Development of Object-oriented Components for ATM Network Simulation with emphasis on Congestion Control

A Thesis

Submitted to the

Faculty of Computer Science and Information Technology
University of Malaya

by

SIN WAI KIT

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 (26 January 2001)

ABSTRACT

Asynchronous Transfer Mode (ATM) is a very high-speed transmission technology. The barrage of technological advancements has continuously enhanced the network performance. There is an increasing dependence on the network. Thus, there is a need to accurately predict the impact of network and applications changes in the dynamic network environment.

The network simulator is a tool to analyze the behavior of ATM networks without the expense of building a real network. The simulation of this simulator will give the results that imitate the real network environment.

The aim of this project is to develop object-oriented components for an ATM Network Simulator that emphasis on congestion control mechanism. In this project, the ATM congestion control system makes use of leaky bucket and token bucket techniques. In addition, the object-oriented approach is used as the construction method for this simulator. The simulator consists of features like reusability, extensibility, portability as well as maintainability.

Furthermore, the simulator is capable of multithreaded operations and platform independent. Thus, the developed simulator can run on different platforms like Windows or Unix.

ACKNOWLEDGEMENT

I would like to take this opportunity to thank those people who had helped me throughout this project.

First of all, I would like to express my gratitude to my respected supervisor, Mr. Ling Teck Chaw for his support, valuable guidance, encouragement and constructive comments during this project.

Special thanks to Mr. Ibrahim for being a considerate and kind moderator who has contributed suggestions and ideas in this project. Besides, I would also like to express my sincere appreciation to my lecturer, Mr. Phang Keat Keong for his brilliant opinions.

Last but not least, my deepest appreciation to my friends, Lim Shiow Hong, Ching Kim Joo, Jimmy Tan Kai Hong, Phung Jacen, Tay Boon Pin, Wong Che Sum, Wong Wing Hong and Yu Soon Lye for their continuous support to complete this project.

TABLE OF CONTENT

LIST OF LIGURES

LIST OF TABLES

Chapter 1: INTRODUCTION

1.1 Introduction to Asynchronous Transfer Mode (ATM)

Asynchronous Transfer mode (ATM) is the primary networking technology for next-generation, multi-media communication. ATM is a high-performance, cell-oriented switching and multiplexing technology that utilizes fixed-length packets to carry different types of traffic (voice, video and data). One most important point in ATM is that it supports quality of service (QoS) requirements.

An ATM cell header can be one of two formats: UNI (user-network interface) or the NNI (network-network interface). The UNI header is used for communication between ATM endpoints and ATM switches in private ATM networks. The NNI header is used for communication between ATM switches. Figure1.1 depicts the basic ATM cell format, the ATM UNI cell-header format, and the ATM NNI cell-header format.

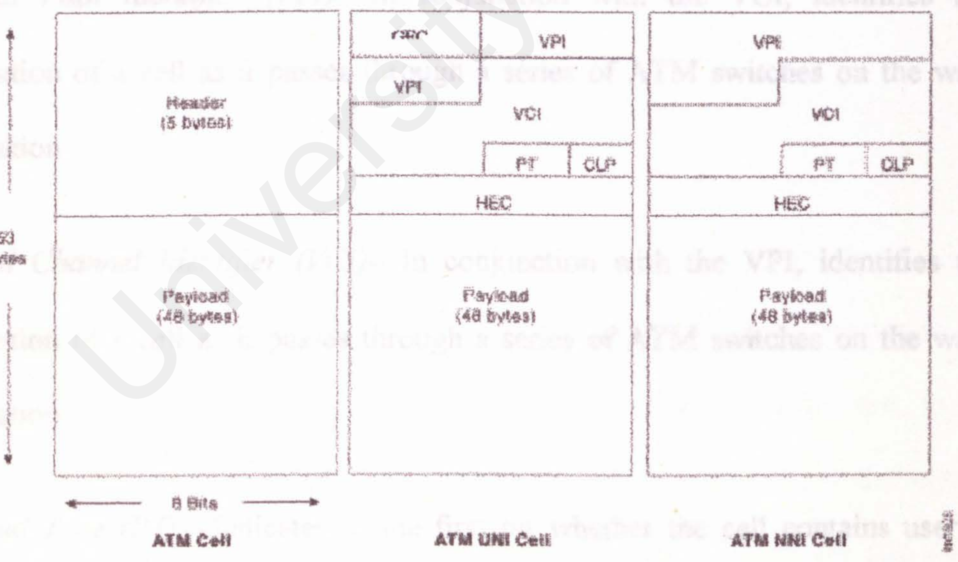


Figure 1.1: An ATM cell, UNI cell, and ATM NNI cell header each contain 48 bytes of payload

ATM transfers information in fixed-size units called cells. Each cell consists of 53 octets, or bytes. The first 5 bytes contain cell-header information, and the remaining 48 contain

the "payload" (user information). Small fixed-length cells are well suited to transfer voice and video traffic because such traffic is intolerant of delays that result from having to wait for a large data packet to download.

Unlike the UNI, the NNI header does not include the Generic Flow Control (GFC) field. Additionally, the NNI header has a Virtual Path Identifier (VPI) field that occupies the first 12 bits, allowing for larger trunks between public ATM switches. In addition to GFC and VPI header fields, several others are used in ATM cell-header fields. The following descriptions summarize the ATM cell-header fields illustrated in Figure 1.1.

- *Generic Flow Control (GFC)*---Provides local functions, such as identifying multiple stations that share a single ATM interface. This field is typically not used and is set to its default value.

- *Virtual Path Identifier (VPI)*---In conjunction with the VCI, identifies the next destination of a cell as it passes through a series of ATM switches on the way to its destination.

- *Virtual Channel Identifier (VCI)*---In conjunction with the VPI, identifies the next destination of a cell as it passes through a series of ATM switches on the way to its destination.

- *Payload Type (PT)*---Indicates in the first bit whether the cell contains user data or control data. If the cell contains user data, the second bit indicates congestion, and the third bit indicates whether the cell is the last in a series of cells that represent a single AAL5 frame.

• *Congestion Loss Priority (CLP)*---Indicates whether the cell should be discarded if it encounters extreme congestion as it moves through the network. If the CLP bit equals 1, the cell should be discarded in preference to cells with the CLP bit equal to zero.

• *Header Error Control (HEC)*---Calculates checksum only on the header itself.

1.2 Introduction to ATM Service Classes

It is very complex to provide desired Qos for different applications. For example, voice is delay-sensitive but not loss-sensitive, data is loss-sensitive but not delay-sensitive, while some other applications may be both delay-sensitive and loss-sensitive.

To make it easier to manage, the traffic in ATM is divided into the following five service classes [1]:

• **CBR** Constant Bit Rate

• **rt-VBR** Real-Time Variable Bit Rate

• **nrt-VBR** Non-Real-Time Variable Bit Rate

• **UBR** Unspecified Bit Rate

• **ABR** Available Bit Rate

These service categories relate traffic characteristics and QoS requirements to network behavior. Functions such as routing, CAC, and resource allocation are, in general, structured differently for each service category. Service categories are distinguished as being either real-time or non-real-time. For real-time traffic, there are two categories: CBR and rt-VBR, while for non-real-time traffic, there are three categories: nrt-VBR, UBR and ABR.

1.2.1 Constant Bit Rate (CBR) Service

The Constant Bit Rate service category is used by connections that request a static amount of bandwidth that is continuously available during the connection lifetime. This amount of bandwidth is characterized by a Peak Cell Rate (PCR) value.

The basic commitment made by the network to a user who reserves resources via the CBR capability is that once the connection is established, the negotiated ATM layer QoS is assured to all cells when all cells are conforming to the relevant conformance tests. Examples of applications that can use CBR are telephone, video conferencing, and television.

1.2.2 Real-Time Variable Bit Rate (rt-VBR) Service

The real-time VBR service category is intended for real-time applications, i.e., those requiring tightly constrained delay and delay variation, as would be appropriate for voice and video applications. rt-VBR connections are characterized in terms of a Peak Cell Rate (PCR), Sustainable Cell Rate (SCR), and Maximum Burst Size (MBS). Sources are expected to transmit at a rate that varies with time. Equivalently the source can be described as "bursty". Cells that are delayed beyond the value specified by maxCTD are assumed to be of significantly reduced value to the application. Example of real-time VBR is interactive compressed video.

1.2.3 Non-Real-Time (nrt-VBR) Service

The non-real-time VBR service category is intended for non-real-time applications which have bursty traffic characteristics and which are characterized in terms of a PCR, SCR, and MBS. For those cells which are transferred within the traffic contract, the application expects a low cell loss ratio. Non-real-time VBR service may support statistical multiplexing of connections. No delay bounds are associated with this service category. Example for non-real-time VBR is multimedia e-mail.

1.2.4 Unspecified Bit Rate (UBR) Service

The Unspecified Bit Rate (UBR) service category is intended for non-real-time applications, i.e., those not requiring tightly constrained delay and delay variation. UBR service does not specify traffic related service guarantees. No numerical commitments are made with respect to the Cell Loss Ratio (CLR) experienced by a UBR connection, or as to the Cell Transfer Delay (CTD) experienced by cells on the connection. The UBR service is indicated by use of the Best Effort Indicator in the ATM User Cell Rate Information Element. Examples of such applications are traditional computer communications applications, such as file transfer and email.

1.2.5 Available Bit Rate (ABR) Service

ABR is an ATM layer service category for which the limiting ATM layer transfer characteristics provided by the network may change subsequent to connection establishment. A flow control mechanism is specified that supports several types of feedback to control the source rate in response to changing ATM layer transfer characteristics. It is expected that an end-system that adapts its traffic in accordance with the feedback will experience a low cell loss ratio and obtain a fair share of the available bandwidth according to a network specific allocation policy.

On the establishment of an ABR connection, the end-system shall specify to the network both a maximum required bandwidth and a minimum usable bandwidth. These shall be designated as peak cell rate (PCR), and the minimum cell rate (MCR), respectively. The MCR may be specified as zero. The bandwidth available from the network may vary, but shall not become less than MCR.

1.3 Introduction to Connection Parameters

1.3.1 Quality of Service (QoS)

The ability for an application to demand QoS from the network is now becoming commercially important. In order to provide a uniform framework for different applications to specify required performance guarantee and for systems to provide the required guarantee, a concept of QoS has been introduced.

A set of parameters is negotiated when a connection is set up on ATM networks. These parameters are used to measure the Quality of Service (QoS) of a connection and quantify end-to-end network performance at ATM layer. The network should garuantee the QoS by meet certain values of these parameters. Those negotiated parameters are as follows [1]:

• **Cell Transfer Delay (CTD)**

The delay experienced by a cell between the first bit of the cell is transmitted by the source and the last bit of the cell is received by the destination. Maximum Cell Transfer Delay (Max CTD) and Mean Cell Transfer Delay (Mean CTD) are used.

• **Peak-to-peak Cell Delay Variation (CDV)**

The difference of the maximum and minimum CTD experienced during the connection. Peak-to-peak CDV and Instantaneous CDV are used.

• **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.

The Cell Loss Ratio is defined for a connection as:

$$\text{CLR} = \frac{\textbf{Lost Cells}}{\textbf{Total Transmitted Cells}}$$

Figure 1.2 below illustrates the probability density function of the CTD in CBR and real-time VBR services, and relates it to the peak-to-peak CDV and maxCTD parameters.

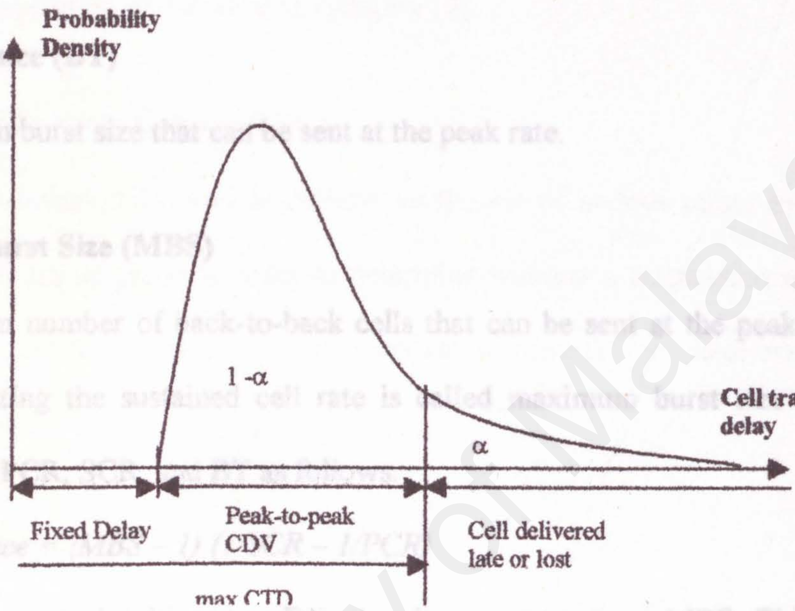


Figure 1.2: Cell transfer delay probability density function (for real-time service categories)

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 (1-α) portions are within the requested QoS. The range between the fixed delay and *maxCTD* is referred to the *peak-to-peak CDV.*

1.3.2 Usage Parameters

Another set of parameters is also negotiated when a connection is set up. These parameters discipline the behavior of the user. The network only provides the QoS for the cells that do not violate these specifications [2].

- **Peak Cell Rate (PCR)**

  The maximum instantaneous rate at which the user will transmit.

- **Sustained Cell Rate (SCR)**

  This is the average rate as measured over a long interval.

- **Burst Tolerance (BT)**

  The maximum burst size that can be sent at the peak rate.

- **Maximum Burst Size (MBS)**

  The maximum number of back-to-back cells that can be sent at the peak cell rate but without violating the sustained cell rate is called maximum burst size (MBS). It is related to the PCR, SCR, and BT as follows:

  Burst Tolerance = (MBS – 1) (1/SCR – 1/PCR)

  Since MBS is more intuitive than BT, signaling messages use MBS. This means that during connection setup, a source is required to specify MBS. BT can be easily calculated from MBR, SCR, and PCR.

- **Minimum Cell Rate (MCR)**

  The minimum cell rate desired by a user.

1.4 Introduction to Traffic Management in ATM Network

ATM technology is intended to support a wide variety of services and applications. The control of ATM network traffic is fundamentally related to the ability of the network to provide appropriately differentiated Quality of Service (QoS) for network applications. A primary role of traffic management is to protect the network and the end-system from congestion in order to achieve network performance objectives. An additional role is to promote the efficient use of network resources.

To meet these objectives, the following generic functions form a framework for managing and controlling traffic and congestion in ATM networks and may be used in appropriate combinations depending on the service category [3].

- ***Connection Admission Control (CAC)***

  Connection Admission Control is defined as the set of actions taken by the network during the call set-up phase in order to determine whether a connection request can be accepted or should be rejected (or whether a request for re-allocation can be accommodated).

- ***Feedback Control***

  Feedback Control are defined as the set of actions taken by the network and by end-systems to regulate the traffic submitted on ATM connections according to the state of network elements.

- ***Usage Parameter Control (UPC)***

  Usage Parameter Control (UPC) is defined as the set of actions taken by the network to monitor and control traffic at the end-system access. Its main purpose is to protect network resources from user misbehavior, which can affect the QoS of other connections, by detecting violations of negotiated parameters and taking appropriate actions.

• ***Cell Loss Priority control***

For some service categories the end system may generate traffic flows of cells with Cell Loss Priority (CLP) marking. The network may follow models which treat this marking as transparent or as significant. If treated as significant, the network may selectively discard cells marked with a low priority to protect, as far as possible, the QoS objectives of cells with high priority.

• ***Traffic Shaping***

Traffic shaping mechanisms may be used to achieve a desired modification to the traffic characteristics of a connection. The objectives of this function are to achieve a better network efficiency whilst meeting the QoS objectives and/or to ensure connection traffic conformance at a subsequent interface.

• ***Network Resource Management***

Network Resource Management (NRM) is responsible for the allocation of network resources in order to separate traffic flows according to different service characteristics, to maintain network performance and to optimize resource utilization. This function is mainly concerned with the management of virtual paths in order to meet QoS requirements.

• ***Frame Discard***

If a congested network needs to discard cells, it may be better to drop all cells of one frame than to randomly drop cells belonging to different frames, because one cell loss may cause the retransmission of the whole frame, which may cause more traffic when congestion already happened. Thus, frame discard may help avoid congestion collapse and can increase throughput. If done selectively, frame discard may also improve fairness.

1.5 Introduction to Congestion Control in ATM network

Traffic management is concerned with ensuring that users get their desired quality of service. The problem is especially difficult during periods of heavy load particularly if the traffic demands cannot be predicted in advance. This is why congestion control, although only a part of the traffic management issues, is the most essential aspect of traffic management.

Congestion control is critical in both ATM and non-ATM networks. When two bursts arrive simultaneously at a node, the queue lengths may become large very fast resulting in buffer overflow. Congestion happens whenever the input rate is more than the available link capacity:

Sum(Input Rate) > Available link capacity

Most congestion control schemes consist of adjusting the input rates to match the available link capacity (or rate). One way to classify congestion control schemes is by the layer of ISO/OSI reference model at which the scheme operates. For example, there are data link, routing, and transport layer congestion control schemes. Typically, a combination of such schemes is used. The selection depends upon the severity and duration of congestion.

Figure 1.3 [4] shows how the duration of congestion affects the choice of the method. The best method for networks that are almost always congested is to install higher speed links and redesign the topology to match the demand pattern.

Congestion Duration	**Congestion Mechanism**
Long	**Capacity planning and network design**
	Connection admission control
	Dynamic routing
	End-to-end feedback
	Link-by-link feedback
Short	**Buffering**

Figure 1.3: Congestion techniques for various congestion durations

1.6 Simulation of ATM Network

With the rapid development of high-speed networks, such as ATM, there is a need to study some of the issues confronting the design of the networks. The performance of these networks needs to be analyzed.

Research has been done for traffic management and congestion control but the traffic patterns that may prevail in the future networks has been rather difficult to predict, and various kinds of assumptions that needed for a theoretical analysis has been difficult to justify. The throughput of different network topologies has to be tested and analyzed in order to maximize the utilization of resources.

Simulation is the basis for making decisions. Decisions are formulated based on the information resulting from the simulation. A Network simulator can be used as a tool for ATM network planning or as a tool for ATM protocol performance analysis. It is useful for modeling network behavior under different conditions and with settings for the various network components. With the use of this network simulator, researchers and network planners are able to analyze networks without the expense of setting up a real network.

1.7 Project Objectives

Traffic control is an essential part in ATM network, which affects the overall performance of the network. Proper traffic management helps in ensuring efficient and fairness operation in the network.

The first objective in this project is to study on the ATM network field especially in congestion control mechanism. The second objective is to develop an ATM network simulator. The needs to simulate and analyze the ATM network are getting more important as the evolution is towards high-speed networks. Since very little is known about the performance of the available ATM switches, they need to be tested to provide maximum usage.

1.8 Project Scope

The project started with a study on ATM which emphasis on congestion control mechanism. The study includes ATM classes of service, connection parameters such as Quality of Service (QoS) and usage parameters. Besides, it also covers the ATM traffic management and congestion control schemes so that the congestion control method can be implemented in the simulator.

Furthermore, the scope covers studies on existing ATM simulators. It includes the analysis on advantages and disadvantages of these simulators. This is to determine the basic requirements of developing a practical simulator.

The review includes the simulator components needed for an ATM network, such as ATM switch, Broadband Terminal Equipment (B-TE), ATM application and physical link in order to get the idea of how to build the ATM simulator.

This project will develop an ATM network simulator that will have the following features:

- A user friendly graphical user interface (GUI)
- Platform independent
- Discrete event schedule
- Integrated data analysis tool or mechanisms recording the simulation results and network configuration
- Default parameters for simulator components

1.9 Project Schedule

The project schedule is shown in Figure 1.4.

Task		Jun	July	Aug	Sep	Oct	Nov	Dec	Jan
Literature Review	13-6-00 To 4-9-00	▨	▨	▨	▨				
System Analysis	8-8-00 To 10-9-00			▨	▨				
System Design	11-9-00 To 20-10-00				▨	▨			
System Implementation	21-10-00 To 20-12-00					▨	▨	▨	
Testing	18-11-00 To 7-1-01						▨	▨	▨
Documentation	19-6-00 To 20-1-01	▨	▨	▨	▨	▨	▨	▨	▨

Figure 1.4: Project Schedule

1.10 Report Organization

This report is organized as follows:

Chapter1 provides an introduction to ATM network, ATM service classes, QoS, traffic management and congestion control. Besides, it includes project objectives and project scope.

Chapter2 provides a study on various available simulators. The advantages and disadvantages of each simulator is pointed out, in which each are analyzed carefully. It then discusses the simulator's platform available and Programming Language that will be used in developing the simulator. Besides, survey on various congestion control schemes is done. Each congestion control scheme is explained in detailed.

Chapter3 gives some overview on the Javasim package, leaky bucket and token bucket. It discusses the major classes used in Javasim package. Besides, it also includes discussion on the features and components in the leaky bucket and token bucket.

Chapter4 includes the system analysis part. The requirement specifications of the simulator are pointed out. Simulator's architecture that includes the major components is discussed. In addition, it focuses deep into the model that will be used in traffic management system, with emphasizing on congestion control for the network.

Chapter5 includes the system design part. The design consideration, which includes the user interface design and classes design, is focused. The leaky bucket and token bucket design is included. The features of the ATM Network Simulator that will be developed are included at the end of the chapter.

Chapter6 covers the system implementation part. The implementation of the ATM Network Simulator, which includes the Leaky Bucket and Token Bucket, is explained in detail.

Chapter7 includes the unit testing and system testing part. The implemented Leaky Bucket and Token Bucket are tested in the testing phase. The strategies used in testing the functionality of the Leaky Bucket and Token Bucket are discussed deeply.

Chapter8 concludes the work on the development of the ATM Network Simulator that emphasizing on congestion control in the switch. It summarizes the project findings, objectives achieved, simulator strength, limitation and future enhancement on the simulator.

Chapter2: LITERATURE REVIEW

This chapter begins with brief introduction to network simulation. A detail description for the simulators that includes their advantages and disadvantages is provided. The following section covers the platform and programming language chosen to develop the simulator. The final section discusses the ATM traffic management that emphasis on congestion control techniques.

2.1 Network Simulation

A network simulator provides a means for researchers and network planners to analyze the behavior of the network without the expense of building a real network. The simulator is a tool that gives the user an interactive modeling environment with a graphical user interface. With this tool the user may create different network topologies, control component parameters, measure network activity, and log data from simulation runs. There are two major uses for the simulator: as a tool for ATM network planning and as a tool for ATM protocol performance analysis [5].

- As a *planning tool*, a network planner can run the simulator with various network configurations and traffic loads to obtain statistics such as utilization of network links and throughput rates of virtual circuits. It could be used to answer questions such as: where will be the bottlenecks in the planned network, what is the effect of changing the speed of a link, and will adding a new application cause congestion. Statistics are reported directly to the screen or logged in a data file for further processing.
- As a *protocol analysis tool*, a researcher or protocol designer could study the total system effect of a particular protocol. For example, one could investigate the effectiveness of various flow control mechanisms for ATM networks and address such

issues as: mechanisms for fair bandwidth allocation, protocol overhead, and bandwidth utilization.

2.2 Introduction to Various Simulators

There are a few existing network simulators available. These simulators run on either the Windows platform or UNIX platform. Study and survey is done on these simulators and the performance and their features are compared. The studies on the simulators include: NIST ATM/HFC Network Simulator, YATS - Yet Another Tiny Simulator (ATM Simulation), REAL Network Simulator, and INSANE ATM Simulator.

2.2.1 NIST ATM/HFC Network Simulator

ATM/HFC Network Simulator was developed at the National Institute of Standards and Technology (NIST) to provide a flexible testbed for studying and evaluating the performance of ATM and HFC networks [5]. The simulator is a tool that gives the user an interactive modeling environment with a graphical user interface. NIST has developed this tool using both C language and the X Window System running on a UNIX platform. This tool is based on a network simulator developed at MIT1 that provides support for discrete event simulation techniques and has graphic user interface (GUI) representation capabilities.

The ATM/HFC Network Simulator allows the user to create different network topologies, set the parameters of component operation, and save/load the different simulated configurations. While the simulation is running, various instantaneous performance measures can be displayed in graphical/text form on the screen or saved to files for subsequent analysis.

2.2.1.1 Advantages

This simulator gives user an interactive modeling environment with graphical user interface (GUI) representation capabilities. User could create different network topologies and simulate these topologies to get different results for comparison purposes. Furthermore, user can save/load different network topologies and log data during execution of simulation.

2.2.1.2 Disadvantages

This simulator consists of too many parameters to be considered during setting up the network topology. User must know well about these parameters before can use it. Besides, user needs to have a strong programming language, especially in C Language in order to customize the simulator's components. This simulator only can run on UNIX or LINUX platform that give a platform limitation problem.

2.2.2 YATS - Yet Another Tiny Simulator (ATM Simulation)

YATS is a small cell-level simulation tool for ATM networks. Its kernel comprises the event scheduler, a symbol manager and a scanner/ parser front end. An input file describes the arbitrary - model network configuration, the simulation actions and the way to analyze the results. The system is written in C++. All network nodes are objects which communicate over standardized messages [6].

2.2.2.1 Advantages

The input language is a simple script language which allows for a flexible problem description (loops, macros and basic mathematical capabilities are provided). The discrete-time event scheduler applies a static calendar queue and unusual event memory management which results in good simulation speed. Graphical object classes are able to display the time dependent behavior of variables and distributions inside of other model objects (without adding complexity to these network objects).

2.2.2.2 Disadvantages

The following limitations mainly stem from design rules aiming at high simulation speed:

- The pure slotted operation causes some restriction when simulating different line speeds in the same model. It's only possible to choose speeds for which the cell transfer time is an integer multiple of a basic time used for the whole model.
- Currently, batch arrivals are not possible. On each connection between two network objects, only one cell (or other data item, for example: a frame) can be transmitted per basic time slot. In order to compose background traffic, therefore always have to multiplex multiple traffic streams in a multiplexer.
- While the language based model description yields a high flexibility, the input may become a bit irritating in case of larger networks. This especially holds, if there is no regularity in the model structure which would allow user to use like loops and macros.

2.2.3 REAL Network Simulator

REAL is a simulator for studying the dynamic behavior of flow and congestion control schemes in packet switch data networks. It provides users with a way of specifying such networks and to observe their behavior.

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.

REAL simulator runs on Sun3s, Sparcs, MIPS boxes, Vaxen and 3B2, under 4.3BSD-like operating systems: SunOS, IRIX, UMIPS, and Ultrix [7].

2.2.3.1 Advantages

REAL Network Simulator provides a flexible testbed for studying the dynamic behavior of flow and congestion control schemes in packet switch data networks. Source code is provided so that interested users can modify the simulator to their own purposes.

2.2.3.2 Disadvantages

This simulator does not give a user an interactive modeling environment with graphical user interface (GUI) representation capabilities. The GUI features only available in REAL version 5.0. New version has Java interface for Web-based simulation. Besides, user must have strong foundation in C Programming Language in order to change the source code provided to modify the simulator to for their own purposes.

2.2.4 INSANE ATM Simulator

INSANE is a network simulator designed to test various IP-over-ATM algorithms with realistic traffic loads derived from empirical traffic measurements [8]. 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, slowstart, flow and congestion control, retransmission, and fast retransmit.

2.2.4.1 Advantages

INSANE is designed to run large simulations whose results are processed off-line. 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). Although there is no graphical user interface, a (optional) Tk-based graphical simulation monitor provides an easy way to check the progress of multiple running simulation processes.

2.2.4.2 Disadvantages

INSANE is restricted to run in a limited hardware and platforms. Besides, it can not run it on its own. It currently requires the following other software packages:

- 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.

Table 2.1 gives a comparison among the studied simulators. Features being compared are discrete event simulation, object-oriented, graphical user interface (GUI), multithread, web-enable and platform independent.

Table 2.1: Comparison among Various Simulators

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

2.3 Simulator's Platform

Most of the currently available simulators are running on UNIX platforms, with only a few that run on Windows platform.

With the rapid development on Windows system, the processing speed of a PC is comparable with a UNIX system and the price of a PC that runs on Windows platform is lower compare to a UNIX workstation. So the Window platform becomes dominant at the recent days. Although the Windows platform becomes more common, the UNIX platform can't be neglected.

A network simulator that is cross-platform is more important as it is supported by most of the platform, for example UNIX or Windows.

2.4 Programming Language

The programming language chosen to be used in developing the simulation model for ATM network simulator depends solely on the features provided by the programming language. Furthermore, the programming language must provide sufficient features to meet the needs of the simulator requirements. Since the simulator to be developed must be platform independent, the programming language used must be satisfied this main requirement.

2.4.1 Java Programming Language

Java Programming Language will be used in developing the simulator as it provides the feature that enables the simulator portable to the World-Wide-Web. On the other hand, the simulator developed with Java Programming Language will be platform independent. Other Java features that make it to be used in this project are as follows:

2.4.1.1 Object-oriented

Java is object-oriented [9]. Object-oriented Programming encapsulates data and methods into objects; the data and methods of an object are intimately tied together. Objects have the property of information hiding. Objects normally are not allowed to know how other objects are implemented. Other key benefits of the Java Object-oriented are:

- Extensibility – Existing objects can be modified to add new features to the system where changes on new objects can be done.
- Reusability – Objects that are used in one system can be used in another newly system without any changes to the objects.

2.4.1.2 Exception Handling

Java exception handling enables a program to catch all types of exceptions, or to catch all exceptions of a certain type or related types. This makes programs more robust by reducing the likelihood that errors will not be caught by a program. Exception handling is provided to enable programs to catch and handle errors rather than letting them occur and suffering the consequences [9].

2.4.1.3 Multithreading

Java is unique among popular general-purpose programming languages in that it makes concurrency primitives available to the applications programmer. Concurrency is very important in a simulation model as there might be many objects doing their own processes at the same time. It is impossible to allow them execute processes in a sequential manner, as this could not be appropriate as compared to real time simulation result.

2.4.2 Programming Language Tool

Since the Java Programming Language has been chose, the programming language tool that supports this language should be used. This tool should include the general features such as a language compiler and debugger, as well as language libraries and graphical user interface in building environment.

Borland JBuilder Enterprise version 3.5 is the choice to develop the ATM network simulator to enable cross-platform development and enabling web-based deployment. JBuilder 3.5 uniquely delivers the key features required for productive Java development [10], including:

- Unrivaled support for the Java 2 platform to deliver the most reliable, scalable, and preferment Java solutions.
- Visual tools and reusable components for rapidly creating platform independent Java applications, servlets, and applets.
- Components Wizards and designers for creating reusable JavaBeans and Enterprise JavaBeans

Besides, JBuilder 3.5 can be used in Windows platform as well as UNIX platform. Therefore, the simulator built with this tool can be used in both platforms too.

2.5 Congestion Control Schemes in ATM networks

Congestion control lies at the heart of the general problem of traffic management for ATM networks [11]. Congestion happens when the demand of resource exceeds the availability. There is two type of mechanism to handle congestion: *Congestion control*, which is done when the network is overloaded, and *Congestion avoidance*, which is done before the network is overloaded or when congestion is predictable. There are several congestion control schemes that have been proposed.

2.5.1 Fast Resource Management

This method [12] requires sources to send a resource management (RM) cell requesting the desired bandwidth before actually sending the cells. If a switch cannot grant the request it simply drops the RM cell; the source times out and resends the request. If a switch can satisfy the request, it passes the RM cell on to the next switch. Finally, the destination returns the cell back to the source which can then transmit the burst. The burst has to wait for at least one round trip delay at the source even if the network is idle (as is often the case). To avoid this delay, an immediate transmission (IT)" mode was also proposed in which the burst is transmitted immediately following the RM cell. If a switch cannot satisfy the request, it drops the cell and the burst and sends an indication to the source.

2.5.2 Backward Explicit Congestion Notification (BECN)

This proposed method consists of switches monitoring their queue length and sending an RM cell back to source if congested [13]. The sources reduce their rates by half on the receipt of the RM cell. If no BECN cells are received within a recovery period, the rate for that VC is doubled once each period until it reaches the peak rate. To achieve fairness, the source recovery period was made proportional to the VC's rate so that lowers the transmission rate the shorter the source recovery period. This scheme was dropped because it was found to be unfair. The sources receiving BECNs were not always the ones causing the congestion.

2.5.3 Delay-based Rate Control

This method requires that the sources monitor the round trip delay by periodically sending resource management (RM) cells that contain timestamp. The cells are returned by the destination. The source uses the timestamp to measure the roundtrip delay and to deduce the level of congestion. This approach has the advantage that no explicit feedback is expected from the network and, therefore, it will work even if the path contained non-ATM networks [14].

2.5.4 Early Packet Discard

This method based on the observation that a packet consists of several cells. It is better to drop all cells of one packet then to randomly drop cells belonging to different packets. In AAL5, when the first bit of the payload type bit in the cell header is 0, the third bit indicates end of message (EOM). When a switch's queues start getting full, it looks for the EOM marker and it drops all future cells of the VC until the end of message marker is seen again.

It may not be fair in the sense that the cell to arrive at a full buffer may not belong to the VC causing the congestion. Note that this method does not require any inter-switch or source-switch communication and, therefore, it can be used without any standardization.

2.5.5 Partial Packet Discard

Partial Packet Discard (also called Tail Packet discard) is employed when the UPC in a switch determines that a cell must be dropped [15]. Whenever it has to discard a cell due to buffer overflow, all subsequent cells (except the last cell) belonging to the same packet will be discarded. This method improves the network utilization since it drops half the packets. Although this method provides better performance than TCP over plain ATM, but it is not as effective because it only discards the "tail end" of the datagram.

2.5.6 Link Window with End-to-End Binary Rate

This method consists of combining good features of the credit-based and rate-based proposals. It consists of using window flow control on every link and to use binary (EFCI-based) end-to-end rate control. The window control is per-link (and not per-VC as in credit-based scheme). It is, therefore, scalable in terms of number of VCs and guarantees zero cell loss. It was no accepted since it contained elements from both credit-based and rate-based camp.

2.5.7 Fair queuing with Rate and Buffer feedback

This method [17] consists of sources periodically sending RM cells to determine the bandwidth and buffer usage at their bottlenecks. The switches compute fair share of VCs. The minimum of the share at this switch and that from previous switches is placed in the RM cells. The switches also monitor each VC's queue length. The maximum of queue length at this switch and those from the previous switches is placed in the same RM cell.

2.5.8 Traffic Policing (Usage Parameter Control)

Traffic policing, also known as Usage Parameter Control (UPC), is a method of ensuring fair allocation of network resources and of assessing the cells entering the switch for conformance with pre-established traffic bandwidth contracts. Those cells that exceed the specified contract are "tagged" or "dropped", depending on what is defined in the contract. This ensures that the connections with reserved bandwidth are not exceeding their reservations.

2.5.8.1 Non-conforming Cells: Tagging vs. Dropping

It is important to understand the concept of tagging and dropping. Each ATM cell has a Cell Loss Priority (CLP) bit which indicates if the network can drop it under congested conditions. When the CLP bit is set to 0 (or CLP=0), the cell is assessed for compliance with traffic parameters associated with the CLP=0 stream. If the parameters dictate that non-compliant cells should be "tagged", the CLP bit is set to 1 (or CLP=1) by the UPC contract, which means that upon experiencing congestion further in the network, these CLP=1 cells are dropped in preference to CLP=0 cells.

2.5.8.2 UPC Traffic Contract Parameters

The ATM Forum has defined different types of traffic contracts to be used in conjunction with leaky buckets. The parameters that make up these types of contracts are defined as follows:

- pcr0 – PCR for cells with CLP=0
- pcr01 – PCR for the aggregate of the CLP=0 cells and the CLP=1 cells (all cells)
- scr0 – SCR for cells with CLP=0
- scr01 – SCR for the aggregate of the CLP=0 cells and the CLP=1 cells (all cells)
- mbs0 – MBS for cells with CLP=0
- mbs01 – MBS for the aggregate of the CLP=0 cells and the CLP=1 cells (all cells)
- tag – sets CLP bit=1 for CLP=0 cells that fail the PCR0 test for CBR0 contracts or the SCR0/ MBS0 test for VBR contracts

The specific combinations of these parameters that make up the ATM Forum contracts are defined as follows:

- cbr <pcr01>
- cbr0 <pcr0> <pcr01> [tag]

- vbr <prc01> <scr01> <mbs01>
- vbr01 <pcr01> <scr0> <mbs0> [tag]
- abr <pcr01> <mcr>

The cbr <pcr01> contract is for CBR traffic. It only uses the leaky bucket to assess the conformance to PCR of the aggregate of the CLP=0 cells and the CLP=1 cells. Cells which fail the PCR CLP=0+1 test are discarded.

The cbr0 <pcr0> <pcr01> [tag] contract is for CBR traffic. It uses the first leaky bucket to assess the conformance to PCR of the CLP=0 cells. It uses the second leaky bucket to assess the conformance to PCR CLP=0 and the CLP=1 cells. If the tag option is set, the cells which fail the PCR CLP=0 test are tagged as CLP=1 and passed on to the second leaky bucket to be tested for PCR CLP=0+1 conformance. Cells which fail the PCR CLP=0 test are discarded and cells which fail the PCR test on CLP=0+1 are discarded.

The vbr <prc01> <scr01> <mbs01> contract is for VBR traffic. The first leaky bucket assesses the conformance to PCR of the aggregate of CLP=0 cells and the CLP=1 cells and the second leaky bucket assesses the conformance to SCR and BT of this same combination. Cells which fail the PCR test are dropped. Cells which pass the PCR test, but which fail SCR an BT test are dropped.

The vbr01 <pcr01> <scr0> <mbs0> [tag] contract is for VBR traffic. It uses the first leaky bucket to assess the conformance to PCR of the aggregate of the CLP=0 and the CLP=1 cells. Cells that fail this test are discarded. It uses the second leaky bucket to assess the conformance to SCR and BT of the CLP=0 cells. If the tag option is set, the cells which fail the SCR and BT CLP=0 test are tagged as non-conforming cells. If the tag option is not set, cells which fail the SCR and BT CLP=0 test are discarded.

The abr <pcr01> <mcr> contract is for ABR traffic. It only uses the first leaky bucket to assess the conformance to PCR of the aggregate of the CLP=0 cells and the CLP=1 cells. Cells which fail the PCR CLP=0+1 test are discarded. The MCR is the guaranteed minimum rate and the PCR is the maximum required rate. The MCR may be set to 0, which indicates "best effort" service. If MCR is set greater than 0, then the rate is guaranteed, up to MCR. However, between MCR and PCR, cells may be dropped when congestion is experienced.

2.6 Chapter Summary

This chapter has covered the review of various network simulators available. The review includes the strengths and weakness of these simulators. It also discusses on programming language approaches. Besides, it has covered the ATM network traffic management techniques especially in congestion control.

The network simulator will be developed with Java Programming Language by using Jbuilder 3.5 Programming Tool. This simulator will be platform independent, which means it can be used in either Windows platform or UNIX platform.

The next chapter will discuss the overview of javasim package, Leaky Bucket and Token Bucket.

Chapter3: JavaSim Package, Leaky Bucket and Token Bucket Overview

3.1 Javasim Package Overview

The existing ATM Network Simulator is developed in a package called *javasim*. This package consists of all information and classes for the execution of a simulator which includes classes like CBRApp, Cell, GenericATMSwitch, GenericBTE, GenericLink, JavaSim, NodeID, NSAP, PGID, SimClock, SimComponent, SimEvent, SimLog, SimMeter, SimPanel, SimParamBool, SimParamDouble, SimParameter, SimParamInt, SimParamNSAP, SimParamRTable, SimProvider and finally VBRApp.

3.1.1 Overview of the javasim package hierarchy

There are a few classes that are the main part of the simulation engine while the others are just inherit from these classes. Among these classes, JavaSim, SimClock, SimComponent, SimEvent, SimLog, SimMeter, SimPanel, SimParameter and SimProvider are the main classes in the simulator.

The JavaSim class is the main object of the simulator. It keeps a list of all the network components and a list of events. The GenericATMSwitch, GenericLink, GenericBTE, CBRApp and VBRApp classes are inherited from the SimComponent class while SimParamInt, SimParamDouble, SimParamBool, SimParamNSAP and SimParamRTable classes are inherited from SimParameter class. The hierarchy of all the significant objects in the simulator is shown in Figure 3.1.

All classes within the dotted rectangle in Figure 3.1 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.

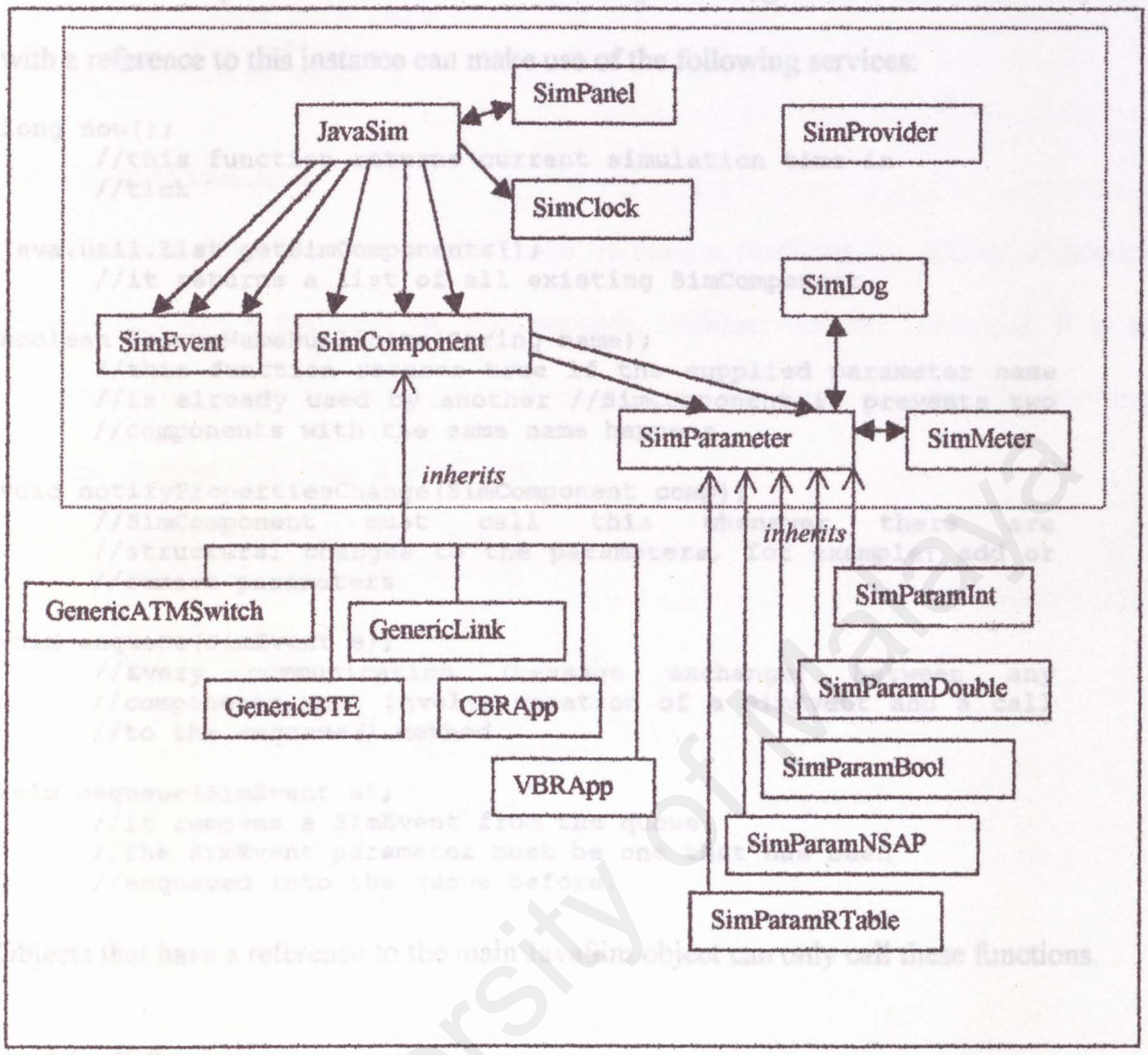


Figure 3.1: Hierarchy of all the significant objects in the simulator

3.1.1.1 JavaSim.java

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

- 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 the 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 happens

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

void dequeue(SimEvent e);
      //it removes a SimEvent from the queue.
      //The SimEvent parameter must be one that has been
      //enqueued into the queue before.
```

Objects that have a reference to the main JavaSim object can only call these functions.

3.1.1.2 Cell.java

The Cell class is a data resource class used by components like CBRApp, VBRApp, 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
  int pti=0;  // payload type identifier
  int CLP=0;  // cell loss priority
```

At the initial stage, *vpi*, *vci*, *pti* and *CLP* values are set to zero. The payload type identifier is used by the ATM end system to determine how to handle incoming cells. The pti value can be 0,1,2 or 3.

```
// 0 - last data cell
  // 1 - not last data cell
  // 2 - forward RM cell
  // 3 - backward RM cell
```

3.1.1.3 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 is as follows:

```
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
```

3.1.1.4 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 the following functions:

```
int getType();
     //gets the event type

SimComponent getSource();
     //gets the source SimComponent

SimComponent getDest();
     //gets the destination SimComponent

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

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

3.1.1.5 SimComponent.java

SimComponent class is the most important class in the simulator in order to develop new components. Network components like GenericATMSwitch, GenericLink, GenericBTE, CBRApp 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,
     //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 are 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 or VBRApp) only allowed to connect
      //   to B-TE.
      //2) B-TE only allowed to connect to Application (CBRApp or
      //   VBRApp) 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, start and resume.

```
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 click
      //the "start" button.

void resume();
      //One possible use of this function is to capture any
      //special changes that have been done by the user during the
      //pause period. It takes action when user clicks the
      //"Resume" button after a pause.
```

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.
```

3.1.1.6 SimParameter.java

Classes like SimParamInt, SimParamDouble, SimParamBool, SimParamNSAP and SimParamRTable 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 the owns that parameter
      //creationTick - time when the parameter is created created
      //isLoggable - whether the parameter can be logged in the
      //             log file
```

3.2 Leaky Bucket Overview

The most famous algorithm for traffic shaping is leaky bucket algorithm . This is a congestion avoidance method in which it monitors the traffic to prevent congestion happens. Leaky buckets are a mechanism by which cells entering the switch fabric are monitored for compliance with UPC traffic contracts that have been negotiated at connection set-up time. Before discussing the leaky buckets, it is important to understand

the parameters that are being measured by the buckets. These parameters are informally defined as follows:

- Peak Cell Rate (PCR)
- Cell Delay Variation Tolerance (CDVT)
- Sustainable Cell Rate (SCR)
- Burst Tolerance (BT)
- Minimum Cell Rate (MCR)

This method provides a pseudo-buffer. Whenever a user sends a cell, the queue in the pseudo-buffer is increased by one. The pseudo-server serves the queue and the service-time distribution is constant. Thus there are two control parameters in the algorithm: the service rate of the pseudo-server and the pseudo-buffer size.

As long as the queue is not empty, the cells are transmitted with the constant rate of the service rate. So the algorithm can receive a bursty traffic and control the output rate. If excess traffic makes the pseudo-buffer overflow, the algorithm can choose discarding the cells or tagging them with CLP=1 and transmitting them. PCR or SCR can be controlled by choosing appropriate values of service rate and buffer size. In addition, PCR and SCR can both be controlled by combining two buckets with one for each of the parameters.

The leaky bucket algorithm is basically a timer which assesses if cells entering the switch fabric conform to the parameters listed above. As a cell arrives, the timer assesses if the cell is on time, late or early. If the cell is determined to be on time or late (based on the traffic parameters), the cell is allowed to pass unchanged. If the cell is early (which, in turn, causes the cell stream to exceed the specified parameters), the cell is considered non-conforming and is either dropped or tagged (the CLP bit is set to 1), depending on the specified contract. This is known as continuous-state leaky bucket algorithm. This algorithm is shown is Figure3.2.

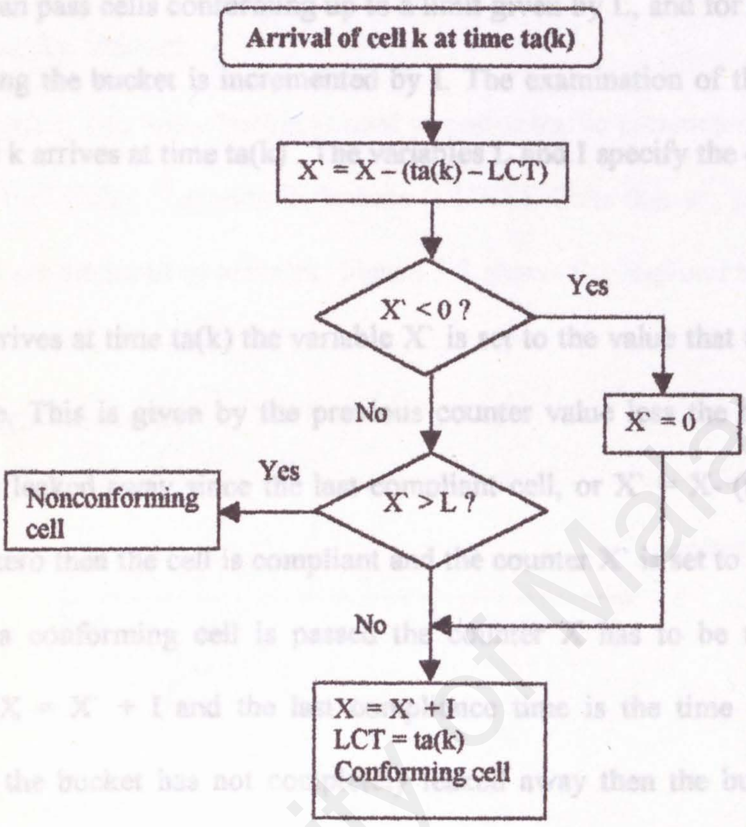


Figure 3.2: Continuous-state leaky bucket algorithm

Definition of the variables used in the Figure 3.1 is:

I = Increment

L = Limit

ta(k) = Time of arrival of a cell

X = Value of the leaky bucket counter

X' = Auxiliary variable

LCT = Last compliance time

There are a number of variables needed to define the algorithm. The Last Compliance Time (LCT) is defined to be the last time that a conforming cell passed through the algorithm. How full the bucket is given by X, the leaky bucket counter, and there is an auxiliary variable X` for predicting the value of the leaky bucket counter. The leaky bucket counter can pass cells conforming up to a limit given by L, and for each cell that is passed conforming the bucket is incremented by I. The examination of the leaky bucket occurs when cell k arrives at time ta(k) . The variables L and I specify the operation of the algorithm.

When a cell k arrives at time ta(k) the variable X` is set to the value that the counter will have at this time. This is given by the previous counter value less the amount that the bucket will have leaked away since the last compliant cell, or X` = X- (ta(k) - LCT). If this is less than zero then the cell is compliant and the counter X` is set to zero and passes the cell. When a conforming cell is passed the counter X has to be updated by the increment I by X = X` + I and the last compliance time is the time of this cell, or LCT = ta(k). If the bucket has not completely leaked away then the bucket has to be checked to see if the limit is going to be exceeded or X` ≥ L. If the limit is not exceeded then the cell is conforming and the cell can be passed as before. Otherwise the cell is non-compliant and no updates are done on the variables.

The operation of the leaky bucket is that a splash is added to the bucket (counter increment) for each incoming cell when the bucket is not full. When the bucket is full cells cannot pass through to the network un-marked but the bucket leaks away at a constant rate. It is assumed here that if the cells cannot pass the leaky bucket without being marked then they are lost because they are non-conforming. These lost cells do not count in the cell loss rate as the cell loss rate is only specified for comforming cells. This is because the network will only give guarantees to the conforming or un-marked cells.

There are two types of leaky bucket: Single Leaky Bucket and Double Leaky Bucket. These leaky buckets will be explained in the following section.

3.2.1 Single Leaky Bucket

In single leaky bucket, one leaky bucket is used to police traffic parameters like Peak Cell Rate (PCR) and Cell Delay Variation Tolerance (CDVT). Cells that are conforming with this leaky bucket are admitted to network. Figure 3.3 shows the implementation of Single Leaky Bucket.

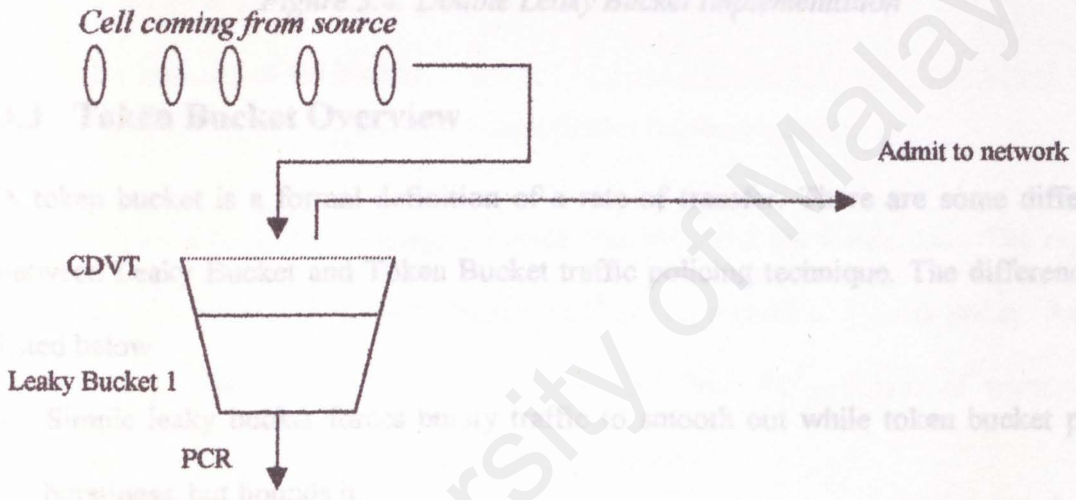


Figure 3.3: Single Leaky Bucket Implementation

3.2.2 Double Leaky Bucket

Two leaky buckets are used in double leaky bucket. Each leaky bucket controls different parameters in the simulator. The first leaky bucket controls the parameters like Peak Cell Rate (PCR) and Cell Delay Variation Tolerance (CDVT) while the second leaky bucket controls the Sustained Cell Rate (SCR) and Burst Tolerance (BT). Only cells that are conforming with both leaky buckets are admitted to the network. Figure 3.4 shows the implementation of Double Leaky Bucket.

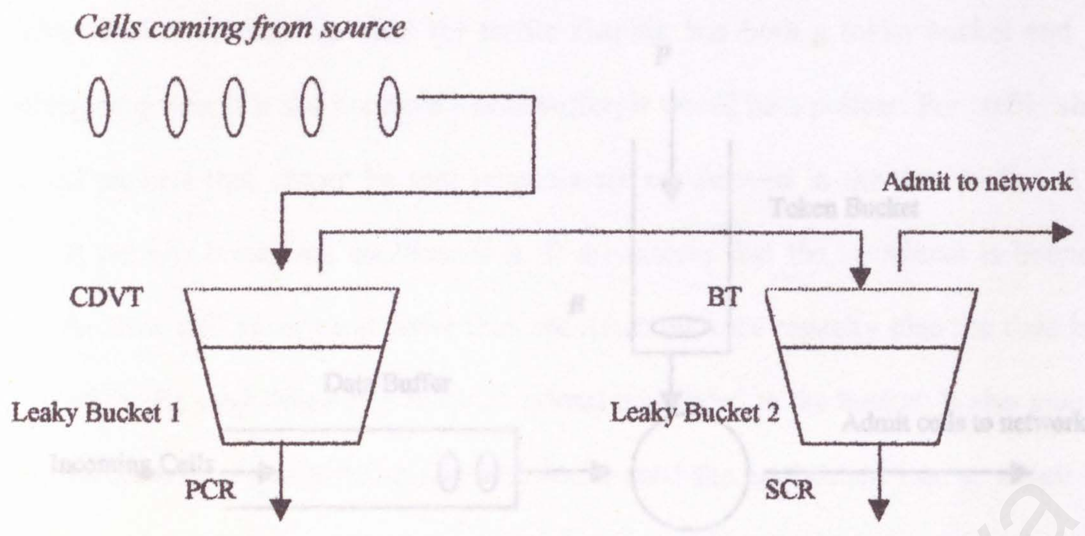


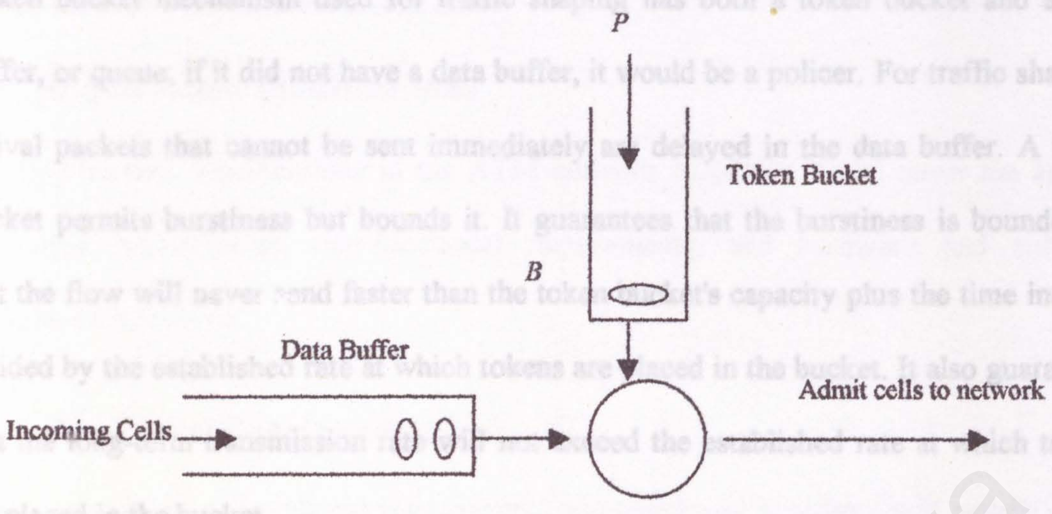
Figure 3.4: Double Leaky Bucket Implementation

3.3 Token Bucket Overview

A token bucket is a formal definition of a rate of transfer. There are some differences between Leaky Bucket and Token Bucket traffic policing technique. The differences are listed below:

- Simple leaky bucket forces bursty traffic to smooth out while token bucket permits burstiness, but bounds it.
- Simple leaky bucket guarantees that the flow will never send faster than total worth of packets per second; token bucket guarantees that the burstiness is bounded so that the flow never sends more than tokens worth of data in an interval time and the long-term transmission rate will not exceed.
- Another difference between leaky bucket and token bucket is that token bucket has no discard or priority policy.

Token bucket is easy to implement. Each flow needs just a counter to count tokens and a timer to determine when to add new tokens to the counter. Figure 3.5 shows the implementation of Token Bucket.



P – rate at which tokens are placed in the bucket

B – capacity of the bucket

Figure 3.5: Token Bucket Implementation

A token bucket is used to manage a device that regulates the flow's data. The regulator might be a traffic policer. A token bucket itself has no discard or priority policy. Rather, a token bucket discards tokens and leaves to the flow the problem of managing its transmission queue if the flow overdrives the regulator.

In the token bucket metaphor, tokens are put into the bucket at a certain rate. The bucket itself has a specified capacity. If the bucket fills to capacity, newly arriving tokens are discarded. Each token is permission for the source to send a certain number of bits into the network. To transmit a packet, the regulator must remove from the bucket a number of tokens equal in representation to the packet size.

If not enough tokens are in the bucket to send a packet, the packet either waits until the bucket has enough tokens or the packet is discarded. If the bucket is already full of tokens, incoming tokens overflow and are not available to future packets. Thus, at any time, the largest burst a source can send into the network is roughly proportional to the size of the bucket.

Token bucket mechanism used for traffic shaping has both a token bucket and a data buffer, or queue; if it did not have a data buffer, it would be a policer. For traffic shaping, arrival packets that cannot be sent immediately are delayed in the data buffer. A token bucket permits burstiness but bounds it. It guarantees that the burstiness is bounded so that the flow will never send faster than the token bucket's capacity plus the time interval divided by the established rate at which tokens are placed in the bucket. It also guarantees that the long-term transmission rate will not exceed the established rate at which tokens are placed in the bucket.

3.4 Chapter Summary

This chapter has covered the overview of JavaSim Package, Leaky Bucket and Token Bucket. The explanation of Leaky Bucket is separated into two parts; Single Leaky Bucket and Double Leaky Bucket. Next chapter will discuss the system analysis part.

Chapter4: SYSTEM ANALYSIS

4.1 Requirement Specifications

The requirement specifications in the ATM network simulator model cover the area of functional requirements, non-functional requirements, and hardware and software requirements analysis.

4.1.1 Functional Requirements

With this analysis of functional requirements, the developed simulator will satisfy user's needs and requirements.

The network to be simulated consists of several components sending messages to each other. The network simulator should include the basic components like ATM switch, Broadband Terminal Equipment (B-TE), ATM Applications and Physical Link. The further descriptions of these components are as follows:

4.1.1.1 Network minimum components

The simulator should at least have the minimum components like Switch, Broadband Terminal Equipment (B-TE), ATM Application and Physical Link for user to build the ATM network topology.

• Switch

This is the component used to switch or route cells over several virtual channel links. W hen a switch accepts an incoming cell from a Physical Link it looks in its routing table to determine which outgoing link should send it. If the outgoing link is busy, the switch will queue the cells destined for that link and not send them until free cell slots are available for transmission. The user may specify the processing delay time, maximum output queue size, and queue size thresholds. The parameters that can be monitored for a switch include the number of cells received, number of cells in an output queue, number of cells dropped, and the status of congestion flags.

- Broadband Terminal Equipment (B-TE)

This is a component to simulate a broadband ISDN node, e.g., host computer, workstation, etc. A B-TE component has one or more ATM Applications on one side and a physical link on the other side. Cells received from the Application side are forwarded to the physical link; if the link is busy the cells go into a queue. The user can specify the maximum output queue size. The parameters that can be monitored are the number of cells in an output queue and the number of cells dropped.

- ATM Application

This is a component to emulate the behavior of an ATM application at the end-point of a link. It can be considered as a traffic generator, either with a constant or variable bit rate. The user specifies the bit rate for constant bit rate (CBR) applications. For variable bit rate (VBR) applications the user sets the burst length, interval between bursts, and the mean rate. For lower priority traffic, the user may create an available bit rate (ABR) application. For all of the application types, the user sets the start time and the number of megabytes to be sent. Other application types that can be simulated include TCP/IP, VBR MPEG, and VBR self-similar traffic applications.

- Physical Link

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).

4.1.1.2 Interface

An interface is to display the network configurations. This workspace is used while creating the network topology and to show the network activity while the simulation is in progress.

4.1.1.3 Simulation Analysis

The simulator should provide analysis for the simulation result. Output of the simulation should be captured and show to user in either text format or graphical format.

The logging file technique will be used in the text format. This technique enables user to analyze the simulation result after simulation. Output parameter values may be tagged for logging to a file. User can determine the data logging frequency by specifying the value in *logging every n tick* parameter. For example, if user specifies logging every 100 ticks, then the simulator will log data to a file every 100 ticks if there is some changes. The logging frequency should not be too small because it will affect the simulation speed.

The meter will be used in the graphical format. This technique enables user to analyze the simulation result during simulation. User can view the real-time simulation result as the simulation running. Thus, user can modify the simulation parameters to get the best simulation result for the designed topology.

4.1.2 Non-Functional Requirements

The following are the basic non-functional requirements.

4.1.2.1 Physical Environment

This simulator will be developed using SUN Solaris SPARC processor 440Mhz with memory size 128MB. Since this simulator will be platform independent, it can be used in Window environment too. For the use ATM Network Simulator in Windows environment, at least 80486-compatible PC and 16 Mb of RAM is recommended in order to run the simulator.

4.1.2.2 Users and human factor

- For novice user, the knowledge about the relation and concept of network components and network topology is needed.
- The user should understand that the basic requirements in creating a simulation topology must have a set of network components and route.
- User also should understand some parameters that are used in the simulator. For example link speed and link distance in the Physical Link component.

4.1.2.3 Reusability

The components and objects of the simulator should be easily changed in future for redesign purposes. The components and objects should be independent so that modification of the simulator will just require minimum steps.

4.1.2.4 Meeting the user requirement

The final developed simulator must meet the objective, specification and requirement of the users. The simulator must simulate the network topology that imitates the real network environment.

4.2 Simulator Architecture

4.2.1 Simulation Clock Component

The simulator is event driven. Components send each other events in order to communicate and send cells through network. The simulator contains an event manager which provides a general facility to schedule and send or "fire" an event. With the use of multithreading operation, the simulation model is acting much more like the real world simulator. Each object can deal concurrent execution with other objects while at the same time handle their own events. Since this ATM network simulator is using multithreading method, a global clock component is important in synchronizing the processes of objects that run simultaneously. Therefore, there is a need to have a simulation clock in this simulator to handle the simulation timing and synchronization.

The basic architecture of the ATM Network Simulator is shown in *Figure 4.1*.

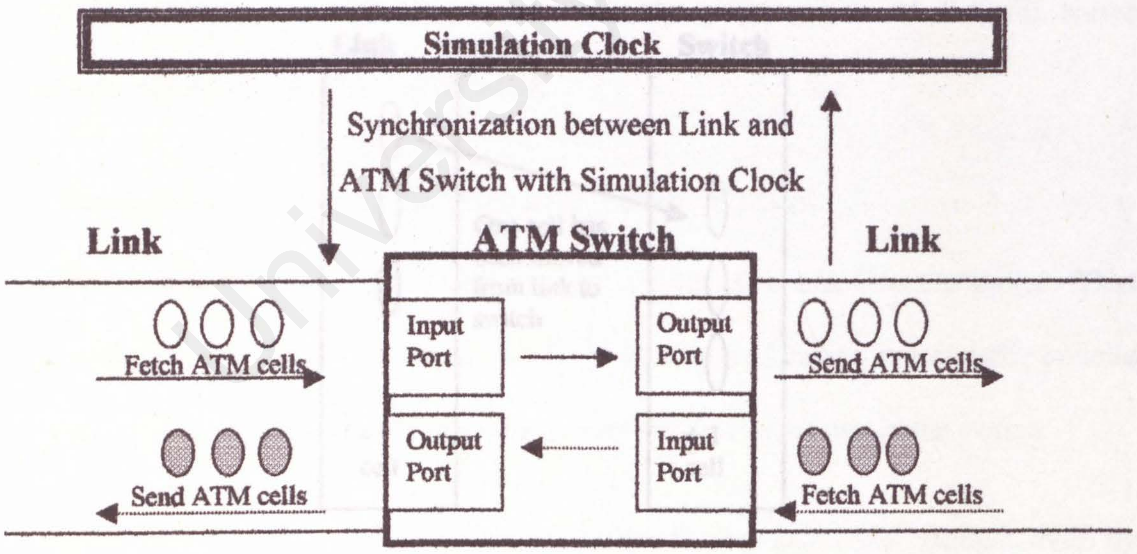


Figure 4.1: ATM Network Simulator architecture

4.2.2 Link Component

This component simulates the physical medium (copper wire or optical fiber) on which cells are transmitted. The length of the link is specified in unit of Kilometer (KM) while the link speed is specified in unit of bit rate (Mbps). This component holds the process of connecting the end systems and switch.

4.2.3 Inter-objects Communication

Inter-objects Communication enables the objects in the simulator to communicate and interact with each other. Message can be passed among objects. For example, when a cell enters a link, it will queue in the link queue. Then this cell will enter the input port of the ATM switch and being passed to the output port.

Therefore, interaction has happened between the link and the switch. The queue of the link will decrement one cell while the switch will increment one cell when a cell is being passed from link queue to switch.

Figure 4.2 shows the interaction between the link and the switch.

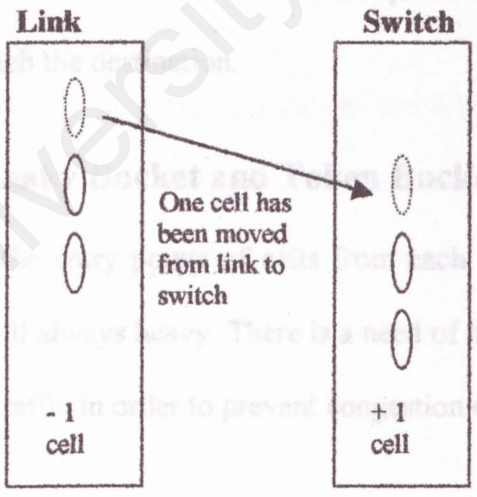


Figure 4.2: Interaction between link object and switch object

Based on the inter-object communication feature, each object in the simulator can communicate or pass message with other objects.

4.3 Queuing Model

In this project, ATM cells in the input port are queued-up using First In First Out (FIFO) queuing discipline. FIFO is a simple queuing discipline. The new incoming cells into the input port of the switch will be placed at the end of the queue. As long as the queue in input port is not full, the ATM cells are possible to be fetched in. Figure 4.3 shows the FIFO queuing model in input port.

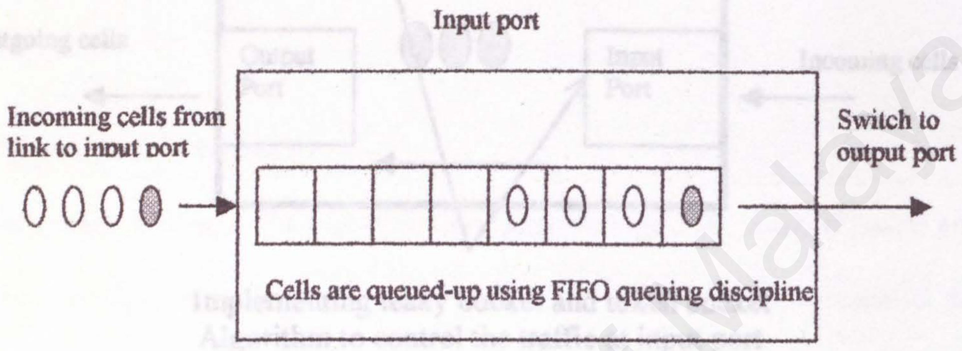

Input port
Incoming cells from link to input port
Switch to output port
Cells are queued-up using FIFO queuing discipline

Figure 4.3: FIFO queuing model in input port.

From this input port, the cells are removed from the queue and switched to the correct output port and finally reach the destination.

4.4 Implementing Leaky Bucket and Token Bucket

Switch's input ports are the entry points of cells from each link into the switch. Thus, traffic at the input ports will always heavy. There is a need of having some traffic policing techniques to control the traffic in order to prevent congestion occurs in the switch.

The traffic policing techniques used includes Leaky Bucket and Token Bucket. With the implementation of leaky bucket and token bucket algorithm in the input port, the traffic can be monitored and congestion can be control. Therefore, it guarantees the QoS while fully utilized the network resource.

Figure 4.4 shows the implementation of leaky bucket and token bucket algorithm at input port.

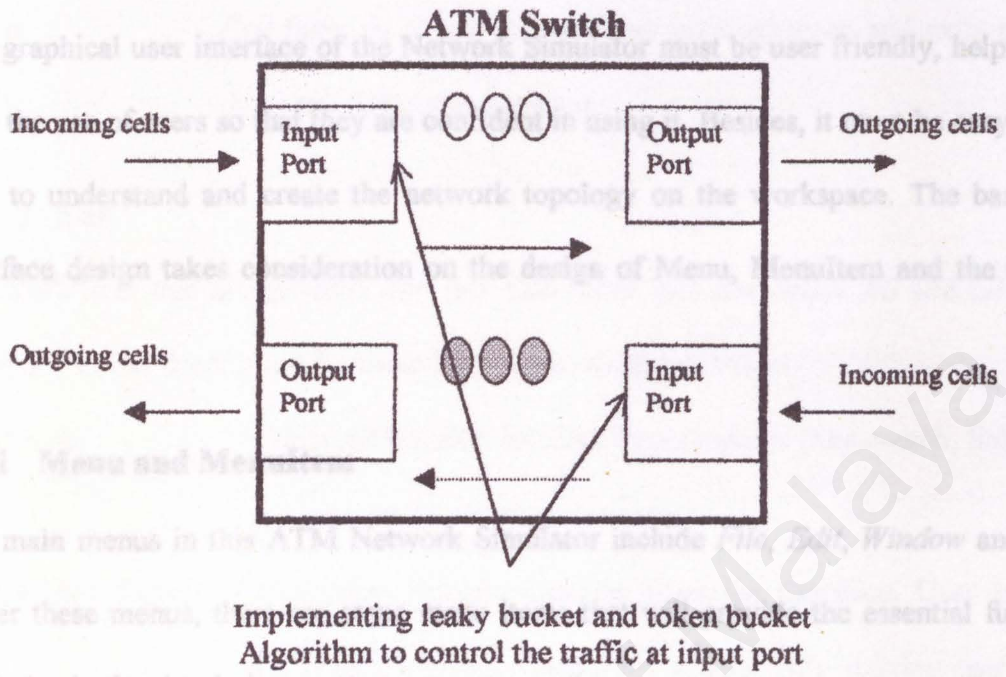


Figure4.4: Implementing leaky bucket and token bucket at input port

4.5 Chapter Summary

This chapter has covered the system analysis part in the ATM Network Simulator. The context and architecture of the entire simulation model was explained in this chapter. Analysis has been done on the queuing model in the input port. Besides, the Leaky Bucket and Token Bucket traffic policing techniques that will be implemented is analyzed and described in detail.

Chapter5: SYSTEM DESIGN

5.1 User Interface Design

The graphical user interface of the Network Simulator must be user friendly, helpful, and ease the use of users so that they are confident in using it. Besides, it must be easy for the user to understand and create the network topology on the workspace. The basic user interface design takes consideration on the design of Menu, MenuItem and the Control Bar.

5.1.1 Menu and MenuItem

The main menus in this ATM Network Simulator include *File*, *Edit*, *Window* and *Help*. Under these menus, there are some menu items that will provide the essential functions and helps in the simulation process.

5.1.1.1 File Menu

The MenuItem that are grouped into this *File Menu* includes *New*, *Open*, *Save*, *Save As*, *Reset Log File* and *Exit*. Further description for these MenuItem are covered in the section below.

- *New* - *New* function closes all topology on the workspace and lets user to create a new topology on the workspace.
- *Open* - *Open* function opens the saved topology to the workspace for further simulation.
- *Save* - *Save* function saves the changes for the current topology on the workspace to file.
- *Save As* - *Save As* function saves the new topology created on workspace to a new file.

• *Reset Log File* - *Reset Log File* function clear the data that has been logged to a log file.

• *Exit* - *Exit* function closes the topology on workspace and exits the Java Network Simulator program.

5.1.1.2 Edit

The MenuItem that are grouped into this *Edit menu* includes *Select All* and *GUI High Priority*. Further description for these MenuItem are shown below.

• *Select All* - *Select All* function selects all components (like switch, link, B-TE and application) on the workspace.

• GUI High Priority - *GUI High Priority* function gives the Graphical User Interface (GUI) higher priority than other events. This function enables any movement on the components or properties dialog box during simulation and don't give any jitter effect to the Network Simulator screen.

5.1.1.3 Window

The MenuItem that is grouped into this *Window menu* includes *Close All*. Further description for this MenuItem is covered in the section below.

• *Close All* - *Close All* function is used to close all properties windows opened on the workspace.

5.1.1.4 Help

The MenuItem that is grouped into this *Help menu* includes *About*. Detail description for this MenuItem is shown below.

•*About* - *About* function gives the version of the ATM Network Simulator.

5.1.2 Control Bar

A control bar consists of several buttons that is essential in the simulation process. The buttons include Start, Pause, Resume, Reset, Connect Mode, Fit All, and Test. The Digital Global Clock also included in the control bar area. The following section describes the function of each control element in the control bar.

Buttons function:

- Start – Start button starts the new simulation on the topology.
- Pause – Pause button pause the simulation.
- Resume – Resume button resumes the simulation after the simulation being paused.
- Reset – Reset button reset all parameters to its original state after simulation.
- Connect Mode – Connect Mode button enables user to connect the components like switch, link, B-TE and application becomes a topology for simulation purpose.
- Fit All – Fit All button fits the topology to the workspace so the user can use the scrollbar to view the full topology.
- Test – Test button enables user to simulate the test topology that has been created.

Digital Global Clock function:

Digital Global Clock is located at the bottom of the right side. It shows the current simulation time to user. Besides, it also synchronizes all processes that happens during the showing simulation time.

5.2 Class Design

The simulator consists of various types of objects or classes. Proper designing of these classes ensures that the simulator can really simulates the ATM network that imitates the real network. The simulator basically builds from major classes like ATM Cell class, switch class, B-TE class, link class and application class.

5.2.1 ATM Cell Class

Since the simulator is designed to simulate ATM networks, a cell data type has been defined. A cell constitutes a very important data type in the simulator because it contains the route number needed for routing by ATM switches. The structure may contain different elements to tailor the cell for different applications, but must always contain the route number. For switching or routing purposes, an ATM switch read off the route number found in the cell, then looks up its routing table to forward the cell via the next link to the next switch (or to the next B-TE if at the end of a connection).

The basic elements or parameters that include in the ATM Cell class are shown in Table 5.1.

Table 5.1: ATM cell class elements and its description

Major Elements	*Description*	*Units*
Virtual Path Identifier (VPI)	Identifies virtual path of the cell.	Integer
Virtual Channel Identifier (VCI)	Identifies virtual channel of the cell.	Integer
Payload Type (PT)	Indicates the types of data in the cell.	Integer
Cell Loss Priority (CLP)	Indicates whether the cell should be discarded if congestion encountered.	Integer

These parameters ensure that the ATM cell carries the data types important to the simulator. With the parameters specified, the cell can be sent from the source and reach at the desired destination.

5.2.2 Switch class

This class contains the general information for an ATM switch to ensure the extensibility of ATM switch to different models in future. General elements or parameters in the switch class are name, delay to process a cell, switching slot time, output queue size, high threshold, low threshold, logging every n ticks, cell received, percent cell drop, cells in xBR queue to link n, cells dropped in xBR queue to link n and congestion for link n. Additional parameters for the use of leaky bucket policing are Peak Cell Rate (PCR), Cell Delay Variation Tolerance (CDVT), Sustained Cell Rate (SCR), Maximum Burst Size (MBS), PCR drop cells in xBR queue and SCR drop cells in xBR queue. Parameter for the use of Token Bucket like Generate Token every n uSec is also included. Further description of these elements is shown in Table 5.2.

Table 5.2: Switch class elements and its description

Major Elements	*Description*	*Units*
Name	ID of the component	String
Delay to process a cell	An increment of time after the arrival of a cell at the switch before the switch places the cell on the outgoing link.	Usec
Switching Slot time	The rate at which cells are switched from an input port to an output port.	Mbit/s
Output q_size	Available buffer space for the queue; the same value is used for every queue in the switch. When a cell is ready for transmission but a slot on that link is not available, it waits in a queue at that port.	Cells
High threshold	If the number of cells in any queue exceeds this value the congestion flag is set.	Cells
Low threshold	The congestion flag is cleared when the numbers of cells in all queues fall below this value.	Cells
Peak Cell Rate (PCR)	The maximum rate at which a connection can transmit.	Mbit/s
Cell Delay Variation Tolerance (CDVT)	The difference of the maximum and minimum cell transfer delay experience during the connection.	Usec
Sustained Cell Rate (SCR)	This is the average rate as measured over a long interval.	Mbit/s
Maximum Burst Size (MBS)	The maximum number of cells that may burst at the PCR but still be compliant.	Cells
Generate Token	Time to generate a new token in Token Bucket every n	Usec

every n uSec	uSec	
Logging every n ticks	Time to log data every n ticks.	Ticks
Cells received	Total number of cells received by the switch.	Integer
Percent cell drop	Number of cells drop by the switch as a percentage of the total cells received.	%
Cells in xBR Q to link n	Cells awaiting transmission in a given priority queue. There are two types of queues for each port – a CBR/VBR queue and ABR queue. Cells in the CBR/VBR queue have top priority; a cell from the ABR queue will be sent only if the CBR/VBR queue is empty.	Cells
Cells dropped in xBR Q to link n	Cells dropped at a port when a queue exceeds its maximum size.	Cells
PCR drop cells in xBR Q	The number of cells dropped that are not compliant with Peak Cell Rate (PCR).	Cells
SCR drop cells in xBR Q	The number of cells dropped that are not compliant with Sustained Cell Rate (SCR).	Cells
Congestion for Link n	There is one congestion flag for each port. The flag is set when a queue exceeds its High Threshold value, cleared when both queues fall below the Low Threshold.	Boolean

The switch is the component that switches or routes cells over several virtual channel links. This class contains the general information for an ATM switch to ensure the extensibility of ATM switch to different models in future.

5.2.3 B-TE class

Cell received from the application sides are queued in one of the two priority queues if no link slot is available for the transmission. If either queue exceeds its size limit cells will be dropped. The major elements of the B-TE class are name, maximum output queue size, logging every n ticks, cells received, NSAP for interface to link n, cells in xBR Q to link n and cells dropped in xBR Q to link n. The detailed description of these elements is shown in Table 5.3.

Table 5.3: B-TE class elements and its description

Major Elements	*Description*	*Units*
Name	ID of the component	String
Maximum	Available buffer space for each type of queue.	Cells

output queue size		
Logging every n ticks	Time to log data every n ticks.	Ticks
Cells Received	Total number of cells received by the B-TE.	Cells
NSAP for interface to link n	The ATM address for this B-TE.	Integer
Cells in xBR Q to link n	Cells awaiting transmission in a given priority queue. There are two types of queues – a CBR/VBR queue and ABR queue. Cells in the CBR/VBR queue have top priority; a cell from the ABR queue will be sent only if the CBR/VBR queue is empty.	Cells
Cells dropped in xBR Q to link n	Cells dropped at the network port when a queue exceeds its maximum size.	Cells

5.2.4 Link class

The major elements of the link class are name of the link, link speed and distance. Further description of these elements is shown in Table 5.4.

Table 5.4: Link class elements and its description

Major Elements	*Description*	*Units*
Name	ID of the component	String
Link speed	Specified the speed of the link between end system and switch.	Mbits/sec
Distance	Distance between switch and B-TE.	Km

5.2.5 Application class

The ATM application consists of Constant Bit Rate (CBR) application class and Variable Bit Rate (VBR) application class in this ATM simulator.

5.2.5.1 CBR class

In CBR application, cells are generated at a constant rate for the duration of the simulation. The major elements of CBR class are name, bit rate, start time, number of Mbits to be sent, repeat count, port number, destination NSAP, destination port number, calls attempted, calls accepted, incoming calls and total incoming calls.

Further description of these elements is shown in Table 5.5.

Table 5.5: CBR class elements and its description

Major Elements	*Description*	*Units*
Name	ID of the component	String
Bit rate	The rate at which this application will send cells.	Mbit/s
Start time	The time of this application to start sending cells.	Usec
Number of Mbits to be sent	The number of Mbits this application will send during current connection.	Mbits
Repeat Count	The number of times this application will send the same amount of data (in Mbits) to destination.	Integer
Port Number	This application's own port number.	Integer
Destination NSAP	NSAP ATM address for destination.	Integer
Destination port number	Port number for destination.	Integer
Calls attempted	Number of calls attempted to send data to destination during current connection.	Integer
Calls accepted	Number of calls accepted by the destination during current connection.	Integer
Incoming Calls	Number of incoming calls from source during current connection.	Integer
Total Incoming Calls	Total incoming calls accepted at the end of current connection.	Integer

5.2.5.2 VBR class

In VBR application, traffic is generated as an ON-OFF source. Cells are generated at the specified bit rate during a burst. Mean burst length and mean interval between bursts are user specified. The major elements of VBR class are name, bit rate, mean burst length, mean interval between burst, start time, number of Mbits to be sent, repeat count, port number, destination NSAP, destination port number, calls attempted, calls accepted, incoming calls and total incoming calls

Further description of these elements is shown in Table 5.6.

Table 5.6: VBR class elements and its description

Major Elements	*Description*	*Units*
Name	ID of the component	String
Bit rate	The rate at which this application will send cells.	Mbit/s
Mean Burst Length	The average duration of burst.	Usec

Mean Interval Between Burst	The average time between 2 burst.	Usec
Start time	The time of this application to start sending cells.	Usec
Number of Mbits to be sent	The number of Mbits this application will send during current connection.	Mbits
Port Number	This application's own port number.	Integer
Destination NSAP	NSAP ATM address for destination.	Integer
Destination port number	Port number for destination.	Integer
Calls attempted	Number of calls attempted to send data to destination during current connection.	Integer
Calls accepted	Number of calls accepted by the destination during current connection.	Integer
Incoming Calls	Number of incoming calls from source during current connection.	Integer
Total Incoming Calls	Total incoming calls accepted at the end of current connection.	Integer

5.3 Leaky Bucket Design

The Leaky Bucket is designed in a flexible way. In every input port, there will be two types of Leaky Bucket: Single Leaky Bucket and Double Leaky Bucket. The incoming cells from CBR traffic will be directed into Single Leaky Bucket for policing. This leaky bucket polices the PCR and CDVT traffic parameters. Incoming cells from VBR traffic will be directed into Double Leaky Bucket. The cells will go to first Double Leaky Bucket for PCR and CDVT traffic parameters policing. Then the cells will enter the second Double Leaky for SCR and BT policing. Cells that are conforming to these buckets will be admitted to network.

The designed Single Leaky Bucket and Double Leaky Bucket is shown in Figure 5.1.

5.4 Token Bucket Design

A Token Generator, Token Bucket and data buffer is required in order to implement Token Bucket technique in the switch. Token generator is designed to generate new token in the time interval that has been specified. Token Bucket is used to keep the generated

Token. New generated token will be discarded if the token bucket is full. The data buffer's function is to queue the incoming cells in a buffer before getting a token and admit to network. These components will be placed at the input port. The designed Token Bucket and its components is shown in Figure 5.2.

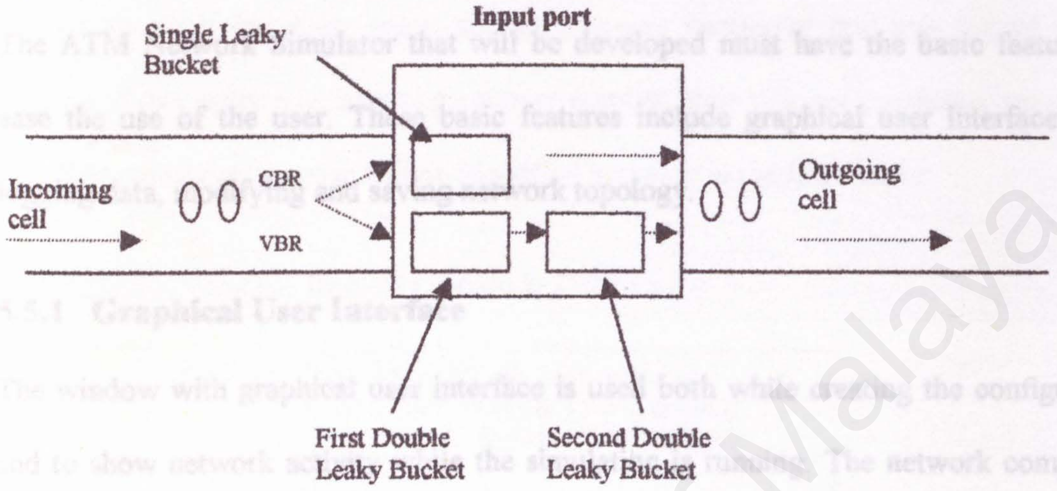


Figure 5.1: Single Leaky Bucket and Double Leaky Bucket design

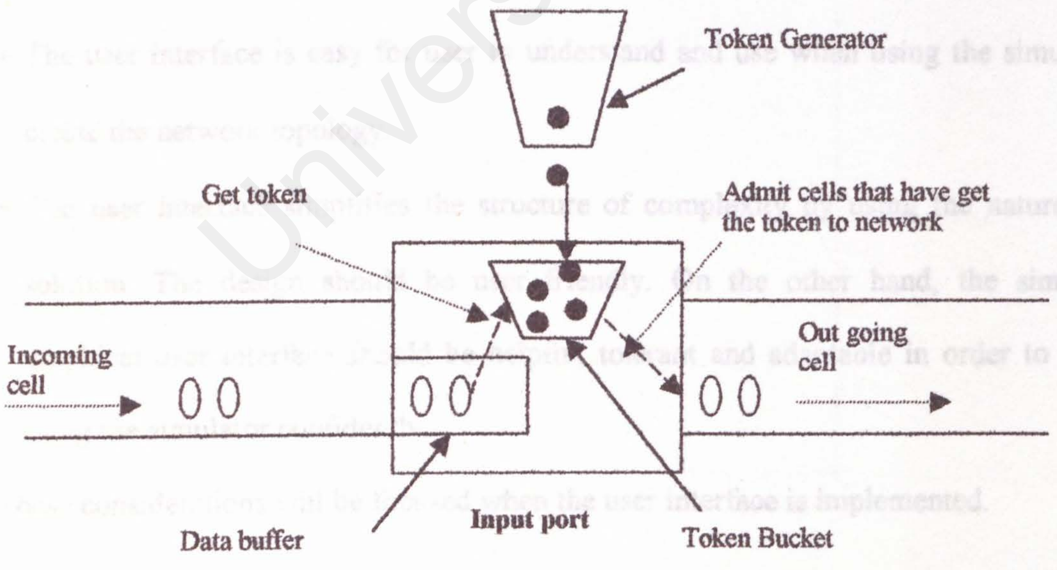


Figure 5.2:Token Bucket and components design

Each time a new connection is connected to the switch, a new token generator is automatically created for that connection. The token generate time is designed in a flexible way. It enables user to specify the token generate time value. A buffer is designed for each connection. Incoming cells from each connection to the switch will be kept in the individual data buffer before entering the network.

5.5 Features of the ATM Network Simulator

The ATM Network Simulator that will be developed must have the basic features that ease the use of the user. These basic features include graphical user interface (GUI), logging data, modifying and saving network topology.

5.5.1 Graphical User Interface

The window with graphical user interface is used both while creating the configurations and to show network activity while the simulation is running. The network components should be able to be created in the workspace window.

A few considerations have been put forward as an idea of how user interface will be designed:

- The user interface is easy for user to understand and use when using the simulator to create the network topology.
- The user interface simplifies the structure of complexity by using the nature of the solution. The design should be user friendly. On the other hand, the simulator's graphical user interface should be helpful, tolerant and adaptable in order to let user using the simulator confidently.

These considerations will be focused when the user interface is implemented.

5.5.2 Logging Data

The simulator should provide analysis for the simulation result. Output parameter values may also be tagged for logging to a file. Logging data while running the simulator provides a means of simulation analysis. With this logged file, user can carry out further analysis on the behavior of the designed network topology.

When logging for a particular parameter is checked, every new value of that parameter with a corresponding time stamp is saved in a file. The file created by this process will contain an entry for every value change of every parameter that was tagged for data logging. Every entry will consist parameter number, time tick and parameter value at that tick. The parameter number will be identified by name in the file header.

The example of log file format is shown in Figure 5.3.

```
ID 2 "sw1" "Cells Received"
ID 3 "sw1" "Cells in VBR Q to link2"
ID 1 "bte1" "Cells in VBR Q to link1"
323 2 1
9126 2 2
9605 1 1
9928 2 3
10101 3 1
11018 1 1
11341 2 4
11466 3 1
12431 1 1
12754 2 5
12831 3 2
13844 1 1
14167 2 6
14196 3 3
14341 3 2
15257 1 1
15580 2 7
15834 3 3
16670 1 1
16993 2 8
17199 3 4
18083 1 1
18406 2 9
--- --- ---
```

Figure 5.3: Example of log file format

The lines at the head of the file starting with 'ID' sign are a listing of all parameters that were marked for data logging when the simulator was running. The number immediately following the "ID" sign is the ID number that will be used in the remainder of the file to identify the parameter. The rest of the line gives the component name and parameter name respectively.

All lines following after the ones marked with "ID" sign are the actual data recorded during the simulation. The first column is the time (in ticks), the second column is the parameter ID and the third column is the value of the parameter at that time.

5.5.3 Making Modifications

A component should be able to be selected and modify it. The component can be delete, edit or move in a network topology. The component that has been modified should be updated into the list and the interface should show the latest topology in updated view.

5.5.4 Saving a Network configuration

The network configuration that has been created with the ATM Network Simulator is saved in files for future usage. The records of the network components and the topology will be saved into a file with .sim extension as an input file. File saving in this network simulator considers the factors like maintainability and platform independent.

- **Maintainability** – The saved file should be made suitable for maintaining and constructing a simple local database for the simulation records. It allows easy manipulating and storing of variable length of each component.

- **Support for many Platforms** – The saving file system should support for many platforms, including Windows and UNIX. Thus, the saved topology file should be able to be used in either one platform. This is an important consideration as the network topology and simulation result can be platform independent.

5.6 Chapter Summary

This chapter has covered the system design for the ATM network simulator. Detailed description about the user interface design and classes design are done. The user interface design includes the features that can be seen by user while the classes design includes the classes attributes and function. The Leaky Bucket and Token Bucket design were explained in detail. Besides, the features of the simulator and the design considerations for the development process were described.

Chapter6: SYSTEM IMPLEMENTATION

6.1 System Implementation

In this phase, the plans in design stage are transformed into reality. The designed simulator is converted into coding. The implementation approach used in the ATM Network Simulator is the object modeling. Object oriented methodology is used at the implementation phase of the development process through the Java programming language.

6.2 Leaky Bucket Implementation

Leaky bucket policing function will be implemented in the GenericATMSwitch class. It will be built on top of javasim package. There are two types of leaky bucket that will police the traffic at the switch: Single Leaky Bucket and Double Leaky Bucket. Single Leaky Bucket will police traffic from source CBR application while Double Leaky Bucket will police traffic from source VBR application.

6.2.1 Single Leaky Bucket Implementation

When a new cell is passed from the link to switch, the switch will check the cell's connection type with the statement:

```
        connection_type=rec.contype;
        if(connection_type==Cell.CON_CBR)
```

If the connection type is CBR application, the switch will pass the cell to Single Leaky Bucket for policing purpose. The Single Leaky Bucket function is shown below.

```
private void SingleBucket(Port voport,Cell cell_from_SingleLB,CallRecord
                          rec_from_SingleLB)
      //this function receive CBR cells from input port and check
      //the cells compliance.
      //parameters being passed into this function is cells record
      //and output port value.
```

This leaky bucket will check the cell's conformance. If the cell is compliant to the Peak Cell Rate (PCR) and Cell Delay Variation Tolerance (CDVT) value, the switch will pass the cell to the output port and send it to the destination. Otherwise, the switch will drop the cell or admit the cell to network depending on the current traffic situation in the switch.

If the current queue size in the output port is less than 70% of High Threshold value, the switch will pass the non-conforming cell to network. Otherwise the switch will drop the non-conforming cell that has violated the traffic parameters. The implementation of Single Leaky Bucket policing is shown in Figure 6.1.

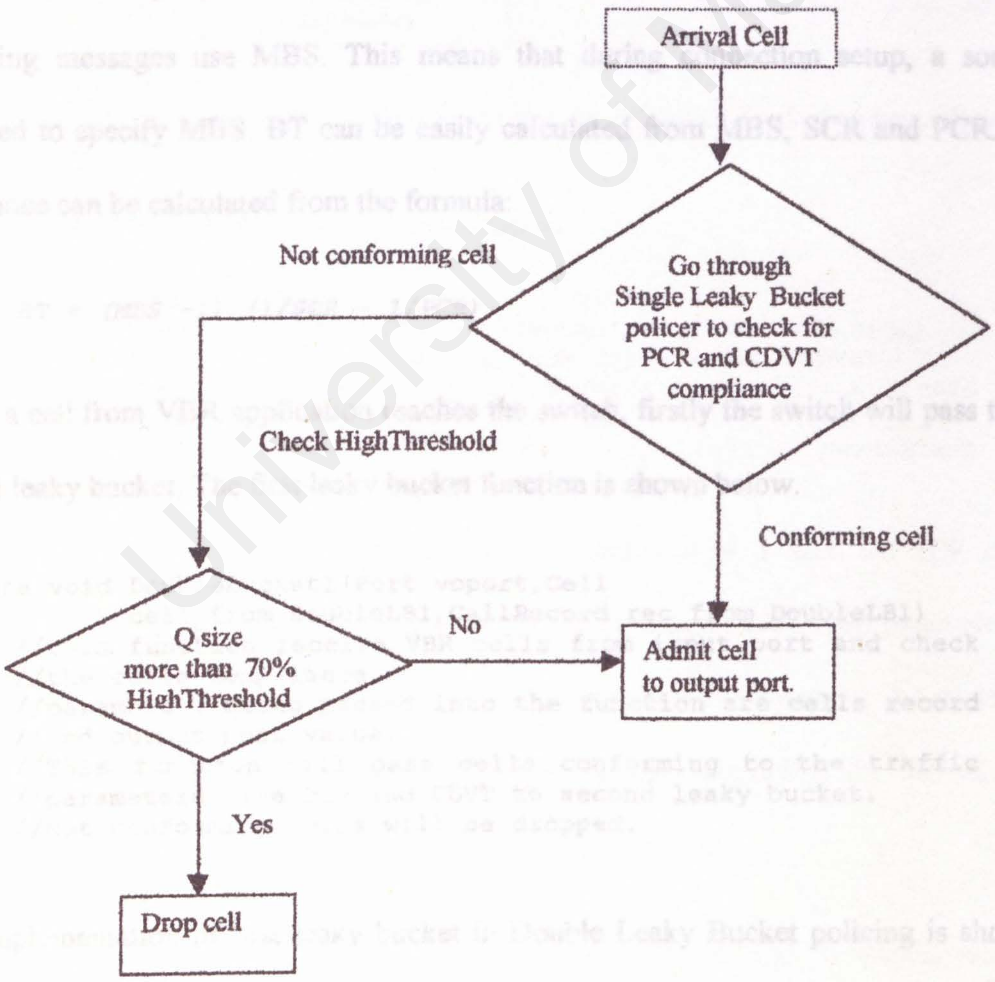

Arrival Cell
Go through Single Leaky Bucket policer to check for PCR and CDVT compliance
Not conforming cell
Check HighThreshold
Conforming cell
Q size more than 70% HighThreshold
No
Admit cell to output port.
Yes
Drop cell

Figure 6.1: Implementation of Single Leaky Bucket policing

6.2.2 Double Leaky Bucket Implementation

Double Leaky Bucket will police traffic from source VBR application. In order to check the cell's connection type, the following statement is used:

```
        if(connection_type==Cell.CON_VBR)
```

This statement will ensure that the cells from VBR application will be passed to Double Leaky Bucket for traffic policing purpose.

There are two leaky buckets in the Double Leaky Bucket policing. First leaky bucket will police the traffic parameters like Peak Cell Rate (PCR) and Cell Delay Variation Tolerance (CDVT) while the second one will police the Sustained Cell Rate (SCR) and Burst Tolerance (BT). Since Maximum Burst Size (MBS) is more intuitive than BT, signaling messages use MBS. This means that during connection setup, a source is required to specify MBS. BT can be easily calculated from MBS, SCR and PCR. Burst Tolerance can be calculated from the formula:

$$BT = (MBS - 1)\ (1/SCR - 1/PCR)$$

When a cell from VBR application reaches the switch, firstly the switch will pass the cell to first leaky bucket. The first leaky bucket function is shown below.

```
private void DoubleBucket1(Port voport,Cell
             cell_from_DoubleLB1,CallRecord rec_from_DoubleLB1)
      //this function receive VBR cells from input port and check
      //the cells compliance.
      //parameters being passed into the function are cells record
      //and output port value.
      //This function will pass cells conforming to the traffic
      //parameters like PCR and CDVT to second leaky bucket.
      //Not conforming cells will be dropped.
```

The implementation of first leaky bucket in Double Leaky Bucket policing is shown in Figure 6.2.

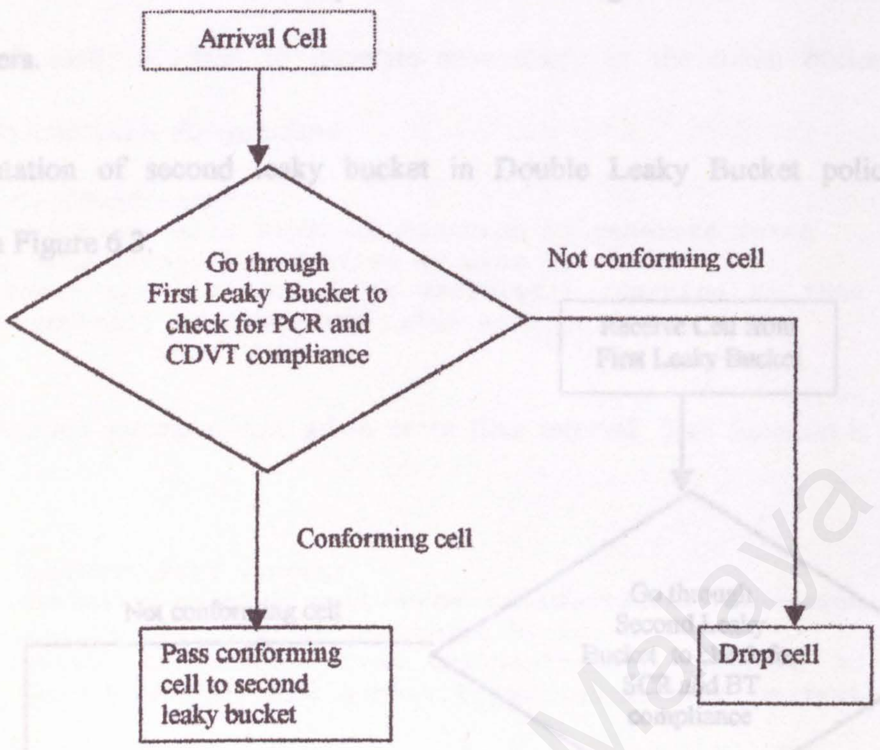

Arrival Cell
Go through First Leaky Bucket to check for PCR and CDVT compliance
Not conforming cell
Conforming cell
Pass conforming cell to second leaky bucket
Drop cell

Figure 6.2: Implementation of first leaky bucket in Double Leaky Bucket policing

The second leaky bucket function is shown below.

```
private void DoubleBucket2(Port voport,Cell
          cell_from_DoubleLB2,CallRecord rec_from_DoubleLB2)
     //this function receives cells from first leaky bucket.
     //parameters being passed into the function are cells record
     //and output port value.
     //it will admit cells conforming to traffic parameters
     //like SCR and BT to network.
```

The successful cell that passed from first leaky bucket will be policed for SCR and BT traffic parameters in second leaky bucket. If the cell is compliant to these parameters, the switch will admit the cell to output port and then to network. Otherwise, the switch will drop the cell or tag the cell and then admit it to network depending the current traffic situation in the switch.

If the current queue size in the output port is less than 50% of High Threshold value, the switch will tag the cell and pass the non-conforming cell in second leaky bucket to network. Otherwise the switch will drop the non-conforming cell that has violated the traffic parameters.

The implementation of second leaky bucket in Double Leaky Bucket policing is summarizing in Figure 6.3.

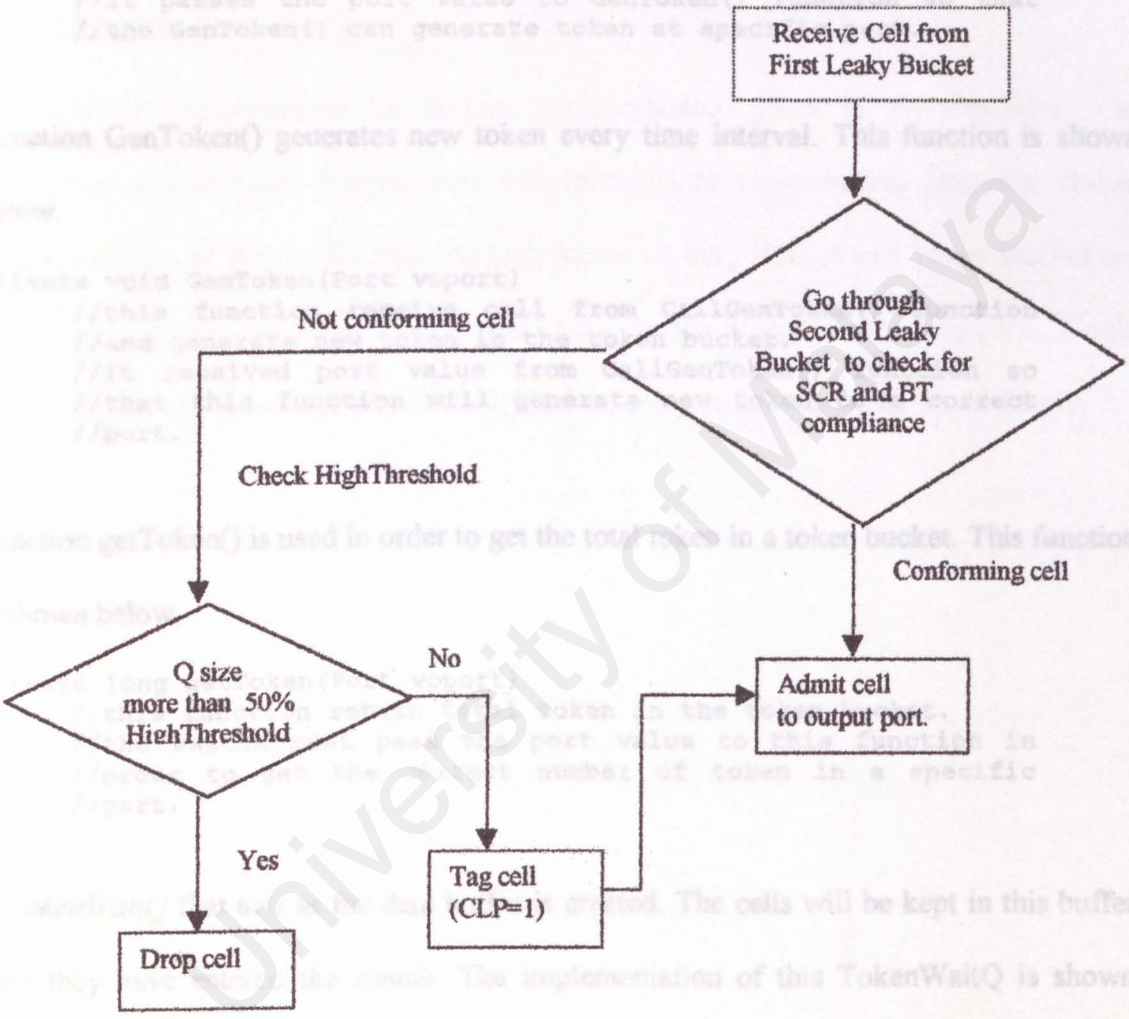

Receive Cell from First Leaky Bucket
Go through Second Leaky Bucket to check for SCR and BT compliance
Not conforming cell
Check HighThreshold
Conforming cell
Q size more than 50% HighThreshold
No
Yes
Admit cell to output port.
Tag cell (CLP=1)
Drop cell

Figure 6.3: Implementation of second leaky bucket in Double Leaky Bucket policing

6.3 Token Bucket Implementation

Token Bucket is implemented in the GenericATMSwitch class, which is on top of the javasim package. There are three important functions in the Token Bucket generator. These functions are CallGenToken(), GenToken() and getToken().

The switch will invoke CallGenToken() every time interval specified by user. Its function is to call GenToken() function to generate new token in the token bucket. The CallGenToken() function is shown below.

```
private void CallGenToken()
      //this function calls GenToken function to generate Token
      //every time interval specified by user (in Usce).
      //it passes the port value to GenToken() function so that
      //the GenToken() can generate token at specific port.
```

Function GenToken() generates new token every time interval. This function is shown below.

```
private void GenToken(Port voport)
      //this function receive call from CallGenToken() function
      //and generate new token in the token bucket.
      //it received port value from CallGenToken() function so
      //that this function will generate new token to a correct
      //port.
```

Function getToken() is used in order to get the total token in a token bucket. This function is shown below.

```
private long getToken(Port voport)
      //this function return total token in the token bucket.
      //the switch must pass the port value to this function in
      //order to get the correct number of token in a specific
      //port.
```

A *TokenWaitQ* that acts as the data buffer is created. The cells will be kept in this buffer once they have entered the switch. The implementation of this TokenWaitQ is shown below.

```
      voport.TokenWaitQ.add(cell);
```

When a new cell is received by switch, the switch will passed the cell to the TokenBucket() function. The TokenBucket() function is shown below.

```
private void TokenBucket(Port voport)
      //this function will keep the cells in the buffer. If there
      //is available token, the token bucket will admit the cells
      //to network.
```

The Token Bucket will check whether there is available token. If the token is available, the data cell will grab the token and admit to network. Then the total token in Token Bucket will decrease by one. Otherwise, the cell will wait at the *TokenWaitQ* queue until a new token is available.

6.4 Chapter Summary

This chapter has discussed the system implementation phase in the simulator. The conceptual and abstract designs were transformed into programming language codes. Implementation of two traffic policing techniques – Leaky Bucket and Token Bucket are described in detail. The description on the Leaky Bucket is separated to two parts: Single Leaky Bucket and Double Leaky Bucket. Each policing technique is explained deeply.

Chapter7: SYSTEM TESTING

7.1 Unit Testing

Unit testing section is separated into two parts. The first part will carry out the Leaky Bucket unit testing while the second part will carry out the Token Bucket unit testing.

7.1.1 Leaky Bucket Unit Testing

For Single Leaky Bucket and First Leaky Bucket in Double Leaky Bucket, unit testing is done on the parameters used in the bucket to police the traffic. These parameters are cell arrival time (tka), Last compliance time (LCT), value of the leaky bucket counter (X), Auxiliary variable (X'), Increment (I) and Limit (L). I value takes consideration of Peak Cell Rate (PCR) value while the L value takes consideration of Cell Delay Variation Tolerance (CDVT) value. The I and L value is represented with the following equations:

I= (double) (424.0 / sw_pcr.getValue());
L= (double) sw_cdvt.getValue();

In order to test the I and L value, PCR and CDVT value is entered into the switch. Then these values are printed out to ensure that the calculated I and L value is correct. The entered PCR and CDVT value is:

PCR = 50
CDVT = 15

The output is displayed with the following statements:

```
System.out.println("I value : " + rec.I);
System.out.println("L value : "+ rec.L);
```

Output:

```
I value  : 8.48
L value : 15.0
```

To test other parameters, a quantity of cells is admitted to the switch. The tka values is updated each time a new cell coming in to the leaky bucket. First, the X' value is calculated and compare with the L value. When X` value is less than L value, the LCT value is updated with the tka value and the X value is updated with the addition value between X` and I. When X` value is more the L value, no update is performed. From this test, it proved that the Single Leaky Bucket and the First Leaky Bucket are working properly.

For Second Leaky Bucket in Double Leaky Bucket, unit testing is done on the new parameters like I2 and L2. I2 value takes consideration of Sustained Cell Rate (SCR) value while L2 value takes consideration of Burst Tolerance (BT) value. The I2 and L2 value is represented with the following equations:

I2= (double) (424.0 / sw_scr.getValue());

*L2= (double) (424.0 * (sw_mbs.getValue()-1) * (1/sw_scr.getValue()-1/sw_pcr.getValue()) + sw_cdvt.getValue());*

The entered PCR, CDVT, SCR and MBS value in the switch is:

PCR = 50
CDVT = 15
SCR = 30
MBS = 10

The I2 and L2 value is printed out to ensure that the calculated value is correct. The output is displayed with the following statements:

```
System.out.println("I2 value : " + rec.I2);
System.out.println("L2 value : "+ rec.L2);
```

Output:

```
I2 value  : 14.13333333
L2 value : 65.88
```

A quantity of cells are admitted to the switch to test other parameters in the bucket. Each time a cell is passed in from the First Double Leaky Bucket, the tka value in the bucket is updated to the latest value. When the calculated X` value is less than L2 value, update process is done to X and LCT value. No update is performed when X` value is more than L2 value. Therefore, this proved that the Second Leaky Bucket is successfully running.

7.1.2 Token Bucket Unit Testing

In the Token Bucket unit testing, 3 important components in the Token Bucket are tested. These components are Token Generator, Token Bucket and Data Buffer.

To test the Token Generator component, the time to generate token in the switch is specified to 10 Usec. As expected, the token is generated in the specified time interval. This has proved that the Token Generator is working.

To test the Token Bucket component, the Token Bucket maximum size is specified to 50. When the generated token is put into the bucket, the bucket counter is increased. No further increment is done when the bucket counter reached the 50 value. This proved that the Token Bucket could keep the generated token until a maximum value. When a quantity of cells are admitted to the switch, the bucket counter is decreased. It means that the cells successfully get the token from the Token Bucket. Therefore, the Token Bucket is working correctly.

To test the Data Buffer component, the token generate time is adjusted to a large value. The purpose is to make sure that the cells will keep in the buffer as there is no token available. Then a quantity of cells is admitted to the switch. As expected, the counter in the Data Buffer is increased. This testing proved that the Data Buffer successfully keep the incoming cells. When there is token available, the Data Buffer counter is decreased by one. The result of the testing showed that the Data Buffer is successfully implemented.

7.2 System Testing

System Testing is separated into two parts, Leaky Bucket Testing and Token Bucket Testing. The Leaky Bucket testing is further divided into Single Leaky Bucket and Double Leaky Bucket testing.

A complete topology is created for the system testing. The topology consists of 3 switches (sw1 – sw3), 9 links (link1 – link9), 6 B-TE (bte1 – bte6), 6 CBR Application (cbr1 – cbr6) and 6 VBR Application (vbr1 – vbr6). The created topology is shown in Figure 7.1.

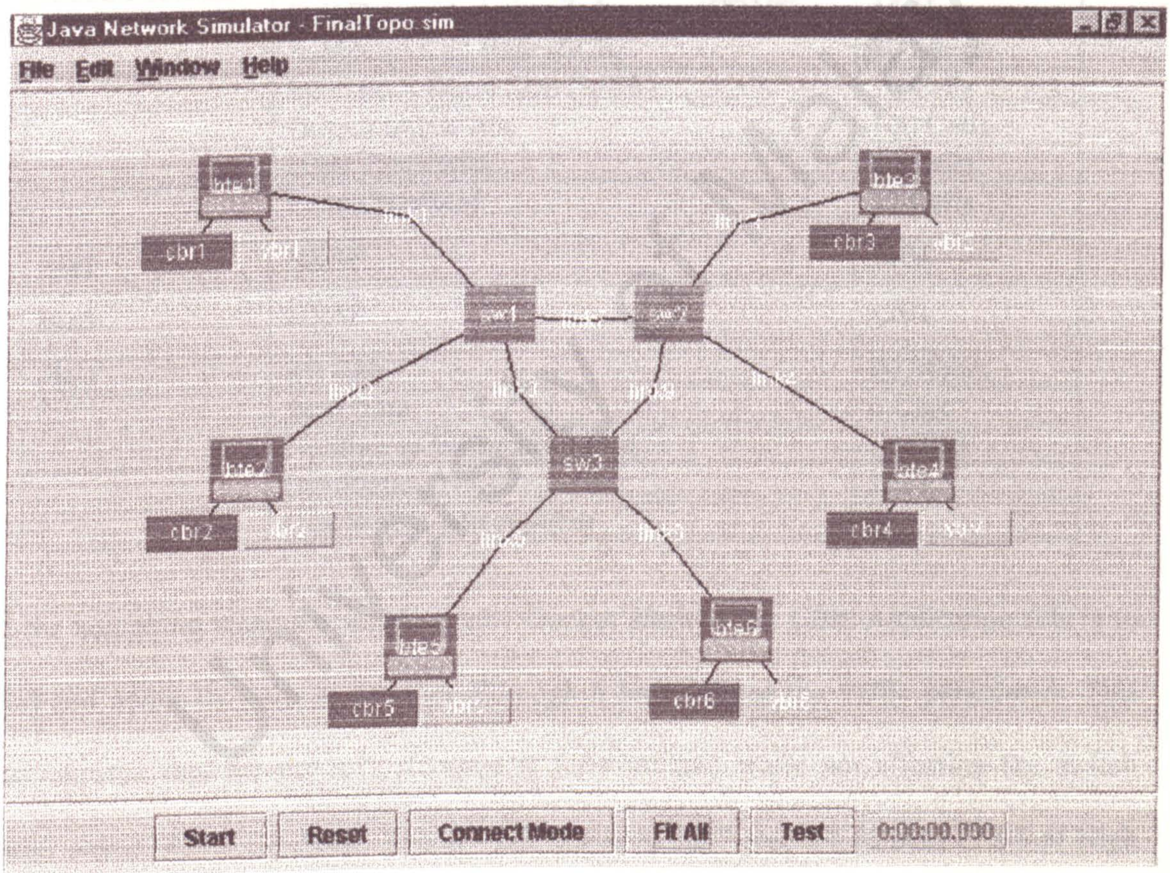


Figure 7.1: Complete Topology to test Leaky Bucket and Token Bucket

7.2.1 Single Leaky Bucket System Testing

In order to test the functionality of the Single Leaky Bucket, sw1 and sw2 switch in Figure 7.1 are used. Application cbr1 from bte1 is used to send cells to application cbr2 in bte2. The parameters that entered in the corresponding switches, B-TEs, and CBR Application is shown in Table 7.1.

Table 7.1: Parameters entered for network components for Single Leaky Bucket testing

Components	Parameters	Value
sw1	Peak Cell Rate (PCR)	60 Mbit/s
	Cell Delay Variation Tolerance (CDVT)	15 Usec
	Sustained Cell Rate (SCR)	40 Mbit/s
	Maximum Burst Size (MBS)	10 Cells
	Output queue size	100 Cells
	High Threshold	80 Cells
	Low Threshold	20 Cells
sw2	Peak Cell Rate (PCR)	50 Mbit/s
	Cell Delay Variation Tolerance (CDVT)	15 Usec
	Sustained Cell Rate (SCR)	40 Mbit/s
	Maximum Burst Size (MBS)	10 Cells
	Output queue size	100 Cells
	High Threshold	80 Cells
	Low Threshold	20 Cells
bte1	NSAP	1101
bte3	NSAP	1201
cbr1	Bit Rate	70 Mbit/s
	Start time	0 Usec
	Number of Mbits to be sent	1 MBits
	Destination NSAP	1201
	Destination port number	1

At the beginning stage the Single Leaky Bucket allowed the CBR Application (cbr1) to send and admit the cells to network although it has violated the traffic parameters. This way ensured that the network resource is fully utilized while not affecting the switch performance. When the queue size in VBR queue to link7 reached 56 cells (70% of High Threshold value), the Single Leaky Bucket in switch sw1 started dropping cells that are not compliant.

The same case is done to switch sw2. It started dropping cells when the queue size in VBR queue to link3 reached 56 cells. By this precaution, the Single Leaky Bucket has performed its role to prevent the violating CBR traffic to cause congestion in the switch.

This testing has proved that the Single Leaky Bucket is functioning properly in policing the traffic from CBR Application.

7.2.2 Double Leaky Bucket System Testing

The testing has to consider the functionality of each leaky bucket. The leaky buckets should discard cells that violated the negotiated parameters during the connection setup.

In order to test the functionality of both leaky buckets, the same topology that shown in Figure 7.1 is used. The components involved are switch sw2 and sw3, link4, link6 and link9, bte4 and bte6, application vbr4 and vbr6. In this testing, application vbr4 from bte4 is used to send cells to application vbr6 in bte6. The parameters entered to the involving network components are shown in Table 7.2.

Table 7.2: Parameters entered for network components for Double Bucket testing

Components	**Parameters**	**Value**
sw2	Peak Cell Rate (PCR)	50 Mbit/s
	Cell Delay Variation Tolerance (CDVT)	15 Usec
	Sustained Cell Rate (SCR)	40 Mbit/s
	Maximum Burst Size (MBS)	10 Cells
	Output queue size	100 Cells
	High Threshold	80 Cells
	Low Threshold	20 Cells
sw3	Peak Cell Rate (PCR)	50 Mbit/s
	Cell Delay Variation Tolerance (CDVT)	10 Usec
	Sustained Cell Rate (SCR)	35 Mbit/s
	Maximum Burst Size (MBS)	10 Cells
	Output queue size	100 Cells
	High Threshold	80 Cells
	Low Threshold	20 Cells
bte4	NSAP	1202
bte6	NSAP	1302
vbr4	Bit Rate	45 Mbit/s
	Mean Burst Length	15 Usec
	Mean Interval Between Bursts	20 Usec

	Start time	0 Usec
	Number of Mbits to be sent	1 MBits
	Destination NSAP	1302
	Destination port number	2

The VBR source may transmit at maximum average cell rate of SCR. When the source is exceeding the SCR it is allowed to burst at maximum MBS cells at a rate of PCR. The variation in the PCR may not exceed CDVT. The Double Leaky Bucket in switch sw2 and sw3 is predicted to drop cells as the sending VBR application (vbr4) has violated the SCR parameters. It sent data at rate 45Mbit/s, which is more than the SCR value in both switches.

When simulation started, the Double Leaky Bucket in both switches let cells to enter the switch. Once the number of cells in VBR queue to link9 in switch sw2 reached 40 cells (50% of High Threshold value), the Double Leaky Bucket started dropping cells that violated the SCR traffic parameter. This is shown by the value in *SCR dropped cells in VBR Q to link9* in sw2 properties popup menu.

The successful cells that passed from switch sw2 into the switch sw3 are policed again. When the VBR queue to link6 in switch sw3 reached 40 cells, the Double Leaky Bucket started dropped the non-conforming VBR cells. This test has proved that the Double Leaky Bucket is successfully implemented to police the VBR traffic.

7.2.3 Token Bucket System Testing

In order to test the functionality of Token Bucket, the same topology in Figure 7.1 is used. The network components involved are switch sw1 and sw3, link2, link5 and link8, bte2 and bte5, cbr2 and cbr5, vbr2 and vbr5. The parameters entered to these network components are summarized in Table 7.3.

Table 7.3: Parameters entered for network components for Token Bucket testing.

Components	Parameters	Value
sw2	Output queue size	100 Cells
	High Threshold	80 Cells
	Low Threshold	20 Cells
	Generate Token every (Usec)	1 – 8 Usec
sw3	Output queue size	100 Cells
	High Threshold	80 Cells
	Low Threshold	20 Cells
	Generate Token every (Usec)	1 – 8 Usec
bte2	NSAP	1102
bte5	NSAP	1301
cbr1	Bit Rate	100 Mbit/s
	Start time	0 Usec
	Number of Mbits to be sent	10 MBits
	Destination NSAP	1301
	Destination port number	1
vbr2	Bit Rate	50 Mbit/s
	Mean Burst Length	15 Usec
	Mean Interval Between Bursts	20 Usec
	Start time	0 Usec
	Number of Mbits to be sent	10 MBits
	Destination NSAP	1301
	Destination port number	2

The Generate Token time is adjusted to get different simulation results. This is to get the best value in controlling the cells entering the network. From the tests, the relation between Generate Token time and switch sw1 and sw3 status is shown in Table 7.4.

Table 7.4: Relation between Generate Token time and switch sw1 and sw3 status

Generate Token time	sw1 status	sw3 status
1 Usec	Congestion in link8	Congestion in link5
2 Usec	Congestion in link8	Congestion in link5
3 Usec	Congestion in link8	Congestion in link5
4 Usec	Congestion in link8	Congestion in link5
5 Usec	Congestion in link8	Congestion in link5

6 Usec	Congestion in link8	Congestion in link5
7 Usec	Queue size in link8 is kept below High Threshold	Congestion in link5
8 Usec	Queue size in link8 is kept below High Threshold	Queue size in link5 is kept below High Threshold

From the testing, it found that generate token every 8Usec is an ideal value. The token bucket allowed burst traffic entering the network but bounded it. Although cbr2 sent cells at rate 100Mbit/s and vbr2 sent at rate 50Mbit/s, the token bucket has prevented the congestion occurred in the switch sw1 and sw3 causing by the burst traffic. The queue size is always kept below the high threshold value. This testing has showed that the Token Bucket policing is implemented successfully. The successfulness of the testing proved that this simulation model could simulate in a proper manner.

7.3 Chapter Summary

This chapter has discussed the testing process in detail. The components are tested for their functionality and behavior during their operation. Testing are done for Single Leaky Bucket, Double Leaky Bucket and finally the Token Bucket. The test has proved that the simulator is successfully executed with the designed model.

Chapter8: CONCLUSION

8.1 Project Finding

During the implementation of this project, different kinds of experiences are gained. The most important thing is the understanding of leaky bucket and token bucket traffic policing method for ATM network. Further more, the experience gained in problems solving during the design, implementation and testing of the simulator is really invaluable.

Choosing a suitable language in developing the ATM network simulator is important. Java Programming Language, which is object-oriented programming language really fulfils and satisfies all the conditions needed to construct the simulator. It has features such as reusability, maintainability, extensibility and modifiability. It also enables the developed simulator support multithreading and the ability of the threads to run simultaneously.

8.2 Objectives Achieved

The objectives for this project have been achieved and they are discussed in detail.

A thorough survey has made on most of the current network simulators in order to gain a better insight into the working of a network simulator. Special attention is given to the methods currently being used to overcome congestion in an ATM network environment. Therefore, through and extensive study the project could integrate the best congestion control mechanism into the network simulator. The leaky bucket and token bucket traffic policing techniques are considered to be ideal techniques in controlling the traffic in ATM network.

Based on the objectives as laid out in this project, the developed ATM Network Simulator must be object-oriented and able to simulate the ATM network. With creativity and innovations, the various components in Leaky Bucket and Token Bucket are successfully written and tested to produce the expected results.

8.3 Simulator Strength

The developed ATM Network Simulator has its strengths. These major strengths are listed below:

- **Platform independent**. The developed simulator using Java Programming Language enables it to be cross-platform. Thus, the simulator works well in Windows, Unix or Linux environment.
- **Simple and User Friendly Interface**. The designed graphical user interface facilitates user in creating a network topology on the workspace. The menu items and components in the Control Bar provide the essential functions for user to create, save, open or modify the network topology easily. Creation of a network component is just a click on the workspace interface. User can enter or modify parameters in each component in the pop up properties box.
- **Object Oriented**. The simulator is fully developed in object-oriented environment. All functions and modules are built in class. Thus, creation or modification of components can be done easily.
- **Analysis of Simulation result.** The system provides the log file and meter function. The log file function enable user to analyse the performance of the developed network topology after the simulation while the meter function plots the simulation result to graph during simulation running. These functions enable user to performed further analysis.
- **Flexible traffic policing selection.** The developed simulator enable user to choose either using traffic policing (Leaky Bucket or Token Bucket) or without traffic

policing in the switch to simulate the network topology. Thus, user can compare the performance between the each policing technique.

8.4 Simulator Limitation

The simulator is implemented with a limited set of network component type. These component types are not a complete set of existing component according to NIST Network Simulator standard. The simulation run time is slow. It also uses a lot of resource memory. Besides, Java Virtual Machine is needed in order to run the simulator.

8.5 Future Enhancement

The traffic policing techniques implemented in this simulator are Leaky Bucket, which include Single Leaky Bucket and Double Leaky Bucket and Token Bucket. Leaky Bucket and Token Bucket is implemented separately to police the traffic. In future, combination of Token Bucket and Leaky Bucket policing technique can be implemented easily.

8.6 Summary

This project managed to achieve the overall project objectives and requirements as a network simulator system as determined during the system analysis. The simulator testing phase has proved that the project is implemented successfully. This project has take quite a long time to bring it a success. The experiences gained are memorable and meaningful.

REFERENCES

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

[2] W. Stallings. 1998. *High-Speed Networks TCP/IP and ATM Design Principles.* Prentice-Hall Int., Inc. New Jersey.

[3] ATM Congestion Control. Http://www.cis.ohio-state.edu/~jain/cis788-95/atm_cong/index.html Last Modified: Augest 21st, 1995. Fang Lu.

[4] Raj Jain. 1996. *Congestion Control and Traffic Management in ATM Networks: Recent Advances and Survey.* Computer Network and ISDN System. Vol. 28. 1996. pp. 1723-1738.

[5] Nada Golmie, F. Mouveaux, L. Hester, Y. Saintillan, A Koenig, D.Su. Dec 1998. *The NIST ATM/HFC Network Simulator Operation and Programming Guide.* Version *4.0*. National Institute of Standards and Technology.

[6] YATS- Yet Another Tiny Simulator (ATM Simulation) http://www.ifn.et.tu-dresden.de/TK/yats/yats.html.

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

[8] INSANE: An Internet Simulated ATM Networking Environment. http://www.ca.sandia.gov/~bmah/Software/Insane/ . Last modified: Nov25, 1998. Bruce A. Mah.

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

[10] Jbuilder Overview. http://www.jbuilder-training.com/overview.htm. Last modified: March 22, 2000. The DSW Group.Ltd.

[11] Kai Yeung Siu, Raj Jain. 1997. *A Brief Overview of ATM: Protocol Layers, LAN Emulation, and Traffic Management.*

[12] P.E. Boyer, D.P. Trachier. 1992. *A Reservation Principle with Applications to the ATM Traffic Control. Computer Network and ISDN Systems.* Vol. 24. Pp. 321-334.

[13] P. Newman and G. Marshall. 1993. *Update on BECN Congestion Control. AF-TM 94-855R1.* September 1993.

[14] D. Kataria. 1994. *Comments on Rate-based Proposal. AF-TM 94-0384.* May 1994.

[15] Per Skovgaard. 1996. *ATM Traffic Management. A Technology White Paper.*

[16] H. Tzeng and K. Siu. 1994. *Adaptive Proportional Rate Control for ABR Service in ATM Networks.* University of California, Irvine, Technical Report 94-07-01.

[17] B. Lyles and A. Lin. *Definition and Preliminary Simulation of Rate-based Congestion Control Mechanism with Explicit Feedback of Bottleneck Rates.* AF-TM 94-0708, July 1994.

[18] V. F. Hartanto, H.R. Sirisena. 1993. *User-Network Policer: A New Approach for ATM Congestion Control.*

[19] ATM Connections. http://www.ieng.com/univercd/cc/td/doc/product/wanbu/9_1/bpx/bpxref/index.htm. Last Updated: 1998.

[20] V.G. Kulkarni, Natarajan Gautam. December1995. *Leaky Bucket: Sizing and Admission Control.*

[21] Michael G. Hluchyj, Nanying Yin. 1996. *A Second-Order Leaky Bucket Algorithm to Guarantee QOS in ATM Networks.*

[22] S.Mascolo, D.Cavendish, M.Gerla. 1996. *ATM Rate Based Congestion Control Using a Smith Predictor: an EPRCA Implementation.*

[23] Fabio M.Chuissi and Y.T. Wang. 1997. *An ABR Rate-Based Congestion Control Algorithm for ATM Switches with Per-VC Queuing.*

[24] Todd Lizambri, Fernado Duran and Shukri Wakid. *Priority Scheduling and Buffer Management for ATM Traffic Shaping.*

[25] A. Agrawal, M. Bayoumi, and A. Elechouemi. 1995. *A New ATM Congestion Control Scheme For Shared Buffered Switch Architectures.*

[26] Toshihisa OZAWA. 19 December 1999. Performance Characteristics of a Packet-Based Leaky-Bucket Algorithm for ATM Networks.