virtual output queued ATM switch*

The main drawback of VIQ switches is that all the output queues dedicated to a particular input port contend to access the switching fabric.

2.6.6 Multiple input output queued ATM switches

This queuing technique is a combination of input and output queuing switches. It is designed to overcome the drawbacks of multiple input and multiple output queued switches without compromising the performance of a switch. In this switch architecture, as shown in Fig. 2.12 [11], the incoming packet is placed in the FIFO queue of the input buffer which is dedicated to the output port for which the cell is destined. At the output the packets are placed in the queues dedicated to the input port from where they were routed via switching fabric . The main advantage is that the HOL blocking is eliminated and memory speed of each queue matches the line rate. The main disadvantage of this approach is that there is a need to design a switch fabric that can support maximum of $N^2 \times N^2$ connections simultaneously. The packets are lost in such switches when either the input or output buffer is full or when internal and output blocking occurs.

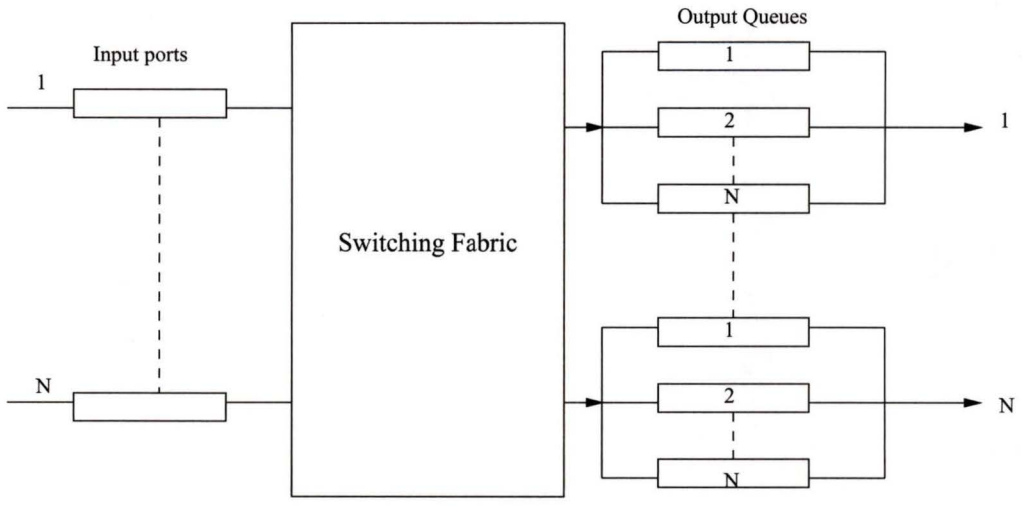

Figure 2.11. *A virtual input queued ATM switch*

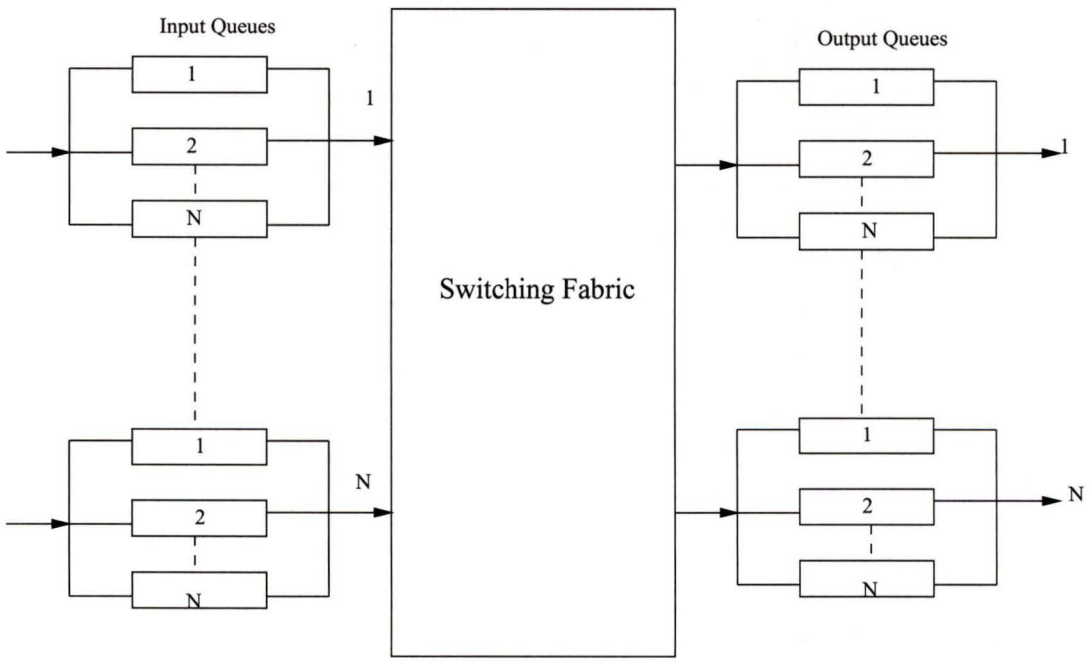

Figure 2.12. *Multiple input and output queued ATM switch*

2.7 ATM switch requirements

Broadband networks must be capable of transporting all kinds of information, ranging from voice to high quality video. These services have different requirements in terms of bit-rate (from a few Kbps up to hundreds of Mbps), behavior in time (constant bit-rate or variable bit-rate), semantic transparency (cell loss rate and bit error rate) and time transparency (delay and delay jitter). These different service requirements have to be met by broadband ATM switches.

- Information Rate : Since the information rates of the different services are very diverse, a large number of information rates must be supported in broadband switches. These rates range from a few Kbps (e.g. for voice) up to values of 150 Mbps (e.g., high definition television HDTV). If an ATM switch operates at 150 Mbps, it does not mean that the switch internally operates at 150 Mbps. Its internal operating speed may be lower or greater than 150 Mbps depending on its architecture. For example, switching can be realized over parallel wires resulting in a lower internal speed, or several 150 Mbps input ports can be multiplexed on a single link resulting in a higher internal speed.

- Performance : The performance of classical STM switches is mainly characterized by the switch's throughput, connection blocking probability, bit error rate and switching delay. In an ATM environment, however, two other parameters are important, namely the cell loss or cell miss-insertion probability and the delay jitter. The effect of connection blocking, cell loss/miss-insertion probability and switching delay are different in ATM switches and are discussed below.

  - Connection blocking: Since ATM is connection oriented during the connection set-up phase, a logical connection needs to be established between a logical input port and output port. The probability of not finding enough resources between the input and output ports of the switch to guarantee the quality of all existing connections and the new connection is defined as connection blocking. Some switch implementations do not have inter-

nal connections. This means that if enough resources (i.e., bandwidth and header values) are available on the input port and the output port of the switch, no new connection is blocked. Thus, a new connection is always accepted if enough resources are available on the external links, without an explicit check of the internal switch resources. Other switch implementations that have internal connections exhibit internal connection blocking. Internal connection blocking occurs when there are not enough resources for the new connection that can be allocated within the switch. The switch's dimension, the number of existing internal connections, and the load on those connections determine the blocking probability of internal connection blocking switches.

- Cell loss and cell miss-insertion probability: In ATM switches, it is possible for many cells to be destined for the same link (this link can be internal or external to the switch). This may cause more cells than the queue's capacity to compete for the queue's storage space, resulting in cells being dropped or lost. The probability of losing a cell must be kept within limits to ensure a high semantic transparency (clear communication between the transmitting and receiving application). Typical values for cell loss probability for ATM switches range between 10^{-8} and 10^{-11} . Some switch architectures are designed in such a way that they do not suffer from cells competing for the same resource internally (e.g. a queue). These architectures do not lose ATM cells internally, but may lose cells only at their input and output ports. These switches are called internally non-blocking switches. Sometimes cells in an ATM switch may be misrouted only to arrive erroneously on another logical connection. This occurs when the cell header gets corrupted during transmission and it goes undetected through the error checking stage. The probability of this cell miss-insertion must be kept within limits. Typical, cell miss-insertion values are a factor of 1000 times less than the cell loss rate.

- Switching delay: Switching delay is the time to transmit an ATM cell

through the switch. Typical values for the switching delay of ATM switches range between 10 and 1000 micro-seconds with a jitter of few 100 micro-seconds or less. Jitter is determined by the probability that the delay of the switch will exceed a certain value. For example a jitter of 100 microseconds for 10^{10} cells, means that the probability that the delay in the switch is larger than 100 microseconds is smaller than 10^{-10} .

2.8 Summary

The ATM switching techniques combine the advantages of both circuit and packet switching techniques by using the concept of virtual channels. One of the most attractive feature of ATM switches is the use of fixed size cells which reduces the queuing delays for higher priority cells, allows cells to be routed more efficiently, and allows for easy implementation in hardware.

ATM switches can be broadly classified on the basis of their stage (single and multistage), path (single and multipath), blockage (blocking and non-blocking), user-participation (single and multicast), internal structure (crossbar, disjoint and banyan based), and buffering stratergies (input, output, shared or combination of these).

Based on the location of buffers inside the switch the ATM switches are classified as input, output or shared buffered switches. The original versions of these buffering stratergies were modified to improve the switch performance. Some of the prominent examples include combined input-output buffering, virtual output queuing, and virtual input queuing.

Chapter 3

Virtual Routing and Queuing (VRQ) Switch

3.1 Introduction

With the increase in the traffic on the Internet, there is a need to develop switch architectures that can meet the requirements of different users without compromising the speed, scalability and QoS. During the last two decades a great deal of work has been done to improve the performance of switches. Different switch architectures including the ones based on different buffering stratergies (input, output, and shared) have been proposed. Amongst these buffering stratergies, the input buffered switches are easiest to implement but their major drawback is that they suffer from head of line (HOL) blocking. To eliminate this problem several different techniques were proposed, the prominent amongst which are windowing [24], input channel grouping [28], output channel grouping [27], switch expansion [29], virtual output queuing [33,63] and combined input output queuing [35,64,65]. Amongst these techniques, the virtual output queuing (VOQ) has received attention of a number of researchers [35-37,66].

The concept of VOQ was first introduced in 1988 by Tamir et al [63]. In VOQ each input port maintains a seperate queue for each output port. This eliminates HOL blocking suffered by input buffered switches as the cells destined to same output ports are only queued behind one another and the cells destined to any other output port are not held behind. VOQ has been employed recently in many studies [64,66]. However, these proposed switches exhibit certain drawbacks. In these switches, during each time slot only one cell can be sent from any input port to any output port and this

prohibits multicasting. Also the cells in the queues contend to access the switching fabric. To overcome this problem a switch architecture named virtual routing and queuing (VRQ) has been recently proposed [11,36]. This switch architecture not only eliminates the problems suffered by earlier switches but also has some attractive features that make it suitable for present and next generation network switches. This chapter describes the architectural details of the virtual routing and queuing switch, and presents the design of its multi-ported input buffer. The design of the multi-ported buffer was obtained using the hardware design language VHDL.

3.2 Introduction to the VRQ Switch

Some of the main aspects of the VRQ switch are:

- Line rate: The switch can operate at the line rate ranging from OC-3 (155.52 Mbps) to OC-48 (2.488 Gbps) and its operating speed matches the line rate.

- Quality of service: Quality of service is a mechanism that defines absolute and relative network performance requirements for various streams of traffic on a network. The switch is capable of supporting different QoS requirements and can also support different service classes such as constant bit rate (CBR), variable bit rate (VBR), unspecified bit rate (UBR), available bit rate (ABR).

- Traffic support: The switch is capable of supporting both broadcast and multicast traffic.

- Packet loss: The switching fabric of the switch is contentionless and this eliminates the packet loss due to internal blocking.

- Modularity: Modularity is a feature which allows a program or an entity to be separated into modules, with each module performing only one function. The switch is inherently modular in its architecture, since the input ports and the output ports are divided into completely independent modules that do not

communicate among themselves.

- Speedup: The switch does not require any speed up unlike other VOQ switches [35,66] since the switching fabric is composed of backplane buses with each bus dedicated to an output port (as shown in Fig. 3.2)

Some of the main limitations of the VRQ switch are:

- Bus matrix is an expensive approach.
- Updating of vitual queues at the output may cause contention.
- Number of buffers per input port increases with an increase in the number of ports.

3.3 Architecture of the Virtual Routing and Queuing (VRQ) Switch

An $N \times N$ VRQ switch as shown in Fig. 3.1 consists of N input and N output ports connected via contentionless switching fabric. At each input port there are N queues, one dedicated to each output port. Each output port contains pointers pointing to the location of the packets stored in the input buffer instead of actual packets. This reduces the output buffer size.

3.3.1 Input port

In the VRQ switch, each input port contains input buffers to store incoming packets as shown in Fig. 3.2. When a cell arrives at the input port, the input port controller processes its header to attain the information about its destined output port. It then stores the cell in the queue dedicated to that output port. The input port controller also informs the destination output port that the packet has arrived by providing it

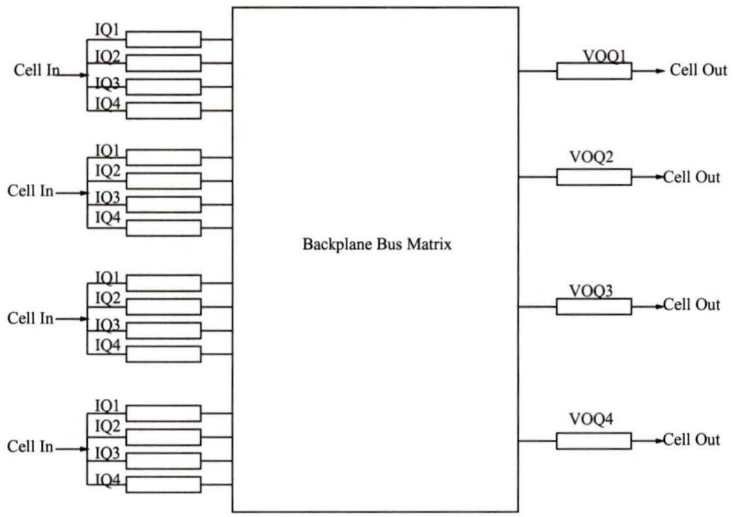

Figure 3.1. *A simple implementation of the Virtual Routing and Queuing switch*

with the address of the location where the packet is stored. Thus packet routing is virtually accomplished through the exchange of this information and hence the name virtual routing. The packet is actually routed out of the switch only when the output port selects it based on some routing algorithm.

3.3.2 Output port

Each output port contains one or more queues which has access to all the input ports and hence the name virtual queuing. The output port contains the addresses of the packets stored in the input queues. The advantages of this approach are: 1) this gives designer much freedom in configuring the queues based on actual traffic requirements, 2) communications between input and output ports is done using much smaller protocol data units (PDU), 3) the storage space at the output is reduced since it stores pointers instead of actual packets, and 4) packet broadcast is easily supported by sending the pointers to the output FIFO queues while keeping only one copy of the actual packet at the input buffer.

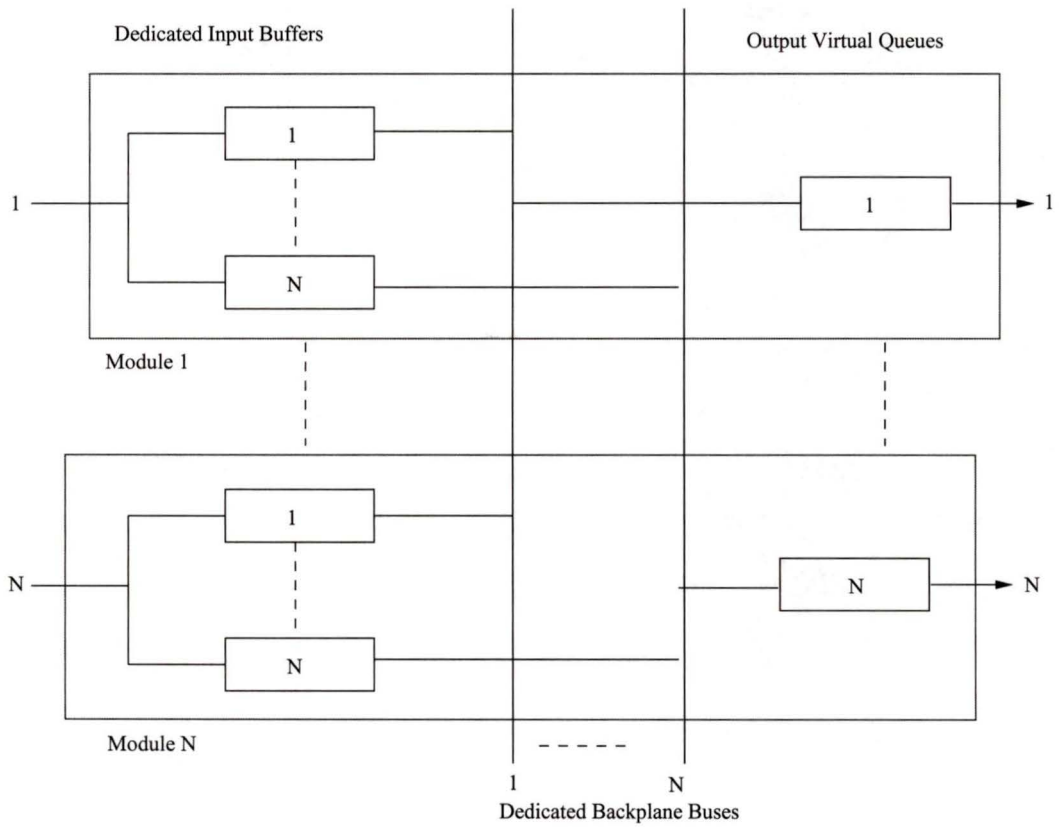

Figure 3.2. *Virtual Routing and Queuing switch*

3.3.3 Switching fabric

The VRQ consists of a contentionless switch fabric composed of backplane buses. Each bus is dedicated to an output port and access to the bus is controlled by the output ports. Also the bus speed matches the line speed and therefore no speedup is required. Additionally, the fabric does not incur any packet loss and internal blocking is also completely removed.

3.4 Switch operation

Every clock cycle a cell arrives at each input port. The input port controller reads the header of the incoming cell to determine its destination. The input port controller then stores the cell in the input queue that is dedicated to the output port for which the cell is destined. The input port controller also informs the destined output port that the cell has arrived and is stored in the particular input port by providing the address of the location where the cell is stored. The information exchanged at this stage is the data unit (DU) whose contents depend on the protocol being used. In case of an ATM switch, the data units carry the information about VPI/VCI and the input ports where the cell is stored.

During the next clock cycle all the cells whose addresses are stored at the head of FIFO output queues are routed out. This operation is done simultaneously by all output ports. Here the packet service protocol is local and is performed in parallel by all output modules. Since multiple output ports can access cells placed in same input port therefore the cells in the queues do not contend over the switching fabric. The switch does not suffer from any blocking due to internal switching fabric since it is output driven and composed of buses dedicated to each output port, as already mentioned.

3.5 Advantages of the VRQ over other Switches

1. The HOL blocking in VRQ switch is completely eliminated since the cells destined to same output port only gets queued behind one another.

2. The input buffers operate at line rate and each output queue needs to process at most N pointers which is much simpler then processing N packets.

3. The number of queues at each output port can be increased depending on the scheduling algorithm used (priority based, service based, etc.).

4. Like other VOQ switches, VRQ also has a virtual queue at the output which contains the addresses of the packets stored in the input buffer rather then the actual packets. This allows the output buffer to store more addresses thereby reducing output buffer occupancy.

5. Data broadcast is also very simple to implement without making any extra copies since each input port is connected to the bus destined to particular output port only and the output port can access cells from any of the input ports.

6. The main advantage of VRQ's architecture over ones proposed earlier is that two or more output ports can read the cell from the same input port in a single clock cycle. This is made possible by modifying the input buffer design. Input buffer is basically a memory that stores cells temporarily. The main limitation of the earlier designed input port buffers is that they can at most perform one read and one write operation per clock cycle. The design of VRQ's multi-ported input buffer, presented in section 3.6, explains how this limitation is overcome. This multi-ported input buffer performs one write and N read operations per clock cycle. This feature makes the VRQ switch suitable for very high data rates e.g., upto tera-bits/sec.

7. The VRQ does not suffer from any internal and output blocking.

3.6 Design of a Multi-ported Input Buffer

A multi ported buffer can support high data rates without any speed up requirements. It is capable of performing one write and N read operations simultaneously as opposed to traditional buffers that can perform one read and one write operation per clock cycle. This feature is made possible by using dedicated back plane buses.

The $N \times N$ switch is composed of N input and N output ports. At each output port we have a virtual queue that can access packets from any of the input ports. Each input port is composed of N queues, where N is the number of output ports for $N \times N$ switch. In this section we focus our attention on the hardware details of one input port only and the replication of this module can be used to design a switch of any size. Each input port will have a 1-input, N-output buffer that is capable of performing one write and N reads per clock cycle and is composed of three main hardware components, the write controller, the read controller and the buffer as shown in Fig. 3.3

3.6.1 Write controller

Every clock cycle a cell arrives at the input port via incoming line. The incoming cell is sent to the write controller for header processing before being written into the buffer. The input port controller determines the cell type from the cell header i.e., if the cell is user, signalling, empty or OAM type. If the cell is empty, it is dropped and is not stored inside the buffer. If the cell is OAM type (operation, administration and management) or user type, then the write controller sends its VPI/VCI information to the the look-up table for translation. There is a seperate controller for each input port rather than a single controller, and this avoids access conflicts. After receiving the translated VPI/VCI and the output port information from the look-up table, the write controller updates the cell header. It then transfers this cell to the queue dedicated to the destined output port. The buffer stores the cell till it is read by the output port. The write controller is further composed of three submodules: the cell recognizer, the look-up table and the VPI/VCI translator as shown in Fig. 3.4. These components are discussed below in detail.

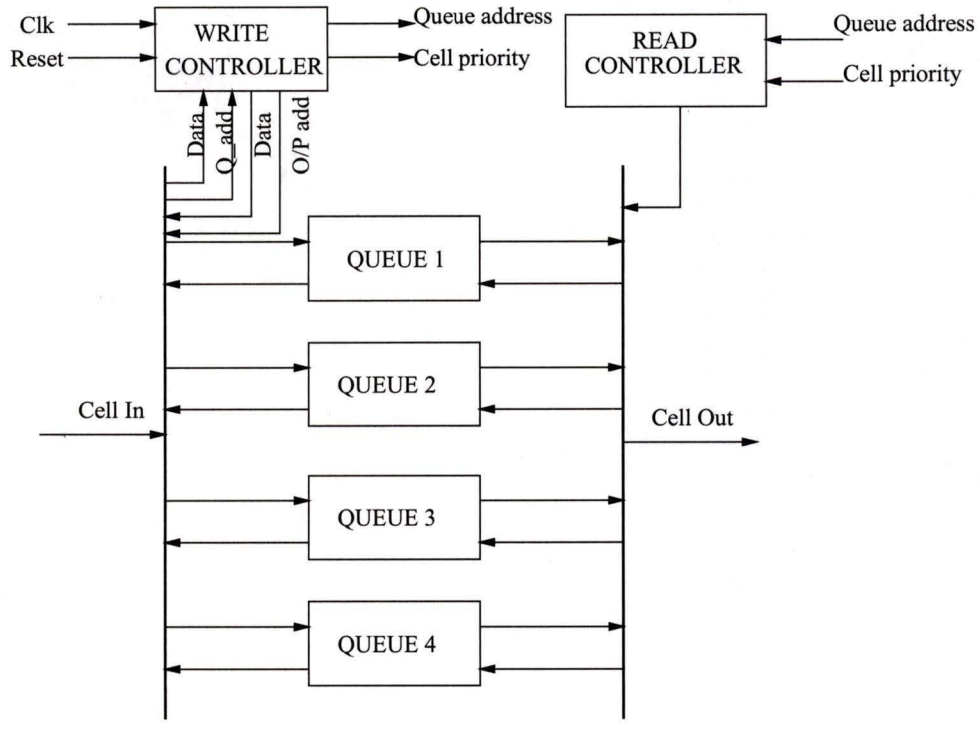

Figure 3.3. *Schematic of input port of a VRQ Switch*

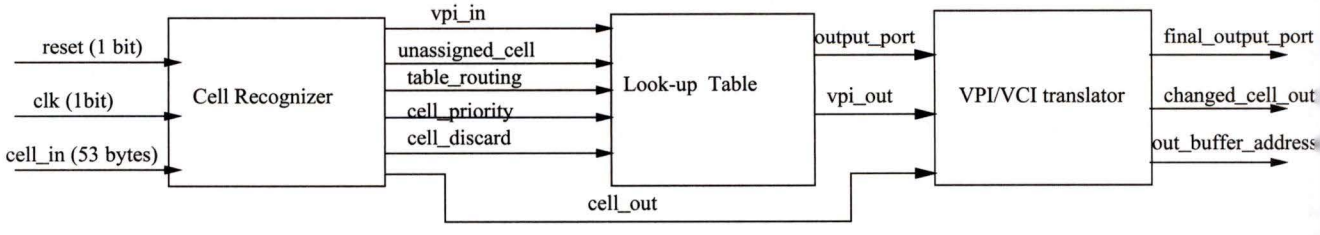

Figure 3.4. *Main components of write controller of VRQ switch*

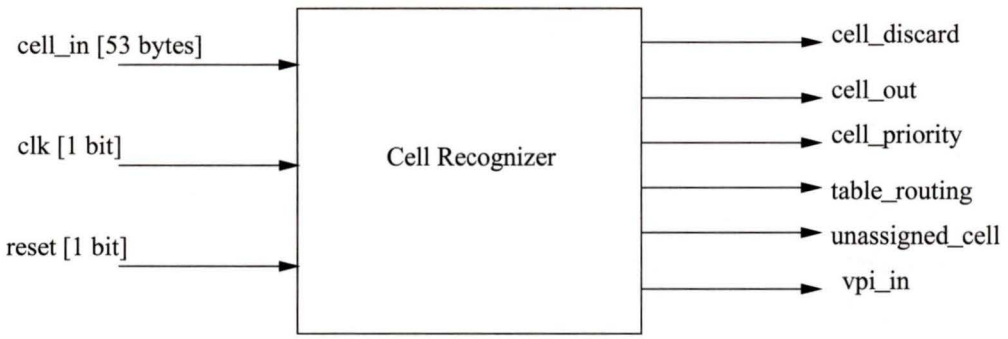

Figure 3.5. *Block diagram of a Cell Recognizer*

Table 3.1. *ATM predefined headers for determining cell types*

Type	Octet1	Octet2	Octet3	Octet4
Unassigned	00000000	00000000	00000000	0000xxx0
Signalling	00000000	00000000	00000000	01010aac
Segment VP OAM	0000aaaa	aaaa0000	00000000	00110a0a
End-to-End VP OAM	0000aaaa	aaaa0000	00000000	01000a0a
ILMI	00000000	00000000	00000001	0000aaa0

"a" indicates that the bit is available for ATM functions, "x" is don't care bit, and "c" means sender sets CLP=0, but may be changed in network.

3.6.1.1 Cell Recognizer

The block diagram of cell recognizer with various input and output signals is shown in Fig. 3.5. The cell recognizer of the write controller determines the type of cell from the four octet cell header field of the incoming cell (cell_in signal). The last octet is neglected since it is used for header error control (HEC) in the physical layer. The four octets of the header contains pre-assigned cell header values at UNI as shown in Table. 3.1 and pay load (PT) type as shown in Table. 3.2 respectively.

After determining the type of cell i.e., user, OAM, signalling, or empty, the cell recognizer makes the appropriate decision about whether a cell is to be discarded or not, whether it has priority over other cells, and whether the routing table needs to be looked up. If the cell is user or OAM type then the cell recognizer sends their

Table 3.2. *Defined payload types*

PT code	Meaning
000	User data cell, congestion not experienced, AAI=0*
001	User data cell, congestion not experienced, AAI=0*
010	User data cell, congestion not experienced, AAI=0*
011	User data cell, congestion not experienced, AAI=0*
100	Segment VC OAM
101	End to End VC OAM

Table 3.3. *Possible cell types*

code	Type of cells
000	Empty
001	Signalling
010	Segment VP OAM
011	End to End VP OAM
100	Segment VC OAM
101	End to End VC OAM
110	ILMI
111	User

VPI/VCI information to look-up table for translation. It also activates the look up table by sending a signal (table_routing signal) to it. In this design the cell recognizer sends only the VPI information to the look up table (vpi_in signal). The look-up table may either translate VPI, VCI or both. The empty cells are dropped (cell_discard signal) without any further processing. The signalling cells are sent to higher layers (control layer) of the ATM protocol model and the the other components of write controller are also informed about it (unassigned_cell signal). Also, if the cells are user type then their priority has to be determined (cell_priority signal). If the 31^{st} bit of cell header is zero, then the cell is of highest priority and if it's value is 1 then it has the lower priority. After determining the cell type, the cell recognizer arranges the cell in the order shown in Table. 3.3.

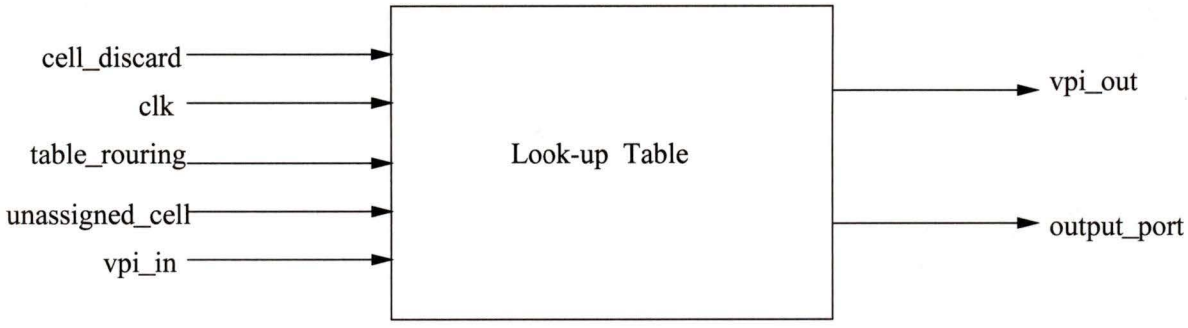

Figure 3.6. *Block diagram of look-up table*

3.6.1.2 Look up table

The block diagram of look-up table is shown in Fig. 3.6. The look up table receives signals from the cell recognizer indicating the validity of a cell. For empty (cell_discard signal) and signalling (unassigned_cell signal) cells the look up table remains inactivated. For user cells the cell recognizer initiates the look up table by sending a signal (table_routing=1) to it. The look up table also receives the VPI, VCI or both from the cell recognizer of write controller (vpi_in signal). It then looks up for the matching outgoing VPI/VCI and the corresponding output port. For simplicity, only 8-bit VPI is considered. The look-up table matches this VPI with the available incoming VPIs present inside it. If a match is found then a corresponding outgoing VPI (vpi_out signal) along with the associated output port (output_port signal) information is sent out. If there is no match found then the look up table sets the outgoing translated VPI to "ZZZ...." and output port information to "ZZZ" indicating that the cell is to be discarded. The look up table derives these VPI values from its internal tables which are setup either manually for permanent virtual connections or through signalling for switched virtual connections. In this study this table is set manually as shown in Table. 3.4

3.6.1.3 VPI/VCI translator

The block diagram of VPI/VCI translator is shown in Fig. 3.7. Once the translated VPI (vpi_out signal) along with its output port (output_port signal) information is received from the look-up table, the write controller updates the cell header. It then

Table 3.4. *VPI translation table*

Possible incoming VPI	Matching outgoing VPI	Associated Output port
00000001	00000011	1
00000010	00000111	2
00000100	00001111	3
00001000	00011111	4

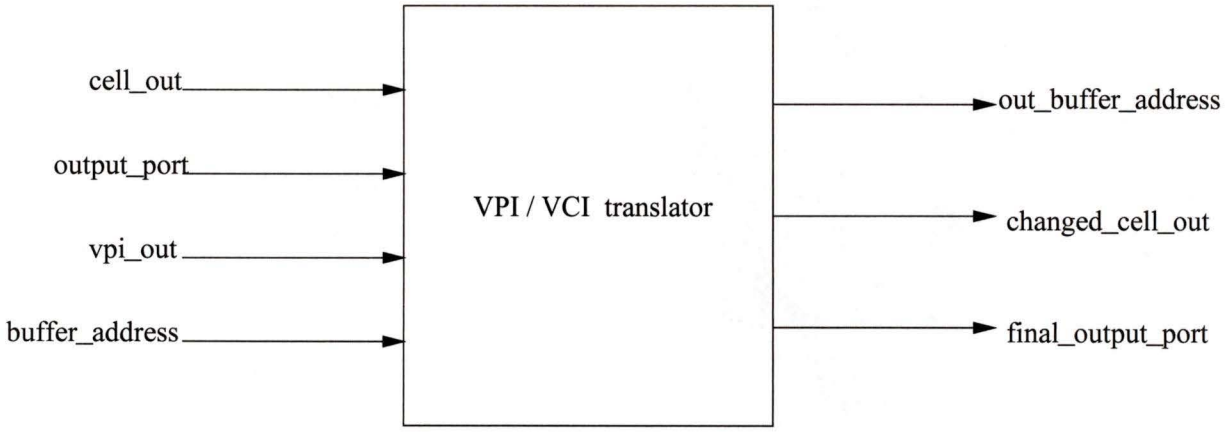

Figure 3.7. *Block diagram of VPI/VCI translator*

transfers the cell (changed_cell_out signal) into the queue associated with the output port (final_output_port signal) to which the cell is destined. It also sends a pointer to the output port, pointing to the input port where cell is stored (out_buffer_address signal). If no match is found for the incoming VPI in the look-up table then the controller discards that cell without further processing.

The complete design of the write controller of the multi-ported buffer as shown in Fig. 3.4 is obtained after synthesis of the design obtained by using VHDL. The VHDL code for the various components (cell recognizer, look-up table, and VPI/VCI translator) are attached at the end of this thesis in Appendix A. The functioning of the queue and read controller is briefly discussed in the next two sections. VHDL design and synthesis of these two components was not performed.

3.6.2 Queue

In case of the VRQ switch there are N queues (for $N \times N$ switch) at each input port, i.e., one queue for each output port. The cells destined to the similar output port only get queued behind one another in these queues and this eliminates HOL blocking. For simplicity only one queue (say queue number 1) of input port 1 is designed. Queue number 1 indicates that this queue contains cells destined to the output port 1. Most of the earlier studies [35,66] dealing with VOQ's placed the cells inside these queues in a random manner. They did not develop any logic for placing cells inside the queue. To make the design simple and easy to implement in hardware, a logic to place the cells inside the particular queue is developed. Each queue is further divided into N sub-modules and each sub-module consists of a flip-flop and a combinational logic. This logic is used to maintain equal traffic flow in all sub-modules and to provide quick access to the output ports. This feature makes the switch ideal for supporting very high data rates and to support even bursty traffic without compromising the switch's performance.

Each flip-flop contains the information about the sub-modules i.e., if it is empty or not. If the sub-module is empty then the flip flop is set to zero, otherwise it is set to one. The combinational logic is used to determine the next free location inside the queue and its decision is based on previous and present flip flop status. If the previous state of flip-flop is empty (i.e., if 1^{st} sub-module's flip-flop=0), then the cell is stored in that location otherwise the next available location is checked. The truth table according to which this logic works to determine next free location is shown in Table. 3.5

3.6.3 Read controller

The read controller associated with each input port works in association with the output port controller during read operation of the switch. During read operation the cells whose pointers are at the head of the queues at the output port are routed out of the switch. Since in the VRQ switch, the output port contains only cells pointers instead of actual cells, the output port need to contact input port during read operation. This may be done by providing a separate read controller for each input port.

Table 3.5. *Truth table*

Initial state of flip-flop	Present state of flip-flop	Result of combinational logic	Signal from combinational logic
0	X	0	Enable (store cell in the first submodule)
1	0	0	Enable (store cell in the current free submodule)
1	1	1	Disable (Present location is busy and cannot store the cell)

The output port controller requests the read controller of the input port that a cell stored in particular location has to be routed out. This information to the read controller is accompanied by information about the input port number and the cell's location number in that input port. Since input port consists of dedicated queues for different output ports, the read controller contacts only the relevant input queue. From the sub-modules the cells are selected depending on the cell's location number. After the cell is read from the selected sub-module, the flip-flop of that sub-module is set to zero. This location can then be used to store a new incoming cell.

3.7 Summary

This chapter describes the architectural details of a new switch, Virtual Routing and Queuing, and presents the design of its multi-ported input buffer. The VRQ switch overcomes the limitations of earlier proposed switches, in particular HOL blocking (suffered by input buffered switches) and switching fabric contention suffered by VOQ switches. Unlike VOQ switches, VRQ is capable of supporting multicasting as well. The switching fabric of VRQ is composed of backplane buses which eliminates the internal blocking completely. Also, since the output port contains pointers to packets other than the actual packets, VRQ does not suffer from output blocking.

The multiported input buffer of the VRQ switch consists of a read controller, write controller and the queues dedicated to output ports. For simplicity, it is assumed that the output port consists of only one output queue although in principle any number of output queues can be configured depending on the scheduling algorithm used. The multi-ported buffer allows VRQ to perform N read operations per clock cycle enabling it to support multicasting.

The write controller of the multi-ported input buffer consists of cell recognizer, look-up table, and VPI/VCI translator modules. The cell recognizer determines the cell type and based on its header decides if the cell is to be discarded, if the look-up table needs to be consulted, and if the cell has a higher priority. If required, the look-up table is consulted to obtain information about the new VPI/VCI and the output port for which the cell is destined. The VPI/VCI translator finally updates the cell header which togather with the information about destination output port and location of the cell in the input buffer is sent to the output port. The N queues in each buffer store the packets till they are routed out of the switch. During the read operation, depending on the pointer address, the read controller is used to route the packet out from the input port.

Chapter 4

Generation of Traffic

4.1 Introduction

Characteristics of traffic play a crucial role in performance analysis and design of computer networks. Models of computer network traffic help to design switching hardware, routers, and networks. To develop accurate traffic models it is important to understand the nature of the incoming load. Models proposed earlier assumed traffic to be Markovian in nature and exhibited short range dependencies. Markovian traffic models assumed arrival rate to be Poisson distributed and the length of messages were distributed exponentially. Such traffic models were characterized with burst lengths that tended to smoothen when averaged over long time-scales. During mid 80's discrepancy was observed in predicted and real traffic data and it became apparent that traditional traffic models for computer networks were less appropriate. This was primarily due to the increase in the number of Internet users. It has been observed that there is approximately 20 percent average monthly load increase on the Internet and that by itself can qualitatively change the nature of the traffic [73,74]. The increase in traffic load encouraged researchers to explore the impact of traffic characterization and its implementation on the design of computer networks [67-71] .

In 1986 Jain [72] proposed the concept of packet trains in order to capture the observed burstiness in actual packet streams. His model assumed that a group of packets travel together as a train, contrary to Poisson model that assumed the packets are independent. However, recent studies have shown that the traffic exhibits self similar behaviour. Self-similarity is the property mostly associated with fractals whose visual appearance remains unchanged regardless of the scale at which they are viewed

[70]. Since a self-similar process has observable bursts at a wide range of time-scales, it can exhibit long range dependence i.e., values at any instant are correlated to future values. The importance of long range dependence in network traffic has also been observed in some of the recent studies [75-77], which show that the packet loss and delay behaviour is radically different when simulations use either real traffic data or synthetic data that incorporates long-range dependence. This chapter provides a brief overview of traffic models that are used to generate synthetic traffic, and then discusses features of the poisson and pareto traffic models which are used in this study to model traffic data.

4.2 An overview of traffic models

Traffic models are used as input to analytical or simulation studies of switches. Important features of traffic that have significant impact on the performance of a network include traffic correlation, burstiness and the average data rate. Burstiness describes the tendency of the traffic to occur in clusters. It affects the buffer occupancy and leads to network congestion and data loss. The correlation describes the relation between packets at different times. It was recently discovered that realistic network traffic exhibits long range dependence i.e., the auto-correlation fuction approaches zero very slowly in comparison with the exponential decay characterizing short range dependent traffic [69,73]. In this section the short-range and long-range dependent traffic models are discussed briefly.

4.2.1 Poisson arrival model

The most commonly used model for packet arrivals in analytical modelling is the Poisson arrival model [78]. One important property of Poisson processes is that the sum of several independent processes result in a new Poisson process whose rate is sum of the component rates [79]. Poisson traffic models have been shown to accurately describe user initiated TELNET and FTP connections [68]. In this model the inter-arrival times t_i (time between arrival of packets i and $i+1$) are independent, and are assumed to be exponentially distributed given by the probability density function, $f_T(t) = \lambda e^{-\lambda t}$, where λ (packets/s) is the average packet arrival rate.

The histogram of inter-arrival times, in Poisson arrival model, exhibits a exponentially decreasing function. Exponential distribution is characterized by a coefficient of variation (the ratio of standard deviation to the mean) value equal to one and exponentially distributed data plots as a straight line on a logarithmic scale. Feldmeier [80] plotted observed traffic's histogram data on a logarithmic scale to verify if the data were exponentially distributed. These data exhibited different behaviour than the expected exponential characteristics and this led to the failure of the Poisson model. Similar results were obtained by other researchers as well [68,69,73].

The Poisson model was subsequently extended to compound Poisson arrival process [81] in which the packets arrive in batches. The inter-arrival times between these batches were independent and exponentially distributed. Though the extended Poisson arrival model exhibited realistic behaviour compared to Poisson arrival process it was found impractical. This was due to the presence of lull periods in the traffic.

4.2.2 Packet train model

In the packet train model the packets travel together in a group. The first packet of the train makes the routing decision and the remaining packets follow it. This model is applicable only when the packets are coming or going to a single source. Unlike Poisson model, the train model is not additive i.e., the sum of number of trains is not a train.

Analytical modeling using a simplified form of train model can be done using a two-state markov model. The source model can be either in a generation (train) or an idle state (inter-train). The transitions between these states are memoryless (Markovian). The duration of these two states is exponentially distributed, with inter-arrival time of the order of several seconds. The main drawback of this approach is that the crucial parameters like the length of the train or inter-arrival time between the trains have to be selected arbitrarily [67]. Walter et al. [67] modified the packet train model by using long packet trains (ON periods) with long inter-arrival times (OFF periods). The superposition of long ON and OFF periods, that do not strictly

alternate each other, is designed to exhibit self similar behaviour on large time scales.

4.2.3 Self-similar model

Recent work [68,69], has shown that traffic can be modelled in a better way using statistically self-similar processes. As mentioned earlier, self-similar phenomenon displays structural similarities across a wide range of spatial or temporal scales. For example, Ethernet LAN traffic is found to exhibit "fractal" self-similar behaviour, in the sense that there is no natural length of the burst. Traffic that is bursty on many or all time scales can be described statistically using the notion of self-similarity. Self-similar traffic exhibits similar statistical properties at a range of time-scales; milliseconds, seconds, minutes, hours, days, and even weeks. It has also been observed that merging of self-similar traffic streams does not result in smoothing of traffic [69]. Traditional models described using markovian models have limited memory of the past and they reflect short range dependence. Also when averaged over long periods of time, traffic data generated by such models resulted in smoothing of the data stream. It has also been found that the degree of the self-similarity tends to increase with the load level on the Ethernet [69].

The long-range dependence essentially means that significant correlation exists across all time scales. On the contrary, Poisson and other models that do not accurately model long-range dependence can underestimate performance measures such as average packet delay or maximum queue size. The theory of self-similar traffic is not as well developed as Poisson, but given that self similar traffic models are better than Poisson models [68], self-similar traffic models have become important tools for generating traffic.

Several models for generating self-similar traffic have been proposed over the last decade and some of these models are briefly explained here.

1. Alternating renewal process R(t): In this model the duration of OFF and ON periods are derived from a "heavy tailed" Pareto distribution. By multiplexing n independent instances of the R(t) process and taking only ON series into consideration self-similar traffic can be generated [69].

2. M/G/∞ queue model: This model considers the customer arrival as a Poisson process and service time is drawn from Pareto distribution with infinite variance [68,69,85]. As per this model the process of customer arrival at time 't' is asymptotically self-similar.
3. Fractional Brownian Motion (FBM): It has been observed that the traffic generated by fractional Brownian motion also exhibits self-similar behaviour [83] and can be generated using; fourier transform [84], displacement process [85], Pareto distribution[86], and Markov process [87].

The self-similar behaviour of observed data traffic as shown in Fig. 4.1 depicts a sequence of sample plots of packet counts (i.e., number of packets arriving per unit time) versus time for both Poisson and Pareto distributed traffic patterns. It can be seen from the plots that Poisson traffic tends to smoothen when simulated for long time scales. This is in contrast to self-similar traffic that exhibits same behaviour over all time scales. This is inline with the results obtained by [69] who examined the Ethernet network for 27 consecutive hours in August 1989.

4.3 Traffic Generation

Simple traffic generation models can be considered as point processes which estimate the number of packets that arrive in a time interval. These point processes give a random sequence, which may be used to determine the inter arrival time between the packets. Traffic can also be modeled using fluid flow models. These models group the traffic into flows and these flows are characterized by average and burst data rates [11].

Any traffic data can be essentially described in terms of two characteristics. First, the description of its flow which necessarily means taking into account the average data rate, the maximum burst rate and the average packet length. Second, the description of inter-arrival time which necessarily means taking into account the randomness of periods between arrival of packets. Similar to flow description, inter-arrival time can be described in terms of an average interval time and some minimum inter-arrival time. Thus the knowledge about how often packets arrive (i.e., inter-arrival

Figure 4.1. *Self-similar behaviour of traffic*

time information) or how many packets arrive (i.e., flow description information) can be used to generate traffic. This section describes these two characteristics for two traffic models which are used in this study to generate traffic. First, the poisson traffic model is used to generate traffic which does not exhibit self-similarity and is described in section 4.3.1. Second, the pareto traffic model is used to generate self-similar traffic and is described in section 4.3.2.

4.3.1 Poisson traffic model

The flow description of the traffic data is provided in terms of some or all three basic parameters 1) ρ_a, the average data rate (bps), 2) σ, the maximum burst rate (bps), and 3) L, the average packet length (bits). The Poisson model assumes that both traffic flow and inter-arrival time are distributed exponentially.

4.3.1.1 Flow description

The probability density function (PDF) of flow rate is given by

$$f_R(\rho) = be^{-b\rho} \tag{4.1}$$

where b is the shape parameter that determines the steepness of the exponential curve. Equation 4.1 imply that the probability of occurence of flow rate $\rho = 0$ is b and higher flow rates occur with decreasing probability of occurence, i.e., as $\rho \to \infty$, $f_R(\rho)$ becomes vanishingly small. The average data rate, ρ_a for traffic flow described by equation 4.1 is given by

$$\rho_a = \int_0^\infty \rho b e^{-b\rho} d\rho = \frac{1}{b} \tag{4.2}$$

which implies that the shape parameter b can be determined from the average data rate b=$1/\rho_a$. Equation 4.1 can thus be rewritten in terms of ρ_a as

$$f_R(\rho) = \frac{1}{\rho_a} e^{\frac{-\rho}{\rho_a}} \tag{4.3}$$

4.3.1.2 Inter-arrival time description

The inter-arrival time distribution is also assumed to be exponential, however, the distribution is modified such that the minimum inter-arrival time is a. The modified

inter-arrival time distribution is given by

$$f_T(t) = \begin{cases} 0 & t < a \\ be^{-b(t-a)} & t \geq a \end{cases} \tag{4.4}$$

where $a \geq 0$ is the location parameter (the minimum inter-arrival time in our case) and b is again the shape parameter. The average inter-arrival time is given by

$$t_a = \int_{t=a}^{t=\infty} tbe^{-b(t-a)}dt \tag{4.5}$$

$$t_a = a + \frac{1}{b} \quad \text{s} \tag{4.6}$$

Note that as the shape parameter b decreases (increases) the average inter-arrival time increases (decreases). The a and b parameter are related to the basic flow parameter (ρ,σ,and L) as follows. The maximum number of packets in a time period t that could be produced by a given source is given by

$$N_m = \frac{\sigma t}{L} \quad \text{(packets)} \tag{4.7}$$

and this number can be used to determine the minimum time between two adjacent packets, that is

$$a = \frac{t}{N_m} = \frac{L}{\sigma} \quad \text{(seconds)} \tag{4.8}$$

In the same time period t, the average number of packets that are produced by a given source is given by

$$N_a = \frac{\rho t}{L} \quad \text{(packets)} \tag{4.9}$$

The average inter-arrival time between two adjacent packets is thus

$$t_a = \frac{t}{N_a} = \frac{L}{\rho} \quad \text{(seconds)} \tag{4.10}$$

However, t_a is also described by equation 4.6 substituting the value of t_a and a in equation 4.6 and solving for b using equation 4.10 yields

$$b = \frac{\rho\sigma}{L(\sigma - \rho)} \quad \text{(packets/second)} \tag{4.11}$$

The PDF for the inter-arrival time, in terms of three basic flow description parameters ρ, σ, and L, can thus be written as

$$f_T(t) = be^{-b(t-a)} \tag{4.12}$$

$$f_T(t) = \frac{\rho\sigma}{L(\sigma-\rho)} e^{\frac{-\rho(\frac{\rho t}{L}-1)}{\sigma-\rho}} \tag{4.13}$$

4.3.2 Pareto traffic model

Heavy-tailed distributions such as the Pareto are used to model traffic that exhibits long-range dependence and self-similarity. Heavy-tailed distributions usually exhibit high or infinite variance, and $p(X > x) \propto 1/x^{\alpha}$ as $x \to \infty$, i.e., the probability of occurrence of x decreases at a rate of $1/x^{\alpha}$. A higher value of parameter α results in a lower probability of occurrence of x.

4.3.2.1 Flow description

The Pareto distribution is defined by the PDF

$$f_R(\rho) = \begin{cases} \frac{ba^b}{\rho^{b+1}} & \rho_{min} \leq \rho \leq \rho_{max} \end{cases} \tag{4.14}$$

where ρ_{min} is the minimum data rate which could be zero and ρ_{max} is the maximum burst rate and equal to σ in our notation. a is the location parameter and b is the shape parameter that, similar to the Poisson distribution, determines the steepness of the curve and is analogous to parameter α mentioned above.

The mean (μ) and variance (σ_v^2) of Pareto distributed numbers is given by

$$\mu = \frac{ba}{b-1} \tag{4.15}$$

$$\sigma_v^2 = \frac{ba^2}{(b-1)^2(b-2)} \quad where \quad b > 2 \tag{4.16}$$

Given the value of b (e.g., determined from observed traffic characteristics or if assumed) the value of parameter a can be determined for some average data rate ρ_a

$$\rho_a = \mu = \frac{ba}{b-1} \tag{4.17}$$

$$a = \frac{\rho_a(b-1)}{b} \tag{4.18}$$

4.3.2.2 Inter-arrival time description

The distribution of inter-arrival time between adjacent packets is given by an equation similar to 4.14

$$f_T(t) = \frac{ba^b}{t^{b+1}}, a \leq t < \infty, \tag{4.19}$$

where a is the minimum inter-arrival time, which similar to the poisson model, is given by $a = L/\sigma$, and b is the shape parameter. The average inter-arrival time is given by $t_a = L/\rho_a$ seconds as mentioned earlier in section 4.3.1. An expression of average inter-arrival time can also be obtained in terms of parameters a and b by

$$t_a = \int_{t=a}^{t=\infty} t\frac{ba^b}{t^{b+1}}dt = \frac{ba}{b-1} \quad \text{seconds} \tag{4.20}$$

Using equation 4.20 and the general expression for t_a

$$t_a = \frac{L}{\rho_a} = \frac{ba}{b-1} \tag{4.21}$$

the value of b can be determined as

$$b = \frac{L}{L-\rho_a} = \frac{\sigma}{\sigma-\rho_a} \tag{4.22}$$

The shape parameter for inter-arrival time distribution thus depends only on the average and burst rates. The shape parameter lies between the following extreme values

$$\begin{aligned} b &= 1 \quad \text{when } \sigma \gg \rho \\ b &= \infty \quad \text{when } \sigma = \rho \end{aligned} \tag{4.23}$$

The value of $b = 1$ implies fairly bursty traffic while $b = \infty$ implies a constant traffic rate. Thus the shape parameter b for the inter-arrival time distribution varies between 1 and ∞.

4.3.3 Generating traffic using cumulative distribution functions

Given the PDF, random numbers drawn from a distrbution can be easily generated with the help of its cumulative distribution function (CDF). The expression for CDF is obtained by integrating the expression for PDF

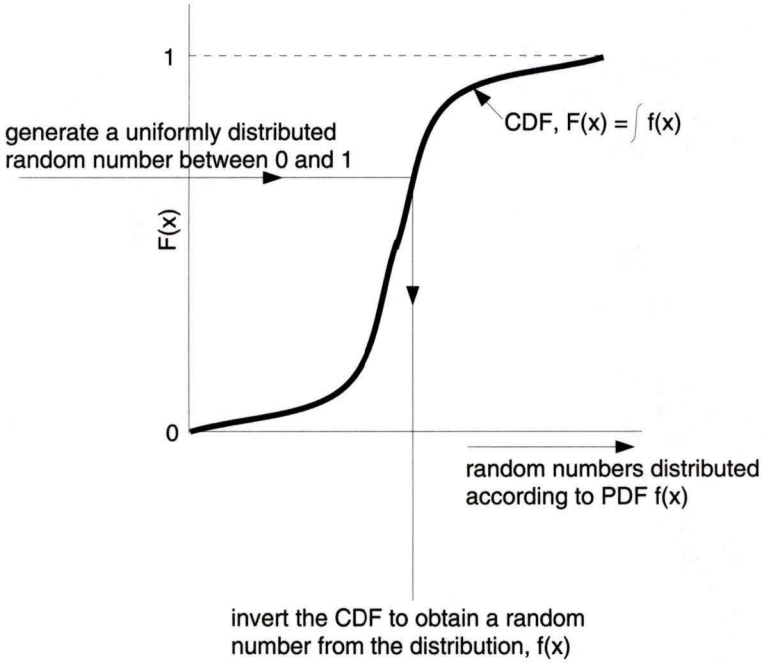

Figure 4.2. *Generation of random numbers (representive of incoming traffic or inter-arrival time) using CDF*

$$F(x) = \int_{-\infty}^{x} f(x)dx \tag{4.24}$$

For example, the CDF for the pareto distribution is given by

$$F(x) = \int_{-\infty}^{x} f(x)dx = \int \frac{ba^b}{x^{b+1}}dx = 1 - (\frac{a}{x})^b \tag{4.25}$$

The random numbers drawn from a given distribution could be interpreted as the numbers of packets arriving per unit time or the inter-arrival time between the packets, as the case may be.

This section explains how random numbers may be drawn from a certain distribution. Lets assume the parameters of the distribution are known [11]. First, a uniformly distributed random number y is obtained between 0 and 1. This number is assumed to represent a point on the CDF such that $F(x) = y$. The corresponding x value, which would represent the random number drawn for a given distribution is then obtained by inverting the expression for the CDF. For example, for obtaining pareto distributed numbers equation 4.25 is inverted,

$$x = \frac{a}{(1 - F(x))^{\frac{1}{b}}} = \frac{a}{(1 - y)^{\frac{1}{b}}} \tag{4.26}$$

where y is uniformly distributed random number between 0 and 1 as mentioned earlier. This procedure is repeated, as many times as desired, to obtain xs for different values of ys. The approach is illustrated graphically in Fig. 4.2

4.4 Summary

Modelling of traffic plays an important role in analysis and design of communication networks. Models which describe the behaviour of incoming load more realistically are likely to give a better insight into the performance of switches and other network components than those which cannot model traffic realistically. For example, recent work has shown that network traffic exhibits self-similarity which is not accounted for by poisson model and this can lead to underestimation of performance measures such as average packet delay and maximum queue size. At present various models

are available which attempt to model the self-similar behaviour traffic.

Traffic data can be essentially described in terms of its two primary characteristics. First, how often the packets arrive (i.e, information about inter-arrival time) and second, how many packets arrive (i.e, information about flow description). Knowledge about any one of these characteristics can be used to describe and generate traffic. In this chapter inter-arrival time and flow description characteristics of two traffic models are described. The Poisson model is used to generate traffic data which does not exhibit self-similar characteristic and Pareto model is used to generate self-similar traffic. The next chapter uses this data to simulate the VRQ switch along with three other switches based on three different buffering stratergies.

Chapter 5

Performance Analysis of the VRQ switch

5.1 Introduction

Before a switch can be implemented in a communication network it is necessary to assess its performance. The performance analyses are usually performed by simulating the switch with the help of commercially available software (such as OPNET) or such software is written by the person resposible for performing analysis. What ever the case, the principle behind performance analysis is to gain insight into the performance of the switch as it would operate in real world. A model of a switch is constructed which is then driven with synthetic or, if available, observed traffic data. Performance analysis is also helpful in comparing different switches, for example in terms of their architecture and/or assessing the effect of traffic characteristics on switch's performance.

A switch's performance is usually evaluated in terms of three basic criteria: cell loss probability, throughput and cell delay. Cell loss probability is defined as the fraction of cells lost within the switch. Cell loss occurs due to blocking or buffer overflow. In ATM switches, it is possible for many cells to be destined for the same link (this link can be internal or external to the switch). This may cause more cells than the queue's capacity to compete for the queue's storage space, resulting in cells being dropped or lost. The probability of losing a cell must be kept within limits to ensure a high semantic transparency (clear communication between the transmitting and receiving application). Some switch architectures are designed in a way such that

they do not suffer from cells competing for the same resource internally. These architectures do not lose ATM cells internally, but may loose cells only at their input and output ports. These switches are called internally non-blocking switches. Sometimes cells in an ATM switch may be misrouted only to arrive erroneously on another logical connection. This occurs when the cell header gets corrupted during transmission and it goes undetected through the error checking stage. The probability of this cell miss-insertion must be kept within limits. Typically, cell miss-insertion values are a factor of 1000 times less than the cell loss rate. Switching delay is the time to transmit an ATM cell through the switch. Typical values for the switching delay of ATM switches range between 10 and 1000 micro-seconds. Switching delays are primarily caused when the cells wait inside the input, output or shared buffers before being delievered to their destined output ports. An efficient switch design would result in minimum delay. Throughput is defined as the the average number of cells which are successfully delivered by the switch per unit time and can be estimated as $(1-C_L)\rho_a$, where C_L is the cell loss fraction and ρ_a is the average data rate. Efficient switches are expected to support high throughput rates.

The three performance criteria (cell loss probability, throughput, and cell delay) among other factors are influenced by queue size, memory speed, memory control, traffic load and switching fabric.

- Queue size: The queue size determines the number of ATM cells that can be stored within a queue in case of congestion. Its size depends on the performance requirements of the system and the selected queuing methodology.
- Memory speed: The access time required by the queuing memory depends on the queuing methodology used. Speed of the input, output and shared buffer can be a bottleneck resulting in increased delays and cell loss.
- Memory control: To control the queue of a switch, additional control logic is required. The complexity of this control logic depends on the queuing methodology. For instance, the FIFO queuing methodology used in input and output queuing requires only simple control logic, whereas shared queuing requires a dynamic memory management function. With the increase in the complexity, there is a decline in the speed and this results in increased delay.

- Traffic load: The nature of the incoming traffic has a significant effect on the performance of a switch. With the increase in the traffic load, the probability of cell loss increases and delays are large. Also, bursty nature of the traffic results in increased cell loss and larger delays, and traffic which doesn't exhibit burstiness, results in decreased cell loss and delays.
- Switching fabric: The switching fabric is used to route the data from any input port to their destined output port from where it is routed to its proper destination. The switching fabric choice effects the performance of the switch in a significant manner. Efficient switching fabric will result in minimum cell loss and minimum switching fabric contention.

In this chapter the performance of the VRQ switch is assessed in terms of the three basic performance criteria. The VRQ switch is simulated using C programs. Other commercially available softwares such as MATLAB or OPNET were not used due to the lack of flexibility they offer in terms of programming. The VRQ switch is simulated using Pareto distributed traffic and is also compared with input, output and shared buffered switches, in terms of its performance. The effect of traffic data on VRQ and input, output and shared buffered switches is also assessed and discussed.

5.2 Performance analysis of the switches

To simulate the switches numerically following assumptions were made,

- All switches are assumed to be of dimension 8×8, i.e., they all have eight input and eight output ports.
- The switches have contentionless switching fabric and there is no loss or delay due to the switching fabric used.
- The traffic is modelled using flow description methodology [11], as explained in Chapter 4. This technique is used to determine number of packets arriving per unit time.
- The switches were simulated for 10,000 seconds to record their cell losses and delays.

- The average date rate was assumed to be 1000 packets/second.
- The values of parameters a and b for Pareto distribution are taken as 334 packets/second and 1.5 respectively and were calculated as shown below [11].

$$b = \frac{\sigma}{(\sigma - \rho)} \quad \text{(packets/second)} \tag{5.1}$$

Assuming maximum burst is three times the average data rate.

$$b = \frac{3000}{(3000 - 1000)} = 1.5 \tag{5.2}$$

A value of b close to one implies bursty traffic and value of b close to 2 implies smoothened traffic.

$$a = \frac{\rho_a(b-1)}{b} \tag{5.3}$$

$$a = \frac{(1000)(1.5-1)}{1.5} = 334 \quad \text{(packets/second)} \tag{5.4}$$

The results obtained from these simulations are discussed in the following sections.

5.2.1 Input Buffered Switch

In input buffered switch, the incoming packets at the eight input ports are stored in the respective input buffers and a decision is made about their destined output port in a random manner as explained in section 2.6.1.

In these simulations the cell loss and delay are measured on the following basis.

- Packets are lost when a input buffer is full. This is either due to the arrival of a huge burst at any input port or due to HOL blocking.
- Packets are delayed as they wait to be transferred to their destination. Only when a packet at the head of the queue has been transferred to its destination, the packet behind it get a chance to be transferred.

The HOL blocking is the primary cause of increased delay and cell loss in input buffered switches.

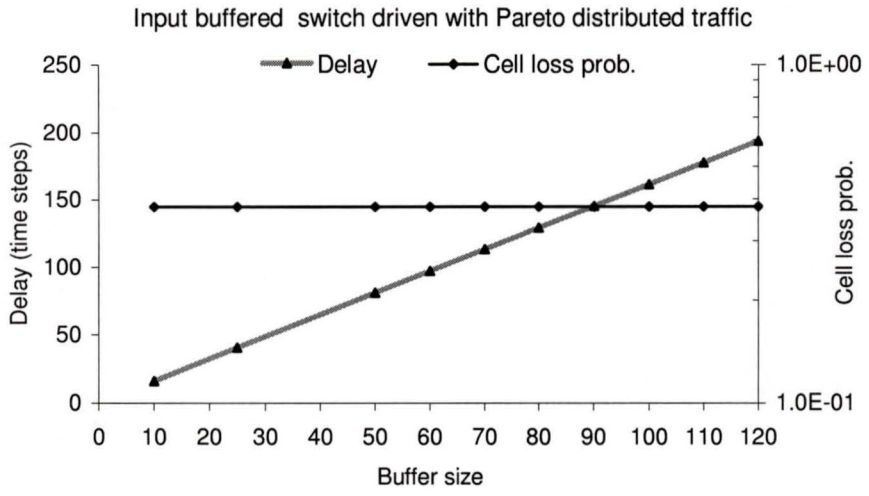

Figure 5.1. *Performance of the input buffered switch when driven with Pareto distributed traffic.*

Fig. 5.1 shows the performance of the input buffered switch when driven with Pareto distributed traffic. The cell delay is simulated to increase linearly with the buffer size. The cell loss remains constant at a high value of about 37%. This cell loss value is similar to the theoretical value of 42% obtained by [24]. This is expected since Head of Line (HOL) blocking, the primary and major cause of cell loss in input buffered switches, does not cause the switch's performance to improve even with an increase in buffer size. The schematic representation of the working of the input buffer switch is also given in Fig. 5.2

5.2.2 Output buffered switch

The incoming packets are stored temporarily at the input port for header processing. Depending on the header information, the input ports then route the packets to the appropriate output ports. The packets at the output ports are stored in the buffers and are serviced in a FIFO manner as explained in section 2.6.2. The losses and delays in the output buffered switches are caused by the following reasons.

- Packets are lost when the output buffer is full, independent of the size of the input buffer, and
- The cell delay occurs only when the cells in the FIFO queue have to wait to be

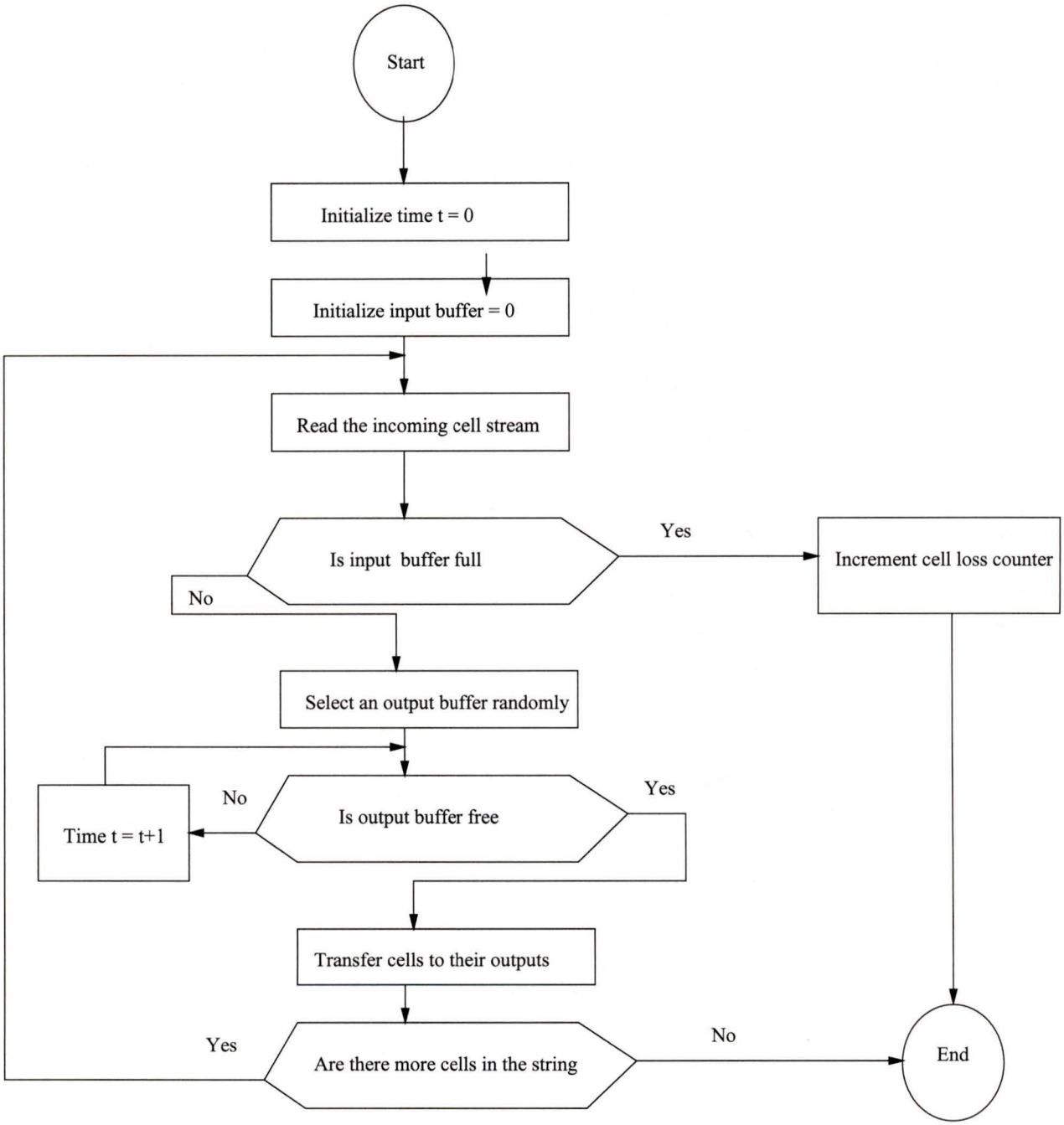

Figure 5.2. *Flowchart explaining the working of input buffer switch*

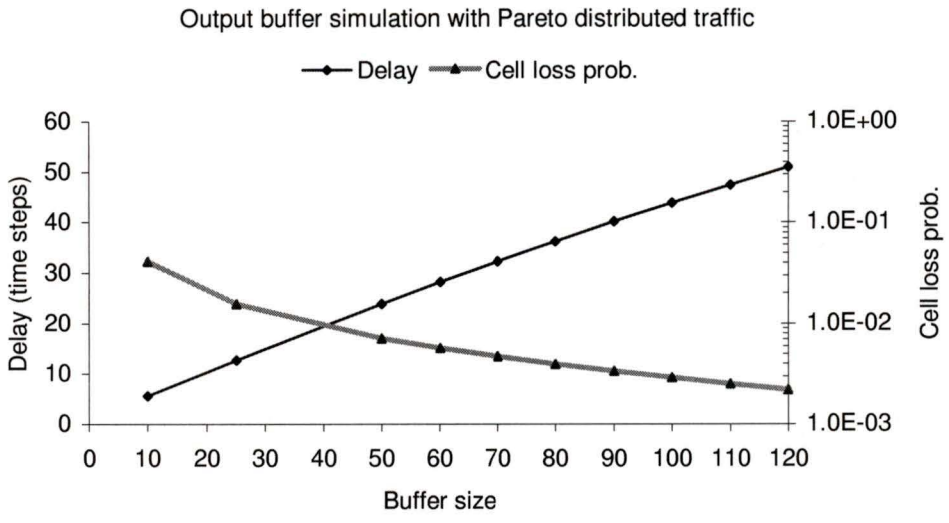

Figure 5.3. *Performance of the output buffered switch when driven with Pareto distributed traffic.*

serviced.

Unlike an input buffered switch, an output buffered switch does not suffer from HOL blocking and this leads to significant improvement in throughput. Figure 5.3 shows the cell loss and delay for the output buffered switch when driven with Pareto distributed traffic. The improvement in cell loss, compared to the input buffered switch due to elimination of HOL blocking, is clearly seen. Also as expected, the increase in buffer size results in higher delays and lower cell loss probabilities.The complete working of the output buffer switch is explained in the form of flowchart in Fig. 5.4.

5.2.3 Shared buffered switch

The shared buffered switch consists of a single common buffer used to store both incoming and outgoing packets. Like output buffered switch, the shared buffered switch also consists of a small buffer at the input ports. During the next time slot, the input port routes the packet into the shared buffer. The shared buffer then queues the packets into the appropriate output queues as explained in section 2.6.3. These

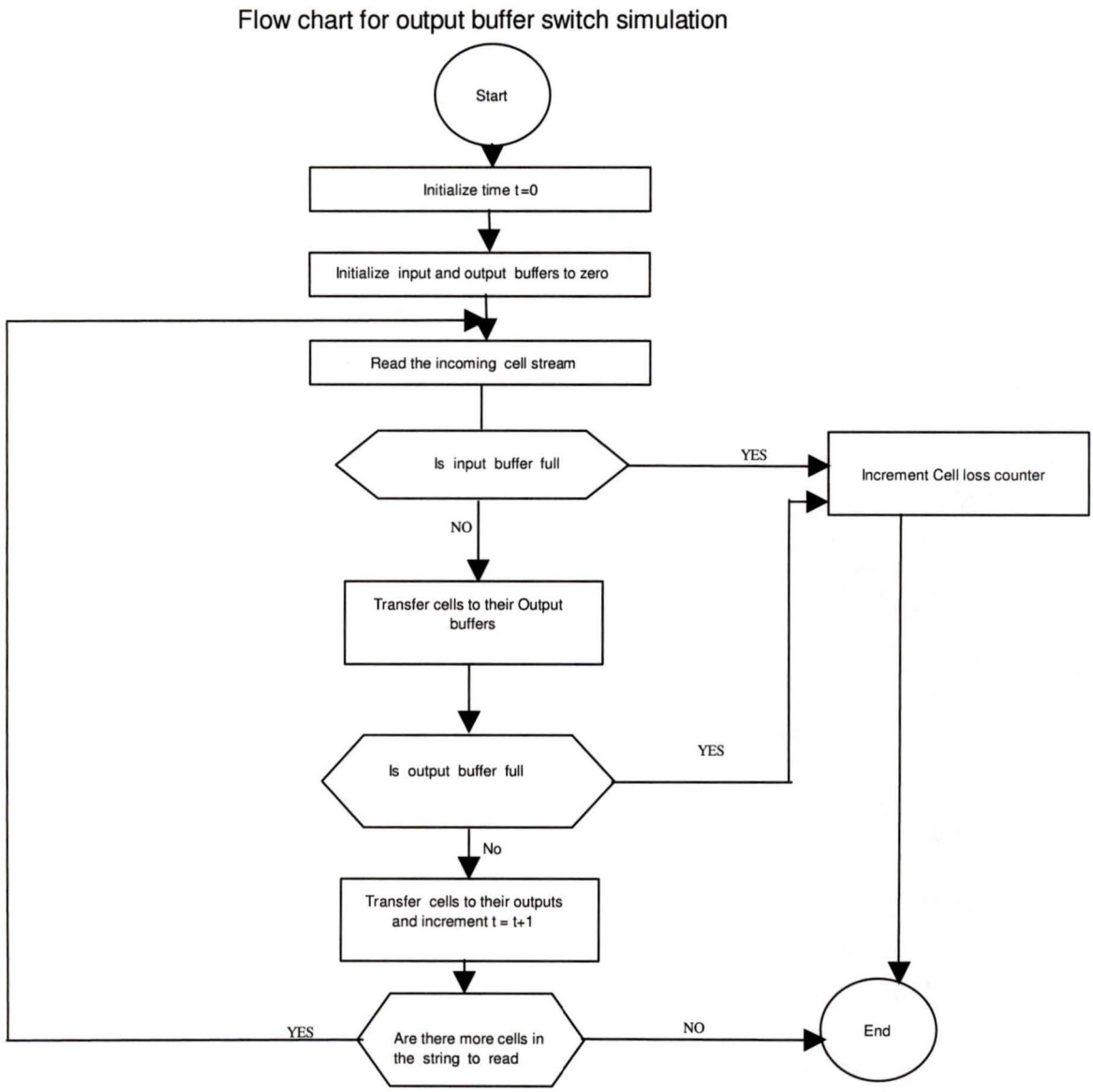

Figure 5.4. *Flowchart explaining the working of output buffer switch*

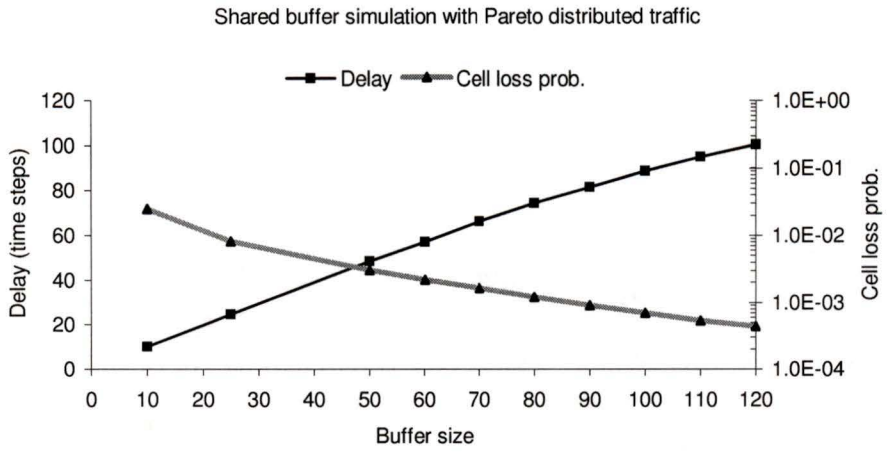

Figure 5.5. *Performance of the shared buffered switch when driven with Pareto distributed traffic.*

packets are then routed out to the appropriate output ports during the next time slot. The primary difference between the output and shared buffered switches is the ability of the shared buffered switches to store cells as long as the complete buffer space is not full. For example, if the output buffer for port 1 is full and a cell arrives destined for port 1 then in an output buffered switch cell loss will occur. However, in a shared buffered switch the incoming cell can be stored in empty slots designated for queues of other ports. This increases the efficiency with which the available space may be used and reduces cell loss. The losses and delays in shared buffer are due to the following reasons.

- Packets are lost when the shared buffer is completely full, and
- Cell delay occurs when either the packets have to wait to be queued into proper queues or when the packet in the queue waits for the service. The cells are serviced in FIFO manner.

Because shared buffered switches attempt to use the total buffer space in a manner efficient than the output buffered switches they are likely to reduce cell loss.

Figure 5.5 shows the performance of the shared buffered switch when driven with Pareto distributed traffic. The efficient performance of the shared buffer switch results little cell loss values, an order of magnitude smaller than the output buffered

switch. The cell delay for shared buffered switch irrespective of the traffic is higher than output buffered switch. An increase in cell delay is an inevitable result of decrease in cell loss. In the shared buffered switch cell loss is reduced and therefore cells stay much longer in the buffer causing the cell delay to increase. Complete working of the shared buffer is explained in the form of flowchart in Fig. 5.6

5.2.4 VRQ switch

The VRQ switch consists of buffers at each input port to store incoming packets. So in principle VRQ is an input buffered switch. However, unlike input buffered switch, each buffer in VRQ is composed of queues, one dedicated to each output port. The incoming packets are stored in these buffers and depending on their header information the pointer is sent to the output port, pointing to the location of storage of the packet in the input buffer. Packets are moved out from the input port only when the output port requests it. The output port may consist of one or more queues depending upon the scheduling algorithm used. For example for priority based scheduling algorithm, each output port will have two queues, one storing the pointers of higher priority packets and another storing the pointers for lower priority packets. For the simulations presented here FIFO based scheduling algorithm was used. The switching fabric is composed of broadcast matrix of backplane buses and is contentionless as explained in sections 3.3 and 3.4. For the simulations performed here the output queue for the VRQ switch was fixed to 300.

In VRQ switch the losses and delays occur due to the following reasons.

- Packets are lost only when the input buffer is full or output queue is full, and
- Cell delay occurs when the pointers wait in the output queues for their service.

Figure 5.7 show the performance of the VRQ switch driven with Pareto distributed traffic. Comparison of performance of the VRQ switch with other switches show that despite being an input buffered switch the VRQ performs in a manner similar to the output buffered switch. This improvement is brought upon by storing the pointers

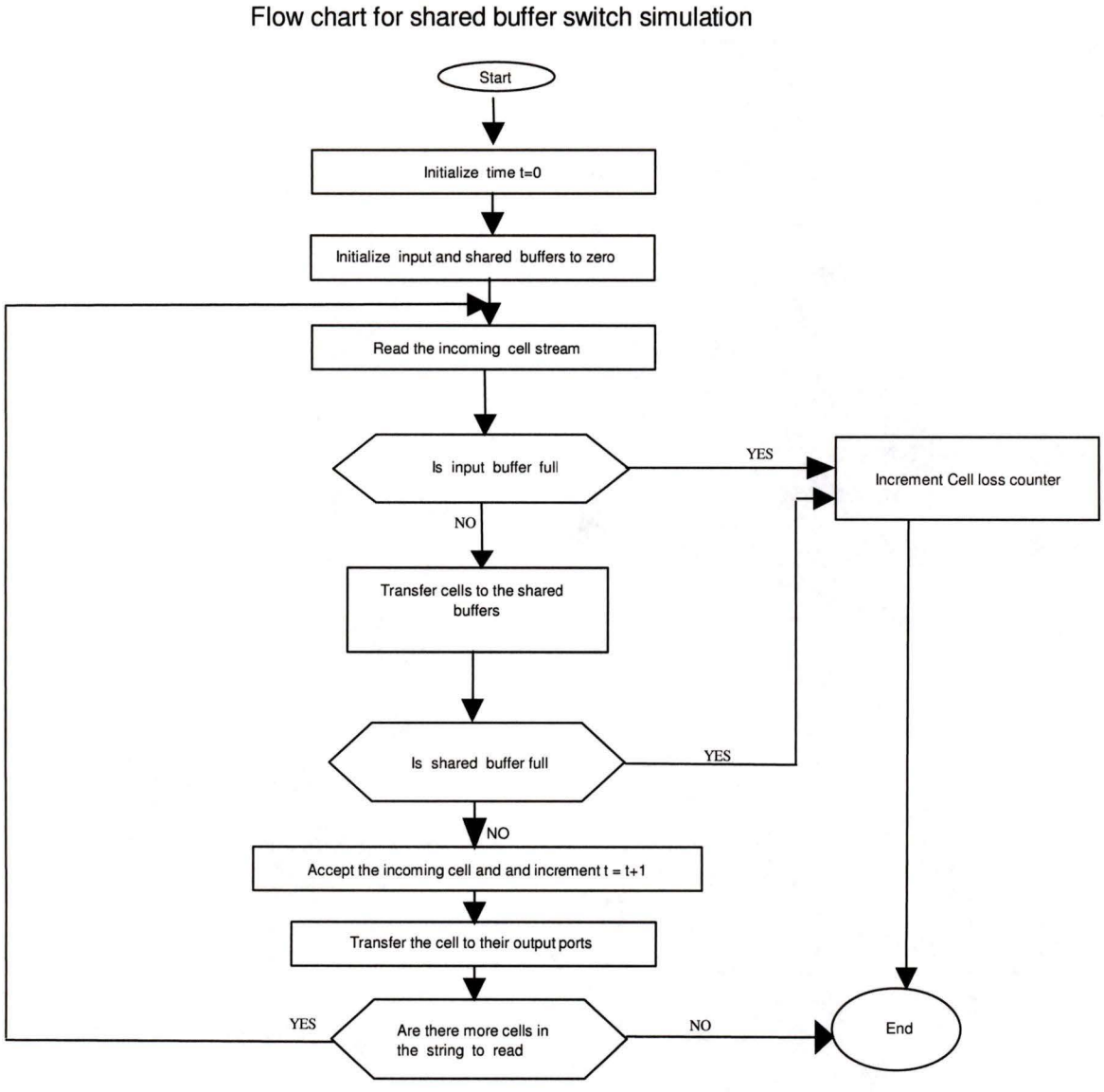

Figure 5.6. *Flowchart explaining the working of shared buffer switch*

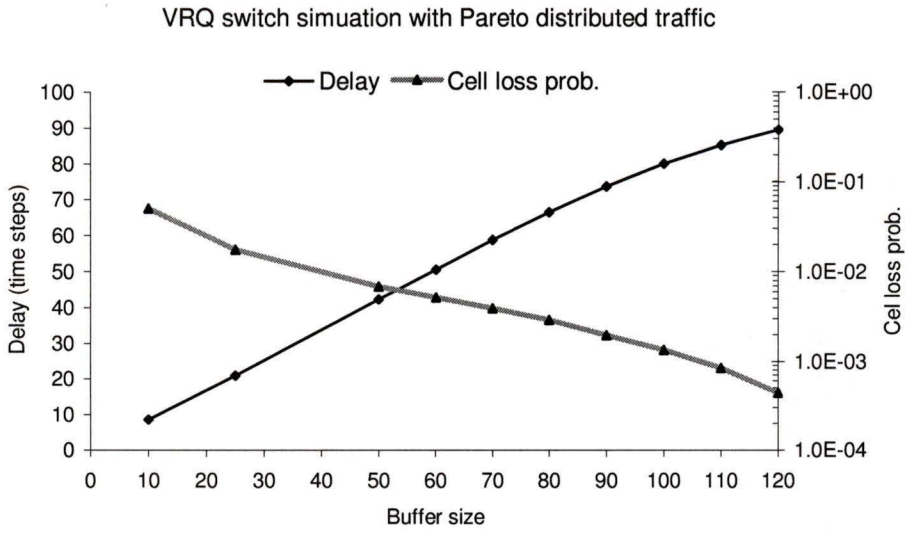

Figure 5.7. *Performance of the VRQ switch when driven with Pareto distributed traffic.*

to the packets at the output port (rather than the packets themselves) and providing separate dedicated queues for packets destined to different output queues. The performance of the VRQ switch is assessed in more detail in the next section where inter-comparisons are performed. The complete working of the VRQ is explained in the form of flowchart in Fig. 5.8

5.3 Comparison of VRQ switch with other switches

This section compares the VRQ, and the input, output and shared buffered switches in terms of cell loss and delay, and their hardware complexity. The cell loss and delay inter-comparisons are performed for Pareto distributed traffic only since it exhibits realistic traffic characteristics.

Figure 5.9 compares the cell loss (a) and cell delay (b) for the four switches when driven with Pareto distributed traffic. Figure 5.9 shows that despite being an input buffered switch the performance of the VRQ switch lies somewhere between the performance of the output and shared buffered switches both in terms of cell loss and

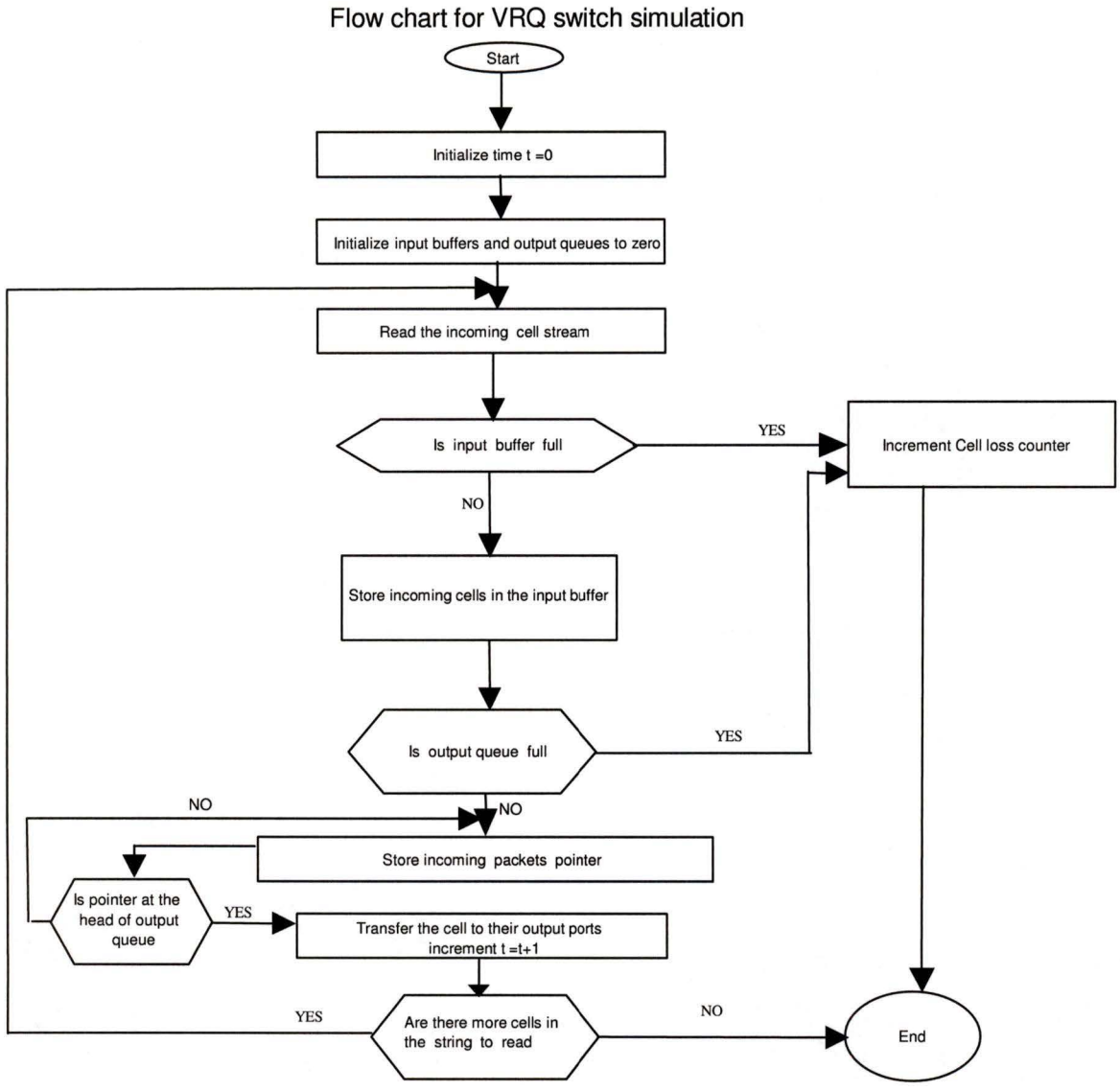

Figure 5.8. *Flowchart explaining the working of VRQ switch*

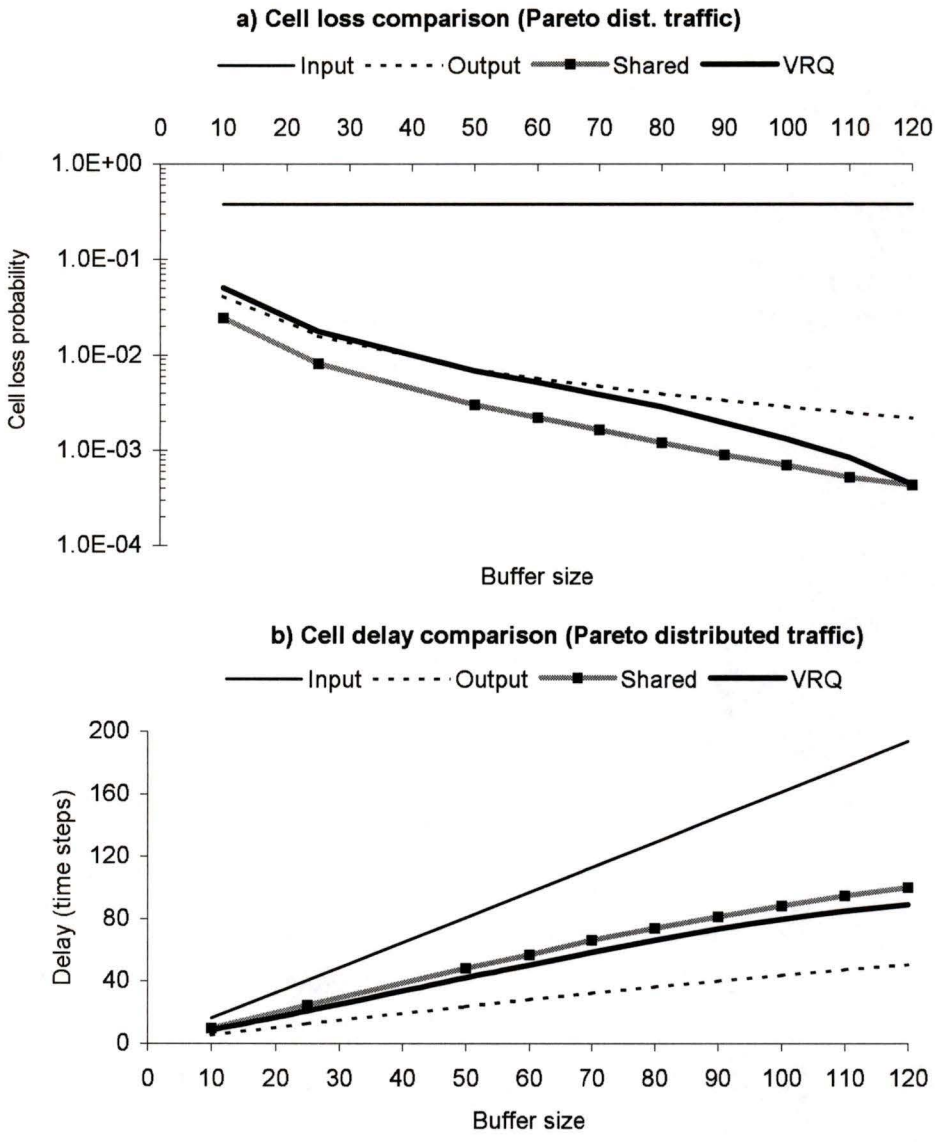

Figure 5.9. *Intercomparison of the VRQ and input, output, and shared bufferedswitches when driven with pareto distributed traffic for cell loss (a) and cell delay (b).*

delay. These results are similar to the ones obtained by Agarwal [88] who performed simulations for the VRQ switch using Bernoulii traffic data.

In Figure 5.9a the cell loss is minimum for the shared buffered switch primarily because of its efficient use of buffer space. Due to the HOL blocking the cell loss in the input buffered switch does not reduce with an increase in buffer size. This is not true for other switches which experience decreased cell loss as the buffer size is increased. The limitation of HOL blocking in improving a switch's performance is clearly obvious. The performance of a switch in terms of cell loss improves from the input, to output, to shared buffered switch.

The cell delay for the VRQ switch is slightly higher than the output buffered switch but far less than the input buffered switch. Infact, both VRQ and the output buffered switch displays minimum cell delay. The cell delay is highest for input buffered switch due to HOL blocking. Figure 5.9b shows that the input buffered switch stands alone from the rest of the switches in terms of its performance. Elimination of HOL blocking from other switches (output and shared buffered, and the VRQ) clearly distinguishes them from the input buffered switch but not so much so from one another.

In terms of hardware complexity, input buffered switches are easiest to implement. The output buffered switches are next hardest to implement because they require buffers at the input port for header processing and also the buffers at the output port for storing the cells. The shared buffered switches are the most difficult to implement because like output buffered switches they also require a buffer at the input port for header processing plus the implementation of shared buffer is much more complex architecturally. Since VRQ is an input buffered switch it is easy to implement in hardware.

5.4 Summary

Performance analysis give insight into the operation of a switch as it may behave in a real world. Commercially available software or programs written by person responsible for performance analysis may be used. The three basic performance criteria are cell loss, cell delay and throughput, and among other factors these are affected by queue size, memory speed, memory control, traffic load and switching fabric.

This chapter present results from simulation of input, output and shared buffered switches and the VRQ switch when driven with Pareto distributed traffic. It is seen that the performance of the switches improve from input, to output, to shared buffered switches in terms of cell loss because of the increasingly efficient use of buffer space to store incoming cells. In terms of hardware complexity, the complexity of the switches increases in approximately the same order.

The performance of the VRQ switch is shown to be comparable to that of the output buffered switches while being principally an input buffered switch. This improvement is brought upon by providing dedicated input queues for different output ports and storing pointers to the incoming packets rather than the actual packets at the output port. The analysis presented in this chapter and the discussion about hardware complexity imply that VRQ meets the primary characteristics of an ideal switch reasonably well.

Chapter 6

Contributions and Future Work

6.1 Thesis Contributions

The design of efficient and high performance switches, among other issues, is motivated by the need to cope up with future traffic, QoS, and bandwidth demands. The aim of this thesis was to gain insight into the improved performance of the VRQ switch based on its capability to reduce cell loss and delay using the principle of virtual routing and queuing.

The ATM switching technology has dominated the industry because of the advantages associated with the use of fixed sized cells. Various switch architectures are available at present and this thesis attempts to compare the performance of the VRQ switch with those proposed earlier based on their buffering stratergy. The results suggest that while being principally an input buffered switch, the VRQ performs reasonably well and its performance is comparable to that of an output buffered switch. This improvement in the performance is achieved by eliminating head of line blocking completely. One of the reasons for HOL blocking is the contention in the switching fabric. In the VRQ switch the switching fabric contention is eliminated completely by using dedicated backplane buses. These dedicated backplane buses are simple to implement in hardware but may require additional space. Here, and in all switching architectures, there is always a trade-off between complexity, cost and speed. Other switches available [66] in the market may be better than the VRQ in terms of the space they occupy for switching fabric but due to the complex speed-up technology they are much harder to implement in hardware. In addition these switches require a speed-up of N times for $N \times N$ switch, which is not possible practically.

One other main feature of the VRQ switch is its ability to support upto N read operations per clock cycle. This is also achieved by the use of backplane buses. Each input buffer has sub-buffers connected to their respective backplane buses. During read operation, depending on the pointer addresses, the input buffer can send the cells to their respective destinations. This multi-ported buffer was designed and implemented using VHDL [89].

The simulation of switches shows that the characteristics of traffic can have a significant influence on the performance of a switch. It is shown that when driven with Pareto distributed traffic, switches generally exhibit higher cell loss and delay. This is due to the bursty nature of the Pareto traffic. The differences in cell loss and delay are not obvious when the simulation is performed for small time scales but become prominent as the simulation time scale is increased. These analysis also suggest that it is important to use realistic traffic data to gain insight into the performance of a switch as it may operate in a real world. Unrealistic traffic data may give little or no cell loss and delay but does not exhibit the actual performance of the switch.

6.2 Suggested Future Work

In this thesis an attempt has been made to generate the traffic and simulate the switches using C programs. The code for the generation of traffic data and switch models were written manually to get a better understanding of the working of these models. Although the simulations presented in this study have given a fair insight into the performance of the VRQ switch still more work can be done. Some suggestions for further work are outlined below.

- The switches can be simulated using OPNET software tool and the results can be compared to those obtained in this thesis.
- The VLSI implementation of the multi-ported buffer to get an ASIC or FPGA design can be done.
- Hardware design and synthesis of the output port with one or more queues,

depending upon the scheduling algorithm, can be obtained.

- Design and development of an efficient scheduling algorithm for the switch is also recommended.
- Finally, the VLSI implementation of the complete VRQ switch, to obtain an ATM chip, is also suggested.

Bibliography

[1] R.Y.Awdeh and H.T.Mouftah, "Survey of ATM switch architectures", Computer networks and ISDN systems, Vol. 27, 1995, pp 1567-1613.

[2] A.Pattavina, "Non blocking architectures for ATM switching", IEEE communication magazine, Feb 1993, pp 38-48.

[3] Y.Oie, T.Suda and M.Murata and H.Miyhara, "Survey of switching techniques in high speed networks and their performance", International Journal on Satellite Communications, vol. 9, 1991, pp 285-303.

[4] J.Turner and N.Yamanaka, "Architectural choices in large scale ATM switches", IEICE Transaction on Communications, vol. E81-B, No. 2, Feb 1998.

[5] M.Listani and A.Roveri, "Switching structures for ATM", Computer Communication, vol. 12, No. 6 pp349-358 dec1989.

[6] M.G.Hluchyj, and M.J.Karol, "Queuing in high performance packet switching", IEEE Transaction on Communications, Vol. 6, No. 9, pp 1587-1597, Dec 1988.

[7] M.J.Karol, M.G.Hluchyj, and S.P. Morgan, "Input versus output queuing on a space division packet switching", IEEE Transaction on Communications, vol. 35, No. 12, pp 1347-1356, Dec 1988.

[8] William stallings, "Data and Computer Communications", Sixth addition, Prentice hall, ISBN. 0-13-084370-9.

[9] William stallings, "High Speed Networks", Prentice hall, ISBN. 0-13-525965-7.

[10] Andrew S.Tanenbaum, "Computer Networks", third addition, Prentice hall, ISBN. 0-13-349945-6

[11] Fayez Elguibaly, "Computer Communications Networks", second addition, Northstar digital design, ISBN. 0-9686116-2-1.

[12] M.Listani and A.Roveri, "Switching structures for ATM", Computer Communication, Vol. 12. No. 6 pp. 349-358 dec1989

[13] P.Newman, "ATM technology for corporate networks", IEEE Communications Magazine, pp 90-101, Apr 1992.

[14] Y.Oie, T.Suda, M.Murata and H.Hiyashara, "Survey of switching techniques in high speed networks and their performance", Int. J. Satellite Communications, Vol. 9, pp. 285-303,1991.

[15] X.Chen , "A survey of multistage interconnection networks for fast packet switches", International Journal on Analog and Digital Communications Systems, Vol. 4, pp33-59,1991

[16] X.Chen, and J.F.Hays, "Call scheduling in multicasting packet switching", Proceedings of IEEE ICC'92, pp. 895-899, 1992.

[17] A.Pattavina, "Non blocking architectures for ATM switching", IEEE Communications Magazine, No. 2, pp 38-48, feb 1993

[18] H.Ahmadi, and W.E.Denzel, "A survey of modern high performance switching techniques", IEEE Journal on Selected Areas Communication, Vol. 7, No. 7, pp. 1091-1103

[19] R.Y.Awdeh and H.T.Mouftah, "Broadband Packet switch architectures", proceedings of Photonics'93, 3rd IEEE International Workshop on Photonic Networks, Components and Applications, Atlanta, GA, pp. 183-188, Sep 93.

[20] R.Y.Awdeh and H.T.Mouftah, " A contention resolution algorithm for input buffered batcher-banyan networks", International Journal on Communications, Vol. 7, No. 1, Jan 1994, pp. 33-38.

[21] H.F.Badran and H.T.Mouftah, "Head of line arbitration in ATM switches with input-output buffering and backpressure control", proceedings of IEEE GLOBECOM'91, phoenix, AZ, Dec. 1991, pp. 347-351.

[22] H.F.Badran and H.T.Mouftah, "Fairness for broadband integrated switch architectures under backpressure mechanism", proceedings of IEEE ICC'91, Denver, CO, June 1991, pp. 1033 - 1037.

[23] J.S.Chen, and T.E.Stern, "Throughput analysis optimal buffer allocation and traffic imbalance study of generic non-blocking packet switch", IEEE Journal on Selected Areas of Communications, Vol. 9, No. 3, Apr. 1991, pp. 439 - 449.

[24] M.G.Hluchyj, and M.J.Karol, "Queuing in high performance packet switching", IEEE Transaction on Communications, Vol. 6, No. 9, pp. 1587-1597, Dec. 1988.

[25] M.J.Karol, M.G.Hluchyj, and S.P. Morgan, "Input versus output queuing on a space division packet switching", IEEE Transaction on Communications, Vol. 35, No. 12, pp. 1347-1356, Dec. 1988

[26] J.Peterson, "Throughput limitations by head of line blocking", proceedings ITC-13, pp. 659-663, 1991.

[27] A.Pattavina, "Switching theory: Architecture and performance in broadband ATM networks", ISBN. 0-471-96338-0

[28] S.Q.Li, "Performance of trunk grouping in packet switch design", Proceedings of IEEE GLOBECOM'90, Miami, Fl, pp. 688-693, Apr. 1991.

[29] S.C.Liew, "Performance of input buffered and output buffered ATM switches

under bursty traffic", Proceedings of IEEE GLOBECOM'90, San Diego, CA, pp. 1919-1925, Dec. 1990.

[30] H.Obara, and T.Yasushi, "An efficient contention resolution algorithm for input queuing ATM cross connect switches", International Journal on Digital and Analog Cabled Systems. Vol. 2, No. 4, pp. 261 - 267, Dec. 1989.

[31] M.J.Karol, K.Y.Eng, and H.Obara, " Improving the performance of input queued ATM packet switches", Proceedings of IEEE INFOCOM'92, Florence, Italy, pp. 110-115, May 1992.

[32] H.Matsunaga, and H.Uematsu, "A 1.5 Gb/s 8×8switch using a time-reservation algorithm", IEEE Journal on Selected Areas of Communications, Vol. 9, No. 8, pp. 1308-1317, Oct. 1991.

[33] M.K.Ali, and M.Youssefi, "The performance of an input access scheme in high speed packet switch", Proceedings of IEEE INFOCOM'91, Miami, FL, pp. 454-461, Apr. 1991.

[34] J.S.Chen, and R.Guerin, "Input queuing and internally non blocking packet switch with two priority classes", Proceedings of IEEE INFOCOM'89 , pp. 529-537, 1989

[35] Nick.Mckeown, Balaji.Prabhakar, and Mingyan Zhu, "Matching output queuing with combined input and output queuing", Proceedings of 35th annual allerton conference on communications, control and computing. Monticello, Illinois, Oct. 1997.

[36] A.Sabaa, F. Elguibaly, and D.Shpak, "Design and modelling of a non blocking input buffer ATM switch", Canadian Journal of Electrical and Computer Engineering, vol. 22, No. 3, 1997.

[37] A.Hung, G.Kesidis, and N.Mckeown, "ATM Input buffered switches with guaranteed rate property", Proceedings of IEEE ISCC'98, Athens, Jul. 1998, pp. 331-335.

[38] G.Thomas and J.Man, "Improved windowing rule for input buffered packet switches", Electronics Letters, Vol. 29, No. 4, pp. 393-395, Feb. 1991.

[39] Y.S.Yeh, M.G.Hluchyj, A.S.Acampora, "The knockout switch : a simple modular architecture for high performance packet switching". IEEE Journal of Selected Areas in Communication, Vol. 5, No. 8, Oct. 1987, pp. 1274-1283.

[40] H.Ahmadi, W.E.Denzel, C.A.murphy, E.Port, "A high performance switch fabric for integrated circuit and packet switching", Proc of INFOCOM'88, New Orleans, La, Mar. 1988, pp. 9-18.

[41] Hiroshi Suzuki, Hiroshi Nagano and Toshio Suzuki, "Output buffer switch architecture for asynchronous transfer mode", IEEE 1989, pp. 99-103.

[42] H.J.Chao, B.S.choe, "Design and analysis of a large scale multicast output buffered ATM switch", IEE/ACM Transactions on Networking, Vol. 3, No. 2, Apr. 95, pp. 126-138.

[43] Jin Li, "An output-shared buffer ATM switch", IEEE'96.

[44] Keun.Bae.Kim and Hyup.Jong.Kim, "Back pressure buffering scheme to improve the cell loss property on the output buffered ATM switch." , IEEE'96

[45] G.Kesidis and N.Mckeown, "Output buffer ATM packet switching for integrated service communication networks", ICC'97, Montreal, Canada, Aug. 1997, pp. 1684-1688.

[46] Jian Ma and Peng .Zhang , "Output buffered ATM switch with frame discard", ICCT'98, Beijing, China, Oct 1998.

[47] Feihong Chen, Necdet Uzun and Ali N. Akansu, "A large scale Multicast ATM switch with input and output link sharing", Intended Symposium on high speed networks, New Jersey Center for Multimedia Research, New Jersey.

[48] www.net.com

[49] J.B.Evans, E.Duron, Y.Wang, "Analysis and implementation of a priority knockout switch." INFOCOM'93, pp. 1099-1106.

[50] J.P. Coudreuse, M.Servel, "Prelude: An asynchronous time division switched networks", IEEE International Conference on Communications, Seattle, Washington, June 1987,pp. 769-773

[51] H.Kuwahara, N.Endo, M.Ogino, T.Kozaki, "A shared buffer memory switch for an ATM exchange", IEEE Conference on Communications, Boston, MA, June 1989, pp. 118-122.

[52] T. Kozaki et al., "32 $\times$ 32 shared buffer type ATM switch VLSI's for B-ISDN's" IEEE Journal of Selected Areas of Communications, Vol. 9, Oct. 1991, pp 1239-1247.

[53] N.Endo, M.Ogino, T.Kozaki, H.Kuwahara "A shared buffer memory switch for an ATM exchange", IEEE Transactions on Communications, Vol. 41, Jan. 93, pp. 237-245.

[54] H. Saito, H.Yamanaka, H.Yamada, et al. , "Multicast function and its LSI implementation in a shared multibuffer ATM switch", IEEE INFOCOM, 1994, pp. 315-322.

[55] J.Lin, C.Wu, "A novel architecture for an ATM switch, IEEE International Conference on Computer Design, 1995, pp. 340-345.

[56] S.Kumar, D.Aggarwal, "On multicast support for shared memory based ATM switch architecture", IEEE Conference on Networks, Jan. 1996, pp. 34-39.

[57] H.Yamanaka, H. Saito, H.Kandoh, et.al, "Scalable shared buffering ATM switch

with a versatile searchable queue", IEEE Journal on Selected Areas of Communications, Vol. 15, May. 97, pp. 773-784

[58] M.Y.Hashemi, A.L. Gacia, "The single queue switch : a building block for switches with programmable scheduling", IEEE Journal of Selected Areas of Communications, Vol. 15, May. 97, pp. 785-794.

[59] K.Kumaran and D.Mitra, "Performance and fluid simulations of a novel shared buffer management system", Proceedings of IEEE INFOCOM'98, Mar. 1998, pp. 1449-1461.

[60] S.Krishnan, A.K.Choudhury and Fabio.M.Chiussi, "Dynamic partitioning: A mechanism for shared memory management", Proceedings of IEEE INFOCOM'98,.

[61] S.Kumar, D.Aggarwal, "A shared buffer direct acces switch architecture for ATM based networks", IEEE Conference on Communications, 1994, pp. 101-105.

[62] T.E. Anderson et. al., "High speed switch scheduling for local areas network", ACM Transactions on Computer Systems, Vol. 11, No. 4, Nov. 1993, pp. 319-52.

[63] Tamir. Y and Frazier.G, "High performance multi queue buffers for VLSI communications switches", proceedings of 15th international symposium on Computer Architecture, June 1988, pp. 343-354.

[64] Nong Ge and Munir Hamdi, "On the provision of Quality of Service guarantees for input queued switches", IEEE Journal of Communications, Dec.2000, Vol. 38, No. 12, pp.62-69.

[65] I.Illadis and W.E.Denzel,"Analysis of packet switches with multiple input and output queuing", IEEE Transactions on Communications, May. 1993, Vol. 41, No. 5, pp. 731-40.

[66] N. Mckeown, V.Anatharam, J.Warland, "Achieving 100% throughput in an input queued switches", IEEE Infocom march'1996, Vol 1, pp 296-302 .

[67] W. Willinger, M.S. Taqqu, "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, April 1997.

[68] V. Paxson and S. Floyd, "Wide Area Traffic: The failure of Poisson Modeling", IEEE/ACM transactions on Networking, 3(3), pp. 226-244, June 1995.

[69] W. Leland, M. Taqqu, W. Willianger, and D. wilson, "On the Self-Similar Nature of Ethernet traffic (extended version)", IEEE/ACM Transactions on Networking, Vol. 2, No. 4, pp. 316-336, August 1994.

[70] Mark and Azer, "Self similarity in WWW traffic Evidence and possible causes", IEEE/ACM transactions on Networking, Vol. 5, No. 6, pp. 835-846, Dec. 1997.

[71] Zafer and Tekinay, "On Multimedia Networks: Self similar traffic performance", IEEE Communications Magazine, Jan. 1999, pp. 48-52.

[72] Jain,S., and A. Routhier. "Packet trains: Measurement and a new model for computer network traffic", IEEE Journal on Selected Areas in Communications, Vol. 4, pp 986-995,1986.

[73] Beran, R., Sherman, M., S. Taqqu and W. Willinger, "Variable bit rate video traffic and long range dependence, IEEE/ACM Transactions on Networking", Vol. 2, pp. 1-15, 1994.

[74] G. Stix, "Domesticating cyberspace", Scientific American Special Issue: The Computer in the 21st Century, 1995, pp. 31-38

[75] A. Erramilli, O. Narayan and W. Willinger, "Experimental queuing analysis with long range dependent packet traffic", IEEE/ACM Transanctions on Networking, Vol. 4, No. 2, pp. 209-223,April 1996.

[76] W.E. Leland and D.V. Wilson, "High time resolution measurement and analysis of LAN traffic: Implementations for LAN Interconnections", proceedings of IEEE Infocom'91, pp. 1360-1366, Bal Harbour, Fl, 1991.

[77] Kihong Park, Gi Tae Kim, and Mark E Crovella, "On the relationship between file sizes, transport protocols, and self-similar network traffic", in proceedings of the 4th International Conference on Network Protocols (ICNP'96), pp. 171 -180, Oct. 1996.

[78] K.K. Ramakrishnan and S.K. Tripathi, "A common framework for studying the performance of channel access protocols", proceedings on computer performance evaluation group(CPEUG'82), Oct. 1982, pp. 365-373.

[79] V.S. Frost and B. Malamed, "Traffic modelling for telecommunications network", IEEE Communications Magazine, Vol. 32, Mar. 1994.

[80] D.C.Feldmeier, "Emperical analysis of a token ring network", Mass. Inst. Technological Lab, Computer Science, Cambridge, MA, MIT/LCS/TM-254, Jan 1984.

[81] N.C.Mohanty," Buffer behaviour for batch arrivals and single output in satellite channel," proceeding of IEEE Telecommunication Conference, NTC'78, Birmingham, AL, Dec. 1978, pp. 40.4/1-40.4/3

[82] D. R. Cox, "Long Range Dependence: A Review", in statistics: proceedings of 50th anniversary conference, Iowa state, pp. 55-74, 1984.

[83] S. Rambaldi, and O.Pinazza, " An accurate fractional brownian motion generator", Physica A, Vol. 208, No. 1, pp. 21-30, July1'94.

[84] Ying Chen, Zhang Deng, and Carey L Williamson, "A Model for Self-Similar Ethernet LAN Traffic. SCSC 95, Ottawa, Ontario, Canada, July 1995.

[85] Ilkka Norros, "A storage model with self-similar input", Queuing System", Vol. 16, 1994.

[86] Nicolas D. Georganas, "Self-Similar Traffic in ATM Networks",2nd international workshop, IWACA 1994.

[87] Stephan Robert and Jean Yves Le Boudec, "Can Self-Similar Traffic be modeled by Markovian Processes", COST 242 Stockholm Meeting, May 1995.

[88] Sandeep Agarwal,"Design and Performance of a new ATM switch", M.A.Sc thesis, ECE dept., University of Victoria, 1998.

[89] Peter.J.Ashenden, "The Designer's guide to VHDL", Morgan Kaufmann Publishers, 1995, ISBN 1-55860-270-4.

Appendix A

VHDL code for the multi-ported input buffer of the VRQ switch

A.1 VHDL code for the cell recognizer of the multi-ported buffer

```
library ieee;
library std;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity add_address is

port(
vpi_out :in std_logic_vector(0 to 7);
output_port: in std_logic_vector(0 to 2);
cell_out: in std_logic_vector(0 to 50);
buffer_address:in std_logic_vector(0 to 7);
changed_cell_out: out std_logic_vector(0 to 50);
final_output_port: out std_logic_vector(0 to 2);
out_buffer_address:out std_logic_vector(0 to 7));

end add_address;

architecture behav of add_address is

begin

add_up:process(vpi_out,output_port,cell_out)

variable i: integer;
variable incoming_cell:std_logic_vector(0 to 50);
begin
```

```
--if clk='1' then
incoming_cell:=cell_out;
if vpi_out="ZZZZZZZZ" or output_port="ZZZ" then
    for i in 0 to 50 loop
incoming_cell(i):='Z';
end loop;
changed_cell_out<=incoming_cell;
final_output_port<="ZZZ";
else

incoming_cell(4 to 11):=vpi_out;
changed_cell_out<=incoming_cell;
final_output_port<=output_port;
end if;
out_buffer_address<=buffer_address;

--end if;  -- clk = 1 loop

end process add_up;

end;
```

A.2 VHDL code for the Lookup table of the multi-ported buffer

```
library ieee;
library std;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;



entity lookup_table is

port(vpi_in: in std_logic_vector(0 to 7);
unassigned_cell,table_routing: in std_logic;
cell_discard:in std_logic;
vpi_out :out std_logic_vector(0 to 7);
output_port: out std_logic_vector(0 to 2));

end lookup_table;


architecture behav of lookup_table is
```

```
begin

lookup:process(vpi_in,unassigned_cell,table_routing,cell_discard)

type addresses is array(1 to 4) of std_logic_vector(0 to 7);
constant possible_vpi_in: addresses :=("00000001","00000010","00000100","00001000");
constant corresponding_vpi_out: addresses :=("00000011","00000111","00001111","00011111");

variable i,matching_output_port: integer;

begin

--if clk='1' then
if unassigned_cell='1' or cell_discard='1' then
vpi_out<="ZZZZZZZZ";
output_port<="ZZZ";
else
matching_output_port:=0;
for i in possible_vpi_in'range loop
if vpi_in=possible_vpi_in(i) then
matching_output_port:=i;
end if;
end loop;

if matching_output_port=0 then
vpi_out<="ZZZZZZZZ";
output_port<="ZZZ";
else
vpi_out<=corresponding_vpi_out(matching_output_port);
output_port<=conv_std_logic_vector(matching_output_port,3);
end if;
end if;
--end if;  -- clk = 1 loop


end process lookup;

end;
```

A.3 VHDL code for the Translator of the multi-ported buffer

```
library ieee;
library std;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
```

```
use ieee.std_logic_unsigned.all;

entity add_address is

port(
vpi_out :in std_logic_vector(0 to 7);
output_port: in std_logic_vector(0 to 2);
cell_out: in std_logic_vector(0 to 50);
buffer_address:in std_logic_vector(0 to 7);
changed_cell_out: out std_logic_vector(0 to 50);
final_output_port: out std_logic_vector(0 to 2);
out_buffer_address:out std_logic_vector(0 to 7));

end add_address;

architecture behav of add_address is

begin

add_up:process(vpi_out,output_port,cell_out)

variable i: integer;
variable incoming_cell:std_logic_vector(0 to 50);
begin

--if clk='1' then
incoming_cell:=cell_out;
if vpi_out="ZZZZZZZZ" or output_port="ZZZ" then
    for i in 0 to 50 loop
incoming_cell(i):='Z';
end loop;
changed_cell_out<=incoming_cell;
final_output_port<="ZZZ";
else

incoming_cell(4 to 11):=vpi_out;
changed_cell_out<=incoming_cell;
final_output_port<=output_port;
end if;
out_buffer_address<=buffer_address;

--end if;  -- clk = 1 loop

end process add_up;

end;
```

Appendix B

C language code used for the simulations in this thesis

B.1 C language code for the generation of synthetic traffic data

B.1.1 C language code for the generation of Poisson distributed traffic

```
#include <math.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>


main()
{


FILE *iof,*out;

int i,j,k,seed,sum,rand_num,int_par_rand[200001];
int count_zero,count_total,rem,quot,no_of_zeros,total;
int m,n,no_of_port[9],get_1_or_0[9];

float par_rand[200001],a,b,num,max,min,temp,temp1;
float decide_random[9], average;
double uni_rand[200001];

/*      DEFINE the parameters of the Poisson distribution, 1/average no. of
  packets arriving per unit time (get this from poisson)
 */
```

```
a = 1/1000.0;

srand(97);
rand_num=10000;

/*      print the parameters */

        printf("Poisson's Lambda = %f\n",a);


/* generate uniformly dist. random numbers */

for(i=1;i<=rand_num;++i)
{
  num = rand();
  uni_rand[i]=num/32767;
          uni_rand[i]=uni_rand[i]*0.8 + 0.1;
  if(uni_rand[i]>0.9999999)uni_rand[i]=0.9999999;
/*
  if(uni_rand[i]>0.99999){
   num=rand();
   uni_rand[i]=num/32767;
          }
*/
}


/* use these to generate poisson dist. random numbers */

for(i=1; i<=rand_num; ++i)
{
  par_rand[i]=-log(1-uni_rand[i])/a;
}



/* find min and max. to see if everything is okay */

max = -999999999999.0;
        min =  999999999999.0;
        for(i=1; i<=rand_num; ++i)
        {
         if(par_rand[i]<min)min=par_rand[i];
         if(par_rand[i]>max)max=par_rand[i];
        }


        printf("\nMax of poisson distributed no.'s = %f\n",max);
```

```
        printf("Min of poisson distributed no.'s = %f\n",min);

/* converting poisson distributed numbers to integers */

  for(i=1; i<=rand_num; ++i)
{
 int_par_rand[i]=par_rand[i];
}


/* trying to sort and print the generate random numbers */
/*
n = rand_num;

        for(i=1;i<=n;++i){
         for(j=i;j<=n;++j){
          if( par_rand[i]>par_rand[j]){
           temp=par_rand[i];
           par_rand[i]=par_rand[j];
           par_rand[j]=temp;
          }
         }
         printf("sorted array is %f\n",par_rand[i]);
        }

*/


max = -9999999.0;
        min =  9999999.0;
        for(i=1; i<=rand_num; ++i)
        {
         if(int_par_rand[i]<min)min=int_par_rand[i];
         if(int_par_rand[i]>max)max=int_par_rand[i];
        }


        printf("\nMax of poisson distributed no.'s = %f\n",max);
        printf("Min of poisson distributed no.'s = %f\n",min);

/*
for(i=1; i<=15; ++i)
{
 printf("%2d -> %d\n",i,int_par_rand[i]);
}

*/

sum=0;
```

```
for(i=1; i<=rand_num; ++i)
{
 sum=sum+int_par_rand[i];
}

   printf("\nsum of numbers is:  %d\n",sum);

        average = sum/rand_num;

   printf("\nAverage no. of packets arriving per unit time = %8.2f\n\n",average);


/* counting off periods */

count_zero=0;
count_total=0;
for(i=1;i<=rand_num;++i)
{
 rem = int_par_rand[i]%8;
 no_of_zeros = 8-rem;
 count_zero = count_zero + no_of_zeros;

 quot = (int_par_rand[i]-rem)/8;
 if(rem==0){
  total = (quot)*8;
 }
 else{
  total = (quot+1)*8;
 }
 count_total = count_total + total;
 if(i%25000==0){
  printf("Total so far [%d] = %d\n",i,count_total/8);
 }
}

printf("\nno of zeros = %d\n",count_zero);
printf("total zeros and ones = %d\n",count_total);

temp = count_zero;
temp1 = count_total;
temp = (temp/temp1)*100.0;
printf("off period percentage = %f\n",temp);


/* now we know that without including any off periods the traffic
when is distributed on the 8 ports, because of the mod 8 thing,
results in 3.82 say 4% off periods, i.e, about 96% load, pretty
close to 100%, so lets do our first simulation with this 96% load
without adding any additional off periods.
```

```
first lets generate the 0 and 1's at all the 8 input ports, put them
in a file and we will use it later to simulate the switch

Also from above we know that we have a total of 5853744 0's and 1's

5853744/8 = 731718 time steps of 3.42 micro second each during which
the traffic would come
*/

/* exit(-1); */

out=fopen("actual_poisson_traffic_96.dat","w");

for(i=1;i<=rand_num;++i)
{

/*  printf("%d. poisson dist. random no. = %d\n",i,int_par_rand[i]);   */


 rem = int_par_rand[i]%8;
 no_of_zeros = 8-rem;

 quot = (int_par_rand[i]-rem)/8;
 total = (quot+1)*8;

 for(j=1; j<=quot; ++j){
  for(n=1; n<=8; ++n){
   fprintf(out," 1");
  }
  fprintf(out,"\n");
 }

/*  okay, so far we have written down all the 1's, now we have rem no.
 of packets which need to be randomly distributed at the 8 ports

 To do this we will generate 8 uniformly dist. random numbers and
 the ports with the largest rem random numbers will get the packets.

 so first lets generate 8 uniformly dist. random numbers
*/

/*  printf("\nNo. of remaining packets = %d\n",rem);   */

 srand(567);
 for(k=1; k<=8; ++k){
  num = rand();
  decide_random[k]=num/32767;
  no_of_port[k]=k;
```

```
/*   printf("random no. %d = %f\n",k,decide_random[k]);   */
 }

        for(j=1;j<=8;++j){
         for(k=j;k<=8;++k){
          if(decide_random[j]<decide_random[k]){
           temp=decide_random[j];
           decide_random[j]=decide_random[k];
           decide_random[k]=temp;
           temp1=no_of_port[j];
           no_of_port[j]=no_of_port[k];
           no_of_port[k]=temp1;
          }
         }
 }

/*  okay so far we have sorted the random numbers and the ports they were
 associated with, first lets initialize all ports to zero
*/

 for(j=1; j<=8; ++j){
/*   printf("%d. rand. no. %f -> port no. %d\n",j,decide_random[j],no_of_port[j]);  */
  get_1_or_0[j]=0;
 }

/*  and then give the remaining packets (after the mod thing) only to those
 ports with highest random numbers in descending order
*/
 for(j=1; j<=rem; ++j){
  get_1_or_0[no_of_port[j]]=1;
 }

/*  and now we write these down */

 if(rem!=0){
  for(j=1; j<=8; ++j){
   fprintf(out," %d",get_1_or_0[j]);
  }
  fprintf(out,"\n");
 }

} /* this is i=1,rand_num loop */

fclose(out);

}   /* this is the main loop */
```

B.1.2 C language code for the generation of Pareto distributed traffic

```
#include <math.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>


main()
{


FILE *iof,*out;

int i,j,k,seed,sum,rand_num,int_par_rand[100001];
int count_zero,count_total,rem,quot,no_of_zeros,total;
int m,n,no_of_port[9],get_1_or_0[9];

float par_rand[100001],a,b,num,max,min,temp,temp1;
float decide_random[9];
double uni_rand[100001];

/*      DEFINE the parameters of the pareto distribution */

a = 350.0;
        b = 1.5;



srand(67);
rand_num=10000;

/*      print the parameters */


        printf("Pareto distribution's a = %f\n",a);
        printf("Pareto distribution's b = %f\n",b);


/* generate uniformly dist. random numbers */

for(i=1;i<=rand_num;++i)
{
  num = rand();
  uni_rand[i]=num/32767;
  if(uni_rand[i]>0.9999999999)uni_rand[i]=0.9999999999;
}
```

```
/* use these to generate pareto dist. random numbers */

for(i=1; i<=rand_num; ++i)
{
  par_rand[i]=a/pow((1-uni_rand[i]),1./b);
}



/* find min and max. to see if everything is okay */

max = -9999999.0;
        min =  9999999.0;
        for(i=1; i<=rand_num; ++i)
        {
         if(par_rand[i]<min)min=par_rand[i];
         if(par_rand[i]>max)max=par_rand[i];
        }


        printf("\nMax of pareto distributed no.'s = %f\n",max);
        printf("Min of pareto distributed no.'s = %f\n",min);

/* converting pareto distributed numbers to integers */

  for(i=1; i<=rand_num; ++i)
{
 int_par_rand[i]=par_rand[i];
}


max = -9999999.0;
        min =  9999999.0;
        for(i=1; i<=rand_num; ++i)
        {
         if(int_par_rand[i]<min)min=int_par_rand[i];
         if(int_par_rand[i]>max)max=int_par_rand[i];
        }


        printf("\nMax of pareto distributed no.'s = %f\n",max);
        printf("Min of pareto distributed no.'s = %f\n",min);

/*
for(i=1; i<=15; ++i)
{
```

```
 printf("%2d -> %d\n",i,int_par_rand[i]);
}

*/

sum=0;
for(i=1; i<=rand_num; ++i)
{
 sum=sum+int_par_rand[i];
}

   printf("\nsum of numbers is:  %d\n",sum);


/* counting off periods */

count_zero=0;
count_total=0;
for(i=1;i<=rand_num;++i)
{
 rem = int_par_rand[i]%8;
 no_of_zeros = 8-rem;
 count_zero = count_zero + no_of_zeros;

 quot = (int_par_rand[i]-rem)/8;
 if(rem==0){
  total = (quot)*8;
 }
 else{
  total = (quot+1)*8;
 }
 count_total = count_total + total;
 if(i%25000==0){
  printf("Total so far [%d] = %d\n",i,count_total/8);
 }
}

printf("\nno of zeros = %d\n",count_zero);
printf("total zeros and ones = %d\n",count_total);

temp = count_zero;
temp1 = count_total;
temp = (temp/temp1)*100.0;
printf("off period percentage = %f\n",temp);


/* now we know that without including any off periods the traffic
when is distributed on the 8 ports, because of the mod 8 thing,
results in 3.82 say 4% off periods, i.e, about 96% load, pretty
```

```
close to 100%, so lets do our first simulation with this 96% load
without adding any additional off periods.

first lets generate the 0 and 1's at all the 8 input ports, put them
in a file and we will use it later to simulate the switch

*/


out=fopen("traffic_at_ports_96.dat","w");

for(i=1;i<=rand_num;++i)
{

/*  printf("%d. pareto dist. random no. = %d\n",i,int_par_rand[i]);   */
 rem = int_par_rand[i]%8;
 no_of_zeros = 8-rem;

 quot = (int_par_rand[i]-rem)/8;
 total = (quot+1)*8;

 for(j=1; j<=quot; ++j){
  for(n=1; n<=8; ++n){
   fprintf(out," 1");
  }
  fprintf(out,"\n");
 }

/*  okay, so far we have written down all the 1's, now we have rem no.
 of packets which need to be randomly distributed at the 8 ports

 To do this we will generate 8 uniformly dist. random numbers and
 the ports with the largest rem random numbers will get the packets.

 so first lets generate 8 uniformly dist. random numbers
*/

/*  printf("\nNo. of remaining packets = %d\n",rem);   */

 srand(567);
 for(k=1; k<=8; ++k){
  num = rand();
  decide_random[k]=num/32767;
  no_of_port[k]=k;
/*   printf("random no. %d = %f\n",k,decide_random[k]);   */
 }

         for(j=1;j<=8;++j){
          for(k=j;k<=8;++k){
```

```
            if(decide_random[j]<decide_random[k]){
             temp=decide_random[j];
             decide_random[j]=decide_random[k];
             decide_random[k]=temp;
             temp1=no_of_port[j];
             no_of_port[j]=no_of_port[k];
             no_of_port[k]=temp1;
            }
           }
 }

/*  okay so far we have sorted the random numbers and the ports they were
 associated with, first lets initialize all ports to zero
*/

 for(j=1; j<=8; ++j){
/*   printf("%d. rand. no. %f -> port no. %d\n",j,decide_random[j],no_of_port[j]);  */
  get_1_or_0[j]=0;
 }

/*  and then give the remaining packets (after the mod thing) only to those
 ports with highest random numbers in descending order
*/
 for(j=1; j<=rem; ++j){
  get_1_or_0[no_of_port[j]]=1;
 }

/*  and now we write these down */

 if(rem!=0){
  for(j=1; j<=8; ++j){
   fprintf(out," %d",get_1_or_0[j]);
  }
  fprintf(out,"\n");
 }

} /* this is i=1,rand_num loop */

fclose(out);
```

```
}   /* this is the main loop */
```

B.2 C language code for the simulation of input buffered switch

```
#include <math.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

/* this program to read in the traffic we have already generated
poisson or pareto, and then simulate the input buffered switch
*/


main()
{

FILE *iof, *out;

int time_count,cell_at_port[9],cell_loss_count,total_cells_arriving_at_all_ports;
int i,j,k,l,m,n,input_buffer[9],output_queue[9],previous_output_queue[9];
int input_buffer_size,output_queue_size,decide_random[9];
int output_queue_location[202][10],max,cells_routed_outside;
int previous_cells_routed_outside,output_port_address_indicator[202][10];
int this_port_done[9],cell_out_count,cell_loss_occurence_at_queue[9];
int cell_loss_count_at_input_buffer,cell_loss_count_at_output_queue;
int seed;

int max_no_of_cells_waiting_on_any_port,arrival_time_array[205][10];
int no_of_cells_that_want_to_go_out[10],cell_going_out_from_input_port[10];
int no_of_cells_wanting_to_go_out,same_arrival,input_ports_cells_wanting_to_go_out[10];
int decide_port,same_arrival2,earliest_arrival_time,no_cells_arriving_at_earliest;
int input_port_of_cell_arriving_earliest[10],data_type;

float num,num1,useless,sum,balance,percent_lost,temp,temp1,sum1,sum2;
float delay_so_far_sum,delay;
float ave_buffer_length_sum,ave_queue_length_sum;
float ave_buffer_length,ave_queue_length,num2,num3,real_min;

float decide_ran[10],real_max;

char line[100],per[3],command[100],string[100];

strcpy(per,"%");
```

```
/*      the key here is to seed with a different number everytime
        so lets make seed a function of time
*/

        sprintf(command,"date > temp.txt");
        system(command);


        iof=fopen("temp.txt","r");
        for(j=1; j<=3; ++j){
         fscanf(iof,"%s",string);
        }
        fscanf(iof,"%2d%1s%2d%1s%2d",&num1,string,&num2,string,&num3);
        fclose(iof);
/*
        printf("num1 = %d\n",num1);
        printf("num2 = %d\n",num2);
        printf("num3 = %d\n",num3);
*/

        seed=3.18*num1+10.34*num2+pow(num3,2);

        srand(seed);



/* parameters of the model */

printf("\nStarting INPUT BUFFERED Switch simulation, please specify parameters ...\n");
printf("Input buffer size :");
scanf("%d",&input_buffer_size);

/* ask user which traffic data he or she wants to use */

again: printf("\nPlease choose the traffic data which you would like \n to drive your model with ...\n");
printf("     1. Pareto  distributed traffic 96%s load\n",per);
printf("     2. Poisson distributed traffic 96%s load\n",per);
printf("     3. Poisson distributed traffic 90%s load\n",per);
scanf("%d",&data_type);


if(data_type==1){
 iof = fopen("pareto_traffic_96.dat","r");
}
else if(data_type==2){
 iof = fopen("poisson_traffic_96.dat","r");
}
else if(data_type==3){
 iof = fopen("poisson_traffic_90.dat","r");
```

```
}
else{
 printf("\nPlease choose proper data type ......\n");
 goto again;
}



/* initialise input buffers queue sizes to 0 */

for(i=1; i<=8; ++i){
 input_buffer[i]=0;
}
cell_loss_count=0;
total_cells_arriving_at_all_ports=0;
cells_routed_outside=0;
cell_loss_count_at_input_buffer=0;
ave_buffer_length_sum=0;
ave_queue_length_sum=0;

printf("\nInput buffer size = %d\n",input_buffer_size);



printf("\nTime       ARRIVED    ROUTED      LOST (Loss Prob.)    IN-BUFFER   BALANCE  Ave. Delay\n");

        while(fgets(line,100,iof)!=NULL){
/*  ------------------------------------------------------------------ */
         ++time_count;
/*  printf("Current time step = %d\r",time_count);  */
/*  if(time_count==40){ exit(-1); }  */


/*
 first we route the cells out as per our two arrays
 1. Arrival Time Array, and
 2. Output port address indicator Array
*/


 if(time_count>1){
/*       ROUTING CELLS OUT  */
/*       0000000000000000000000000000000000000000000000000000000000000000000000000000   */
/*
  first we look at the Output port address indicator Array and see
  if any cells want to go to the same output port
*/
  for(i=1; i<=8; ++i){   /* we make a list of numbers of cells that want to go thru port i */
   no_of_cells_that_want_to_go_out[i]=0;
```

```
   for(j=1; j<=8; ++j){   /* that's all the input ports */
    if(output_port_address_indicator[1][j]==i){
     ++no_of_cells_that_want_to_go_out[i];
    }
   }
  }

/*   now lets see if more than 1 cell wants to go to a certain output port */
/*   printf("\n ..... NOW ROUTING CELLS OUT ..... \n");  */
/*
  for(i=1; i<=8; ++i){
   printf("Cells that want to go to output port %d => %d\n",i,no_of_cells_that_want_to_go_out[i]);
  }
  printf("\n");
*/
/*   getchar();  */

/*   now we go thru all the output ports and see if it has more than 1 cell
  wanting to go out thru it, if yes, we will decide which cell would we
  route out, and make a note of that, which we will use to update our
  two arrays
*/

  for(j=1; j<=8; ++j){ /* first we say nothing goes out of any input port */
   cell_going_out_from_input_port[j]=0;
  }


  for(i=1; i<=8; ++i){ /* these are the 8 output ports --------------------------- */

   if(no_of_cells_that_want_to_go_out[i]==1){ /* then we determine which input port cell that is */
    for(j=1; j<=8; ++j){
     if(output_port_address_indicator[1][j]==i){
      cell_going_out_from_input_port[j]=1;
/*         printf("\n..... 1 cell goes out from input port %d\n",j);  */
     }
    }
   }
   else if(no_of_cells_that_want_to_go_out[i]>1){
/*     this is real tricky, first we determine how many cells want to go out via this output port */
    no_of_cells_wanting_to_go_out=0;

    for(j=1; j<=8; ++j){
     input_ports_cells_wanting_to_go_out[j]=0;
    }

    for(j=1; j<=8; ++j){
     if(output_port_address_indicator[1][j]==i){
      ++no_of_cells_wanting_to_go_out;
```

```
      input_ports_cells_wanting_to_go_out[no_of_cells_wanting_to_go_out]=j;
     }
    }
/*     printf("\n........ %d cells want to go out via output port %d\n",no_of_cells_wanting_to_go_out,i); */
/*
    there are more than 1 cell wanting to go out via output port i
    to decide which one goes we, first find the earliest arrival time
    if there if only 1 cell which arrived at this time it goes out, if
    there are more cells we decide randomly
*/
           earliest_arrival_time=9999999999999999.0;
           for(j=1; j<=no_of_cells_wanting_to_go_out; ++j){
            if(arrival_time_array[1][input_ports_cells_wanting_to_go_out[j]]<earliest_arrival_time){
             earliest_arrival_time=arrival_time_array[1][input_ports_cells_wanting_to_go_out[j]];
            }
           }

/*     printf("    of these %d cells the earliest ones arrive at time = %d\n",no_of_cells_wanting_to_go_out,earliest.

/*         now we see how many cells arrive at this earliest arrival time, if only one
           then we route that out, if more than one than we have to decide randomly
*/
           no_cells_arriving_at_earliest=0;
           for(j=1; j<=8; ++j){
            input_port_of_cell_arriving_earliest[j]=0;
           }

           for(j=1; j<=no_of_cells_wanting_to_go_out; ++j){
            if(arrival_time_array[1][input_ports_cells_wanting_to_go_out[j]]==earliest_arrival_time){
             ++no_cells_arriving_at_earliest;
             input_port_of_cell_arriving_earliest[no_cells_arriving_at_earliest]=input_ports_cells_wanting_to_go_out
            }
           }

/*     printf("       no_cells_arriving_at_earliest = %d\n",no_cells_arriving_at_earliest); */

/*         now if no_cells_arriving_at_earliest=1 then we route the cell from corresponding
           input port, else decide randomly
*/
           if(no_cells_arriving_at_earliest==1){
/*      printf("       ONLY 1 CELL ARRIVES AT EARLIEST TIME\n");  */
            decide_port=input_port_of_cell_arriving_earliest[1];
            cell_going_out_from_input_port[decide_port]=1;
/*      printf("         and we decide that cell will go out from input port %d\n",decide_port); */
           }
           else{
/*      printf("       MORE THAN 1 CELL ARRIVES AT EARLIEST TIME\n");  */
            for(j=1; j<=no_cells_arriving_at_earliest; ++j){
             decide_ran[j] = 0;
```

```
            num=rand();
            decide_ran[j] = num/32767;
/*       printf("Random numbers corresponding to input port %d -> %f\n",input_port_of_cell_arriving_earliest[j],deci
           }
           real_max=-99.0;
           decide_port=0;
/*         now we find max of these random numbers */
           for(j=1; j<=no_cells_arriving_at_earliest; ++j){
            if(decide_ran[j]>real_max){
             real_max=decide_ran[j];
             decide_port=input_port_of_cell_arriving_earliest[j];
            }
           }
           cell_going_out_from_input_port[decide_port]=1;
/*      printf("         and we decide that cell will go out from input port %d\n",decide_port);  */

         } /* this is else if no_cells_arriving_at_earliest !=1 loop */

        }    /* else if more than 1 cells want to go out via this o/p port loop */

 } /* these are the i=1,8 output ports ------------------------------ */

/*
 printf("\n");
 for(i=1; i<=8; ++i){
  if(cell_going_out_from_input_port[i]==1){
   printf("FINAL DECISION -> 1 cell goes out of input port %d\n",i);
  }
 }
*/


/*        calculate the average delay suffered by cells which have been
          routed during this time step
*/

          cell_out_count=0;
          sum = 0;
          for(j=1; j<=8; ++j){
           if(cell_going_out_from_input_port[j]==1){
            ++cell_out_count;
/*          this means a cell is routed from here, and we know its arrival time,
            and we know the current time, so we can find the delay
*/
            sum = sum + time_count-arrival_time_array[1][j];
           }
          }
          sum = sum/cell_out_count;
          if(time_count>1){
```

```
          delay_so_far_sum = delay_so_far_sum + sum;
         }


/*
  we also have to reduce the no of cells in the input buffer
  well, lets do this first
  also up date the two controlling arrays
*/

  previous_cells_routed_outside=cells_routed_outside;
  for(i=1; i<=8; ++i){
   if(cell_going_out_from_input_port[i]==1){
    for(k=2; k<=input_buffer[i]; ++k){
     arrival_time_array[k-1][i]=arrival_time_array[k][i];
     output_port_address_indicator[k-1][i]=output_port_address_indicator[k][i];
    }
    arrival_time_array[input_buffer[i]][i]=0;
    output_port_address_indicator[input_buffer[i]][i]=0;
    --input_buffer[i];
    ++cells_routed_outside;
   }
  }

/*        print how many packets are in the buffer */

/*        ================================================= */
/*
          printf("\nPackets in input buffers AFTER routing ......\n");
          for(i=1; i<=8; ++i){
           printf(" %3d",input_buffer[i]);
          }
          printf("\n");
          printf("\n");

          printf("At time %d ->  %d (%d) cells are routed --------------- TOTAL cells routed = %d\n",time_count,cell
          printf("Average delay suffered by these cells = %f\n",sum);

*/
/*        ================================================= */

/*        getchar();  */

          max_no_of_cells_waiting_on_any_port=-999;
          for(i=1; i<=8; ++i){
           if(input_buffer[i]>max_no_of_cells_waiting_on_any_port){
            max_no_of_cells_waiting_on_any_port=input_buffer[i];
           }
          }
```

```
/*   printf("\n AFTER ROUTING max_no_of_cells_waiting_on_any_port = %d\n",max_no_of_cells_waiting_on_any_port); */

/*        now lets write Arrival Time array  */
/*
          printf("\nArrival Time Array ..... AFTER ROUTING \n");
          for(j=1; j<=max_no_of_cells_waiting_on_any_port; ++j){
           for(i=1; i<=8; ++i){
            printf("%3d ",arrival_time_array[j][i]);
           }
           printf("\n");
          }
*/
/*        now lets write output_port_address_indicator array  */
/*
          printf("\nOutput port address indicator Array ..... AFTER ROUTING\n");
          for(j=1; j<=max_no_of_cells_waiting_on_any_port; ++j){
           for(i=1; i<=8; ++i){
            printf("%3d ",output_port_address_indicator[j][i]);
           }
           printf("\n");
          }
*/
/*        getchar();  */


/*       0000000000000000000000000000000000000000000000000000000000000000000000000000    */
 } /* this is if time > 1 loop */


/*  printf("\n++++++++++++++ Time = %d ++++++++++++++++++++++++++++++++++\n",time_count);  */

/*  read the arriving cells from the traffic file */

 sscanf(line,"%d %d %d %d %d %d %d",&cell_at_port[1],&cell_at_port[2],&cell_at_port[3],&cell_at_port[4],&cell_at_

/*  make load 100% to see what happens */
/*
 for(i=1; i<=8; ++i){
  cell_at_port[i]=1;
 }
*/

/*  ================================================== */

/*  printf("CELLS ARRVING %8d. ",time_count);  */
```

```
 for(i=1; i<=8; ++i){
/*   printf(" %d",cell_at_port[i]);   */
  if(cell_at_port[i]==1){
   ++total_cells_arriving_at_all_ports;
  }
   }
/*
 printf("\n");
 printf("Total cells arrived so far = %d\n",total_cells_arriving_at_all_ports);
*/
/*        getchar();  */


/*  ================================================== */



/*  okay now we know whether a cell is coming into each of the 8 input ports
 or not, now we update the input_buffers
*/

 for(i=1; i<=8; ++i){
  if(input_buffer[i]<input_buffer_size&&cell_at_port[i]==1){
   ++input_buffer[i];
  }
  else if(input_buffer[i]>=input_buffer_size&&cell_at_port[i]==1){
   ++cell_loss_count;
   ++cell_loss_count_at_input_buffer;
   cell_at_port[i]=0;
/*    printf("CELL LOSS INPUT BUFFER FULL ..... at input port %d, Total cells lost = %d\n",i,cell_loss_count);  */
/*    getchar(); */
  }
   }


/*  print how many packets are in the buffer */

/*  ================================================== */
/*
 printf("\nPackets in input buffers ......\n");
 for(i=1; i<=8; ++i){
  printf(" %3d",input_buffer[i]);
 }
 printf("\n");

*/
/*        getchar();  */

/*  ================================================== */
```

```
/*  now we generate random numbers between 1 and 8 for each port, and accordingly
 decide to which output port the cell arriving at a certain input port
 is destined
*/
 num1 = 0.0;
 for(i=1; i<=8; ++i){
  decide_random[i] = 0; /* say that cell is going no where */
  num=rand();
  num1 = ((num/32767)*8.0);
  num1 = num1 + 1.0;
  if(cell_at_port[i]==1){  /* if a cell has arrived at a port */
   decide_random[i] = num1;
   if(decide_random[i]>8){
    decide_random[i]=8;
   }
  }
/*   printf("Cell at port %d is going to output port ----> %d (%f)\n",i,decide_random[i],num/32767);  */
 }


/*        getchar();  */

/*  to keep track of ave delay we have to maintain an array which will
 have the same number of non-zero numbers as the cells in the input
 buffers. In our case say it will be an array of 8 rows (corresponding
 to 8 input ports) and max. 200 length (max. length of input buffer).
 The idea is to keep the non-zero numbers in array show the time
 at which the cells arrive, one after the other. As the cells
 will be routed out the array will move forward, i.e., columns
 replaced. Using the time of arrival of a cell and its time
 of departure we can calculate the cell delay for every time
 instant.

 also find the max no of cells waiting on any port, we need this
 for writing this array and seeing if everything is okay

 we also have to make another array which will keep track of
 which output port a particular cell wants to go, this array
 will also have to be updated through time.

*/

 max_no_of_cells_waiting_on_any_port=-999;
 for(i=1; i<=8; ++i){
  if(input_buffer[i]>max_no_of_cells_waiting_on_any_port){
   max_no_of_cells_waiting_on_any_port=input_buffer[i];
  }
```

```
 }

/*  printf("\nmax_no_of_cells_waiting_on_any_port = %d\n",max_no_of_cells_waiting_on_any_port); */

/*
 Keep in my mind that we only have to update the
 last value in every row. This can be tricky.
*/

 for(i=1; i<=8; ++i){
  k=input_buffer[i];
  if(cell_at_port[i]==1){
   arrival_time_array[k][i]=time_count;
   output_port_address_indicator[k][i]=decide_random[i];
  }
 }

/*  now lets write Arrival Time array  */
/*
 printf("\nArrival Time Array ..............\n");
 for(j=1; j<=max_no_of_cells_waiting_on_any_port; ++j){
  for(i=1; i<=8; ++i){
   printf("%3d ",arrival_time_array[j][i]);
  }
  printf("\n");
 }
*/

/*  now lets write output_port_address_indicator array  */
/*
 printf("\nOutput port address indicator Array ..............\n");
 for(j=1; j<=max_no_of_cells_waiting_on_any_port; ++j){
  for(i=1; i<=8; ++i){
   printf("%3d ",output_port_address_indicator[j][i]);
  }
  printf("\n");
 }
*/

/*        getchar();  */



         sum = 0;
         for(i=1; i<=8; ++i){
          sum = sum + input_buffer[i];
         }

         balance=total_cells_arriving_at_all_ports-cells_routed_outside-cell_loss_count-sum;
```

```
        temp=cell_loss_count;
        temp1 = total_cells_arriving_at_all_ports;

/*   okay this not percent any more, this is now loss probability */
        percent_lost=(temp/temp1);

        delay=delay_so_far_sum/time_count;



/*
 if(time_count%50000==0){
  printf("%8d. %8d  %8d  %8d (%8.6e)  %8.0f  %8.0f   %8.3f\n",time_count,total_cells_arriving_at_all_ports,cells_ro
 }
 else{
  printf("\nTime        ARRIVED    ROUTED      LOST (Loss Prob.)    IN-BUFFER   BALANCE  Ave. Delay\n");
  printf("%8d. %8d  %8d  %8d (%4.3f)  %8.0f  %8.0f   %8.3f\r",time_count,total_cells_arriving_at_all_ports,cells_rou
 }
*/

/*  ------------------------------------------------------------- */


 if(time_count==395389||time_count==726223||time_count==996233||time_count==1328050){
  printf("%8d. %8d  %8d  %8d (%8.6e)  %8.0f  %8.0f   %8.3f\n",time_count,total_cells_arriving_at_all_ports,cells_rou
 }


/*  getchar();    */
       } /* this is the while loop for the cells coming in */

fclose(iof);

/*
printf("%8d. %8d  %8d  %8d (%8.6e)  %8.0f  %8.0f   %8.3f\n",time_count,total_cells_arriving_at_all_ports,cells_route
*/

printf("\n\n");


printf("Total cell loss           = %d -> %e loss prob.\n",cell_loss_count,percent_lost);
temp=cell_loss_count_at_input_buffer;
temp1=total_cells_arriving_at_all_ports;
num=temp/temp1;
printf("Cell loss at Input Buffer = %d -> %e loss prob.\n",cell_loss_count_at_input_buffer,num);

printf("\n");
```

```
} /* this is the main loop */
```

B.3 C language code for the simulation of output buffered switch

```
#include <math.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

/* this program to read in the traffic we have already generated
poisson or pareto, and then simulate the OUTPUT buffered switch
*/


main()
{

FILE *iof, *out;

int time_count,cell_at_port[9],cell_loss_count,total_cells_arriving_at_all_ports;
int i,j,k,l,m,n,output_buffer[9],output_queue[9],previous_output_buffer[10];
int output_buffer_size,output_queue_size,decide_random[9];
int output_queue_location[202][10],max,cells_routed_outside;
int previous_cells_routed_outside,output_port_address_indicator[202][10];
int this_port_done[9],cell_out_count,cell_loss_occurence_at_queue[9];
int cell_loss_count_at_output_buffer,cell_loss_count_at_output_queue;
int seed;

int max_no_of_cells_waiting_on_any_port,arrival_time_array[205][10];
int no_of_cells_that_want_to_go_out[10],no_of_cells_going_to_output_buffer[10];
int no_of_cells_wanting_to_go_out,same_arrival,input_ports_cells_wanting_to_go_out[10];
int decide_port,same_arrival2,earliest_arrival_time,no_cells_arriving_at_earliest;
int input_port_of_cell_arriving_earliest[10],data_type;

float num,num1,useless,sum,balance,percent_lost,temp,temp1,sum1,sum2;
float delay_so_far_sum,delay;
float ave_buffer_length_sum,ave_queue_length_sum;
float ave_buffer_length,ave_queue_length,num2,num3,real_min;

float decide_ran[10],real_max;

char line[100],per[3],command[100],string[100];

strcpy(per,"%");
```

```
/*      the key here is to seed with a different number everytime
        so lets make seed a function of time
*/

        sprintf(command,"date > temp.txt");
        system(command);


        iof=fopen("temp.txt","r");
        for(j=1; j<=3; ++j){
         fscanf(iof,"%s",string);
        }
        fscanf(iof,"%2d%1s%2d%1s%2d",&num1,string,&num2,string,&num3);
        fclose(iof);
/*
        printf("num1 = %d\n",num1);
        printf("num2 = %d\n",num2);
        printf("num3 = %d\n",num3);
*/

        seed=3.18*num1+10.34*num2+pow(num3,2);

        srand(seed);



/* parameters of the model */

printf("\nStarting OUTPUT BUFFERED Switch simulation, please specify parameters ...\n");
printf("Output buffer size (max. 200):");
scanf("%d",&output_buffer_size);

/* ask user which traffic data he or she wants to use */

again: printf("\nPlease choose the traffic data which you would like \n to drive your model with ...\n");
printf("     1. Pareto  distributed traffic 96%s load\n",per);
printf("     2. Poisson distributed traffic 96%s load\n",per);
printf("     3. Poisson distributed traffic 90%s load\n",per);
scanf("%d",&data_type);


if(data_type==1){
 iof = fopen("pareto_traffic_96.dat","r");
}
else if(data_type==2){
 iof = fopen("poisson_traffic_96.dat","r");
}
else if(data_type==3){
```

```
 iof = fopen("poisson_traffic_90.dat","r");
}
else{
 printf("\nPlease choose proper data type ......\n");
 goto again;
}



/* initialise output buffers queue sizes to 0 */

for(i=1; i<=8; ++i){
 output_buffer[i]=0;
}
cell_loss_count=0;
total_cells_arriving_at_all_ports=0;
cells_routed_outside=0;
cell_loss_count_at_output_buffer=0;
ave_buffer_length_sum=0;
ave_queue_length_sum=0;

printf("\nOutput buffer size = %d\n",output_buffer_size);



printf("\nTime       ARRIVED    ROUTED      LOST (Loss Prob.)    IN-BUFFER   BALANCE  Ave. Delay\n");

        while(fgets(line,100,iof)!=NULL){
/*  ------------------------------------------------------------- */
         ++time_count;
/*  printf("Current time step = %d\r",time_count);  */
/*  if(time_count==40){ exit(-1); }  */


/*
 first we route the cells out as per our two arrays
 1. Arrival Time Array, and
 2. Output port address indicator Array
*/


 if(time_count>1){
/*        ROUTING CELLS OUT  */
/*        0000000000000000000000000000000000000000000000000000000000000000000000000000   */

/*        calculate the average delay suffered by cells which will be
          routed during this time step
*/
```

```
          cell_out_count=0;
          sum = 0;
          for(j=1; j<=8; ++j){
   if(arrival_time_array[1][j]>0){ /* ie if there is cell in buffer */
            ++cell_out_count;
/*          this means a cell is routed from here, and we know its arrival time,
            and we know the current time, so we can find the delay
*/
            sum = sum + time_count-arrival_time_array[1][j];
           }
          }
          sum = sum/cell_out_count;
          if(time_count>1){
           delay_so_far_sum = delay_so_far_sum + sum;
          }


/*   this should be too complicated this time, all we have to do is look
  at the arrival time array and if there is a non zero value in the
  first line of array, we route those cells out, and move the array
  foward in time
*/

  previous_cells_routed_outside=cells_routed_outside;
  for(j=1; j<=8; ++j){
   if(arrival_time_array[1][j]>0){ /* ie if there is cell in buffer */
     ++cells_routed_outside;
             for(k=2; k<=output_buffer[j]; ++k){
              arrival_time_array[k-1][j]=arrival_time_array[k][j];
             }
             arrival_time_array[output_buffer[j]][j]=0;
     --output_buffer[j];
   }
  }

/*
          printf("At time %d ->  %d (%d) cells are routed ---------------- TOTAL cells routed = %d\n",time_count,cells
          printf("Average delay suffered by these cells = %f\n",sum);

  getchar();
*/


/*        print how many packets are in the buffer */

/*        ================================================== */

/*
          printf("\nPackets in output buffers AFTER routing ......\n");
```

```
          for(i=1; i<=8; ++i){
           printf(" %3d",output_buffer[i]);
          }
          printf("\n\n");
*/

          max_no_of_cells_waiting_on_any_port=-999;
          for(j=1; j<=8; ++j){
           if(output_buffer[j]>max_no_of_cells_waiting_on_any_port){
            max_no_of_cells_waiting_on_any_port=output_buffer[j];
           }
          }

/*   printf("\n AFTER ROUTING max_no_of_cells_waiting_on_any_port = %d\n",max_no_of_cells_waiting_on_any_port); */

/*        now lets write Arrival Time array  */
/*
          printf("\nArrival Time Array ..... AFTER ROUTING \n");
          for(j=1; j<=max_no_of_cells_waiting_on_any_port; ++j){
           for(i=1; i<=8; ++i){
            printf("%3d ",arrival_time_array[j][i]);
           }
           printf("\n");
          }


  getchar();

*/


/*        0000000000000000000000000000000000000000000000000000000000000000000000000000   */
 } /* this is if time > 1 loop */



/*  printf("\n++++++++++++++ Time = %d ++++++++++++++++++++++++++++++++++++\n",time_count);  */

/*  read the arriving cells from the traffic file */

 sscanf(line,"%d %d %d %d %d %d %d %d",&cell_at_port[1],&cell_at_port[2],&cell_at_port[3],&cell_at_port[4],&cell_at_


/*  make load 100% to see what happens */
/*
 for(i=1; i<=8; ++i){
  cell_at_port[i]=1;
 }
*/
```

```
/*  ================================================ */

/*  printf("CELLS ARRVING %8d. ",time_count);   */
 for(i=1; i<=8; ++i){
/*   printf(" %d",cell_at_port[i]);   */
  if(cell_at_port[i]==1){
   ++total_cells_arriving_at_all_ports;
  }
   }

/*
 printf("\n");
 printf("Total cells arrived so far = %d\n",total_cells_arriving_at_all_ports);

 getchar();
*/

/*  ================================================ */



/*  first we determine which output port each of these cells
 wants to go to and we decide this randomly
*/

         num1 = 0.0;
         for(i=1; i<=8; ++i){
          decide_random[i] = 0;          /* say that cell is going no where */
          num=rand();
          num1 = ((num/32767)*8.0);
          num1 = num1 + 1.0;
          if(cell_at_port[i]==1){        /* if a cell has arrived at a port */
           decide_random[i] = num1;
           if(decide_random[i]>8){
            decide_random[i]=8;
           }
          }
/*   printf("Cell at port %d is going to output port ----> %d (%f)\n",i,decide_random[i],num/32767);  */
         }


/*  now we determine how many cells are going to each output port buffer
 once we know this we can update the no. of cells in the output port
 buffers, we will also drop cells if the output port buffers are full

 so lets first determine how many cells are going to each output
 port buffer
*/
```

```
        for(j=1; j<=8; ++j){  /* first we say nothing goes to any output port buffer*/
         no_of_cells_going_to_output_buffer[j]=0;
        }

 for(j=1; j<=8; ++j){ /* these are the 8 output ports */
  for(i=1; i<=8; ++i){ /* these are the 8 (or less) cells which just came */
   if(decide_random[i]==j){
    ++no_of_cells_going_to_output_buffer[j];
   }
  }
 }

/*  write down the no. of cells wanting to go to each output port or buffer */
/*
 for(j=1; j<=8; ++j){
  printf("No. of cells wanting to go to output buffer %d = %d\n",j,no_of_cells_going_to_output_buffer[j]);
 }

 getchar();
*/


/*  now we update the output buffers accordingly, if the max. output buffer size
 is exceeded we drop cells
*/


 for(j=1; j<=8; ++j){
  previous_output_buffer[j]=output_buffer[j];
 }

/*
        printf("\nNumber of cells in OUTPUT buffers for the PREVIOUS time step .......\n");
        for(j=1; j<=8; ++j){
         printf("    No. of cells in output buffer %d = %d\n",j,previous_output_buffer[j]);
        }
        printf("\n");
        getchar();
*/


        for(j=1; j<=8; ++j){ /* all 8 output buffers */
  output_buffer[j]=output_buffer[j]+no_of_cells_going_to_output_buffer[j];

  if(output_buffer[j]>output_buffer_size){
   cell_loss_count = cell_loss_count + (output_buffer[j]-output_buffer_size);
```

```
   cell_loss_count_at_output_buffer = cell_loss_count_at_output_buffer + (output_buffer[j]-output_buffer_size);
/*    printf("CELL LOSS OUTPUT BUFFER FULL ..... at output port %d, Total cells lost = %d\n",j,cell_loss_count); *

   output_buffer[j]=output_buffer_size;
  }

        }

/*  lets write the no. of cells in each output port buffer */

/*
 printf("\nNumber of cells in OUTPUT buffers AFTER new cells are added/dropped .............\n");
 for(j=1; j<=8; ++j){
  printf("    No. of cells in output buffer %d = %d\n",j,output_buffer[j]);
 }
 printf("\n");
 getchar();
*/


/*  now we update the time arrival array */

 for(j=1; j<=8; ++j){
  for(k=previous_output_buffer[j]+1; k<=output_buffer[j]; ++k){
   arrival_time_array[k][j]=time_count;
  }
 }


/*  find the max. no. of cells in the output buffer */

        max_no_of_cells_waiting_on_any_port=-999;
        for(j=1; j<=8; ++j){
         if(output_buffer[j]>max_no_of_cells_waiting_on_any_port){
          max_no_of_cells_waiting_on_any_port=output_buffer[j];
         }
        }

/*  printf("\nmax_no_of_cells_waiting_on_any_port = %d\n",max_no_of_cells_waiting_on_any_port);  */


/*  now we write the current arrival_time_array */
/*

        printf("\nArrival Time Array .............\n");
        for(j=1; j<=max_no_of_cells_waiting_on_any_port; ++j){
         for(i=1; i<=8; ++i){
          printf("%3d ",arrival_time_array[j][i]);
         }
```

```
         printf("\n");
        }


 getchar();

*/



        sum = 0;
        for(i=1; i<=8; ++i){
         sum = sum + output_buffer[i];
        }

        balance=total_cells_arriving_at_all_ports-cells_routed_outside-cell_loss_count-sum;
        temp=cell_loss_count;
        temp1 = total_cells_arriving_at_all_ports;

/*      okay this not percent any more, this is now loss probability    */

        percent_lost=(temp/temp1);

        delay=delay_so_far_sum/time_count;


/*
 if(time_count%50000==0){
  printf("%8d. %8d  %8d  %8d (%8.6e)  %8.0f  %8.0f   %8.3f\n",time_count,total_cells_arriving_at_all_ports,cells_rou
 }
 else{
  printf("\nTime       ARRIVED    ROUTED      LOST (Loss Prob.)    IN-BUFFER   BALANCE  Ave. Delay\n");
  printf("%8d. %8d  %8d  %8d (%4.3f)  %8.0f  %8.0f   %8.3f\r",time_count,total_cells_arriving_at_all_ports,cells_rou
 }
*/


/*  ------------------------------------------------------------- */


 if(time_count==395389||time_count==726223||time_count==996233||time_count==1328050){
  printf("%8d. %8d  %8d  %8d (%8.6e)  %8.0f  %8.0f   %8.3f\n",time_count,total_cells_arriving_at_all_ports,cells_rou
 }



/*  getchar();    */
       } /* this is the while loop for the cells coming in */
```

```
fclose(iof);

/*
printf("%8d. %8d  %8d  %8d (%8.6e)  %8.0f  %8.0f   %8.3f\n",time_count,total_cells_arriving_at_all_ports,cells_rout
*/

printf("\n\n");


printf("Total cell loss           = %d -> %e (%f percent) loss prob.\n",cell_loss_count,percent_lost,percent_lost*1
temp=cell_loss_count_at_output_buffer;
temp1=total_cells_arriving_at_all_ports;
num=temp/temp1;
printf("Cell loss at Output Buffer = %d -> %e (%f percent) loss prob.\n",cell_loss_count_at_output_buffer,num,num*1

printf("\n");



} /* this is the main loop */
```

B.4 C language code for the simulation of shared buffered switch

```
#include <math.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

/* this program to read in the traffic we have already generated
poisson or pareto, and then simulate the SHARED buffered switch
*/


main()
{

FILE *iof, *out;

int time_count,cell_at_port[9],cell_loss_count,total_cells_arriving_at_all_ports;
int i,j,k,l,m,n,output_buffer[9],output_queue[9],previous_output_buffer[10];
int output_buffer_size,output_queue_size,decide_random[9];
int output_queue_location[202][10],max,cells_routed_outside;
int previous_cells_routed_outside,output_port_address_indicator[202][10];
int this_port_done[9],cell_out_count,cell_loss_occurence_at_queue[9];
int cell_loss_count_at_output_buffer,cell_loss_count_at_output_queue;
```

```
int seed;

int max_no_of_cells_waiting_on_any_port,arrival_time_array[205][10];
int no_of_cells_that_want_to_go_out[10],no_of_cells_going_to_output_buffer[10];
int no_of_cells_wanting_to_go_out,same_arrival,input_ports_cells_wanting_to_go_out[10];
int decide_port,same_arrival2,earliest_arrival_time,no_cells_arriving_at_earliest;
int input_port_of_cell_arriving_earliest[10],data_type;
int place_of_found_cell,logical_shared_cells_added[10],total_sum;
int total_buffer_capacity;

int no_of_shared_buffer_cells,shared_cells_arrival[1601],shared_cells_output_port[1601];
int no_of_extra_cells,pre_no_of_shared_buffer_cells;

float num,num1,useless,sum,balance,percent_lost,temp,temp1,sum1,sum2;
float delay_so_far_sum,delay;
float ave_buffer_length_sum,ave_queue_length_sum;
float ave_buffer_length,ave_queue_length,num2,num3,real_min;

float decide_ran[10],real_max;

char line[100],per[3],command[100],string[100];

strcpy(per,"%");

/*      the key here is to seed with a different number everytime
        so lets make seed a function of time
*/

        sprintf(command,"date > temp.txt");
        system(command);


        iof=fopen("temp.txt","r");
        for(j=1; j<=3; ++j){
         fscanf(iof,"%s",string);
        }
        fscanf(iof,"%2d%1s%2d%1s%2d",&num1,string,&num2,string,&num3);
        fclose(iof);
/*
        printf("num1 = %d\n",num1);
        printf("num2 = %d\n",num2);
        printf("num3 = %d\n",num3);
*/

        seed=3.18*num1+10.34*num2+pow(num3,2);

        srand(seed);
```

```
/* parameters of the model */

printf("\nStarting SHARED BUFFERED Switch simulation, please specify parameters ...\n");
printf("Shared buffer size (max. 200):");
scanf("%d",&output_buffer_size);

/* ask user which traffic data he or she wants to use */

again: printf("\nPlease choose the traffic data which you would like \n to drive your model with ...\n");
printf("     1. Pareto  distributed traffic 96%s load\n",per);
printf("     2. Poisson distributed traffic 96%s load\n",per);
printf("     3. Poisson distributed traffic 90%s load\n",per);
scanf("%d",&data_type);


if(data_type==1){
 iof = fopen("pareto_traffic_96.dat","r");
}
else if(data_type==2){
 iof = fopen("poisson_traffic_96.dat","r");
}
else if(data_type==3){
 iof = fopen("poisson_traffic_90.dat","r");
}
else{
 printf("\nPlease choose proper data type ......\n");
 goto again;
}



/* initialise output buffers queue sizes to 0 */

for(i=1; i<=8; ++i){
 output_buffer[i]=0;
}
cell_loss_count=0;
total_cells_arriving_at_all_ports=0;
cells_routed_outside=0;
cell_loss_count_at_output_buffer=0;
ave_buffer_length_sum=0;
ave_queue_length_sum=0;
no_of_shared_buffer_cells=0;


total_buffer_capacity=8*output_buffer_size;

printf("\nShared buffer size = %d\n",output_buffer_size);
```

```
/* printf("\ntotal_buffer_capacity = %d\n",total_buffer_capacity); */


printf("\nTime       ARRIVED    ROUTED      LOST (Loss Prob.)    IN-BUFFER   BALANCE  Ave. Delay\n");

        while(fgets(line,100,iof)!=NULL){
/*  ------------------------------------------------------------- */
         ++time_count;
/*  printf("Current time step = %d\r",time_count);  */
/*  if(time_count==40){ exit(-1); }  */


/*
 first we route the cells out as per our two arrays
 1. Arrival Time Array, and
 2. Output port address indicator Array
*/


 if(time_count>1){
/*        ROUTING CELLS OUT  */
/*        0000000000000000000000000000000000000000000000000000000000000000000000   */

/*        calculate the average delay suffered by cells which will be
          routed during this time step
*/

          cell_out_count=0;
          sum = 0;
          for(j=1; j<=8; ++j){
   if(arrival_time_array[1][j]>0){ /* ie if there is cell in buffer */
            ++cell_out_count;
/*          this means a cell is routed from here, and we know its arrival time,
            and we know the current time, so we can find the delay
*/
            sum = sum + time_count-arrival_time_array[1][j];
           }
          }
          sum = sum/cell_out_count;
          if(time_count>1){
           delay_so_far_sum = delay_so_far_sum + sum;
          }


/*   this should be too complicated this time, all we have to do is look
  at the arrival time array and if there is a non zero value in the
  first line of array, we route those cells out, and move the array
  foward in time
```

```
*/

  previous_cells_routed_outside=cells_routed_outside;
  for(j=1; j<=8; ++j){
   if(arrival_time_array[1][j]>0){ /* ie if there is cell in buffer */
     ++cells_routed_outside;
             for(k=2; k<=output_buffer[j]; ++k){
              arrival_time_array[k-1][j]=arrival_time_array[k][j];
             }
             arrival_time_array[output_buffer[j]][j]=0;
     --output_buffer[j];
   }
  }

/*
          printf("\nAt time %d ->  %d (%d) cells are routed ---------------- TOTAL cells routed = %d\n",time_count,ce
          printf("Average delay suffered by these cells = %f\n",sum);

  getchar();
*/



/*        print how many packets are in the buffer */

/*        ================================================== */

/*
          printf("\nPackets in output buffers AFTER routing ......\n");
          for(i=1; i<=8; ++i){
           printf(" %3d",output_buffer[i]);
          }
          printf("\n\n");
*/

          max_no_of_cells_waiting_on_any_port=-999;
          for(j=1; j<=8; ++j){
           if(output_buffer[j]>max_no_of_cells_waiting_on_any_port){
            max_no_of_cells_waiting_on_any_port=output_buffer[j];
           }
          }

/*   printf("\n AFTER ROUTING max_no_of_cells_waiting_on_any_port = %d\n",max_no_of_cells_waiting_on_any_port); */

/*        now lets write Arrival Time array  */

/*
          printf("\nArrival Time Array ............. AFTER ROUTING \n");
          for(j=1; j<=max_no_of_cells_waiting_on_any_port; ++j){
```

```
          for(i=1; i<=8; ++i){
           printf("%3d ",arrival_time_array[j][i]);
          }
          printf("\n");
         }


  getchar();

*/


/*        0000000000000000000000000000000000000000000000000000000000000000000000000000    */
 } /* this is if time > 1 loop */



/*  printf("\n++++++++++++++ Time = %d ++++++++++++++++++++++++++++++++++\n",time_count);  */

/*  read the arriving cells from the traffic file */

 sscanf(line,"%d %d %d %d %d %d %d",&cell_at_port[1],&cell_at_port[2],&cell_at_port[3],&cell_at_port[4],&cell_at_


/*  make load 100% to see what happens */
/*
 for(i=1; i<=8; ++i){
  cell_at_port[i]=1;
 }
*/

/*  ================================================== */

/*  printf("CELLS ARRVING %8d. ",time_count);   */
 for(i=1; i<=8; ++i){
/*   printf(" %d",cell_at_port[i]);   */
  if(cell_at_port[i]==1){
   ++total_cells_arriving_at_all_ports;
  }
   }

/*
 printf("\n");
 printf("Total cells arrived so far = %d\n",total_cells_arriving_at_all_ports);

 getchar();
*/
```

```
/*  ================================================ */


/*  first we determine which output port each of these cells
 wants to go to and we decide this randomly
*/

        num1 = 0.0;
        for(i=1; i<=8; ++i){
         decide_random[i] = 0;          /* say that cell is going no where */
         num=rand();
         num1 = ((num/32767)*8.0);
         num1 = num1 + 1.0;
         if(cell_at_port[i]==1){        /* if a cell has arrived at a port */
          decide_random[i] = num1;
          if(decide_random[i]>8){
           decide_random[i]=8;
          }
         }
/*   printf("Cell at port %d is going to output port ----> %d (%f)\n",i,decide_random[i],num/32767);  */
        }


/*  now we determine how many cells are going to each output port buffer
 once we know this we can update the no. of cells in the output port
 buffers, we will also drop cells if the output port buffers are full

 so lets first determine how many cells are going to each output
 port buffer
*/



        for(j=1; j<=8; ++j){  /* first we say nothing goes to any output port buffer*/
         no_of_cells_going_to_output_buffer[j]=0;
        }

 for(j=1; j<=8; ++j){ /* these are the 8 output ports */
  for(i=1; i<=8; ++i){ /* these are the 8 (or less) cells which just came */
   if(decide_random[i]==j){
    ++no_of_cells_going_to_output_buffer[j];
   }
  }
 }

/*  write down the no. of cells wanting to go to each output port or buffer */

/*
```

```
 for(j=1; j<=8; ++j){
  printf("No. of cells wanting to go to output buffer %d = %d\n",j,no_of_cells_going_to_output_buffer[j]);
 }

 getchar();
*/


/*  now we update the output buffers accordingly, if the max. output buffer size
 is exceeded we drop cells
*/

 for(j=1; j<=8; ++j){
  previous_output_buffer[j]=output_buffer[j];
  logical_shared_cells_added[j]=0;
 }

/*
         printf("\nNumber of cells in OUTPUT buffers for the PREVIOUS time step .......\n");
         for(j=1; j<=8; ++j){
          printf("    No. of cells in output buffer %d = %d\n",j,previous_output_buffer[j]);
         }
         printf("\n");
         getchar();

*/



/*  we have to take care of what we have the shared buffer space as well,
 we can't exceed the total space
*/

 for(i=1; i<=8; ++i){
  for(k=1; k<=no_of_cells_going_to_output_buffer[i]; ++k){
   sum = 0;
   for(j=1; j<=8; ++j){
    sum = sum + output_buffer[j];
   }
   if((sum+no_of_shared_buffer_cells)<total_buffer_capacity){
    ++output_buffer[i];
   }
   else{
    ++cell_loss_count;
    ++cell_loss_count_at_output_buffer;
/*     printf("CELL LOSS OUTPUT BUFFER FULL ..... at output port %d, Total cells lost = %d\n",j,cell_loss_count);
   }
  }
```

```
 }


/*  calculate how many cells we have all in all in our output buffer memory
*/

 sum=0;
 for(i=1; i<=8; ++i){
  sum = sum+output_buffer[i];
 }

/*  now see if cell loss is occuring or not */

         for(j=1; j<=8; ++j){ /* all 8 output buffers */

  if(output_buffer[j]>output_buffer_size){

/*    printf("OUTPUT BUFFER EXCEEDS CAPACITY FOR buffer no. --------> %d \n",j); */

/*    here we implement the shared buffer strategy, if a given output buffer is full then
   we still may have space lying around, so that cell loss does not take place
*/
   no_of_extra_cells=(output_buffer[j]-output_buffer_size);

/*    printf("         no_of_extra_cells = %d\n",no_of_extra_cells); */

   for(i=1; i<=no_of_extra_cells; ++i){

    if( (sum+no_of_shared_buffer_cells)<total_buffer_capacity){ /* ie if we still have space lying around */
/*      printf("       BECAUSE WE HAVE SPACE ------- put these in shared space\n"); */
     ++no_of_shared_buffer_cells;
/*      printf("            No. of shared buffer cells = %d\n",no_of_shared_buffer_cells); */
     shared_cells_arrival[no_of_shared_buffer_cells]=time_count;
     shared_cells_output_port[no_of_shared_buffer_cells]=j;

/*
     printf("\n");
     for(k=1; k<=no_of_shared_buffer_cells; ++k){
       printf("Shared cell no. %d arrived at %d, and is going to %d output port.\n",k,shared_cells_arrival[k],shared
     }
     printf("\n");
*/
    }
    else{
/*      printf("       WE DON'T HAVE SPACE ------- \n"); */
     ++cell_loss_count;
     ++cell_loss_count_at_output_buffer;
/*      printf("CELL LOSS OUTPUT BUFFER FULL ..... at output port %d, Total cells lost = %d\n",j,cell_loss_count);
    }
```

```
    output_buffer[j]=output_buffer_size;

   } /* this is i=1,no_of_extra_cells loop */


  } /* if output_buffer[j]>output_buffer_size loop */
  else if(output_buffer[j]<output_buffer_size) { /* ie if we have space in the queues */

/*    printf(" WE STILL HAVE SPACE in buffer %d LETS SEE IF WE CAN MOVE CELLS AROUND ------------ \n",j); */

   pre_no_of_shared_buffer_cells=no_of_shared_buffer_cells;

   for(i=1; i<=pre_no_of_shared_buffer_cells; ++i){   /* we find to put any cells in this queue */

    place_of_found_cell=0;
    if(shared_cells_output_port[i]==j&&output_buffer[j]<output_buffer_size){
/*      printf("        YES shared cell no. %d (arrival %d) wants to go to output port %d\n",i,shared_cells_arrival[
     place_of_found_cell=i;
     --no_of_shared_buffer_cells;
/*      printf("            No. of shared buffer cells = %d\n",no_of_shared_buffer_cells); */
     ++output_buffer[j];
     logical_shared_cells_added[j]=1;
     arrival_time_array[output_buffer[j]][j]=shared_cells_arrival[i];
/*      printf("     updated arrival time array [%d][%d] = %d\n",output_buffer[j],j,shared_cells_arrival[i]); */

/*      don't forget to move the no_of_shared_buffer_cells arrays backwards */
     for(k=place_of_found_cell+1; k<=no_of_shared_buffer_cells+1; ++k){
      shared_cells_arrival[k-1]=shared_cells_arrival[k];
      shared_cells_output_port[k-1]=shared_cells_output_port[k];
     }
     shared_cells_arrival[no_of_shared_buffer_cells+1]=0;
     shared_cells_output_port[no_of_shared_buffer_cells+1]=0;

/*
     printf("\n");
     for(k=1; k<=no_of_shared_buffer_cells; ++k){
       printf("Shared cell no. %d arrived at %d, and is going to %d output port.\n",k,shared_cells_arrival[k],shared_
     }
     printf("\n");
*/

    }

   } /* this is i=1,no_of_shared_buffer_cells loop */

  } /* else if output_buffer[j]<output_buffer_size loop */

        } /* for j=1,8 output buffer loop */
```

```
 total_sum = sum + no_of_shared_buffer_cells;
 if(total_sum>total_buffer_capacity){
  printf("\n\nAt time ---------> %d\n",time_count);
  printf("Total no. of cells %6.0f + %d = %d is greater than buffer capacity\n\n",sum,no_of_shared_buffer_cells,tota
  getchar();
 }

/*  lets write the no. of cells in each output port buffer */

/*
 printf("\nNumber of cells in OUTPUT buffers AFTER new cells are added/dropped .............\n");
 for(j=1; j<=8; ++j){
  printf("    No. of cells in output buffer %d = %d\n",j,output_buffer[j]);
 }
 printf("\n");
 getchar();

*/


/*  now we update the time arrival array
*/

 for(j=1; j<=8; ++j){
  for(k=previous_output_buffer[j]+1; k<=output_buffer[j]; ++k){
   if(logical_shared_cells_added[j]==0){
    arrival_time_array[k][j]=time_count;
   }
  }
 }


/*  find the max. no. of cells in the output buffer */

         max_no_of_cells_waiting_on_any_port=-999;
         for(j=1; j<=8; ++j){
          if(output_buffer[j]>max_no_of_cells_waiting_on_any_port){
           max_no_of_cells_waiting_on_any_port=output_buffer[j];
          }
         }

/*  printf("\nmax_no_of_cells_waiting_on_any_port = %d\n",max_no_of_cells_waiting_on_any_port);    */


/*  now we write the current arrival_time_array */

/*
         printf("\nArrival Time Array .............\n");
```

```
        for(j=1; j<=max_no_of_cells_waiting_on_any_port; ++j){
         for(i=1; i<=8; ++i){
          printf("%3d ",arrival_time_array[j][i]);
         }
         printf("\n");
        }

 getchar();
*/



        sum = 0;
        for(i=1; i<=8; ++i){
         sum = sum + output_buffer[i];
        }

        balance=total_cells_arriving_at_all_ports-cells_routed_outside-cell_loss_count-sum-no_of_shared_buffer_cells
        temp=cell_loss_count;
        temp1 = total_cells_arriving_at_all_ports;

/*      okay this not percent any more, this is now loss probability    */

        percent_lost=(temp/temp1);

        delay=delay_so_far_sum/time_count;


/*
 if(time_count%50000==0){
  printf("%8d. %8d  %8d  %8d (%8.6e) %6.0f+%4d  %8.0f   %8.3f\n",time_count,total_cells_arriving_at_all_ports,cells_
 }
 else{
  printf("%8d. %8d  %8d  %8d (%8.6e) %6.0f+%4d  %8.0f   %8.3f\r",time_count,total_cells_arriving_at_all_ports,cells_
 }
*/

/*  ------------------------------------------------------------- */


 if(time_count==395389||time_count==726223||time_count==996233||time_count==1328050){
  printf("%8d. %8d  %8d  %8d (%8.6e) %6.0f+%4d  %8.0f   %8.3f\n",time_count,total_cells_arriving_at_all_ports,cells_
 }



/*  getchar();    */
       } /* this is the while loop for the cells coming in */
```

```
fclose(iof);

/*
printf("%8d. %8d  %8d  %8d (%8.6e)  %8.0f  %8.0f   %8.3f\n",time_count,total_cells_arriving_at_all_ports,cells_route
*/

printf("\n\n");


printf("Total cell loss           = %d -> %e loss prob. (%f percent)\n",cell_loss_count,percent_lost,percent_lost*10
temp=cell_loss_count_at_output_buffer;
temp1=total_cells_arriving_at_all_ports;
num=temp/temp1;
printf("Cell loss at Shared Buffer = %d -> %e loss prob. (%f percent)\n",cell_loss_count_at_output_buffer,num,num*10

printf("\n");



} /* this is the main loop */
```

B.5 C language code for the simulation of the VRQ switch

```
#include <math.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

/* this program to read in the traffic we have already generated
and then simulate the switch
*/


main()
{

FILE *iof, *out;

int time_count,cell_at_port[9],cell_loss_count,total_cells_arriving_at_all_ports;
int i,j,k,l,m,n,input_buffer[9],output_queue[9],previous_output_queue[9];
int input_buffer_size,output_queue_size,decide_random[9];
```

```
int output_queue_location[202][10],max,cells_routed_outside;
int previous_cells_routed_outside,output_queue_address_indicator[202][10];
int this_port_done[9],cell_out_count,cell_loss_occurence_at_queue[9];
int cell_loss_count_at_input_buffer,cell_loss_count_at_output_queue;
int cell_loss_at_particular_output_queue[9],seed,data_type;

float num,num1,useless,sum,balance,percent_lost,temp,temp1,sum1,sum2;
float delay_so_far_sum,delay;
float ave_buffer_length_sum,ave_queue_length_sum;
float ave_buffer_length,ave_queue_length,num2,num3;

char line[100],per[3],command[100],string[100];


/*      the key here is to seed with a different number everytime
        so lets make seed a function of time
*/

        sprintf(command,"date > temp.txt");
        system(command);


        iof=fopen("temp.txt","r");
        for(j=1; j<=3; ++j){
         fscanf(iof,"%s",string);
        }
        fscanf(iof,"%2d%1s%2d%1s%2d",&num1,string,&num2,string,&num3);
        fclose(iof);
/*
        printf("num1 = %d\n",num1);
        printf("num2 = %d\n",num2);
        printf("num3 = %d\n",num3);
*/

        seed=3.78*num1+10.34*num2+pow(num3,2);

        srand(seed);


/* parameters of the model */

printf("\nStarting VRQ Switch simulation, please specify parameters ...\n");
printf("Input buffer size :");
scanf("%d",&input_buffer_size);
again: printf("Output queue size :");
scanf("%d",&output_queue_size);

if(output_queue_size>200){
 printf(" .... please specify an output queue size less than 200.\n");
```

```
 goto again;
}


/*      ask user which traffic data he or she wants to use */

again2:  printf("\nPlease choose the traffic data which you would like \n to drive your model with ...\n");
        printf("     1. Pareto  distributed traffic 96%s load\n",per);
        printf("     2. Poisson distributed traffic 96%s load\n",per);
        printf("     3. Poisson distributed traffic 90%s load\n",per);
        scanf("%d",&data_type);


        if(data_type==1){
         iof = fopen("pareto_traffic_96.dat","r");
        }
        else if(data_type==2){
         iof = fopen("poisson_traffic_96.dat","r");
        }
        else if(data_type==3){
         iof = fopen("poisson_traffic_90.dat","r");
        }
        else{
         printf("\nPlease choose proper data type ......\n");
         goto again2;
        }



/* initialise input buffers and output queue sizes to 0 */

for(i=1; i<=8; ++i){
 input_buffer[i]=0;
 output_queue[i]=0;
 cell_loss_at_particular_output_queue[i]=0;
}
cell_loss_count=0;
total_cells_arriving_at_all_ports=0;
cells_routed_outside=0;
cell_loss_count_at_input_buffer=0;
ave_buffer_length_sum=0;
ave_queue_length_sum=0;

printf("\nInput buffer size = %d\n",input_buffer_size);
printf("Output queue size = %d\n",output_queue_size);


strcpy(per,"%");
```

```
printf("\nTime       ARRIVED    ROUTED      LOST (Loss Prob.)    IN-BUFFER   BALANCE  Ave. Delay\n");

        while(fgets(line,100,iof)!=NULL){
/*  ------------------------------------------------------------- */
         ++time_count;
/*  printf("Current time step = %d\r",time_count);  */
/*  if(time_count==40){ exit(-1); }  */

/*  ================================================ */
/*
  printf("\n++++++++++++++ Time = %d ++++++++++++++++++++++++++++++++++\n",time_count);
*/
/*  ================================================ */

/*  if there are any cells in the first column of output queue location
 array then route them outside and calculate the average delay

 first write the first column of this array and see if we have
 some cells to route outside
*/

/*  ================================================ */
/*
  printf("\nThe first column of the output queue location array ......\n");
  for(j=1; j<=8; ++j){
   printf(" %3d",output_queue_location[1][j]);
  }
  printf("\n");

  printf("\nThe first column of the output queue address indicator array ......\n");
  for(j=1; j<=8; ++j){
   printf(" %3d",output_queue_address_indicator[1][j]);
  }
  printf("\n");
*/
/*  ================================================ */


 previous_cells_routed_outside=cells_routed_outside;

/*  route cells outside, decrease the output queue size, and decrease
 the input buffer size
*/

 for(j=1; j<=8; ++j){
  if(output_queue_location[1][j]>0){
   cells_routed_outside++;
   --output_queue[j];
```

```
   if(output_queue[j]<0){
     printf(" >>>>>>>>>>>> OUTPUT QUEUE at Port %d becoming -VE <<<<<<<<<<<<<\n",j);
     printf("\nThe first column of the output queue location array ......\n");
     for(j=1; j<=8; ++j){
      printf(" %3d",output_queue_location[1][j]);
     }
     printf("\n");
     printf("\nThe first column of the output queue address indicator array ......\n");
     for(j=1; j<=8; ++j){
      printf(" %3d",output_queue_address_indicator[1][j]);
     }
     printf("\n");
     exit(-1);
   }
   for(n=1; n<=8; ++n){
    if(output_queue_address_indicator[1][j]==n){
     --input_buffer[n];
    }
   }
  }
 }


/*  calculate average buffer and queue length */
 sum1 = 0;
 sum2 = 0;
 for(j=1; j<=8; ++j){
  sum1 = sum1 + input_buffer[j];
  sum2 = sum2 + output_queue[j];
 }
 if(sum1>0.0){ sum1 = sum1/8; }
 if(sum2>0.0){ sum2 = sum2/8; }
 ave_buffer_length_sum = ave_buffer_length_sum + sum1;
 ave_queue_length_sum = ave_queue_length_sum + sum2;



/*  calculate the average delay suffered by cells which have been
 routed during this time step
*/

 cell_out_count=0;
 sum = 0;
 for(j=1; j<=8; ++j){
  if(output_queue_location[1][j]>0){
   ++cell_out_count;
/*    this means a cell is routed from here, and we know its arrival time,
   and we know the current time, so we can find the delay
*/
```

```
   sum = sum + time_count-output_queue_location[1][j];
  }
 }
 sum = sum/cell_out_count;
 if(time_count>1){
  delay_so_far_sum = delay_so_far_sum + sum;
 }

/*  print how many packets are in the buffer */

/*  ================================================= */
/*
 printf("\nPackets in input buffers AFTER routing ......\n");
 for(i=1; i<=8; ++i){
  printf(" %3d",input_buffer[i]);
 }
 printf("\n");

 printf("At time %d ->  %d cells are routed (TOTAL cells routed = %d)\n",time_count,cells_routed_outside-previous_ce
 printf("Average delay suffered by these cells = %f\n",sum);
*/
/*  ================================================= */

/*  also move the output queue location array forward in time */

 for(k=2; k<=output_queue_size; ++k){
  for(j=1; j<=8; ++j){
   output_queue_location[k-1][j]=output_queue_location[k][j];
   output_queue_address_indicator[k-1][j]=output_queue_address_indicator[k][j];
  }
 }
 for(j=1; j<=8; ++j){
  output_queue_location[output_queue_size][j]=0;
  output_queue_address_indicator[output_queue_size][j]=0;
 }



/*  write the updated first column of the output queue location array */

/*  ================================================= */
/*
 printf("\nUPDATED first column of the output queue location array ......\n");
 for(j=1; j<=8; ++j){
  printf(" %3d",output_queue_location[1][j]);
 }
 printf("\n");

 printf("\nUPDATED first column of the output queue address indicator array ......\n");
```

```
 for(j=1; j<=8; ++j){
  printf(" %3d",output_queue_address_indicator[1][j]);
 }
 printf("\n");
*/
/*  ================================================== */




/*  read the arriving cells from the traffic file */

 sscanf(line,"%d %d %d %d %d %d %d %d",&cell_at_port[1],&cell_at_port[2],&cell_at_port[3],&cell_at_port[4],&cell_at_

/*  ================================================== */

/*  printf("CELLS ARRVING %8d. ",time_count);  */
 for(i=1; i<=8; ++i){
/*   printf(" %d",cell_at_port[i]);   */
  if(cell_at_port[i]==1){
   ++total_cells_arriving_at_all_ports;
  }
   }
/*
 printf("\n");
 printf("Total cells arrived so far = %d\n",total_cells_arriving_at_all_ports);
*/

/*  ================================================== */



/*  okay now we know whether a cell is coming into each of the 8 input ports
 or not, now we update the input_buffers
*/

 for(i=1; i<=8; ++i){
  if(input_buffer[i]<input_buffer_size&&cell_at_port[i]==1){
   ++input_buffer[i];
  }
  else if(input_buffer[i]>=input_buffer_size&&cell_at_port[i]==1){
   ++cell_loss_count;
   ++cell_loss_count_at_input_buffer;
   cell_at_port[i]=0;
/*    printf("CELL LOSS INPUT BUFFER FULL ..... at input port %d, Total cells lost = %d\n",i,cell_loss_count);  */
  }
   }
```

```
/*  print how many packets are in the buffer */

/*  ================================================= */
/*
 printf("\nPackets in input buffers ......\n");
 for(i=1; i<=8; ++i){
  printf(" %3d",input_buffer[i]);
 }
 printf("\n");
*/
/*  ================================================= */


/*  now we generate random numbers between 1 and 8 for each port, and accordingly
 decide to which output port the cell arriving at a certain input port
 is destined
 */
 num1 = 0.0;
 for(i=1; i<=8; ++i){
  decide_random[i] = 0; /* say that cell is going no where */
  num=rand();
  num1 = ((num/32767)*8.0);
  num1 = num1 + 1.0;
  if(cell_at_port[i]==1){  /* if a cell has arrived at a port */
   decide_random[i] = num1;
   if(decide_random[i]>8){
    decide_random[i]=8;
   }
  }
/*   printf("Cell at port %d is going to output port ----> %d (%f)\n",i,decide_random[i],num/32767);  */
 }


/*  okay, now we know that cells from input port i are destined for
 output port given by decide_random[i], so lets form the output queues.
 This is gonna be a little tricky, there are two things to do here,
 first update how may cells are in the output queue, and second at which
 locations the addresses of the cells are actually sitting. This info
 can be stored in an array of size 100 x 8, 100 corresponding to the
 size of the output queue and 8 corresponding to the 8 input ports. Also
 in order to find the delay a postive number at a location in this
 array will tell at what time did a certain cell arrive and this will
 help us to find the delay. Also the colums of the array will need to
 be kept moving forward in time so as to reflect the latest condition
 at any given time

 okay then, store the previous values of output_queue sizes in the
```

```
 previous_output_queue array and lets update the output queue size
*/

 for(j=1; j<=8; ++j){
  previous_output_queue[j]=output_queue[j];
  cell_loss_occurence_at_queue[j]=0;
 }


 for(i=1; i<=8; ++i){
  for(j=1; j<=8; ++j){
   if(decide_random[i]==j){
    if(output_queue[j]<output_queue_size){
     ++output_queue[j];
    }
    else if(output_queue[j]>=output_queue_size){
     ++cell_loss_count;
     ++cell_loss_at_particular_output_queue[j];
     cell_loss_occurence_at_queue[j]=1;
/*      printf("CELL LOSS OUTPUT QUEUE FULL .... at output port %d, Total cells lost so far = %d\n",j,cell_loss_count
     --input_buffer[i];
    }
   }
  }
 }

/*  print the previous and updated no. of header addresses in
 the output queue
*/


/*  ================================================ */
/*
 printf("\nNo. of address headers in the output queue ..................\n");
 printf("Previous: ");
 for(j=1; j<=8; ++j){
  printf(" %3d",previous_output_queue[j]);
 }
 printf("\n");

 printf("Updated:  ");
 for(j=1; j<=8; ++j){
  printf(" %3d",output_queue[j]);
 }
 printf("\n");
*/
/*  ================================================ */
```

```
/*  now we update the array whose locations contain info about when
 did a cell arrive and where the cell lies in the queues
 at all outport ports, we will update the locations in the array
 from the previous_output_queue to output_queue
*/

 for(j=1; j<=8; ++j){ /* for all 8 output ports */
/*   if(cell_loss_occurence_at_queue[j]==0){  */
   for(k=previous_output_queue[j]+1; k<=output_queue[j]; ++k){
     output_queue_location[k][j]=time_count;
   }
/*   }  */
 }

/*  also update the other array which contains the information
 about the input ports packets at the output queue. This array
 is similar to output queue location array, it contains info
 that the addresses at the output queue refer to which input
 port, and we will use this to decrement the no. of cells in
 the input buffers when cells are routed outside.
*/

 for(n=1; n<=8; ++n){
  this_port_done[n]=0;
 }

 for(j=1; j<=8; ++j){ /* for all 8 output ports */
/*   if(cell_loss_occurence_at_queue[j]==0){  */
   for(k=previous_output_queue[j]+1; k<=output_queue[j]; ++k){
    for(n=1; n<=8; ++n){ /* for all 8 input ports */
     if(decide_random[n]==j&&this_port_done[n]==0){
      output_queue_address_indicator[k][j]=n;
      this_port_done[n]=1;
      goto skip;
     }
    }
skip:     useless=1;
   }
/*   } */
 }


/*  now lets write this array to the point where there are no more
 addresses in the output queue

 but first we need to find upto where are we writing, so lets
 find the max. of the queue size at all output ports
```

```
*/

 max = -99;
 for(j=1; j<=8; ++j){
  if(max<output_queue[j]){ max=output_queue[j]; }
 }

/*  ================================================== */
/*
 printf("\nState of the output queue location array .....................\n");
 for(k=1; k<=max+1; ++k){
  printf("%3d. ",k);
  for(j=1; j<=8; ++j){
   printf(" %8d",output_queue_location[k][j]);
  }
  printf("\n");
 }

 printf("\nState of the output queue address indicator array .....................\n");
 for(k=1; k<=max+1; ++k){
  printf("%3d. ",k);
  for(j=1; j<=8; ++j){
   printf(" %8d",output_queue_address_indicator[k][j]);
  }
  printf("\n");
 }
*/
/*  ================================================== */

 sum = 0;
/*  printf("\nPackets in input buffers ......\n");  */
 for(i=1; i<=8; ++i){
/*   printf(" %3d",input_buffer[i]); */
  sum = sum + input_buffer[i];
 }
/*  printf("\n"); */
 balance=total_cells_arriving_at_all_ports-cells_routed_outside-cell_loss_count-sum;
 temp=cell_loss_count;
 temp1 = total_cells_arriving_at_all_ports;

/* okay this not percent any more, this is now loss probability */
 percent_lost=(temp/temp1);

 delay=delay_so_far_sum/time_count;

/*
 if(time_count%50000==0){
  printf("%8d. %8d  %8d  %8d (%8.6e)  %8.0f  %8.0f   %8.3f\n",time_count,total_cells_arriving_at_all_ports,cells_rou
 }
```

```
 else{
  printf("%8d. %8d  %8d  %8d (%4.3f)  %8.0f  %8.0f   %8.3f\r",time_count,total_cells_arriving_at_all_ports,cells_rou
 }
*/

/*  ------------------------------------------------------------- */

 if(time_count==395389||time_count==726223||time_count==996233||time_count==1328050){
  printf("%8d. %8d  %8d  %8d (%8.6e)  %8.0f  %8.0f   %8.3f\n",time_count,total_cells_arriving_at_all_ports,cells_rou
 }

/*  getchar();    */
        } /* this is the while loop for the cells coming in */

fclose(iof);

/*
printf("%8d. %8d  %8d  %8d (%8.6e)  %8.0f  %8.0f   %8.3f\n",time_count,total_cells_arriving_at_all_ports,cells_route
*/

printf("\n\n");


cell_loss_count_at_output_queue=cell_loss_count-cell_loss_count_at_input_buffer;

printf("Total cell loss           = %d -> %e loss prob.\n",cell_loss_count,percent_lost);
temp=cell_loss_count_at_input_buffer;
temp1=total_cells_arriving_at_all_ports;
num=temp/temp1;
printf("Cell loss at Input Buffer = %d -> %e loss prob.\n",cell_loss_count_at_input_buffer,num);

temp=cell_loss_count_at_output_queue;
temp1=total_cells_arriving_at_all_ports;
num=temp/temp1;
printf("Cell loss at Output Queue = %d -> %e loss prob.\n",cell_loss_count_at_output_queue,num);

printf("\n");

sum=0;
for(j=1; j<=8; ++j){
 sum = sum + cell_loss_at_particular_output_queue[j];
 printf("Cell loss at output queue %d = %d\n",j,cell_loss_at_particular_output_queue[j]);
}
printf(".... sum of losses = %10.0f\n",sum);

temp = ave_buffer_length_sum;
temp = temp/time_count;
printf("Average no. of cells in Input Buffer = %f\n",temp);
```

```
temp = ave_queue_length_sum;
temp = temp/time_count;
printf("Average no. of headers in Output Queue = %f\n",temp);



} /* this is the main loop */
```

Appendix C

Glossary

1. Address

   One or a group of characters specifying the recipient or originator of transmitted data. An address can also denote the position of data in computer memory or the data packet itself while in transit through a network. IEEE 802.3 and 802.5 recommend having a unique address for each device worldwide.

2. Asynchronous communications

   A method of transmitting data in which each transmitted character is sent separately. The character has integral start and stop bits so that the character can be sent at an arbitrary time, and separate from any other character. It is the most rudimentary type of communication as the originating and receiving machines do not have to be synchronized. Cheap, reliable and common among PCs and minicomputers, its disadvantage is the large number of extra bits needed for the data to be interpreted.

3. ATM (Asynchronous Transfer Mode)

   A cell- based data transfer technique in which channel demand determines packet allocation. ATM offers fast packet technology, real time, demand led switching for efficient use of network resources.

4. Asynchronous transmission

   Data transmission in which the instant that each character, or block of characters, starts is arbitrary; once started, the time of occurrence of each signal representing a bit within the character, or block, has the same relationship to significant instants of a fixed time frame.

5. Backplane

   The communication channels of a single device's architecture, such as in a hub or concentrator.

6. Bandwidth

The range of frequencies a transmission line or channel can carry: the greater the bandwidth, the greater the information - carrying capacity of a channel. For a digital this is defined in bit/s. For an analog channel it is dependent on the type and method of modulation used to encode the data.

7. Baseband

   A transmission medium through which digital signals are sent without complicated frequency shifting. In general, only one communication channel is available at any given time. Ethernet is an example of a baseband network.

8. B-ISDN (Broadband ISDN)

   The proposed advanced version of ISDN, providing speeds of 155.52Mbit/s and higher. Standards and switching technology that will work this fast are under development. It promises universal coverage based on ATM/SDH technologies and optical fiber, supporting data, voice and video traffic.

9. Bit

   A binary unit of information that can have two values, 0 or, 1. The word comes from a contraction of binary digit.

10. Bits per second

    The rate at which individual bits are transmitted across a communications link or circuit; written bit/s. One thousand bit/s is 1 Kbit/s, and one million bit/s is 1 Mbit/s.

11. Bit rate

    The rate at which bits are transmitted or received during communication, expressed as the number of bits in a given amount of time, usually 1 second.

12. Bridge

    Device connecting two separate networks at the OSI Data Link Layer (Level Two Media Access Control Layer). Once bridging is accomplished, the bridge makes interconnected LANs look like a single LAN, passing data between the networks and filtering local traffic.

13. Broadband

    Also referred to as wideband. A term describing any network that multiplexes multiple, independent network carrier frequencies on to a single cable. It allows multiple simultaneous "conversations", since the independent networks operate on different frequencies and do not interfere with each other.

14. Broadband Network

A network that uses multiple carrier frequencies to transmit multiplexed signals on a single cable. Several networks may coexist on a single cable without interfering with one another.

15. Broadcast

    The simultaneous transmission of data via a network from one terminal to a set of destinations or to all destinations.

16. Buffer

    A memory area or electronic register where data is stored temporarily while awaiting disposition. A buffer compensates for differences in data-flow rates (for example, between a terminal and its transmission line), and is also used as a data backup mechanism. A buffer can hold data that may be retransmitted if an error is detected during transmission.

17. Burst

    A method of data transfer in which information is sent as a large unit in one high-speed transmission.

18. Bus

    A LAN topology in which all the nodes are connected to a single cable. All nodes are considered equal and receive all transmissions on the medium.

19. Cell Relay

    Generic term for a protocol based on small fixed packet sizes capable of supporting voice, video and data at very high speeds. Information is handled in fixed length cells of 53 octets.

20. Cell Delay Variation (CDV)

    ATM performance parameter which specifies the potential variation (+/-) from the expected average transit delay through the network over a given virtual circuit.

21. Cell Transfer Delay (CTD)

    ATM performance parameter which specifies the average transit delay of cells between a source and destination over a given virtual circuit.

22. Circuit switching

    The transmission technique in which a physical circuit is established between sender and receiver before transmission takes place. When the transmission is complete, the circuit is freed.

23. Connection-Oriented

    In data networks, a type of computer relationship in which the network equipment

constructs a circuit between the two devices for the duration of their relationship. Once the circuit is established, the devices pass information back and forth through the circuit without regard to their physical addresses. The circuit may be physical or virtual.

24. Connectionless

    The opposite of connectionless, a type of relationship between two devices where each information packet must contain the address of the partner device.

25. Contention

    A network access method where all the devices on a network have equal chances of gaining control of the network at any time. Includes both collision detection (CSMA/CD) and collision avoidance (CSMA/CA) access methods.

26. CRC

    Cyclic Redundancy Check. A method of insuring data integrity where a calculation is performed using the binary representation of the data itself as the basis of the calculation. The CRC is the numerical result of this calculation and is held separately from the data. The integrity of the data is checked by calculating a new CRC and comparing it to the previously calculated CRC. If the two CRC' s match, then there is a high degree of confidence that the data has not changed.

27. Data

    Information represented in a format readable by a computer.

28. Destination

    Any point or location, such as a node, station, or particular terminal, to which information is to be sent.

29. Encapsulation

    The process of sending data encoded in one protocol format across a network operating a different protocol, where it is not possible or desirable to convert between the two protocols.

30. Error control

    A means of ensuring that information received across a transmission link is correct. The techniques involved typically use error detection to detect if the transmitted data has been corrupted. The error control technique involves asking for data to be retransmitted until a correct version is received.

31. Error correction

    A technique to restore data integrity in received data that has been corrupted during

transmission.

32. Error detection

    A set of techniques that can be used to detect errors in received data. Techniques that are applicable include parity checks involving parity bits, checksums or a Cyclic Redundancy Check.

33. E-mail

    Electronic Mail. A network application that can deliver messages from one computer user to another.

34. Ethernet

    The most widely LAN transmission network. Based on a bus network topology, it runs at a maximum 10Mbit/s - in practice far less - and adopts CSMA/CD techniques operating over convention co-axial cable, thin wire co-axial cable and unshielded twisted pair cabling. A fiber-optic implementation has also been defined.

35. Fiber optic

    A transmission media that use a light wave for signaling.

36. Flow control

    The procedures for controlling the rate of transfer of data between two points in a data network, such as between a protocol converter and a printer. This avoids data loss when a recipient device's buffer is full. Buffers play an essential role in overall flow control in a network.

37. Frame

    A group of bits sent over a link. A frame may contain control and addressing information, as well as error detection - for example CRC information - and forward error correction information. The size and composition of the frame varies according to the protocol. Often used synonymously with packet.

38. Frame relay

    A data communications interface originating from ISDN designed to provide high speed frame or packet transmission with minimum delay and efficient use of bandwidth. It is a variation on the X.25 interface and form of fast packet switching. It derives its name from using the Data Link or "frame" OSI layer Two to route or "relay" a packet directly to its destination instead of terminating the packet at each switching node. This eliminates processing overheads and increases throughput speed. Based on the ITU-TS Lap-D standard, it uses variable-length packets and ap-

plicable only to sub-broadband, T3/E3 or lower, data transmission. Like Ethernet, or token ring, frame relay assumes that connections are reliable. It does not have error detection and error control within the network, which helps to speed up the protocol. When errors occur frame relay relies on higher level protocols for error control. Frame relay is often viewed as a replacement for X.25, primarily for LAN-to-LAN bursty traffic. Voice over frame relay is available, but the subject of debate. It will also become an access method for ATM-based WANs.

39. FTP (File Transfer Protocol)
    The TCP/IP standard, high-level protocol for transferring files from one machine to another. Usually implemented as applications level programs, FTP uses the Telnet and TCP protocols.
40. Full-duplex
    A communication system between two entities in which either entity can transmit simultaneously.
41. Giga
    A prefix denoting a billion (109).
42. Header
    The control information added to the beginning of a transmitted message. This may consist of packet or block address, destination, message number and routing instructions.
43. Hop
    In routed networks, the passage of a packet through a router on the way to its destination.
44. Host
    In terminal emulation, the remote computer that is being controlled by the terminal emulation software.
45. IEEE
    Institute of Electrical and Electronic Engineers: a US publishing and standards organization responsible for many LAN standards such as the 802 series.
46. IEEE 802.2
    The Data Link standard for use with IEEE 802.3, 802.4 and 802.5 standards. It specifies how the basic data connection should be set up over the cable.
47. Internet
    1) The worldwide system of linked networks that is capable of exchanging mail and

data through a common addressing and naming system based on TCP/IP protocols. 2) Any group of linked networks capable of exchanging electronic mail and data using a common protocol.

48. Internet Address

    An address that identifies a communication entity on an internet.

49. IP

    Internet Protocol. The Network Layer protocol used in TCP/IP.

50. ISDN

    Integrated Services Digital Network. The CCITT standard for carry digital voice and data over the same wire.

51. ISO

    The International Standards Organization.

52. Jitter

    The difference between a real signal and its ideal due to distortion.

53. KBPS

    A unit of measure used to describe the rate of data transmission equal to 1000 bits per second.

54. KByte

    A unit of measure used to describe an amount of information equal to 1024 (210) bytes.

55. LAN (Local Area Network)

    A communications system that links computers into a network, usually via a wiring-based cabling scheme. LANs connect PCs, workstations and servers together to allow users to communicate and share resources like hard disk storage and printers. Devices linked by a LAN may be on the same floor or within a building or campus. It is user-owned and does not run over leased lines, though a LAN may have gateways to the PSTN or other, private, networks.

56. latency

    The amount of time it takes a packet to travel from source to destination. Together, latency and bandwidth define the speed and capacity of a network.

57. Layer

    Description of divisions in specifications Such as OSI and SNA communications protocols. Functions are grouped together that comprise one step in the hierarchy necessary for successful data communications.

58. Loss

    The aggregate attenuation of a signal due to interaction with its environment.

59. MAN (Metropolitan Area Network)

    A high speed network designed to link together sites in a metropolitan or campus area. The IEEE has defined its 802.6 standard for MANs based on the Distributed Queue Dual Bus technology.

60. MBPS

    A unit of measure used to describe the rate of data transmission equal to one millions bits per second.

61. MByte

    A unit of measure used to describe an amount of information equal to 1,048,576 (220) bytes.

62. Multi-cast message

    A message that is intended for a set of stations on a network.

63. Medium

    The physical method or equipment used for transmission, from a tangible fiber optic or copper cable to a satellite link.

64. Memory

    In computing, a system where data is stored for direct, high-speed access by a microprocessor.

65. Multi-user

    A term used to describe a computing process that can handle the requirements of several users simultaneously.

66. Multitasking

    A descriptive term for a computing device whose operating system can handle several tasks concurrently. In mono-processors such as the Mac, each active task is given short periods of time to use the CPU in a rotational fashion. There are many varieties and levels of sophistication of multitasking.

67. Network

    1) Referring to the infrastructure that supports electronic data exchange. Computing devices in a network may be separated by a repeater, hub or bridge.

68. Nodes

    Devices on a network that demand or supply services or where transmission paths are connected. Node is often used instead of workstation.

69. OC-1 Optical Carrier level 1

    The lowest optical transmission rate in the incipient Sonet standard at 51.48Mbit/s.

70. OC-3 Optical Carrier level 3.

    The second fastest optical rate in the incipient Sonet standard at 155.52Mbit/s.

71. Octet

    A grouping of eight bits in packet switched networks similar, but not identical to, a byte.

72. Optical fiber

    A thin filament of glass or other transparent material through which a signal encoded light beam may be transmitted by means of total internal reflection.

73. OSI

    Basic Reference Model (Open Systems Interconnection Reference Model)
    An architectural model describing how communications can be achieved between different vendors' systems. It is a logical structure for network operations standardized within ISO and containing seven primary layers.The seven Layers, starting with the lowest are the Physical, Data Link, Network, Transport, Session, Presentation and Applications.

74. Packet

    A collection of bits, including the address, data and control, that are switched and transmitted together. The terms frame and packet are often used synonymously.

75. Packet Switching

    A method of switching data in a network. Individual packets of a set size and format are accepted by the network and delivered to their destination. The sequence of packets is maintained, and destination established, by the exchange of control information (also contained in the packets) between the sending terminal and the network before the transmission starts. The network is open to all users, all the time, with packets from the various nodes being interleaved throughout the network. The packets can be sent in any order, as the control information sent at the beginning of the transmission ensures they are interpreted in the correct order at the receiving end. Because each packet carries its own control instructions, it can use any route to reach its destination. The link lasts only as long as the trarsmission. An ITU-TS standard for packet switched networks information.

76. Protocol

    A set of rules governing the information flow within a communications infrastructure,

often known as "data link control". Protocols control format, timing, error essential correction and running order. They are essential for a device to be able to interpret incoming information. Suites of protocols are often used in networks, with each protocol responsible for one part of a communications function.

77. Protocol data unit(PDU)
    A set of data specified in a protocol of a given layer and consisting of protocol control information of that layer, and possibly user data of that layer.
78. Queue
    A backup of packets awaiting processing.
79. Quality of Service (QoS)
    A mechanism for defining absolute and relative network performance requirements for the various streams of traffic on the internet.
80. Router
    A device that forwards packets between networks according to the rules of a network layer protocol.
81. Routing
    Process of delivering a message across one or more networks via the most appropriate path.
82. Routing table
    Information stored within a router that contains network path and status information. It is used to select the most appropriate route to forward information along.
83. Scalability
    The suitability of a system (particularly a network system) to operate properly and efficiently when configured on a large scale.
84. SDH (Synchronous Digital Hierarchy)
    ITU-TS synchronous transmission standards aligned with Sonet above 155Mbit/s, aimed at network operators. Designed with ATM in mind, SDH has many advantages over existing transmission technologies including flexibility in managing the transmission, reconfiguration and control and switching at data rates to 622Mbit/s and beyond. SDH-based networks are being implemented by operators in various parts of Europe and developed by major players such as Alcatel and Northern Telecom.
85. SONET
    Synchronous Optical Network. A standard for a world-wide digital network using a

common transport. SONET can run over copper or fiber.

86. Standard

    1) A synonym for specification. 2) A component or way of accomplishing a task that is so frequently and widely used that it

87. Switch

    A switch is a device that forwards packets between nodes based on the packet's destination node address (either hardware or protocol), typically with a buffer time longer than a repeater but shorter than the transmission time of the packet.

88. Synchronous

    A communication system where stations may only transmit at prescribed intervals and must provide a timing pulse with their packet.

89. System

    Any computer system that can be controlled by a user consisting of a CPU and optional equipment such as display monitors, disk drives and other peripherals. seems to be part of a specification.

90. TCP (Transmission Control Protocol)

    The standard transport level protocol that provides the full duplex, stream service on which many application protocols depend. TCP allows a process or one machine to send a stream of data to a process on another. Software implementing TCP usually resides in the operating system and uses the IP to transmit information across the network.

91. TDM (Time Division Multiplexer/Multiplexing)

    Multiplexer which apportions the time available on its Composite link between its channels, interleaving data from successive channels. The method divides up digital channels to make maximum use of their bandwidth, by taking input from each source in turn. TDMs use one of two methods to achieve this, bit interleaving for synchronous protocols and character interleaving for asynchronous protocols.

92. Telnet

    A process to access a remote computer system, often a Unix system, over the network. Origin: Teletype Network.

93. Transmission

    The activity of sending or conveying information.

94. UDP (User Datagram Protocol)

    The IP standard protocol that allows an application program on one machine to send

a datagram to an application program on another machine. UDP uses the Internet IP to deliver datagrams.

95. User

    A person who uses a computer system to accomplish a non-computing goal, as compared to a programmer or network manager.

96. Virtual channel

    A logical data channel within an ATM connection.

97. Virtual circuit

    A link that seems and behaves like a dedicated point to point line or a system that delivers packets in sequence, as happens on an actual point to point network. In reality, the data is delivered across a network via the most appropriate route. The sending and receiving devices do not have to be aware of the options and the route is chosen only when a message is sent. There is no prearrangement, so each virtual connection exists only for the duration of that one transmission.

98. Virtual channel identifier

    A 16-bit field in the ATM cell header that labels (identifies) a particular virtual channel.

99. Virtual path A generic term used to describe unidirectional transport of ATM cells belonging to virtual channels that are associated by a common unique identifier value.

100. Virtual path identifier

     The field in the ATM cell header that labels (identifies) a particular virtual path.

101. WAN

     Wide Area Network. A network that is created between and among devices separated by large distances (typically in excess of 50 miles).

PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis: PERFORMANCE ANALYSIS OF VIRTUAL ROUTING AND QUEUING (VRQ) SWITCH.

Author: [redacted]

DEEPALI ARORA

15 Dec 2001