DEVELOPMENT OF OBJECT-ORIENTED COMPONENTS FOR ATM NETWORK SIMULATION WITH EMPHASIS ON SWITCH ARCHITECTURE

A Final Year Project Report

Submitted to the

Faculty of Computer Science and Information Technology

University of Malaya

by

Tay Boon Pin

under the supervision of

Mr. Ling Teck Chaw

Dissertation submitted in partial fulfillment of the requirement for

the Degree of

Bachelor of Computer Science

Session 2000/2001

Submission Date (12 February 2001)

ABSTRACT

Asynchronous Transfer Mode (ATM) is considered to be the ground on which B-ISDN is to be built. It is the new generation of communication networks that are being deployed throughout the telecommunication industry. The basic functions of ATM switch are to direct cells from input port to output port and to buffer cell destined to the same output port from different input port.

This project focuses on the development of an object-oriented simulator for simulation. The simulator is designed and implemented to ensure the correctness of routing within the switch fabric, fairness of switching and to guarantee quality of service for ATM applications. As such, the implemented switching architecture is based on the Banyan NxN technique.

Finally, object-oriented approach will be used for the development of network simulator. The language I'll be using is Java and the tool is Borland JBuilder.

ACKNOWLEDGEMENT

This project has truly been a good experience for me. Along the process it has helped me to have a better understanding of networking, ATM and various switching architectures and Java programming. Besides, I had also gone through a lot of problems and hard work.

Firstly and foremost, I would like to extend my most gratitude to Mr. Ling Teck Chaw, my project supervisor who has provided me with unlimited support and guidance throughout the development of this project and also not forgetting Mr. Ibrahim Abubakar, my project moderator.

Utmost gratitude to Wong Wing Hong, Sin Wai Kit, Yu Soon Lye, Phung Jacen, Jimmy Tan, Wong Chee Sum, Ching Kim Joo, Lim Shiau Hong, and Ang Tan Fong for sharing their knowledge and help throughout this project.

Finally, special thanks to my family members and Looi Se Min for their support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AAL	ATM Adaptation Layer
ABR	Available Bit Rate
ATM	Asynchronous Transfer Mode
B-ISDN	Broadband Integrated Services Digital Network
BTE	Broadband Terminal Equipment
CAC	Connection Admission Control
CBR	Constant Bit Rate
CDV	Cell Delay Variation
CLR	Cell Loss Ratio
CTD	Cell Transfer Delay
EFCI	Explicit Forward Congestion Indication
FIFO	First-In-First-Out
GFR	Guaranteed Frame Rate
GUI	Graphical User Interface
HEC	Header Error Control
IC	Input Controller
IDE	Integrated Development Environment
IM	Input Module
LAN	Local Area Network
MBS	Maximum Burst Size
MCR	Minimum Cell Rate
MIN	Multistage Interconnection Network
NIST	National Institute of Standards and Technology
NPC	Network Parameter Control
OAM	Operations and Maintenance
OC	Output Controller
OM	Output Module
OOP	Object-Oriented Programming
PCR	Peak Cell Rate
QoS	Quality of Service

RAM	Random Access Memory
RCSP	Rate Controlled Static Priority
SCR	Sustainable Cell Rate
SF	Switch Fabric
STM	Synchronous Transfer Mode
TCP/IP	Transmission Control Protocol over Internet Protocol
TCT	Total Cells Transferred
UBR	Unspecified Bit Rate
UDP	User Datagram Protocol
UNI	User Network Interface
UPC	Usage Parameter Control
VBR	Variable Bit Rate
VCI	Virtual Channel Identifier
VPI	Virtual Path Identifier

CHAPTER 1: INTRODUCTION

Asynchronous Transfer Mode (ATM) is the new generation of computer and communication networks that are being deployed throughout the telecommunication industry as well as in campus backbones today. It is designed to handle different kinds of traffic (voice, video and data) in an integrated way. Therefore, it was selected by the telecommunication industry as the technology to deliver Broadband Integrated Services Digital Network (B-ISDN) carrier service. The ATM network transmits data in small packets in fixed size of 53 bytes.

This thesis attempts to present a survey, and describe the design and development of a high-speed Asynchronous Transfer Mode (ATM) switching simulator using an object-oriented approach. The key area of research emphasizes on the switching architecture design used in transferring ATM cells from one ATM switch to another.

1.1 Introduction to Asynchronous Transfer Mode

Asynchronous Transfer Mode (ATM) also known as cell relay is a technology that was developed as part of the work on Broadband ISDN in the 1970s and 1980s. It is a high-performance, cell-oriented switching and multiplexing technology that utilizes fixed-length packets to carry different types of traffic. Technically, it can be viewed as an evolution of packet switching. Like packet switching for data (e.g., TCP/IP), ATM integrates the multiplexing and switching functions. It is well suited for bursty traffic in contrast to circuit switching and allows communications between devices that operate at different speeds. Unlike packet switching, ATM is designed for high-performance multimedia networking.

ATM technology has been implemented in a very broad range of networking devices like PC, workstation, and server network interface cards, switched-Ethernet and token-ring workgroup hubs, workgroup and campus ATM switches, ATM enterprise network switches, ATM multiplexes, ATM–edge switches, and ATM–backbone switches.

ATM is capable to be offered as an end-user service-by-service providers (as a basis for tariff services) or as a networking infrastructure for these and other services. The most basic service building block is the ATM virtual circuit, which is an end-to-end connection that has defined end points and routes but does not have bandwidth dedicated to it. Bandwidth is allocated on demand by the network as users have traffic to transmit. ATM also defines various classes of service to meet a broad range of application needs [1].

1.1.1 Quality of Service

One of the great advantages of ATM is its support for guaranteed QoS in connections. Hence, a node requesting a connection set up can request a certain QoS from the network and can be assured that the network would deliver that QoS for the life of the connection being established. Such connections are categorized into various types of ATM service categories [2]:

- *Constant Bit Rate (CBR)*: real time traffic
- *Real Time Variable Bit Rate (rt-VBR*: real time traffic
- *Non-Real Time Variable Bit Rate (nrt-VBR)*: non real time traffic
- *Available Bit Rate (ABR), Unspecified Bit Rate (UBR)*: non real time traffic
- *Guaranteed Frame Rate (GFR)*: non real time traffic

The service categories are depending on the nature of the QoS guarantee desired on the traffic types. The CBR service is aimed at supporting voice and other synchronous applications, a fixed data rate is required and made available by the ATM provider. The network must ensure the capacity is available for the CBR connection and the subscriber does not exceed its allocation. The VBR (real time and non-real time) services are designed to support video and audio applications, which do not need asynchronous transfer. Real time VBR in intended for application that requires tightly constrained delay and delay variation but could not exhibit the fixed data rate for CBR. Non-real time VBR does not bound this delay variation, and a certain cell loss ratio is allowed.

UBR and ABR services are typically non-real time traffic, which are designed to primarily support data application. UBR is a best-effort service where no amount of capacity is guaranteed, and any cell may be discarded. ABR provides user with a guaranteed minimum capacity. Therefore, a peak cell rate (PCR) that it will use and a minimum cell rate (MCR) that it requires are specified. The network allocates resource so that ABR could receive at least their MCR capacity. When additional capacity is available, the user may burst above the minimum rate with minimized risk of cell loss.

GFR is to support non-real time service application. It is designed for an application, which require minimum rate guarantee and can benefit from accessing additional bandwidth dynamically available in the network.

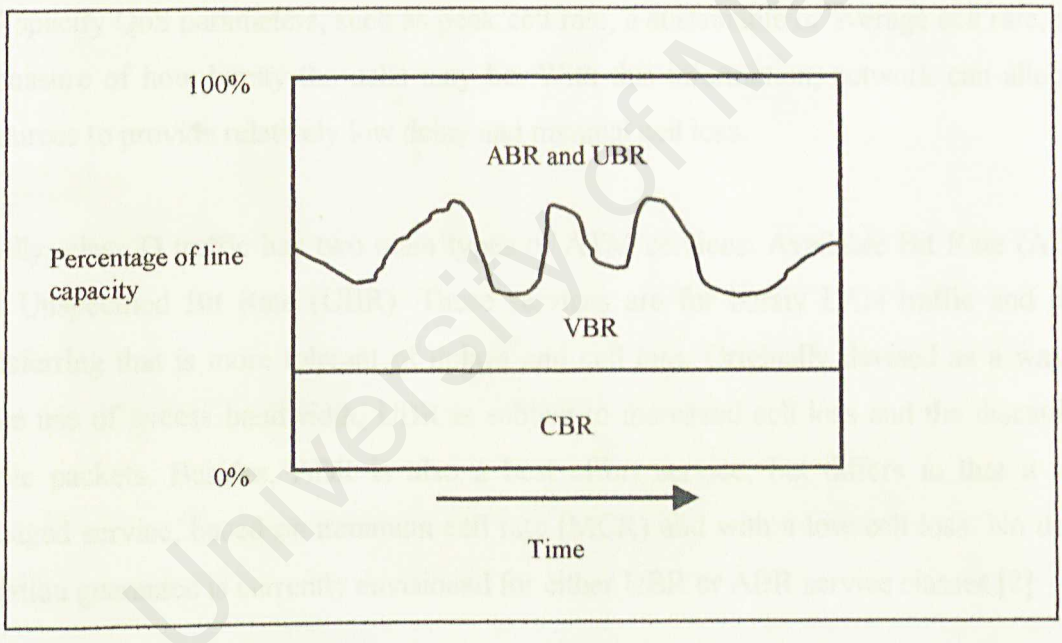

Figure 1.1: ATM Bit Rate Services

The ATM services are defined into four main classes: *Class A*, *Class B*, *Class C* and *Class D*.

Class A is mainly for Constant Bit Rate (CBR) traffic. As discuss earlier, it is mainly used by applications that require a fixed data rate and continuously available during the

connection lifetime and a relatively tight upper bound on transfer delay. Examples of CBR applications include video conferencing, interactive audio, distance learning, and video-on-demand.

Class B traffic is for real-time Variable Bit Rate (rt-VBR). It is intended for applications, which are time-sensitive and their end-to-end delay is critical. Typically, the difference between applications of class B and class A is that class B applications transmit at a rate that varies with time. In other words, class B traffic is considered bursty.

Class C is for non-real time (nrt-VBR) traffic. For this service class, delay is not very critical as compared with rt-VBR of class B. Examples for this service are applications like video playback, training tapes and video mail messages. The application end system will specify QoS parameters, such as peak cell rate, a sustainable or average cell rate, and a measure of how bursty the cells may be. With this information, network can allocate resources to provide relatively low delay and minimal cell loss.

Finally, class D traffic has two main types of ATM services: Available Bit Rate (ABR) and Unspecified Bit Rate (UBR). These services are for bursty LAN traffic and data transferring that is more tolerant of delays and cell loss. Originally devised as a way to make use of excess bandwidth, UBR is subject to increased cell loss and the discard of whole packets. Besides, ABR is also a best effort service, but differs in that it is a managed service, based on minimum cell rate (MCR) and with a low cell loss. No delay variation guarantee is currently envisioned for either UBR or ABR service classes [2].

In [3], there are QoS parameters that correspond to the network performance at the ATM layer that defines the ATM service categories. Those negotiated parameters are as follows:

- *Peak-to-peak Cell Delay Variation (peak-to-peak CDV)*: The difference of the maximum and minimum CDV and Instantaneous CDV are used.

- *Maximum Cell Transfer Delay (maxCTD)*: The delay experience by a cell between the first bit of cell transmitted by the source and the last bit of cell is received by the destination.
- *Cell Loss Ratio (CLR)*: The percentage of cells that are lost in the network due to error or congestion and are not received by the destination.

Table 1.1: ATM Service Classification

	Class A	**Class B**	**Class C**	**Class D**
Timing relation between source and destination	Required		Not required	
Bit rate	Constant	Variable		
Connection mode	Connection oriented			Connectionless
AAL protocol	AAL 1	AAL 2	AAL 3/4	
			AAL 5	

The explanation starts from defining the cell transfer delay (CTD), which is the elapsed time between two cell events. It refers to the time between transmission of the last bit of a cell at the source UNI and the receipt of the first bit of cell at the destination UNI. Typically, it has a probability density function that looks like in Figure 1.2.

There is a minimum delay, called the fixed delay that includes propagation delay through the physical media, delays induced by the transmission system, and a fixed component of switch processing delay. The variation of the cell delay (CDV) is due to the buffering and cell scheduling. CDV usually is negotiated during the establishment of connection. It is the difference between the best and worst case expected end-to-end cell transfer delay.

From Figure 1.2, *maxCTD* is the maximum requested delay for the connection. A fraction α of cells will exceed this threshold and must either be discarded or delivered late. The remaining portions are within the requested QoS. The range between the fixed delay and

maxCTD is referred to the *peak-to-peak CDV*. Finally, the cell loss ratio is simply the ratio of lost cells to the total transmitted cells on the connection.

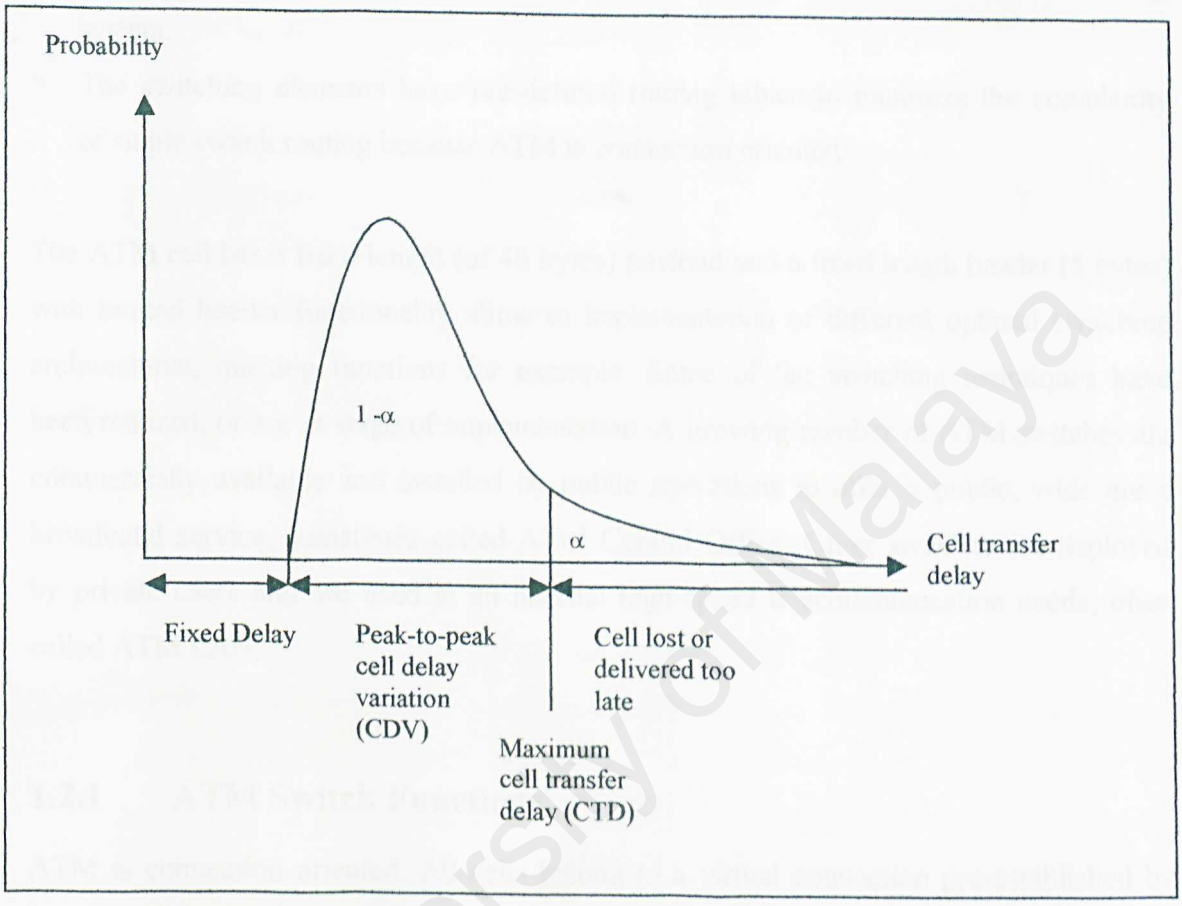

Figure 1.2: Cell Transfer Delay Probability Density Function (For Real-Time Service Categories)

1.2 Introduction to ATM Switching

Various switching architectures were developed in the past for different application such as voice and data, based on modes like STM (Synchronous Transfer Mode) and packet switching. The switching architectures that were previously developed for STM and for conventional packet switching like X.25 are not directly applicable for broadband ATM. Three major factors have a large impact on the implementation of the ATM switching architecture:

- The high speed at which the switch has to operate (from 150Mbit/sec up to 600Mbit/sec).
- The statistical behavior of the ATM stream passing though the ATM switching system.
- The switching elements have pre-defined routing tables to minimize the complexity of single switch routing because ATM is connection oriented.

The ATM cell has a fixed length (of 48 bytes) payload and a fixed length header (5 bytes) with limited header functionality allow to implementation of different optimal switching architectures, queuing functions for example. Some of the switching techniques have been realized, or are in stage of implementation. A growing number of ATM switches are commercially available and installed by public operations to offer a public, wide are a broadband service, sometimes called ATM Central Office. Other switches are deployed by private users and are used in an internal high-speed telecommunication needs, often called ATM LAN.

1.2.1 ATM Switch Functions

ATM is connection oriented. All cells belong to a virtual connection pre-established by the transport network. All traffic is segmented into cells for transmission across the network. The sequence integrity of all the cells in the virtual connection is preserved across each ATM switch to simplify reconstruction of the original traffic at the destination (allows smaller total delay on the net). The ATM cell is 53 bytes long, built of 48 payload bytes and a 5 bytes header. Each cell's header contains a VCI (virtual channel identifier) that identifies the virtual connection to which the cell belongs. The ATM switch has several main tasks:

- *VCI translation*
  - The established connection on the ATM network defines the virtual path through different switches across the network. The VCI is local to each switch port. As each cell travels across an ATM switch, the VCI is translated into a new value.

The switch has to build the new cell header containing the new VCI (and possibly new VPI - virtual path identifier) and calculate the new HEC value.

- *Switching* - Cell transport from its input to its output
  - The transportation of the information (cell) from an incoming logical ATM channel (inlet) to an outgoing logical ATM channel (outlet), is also the responsibility of the ATM switch. The logical ATM channel is characterized by two identifiers:
    1. The physical inlet/outlet, which is characterized by a physical port number.
    2. The logical channel on the physical port, which is identified by the VCI and/or the VPI.
  - In order to provide the switching function, both physical and logical identifiers of the incoming cell have to be related to physical and logical identifiers of the outgoing cell. Two functions have to be implemented in the ATM switching system.
  - The first function is the space switching function. The space switching function is the one, which allows the connection between every input and every output. An important aspect of space switching is the internal routing. This means how the information is routed internally in the switch. The internal structure of the switch must allow connections between every input to every output.
  - The second function is time switching. Since ATM is working in an asynchronous mode, cells which had arrived in various time slots from the different inputs can be delivered from different outputs in different time slots (there is no time identifier in ATM as it is in STM). Since there is no pre-assigned time slot connection, a contention problem arises if more than two logical channels are connected to the same output at the same time slot. This problem in the ATM switch is solved by implementing a queuing function in the ATM switch system.

1.2.2 ATM Switching Structure

ATM switching consists of the following main modules, Input Module (IM), Switch Fabric (SF), and Output Module (OM). The Input Module is responsible for recovering

the STM cells from the incoming bit-stream. Input Module also translates VPI/VCI values and determines the output port. The Output Module performs the inverse of the Input Module functions, mainly translation of ATM cells to the outgoing bit-stream. The switch fabric handles cell buffering, concentration and multiplexing, multicasting, broadcasting, cell discarding, and the actual cell transfer [4].

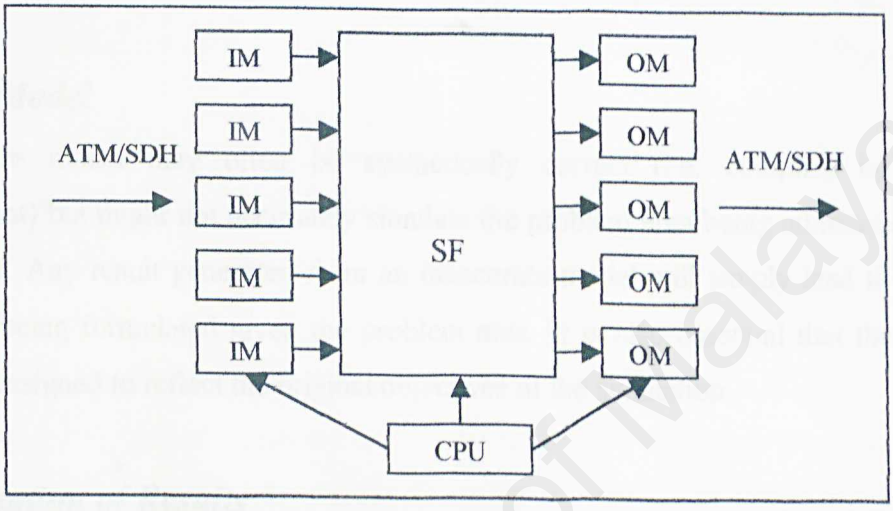

Figure 1.3: A Generic ATM Switching Structure

1.3 Introduction to Network Simulation

Computer Network Simulation is a valuable tool when attempting to form ideas on the correct solution for data communications needs. Complex network architectures and topologies are commonplace. With the advent of ATM high-speed networking solutions, network simulations allow designers make decisions without the need to invest in this new technology. Designers can test their new designs and carry out performance related studies using a simulation and therefore freed from the burden of the "trial and error" implementations. Huge saving can be made both in terms of investment and the cost in terms of unnecessary restructuring for experimentation.

Simulation is the basis for making decision. Decisions are formulated based on the information resulting from the simulation. The objective of simulation is to form a correct

decision. A number of factors influence the probability of making a correct decision. These are summarized here as:

The level of understanding of the Problem

The problem should be well defined and manageable. A clear understanding of the problem is essential before a simulation model can be developed.

Correct Model

A software model may often be syntactically correct (i.e. compiles in software environment) but might not accurately simulate the problem area being addressed (i.e. the semantics). Any result generated from an inaccurate model will simply lead to incorrect decisions being formulated given the problem area. It is also essential that the model is correctly designed to reflect the original objectives of the simulation.

Interpretation of Results

The simulation model simply produces output data. This data must be manipulated and interpreted by the developer. The correct interpretation of this data is dependent on the usefulness of the output data and also the user's understanding of statistical methods (e.g. variance, distribution functions) [5].

1.3.1 Advantages and Disadvantages of Simulation

The advantages of simulation are described as below:

- Economical and quick to assemble.
- Given sufficient computing resources, can do large-scale tests.
- Tests are controlled, reproducible.

However, certain problems exist with simulation too:

- Need to redo code for simulation environment.
- Simulation implementation may differ considerably from real one.
- Synthetic environment may also poorly represent real one [6].

1.3.2 Types of Simulation

There are two forms of network simulation: *Analytical Modeling* and *Discrete Event Simulation*.

Analytical Modeling is a mathematical technique that characterizes a network as a set of equations. The over simplistic view of the network and the inability to simulate the dynamic nature of a computer network are the main disadvantages of this technique.

Discrete Event Simulation is a computer model of some physical system, where the state of the system is assumed to change only at discrete points in simulated time. It is the study of a complex system by computing the times that would be associated with real events in a real-life situation. This could be the average end-to-end delay of packets. Discrete Event Simulation has many advantages; it requires far greater processing time. Also, quite a considerable investment of time is needed to accurately simulate most models [7].

1.4 Project Objectives

The first objective of this project is to study and understand the ATM switching simulation techniques. Through the study of these techniques, we can better model and attempt to improve on the existing switching architectures. Numerous switching techniques have been proposed since the implementation of the ATM switch. The thesis begins with a study on the current switching techniques and their effectiveness.

The second objective, which is of primary importance, is the development of the ATM switching simulator itself. The simulator is developed using an object-oriented approach to take advantage of the features such as modularity, extensibility, reusability and others. Moreover, applying multithreading into a simulator design can be used to closely model the real ATM switch.

The final objective is the creation of a portable, cross-platform and a user-friendly graphical user interface (GUI) simulator.

1.5 Goals

The primary goal of this project is to create a component-based ATM switching simulator. These switching-components should be able to be integrated to form a complete ATM network topology. A typical example is: begin from a source application, flow to the simulated ATM switch, finally connecting to the other end system and reaching the destination application.

This ATM switching simulator shall have the following capabilities:

- Allow the multiplexing of different ATM applications over a single ATM port.
- Allow ATM cells to be routed properly to the output port.
- Control the transfer rate of ATM application within an ATM switch.

1.6 Project Scheduling

Table 1.2: Project Progress

	Jun 00	Jul 00	Aug 00	Sep 00	Oct 00	Nov 00	Dec 00	Jan 01
Research Work	√	√	√	√				
System Analysis and Design				√	√	√		
System Implementation and Testing						√	√	√
Documentation								√

1.7 Report Organization

An introduction to ATM has been covered in Chapter 1. The two major roles of the ATM switch (i.e. Virtual Channel Identifier translation and switching) are also presented. The ATM switching structure (Figure 1.3) shows the flow of the ATM signal from the input module, passing through the switching fabric and finally reaching the output module. The advantages and disadvantages of the simulation and the two types of simulation: analytical modeling and discrete event simulation were also covered in Chapter 1. At the end, a discussion on the project objectives, goals, project scheduling and report organization are covered.

In Chapter 2, a survey on various simulators is covered. This chapter also contains programming techniques, languages and tools that will be used in assisting the development of the system. In addition, a survey of different switching techniques used in ATM networks, different buffering and queuing schemes also had been conducted. Lastly, various switching models and switching performance issues have been discussed.

Chapter 3 is about the analysis and design of the system. This chapter discussed about the details of switching functions, ATM switch architecture, and ATM traffic parameters used in the simulation model. At the end of this chapter, the considerations taken into account in designing the high performance ATM switch, JavaSim architecture, system architecture design, object-oriented design and class design are also presented.

Chapter 4 consists of two parts, which is implementation and testing. Implementation focuses on declaration of objects and testing describes component testing and system testing.

Finally, a conclusion for this project is provided in Chapter 5. Discussions about the strengths of the simulator, its limitations and the possibilities for future enhancements are highlighted here.

CHAPTER 2: LITERATURE REVIEW

This chapter begins with an introduction to various simulators. A detailed description is provided for three simulators that have been studied, including the advantages and disadvantages for each simulator. This chapter also provides an explanation for the reasons why object-oriented approach and not procedural approach was chosen when developing the switching model. It also explains which object-oriented programming language was selected and finally provides a survey of the available Java development tools. A study is conducted on the ATM switching approach, buffering methods, queuing models, switching models and switching performance issues.

2.1 Introduction to Various Simulators

As the emerging of speeds and dynamic of the computer networks today, management of the network traffic is highly important.

Therefore, there are a few simulation packages to describe a number of simulation experiments performed on the number of different configurations without the expanse of building a real network. The simulator actually offers a practical means of obtaining accurate information on which to plan and design a new system. Simulation is a useful technique for computer systems to perform analysis for high-speed network, especially for ATM.

The general requirements of an ATM simulator are to support network performance analysis under varying traffic types and loads, network capacity planning, traffic aggregation studies and ATM network protocol research. This spans a wide range of applications from production use by ATM network planners to ATM switch; network a protocol design by researchers.

The following sub-sections describe various simulators that have been implemented, including NIST ATM/HFC, INSANE, REAL Network Simulator, and Java Network Simulator. At the end of this section, a comparison among these simulators is presented.

2.1.1 NIST ATM/HFC Network Simulator

The ATM/HFC network simulator is a tool to analyze the behavior of ATM and HFC networks without the expense of building a real network. This simulator was developed at the National Institute of Standards and Technology (NIST). This simulator is written in C Language whereby it is written in structural programming approach. Typically, the simulator program includes a graphical interface which provides the user with a means to display the topology of the network, define the parameters and connectivity of the network, log data, and to save and load the network configuration. In addition to the user interface, the simulator has an event manager, I/O routines, and various tools that can be used to build components [8].

Advantages

The user can create different network topologies; adjust the parameters of each component's operation, measure network activity, save/load different simulation configuration and log data during simulation execution. The simulator is equipped with graphical user interface.

Disadvantages

Users of the simulator might face problems setting up the network topology because of the requirement to consider a large number of parameters.

User or programmers needs to have strong foundation in C programming language to customize the simulator's components. Besides, it is using procedural approach whereby the components have overlapped functions between the components. This is not supposed to happen in object-oriented programming approach.

The simulator only can run is UNIX or LINUX platform. This makes the simulator can only run in limited platforms and it is not widely used.

2.1.2 INSANE Simulator

INSANE is a network simulator designed to test various IP-over-ATM algorithms with realistic traffic loads derived from empirical traffic measurements [9]. INSANE's ATM protocol stack provides real-time guarantees to ATM virtual circuits by using Rate Controlled Static Priority (RCSP) queuing. ATM signaling is performed using a protocol similar to the Real-Time Channel Administration Protocol (RCAP).

Internet protocols supported include large subsets of IP, TCP, and UDP. In particular, the simulated TCP implementation performs: connection management, slow-start, flow and congestion control, retransmission, and fast retransmit.

The bulk of INSANE is written in C++. Customization and simulation configuration is performed with Tcl script. Some version of INSANE has been tested and determined to work in the following hardware and platforms:

- DEC Alpha AXP series, Digital UNIX 3.2
- DEC DECstation 5000 series, Ultrix 4.2A
- HP PA-RISC 9000/700 series, HP-UX 9.03
- IBM RS6000 series, AIX 3.2.5
- Intel x86, BSDI BSD/OS 2.0
- Intel x86, FreeBSD 2.1.0 and 2.1.5 release
- Intel x86, NetBSD 1.0A
- SGI Indigo, Irix 5.3
- Sun Sparcstation, Solaris 2.3, 2.4, and 2.5

Advantages

Although there is no graphical user interface, an optional Tk-based graphical simulation monitor provides an easy way to check the progress of multiple running simulation processes. It is designed to run large simulations whose results are processed off-line. Besides, it works quite well on distributed computing clusters (although simulations are all sequential processes, a large number of them can easily be run in parallel).

Disadvantages

The simulator is restricted to work in only certain hardware and platforms as listed above. Also, INSANE currently requires the following other software packages to run the system:

- g++ (version 2.6.3 or greater)
- libg++ (any version consistent with the installed g++)
- GNU make (pretty much any recent version)
- Tcl (version 7.3 or greater). Tk (version 4.0 or greater) is required for the simulation monitor, but is not necessary to run simulations. INSANE has been tested with Tcl 7.5 and 7.6, and Tk 4.1 and 4.2

2.1.3 REAL Network Simulator

The REAL network simulator is a network simulator designed for testing on congestion and flow control mechanisms. The simulator takes as input a scenario, which is description of network topology, protocols, workload and control parameters. It produce as output statistics such as the number of packets sent by each source of data, the queuing point, the number of dropped and retransmitted packets and other similar information. This simulator has many different versions.

For REAL version 5.0, it provides users with a way of specifying such networks and to simulate their behavior. It provides around 30 modules (written in C) that exactly emulate the action of several well-known flow control protocols (such as TCP), and 5 research scheduling discipline (such as Fair Queuing and Hierarchical Round Robin). Besides, it includes a graphical user interface (GUI) written in Java. This allows users to quickly build simulation scenarios with a point-and-click interface [10].

This REAL 5.0 simulator runs on Sun3s, Sparcs, MIPS boxes, Vaxen and 3B2, under 4.3BSD-like operating systems: SunOS, IRIX, UMIPS, Ultrix etc. Besides, the REAL version 4.0 has been successfully ported to i386/FreeBSD 2.0.5 platform and the Linux (Red Hat Release) platform [11].

The files in REAL 5.0 can be divided into the following logical classes:

- *Node functions*: These are the functions that execute protocols in nodes.
- *Queue management and routing*: These manage buffers in nodes and gateways, and perform packet switching.

Node functions implement computation at each of the nodes in the network. There are three types of node function: source, router and sink.

The queue management functions are written in an object oriented and layered style. The queue objects are manipulated by a small set of functions. Each layer provides services that the layer above uses to provide its own services. Packets are buffered in a per-conversation linked list and are accessed by two pointers: one points to the packet at the head of the queue, and another points to the tail. Each packet has a field, which points to the next packet in the queue.

Comparing REAL 5.0 to REAL 4.0 in October 1993, and in the last four years, many changes have been made. These include:

- many new simulation modules
- a Java-based GUI
- faster, smaller, cleaner simulation engine
- ports to FreeBSD, Solaris, and Digital Unix (OSF/3)
- simulation exercises based on my book
- minor bug fixes

Advantages

This simulator provides a flexible test bed for studying the dynamic behavior of flow and congestion control schemes in packet switch data networks. Besides, the user can modify the simulator software to accommodate network components.

Disadvantages

Some of the REAL version's network simulator does not give user an interactive modeling environment with a graphical user interface (GUI) representation capabilities but it is available in REAL version 5.0. This version includes a graphical user interface (GUI) written in Java and it allows users to quickly build simulation scenarios with a point-and-click interface. Besides, knowledge of C programming language and different platforms is a must for programmers to make changes from the source code provided because this simulator only will run in several platforms.

2.1.4 Java Network Simulator

This Java network simulator is a flexible test bed for studying and evaluating the performance of ATM network without the expense of building a real network. This simulator is written in Java language, thus it is written in object-oriented programming approach. Originally, this simulator is a Java version of NIST ATM/HFC network simulator enhanced with object-oriented features. Typically, the simulator is a tool that give user an interactive modeling environment with a graphical user interface which provides the user with a means to display the topology of the network, define the parameters and connectivity of the network, log data from simulation run, and to save and load the network configuration.

Class JavaSim

The *JavaSim* object is the main object of the simulator. It keeps a list of all the network components (all are descendents of *SimComponent*), and a list (a queue) of all events (in the form of *SimEvent*). Every component contains a set of parameters (all inherit from *SimParameter*).

Class Cell

The *Cell* class is a data resource class used by components like *CBRApp*, *GenericBTE* and *GenericATMSwitch* throughout the simulator. As a result, it contains attributes needed by the operation of all executor classes. The main contribution of *Cell* is to

provides necessary information for ATM switching. Major attributes are virtual path identifier and virtual channel identifier which provide switching information.

Class SimClock

The simulator is *event* driven. Components send each other events in order to communicate and send cells through the network. The software contains an event manager, which provides a general facility to schedule and send, or "fire" an event. An event queue is maintained in which events are kept sorted by time. To fire an event, the first event in the queue is removed, the global time is set to the time of that event and any action scheduled to take place is undertaken.

Events can be scheduled at the current time or at any time in the future. Scheduling events for the past is considered illogical. Events scheduled at the same time are not guaranteed to fire in any particular order. Simulator time is maintained by the event manager in units of *ticks*. The time is maintained as an unsigned 32-bit value. The simulator time represented by one tick can be changed by software modification, but not by the simulator user. It provides a set of time translation functions (all static) for normal translation between tick and actual time (in microseconds, milliseconds and seconds).

Class SimEvent

Every *SimComponent* communicates with each other by enqueuing *SimEvent* for the target component. For example, when component A wants to send a packet to component B, component A creates a *SimEvent* that specifies B as its destination, and enqueue the event. The *SimEvent* object also contains a time so that this event is fired at exactly the specified time. Component B will then be able to react to the event accordingly.

Class SimComponent

This is the most important class to understand in the simulator in order to development new components. Every network component in the simulation must inherit *SimComponent*. The *SimComponent* class itself should not be instantiated (although this is possible) because it only provides the skeleton for an actual component. A new

component should extends *SimComponent* and override its various methods in order to provide meaningful operations for the component.

Class SimParameter

Every *SimComponent* can have internal parameters (not shown/accessible by users) or external parameters (shown/accessible by users). All external parameters must inherit *SimParameter*. By extending *SimParameter*, one obtains parameter logging and meter display features automatically. *SimParamInt*, *SimParamDouble*, and *SimParamBool*, provide support for integer, double, and boolean parameters. Other types of parameters can be created by extending SimParameter accordingly.

ATM Generic Switch

The switch is the component that switches or routes cells over several virtual channel links. A local routing table is provided for each switch. This table contains a route number (that is read from the incoming cell structure and is the equivalent of the cell's virtual channel identifier), a next link entry, and a next switch/next BTE (host) entry. Let's consider a cell arriving at the switch from a physical link. At the next switching slot time, after some delay (set by user), the switch looks in its local routing table to determine which outgoing link it should redirect the cell to.

Broadband Terminal Equipment (BTE)

The BTE component simulates a Broadband ISDN node, e.g., a host computer, workstation, etc. A BTE has one or more ATM applications at the user side and a physical link on the network side. Cells received from the application side are forwarded to the physical link.

Link Components

This component simulates the physical medium (copper wire or optical fiber) on which cells are transmitted. The user may choose the link speed from a list of several different

standard rates. The user also specifies the length of the link. The output parameter reported by the simulator is link utilization in terms of bit rate (Mbits/s).

ATM Applications

The ATM application at the end-point of a link is a traffic generator. The traffic source emulated by this component may be a constant bit rate (CBR) source or a variable bit rate (VBR) source. Either source type may be generated at one of three priority levels: a CBR/VBR level (highest priority), the Available Bit Rate (ABR) level where cells are sent on the transmission bandwidth that is available after the higher level traffic has been sent, and the Unspecified Bit Rate (UBR), the lowest priority traffic.

Advantages

The simulator is equipped with better graphical user interface. The user can create different network topologies; adjust the parameters of each component's operation, measure network activity, save/load different simulation configuration and log data during simulation execution. Output performance can be viewed in text based and graphical representation on the screen while the simulation is running. This simulator also works on various platforms.

Disadvantages

This simulator is written in Java language, so it is not a web-enable approach. It requires a lot of memory processing during the simulation.

2.1.5 Comparison

The following table gives a comparison among the studied simulators. Features being compared are discrete event simulation, object-oriented, graphical user interface (GUI), multithread, and web-enable.

Table 2.1 Comparisons among Various Simulators

Simulator	Discrete event simulation	Object-oriented	GUI	multithread	Web-enable	Platform independent
NIST ATM/HFC	√	X	√	X	X	X
INSANE	√	√	Poor	X	X	X
REAL	√	X	Poor	X	X	X
JAVASIM	√	√	√	√	X	√

2.2 Programming Approaches

The selection of programming language is very important in building a simulator or any application programs. Therefore, it is a need to consider many advantages and disadvantages of several programming approaches. Here, procedural programming and object-oriented programming are both discussed.

2.2.1 Procedural Programming

In the early days of computing, programming was an extremely procedural process. The procedural languages placed code into blocks called procedures or function. Procedural program is written as a list of instructions, telling the computer, step-by-step, what to do: Open a file, read a number, multiply by 4, display something. Most traditional computer languages like Pascal, C and FORTRAN are procedural.

Procedural programming is fine for small projects. It is the most natural way to tell a computer what to do, and the computer processor's own language, machine code, is procedural, so the translation of the procedural high-level language into machine code is straightforward and efficient. What is more, procedural programming has a built-in way of splitting big lists of instructions into smaller lists: the functions [MONET].

The goal of each of these blocks was to act like a black box, which completed one task or another. The purist of this type of programming believed that one could always write

these functions without modifying external data. One of the difficult problems with this language method is to write all functions in such a way that they actually do not modify data outside their boundary. So, when functions began changing data outside their boundary (in C this is done by passing a pointer), a problem called coupling began to surface. Therefore, object-oriented programming language occurs. In a procedural-based programming language, a programmer writes out instructions that are followed by a computer from start to finish.

2.2.2 Object-Oriented Programming

Object-oriented programming (OOP) has its key component technologies – inheritance and polymorphism. Inheritance is a form of software reusability in which new classes are created from existing classes by absorbing their attributes and behaviors and embellishing these with capabilities the new classes require. Polymorphism is a character of assigning different meanings to a particular symbol or object, depending upon the context in which it is used. This allows objects to act differently within different situations, it enable the flexibility of a program design.

Objects are the central idea behind OOP. The idea is quite simple. A method is similar to a procedure. The basic idea behind an object is that of simulation. Most programs are written with very little reference to the real world objects the program is designed to work with; in object-oriented methodology, a program should be written to simulate the states and activities of real world objects. This means that apart from looking at data structures when modeling an object, we must also look at methods associated with that object, in other words, functions that modify the objects attributes.

The concept in which objects contain both data and methods is referred to as encapsulation. This could hide unimportant implementation details from other objects, which provides modularity as the source code for an object can be written and maintained independently of the source code for that objects. Similarly, one does not need to know how a class is implemented, but just to know which methods to invoke.

OOP approach has several key benefits. These are as follows:

- Extensibility – By modification of existing objects, new features can be added to the system where changes on new objects can be done.
- Maintainability – Maintenance and modification of objects can be done individually.
- Reusability – Objects, which are used in a system, can also be used in another newly built system with little or no changes.
- Simplicity – It is simple and less complex using the OOP approach while building programs, which attempts to model the objects interactions of the real world. Any changes are easy to modified with no much affect within the entire system.
- Modularity – Objects within the program are individual separate entities, the internal workings of which are isolated and de-coupled from other objects in the system. This solves the problem of coupling in procedural programming approach.

2.3 Programming Language

Java is just a small, simple, safe, object-oriented, interpreted or dynamically optimized, byte-coded, architecture-neutral, garbage-collected, multithreaded programming language with a strongly typed exception-handling mechanism for writing distributed, dynamically extensible programs. Java is developed by Sun Microsystems. It is a powerful programming language built to be secure, cross-platform and international [12].

As in a modern software development, Java is object-oriented from the ground up. The point of designing an object-oriented language is not simply to jump on the latest programming fad. The object-oriented paradigm meshes well with the needs of client-server and distributed software. Benefits of object technology are rapidly becoming realized as more organizations move their applications to the distributed client-server model.

An important characteristic that distinguishes objects from ordinary procedures or functions is that an object can have a lifetime greater than that of the object that created

it. This aspect of objects is subtle and mostly overlooked. In the distributed client-server world, it is possible to have potential for objects to be created in one place, passed around networks, and stored elsewhere, possibly in databases, to be retrieved for future work.

As an object-oriented language, Java draws on the best concepts and features of previous object-oriented languages, primarily Eiffel, SmallTalk, Objective C, and C++. Java goes beyond C++ in both extending the object model and removing the major complexities of C++. With the exception of its primitive data types, everything in Java is an object, and even the primitive types can be encapsulated within objects if the need arises.

In Java, only single inheritance is supported but multiple implementations of interface class is allowed [12]. Security and safety are main features of Java programming language. Its execution semantics guarantees that every run-time error is detected and reflected in a throw exception. Java eliminated the use of pointers of C++ and replaced it with references, which prevents program from accessing illegal areas of the system's memory. Besides, Java also supports dynamically run time method identification. Libraries of Java is supported through the use of packages and allow complicate programs to be build but the overhead of keeping track of all libraries is reduced.

Concurrency is very important in a simulation model as there might be many objects doing their own process at the same time. It is impossible to let them execute in a sequential methods, as this could not be appropriate as compared to real time simulation result. Most of the programming languages do not enable programmers to specify concurrent activities; rather they provide only simple set of control structures where one action is performed after one another [13]. In Java, programmer specifies that application contains threads of execution and the program may execute concurrently with other threads. These powerful capabilities are not available in C and C++. Instead, in C and C++ they have single-threaded languages.

2.4 Programming Tool

There are many types of Java programming tools available in market. One can develop Java programming using pure Java SDK [14] without the supporting of integrated development environment (IDE), or just selects a tool like Microsoft J++, Borland JBuilder, or Symantec Visual Café. This section describes about Borland JBuilder as the selected tool for development.

JBuilder is a group of highly productive tools for creating high-performance, platform-independent applications for Java. It is designed for all levels of development of projects, ranging from applets and applications that require networked database connectivity to client/server and enterprise-wide, distributed, mutli-tier computing solution.

The JBuilder IDE supports a variety of technologies including:

- 100% Pure Java
- JavaBeans
- Java 2
- Java SDK 1.2.2
- JFC/Swing

The additional technologies supported by JBuilder Professional edition are:

- Servlets
- Remote Method Invocation (RMI)
- Java Database Connectivity (JDBC)
- Open Database Connectivity (ODBC)
- All major corporate database servers

The additional technologies supported by JBuilder Enterprise are:

- Enterprise JavaBeans (EJB)
- JavaServer Pages (JSP)
- Common Object Request Broker Architecture (CORBA)

JBuilder also provides developers with a flexible, open architecture that makes it easy to incorporate new SDKs, third-party tools, add-ins, and JavaBean components [15].

2.5 Switching Approaches

ATM switching approaches can be categorized into three categories: shared memory approach, shared medium approach, and fully interconnected approach.

2.5.1 Shared Memory Approach

In the shared memory approach, the switching element convert the incoming cells from serial to parallel at the input port and convert them back to serial port when the cells leave the switching element at the output port. The sequence of the cell to be read into the memory is determined by the controller.

Shared memory approach has several advantages. It can achieve 100% throughput under heavy load (this is an output queuing approach). The size of buffers is minimized to achieve a specified cell loss rate since the shared memory can absorb a large burst of traffic directed to one output port. The approach, however, suffers from a few drawbacks. The memory must operate N times faster than the port speed since the read or write operation must be performed at one time. The controller needs to process the cell header and routing tag at the same speed as memory. This causes scalability problem of the switch and thus difficult for multiple priority classes, complicated cell scheduling, multicasting and broadcasting [16].

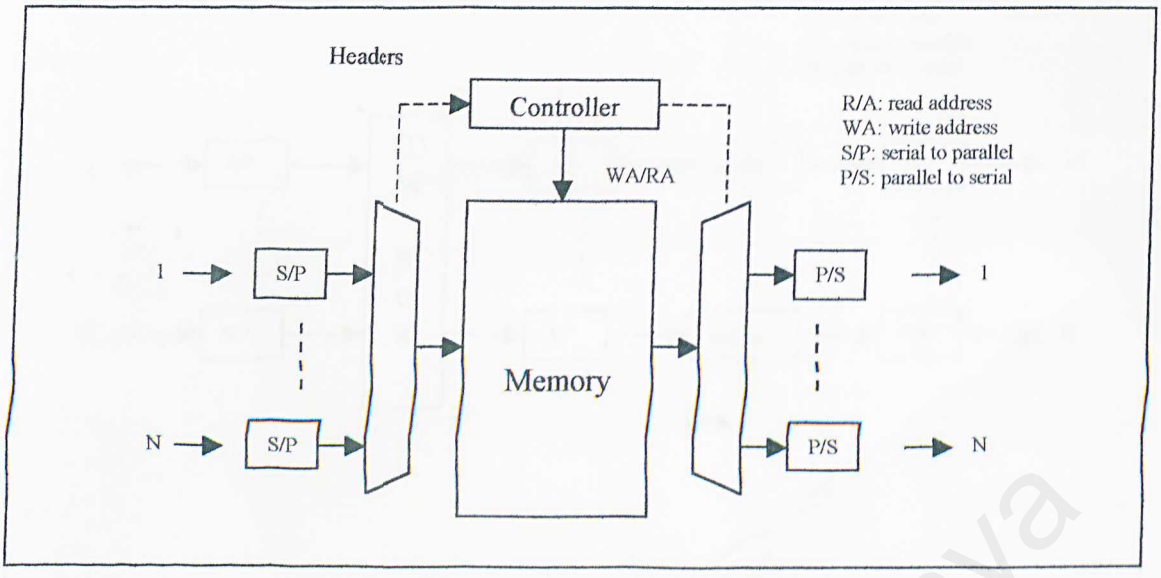

Figure 2.1: Shared Memory Approach

2.5.2 Shared Medium Approach

For shared medium approach, the incoming cells are converted from serial to parallel and then sequentially broadcast on the TDM bus in a round-robin manner. At each output, address filters pass the cells to the appropriate output buffers based on their routing tag. Like shared medium approach, the outgoing cells will be converted back to serial before passing to the external transmission path.

This approach has many advantages. Since the outputs are modular, the address filters and output buffers are easy to implement. The multicasting and broadcasting are quite straightforward due to the broadcast-and-select nature. On the other hand, the address filters and output buffers must operate at N times faster than the port speed and this limitation has placed a scalability problem for the approach. Another disadvantage of this approach is more buffers are needed since the memory is not shared [16].

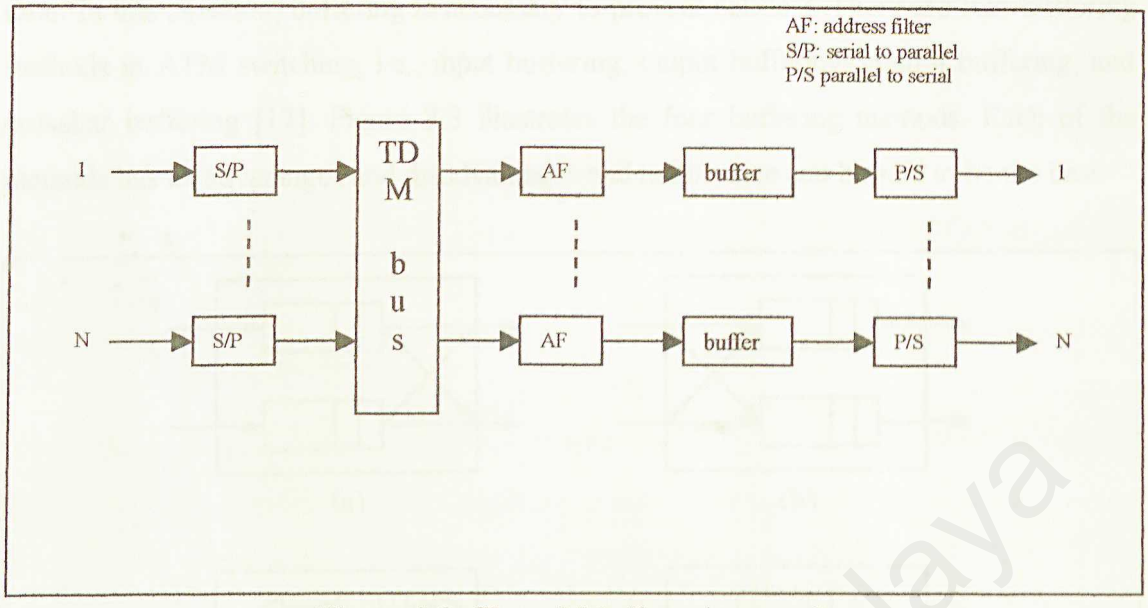

Figure 2.2: Shared Medium Approach

2.5.3 Fully Interconnected Approach

Fully interconnected approach is the easiest switching approach. Each input and output has an independent path and a total of N^2 paths exist for N input/output. All the arriving cells are broadcast on a separate bus to all outputs and address filters pass the appropriate cells to the output queues.

Fully interconnected approach has many advantages. As shared medium approach, all queuing occurs at the outputs. Address filters and output buffers are simple to implement and only need to operate at the same speed as the port. Hence it is scalable to any size and speed. In addition, multicasting and broadcasting are natural. Unfortunately, the quadratic growth of buffers limits the number of output port. Furthermore, fully interconnected approach needs a large amount of buffers [16].

2.6 Buffering Methods

Buffering is necessary to prevent certain queuing problems in ATM switch. For instance, two cells from different input ports may be addressed to same output port at the same

time. In this situation, buffering is necessary to prevent cell lost. There are four buffering methods in ATM switching, i.e., input buffering, output buffering, central buffering, and crossbar buffering [17]. Figure 2.3 illustrates the four buffering methods. Each of the methods has its advantages and disadvantages and neither one can be said to be the best.

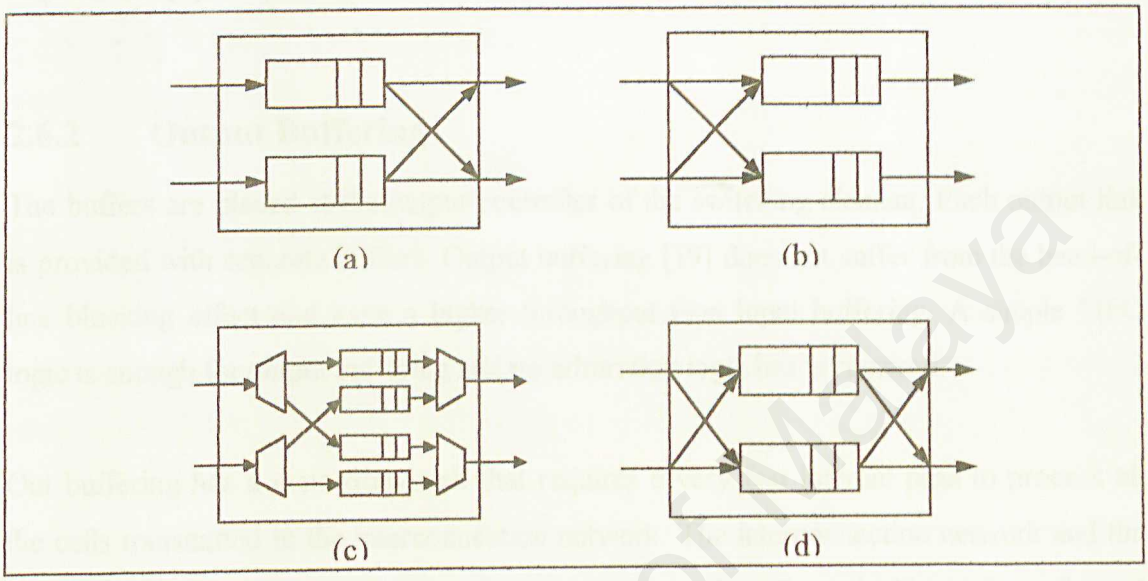

Figure 2.3: Buffering Methods: (a) Input (b) Output (c) Crossbar (d) Central [17]

2.6.1 Input buffering

Input buffering [18] is implemented with placing Input buffers at the input controller (IC). This method with simple first-in-first-out (FIFO) logic has many advantages. It is easy to implement in the sense that the internal links of the switching element have to operate at the same speed as the external input/output line. Hence, there is no requirement for internal speed up. Hardware complexity can be lower than other buffering schemes.

However, a collision may occur if more than one cell compete simultaneously for the same output. This will cause head-of-line blocking and all the cells behind the blocking head of the queue cell are blocked even though they are addressed to an idle output. An arbitration logic is needed to determine which of the first cells held in different input buffers need to be transmitted to the output port. The arbitration logic may be the simple

logic (round robin) or complex method (aiming to keep the same queue length in all the buffers). To solve this problem, random access memory (RAM) may be used instead of FIFO. If the first cell is blocked and the second cell was addressed to an idle output, then the second cell will be selected for transmission. Random access memory approach requires a complex queuing control to ensure the sequence of cell.

2.6.2 Output Buffering

The buffers are placed at the output controller of the switching element. Each output link is provided with separate buffers. Output buffering [19] does not suffer from the head-of-line blocking effect and have a higher throughput than input buffering. A simple FIFO logic is enough for output buffering and no arbitration logic has is required.

Out buffering has a main drawback that requires a very fast internal pass to process all the cells transmitted in the interconnection network. The interconnection network and the output buffer have to handle N cells at one time when there is N number of input controller. This higher speed increases the implementation complexity and cost of the switching element.

2.6.3 Crossbar Buffering

The buffers can also be located at the cross-points inside the switching element. Cells arriving at the inputs are enqueued in the appropriate buffer according to the destination tag. Buffers corresponding to a particular output of a switching element are served in round robin or some other predetermined logic.

This buffering scheme removes the blocking of packets by a packet addressed to a different output of the switching element. All packets arriving at the inputs of a switching element can be transferred to their target buffer within one clock cycle. Here, multicasting is simple to implement but significantly impacts the switch performance.

Crossbar also has the advantage of requiring only one read and one write operation on a buffer during a cycle.

The disadvantage from the performance point of view is that there are many small buffers, each of which is dedicated to a particular input/output par, and no buffer sharing is possible. Therefore, buffers cannot be used efficiently. The total required buffers are also much greater compare to other methods.

2.6.4 Central Buffering

In the central buffering approach, the buffers are shared between all the input and output controller. All the incoming cells will be directed to the central buffer and every output controller will identify the cell, which is addressed to it in a FIFO approach.

Central buffering method is the most efficient method since it only requires the smallest buffer size to minimize the cell loss in a heavy load condition. No head-of-line blocking in this method and optimal throughput/delay performance is achieved.

The disadvantages of this approach are fast memory element is required to allow all the incoming cells and outgoing cells access to the memory port at the same time. Besides, big complexity queuing management is also needed [16][20].

2.7 Queuing Models

The idea of queuing system is depicted as below: An item from some population of items arrives at the system to be served. If the server is idle, an item is served immediately. Otherwise the item joins a waiting line. When the server has completed serving an item, the item departs. If there are items waiting in the queue, one is immediately dispatched to the server.

Certain parameters associated with this queuing system are described as below:

- λ – number of arrivals per second.
- w – number of items waiting.
- T_w – waiting time in queue, including those that do not wait at all.
- T_s – server service time for each item.
- ρ – utilization, fraction of time that the server is busy, measured over some interval of time.
- q – number of items in the system, includes both item being served and waiting items.
- T_q – time that an item spends in the system, waiting and being served.

Certain key characteristics of the model must be chosen before deriving any analytic equations for the queuing model. The following are the typical choices in data communication context:

- Infinite item population. This means that the arrival rate is not altered by the loss of population.
- Infinite queue size. The queue size can grow without bound.
- FIFO dispatching discipline. The first cell of the queue will be selected to dispatch to the server.

2.7.1 Single-Server Queue

Figure 2.4 illustrates the single-server queue. No item will lose in the system if capacity of the queue is infinite; they are just delayed until they can be served. At $\rho = 1$, the server becomes saturated with 100% working of the time. The theoretical maximum input rate (arrival rate) that can be handled by the system is

$$\lambda_{max} = 1/T_s$$

However, the queues become very large near system saturation, growing without bound when $\rho = 1$. Practical considerations, such a response time requirements or buffer sizes, usually limit the input rate for a single server to 70%-90% of the theoretical maximum.

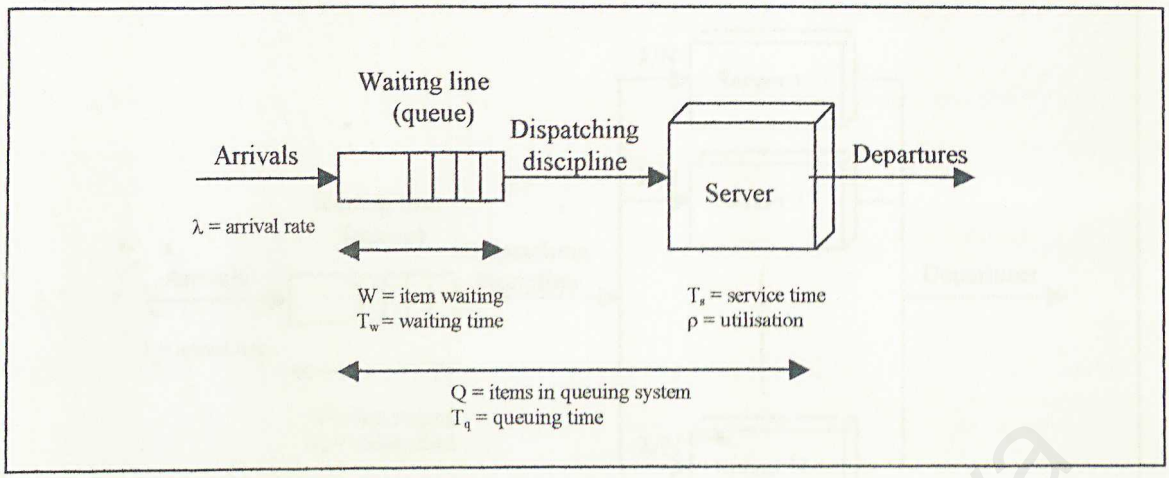

Figure 2.4: Single-Server Queue

2.7.2 Multiserver Queue

Figure 2.5 illustrates a multiserver queue where all servers sharing a common queue. If an item arrives and at least one of the servers is available, the item is immediately dispatched to that server. If none of the servers is available, the item will be kept in waiting line. As soon as one server becomes available, the item is dispatched from the queue to that server. It is assumed that all the servers are identical.

For multiserver queue, with N identical servers, ρ is the utilization of each server and therefore Nρ is the utilization of the entire system. The theoretical maximum utilization is N X 100% and theoretical maximum input rate is

$$\lambda_{max} = N/T$$

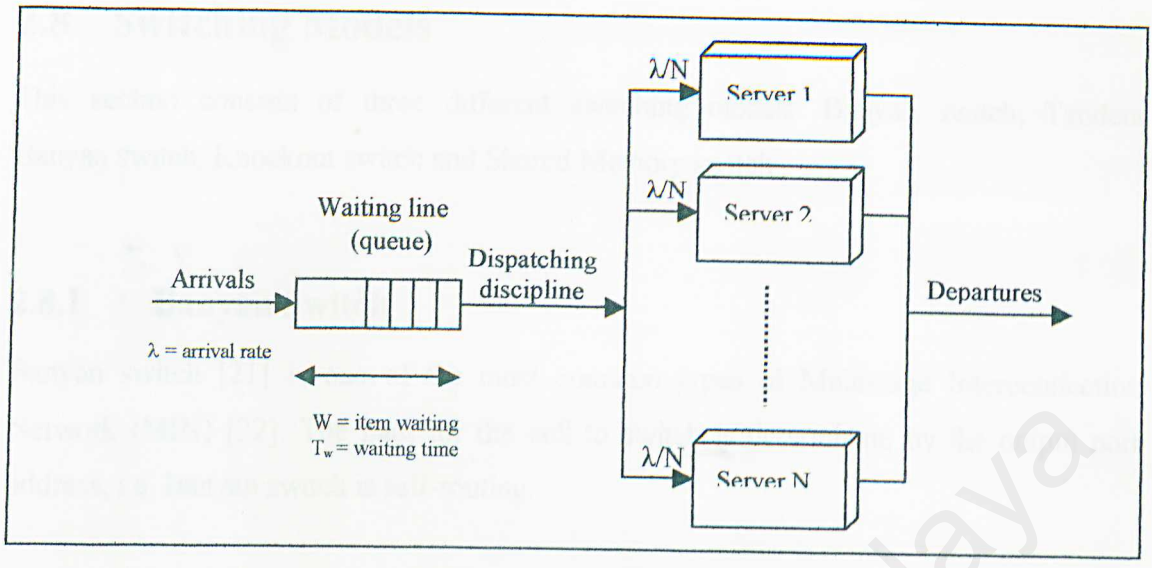

Figure 2.5: Multiserver Queue

2.7.3 Multiple Single-Server Queue

Multiple single-server queues can be considered as the "combination" of single-server queue and multiserver queue. This apparently minor change in structure has a significant impact on performance [2].

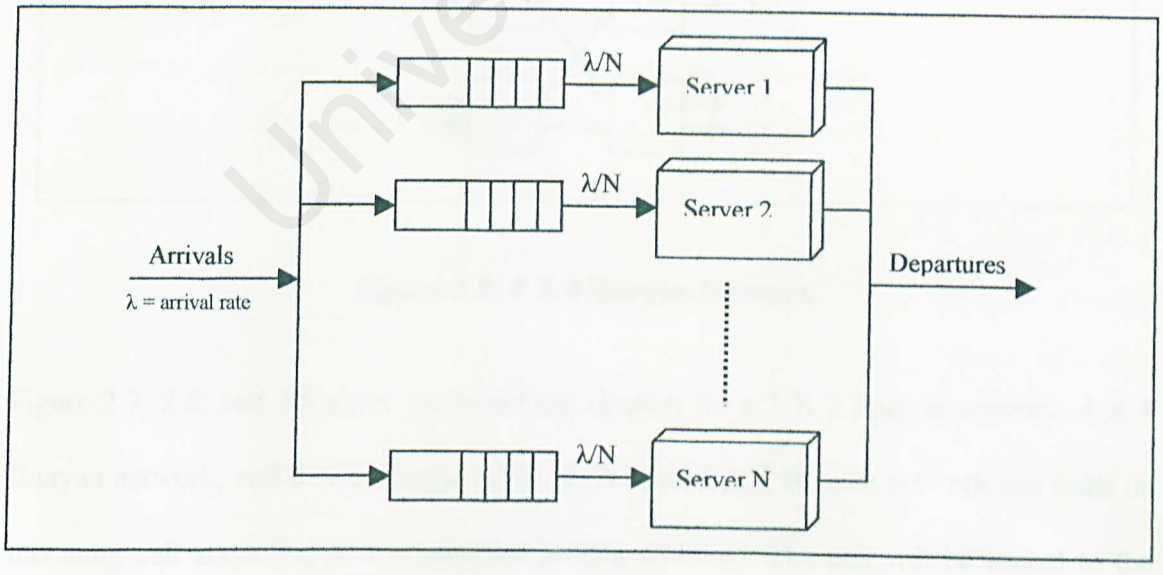

Figure 2.6: Multiple Single-Server Queue

2.8 Switching Models

This section consists of three different switching models: Banyan switch, Tandem Banyan switch, Knockout switch and Shared Memory switch.

2.8.1 Banyan Switch

Banyan switch [21] is one of the most common types of Multistage Interconnection Network (MIN) [22]. The path for the cell to switch is determinate by the output port address, i.e. Banyan switch is self-routing.

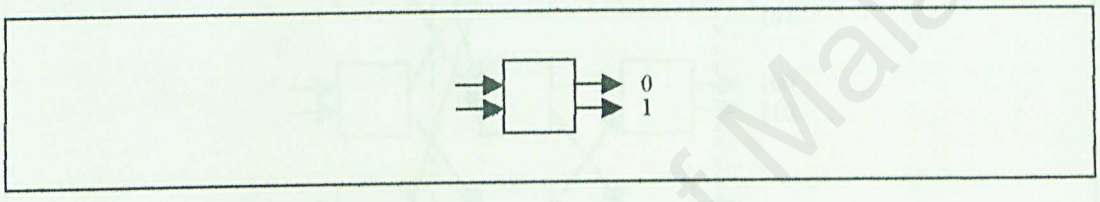

Figure 2.7: 2 X 2 Banyan Network

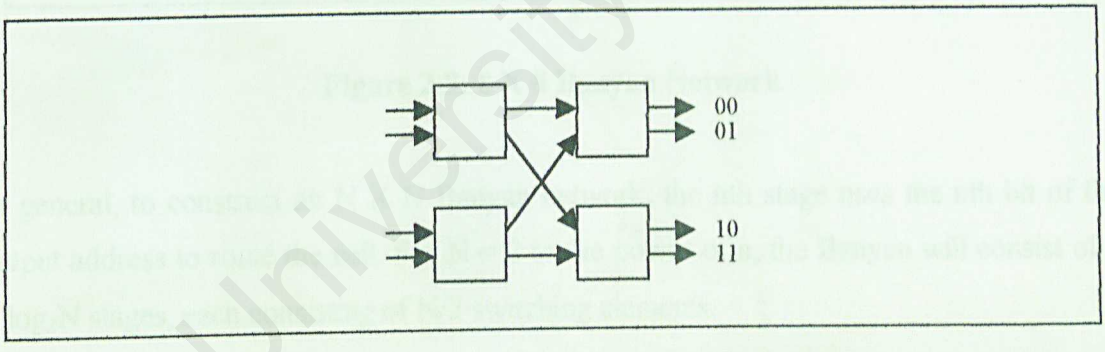

Figure 2.8: 4 X 4 Banyan Network

Figure 2.7, 2.8, and 2.9 show the switching element for a 2 X 2 Banyan network, 4 X 4 Banyan network, and 8 X 8 Banyan network. A basic 2 X 2 Banyan network can route an incoming cell according to a control bit (output address). The cell will be routed to the upper port address for the control bit 0 and to the lower port address if the control bit is 1.

Consider the 4 X 4 switching element, the interconnection of 2 stages of 2 X 2 switching elements can be done by using the first bit of the output address to denote which switching element to route, and then using the last 2 bits to route the cell through the 4 X 4 network to the appropriate output port.

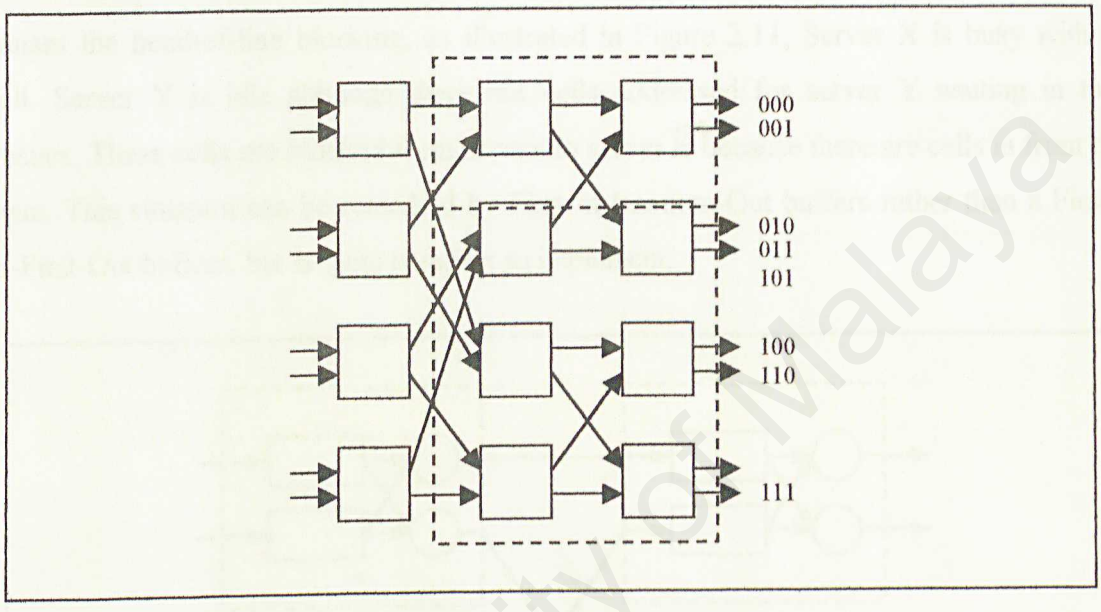

Figure 2.9: 8 X 8 Banyan Network

In general, to construct an N X N Banyan network, the nth stage uses the nth bit of the output address to route the cell. For N = 2 to the power of n, the Banyan will consist of n = $\log_2 N$ stages, each consisting of N/2 switching elements.

The switching in Banyan network is performed by simple switching elements, cells are routed in parallel, and all elements operate at the same speed. Another advantages of it are large switches can be easily constructed modularly and recursively and implemented in hardware. Bellcore's Sunshine switch and Alcatel Data Networks' 1100 are the examples that employ Banyan network technique [23].

Unfortunately, Banyan networks suffer from internal blocking. Two cells from different input, addressed to different output may contend before the last stage. So the overall

throughput is reduced. This problem can be solved by sorting the inputs according to their output destination and this approach lead to the classical Batcher-Banyan network.

Figure 2.10 shows the input queuing model for a 4 X 4 Banyan network. In the input queuing model, each server is connected to two queues. If the first cell in both queues is addressed to same server, the server chooses one cell randomly. However, this model causes the head-of-line blocking, as illustrated in Figure 2.11, Server X is busy with a cell. Server Y is idle although there are cells addressed for server Y waiting in the queues. These cells are blocked from access to server B because there are cells in front of them. This situation can be remedied by First-In-Random-Out buffers rather than a First-In-First-Out buffers, but is quite complex to implement.

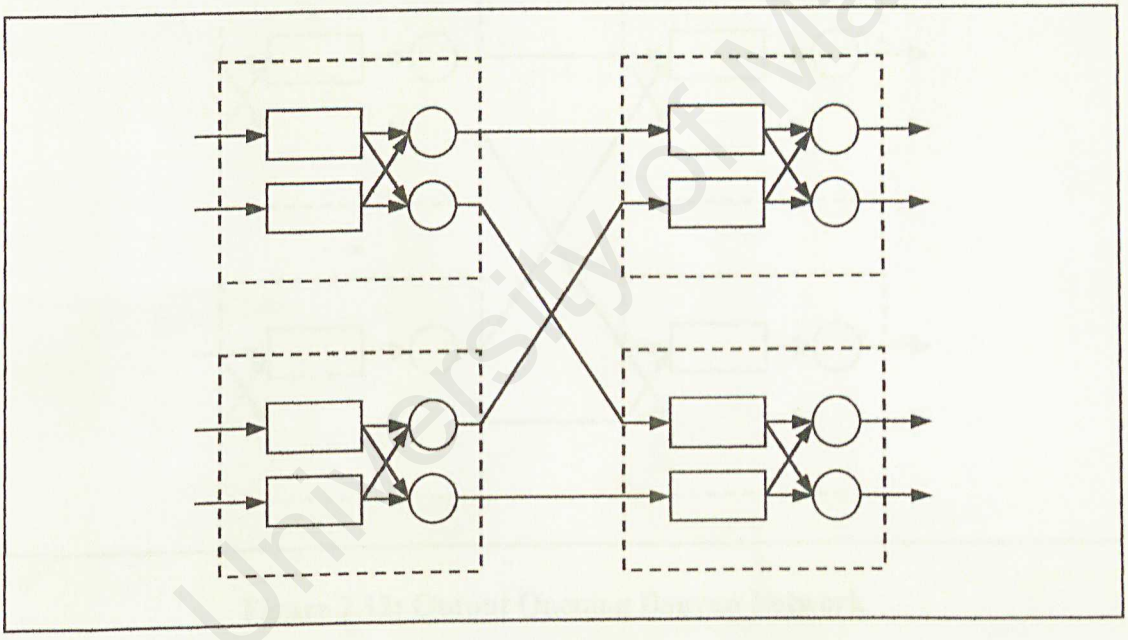

Figure 2.10: Input Queuing Banyan Network

The output queuing model, on the other hand, routes the cells directly to the appropriate queue, thus avoiding head-of-line blocking. However the output queuing model requires higher performance hardware and is more expensive to implement [4].

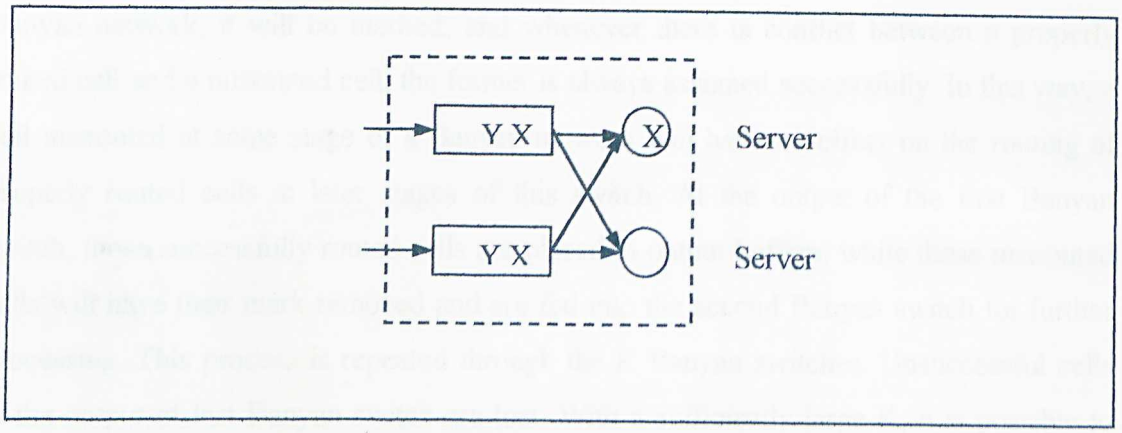

Figure 2.11: Head-of-line Blocking in Input Queuing Banyan Network

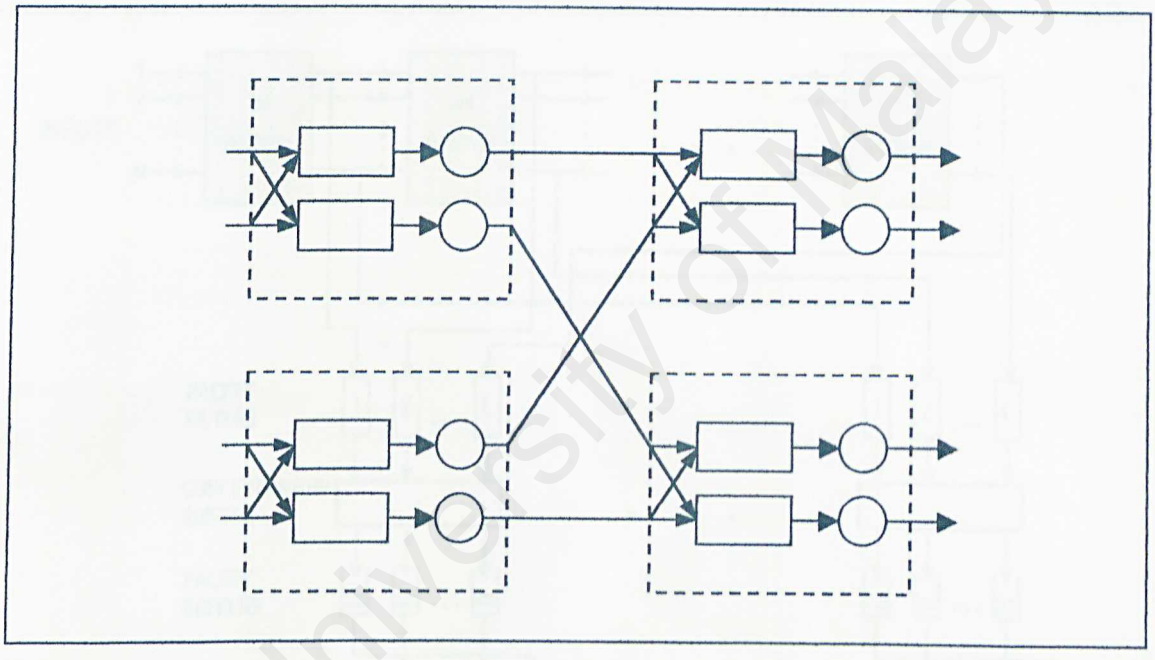

Figure 2.12: Output Queuing Banyan Network

2.8.2 Tandem Banyan Switch

Tandem Banyan switching fabric composes of multiple Banyan switches. These Banyan switches are arranged in series such that each output of every Banyan switch is connected to the input of next Banyan switch. Conflict occurs when two cells are addressed to the same outlet at the same switching element. One of these cells is scheduled correctly while the other is routed the "wrong" way. Furthermore, whenever a cell is misrouted in a

Banyan network, it will be marked; and whenever there is conflict between a properly routed cell and a misrouted cell, the former is always assigned successfully. In this way, a cell misrouted at some stage of a Banyan network will have no effect on the routing of properly routed cells at later stages of this switch. At the output of the first Banyan switch, those successfully routed cells are placed in output buffers, while those misrouted cells will have their mark removed and are fed into the second Banyan switch for further processing. This process is repeated through the K Banyan switches. Unsuccessful cells at the output of last Banyan switch are lost. With a sufficiently large K, it is possible to decrease the cell loss to the desired levels [24].

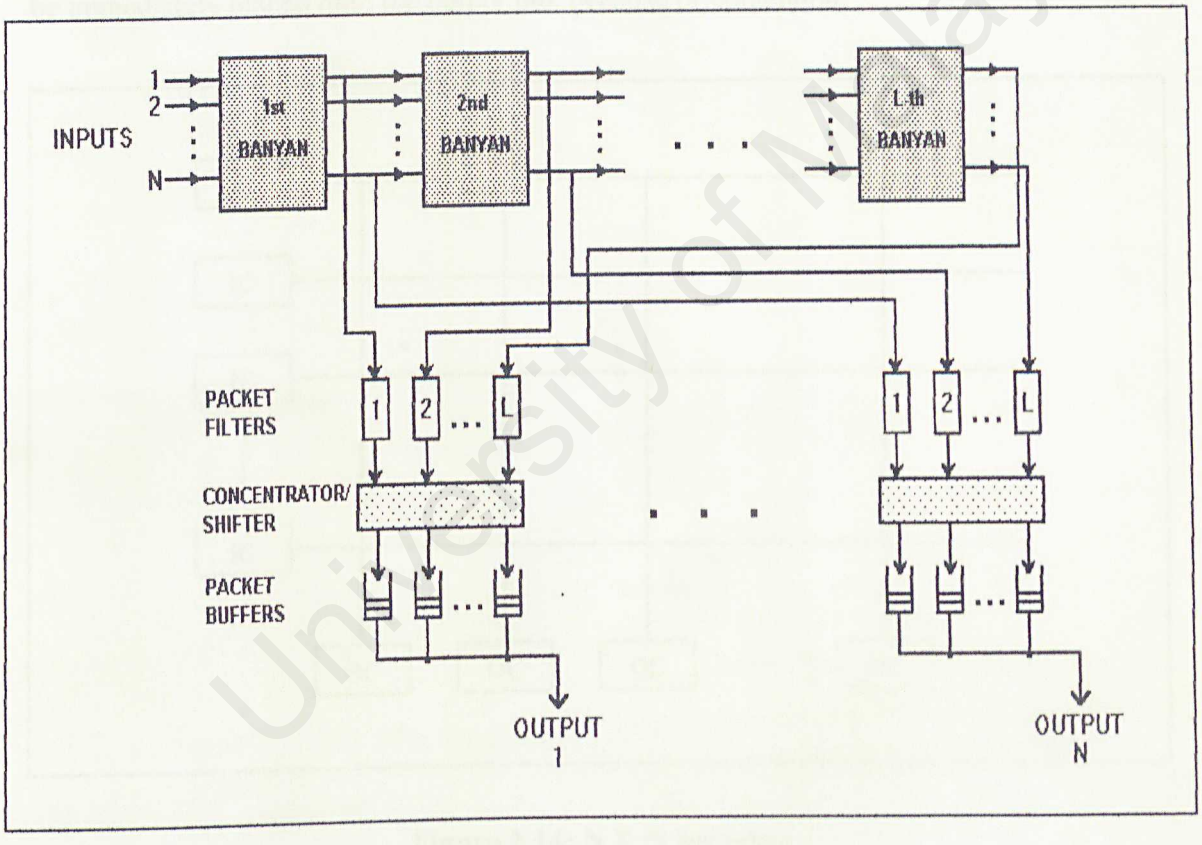

Figure 2.13: Tandem Banyan Switch

2.8.3 Knockout Switch

For ATM switch with a fully interconnected approach, the number of output queues will be N^2 for N X N switches. Knockout switch removes this requirement by adding a concentrator stage to the switch.

Refer to Figure 2.12. In the output queuing model, each fixed-length cell arriving at one of the input ports is placed on a broadcast bus from which each of the output modules taps the cells intended for itself. It is obvious that multicast and broadcast cells are really supported. The output module acts as a statistical multiplexer, deferring cells that cannot be immediately placed onto the output link because of contention.

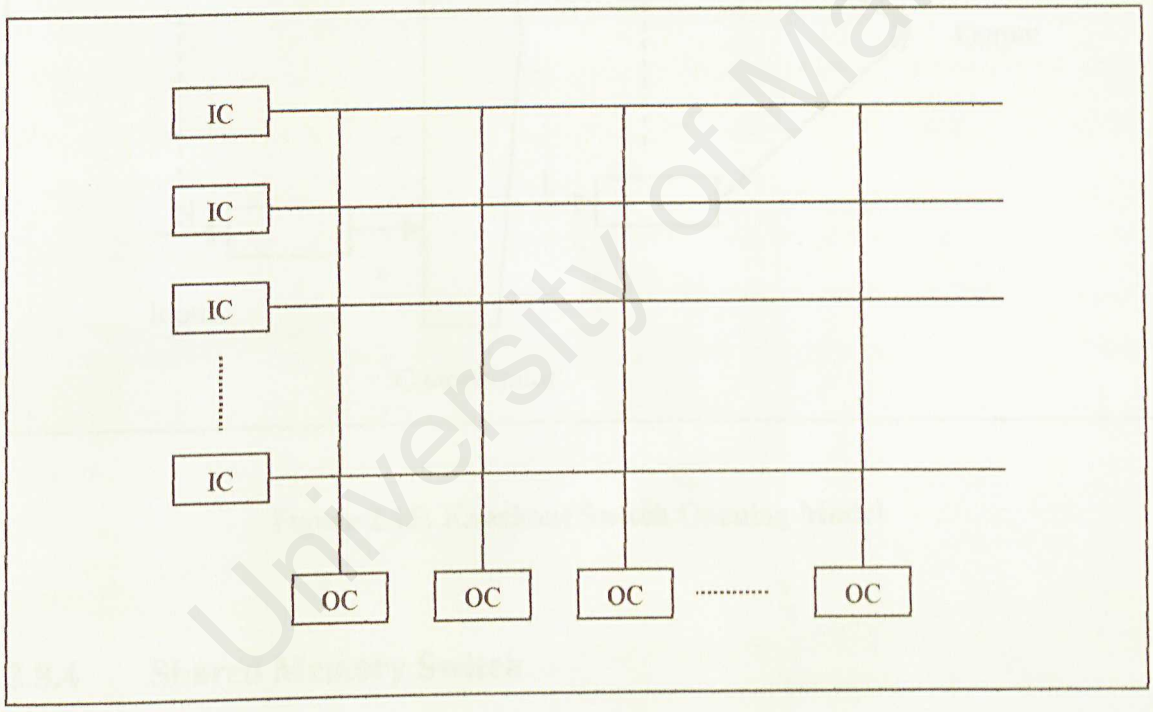

Figure 2.14: N X N Switches

For Knockout switch [25] queuing model, each input to an output module receives the cells broadcasted on the corresponding input bus. The job of each cell filter is simply to pass the cell to the concentrator if the cell is addressed for that output, and to mark the cell as inactive otherwise. The concentrator is to identify among its inputs those cells that are active and route them to its leftmost outputs, one cell per output line.

Assume that the concentrator has only L outputs and the number of inputs is N. If L or less than L cells arriving at the concentrator simultaneously, these cells can moved o the queues directly. If more than L cells arrive simultaneously, a "knockout tournament" is performed to select L of them will be process and the rest will lost in the switch [4].

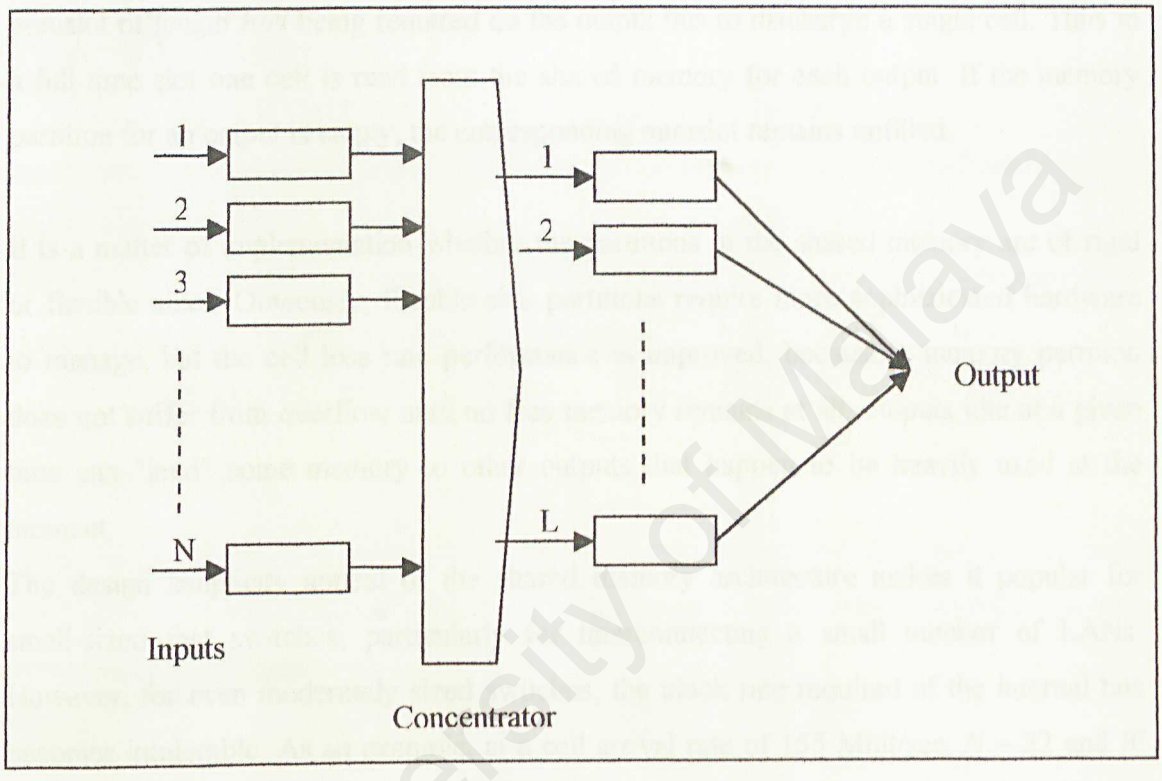

Figure 2.15: Knockout Switch Queuing Model

2.8.4 Shared Memory Switch

The distinctive feature of an $N \times N$ shared memory switch is its use of a high-speed internal bus, with a bit rate N times as large as the rate on each individual input/output line. For a time slot of length F, the internal bus is capable of transferring a cell in a minislot of length F/N. All cells received during a time slot are therefore transferred to the shared memory, albeit with a very short delay (probably during the very next time slot, if double-buffering technique is used within the serial-to-parallel converter). Conversion from serial to parallel is required to maintain acceptable clock rates; the bus

clock rate only needs to be N/W times higher than the incoming bit rate, with W the bus (and memory) width.

In the shared memory, the cells intended for each output are kept in separate partitions. During an output cycle, the cells are discharged to their outputs, again with only a minislot of length F/N being required on the output bus to discharge a single cell. Thus in a full time slot one cell is read from the shared memory for each output. If the memory partition for an output is empty, the corresponding minislot remains unfilled.

It is a matter of implementation whether the partitions in the shared memory are of rigid or flexible sizes. Obviously, flexible-size partitions require more sophisticated hardware to manage, but the cell loss rate performance is improved, because a memory partition does not suffer from overflow until no free memory remains at all; outputs idle at a given time can "lend" some memory to other outputs that happen to be heavily used at the moment.

The design simplicity appeal of the shared memory architecture makes it popular for small-sized fast switches, particularly for interconnecting a small number of LANs. However, for even moderately sized switches, the clock rate required of the internal bus becomes intolerable. As an example, at a cell arrival rate of 155 Mbit/sec, $N = 32$ and $W = 16$, the internal bus has to operate under a clock rate of 310 MHz.

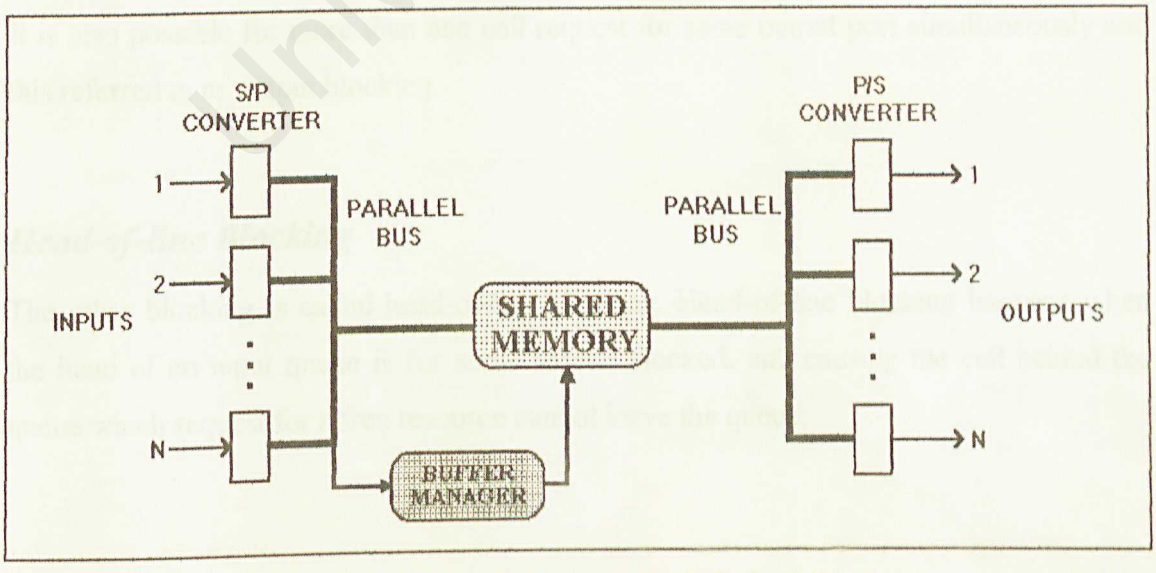

Figure 2.16: Shared Memory Switch Architecture

2.9 Switching Performance Issues

A cell may lose in transmission if less co-ordination among arriving cells as far as their destination requests are concerned and resource limitation within the switch. Certain performance issues might be considered for ATM switching.

Connection Blocking

Connection blocking occurs when a new connection cannot be accepted due to lack of resources. It is defined as the probability that not enough resources can be found to allow all the required physical connections between input ports and output ports at any time. Since ATM is connection-oriented, a logical connection must be found between the input ports and output ports after a connection was set-up.

Internal Blocking

In ATM switching, it is possible for two cells, addressed for different output ports compete for the same internal resource. This situation is called internal blocking. In this case, one cell will be blocked.

Output Blocking

It is also possible for more than one cell request for same output port simultaneously and this referred to as output blocking.

Head-of-line Blocking

The other blocking is called head-of-line blocking. Head-of-line blocking happens when the head of an input queue is for some reason blocked, and causing the cell behind the queue which request for a free resource cannot leave the queue.

Throughput

Throughput is defined as the average number of cells that are successfully delivered by the switch per cell-duration per output line.

Speed-up

The speed-up of a packet switch can be implemented in the space-switching domain or in the time switching domain. For space-switching domain. S disjoint paths are provided simultaneously to any output port. For time switching domain, the switch has a speed up factor S for the switch fabric to operate S times faster than the external lines.

Cell Loss Error

If too many cells in the switch have to be transmitted through the same link and the buffer to hold the cell is full, certain cells may be lost in the switching. The probability of a cell lost is defined as fraction of cells lost within the switch. Typical value is in the range of 10^{-10} to10^{-8}.

Cell Insertion Error

There is another possibility for a cell to be sent to the wrong logical connection. This error will cause one destination to miss a cell and the second destination to accept an additional. Switching element should always make the cell insertion error to be 1000 times better than a cell loss.

Switching Delay

Switching delay is the average time a cell spends in the switch from the time it arrives until the time it delivered to its requested line. A maximum delay in ATM switching has to be guaranteed for a low value of jitter. Typical delay values are between 10 and 1000 ultra-seconds.

Traffic Model

This refers to the traffic in the input ports. Many traffic models can be described in two random processes. The first process governs the arrival cells while the second process describes the distribution by which arriving cells choose their destination ports.

Multicast Connections

A point-to-multi-point connection is the situation where an incoming cell requests P output ports. A point-to-point connection has a value $P = 1$ and $P = N$ refers to broadcasting.

Other Issues

Other issues include the cell sequencing integrity for each input-output pair, scalability to large size and implementation complexity [16][4].

2.10 Summary

Object-oriented approach is the ideal approach due to its simplicity, modularity, modifiability, extensibility, maintainability, and reusability. The Java programming language was selected. This is because it is object-oriented and it contains the built-in support for multithreaded programming. Moreover, Java is robust compared to C++ as it has no pointer references to other data. In addition, Java was selected due to its portability, which makes it platform independent and supports web applications.

The preferred switching model is the Banyan model with input and output buffering. The implemented queue model is the single-server queue where each input buffer is connected to one of the corresponding inlet of the Banyan.

CHAPTER 3: SYSTEM ANALYSIS AND DESIGN

This chapter begins with a detailed illustration about ATM switch functions and switching architectures. The Switch Function is broken down into user plane, control plane and the management plane. The ATM switch architecture gives an overview of the whole ATM structure and also provides detailed explanation on the cell switch fabric where switching is actually performed. At the end, a description of the ATM traffic parameters, the simulation model, the JavaSim architecture, the system architecture, object-oriented and class design are discussed.

3.1 Switch Functions

In ATM switch, switching functions can be examined in the context of the three planes of the ATM model, i.e. user plane, control plane, and management plane.

3.1.1 User Plane

In user plane, the main function for ATM switch is to relay user data cells from input ports to the appropriate output ports. Only 5 bytes cell header will be processed by the switch and the 48 bytes payload is carried transparently. Virtual Path Identifier/Virtual Channel Identifier (VPI/VCI) is used to route the cells to the appropriate output ports. User plane function can be divided into three functional blocks: Input module, output module and the cell switch fabric

3.1.2 Control Plane

The main function in control plane is to establish and control the Virtual Path and Virtual Channel connections. Information in control cells payload is not transparent to the network. The switch identifies signaling cell, and even generates some itself. The Connection Admission Control (CAC) performs major signaling functions. Signaling information may/may not pass through the cell switch fabric, or may be exchanged through a signaling network such as SS7.

3.1.3 Management Plane

Major operations of management plane are fault management functions, configuration management functions, performance management functions, security management functions accounting management, and traffic management. These functions can be represented as being performed by the functional block Switch Management. The Switch Management is responsible for supporting the ATM layer Operations and Maintenance (OAM) procedures. OAM cells may be recognized and processed by the ATM switch [23].

3.2 ATM Switch Architecture

ATM switching architecture includes a few elements. There are input modules, output modules cell switch fabric, connection admission control, and switch management. This switching model is shown in Figure 3.1.

3.2.1 Input Module

ATM input module determines the incoming signal and extracts the ATM cell stream. This involves signal conversion and recovery, processing cell overhead, and cell delineation and rate decoupling. The following functions should be performed for input module:

- Header error checking using Header Error Control (HEC) field.
- Validation and translation of VPI/VCI.
- Determination of the destination output port.
- Passing signalling cell to Congestion Admission Control (CAC).
- Passing Operations and maintenance (OAM) cell to switch Management.
- Usage parameter control/network parameter control (UPC/NPC) for each VPC/VCC.
- Adding internal tag containing internal routing and performance monitoring information for use within the switch only.

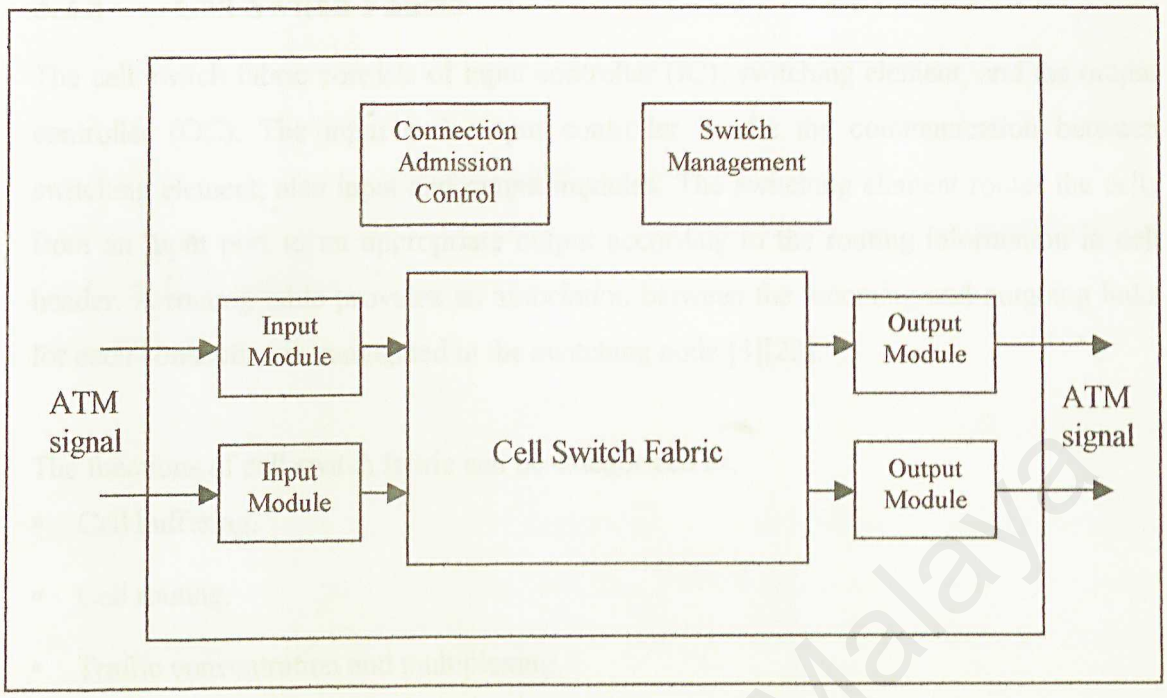

Figure 3.1: ATM Switching Model

3.2.2 Output Module

The function of output module is preparing AM cell streams for physical transmission. It is basically doing the reverse processes of the input module like:

- Removing and processing internal tag.
- Possible translation of VPI/VCI.
- Possible mixing of cells from Switch Management and CAC with outgoing cell streams.
- Cell rate decoupling.
- Mapping cells to appropriate payloads and generate the overhead.
- Conversion of digital bit stream to optical signal.
- Generating Header Error Control (HEC) field.

3.2.3 Cell Switch Fabric

The cell switch fabric consists of input controller (IC), switching element, and the output controller (OC). The input and output controller handle the communication between switching element, also input and output modules. The switching element routes the cells from an input port to an appropriate output according to the routing information in cell header. A routing table provides an association between the incoming and outgoing links for each connection is maintained in the switching node [4][23].

The functions of cell switch fabric can be categorized as:

- Cell buffering.
- Cell routing.
- Traffic concentration and multiplexing.
- Multicasting and Broadcasting.
- Redundancy for fault tolerance.
- Cell scheduling base on delay priorities
- Congestion monitoring and activation of Explicit Forward congestion Indication (EFCI).

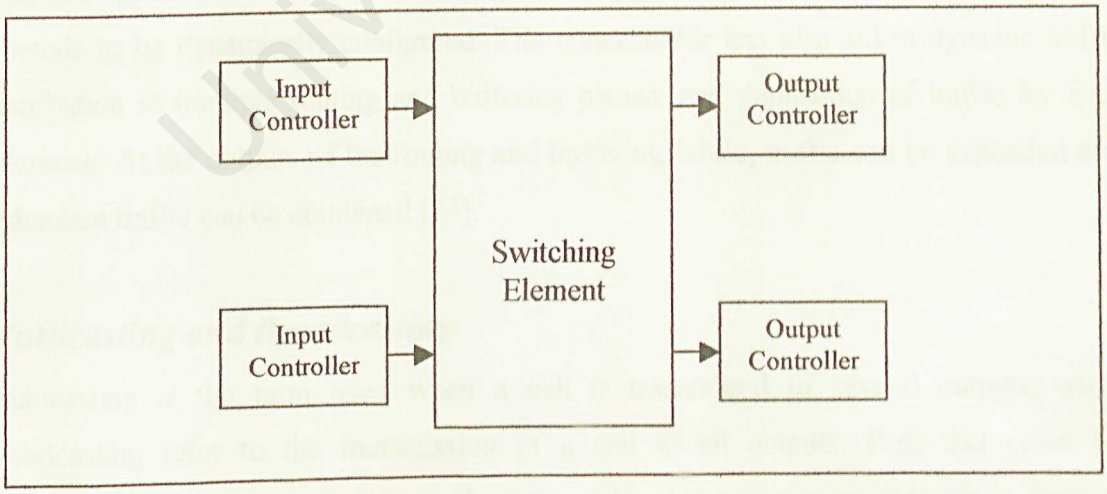

Figure 3.2: Cell Switch Fabric

Cell Buffering

In ATM switch, arriving cell may be aligned in time by means of singe-cell buffers. Cell buffering may be necessary since the cells may be addressed to the same output simultaneously. The switch may buffer cells in input controller, cell switch fabric, output controller, or use a combination of all. Output buffering is desirable from a performance point of view; the hardware requirements on this method are much larger than input buffering [4][23].

Cell Routing

Cell routing mechanism transfers the cells from the input port to the output port. The input module attaches a routing tag to each cell, and the switch fabric routes the arriving cells from its input port to the appropriate output port. It may be necessary to discards. Cell transfer can be achieved with one large switching element or with several smaller interconnected switches [4][23].

Concentration and Multiplexing

Traffic needs to be concentrated at the inputs of the switching fabric to better utilize the incoming link connected to the switch. The concentrator aggregates the lower variable bit rate traffic into higher bit rate for the switching matrix to perform the switch at standard interface speed. The concentration is highly correlated with the traffic characteristics, so it needs to be dynamically configured. The concentrator can also aid in dynamic traffic distribution to multiple routing and buffering planes, and duplication of traffic for fault tolerance. At the outputs of the routing and buffering fabric, traffic can be expanded and redundant traffic can be combined [23].

Multicasting and Broadcasting

Multicasting is the term used when a cell is transmitted to several outputs, while broadcasting refer to the transmission of a cell to all outputs. Both can either be performed directly in the switching elements or by inserting a copy network in front of

the switch fabric. A copy network makes several copies to the multicasted/broadcasted cell and assures that the copies are routed to the appropriate outputs [4].

3.2.4 Connection Admission Control (CAC)

Connection Admission Control responsible for major signaling functions including establishes, modifies, and terminates virtual path/ virtual channel connections. The functions can be illustrated as below.

- High level signalling.
- Signalling ATM Adaptation layer (AAL) functions to interpret or generate signalling cells.
- Interface with a signalling network.
- Renegotiations with users to change established VPCs/VCCs.
- Allocation of switch resources for VPCs/VCCs, including route selection.
- Admission/rejection decisions for requested VPCs/VCCs.
- Generation of UPC/NPC parameters.

CAC can be either placed centralized or distributed to the blocks of input modules. In the former case, a single processing unit would receive signaling cells from the input modules, interpret them, and perform admission decisions and resource allocation decisions for all connections in the switch. In the later case, all the CAC located at each input modules has a smaller number of input ports. This approach divides the job among the various CACs and performs them in parallel, thus solves the connection processing bottleneck problem. However, it is more difficult to implement compare to centralized CACs.

3.2.5 Switch Management

Functions of Switch management are:

- Handle the physical layer OAM, ATM layer OAM.
- Configuration management of switching elements.
- Security control for the switch database.
- Usage measurements of the switch resources, traffic management.
- Administration of a management information base.
- Customer-network management, interface with operations systems.
- Support of network management.

Like CAC, switch management might be centralized or distributed among input modules. Again, a distributed switch management solves the performance bottleneck problem but a lot of co-ordination will be required. Each distributed input module switch management unit can monitor the incoming user data cell streams to performance accounting and performance measurement. Output module switch management units can also monitor outgoing cell streams [23].

3.3 ATM Traffic Parameters

When an ATM source sends ATM cell traffic to its corresponding ATM destination over the network, the traffic characteristics are described or specified by source traffic parameters. These traffic parameters are the values that can indicate the nature of the source-traffic characteristics and they includes *Peak Cell Rate (PCR)*, *Sustainable Cell Rate (SCR)*, *Maximum Burst Size (MBS)*, *Minimum Cell rate (MCR)*, and others.

3.3.1 Peak Cell Rate (PCR)

PCR defines an upper bound on the traffic that can be submitted by a source on an ATM connection. PCR is equal to the inverse of the minimum cell inter-arrival time T:

$$PCR = 1/T$$

3.3.2 Sustainable Cell Rate (SCR)

SCR is the average maximum rate which measurement based on a longer time scale than used for PCR. SCR is needed to specify a VBR source. It enables network to allocate resources efficiently among a number of VBR sources without dedicating the amount of resources required to support a constant PCR rate. The SCR is only useful if SCR < PCR.

3.3.3 Maximum Burst Size (MBS)

MBS is the maximum number of cells that can be sent continuously at the peak cell rate. If cells are presented to the network in clumps equal to the MBS, then the idle gap between clumps must be sufficient so that the overall rate does not exceed the SCR.

3.3.4 Minimum Cell Rate (MCR)

MCR defines the minimum commitment requested of the network. It is used with the ABR service. The quantity (PCR – MCR) represents an elastic component of data flow for which the network provides only the assurance that this capacity will be shared fairly among the ABR flows.

3.4 Switch Architecture: Impact on Traffic Handling

The performance an application requires from ATM network can be defined in terms of the following parameters:

- Throughput – bits per second delivered to the application.
- Latency – total of the transmission delay, propagation delay, and queuing delay through each network element or switch.
- Jitter – variation of delay, or the variation in the intercell arrival of consecutive cell. Some applications are very sensitive to jitter such as voice.

- Cell loss – cell loss already described in section 1.1.1. TCP data services can recover from packet loss and use it for gaining information on the congestive state of the network. Nevertheless, packet retransmission caused by cell loss can seriously affect application throughput [25].

The design of the ATM switch used to create ATM network will have a major impact on the network's ability to support simultaneously the different ATM service class.

Throughput/Cell Lost

For a switch to ensure that it maximizes throughput and minimizes the cell loss, a non-blocking design should be used. A switch must not loose any cells in the traffic switched to each output. If the traffic is less than or equal to physical capacity of the port, a non-blocking switch should exhibit no loss when handling full rate input load if the traffic pattern is congested.

It is also important that the traffic on a congested port does not interfere with traffic on a non-congested port. Input buffered switches can suffer what is known as head-of-line blocking.

Latency/Jitter

Many applications require that latency added by the switch be minimized. Low latency also delivers low jitter for those applications sensitive to jitter. Figure 3.3 illustrates the latency through a switch is significantly influenced by the switches internal design.

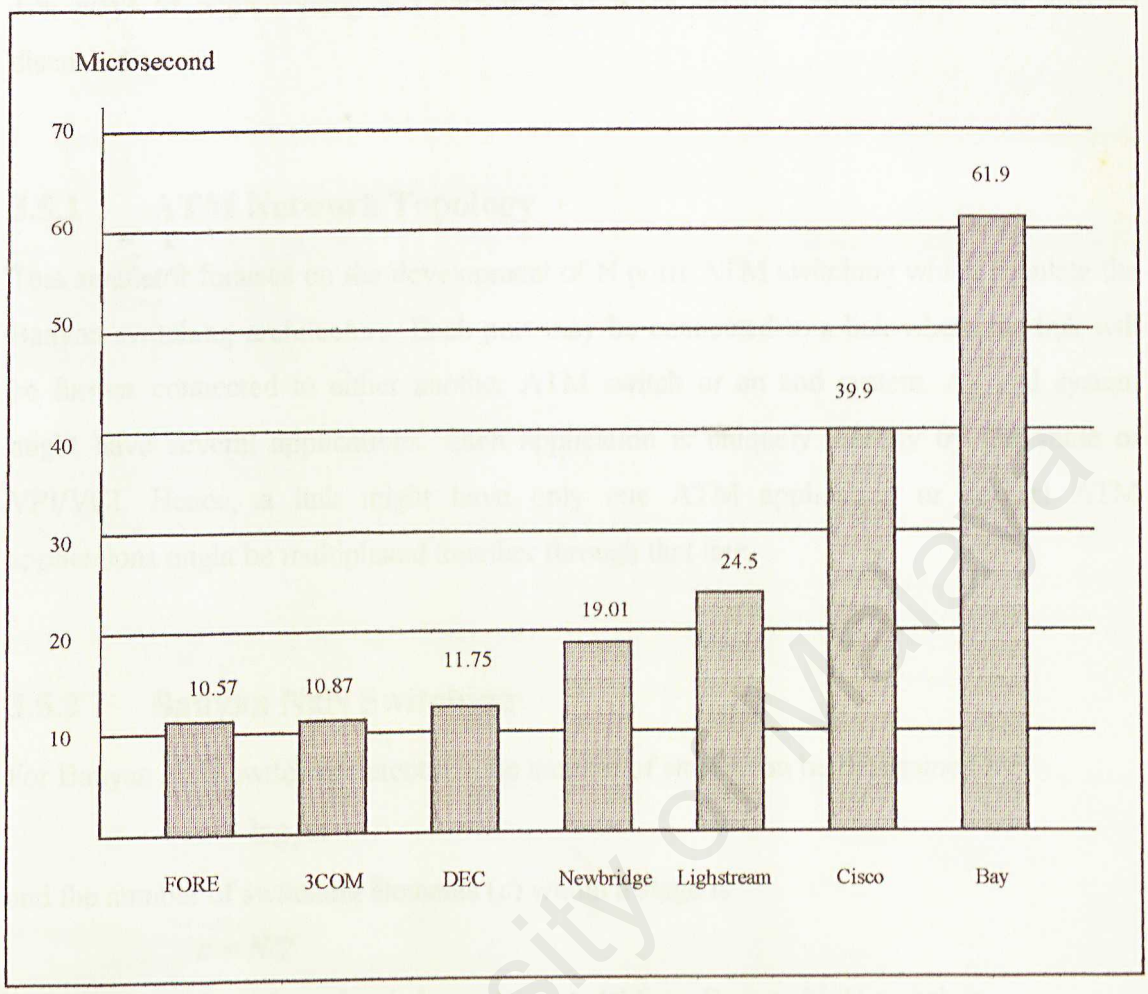

Figure 3.3: Latency per Port [26]

3.5 Switching Model

This subsection gives an idea regarding the whole simulation environment for this project. ATM Network Topology describes how ATM switch is connected in the ATM network environment. Besides, it also gives a description on the basic simulator model: Banyan Switching.

On top of that, this model describes the Buffering in ATM switch and the decision for selecting the cell to transmit when there are two cells addressed to same output outlet.

The ways of implementing multithreading in ATM network environment will also be discussed.

3.5.1 ATM Network Topology

This simulator focuses on the development of N ports ATM switching which simulate the Banyan switching architecture. Each port may be connected to a link where the link will be further connected to either another ATM switch or an end system. An end system might have several applications. Each application is uniquely identify by the value of VPI/VCI. Hence, a link might have only one ATM application or several ATM applications might be multiplexed together through that link.

3.5.2 Banyan NxN Switching

For Banyan NxN switch architecture, the number of stages can be determined by

$$s = \log_2 N$$

and the number of switching elements (c) within a stage is

$$c = N/2$$

Finally, the total number of switching elements (t) for a Banyan NxN switch is

$$t = sc$$

This simulation model simulates Banyan NxN switch. For example, in Banyan 4x4 switch, it consists of 2 stages, and each stage consists of 4 input buffers. The incoming ATM cells will be first placed into the input buffers for switching element in stage 1, these cells will be switched to the intermediate buffers between switching elements in stage 1 and stage 2. At the output of the last stage, there are 4 output buffers where all the switched cells will be placed in here before transmitted through the link component. For Banyan 8x8 switch, it consists of 3 stages and each stage consists of 8 input buffers. Again, all the incoming cells will be placed in those buffers before internal switching is performed within an individual switching element. Finally, all the switched ATM cells will be placed in the output buffers before transmitted to the link components.

3.5.3 Buffering

As mentioned in the sub-section above that a switching element consists of two input buffers and two output buffers. Input buffer and output buffer for switching element at stage n+1 is the output buffer of switching element at stage n and input buffer for switching element at stage n+2. Cells inserted into buffer will be placed in First Come First Serve (or first in first out, FIFO) basis and the cell selected from each buffer is the head-of-line cell of that buffer.

The buffer described is a fix size buffer. All the buffers within an ATM switch have same size. Discarding of cell only happen at input buffers and output buffers to the Switch (not switching elements), which is connected to the outside links. This situation occurs when input buffer to switch is full while the incoming link still sending in the cell or the outgoing link is too slow when the output buffer is full.

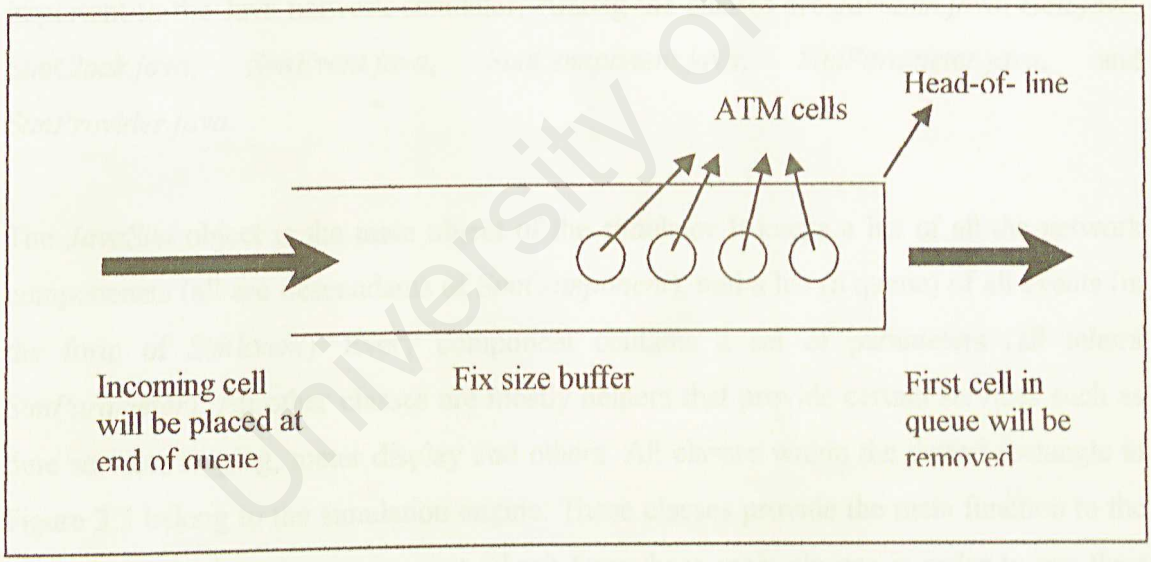

Figure 3.4: First Come Fist Serve Buffer

3.5.4 Switching

A routing table will be created once the network topology is set. All the information for ATM applications, which will be switched through that ATM switch, will be stored in this table. When performing switching for an ATM cell, the output port for that cell will

be determined by that routing table. For switching within a switching element, the output buffer will be checked first. If the buffer is full, the operation will be stopped.

3.5.5 Multithreading

One of the interested part for this model is multithreading. Every component in the network is a single thread and all of them can run simultaneously. The speed of operation for each component is controlled under the general clock. For example, there might be two switches existing in the network environment and both of them might have different speeds.

3.6 JavaSim Architecture

This section will discuss on the object-oriented design and main classes that are important to the Java network simulator. Among the classes are *JavaSim.java*, *Cell.java*, *SimClock.java*, *SimEvent.java*, *SimComponent.java*, *SimParameter.java*, and *SimProvider.java*.

The *JavaSim* object is the main object of the simulator It keeps a list of all the network componenets (all are descendants of *SimComponent*), and a list (a queue) of all events (in the form of *SimEvent*). Every component contains a set of parameters (all inherit *SimParameter*). All other classes are mostly helpers that provide certain services such as time service, logging, meter display and others. All classes within the dotted rectangle in Figure 3.5 belong to the simulation engine. These classes provide the main function to the simulator and other classes can just inherit from these main classes in order to use their service.

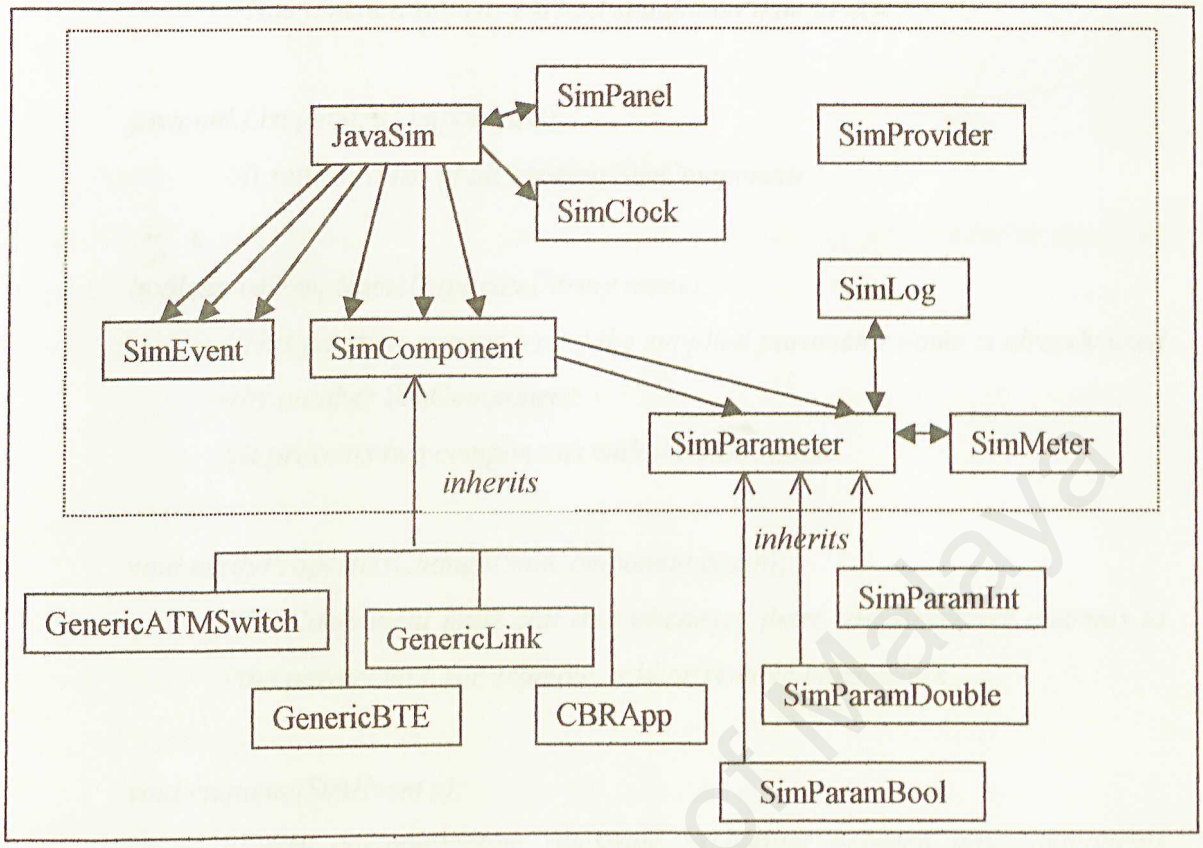

Figure 3.5: Hierarchy of All the Significant Objects in the Simulator

JavaSim.java

The *JavaSim* class is the main class that provides the important function to the simulation engine. This functions include:

- It is the main object that contains everything in the simulator.
- It provides all Graphical User Interface (GUI) functions.
- It provides the main JFrame for the application so that user can use it as a workspace to create the network topology. Closing the JFrame will exit the simulator program.
- It provides the event manager to handle event-passing among all components.

There will be only one instance of the *JavaSim* object throughout the simulation. Anyone with a reference to this instance can make use of the following services:

long now();

```
        //This function returns current simulation time in tick

    java.util.List getSimComponents();
        //It returns a list of all existing SimComponent

    boolean isCompNameDuplicate(String name);
        //This function returns true if the supplied parameter name is already used
        //by another SimComponent
        //It prevents two components with the same name

    void notifyPropertiesChange(SimComponent comp);
        //SimComponent must call this whenever there are structural changes to
        //the parameters, for example, add or remove parameters

    void enqueue(SimEvent e);
        //Every communication (message exchange) between any components
        //must involve creation of a SimEvent and a call to the enqueue() method
```

The above methods can only be called by objects that have a reference to the main *JavaSim* object.

Cell.java

The *Cell* class is a data resource class used by components like *CBRApp*, *GenericBTE* and *GenericATMSwitch* throughout the simulator. As a result, it contains attributes needed by the operation of all executor classes.

The main parameters in *Cell* class are:

```
int vpi=0; //Virtual path identifier
int vci=0; //Virtual channel identifier
```

At the initial stage, *vpi*, and *vci* values are set to zero.

SimClock.java

The *SimClock* class provides a set of time translation functions for normal translation between tick and actual time (in microseconds, milliseconds and seconds). It is an important class to synchronize the time throughout the simulation process. The functions provided by *SimClock* class are as follow:

```
static double Tick2Sec(long tick); //Converts ticks to seconds
static double Tick2MSec(long tick); //Converts ticks to milliseconds
static double Tick2USec(long tick); //Converts ticks to microseconds
static long Sec2Tick(double sec); //Converts seconds to ticks
static long MSec2Tick(double msec); //Converts milliseconds to ticks
static long USec2Tick(double usec); //Converts microseconds to ticks
static double getSec(long tick); //Returns current time in seconds
static double getMSec(long tick); //Returns current time in milliseconds
static double getUSec(long tick); //Returns current time in microseconds
```

SimEvent.java

Every *SimComponent* communicates with each other by enqueuing *SimEvent* for the target component. For example, when component A wants to send a packet to component B, component A creates a *SimEvent* that specifies B as its destination, and enqueue the event. The SimEvent object also contains a time so that this event is fired at exactly the specified time. Component B will then be able to react to the event accordingly.

The constructor for *SimEvent* class is shown below:

```
SimEvent(int aType,SimComponent src,SimComponent dest,long aTick,Object []
        params);
//The constructor needs an event type (as defined in SimProvider or a
private event type), the source and destination SimComponent, a time (in
ticks), and an array of java.lang.Object (which can be anything) holding
various parameters for the event
```

//The event type determines what the array will contain

Upon receiving the *SimEvent* object, its content can be retrieved by the following functions:

int getType(); //Get the event type

SimComponent getSource(); //Get the source SimComponent

SimComponent getDest(); //Get the destination SimComponent

long getTick(); //Get the time (in ticks) to fire the events

Object [] getParams(); //Get the event's parameters

SimComponent.java

SimComponent class is the most important class in the simulator in order to develop new components. Network components like *GenericATMSwitch*, *GenericLink*, *GenericBTE*, and *VBRApp* inherit from *SimComponent*.

This class provides the skeleton for an actual component. A new component should extend *SimComponent* and override its various methods in order to provide meaningful operations for the component. The constructor for *SimComponent* class is shown below:

SimComponent(String aName,int aClass,int aType,JavaSim aSim,Point loc);

//Every new component must provide a constructor with exactly the above parameters and the super(aName,aClass,aType,aSim,loc)function is immediately called as the first statement of the method in order to override its parameters

The *SimComponent* also provides a set of functions according to its types of operations. Among the operations is neighboring operation, copy operation, initial/reset operation and event handler operation.

The functions in neighboring operation are *addNeighbor*, *removeNeighbor*, *removeNeighbors* and *isConnectable*. Any component that needs to handle neighbor connect/disconnect operations should override these methods.

void addNeighbor(SimComponent comp);

//The simulation engine calls this function when a new neighbor is connected to this component

void removeNeighbor(SimComponent comp);

//The simulation engine calls this function when a neighbor is disconnected from this component

void removeNeighbors(java.util.List comps);

//The simulation engine calls this function when a group of neighbors is disconnected from this component

boolean isConnectable(SimComponent comp);

//The simulation engine calls this function when a new component is about to be connected to this component

//This function checks whether two components can be connected together

//The connection rules are:

//1) Application (CBRApp) only allowed to connect to BTE

//2) BTE only allowed to connect to Application (CBRApp) and GenericLink

//3) GenericLink only allowed to connect to GenericBTE and GenericATMSwitch

//4) GenericATMSwitch only allowed to connect to GenericLink

The only function in copy operation is copy.

void copy(SimComponent comp);

//This method is used to copy parameter values of another SimComponent of the same type. This method must be override in order to ensure that all necessary parameter values are copied

The functions in initial/reset operation are reset and start.

void reset();

//This function brings the status of the component back to the same status as if it is just newly created

void start();

//This function starts the simulation when the user clicks the "Start" button

The function in event handler operation is action.

void action(SimEvent e);

//This is the event handler of this component, and will be called by the simulator engine whenever a SimEvent with this component as the destination fires.

SimParameter.java

Classes like *SimParamInt*, *SimParamDouble* and *SimParamBool* inherit from *SimParameter*. These classes provide support for integer, double and boolean parameters. Other types of parameters can be created by extending *SimParameter* accordingly. By extending *SimParameter*, these classes can obtain parameter logging and meter display features automatically. The constructor for *SimParameter* class is shown below:

SimParameter(String aName,String compName,long creationTick,boolean isLoggable);

//The parameters for SimParameter constructor are:

//aName – name of the parameter

//compName – name of the component that owns the parameter

//creationTick – time when the parameter is created

//isLoggable – whether the parameter can be logged in the log file

SimProvider.java

The *SimProvider* object defines all the public events (this is the only part of the simulation engine that requires recompilation in order to allow development of new *SimComponent* and event types). All private events should be defined within the particular *SimComponent* source itself. All private events must be greater than a constant (*SimProvider.EV_PRIVATE*) defined in *SimProvider*. So, the first private event should have a value of *SimProvider.EV_PRIVATE + 1*, the next *SimProvider.EV_PRIVATE + 2*, and so on.

Every *SimComponent* must have a component class (not to be confused with the Java class) and a component type. *SimProvider* class creates two methods that can be used to obtain the component class and type of any *SimComponent* as shown below.

```
private static final String [] classes={
  "ATM Switch",
  "BTE",
  "Link",
  "Application",
};

static final int EV_SELFTEST=0;
static final int EV_RECEIVE=1;
static final int EV_READY=2;
static final int EV_PRIVATE=100;
        //Event type constants

final int getCompClass();
        //Get the component class (e.g. switch, BTE, link etc.)
final int getCompType();
        //Get the component type (further division under a component class)
```

3.7 System Architecture Design

The ATM simulator is created base on the Banyan NxN technique. Banyan 4x4 will be used as an example to illustrate the design architecture.

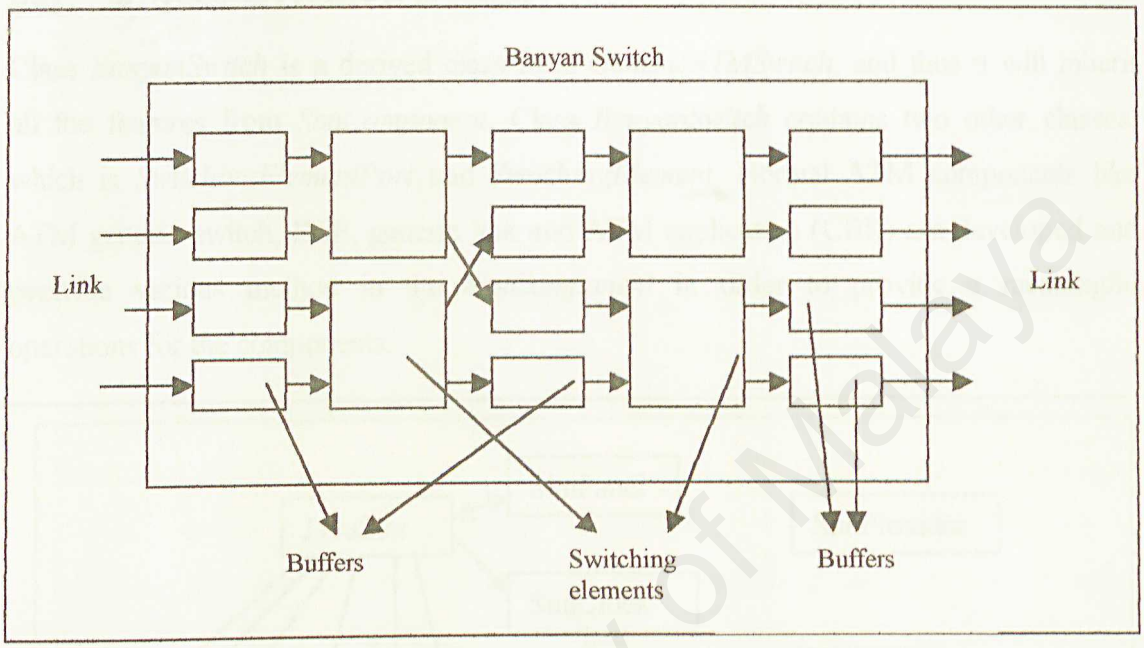

Figure 3.6: Banyan 4x4 Switch Architecture

Switching path from one inlet to one outlet for a switching element need to pay more attention. With referring to Figure 3.6, the following describes Banyan 4x4 switching path:

Stage 1 switching:

- Inlet 1 and inlet 2 at first switching element (i.e., the upper left switching element) are connected to inlet 1 and inlet 3 at stage 2 switching elements.
- Inlet 3 and inlet 4 at second switching element (i.e., the lower left switching element) are connected to inlet 2 and inlet 4 at stage 2 switching elements.

Stage 2 switching:

- Inlet 1 and inlet 2 at first switching element (i.e., the upper right switching element) are connected to outlet 1 and outlet 2.

- Inlet 3 and inlet 4 at second switching element (i.e., the lower right switching element) are connected to outlet 3 and outlet 4.

3.8 Object-Oriented Design

Class *BanyanSwitch* is a derived class from *GenericATMSwitch*, and thus it will inherit all the features from *SimComponent*. Class *BanyanSwitch* contains two other classes, which is *SwitchingElementPort* and *SwitchingElement*. Normal ATM components like ATM generic switch, BTE, generic link and ATM application (CBR) are developed and override various method in the *SimComponent* in order to provide a meaningful operations for the components.

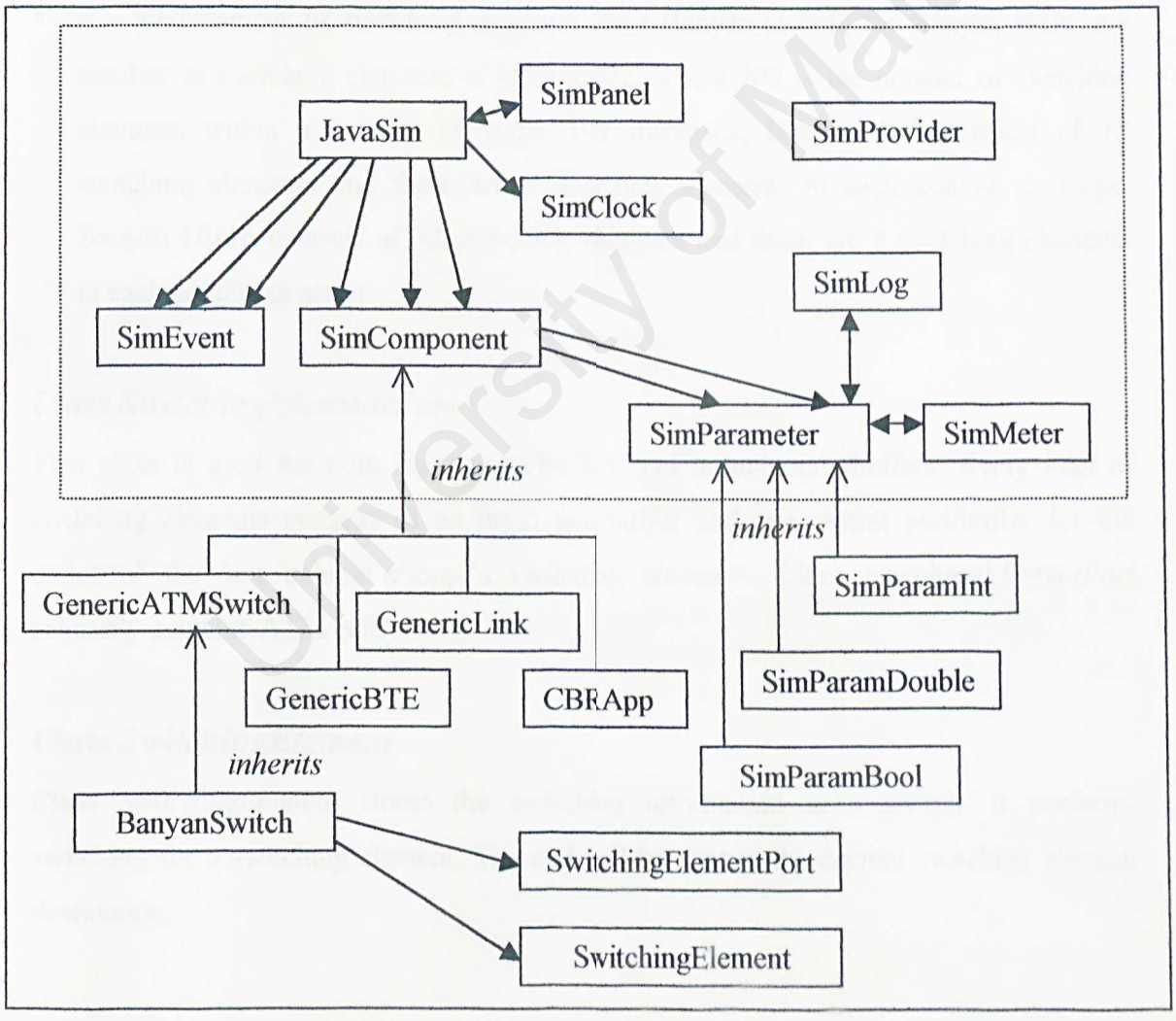

Figure 3.7: Java Switching Simulator Objects

3.9 Class Design

This section gives a description on classes design in this project. Among the classes highlighted are *BanyanSwitch*, *SwitchingElementPort* and *SwitchingElement*.

Class BanyanSwitch

Class *BanyanSwitch* is a derived class from *GenericATMSwitch*. This class is responsible for switching within an ATM switch. In here, Banyan NxN architecture is implemented. This class consists of *SwitchingElementPort* and *SwitchingElement*. Several important attributes are

- *nCol* – total columns or stages within Banyan, for Banyan NxN, the number of columns or stages is $\log_2 N$.
- *n* – total number of switching elements for a Banyan switch, for Banyan NxN, the number of switching elements is *nCol* x *N*/2, where *N*/2 is the number of switching elements within a column or stage. For instances, Banyan 8x8 consists of 12 switching elements and there are 4 switching elements in each column or stage. Banyan 16x16 consists of 32 switching elements and there are 8 switching elements in each column or stage.

Class SwitchingElementPort

This class is used for both input ports/buffers and output ports/buffers. Every inlet of switching elements consists of an input port/buffer and one output port/buffer for the outlet of the last column's/stage's switching elements. Class *SwitchingElementPort* primarily contains ATM cell.

Class SwitchingElement

Class *SwitchingElement* stores the switching information in a switch. It performs switching for a switching element. The cell will be sent to the correct switching element destination.

3.10 Summary

The ATM switch functions can be viewed from three separate ATM planes. Each of the planes has different roles. On the other hand, the ATM switch architecture consists of several parts, which includes congestion admission control, switch management, input module, cell switch fabric and the output module. ATM cells flow from the incoming link to the input module, passing through the cell switch fabric and the output module, and then finally into the outgoing link. For the proposed simulation model, the project has focused primarily on the switching within the ATM switch itself. As a result, congestion admission control and switch management are not within the scope of this project. However, ATM traffic parameters are of concern, because they play an important role in ensuring QoS. An overview of simulator model is also presented in this chapter. Lastly, the chapter concludes by presenting the JavaSim architecture, system architecture design, object-oriented design and class design to be developed.

CHAPTER 4: IMPLEMENTATION AND TESTING

Chapter 4 discusses the implementation and testing phases that need to be done for the simulator. During the implementation phase, all the classes with important attributes will be shown together with the explanation of these attributes as well as methods contained within the classes.

Simulator testing is done in two parts. Component testing will test the resource classes and system testing focuses on cell switching testing.

4.1 Implementation

This ATM simulator consists of one main package: *javasim*. This package consists of all information and classes for the execution of a simulator which includes classes like *CBRApp.java*, *Cell.java*, *GenericATMSwitch.java*, *GenericBTE.java*, *GenericLink.java*, *JavaSim.java*, *SimClock.java*, *SimComponent.java*, *SimEvent.java*, *SimLog.java*, *SimMeter.java*, *SimPanel.java*, *SimParamBool.java*, *SimParamDouble.java*, *SimParameter.java*, *SimParamInt.java*, and finally *SimProvider.java*.

Followings will describe attributes within classes of switching simulator components for this package. Among the classes are *BanyanSwitch*, *SwitchingElementPort* and *SwitchingElement*.

BanyanSwitch

```
class GenericATMSwitch extends SimComponent implements Serializable {
 class BanyanSwitch implements Serializable
 {
  int nInput,nCol; //The number of input ports and columns/stages
  SwitchingElement bs[]; //The switching element
 }
}
```

BanyanSwitch is an inheritance of *GenericATMSwitch* class. *BanyanSwitch* makes use of *nCol* to store a number of columns or stages. *n* is used to store the total number of switching elements for a Banyan switch.

When this object is instantiated, the number of input ports, *nInput* is instantiated through the *BanyanSwitch* constructor.

Two important internal methods are used in *BanyanSwitch*, i.e. *createSwitch* and *in*. These methods are declared internally because they need not accessed by other objects. *createSwitch* creates the link for the switching elements and determines the entire routing path within a Banyan NxN. *in* is responsible in creating the destination path for the ATM cell.

Banyan NxN Implementation

The inputs to the switch are the inputs to the elements in the first column/stage, and the outputs of the last column/stage are the outputs from the switch. In each switching element, one output is connected to the input of the element just horizontally on its right, and the other goes to an element whose line number, represented in binary, differs in precisely the *j*'s bit, where *j* is the column number of the element (counting from 0). This simple rule also tells how to construct a path from any input to any output: in each column/stage *j*, an appropriate switching element should be set in the *"bar"* state if the *j*'s bits of the input and the output numbers equal, and in the *"cross"* state if those bits differ. The implementation of Banyan NxN is shown below.

```
void createSwitch()
{
 int n,i,iTmp,iCol,j;

 nCol=(int)(Math.log(nInput)/Math.log(2)); //The number of columns/stages
 n=(int)(nCol*(nInput/2)); // The number of switching elements
 bs=new SwitchingElement[n];
```

```
for(i=0;i<n;i++)
  bs[i]=new SwitchingElement(""+i);

for(i=0;i<(n-(int)(nInput/2));i++)
{
 bs[i].port[0].link=i+(int)(nInput/2);

 //The creation of link
 iCol=(i/(int)Math.pow(2,(int)((i*2)/nInput)))%2;

 if(iCol==0)
   iTmp=1*(int)Math.pow(2,(i*2)/nInput);

 else
  iTmp=-1*(int)Math.pow(2,(i*2)/nInput);

  bs[i].port[1].link=iTmp+(int)(nInput/2)+i;
}

for(i=i;i<n;i++)
 for(j=0;j<2;j++)
  bs[i].port[j].link=(int)i%(nInput/2)+j;
}
```

SwitchingElementPort

```
class GenericATMSwitch extends SimComponent implements Serializable {
 class BanyanSwitch implements Serializable
 {
  class SwitchingElementPort implements Serializable
  {
   int destination[],link,nCell; //The destination and link of the ATM cell
```

```
        Cell buffer[]; //The buffer in the switching element
      }
    }
  }
```

SwitchingElementPort contains attributes for the destination and link of the ATM cell, and the buffer in the switching element.

When this object is instantiated, the cell (*nCell*), the link (*link*), the destination (*destination*) and the buffer (*buffer*) are instantiated through the *SwitchingElementPort* constructor.

Interface methods, *in* and *out* are used for both input ports/buffers and output ports/buffers. Every inlet of switching elements consists of an input port/buffer and one output port/buffer for the outlet of the last column's/stage's switching elements.

SwitchingElement

```
class GenericATMSwitch extends SimComponent implements Serializable {
 class BanyanSwitch implements Serializable
 {
  class SwitchingElement implements Serializable
  {
   SwitchingElementPort port[]; //The switching element port
   String name; //The name of switching element
  }
 }
}
```

SwitchingElement contains attributes for the switching element port and the name of the switching element.

When this object is instantiated, the name of the switching element (*name*) and the switching element port (*port*) are instantiated through the *SwitchingElement* constructor.

Interface methods, *in* and *out* are used to perform switching for a switching element.

4.2 Component Testing

Component testing is done in several classes like *BanyanSwitch*, *SwitchingElementPort* and *SwitchingElement* to ensure the interaction during the switching process will perform according to the actual theory.

BanyanSwitch

Testing on *BanyanSwitch* is easy. *BanyanSwitch* can be tested by instantiating a *BanyanSwitch* object and assigning the *MAXPORT* value. At the end, the value is printed out.

1. Instantiate object and insert *MAXPORT* value.

```
BanyanSwitch bs=new BanyanSwitch(MAXPORT);
```

2. Display the output.

```
System.out.println("Value for MAXPORT=" +MAXPORT);
```

Output:

```
Value for MAXPORT=16
```

SwitchingElementPort

SwitchingElementPort is tested by instantiating *SwitchingElementPort* object twice and the value that is printed out is exactly the same as the number of times *SwitchingElementPort* has been instantiated.

1. Instantiate object.

```
SwitchingElementPort port[];
```

```
port=new SwitchingElementPort[2];
```

2. Display the output.

```
for(i=0;i<2;i++)
  port[i]=new SwitchingElementPort();
    System.out.println(i);
```

Output:

```
2
```

SwitchingElement

SwitchingElement is tested by instantiating *SwitchingElement* object 32 times and the value that is printed out is exactly the same as the number of times *SwitchingElement* has been instantiated.

1. Instantiate object.

```
SwitchingElement bs[];
bs=new SwitchingElement[32];
```

2. Display the output.

```
for(i=0;i<32;i++)
  bs[i]=new SwitchingElement(""+i);
    System.out.println(i);
```

Output:

```
32
```

4.3 System Testing

The major purpose of the simulator system testing is switching testing. Switching testing is carried out with the purpose to make sure that internal switching for ATM cells is correctly and successfully reaching desired output port.

Figure 4.1 and Figure 4.2 show the topologies that have been used for this testing session.

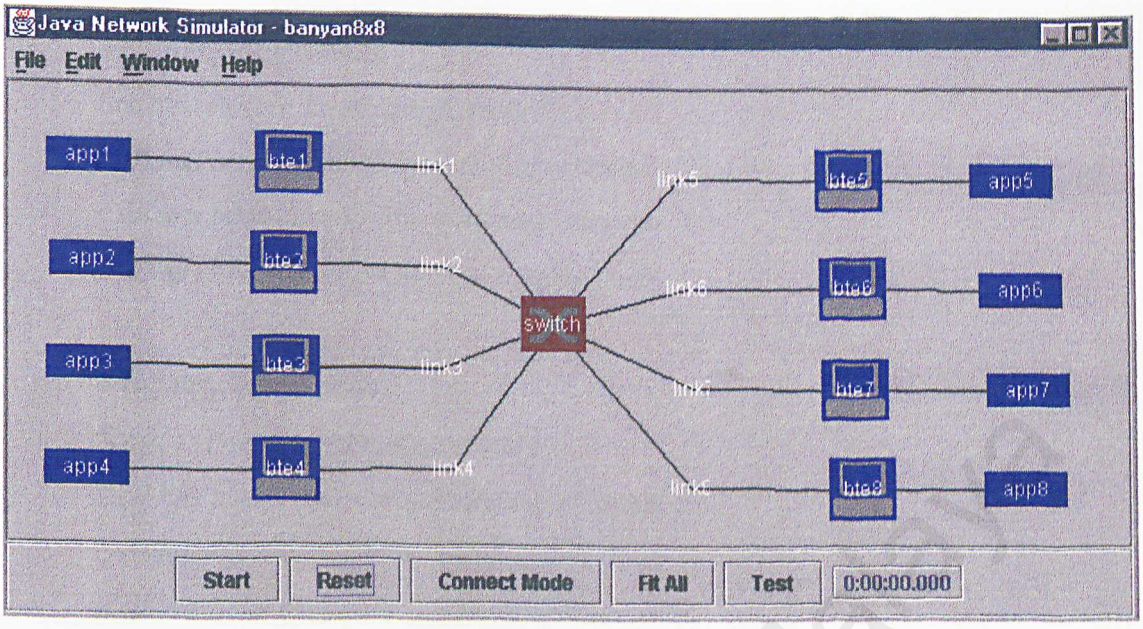

Figure 4.1: Testing Topology for Banyan 8x8

For Banyan 8x8, the followings static route has been made:

- app1 to app8 (Switching will be performed from input port 0 to output port 7 and input port 7 to output port 0)
- app2 to app7 (Switching will be performed from input port 1 to output port 6 and input port 6 to output port 1)
- app3 to app6 (Switching will be performed from input port 2 to output port 5 and input port 5 to output port 2)
- app4 to app5 (Switching will be performed from input port 3 to output port 4 and input port 4 to output port 3)

Output for the Banyan 8x8 topology:

STARTED

This is a Banyan 8x8 Switch Architecture

Number of Columns/Stages: 3

Number of Input and Output Ports: 8

Number of Switching Elements within a Column/Stage: 4

Total Number of Switching Elements for this Banyan Switch is: 12

Switching Information:
Source: Port 0; Destination: Port 7
Cell has been received by switching element: 0;
Cell has been received by switching element: 5;
Cell has been received by switching element: 11;

Switching Information:
Source: Port 7; Destination: Port 0
Cell has been received by switching element: 3;
Cell has been received by switching element: 6;
Cell has been received by switching element: 8;

Switching Information:
Source: Port 1; Destination: Port 6
Cell has been received by switching element: 0;
Cell has been received by switching element: 5;
Cell has been received by switching element: 11;

Switching Information:
Source: Port 6; Destination: Port 1
Cell has been received by switching element: 3;
Cell has been received by switching element: 6;
Cell has been received by switching element: 8;

Switching Information:
Source: Port 2; Destination: Port 5
Cell has been received by switching element: 1;
Cell has been received by switching element: 4;
Cell has been received by switching element: 10;

Switching Information:
Source: Port 5; Destination: Port 2
Cell has been received by switching element: 2;
Cell has been received by switching element: 7;
Cell has been received by switching element: 9;

Switching Information:
Source: Port 3; Destination: Port 4
Cell has been received by switching element: 1;
Cell has been received by switching element: 4;
Cell has been received by switching element: 10;

Switching Information:
Source: Port 4; Destination: Port 3
Cell has been received by switching element: 2;
Cell has been received by switching element: 7;
Cell has been received by switching element: 9;

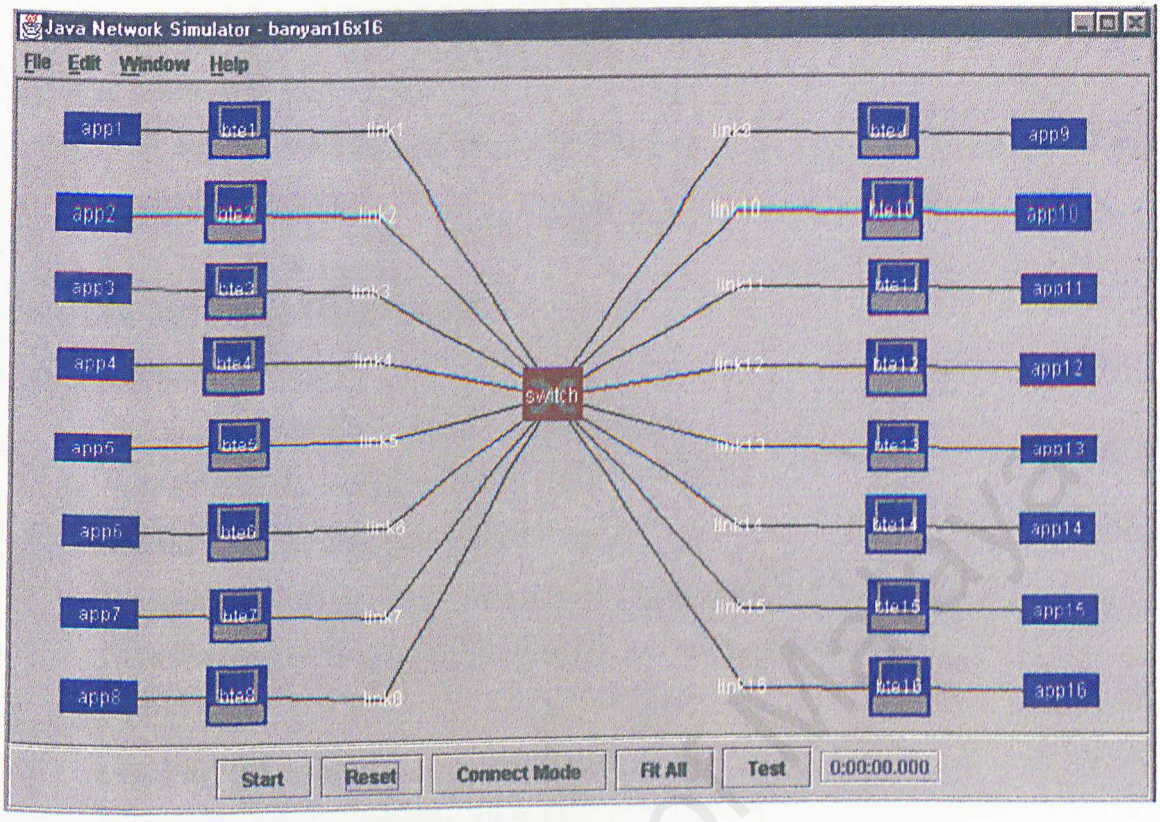

Figure 4.2: Testing Topology for Banyan 16x16

For Banyan 16x16, the followings static route has been made:

- app1 to app16 (Switching will be performed from input port 0 to output port 15 and input port 15 to output port 0)
- app2 to app15 (Switching will be performed from input port 1 to output port 14 and input port 14 to output port 1)
- app3 to app14 (Switching will be performed from input port 2 to output port 13 and input port 13 to output port 2)
- app4 to app13 (Switching will be performed from input port 3 to output port 12 and input port 12 to output port 3)
- app5 to app12 (Switching will be performed from input port 4 to output port 11 and input port 11 to output port 4)
- app6 to app11 (Switching will be performed from input port 5 to output port 10 and input port 10 to output port 5)

- app7 to app10 (Switching will be performed from input port 6 to output port 9 and input port 9 to output port 6)
- app8 to app9 (Switching will be performed from input port 7 to output port 8 and input port 8 to output port 7)

Output for the Banyan 16x16 topology:

STARTED

This is a Banyan 16x16 Switch Architecture

Number of Columns/Stages: 4

Number of Input and Output Ports: 16

Number of Switching Elements within a Column/Stage: 8

Total Number of Switching Elements for this Banyan Switch is: 32

Switching Information:

Source: Port 0; Destination: Port 15

Cell has been received by switching element: 0;

Cell has been received by switching element: 9;

Cell has been received by switching element: 19;

Cell has been received by switching element: 31;

Switching Information:

Source: Port 15; Destination: Port 0

Cell has been received by switching element: 7;

Cell has been received by switching element: 14;

Cell has been received by switching element: 20;

Cell has been received by switching element: 24;

Switching Information:

Source: Port 1; Destination: Port 14

Cell has been received by switching element: 0;

Cell has been received by switching element: 9;

Cell has been received by switching element: 19;
Cell has been received by switching element: 31;

Switching Information:
Source: Port 14; Destination: Port 1
Cell has been received by switching element: 7;
Cell has been received by switching element: 14;
Cell has been received by switching element: 20;
Cell has been received by switching element: 24;

Switching Information:
Source: Port 2; Destination: Port 13
Cell has been received by switching element: 1;
Cell has been received by switching element: 8;
Cell has been received by switching element: 18;
Cell has been received by switching element: 30;

Switching Information:
Source: Port 13; Destination: Port 2
Cell has been received by switching element: 6;
Cell has been received by switching element: 15;
Cell has been received by switching element: 21;
Cell has been received by switching element: 25;

Switching Information:
Source: Port 3; Destination: Port 12
Cell has been received by switching element: 1;
Cell has been received by switching element: 8;
Cell has been received by switching element: 18;
Cell has been received by switching element: 30;

Switching Information:
Source: Port 12; Destination: Port 3
Cell has been received by switching element: 6;
Cell has been received by switching element: 15;
Cell has been received by switching element: 21;
Cell has been received by switching element: 25;

Switching Information:
Source: Port 4; Destination: Port 11
Cell has been received by switching element: 2;
Cell has been received by switching element: 11;
Cell has been received by switching element: 17;
Cell has been received by switching element: 29;

Switching Information:
Source: Port 11; Destination: Port 4
Cell has been received by switching element: 5;
Cell has been received by switching element: 12;
Cell has been received by switching element: 22;
Cell has been received by switching element: 26;

Switching Information:
Source: Port 5; Destination: Port 10
Cell has been received by switching element: 2;
Cell has been received by switching element: 11;
Cell has been received by switching element: 17;
Cell has been received by switching element: 29;

Switching Information:
Source: Port 10; Destination: Port 5
Cell has been received by switching element: 5;

Cell has been received by switching element: 12;
Cell has been received by switching element: 22;
Cell has been received by switching element: 26;

Switching Information:
Source: Port 6; Destination: Port 9
Cell has been received by switching element: 3;
Cell has been received by switching element: 10;
Cell has been received by switching element: 16;
Cell has been received by switching element: 28;

Switching Information:
Source: Port 9; Destination: Port 6
Cell has been received by switching element: 4;
Cell has been received by switching element: 13;
Cell has been received by switching element: 23;
Cell has been received by switching element: 27;

Switching Information:
Source: Port 7; Destination: Port 8
Cell has been received by switching element: 3;
Cell has been received by switching element: 10;
Cell has been received by switching element: 16;
Cell has been received by switching element: 28;

Switching Information:
Source: Port 8; Destination: Port 7
Cell has been received by switching element: 4;
Cell has been received by switching element: 13;
Cell has been received by switching element: 23;
Cell has been received by switching element: 27;

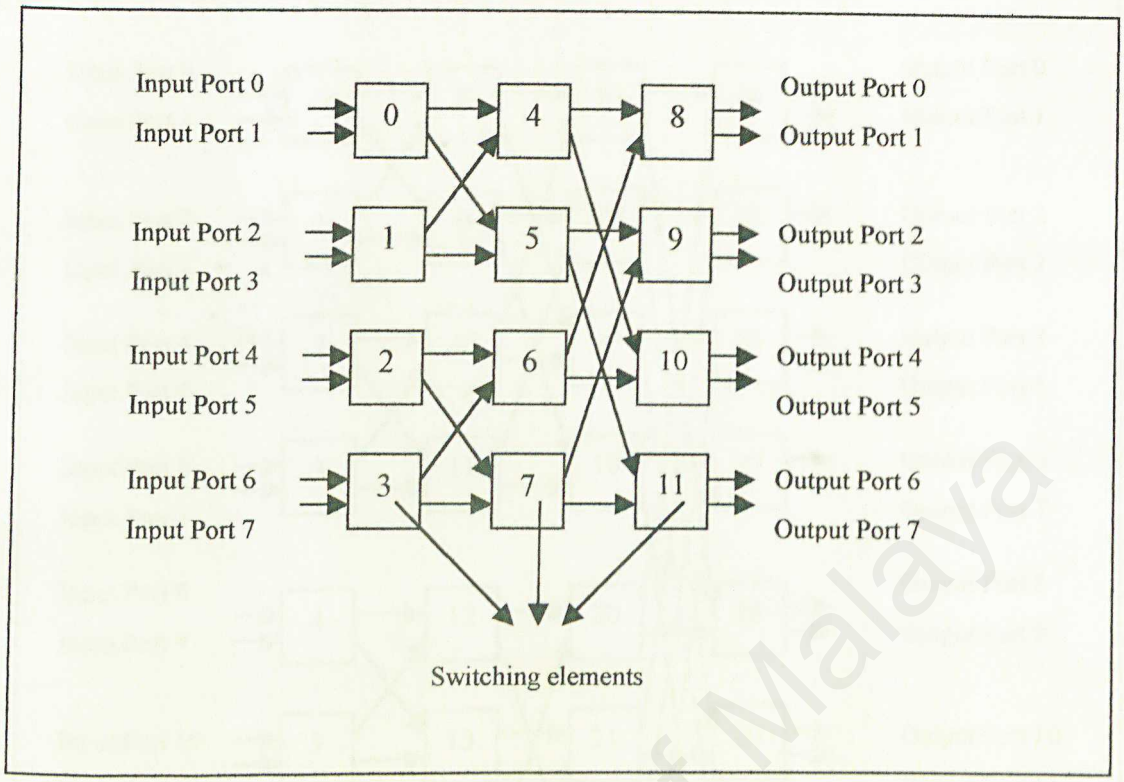

Figure 4.3: The Actual Cell Switching for Banyan 8x8

Based on Figure 4.3 and Figure 4.4, the cells are correctly switched from the source to their destination. This shows that all components are working properly and successful.

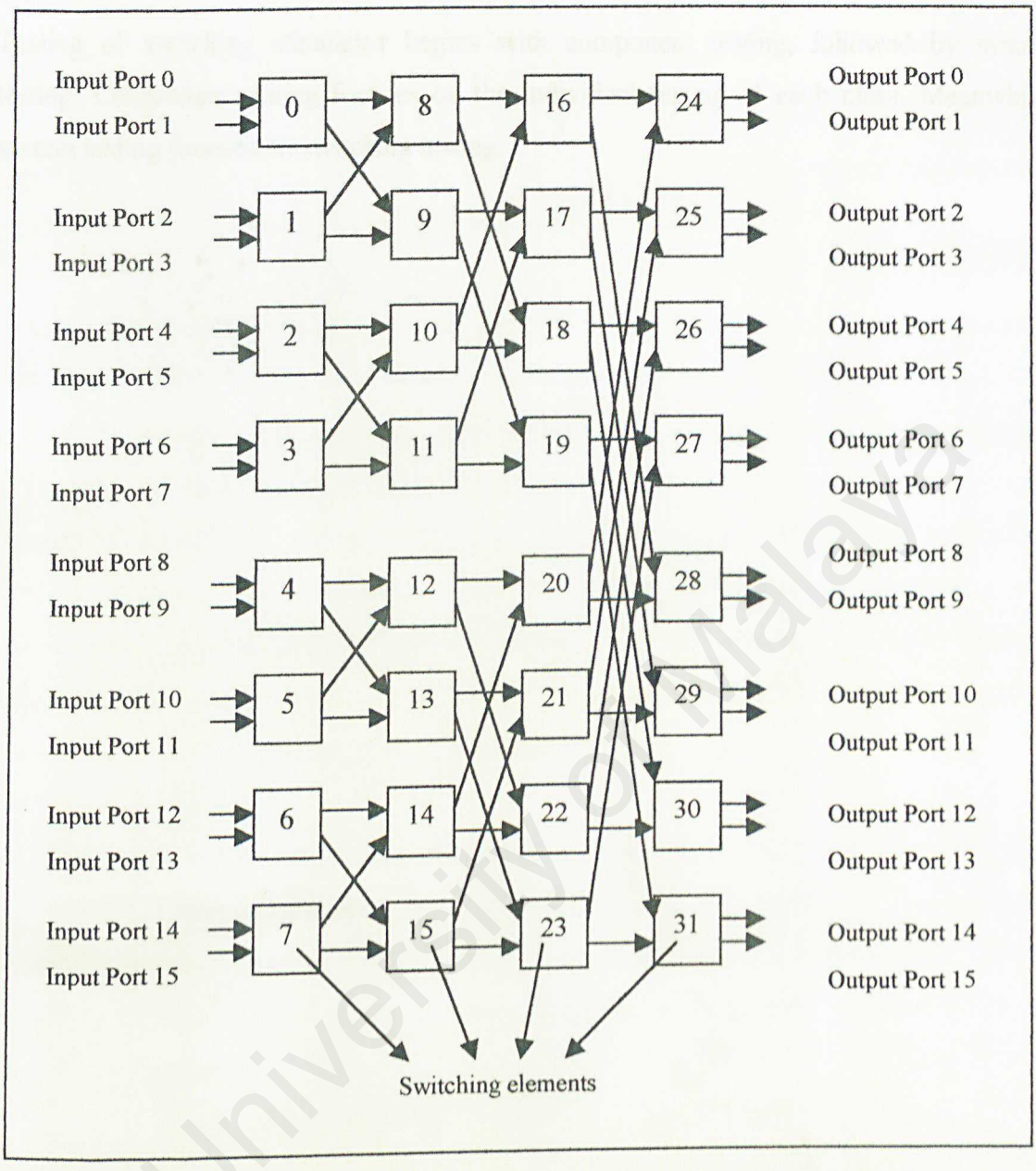

Figure 4.4: The Actual Cell Switching for Banyan 16x16

4.4 Summary

This chapter gives an idea on how the implementation and testing processes on the switching simulator were carried out. Class implementation explains the attributes of each class, which are declared together with their data types. The class implementation also explains the methods in each class. This section is followed by the implementation of Banyan NxN switch.

Testing of switching stimulator begins with component testing, followed by system testing. Component testing focuses on the individual testing of each class. Meanwhile, system testing focuses on switching testing.

CHAPTER 5: CONCLUSION

There are numerous architectures for ATM switches and each has its own merits and drawbacks. This report presents a general Banyan switch for simulating switching architectures in ATM networks using object-oriented programming.

Object-oriented programming is a type of programming in which programmer defines not only the attributes, but also the types of operations that can be applied to the attributes. Both attributes and operations are encapsulated into an object. One of the principal advantages of object-oriented programming techniques over procedural programming techniques is that they enable programmers to create modules that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits features from existing objects. Moreover, object-oriented programming is simple because it models the real world objects. It has attributes such as modifiability, extensibility, maintainability, and reusability.

Java is an object-oriented language, which is similar to C++ but it is more robust than C++ with no pointer references to external data. The other great features of Java are built-in support for multithreading and the ability of the threads to run simultaneously. Finally, the ability of Java to work under different platforms and different browsers fulfill the objectives of this project.

This project is a simulation of Banyan NxN switching architecture. Designing of this simulator is based on the object-oriented approach where the classes with corresponding attributes and functions are defined first.

The implementation phase covers the part of coding for this simulator. Every class and algorithm, which has been designed, should be included in Java code. Testing is done on all of the Java classes. This testing process is also concerned with finding errors, which result from unanticipated interaction.

This project managed to achieve the overall project objectives and goals, i.e. development of a portable, cross-platform and a user-friendly graphical user interface (GUI) simulator. Lastly, the following highlights strengths, limitations as well as the proposed future enhancements.

System Strengths

- The simulator is fully object-oriented whereby all the functions and modules are built in class. In addition, problem of coupling is highly reduced.
- The simulator is built using Java *Thread* technology. When more than one switch is created, switching environment becomes more realistic where all switches run concurrently instead of sequentially.
- The separation of super class and subclass makes the switch design extensible.

System Limitation

- The simulator works based on Java application, therefore it is not web-enabled.
- The simulator works based on Banyan NxN architecture. Other switching models such as Knockout switch are not presented.

Future Enhancements

- It should include other switching models, which allows a user to select the architecture.
- It should also allow the execution of performance comparison for different types of switching architecture.

REFERENCES

[1] Asynchronous Transfer Mode (ATM) Fundamentals Tutorial. *http://www.webproforum.com/atm_fund/topic01.html*. Last updated: October 6, 1999. Web ProForums.

[2] Stallings, William. 1998. *High Speed Networks: TCP/IP and ATM design Principles*. New Jersey, Prentice-Hall, Inc. pp. 85-88.

[3] J. Kenny. 1999. The ATM Forum Traffic Management Specification Version 4.1, *AF-TM0121.000*.

[4] ATM Switching Structures – A Performance Comparison. *http://www.tts.lth.se/Personal/daniels/papers/nts12.abstract.html*. Daniel Sobirk, Johan M Karlsson.

[5] Appendix A. Computer Network Simulation. *http://oak.ece.ul.ie/~dalyf/thesis/aa.htm*. Last updated: October 28, 1997. Fergal.

[6] Application and Protocol Testing through Network Emulation. *http://is2.antd.nist.gov/itg/nistnet/slides/index.htm*. Last updated: September 1997. Internetworking Technologies Group NIST.

[7] Discrete-Event Simulation. *http://may.cs.ucla.edu/slides/meyer-pw98/tsld002.htm*. Last updated: November 16, 1999. UCLA Parallel Computing Laboratory.

[8] Nada G., el. 1998. *The NIST ATM/HFC Network Simulator Operation and Programming Guide Version 4.0*. National Institute of Standards and Technology.

[9] INSANE An Internet Simulated ATM Networking Environment. *http://www.ca.sandia.gov/~bmah/Software/Insane/*. Last modified: November 25, 1998. Bruce A. Mah.

[10] REAL 5.0 Overview. *http://www.cs.cornell.edu/skeshav/real/overview.html*. Last updated: August 13, 1997. S. Keshav, Cornell University.

[11] The REAL Network Simulator. *http://minnie.cs.adfa.edu.au/REAL/index.html*. Last updated: June, 1995. Warren Toomey.

[12] Guo, M., Hoang, D.B. 1998 An Object-based Network Simulator. *Global Telecommunication Conference 1998, GLOBECOM 1998.* Vol.3, pp. 1562-1567. November, 1998.

[13] Deitel & Deitel. 1999. *JAVA How to Program, Third edition.* Prentice Hall Inc. New Jersey.

[14] The Source For Java™ Technology. *http://www.java.sun.com.* Last updated: May 12, 2000. Sun Microsystems, Inc.

[15] Borland JBuilder. *http://www.borland.com/techpubs/jbuilder/jbuilder3-5/qs/intro.html.* Last updated: April 7, 2000. Borland Inprise.

[16] Introduction to ATM switching. *http://www.rad.com/networks/1994/gbiran/atm_swi.htm.* Last Updated: 1994. Giora Biran.

[17] B. Zhou and M. Atiquzzaman. 1995. A Performance Comparison of Buffering Schemes for Multistage Switches. *ICASPP'95, First International Conference on Algorithms And Architectures for Parallel Processing. Brisbane, Australia, April 19-21, 1995*

[18] Hakyong Kim, Ahmad, A., Changhwan Oh, Kiseon Eim. 1997. Performance Comparison of High-Speed Input-buffered ATM Switches. *Proceedings of the IEEE ATM Workshop 1997.* pp. 505 –513.

[19] Kwon. B., Kim. B., Park. J., Yoon. H., Cho. J. 1995. Performance Analysis of Output Buffers in Multistage Interconnection Networks with Multiple Paths. Parallel and Distributed, in, *Proceeding of the Seventh IEEE Symposium.* pp. 260 –265.

[20] Peifang Zhou, Yang, O.W.W. 1997. A New Design of Central Queuing ATM Switches. *Global Telecommunications Conference, 1997, IEEE GLOBECOM '97.* Vol 1. pp. 541 –545.

[21] Alimuddin, M., Alnuweiri, H.M., Donaldson, R.W. 1995. The Fat Banyan ATM Switch. 'Bringing Information to People', in, *Proceedings of the Fourteenth Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE INFOCOM '95.* Vol.2. pp. 659-666.

[22] Aude, J.S., Young, M.T., Bronstein, G. 1998. 1998. A High-performance

Switching Element for A Multistage Interconnection Network. *Integrated Circuit Design, 1998. Proceedings. XI Brazilian Symposium.* pp. 154 –157.

[23] A Survey of ATM Switching Techniques. *ftp://ftp.netlab.ohio-state.edu/pub/jain/courses/cis788-95/atm_switching/index.html.* Last Updated: August 21, 1995. Sonia Fahmy.

[24] Aude, J.S., Young, M.T., Bronstein, G. 1998. 1998. A High-performance Switching Element for A Multistage Interconnection Network. *Integrated Circuit Design, 1998. Proceedings. XI Brazilian Symposium.* pp. 154 –157.

[25] Y. Yeh, M. G. Hluchyj, A. S. Acampora, 1987. The Knockout Switch: A Simple, Modular Architecture for High-Performance Packet Switching. *IEEE Journal on Selected Areas in Communications, VSAC-5.* No 8. pp. 1274-1283.

[26] David M. Drury. 1996. ATM Traffic Management and the Impact of ATM Switch Design, *Computer Network and ISDN Systems.* Vol 28, pp. 471-479. 1996.

[MONET] Procedural Programming Language. *http://monet.uwaterloo.ca/janeli/pp.htm.* Jane Li.