CHAPTER 2: LITERACTURE SURVEY

This chapter begins with an introduction to various simulators. A detailed description is provided for eight simulators that have been studied, including the advantages and disadvantages for each simulator. A comparison table between these simulators is summarised too. The 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 why Java programming language was selected and finally provides a survey of the available Java development tools.

A review is conducted on the ATM switching approach, buffering methods, queuing models and switching models. Once the groundwork has been laid, a discussion on various performance issues will be extended.

2.1 Introduction to Various Simulator

There are currently a few simulators on the market and some of them are the postgraduate thesis. The following sub-sections describe various simulators that have been implemented, including VISTA, NIST ATM/HFC, PARSEC, OPNET, OMNeT++, NetSim, Delsi, and Real Network Simulator. At the end of this section, a comparison among these simulators is presented.

2.1.1 VISTA

VISTA (configurable VISualization and Simulation Tool for ATM switches) is a general-purpose tool for simulating switching architectures and visualising their performance graphically under various traffic conditions and buffering schemes. It is a master thesis done by Narayana R. Tummala, University of Texas at San Antonio in 1996, under the supervision of Kay A. Robbins. VISTA is a discrete-event simulation developed using C. The visualization mainly concentrates on the dynamic distribution of lost cells across output ports as well as on the statistics collection such as average

delay, average throughput, and cell loss ratio. VISTA simulates the parametric uniform, geometrically traffic mode with three different buffering schemes, which are pushout, buffer sharing, and selective discard.

Advantages

- Built-in support for both Tandem Banyan and Knockout switch.
- A MPEG movie making capability.
- Source code provided.

Disadvantages

- VISTA is not object-oriented which make its source code highly couple and difficult to understand.
- Poor graphical user interface.
- Only run in Unix platform.
- User must have a strong background in C programming [11].

2.1.2 NIST ATM/HFC Network Simulator

The ATM/HFC network simulator is a tool to analyse the behaviour 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 [12].

Advantages

- Allows user to create different network topologies.

- Allows user to adjust the parameters of each component's operation, measure network activity, save/load different simulation configuration and log data during simulation execution.
- Provides graphical user interface.

Disadvantages

- Not object-oriented
- 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.

2.1.3 PARSEC

PARSEC (for PARallel Simulation Environment for Complex systems) is a C-based discrete-event simulation language. It adopts the process interaction approach to discrete-event simulation. An object (also referred to as a physical process) or set of objects in the physical system is represented by a logical process. Interactions among physical processes (events) are modelled by time stamped message exchanges among the corresponding logical processes.

Advantages

- Able to execute a discrete-event simulation model using several different asynchronous parallel simulations protocols for a variety of parallel architectures.
- PARSEC is designed to cleanly separate the description of a simulation model from the underlying simulation protocol, sequential or parallel, used to execute it. Thus, with few modifications, a PARSEC program may be executed using the traditional sequential (Global Event List) simulation protocol or one of many parallel optimistic or conservative protocols.
- Provides powerful message receiving constructs that result in shorter and more natural simulation programs.
- Useful debugging facilities are available.
- A front-end for visual specification of simulation models, and a runtime output display and visualisation environment are currently being designed.
- Provides lightweight thread.

Disadvantages

- Shared/Global Variables: A parallel PARSEC simulation should not use `global' variables. This includes function and file scope static variables. Read-only variables initialised at the beginning of the simulation can be global, but must be initialised separately on each processor (on a distributed memory architecture).
- Pointers cannot be passed in messages or entity initialisation parameters. Note that this implies dynamic linked-list data-structures cannot be passed. Arrays must be used instead [13].

2.1.4 OPNET

OPNET (Optimised Network Engineering Tools) is the commercial product of the MIL3 Company of Arlington, VA. It employs a discrete event simulation approach that

allows large numbers of closely spaced events in a sizeable network to be represented accurately and efficiently. OPNET uses a modelling approach where networks are built of components interconnected by perfect links that can be degraded at will. Each component's behaviour is modelled as a state-transition diagram. The process that takes place in each state is described by a program in the C language. It will make the OPNET-based models relatively easy to port to other modelling environments.

Advantages

- Object Orientation: OPNET is object oriented, where each object has a defined set of attributes. These configurable attributes result in a highly flexible development environment.
- Hierarchical Modelling: OPNET employs a hierarchical approach to modelling, having three separate domains to describe any communication network. Each level of the hierarchy describes different aspects of the complete model being simulated. Models developed at one level of the hierarchy are used (or inherited) by models at the next higher level. This leads to a highly flexible simulation environment where generic models can be developed and used in many different scenarios.
- Specialised in communication networks: Detailed library models provide support for existing protocols, allow researchers and developers to either modify these existing models or develop new models of their own. The set of standard models provides an insight into the workings of existing communication processes, protocols, and networks.
- Automatic simulation generation: OPNET models can be compiled into executable code. An executable discrete-event simulation can be debugged or simply executed, resulting in output data.
- Debugging tools: The interactive debugging tool allows model developers to trace a wide range of model characteristics. For example, the contents of a packet or the creation time of a process can be easily examined [7].

Disadvantages

- OPNET currently only runs within the Solaris, Windows NT and HP-UX platforms.
- As it is a commercial software, its is very costly to purchase.

2.1.5 OMNeT++

OMNeT++ (Objective Modular Network Testbed in C++) is an object-oriented modular discrete event simulation tool. It is primarily designed for modelling communication protocols, computer networks, multi-processors and distributed systems, and any other system where the discrete event approach is suitable.

OMNeT++ provides much more than just the simulation library with statistical classes. It has execution environments that support interactive simulation including visualization of collected data. There is a gnuplot-based GUI tool for analysing and plotting simulation results. OMNeT++ also helps in specifying parameter values, managing multiple runs, selecting seed values etc, and supports parallel execution (PVM3-based).

Advantages

- Solid and flexible simulation kernel.
- Powerful GUI environment.
- Easy to build hierarchical and reusable models.
- Source code provided.
- Open interfaces (like human-readable text files).
- Strong user community.

Disadvantages

- User must work on the command line.
- User must know either C or C++ programming language [14][15].

2.1.6 NetSim

NetSim (Network Simulator) is a computer program that simulates theoretical detection and location capability of seismic networks. These simulations are used to assess seismic network capabilities to detect and locate explosions and earthquakes for monitoring of test ban limitation treaties. NetSim is derived from earlier program such as NETWORTH and SNAP/D. The current version of NetSim capable of simulating the detection of regional Pn, Pg, Sn, and Lg as well as teleseismic P, PKP, and S. Location assessment may be performed using all these phases. Furthermore, NetSim attempts to identify the frequency at which the detection will have the largest signal to noise ratio SNR).

Advantages

- Suitable in simulating large networks.
- Can be used as a component of a simulation testbed capable of simulating a full parallel system

Disadvantages

- This simulator that has not been thoroughly tested and may contain bugs
- Does not give user an interactive graphical user interface (GUI) environment.
- User must have a strong knowledge in C programming language.
- No other traffic management and QoS is considered to be present [16].

2.1.7 Delsi

Delsi (Delphi Simulation) is a discrete-event simulation tool intended for simulation of queuing systems with complicated logic. The tool is based on Piecewise Linear Aggregate Simulation Architecture (PLASA) and implemented as a set of components for the popular RAD environment Borland® Delphi™ 3.0 & 4.0. The internal

simulation logic is optimised for high loading, i.e. for a great number of transactions in the model. The implementation of model's logic is based on easy-to-use Delphi event handling. Delphi RAD environment allows users to combine Delsi with other components intended for GUI design, building reports, working with databases etc. Such combination creates a very flexible and productive framework for developing powerful simulation applications as retail products for Windows NT/95/98

Advantages

- Object-orientated, all the components in Delsi are built in objects.
- Ease to use, not much data need to be enter for simulation.
- Provides graphical user interface.

Disadvantages

- User or programmer needs to understand the Delphi object-oriented programming which is Pascal syntax-based [17].

2.1.8 REAL Network Simulator

The REAL network simulator is a network simulator designed for testing congestion and flow control mechanisms. It provides users with a way of specifying such networks and to observe their behaviour. It provides around 30 modules that exactly emulate the actions o several well-known flow control protocols and 5 research scheduling disciplines Source code is provided so that interested users can modify the simulator to their own purposes. The simulator takes as input a scenario, which is a description of network topology, protocols, workload and control parameters. It produces as output statistics such as the number of packets sent by each source of data, the queuing delay at each queuing point, the number of dropped and retransmitted packets and other similar information.

Advantages

- Provides a flexible testbed for studying the dynamic behaviour of flow and congestion control schemes in packet switch data networks
- User can modify the simulator software to accommodate network components

Disadvantages

- Does not give user an interactive modelling environment with a graphical user interface (GUI) representation capabilities
- User must understand the C programming language [18].

2.1.9 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-enabled.

Table 2:1: Comparison among Various Simulators

Simulator	Discrete event simulation	Object-oriented	GUI	multithread	Web-enabled	Platform independent
VISTA	√	X	Poor	X	X	X
NIST ATM/HFC	√	X	√	X	X	X
PARSEC	√	√	Poor	√	X	X
OPNET	√	√	√	X	X	X
OMNet++	√	√	√	X	X	X
NetSim	√	X	Poor	X	X	X
DELSI	√	√	√	X	X	X
REAL	√	X	Poor	X	X	X

All the simulators are discrete event simulators. However, PARSEC, OPNET, OMNet++, and Delsi were developed using object-oriented approach. On the other hand, NIST ATM/HFC, OPNET, OMNet++ provide a powerful graphical user interface while others are poor in user interface design. Unfortunately, only PARSEC

supports multithreading and none of the simulators are web-enabled and platform independent.

2.2 Programming Approaches

Programming language was created to solve some type of particular problem. They are hundreds of programming languages available. Basically, these programming languages can be categorised into two approaches: procedural programming approach and object-oriented programming (OOP) approach.

2.2.1 Procedural Programming

Procedure programming specify an exact sequence or order of operation. Program logic in procedural programming determines the next step to execute and this logic determination is made in response to conditions and user action. A function or procedure is a relatively simple program that is called by other programs and returns a value to the program that called it. The code can be put into blocks called procedures or functions. The goal of each of these blocks was to act like a black box which completed one task or another. Most traditional computer languages like C, Cobol and FORTRAN are procedural.

In a procedural programming language, a programmer writes out instruction that are followed by a computer from start to finish. For example, a procedural program might work like figure 2.1.

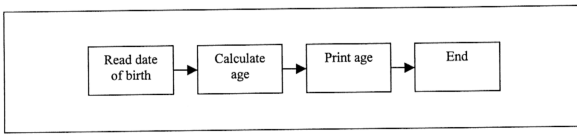

Figure 2.1: A Simple Procedural Programming

A more complex procedural-based program might introduce logical branches such as illustrated in figure 2.2.

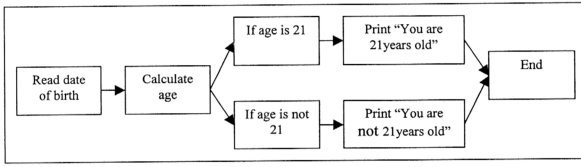

Figure 2.2: A More Complex Procedural Programming

But even in the case of a more complex logical flow, the main idea remains the same: a certain set of instructions is followed through from beginning to end, each step building upon and tied to every previous step. Procedural programming language is a powerful for solving particular problem. However, it is suitable for small and simple program only. When program become more complex, it will make the program source code harder to read and understand.

2.2.2 Object-Oriented Programming

OOP is different from procedural programming language in several ways. Everything in OOP is grouped as *object*. Every object communicates with each other by sending message.

Encapsulation concept is one of the features of OOP, it is the inclusion within a object of all the resources need for the object to function - basically, the data items and the methods. Methods are responsible for manipulating the data in order to reflect its behaviours. The public interface defines how to use this object. Other objects adhere to

these interfaces to use the object without having to be concerned with how the object accomplishes it. The idea is "don't tell me how you do it; just do it".

The second feature of OOP is the concept of inheritance, a class can derive similar or related object from a more general object. This means for the programmer that a derived class need not carry its own definition of data and methods that are generic to the class (or classes) of which it is a part. This not only speeds up program development; it also ensures an inherent validity to the defined subclass object (what works and is consistent about the class will also work for the subclass)

Another powerful feature of OOP is polymorphism, it is the characteristic of being able to assign a different meaning to a particular symbol or in different contexts. The following describe some advantages of OOP

- Simplicity: software objects model real world objects, so the complexity is reduced and the program structure is very clear;.
- Modularity: each object forms a separate entity whose internal workings are decoupled from other parts of the system.
- Modifiability: it is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods.
- Extensibility: adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones.
- Maintainability: objects can be maintained separately, making locating and fixing problems easier.
- Reusability: objects can be reused in different programs [19].

OOP is the technique chosen for project development because it provides a good practice in programming with a lot of benefits compare to procedural programming

techniques. The following section will further describe object-oriented programming language used.

2.3 Programming Language

Java is an object-oriented programming language especially designed for use in internet environment. It was designed to have the "look and feel" of the C++ language, but it is easier to use compare to C++. Java can be used to create complete applications that may run on a single computer or be distributed among servers and clients in a network. It can also be used to build applets for use as part of a Web page.

The major advantages of Java are:

- Simple – Java is simple for building a system that could be programmed easily without a lot of esoteric training and which leveraged today's standard practice.
- Object-oriented – Object-oriented facilities of Java are essentially those of C++, with extensions from Objective C for more dynamic method resolution.
- Network-Savvy – Java has an extensive library of routines for coping easily with TCP/IP protocols like HTTP and FTP. This makes creating network connections much easier than in C or C++.
- Robust – Java objects can contain no references to data external or other known objects. This ensures that an instruction cannot contain the address of data storage in another application or in the operating system itself, either of which would cause the program and the operating system itself to terminate or "crash". Garbage collection is another powerful feature that makes no chance for a Java program to corrupt the memory via a dangling pointer. The Java virtual machine makes a number of checks on each object to ensure integrity.
- Secure – Java's run-time system performs checking to ensure that programs transmitted over a network have not been tampered with. The code produced by the Java compiler is checked for validity, and the program is prevented from performing unauthorised actions.

- Architecture neutral – Networks are composed of a variety of systems with a variety of CPU and operating system architectures. To enable a Java application to execute anywhere on the network, the compiler generates an architecture-neutral object file format – the compiled code is executable on many processors, given the presence of the Java runtime system.
- Portable – Java program is compiled into bytecode instead of bitcode. This feature make Java can be run anywhere in a network on a server or client.
- Interpreted – Java bytecodes are translated on the fly to native machine instructions (interpreted) and not stored anywhere. Since linking is a more incremental and lightweight process, the development process can be much more rapid and exploratory.
- High performance – In addition to being executed at the client rather than the server, a Java applet has other characteristics designed to speed up the execution.
- Multithread – Java has built-in support for multitasking. A Java program may create any number of threads, which appear to execute in parallel.
- Dynamic – Java is a more dynamic language than C or C++. It was designed to adapt to an evolving environment [20][21].

There are numerous object-oriented languages in the market currently. Normally, Java is used as it is simpler and robust when comparing to C++. Moreover, it does not have the headache pointer-referencing problem compare to C++ programmer. Visual Basic and Borland Delphi can provide a powerful graphical user interface and powerful debugging tool but these languages are not platform independent. Moreover, Borland Delphi is only a visual programming language that cannot create web application.

The languages concerned above (except Java) should not be considered. This is not only due to the problems stated above but also none of the programming languages has the built-in support for multithreading. Besides, platform and browser dependent also do not meet the project objective. That is, only Java supports for multithreading,

platform and browser independent, and is object-oriented. The next section further elaborates Java programming tool which is used for the development of switching simulator.

2.4 Programming Tool

There are many types of Java programming tools available in market. One can develop Java programming using pure Java SDK [10] without the supporting of integrated development environment (IDE), or just selects a tools 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 [22].

2.5 Switching Approaches

ATM switching approaches can be categorised into three categories: shared memory approach, shared medium approach, and fully interconnected approach [5]

2.5.1 Shared Memory Approach

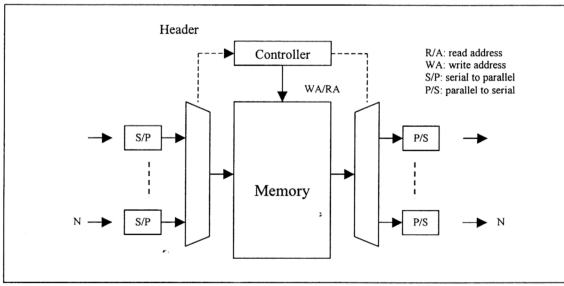

Figure 2.3: Shared Memory Approach

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 minimised 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 [5].

2.5.2 Shared Medium Approach

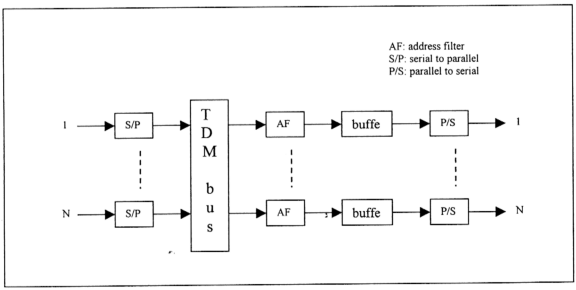

Figure 2.4: Shared Medium Approach

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 [5].

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 [5].

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 [24]. Figure 2.5 illustrates the four buffering

methods. Each of the methods has its advantages and disadvantages and neither one can be said to be the best.

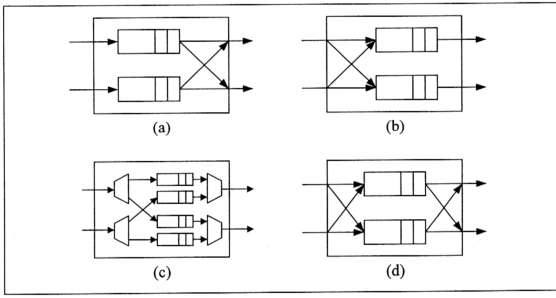

Figure 2.5: Buffering Methods: (a) Input (b) Output (c) Crossbar (d) Central [24]

2.6.1 Input Buffering

Input buffering [25] 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 [26] 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 minimise 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 [5][27].

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 – mean number of arrivals per second.
- w – mean number of item waiting.
- T_w – mean waiting time in queue, including those that do not wait at all.
- T_s – mean server service time for each item.
- ρ – utilisation, fraction of time that the server is busy, measured over some interval of time.
- q – mean number of items in the system, includes both item being served and waiting items .
- T_q – mean 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 2.6 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 ρ = 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 ρ = 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.

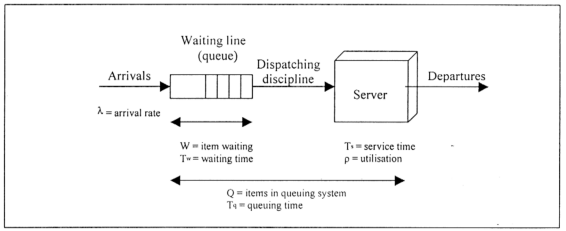

Figure 2.6: Single-server Queue

2.7.2 Multiserver Queue

Figure 2.7 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 utilisation of each server and therefore $N\rho$ is the utilisation of the entire system. The theoretical maximum utilisation is

N X 100% and theoretical maximum input rate is

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

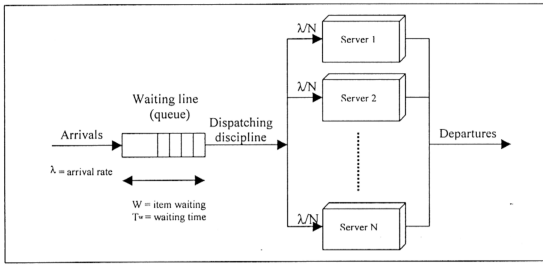

Figure 2.7: Multiserver Queue

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 [28].

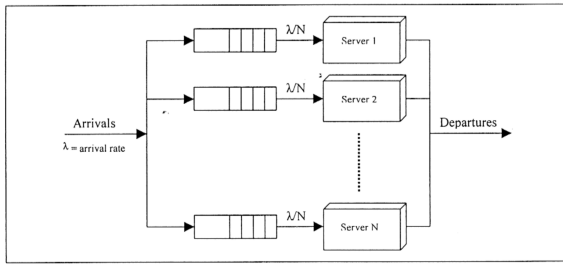

Figure 2.8: Multiple Single-server Queue

2.8 Switching Models

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

2.8.1 Banyan Switch

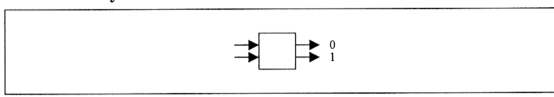

Figure 2.9: 2 X 2 Banyan Network

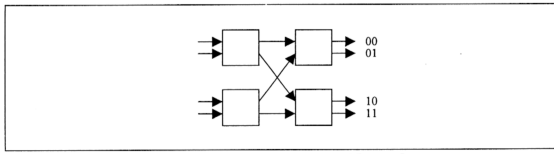

Figure 2.10: 4 X 4 Banyan Network

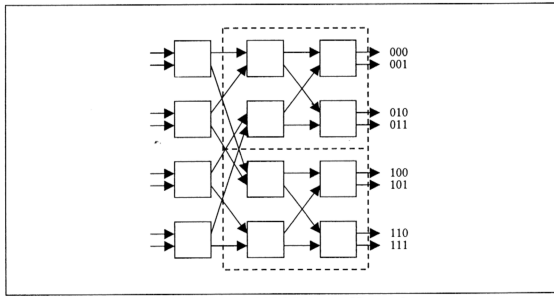

Figure 2.11: 8 X 8 Banyan Network

Banyan switch [29] is one of the most common types of Multistage Interconnection Network (MIN) [30]. The path for the cell to switch is determinate by the output port address, i.e. Banyan switch is self-routing.

Figure 2.9, 2.10, and 2.11 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.

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 suffers 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 [31].

Figure 2.12 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 2.13, 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.

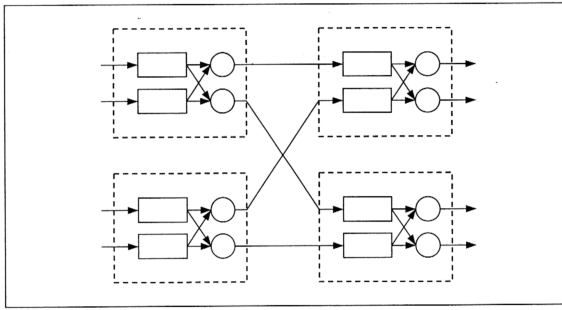

Figure 2.12: Input Queuing Banyan Network

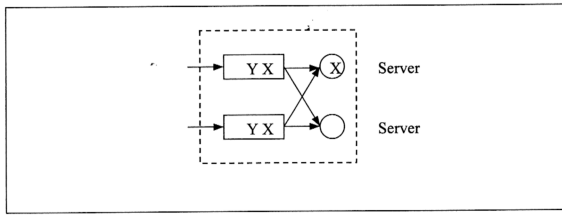

Figure 2.13: Head-of-line Blocking in Input Queuing Banyan Network

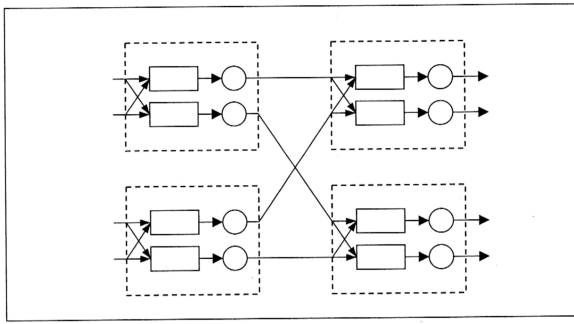

Figure 2.14: Output Queuing Banyan Network

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. [6]

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 [30].

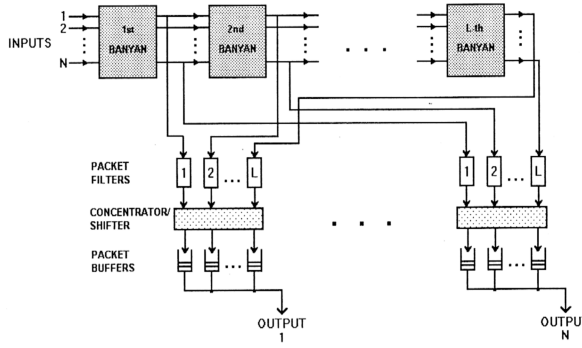

Figure 2.15: Tandem Banyan Switch

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 2.14. 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.

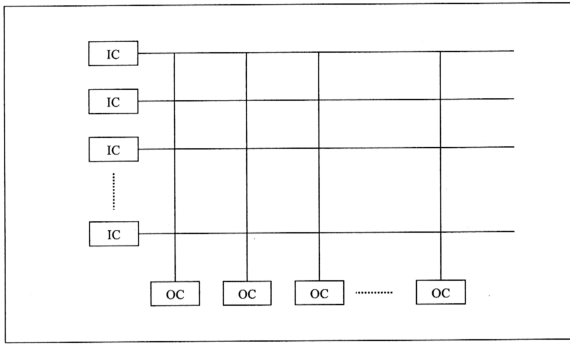

Figure 2.16: N X N Switches

For Knockout switch [33] 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 [6].

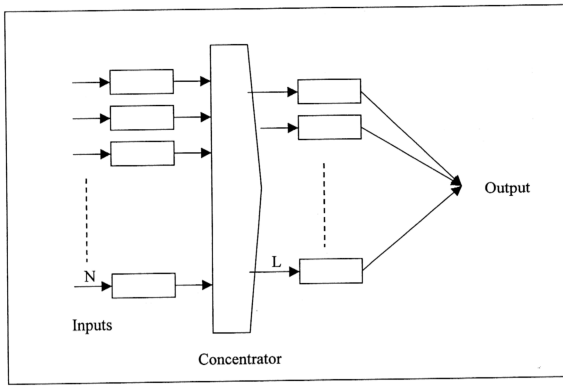

Figure 2.17: Knockout Switch Queuing Model

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 cells 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 [5][6].

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. A more detailed explanation about the Banyan switch and the simulator model will be covered in the following chapter.