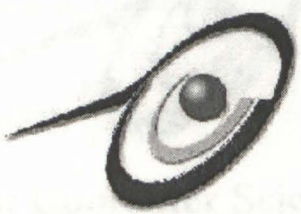
PC-BASED ATM NETWORK SIMULATOR

Submitted by

LAI WEEI SHAN

WET 97035

Submission Date
25th January 2000



Faculty of Computer Science and Information Technology,
University of Malaya
1999/2000

PC-BASED ATM NETWORK SIMULATOR

LAI WEEI SHAN

WET 97035

Under The Supervision of

MR. LING TECK CHAW

Dissertation submitted in partial fulfillment of the requirements for the Degree of Bachelor of Information Technology

Faculty of Computer Science And Information Technology
University of Malaya
1999/2000

ABSTRACT

Asynchronous Transfer Mode (ATM) has emerged as 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 project is divided into two parts:

Part one:

Study the concept of ATM and the important issues regarding it. Various existing simulators and applications on ATM are studied to plan for an up to date simulator.

Part two:

Develop a PC Based simulator. This is motivated by the importance of an ATM network simulator and the changing environment in computer world. The simulator is developed to provide a simulated network management environment of unlimited scale at a fraction of cost, compared to the expense of testing it out in a real network environment. With this simulator, further research on ATM can be carried out smoothly.

ACKNOWLEDGEMENT

My utmost gratitude to Mr. Ling Teck Chaw, supervisor of this project. His patience, guidance and advice throughout the whole development of the project is deeply appreciated. Special thanks to Mr. Phang Keat Keong for being a considerate and kind moderator.

I would like to thank my project partners, Ms. Looa Lee Shin and Ms. Toh Kue Chin for their dedicated work and corporation. They have been a great help and have shown their generosity in sharing their knowledge.

Last but not least, my deepest appreciation to my family and friends for their continuous support to complete this project.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Chapter 1 INTRODUCTION

Asynchronous Transfer mode (ATM) is emerging as the primary networking technology for next-generation, multi-media communications. It is designed to permit telephony, video, and data communications on the same network using statistical multiplexing to provide *bandwidth on demand.* ATM protocols are designed to handle time critical data and more conventional inter-computer data communications. One most important point in ATM is that it supports quality of service (QoS) requirements.

This chapter provides a brief concept on ATM, such as the information transfer, ATM cell identifiers, routing concepts, ATM resources, usage parameter control and flow control. Next, five service categories and various QoS parameters are looked into with more detail. Finally, the need of simulation on ATM network is discussed in short.

1.1 Background

There has been much progress in the transfer of data over networks for the past decades. At first, data were sent over phone lines using modems. Modifications were made to allow quicker transfer of limited data to multiple destinations using the established telephone networks. Then, X.25 evolved with the idea of packet switched data that could be routed to any location. Speed of data transfer and data file size

continues to increase and transfer media changed. Thus, Frame Relay was introduced. The fiber optic transmission medium such as SONET/SDH was then used to provide large bandwidth capacity. To overcome the lack of switching capability, (ATM) was proposed as the switching structure of B-ISDN but has taken on its own role outside of B-ISDN [1].

1.2 Asynchronous Transfer Mode (ATM)

1.2.1 Information Transfer

ATM is a fast packet oriented transfer mode based on asynchronous time division multiplexing and it uses fixed length (53 bytes) cells. Each ATM cell consists of 48 bytes for information field and 5 bytes for header. The header is used to identify cells belonging to the same virtual channel and thus used in appropriate routing. Cell sequence integrity is preserved per virtual channel. The information field of ATM cells is carried transparently through the network. No processing like error control is performed on it inside the network [2]. ATM Adaptation layers (AAL) are used to support various services and provide service specific functions. This AAL specific information is contained in the information field of the ATM cell [3].

Fig 1 ATM Cell Structure at NNI (network-network interface)

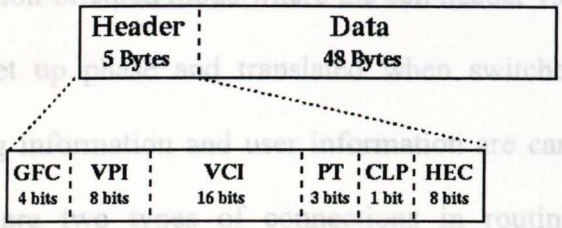
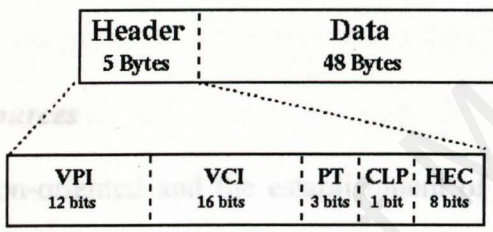


Fig.2 ATM Cell Structure at UNI (user-network interface)



- VPI : Virtual Path Identifier
- VCI : Virtual Channel Identifier
- PTI : Payload Type Identifier
- CLP : Cell Loss Priority
- HEC : Header Error Control
- GFC: Generic Flow Control

1.2.2 ATM Cell Identifiers

ATM cell identifiers, which are the Virtual Path Identifier, Virtual Channel Identifier and Payload Type Identifier (PTI), are used to recognize an ATM cell on a physical transmission medium. VPI and VCI are same for cells belonging to the same virtual connection on a shared transmission medium.

1.2.3 Routing

ATM is a connection-oriented mode where the cell header values are assigned during the connection set up phase and translated when switched from one section to another. Signaling information and user information are carried on separate virtual channels. There are two types of connections in routing, the Virtual channel connection (VCC) and Virtual path connection (VPC). A VPC is an aggregate of VCCs. Switching on cells is first done on the VPC and then on the VCC.

1.2.4 ATM Resources

ATM is connection-oriented and the establishment of the connections includes the allocation of a VCI and/or VPI. It also includes the allocation of the required resources on the user access and inside the network. These resources, expressed in terms of throughput and quality of service, can be negotiated between user and network either before the call-set up or during the call.

1.2.5 Usage Parameter Control (UPC)

Excessive reservation of resources by one user may affect traffic for other users. Thus throughput must be policed at the UNI by UPC function in the network to ensure that each user maintains the negotiated connection parameters per VCC or VPC between network and subscriber. Traffic parameters that describe the desired throughput and QOS in the contract have to be monitored in real time at the arrival of each cell. To enforce the conformance of QoS established, one of this is done at the arrival of cell across the UNI on each connection:

1. pass the cell into the network without changing the CLP bit in the cell header
2. change the CLP bit in the cell header to 1 (tagging)
3. discard the cell [4]

1.2.6 Flow Control

A Generic Flow Control (GFC) mechanism is proposed by CCITT at the User to Network Interface (UNI) in order to control the flow of traffic on ATM connections from a terminal to the network. It is supported by GFC field in the ATM cell header. Two sets of procedures are associated with the GFC field, which are the Uncontrolled Transmission in point-to-point configurations and Controlled Transmission in both point-to-point and shared medium configurations.

1.3 ATM Service Categories

ATM service categories are defined to allow flexible access to network resources and provide the ability to find satisfactory compromise between performance and cost. Without defining the traffic in ATM into service categories, managing and providing desired QoS for different applications will be very complex because of the different delay or loss sensitiveness. Therefor, the ATM Forum Traffic Management Specification 4.0 has defined five service categories:

1.3.1 Constant Bit Rate (CBR)

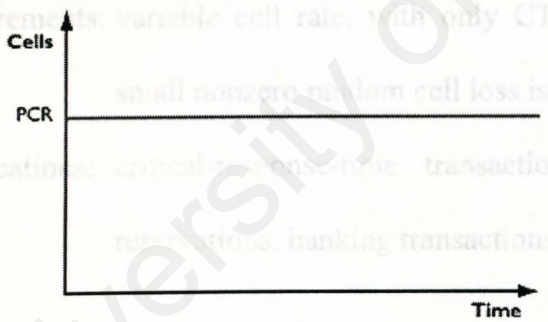
Defined as Class A traffic.

This service category is used for connections needing a continuous static amount of bandwidth throughout the connection. CBR is characterized by Peak Cell Rate (PCR).

Quality requirements: constant cell rate, CTD and CDV tightly constrained; low CLR.

Typical applications: voice and video, including videoconferencing and audio/video distribution and retrieval (television, distance learning, pay-per-view, video-on-demand, and audio library).

Fig. 3 CBR



CBR uses a static amount of bandwidth continuously throughout the connection.

1.3.2 Real-Time Variable Bit Rate (VBRrt)

Defined as Class B traffic.

This service category supplies tightly constrained delay and delay variation, but not necessarily a fixed transmission rate. It is characterized by PCR, Sustained Cell Rate (SCR), and Maximum Burst Size (MBS).

Quality requirements: variable cell rate, with CTD and CDV are tightly constrained; a small nonzero random cell loss is possible.

Typical applications: native ATM voice with bandwidth compression and silence suppression, interactive compressed video, multimedia communications.

1.3.3 Nonreal-Time Variable Bit Rate (VBRnrt)

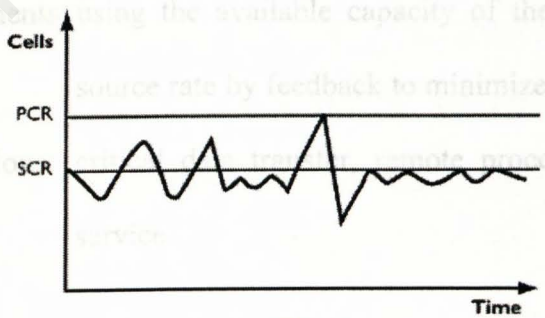
Defined as Class C traffic.

This service category is for nonreal-time bursty traffic that requires service guarantees. It is also characterized in terms of PCR, SCR and MBS.

Quality requirements: variable cell rate, with only CTD are tightly constrained; a small nonzero random cell loss is possible

Typical applications: critical-response-time transaction processing, (e.g. airline reservations, banking transactions) and frame relay.

Fig. 4 VBRrt AND VBRnrt



VBRrt and VBRnrt traffic define an SCR (the upper bound of the average cell rate), a PCR, and an MBS.

1.3.4 Specified Bit Rate (UBR)

Defined as Class D traffic.

This service category is also referred to as a best-effort service. It is for nonreal-time and bursty applications which are tolerant to delay and loss with no QoS gurantees.

Quality requirements: using left-over capacity, no CTD or CDV or CLR constrained.

Typical applications: text, data, and image transfer, email, distributed file services, and computer process swapping and paging, applications such as LAN interconnecting and internetworking which run over router-based protocols like TCP/IP.

1.3.5 Available Bit Rate (ABR)

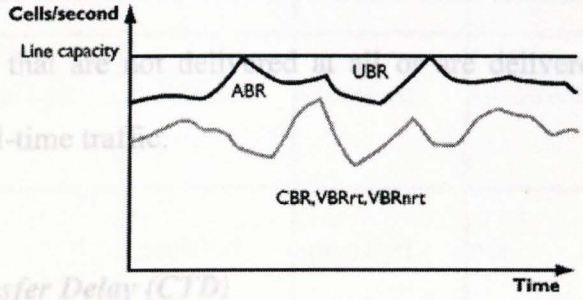
Defined as Class D traffic.

This service category provides a Minimum Cell Rate (MCR) with any available bandwidth. ABR connections are characterized by both a PCR and an MCR, and are used by applications that can adapt to network feedback. A flow-control mechanism fairly allocates the available bandwidth among ABR connections.

Quality requirements: using the available capacity of the network and control the source rate by feedback to minimize CTD, CDV and CLR.

Typical applications: critical data transfer, remote procedure call, distributed file service.

Fig.5 ABR AND UBR



ABR traffic is supplied an MCR and any available bandwidth, but UBR traffic does not define a per-connection negotiated bandwidth.

1.4 Quality of Service (QoS) Parameters

QoS parameters are used to describe the level of service for each connection. It is first used to characterize the performance of data transmission in terms of reliability, delay and throughput. QoS requirements are specified by the application while the QoS guarantee is provided by the system. Six QoS parameters are identified in the ATM Forum's User-to-Network (UNI) 4.0 specification, for which one or more performance objectives may be supported by the network. Through the use of network signaling, three of these describe characteristics of the network and are measured at the receiver.

1.4.1 Cell Loss Ratio (CLR)

CLR is the ratio of lost cells to total transmitted cells. It measures the fraction of the transmitted cells that are not delivered at all or are delivered so late that became useless as for real-time traffic.

1.4.2 Cell Transfer Delay (CTD)

CTD is the delay time between a cell's first bit transmission at the source and its last bit entry at the destination. CTD accounts for both node processing and internode transmission time. Maximum Cell Transfer Delay (Max CTD) and Mean Cell Transfer Delay (Mean CTD) are used.

1.4.3 Cell Delay Variation (CDV)

CDV, sometimes called cell jitter, is a measure of the inter-cell departure of a given connection with respect to its inter-cell arrival. While cells may be sent into the network evenly spaced, a variety of factors may contribute to cell clumping or gaps in the cell stream. If the network cannot properly control CDV, distortion can occur for some real-time services such as voice, video, and multimedia applications.

The QoS and Usage parameters for the five service categories are summarized in Table 1:

Table 1: QoS and usage parameters for ATM Service Categories

Parameters	CBR	rt-VBR	nrt-VBR	UBR	ABR
PCR and CDVT(5)	Specified	specified	specified	specified(3)	specified(4)
SCR, MBS, CDVT(5,6)	n/a	specified	specified	n/a	n/a
MCR(5)	n/a	n/a	n/a	n/a	specified
Peak-to-peak CDV	Specified	specified	Unspecified	unspecified	unspecified
Mean CTD	Unspecified	unspecified	Specified	unspecified	unspecified
Max CTD	Specified	specified	Unspecified	unspecified	unspecified
CLR(5)	Specified(1)	specified(1)	Specified(1)	unspecified	specified(2)

Notes:

1. The CLR may be unspecified for CLP=1.

2. Minimized for sources that adjust cell flow in response to control information.

3. May not be subject to CAC and UPC procedures.

4. Represents the maximum rate at which the source can send as controlled by the control information.

5. These parameters are either explicitly or implicitly specified for PVCs or SVCs.

6. Different values of CDVT may be specified for SCR and PCR. [2]

1.5 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 are yet to be studied. Besides, even though many ATM switches are offered by different vendors, little is known about the performance of these switches in real networking environments. The throughput of different network topologies has to be tested and analyzed in order to maximize the utilization of resources.

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 different settings for the various network components. With the use of a simulator, researchers and network planners are able to analyze networks without the expense of building a real network.

1.6 Project Motivation

This project is to build a PC Based simulator to simulate the ATM networks. The need to simulate and analyze the ATM network is 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. Simulation is needed to study the queuing delays, throughput and buffer occupancies at various nodes in the network.

Although there are several simulators available, not all are for the ATM network. Furthermore, PC Based simulator can hardly be found. Regarding this, a PC Based simulator is aimed to provide a more interactive environment with graphical user interface (GUI) which is more user friendly to operate.

1.7 Project Scope

The project starts with the research and study on ATM. The way of information transfer in ATM is studied to obtain the idea of how the information can be transferred from one endpoint to the other endpoint through the network. The routing protocols, bandwidth allocation, usage parameter control and flow control are also studied to understand the basic concept of ATM networking. Detailed study is done on the five service categories and QoS so that they can be implemented in the simulator.

Various available simulators are determined and compared. The advantages and disadvantages of each simulator are analyzed. This is to determine the basic requirements of developing a practical simulator. The simulator components needed for an ATM network, such as the ATM switches, terminal equipment, ATM

applications and physical links are studied to acquire the idea of how the ATM simulator is to be built.

The proposed simulator will have the features of:

- Running under Windows platform
- A user friendly graphical user interface (GUI)
- Button click command of simulation functions
- Default parameters for simulator components
- Several predefined networks
- Discrete event schedule
- Integrated data analysis tool

1.8 Project Schedule

Porject schedule is attatched next page

		Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb
Literature Review	13-6-99 To 14-8-99									
System Analysis	15-8-99 To 11-9-99									
System Design	12-9-99 To 16-10-99									
System Implementation	17-10-99 To 18-12-99									
Testing	19-11-99 To 30-1-00									
Documentation	19-6-99 To 6-2-00									

CHAPTER 2

LITERATURE REVIEW

Chapter 2 LITERATURE REVIEW

2.1 Introduction

A network simulator is used to study various issues regarding the design of networks, especially high-speed networks. Performance of these networks can be analyzed. Furthermore, there is no need to build a real network for research purposes.

A few existing simulators are reviewed to study the usage of each simulator. This is because different simulator focuses on different issue of simulation and different field of applications. Applications in an ATM network are determined so that they can be used to generate all possible traffic. Besides the standard applications, which are the CBR, VBRrt, VBRnrt, ABR and UBR, there are several other applications that will be reviewed in this chapter. Survey is also done on Unix and Windows to study the need and feasibility of developing a PC Based simulator.

2.2 Existing Simulators

2.2.1 Network Simulator (NETSIM)

NETSIM is a discrete event simulation language. It is designed to simulate large networks that use modern routing techniques, such as worm-hole and virtual-cut-through routing, networks that use store-and-forward routing can also be simulated. NETSIM is capable of simulating a full parallel system that is driven by the execution of real programs. Besides, it can be used in a stand-alone mode where

packets are generated randomly and sent through the network. It is implemented with a set of data structures and subroutines in the C programming language [5].

Advantages and Disadvantages

✓ Support parallel simulation

✓ Modular characteristic to allow specification of different network structures by connecting network building blocks

× No graphical user interface (GUI) to provide an interactive modeling environment

× The traffic management function and QoS is not available

2.2.2 OPNET Simulator

OPNET is a very well designed and built commercial simulation software, which was developed from MIL3 Inc. OPNET is designed to enable full-detail modeling where every tool is given to implement nonstandard protocols or behavior. It provides graphical object-oriented editors for defining topologies and architectures. This simulator operates at three hierarchical levels to describe and control the network. These are network levels where network nodes can be connected by means of unidirectional or bi-directional communication links and bus link, the node level and the process level that operates on the packets as they go through the processor and queue modules of the higher node level [3], [7].

Advantages and Disadvantages

✓ Able to simulate large networks that use modem routing techniques

- ✓ Graphical model editor to create different topologies easily
- × It does not support parallel execution
- × Only fixed (unparametrized) topologies can be created and it lacks efficient support for building large, regular topologies
- × No graphical user interface (GUI) for an interactive modeling environment

2.2.3 Objective Modular Network Testbed in C++ (OMNeT++)

OMNeT++ is a discrete event simulation tool developed on Linux but is usable on most Unix systems, on Windows NT and even DOS. It is primarily designed to simulate computer networks, multi-processors and other distributed systems [8].

Advantages and Disadvantages

- ✓ Support parallel simulation
- ✓ Interactive (GUI) execution environment, and provides more open interfaces
- ✓ Written in full flexibility of C++ and built-in object-oriented mechanisms
- × Command based simulator
- × Lack of model library for standard protocols, applications and devices

2.2.4 Private Network-Network Interface (PNNI) Simulator

This was developed at the National Institute of Standards and Technology (NIST) to study the performance of the PNNI routing protocols in a multi-node network environment. It provides an interactive modeling environment with a user interface. User can enter various physical network topologies and PNNI parameters. It is

written in C language. The codes are the user interface and the PNNI routing protocols [9].

Advantages and Disadvantages

- ✓ Interactive modeling environment
- ✓ Support various physical network topologies and PNNI parameters
- × The user interface is only in text form and not a graphical user interface (GUI)
- × Only PNNI routing protocols can be studied
- × The messages can only be sent between the ATM switch and Physical Links

2.2.5 Data Link Protocol Simulator (DLPsim)

This simulator allows data link protocols to be coded and tested. It allows a single protocol implementation to be `plugged' in at each end of a simulated communications channel. DLPsim checks for common protocol errors such as delivering packets out of order, or losing packets. With a Motif library, it provides a graphical front end that allows some visualization of the activity on the simulated channel. The protocol implementation can be exercised by the simulator, with simulated communications traffic, transmission errors, and time outs. The simulator checks that the protocol does not lose or duplicate any packets, and gives an indication of the throughput of the protocol. It is running under X windows or a traditional `glass teletype' terminal [10].

Advantages and Disadvantages

- ✓ User interface with a pull down menu bar at the top of the main window with buttons
- × The protocol is written in a slightly restricted C.
- × The simulator does not recognize the full generality of the C language. The preprocessor is purely lexical in its operation and the full complexities of C declarations are beyond the scope of the preprocessor

2.2.6 NIST ATM/HFC Network Simulator

The ATM/HFC Network Simulator was developed at the NIST. It is based on a network simulator that supports discrete event simulation techniques with graphical user interface (GUI) representation capabilities, which was developed at MIT1. It provides a flexible testbed for studying and evaluating the performance of ATM and HFC networks. User is allowed 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. Besides, the simulator also has an event manager, I/O routines and various tools that can be used to build components. It is developed in both C language and the X Window System running on a UNIX platform [11].

Advantages and Disadvantages

- ✓ Support discrete event simulation techniques
- ✓ Performance measures can be displayed instantaneously

- ✓ Various ATM network component available for sending messages to one another
- ✓ Graphic user interface (GUI) representation capabilities
- × Too many parameters are needed
- × Specific steps have to be followed when building a network topology

2.3 Simulation of ATM Applications

ATM application, as a component required in the simulator, is used to emulate the behavior of an application at the end-point of a link. It can be considered as a traffic generator where the traffic types may be a Constant Bit Rate (CBR), Variable Bit Rate (VBR), Available Bit Rate (ABR) or Unspecified Bit Rate (UBR). Traffic source is generated at one of the three priority levels:

1. CBR/VBR level – which is the highest priority traffic
2. ABR level – cells are sent on the transmission bandwidth that is available after the higher level traffic has been sent
3. UBR level – which is the lowest priority traffic

For CCBR/VBR and ABR classes, there are three types of traffic generators:

1. Constant – where the user specifies the bit rate and cells will be generated at the specified rate.
2. Poisson – this type of traffic will have an ON-OFF source. Both the burst

period (ON) and the silence period (OFF) are drawn from an exponential distribution. Mean burst length, mean interval between bursts and the bit rate are specified.

3. Batch – mean number of cells generated during a burst and the mean interval between bursts are specified.

Other ATM Applications include the TCP/IP application, which can be used with either the ABR or UBR services. A statistic shows that more than 85% of networks are running TCP/IP, Transmission Control Protocol and Internet Protocol [12]. This is a higher layer application that can run over ATM through LAN Emulation (LANE).

Motion Picture Experts Group, (MPEG) traffic application on VBR service is another application that have to be taken care of. It is different from video transmission that has to use CBR. Due to compression and systems where only changes to a video scan picture are sent, it is now a low-delay, variable bit rate service [4].

There is also self-similar traffic application on VBR service. Self-similar traffic is the data type that displays structural similarities across a wide range of time scales. Any traffic data that is bursty on many or all time scales is considered to be self-similar [13]. By modeling the networks with self-similar traffic sources, the network designer can determine the optimal buffer size of the designed network.

2.4 Unix Based Versus Windows Based Platform

2.4.1 Why Unix Based Simulator

Most simulators were built and used in Unix platform previously. The main reason is the performance of Unix that surpasses the performance of Windows. Unix operating system has been the host operating system developed 25 years ago by Bell Labs [14].

Unix has an Open Systems operating system, which is portable, scalable and powerful. This has provided a superb development environment in Unix. At one time, Unix was the only powerful system that could met user's demand on a 3-D model that can rotate faster and a simulation that can move faster [15]. Applications that are focused on the floating point/Unix equation, such as simulation used to be done on Unix due to the insufficiency of Windows operating system.

2.4.2 X Window System

The X Window System is a technology used in Unix, developed out of a project at MIT in the 1980's. It is a set of tools and standards that allow developers to build applications and write programs that can draw on a bitmapped screen. It can receive input from a keyboard and respond to the movement of a pointing device [14].

2.4.3 The Changing Environment

With the rapid development on Windows system, the gap of performance is closing up. Moreover, Unix was designed by programmers for programmers, where it is complex, hard to use and difficult to administer [14]. It is not suitable for current situation where people tend to find the easiest way out.

Next is the issue of X Windows technology. As Windows is becoming the dominant desktop platform, more user demand a Windows environment. Since X is not Windows, and the need of user is changing towards a better and more interactive graphics in PC, the development of a PC Based simulator cannot be neglected.

2.4.4 Feasibility of Transformation of Unix Base Simulator to PC Based Simulator

Unix operating system is written in high level language, which is the C language. This enables its applications to be easily portable to other operating system [14]. Moreover, the codes are usually very modular. Thus, simulator that is Unix based can be ported to Windows based without much trouble.

X applications are developed using program libraries, where these libraries implement a large number of program structures called "widgets". A widget is a graphical object, which defines a screen object and its behavior. The codes can be rewritten to provide a windows environment.

The price/performance of Windows has increased and is able to compete with Unix. Today, Windows not only provides similar functionality like Unix, its performance is fast enough to support applications that needed Unix previously. A lower initial price and lower cost of ownership [16] has encouraged more and more applications, which used to be strictly for Unix, such as simulations being ported to Windows.

2.5 Conclusion

From the review on Unix and Windows, and the changing environment of computer world, it is feasible to change the platform of current simulator, which is from Unix Based to PC Based. There is also a need to develop a PC Based simulator due to some constraints and shortfalls of simulators in Unix. Some enhancements will be made and some new features will be added in to provide a more interactive and user friendly way of simulation. The simulator will be written in Visual C++ with modular source code. This will allow new functions and applications to be added in easily.

Features:

- Running under Windows platform
- An interactive and user friendly graphical user interface (GUI)
- Ability to emulate real characteristic of ATM network
- Button click command of simulation functions Default parameters for simulator components
- Display and edit parameters during simulation, log data to file
- Several predefined networks
- Discrete event schedule of the simulation
- Integrated data analysis tool to filter the parameters and plot graphs

CHAPTER 3

SYSTEM ANALYSIS AND REQUIREMENTS

Chapter 3 SYSTEM ANALYSIS AND REQUIREMENTS

3.1 Introduction

System analysis is carried out to determine the current system available and to determine what is best for the system to be developed. The simulator should take as input a scenario, which is a description of network topology and control parameters. It should produce as output statistics such as the number of cells received by components, the number of cells dropped, average throughput, and retransmission percentage and other similar information.

3.2 Analysis of Simulator Program

System analysis includes analyzing the ATM features that are necessary to simulate in order to study the performance of a network configuration. In this project, NIST/HFC simulator is used as the ground for analysis of an ATM simulator.

There are two major levels of operation in an ATM network. The first one is at the cell level, which is the basic level of operation of the ATM switch. The second one is at the connection level.

At the cell level, data is packeted into cells and transmitted to the UNI. Cells are queued up at the input or output ports of the switch. Cells are routed based on the VCI in its header. During congestion, cells are preferentially dropped according to

the cell loss priority bits in the header. Congestion control is implemented using a leaky bucket method, suggested by the CCITT and some other mechanisms.

At the connection level, connection parameters like peak rate and average rate are determined. Decisions regarding the routing mechanisms are done at this level. However, the main focus of simulation in this project is on the cell level.

3.2.1 ATM Cell

ATM cell is an important unit in this simulator, which is designed to simulate ATM networks. A cell constitutes a very important data type in the simulator because it contains the route number needed for routing by ATM switches.

ATM cell identifiers, Virtual Path Identifier, Virtual Channel Identifier and Payload Type Identifier (PTI) are used to recognize an ATM cell on a physical transmission medium. VPI and VCI are same for cells belonging to the same virtual connection on a shared transmission medium.

3.2.2 ATM Applications

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

3.2.3 Broadband Terminal Equipment (B-TE)

Broadband Terminal Equipment (B-TE) simulates a Broadband ISDN node like a host computer or workstation. A B-TE has one or more ATM Applications at the user side and a physical link on the network side. Cells received from the Application side are forwarded to the physical link. The cell will be queued in one of the three queues if no slot is available for immediate transmission. When either queue exceeds the specified the maximum output, queue size cells will be dropped. The BTE implements rate-based flow control algorithm for ABR connections.

3.2.4 Hybrid Fiber Coax (HFC) Network

Hybrid Fiber Coax (HFC) simulates a broadband residential Hybrid Fiber Coax network with a single trunk topology. Each source attached to the HFC module represents one station on the HFC network. However, since it shares some functions used in the B-TE, it can be categorized as the B-TE component.

3.2.5 ATM Switch

In a network, switch is the component that switches or routes cells over several virtual channel links. A local routing table is provided for each switch. This table contains a route number that is read from incoming cell structure and is the equivalent of the cell's virtual channel identifier, a next link entry, and a next switch/next B-TE entry.

Consider a cell arriving at the switch from a physical link. At the next switching slot time, after some delay (set by user), the switch looks in its local routing table to

determine which outgoing link it should redirect the cell to. At this point, if the link has an empty slot available, the switch puts the cell on the link. If a link slot is not available, the cell awaits transmission in one of the priority queues, namely, the CBR/VBR queue, the ABR queue or the UBR queue, depending on the type of service provided by this virtual channel.

Cells in the CBR/VBR queue have priority over cells in the ABR queue. This means that the ABR traffic is sent only when the CBR/VBR queue is empty. Similarly, cells in the ABR queue have priority over cells in the UBR queue. If either queue exceeds a High Threshold value, a congestion flag for that port is set to True. The three queues must be below a Low Threshold value for the congestion flag to be reset to False. The Output Queue Size will determine the available buffer space for each type of queue. Cells will be dropped if any queue exceeds the limit.

VPI and VCI translation is performed at the ATM switching and/or cross-connect nodes. At the VP switch, the value of the VPI field of each incoming cell is translated into a new VPI value of the outgoing cell. The values of VPI and VCI are translated into new values at a VC switch.

3.2.6 Link

A Link simulates the physical medium, either copper wire or optical fiber on which cells are transmitted. The information specified with a link are its speed and length. For ATM network, the normal speed of a link is defined as 155Mbits/s.

3.2.7 Routing

ATM is connection-oriented. The header values of a cell (VCI and VPI) are assigned during the connection set up phase and translated when switched from one section to other. Signaling information is carried on a separate virtual channel than the user information. In routing, there are two types of connections, Virtual channel connection (VCC) and Virtual path connection (VPC). A VPC is an aggregate of VCCs. Switching on cells is first done on the VPC and then on the VCC.

3.2.8 Simulation

The simulator is event driven. Components send each other events in order to communicate and send cells through the network. There is an event manager to provide a general facility to schedule and send or "fire" an event. An event queue is maintained in which events are kept sorted by time. To fire an event, the first event in the queue is removed, the global time is set to the time of that event and any action scheduled to take place is undertaken. Events can be scheduled at the current time or at any time in the future. Events scheduled at the same time are not guaranteed to fire in any particular order. Simulator time is maintained by the event manager in units of ticks.

3.3 Flow of Events In The Simulation Program

3.3.1 Main program flow

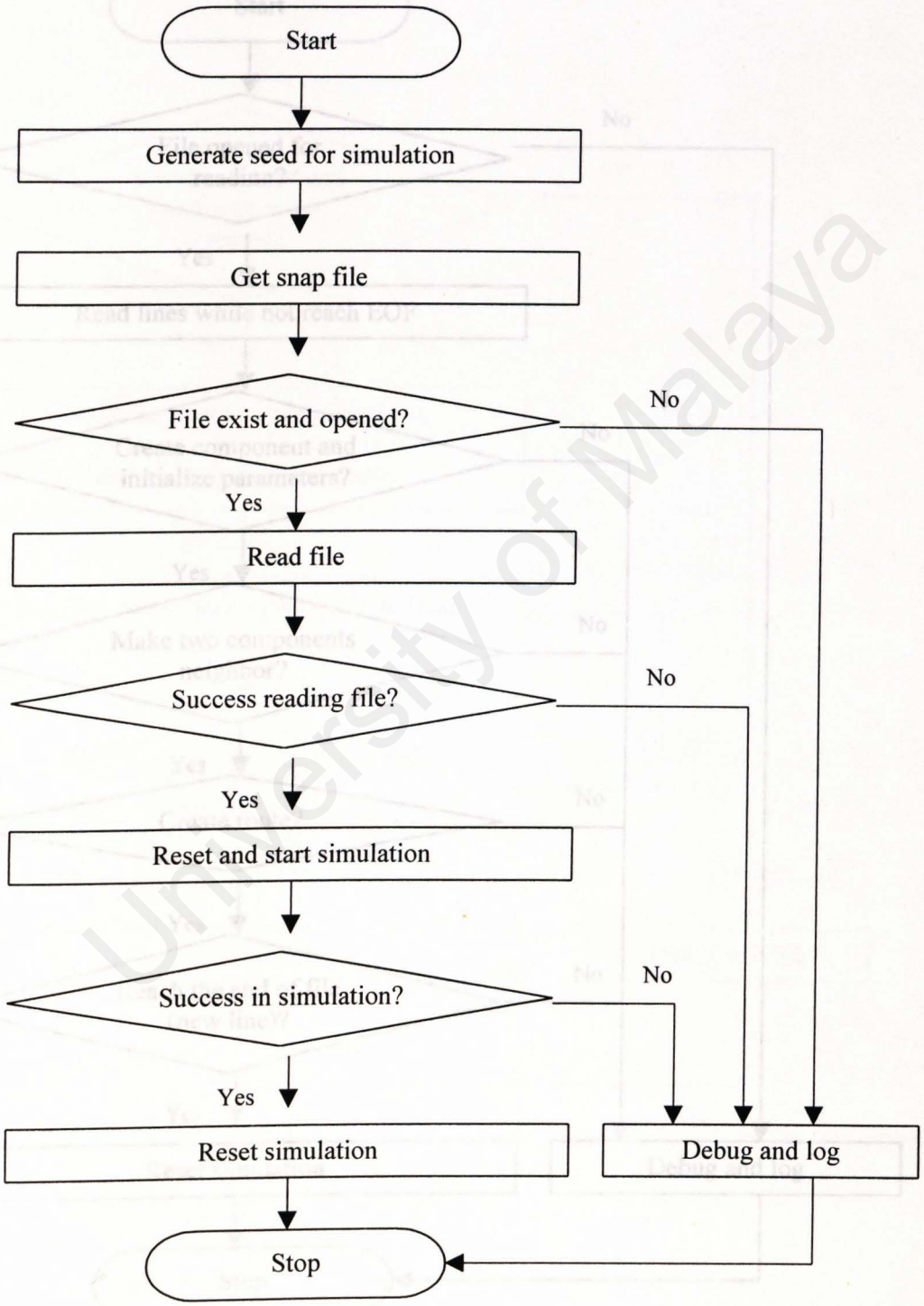


Fig. 6 Flow diagram of Main event

3.3.2 *File reading*

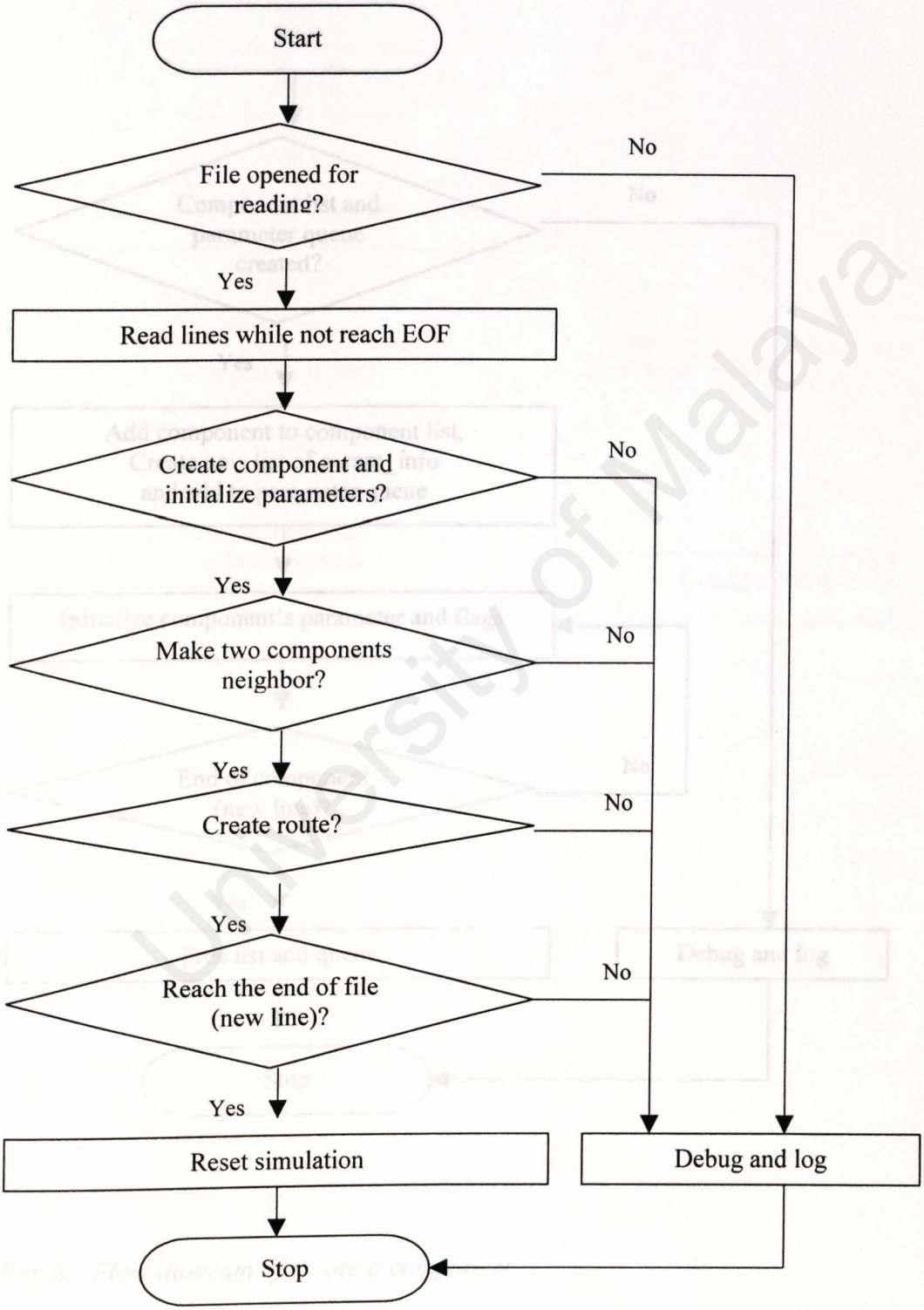


Fig. 7 Flow diagram of file reading

3.3.3 *Create a component*

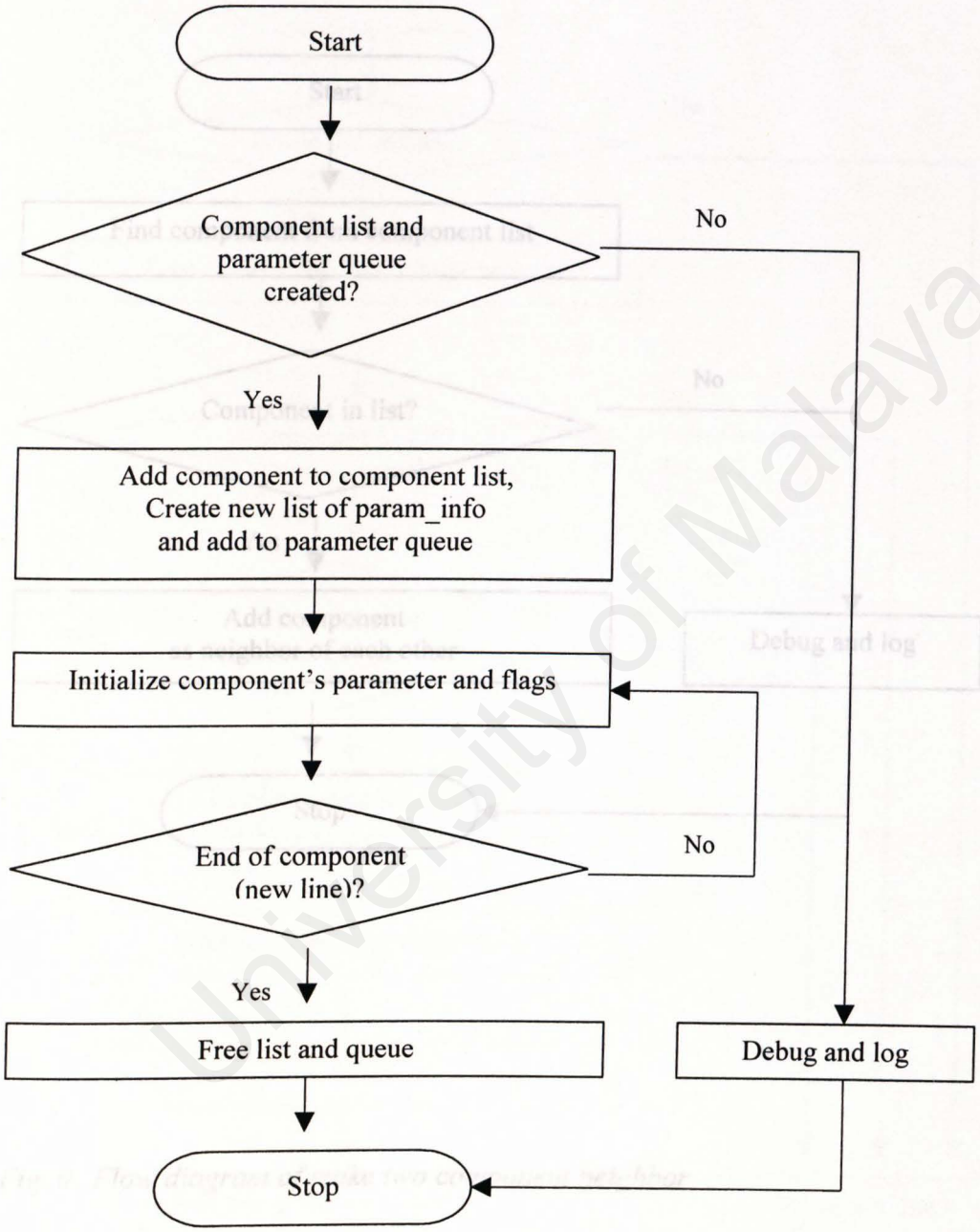


Fig. 8 Flow diagram of create a component

3.3.4 Make two components neighbor

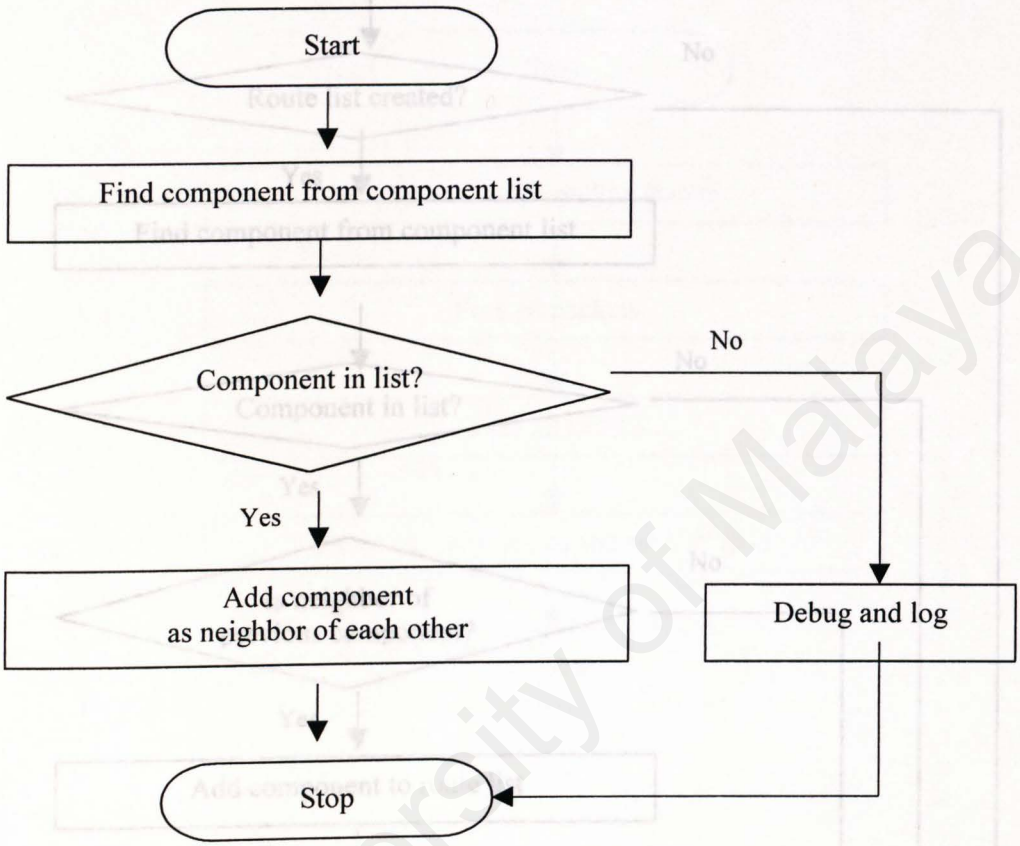


Fig. 9 Flow diagram of make two component neighbor

3.3.5 *Create route*

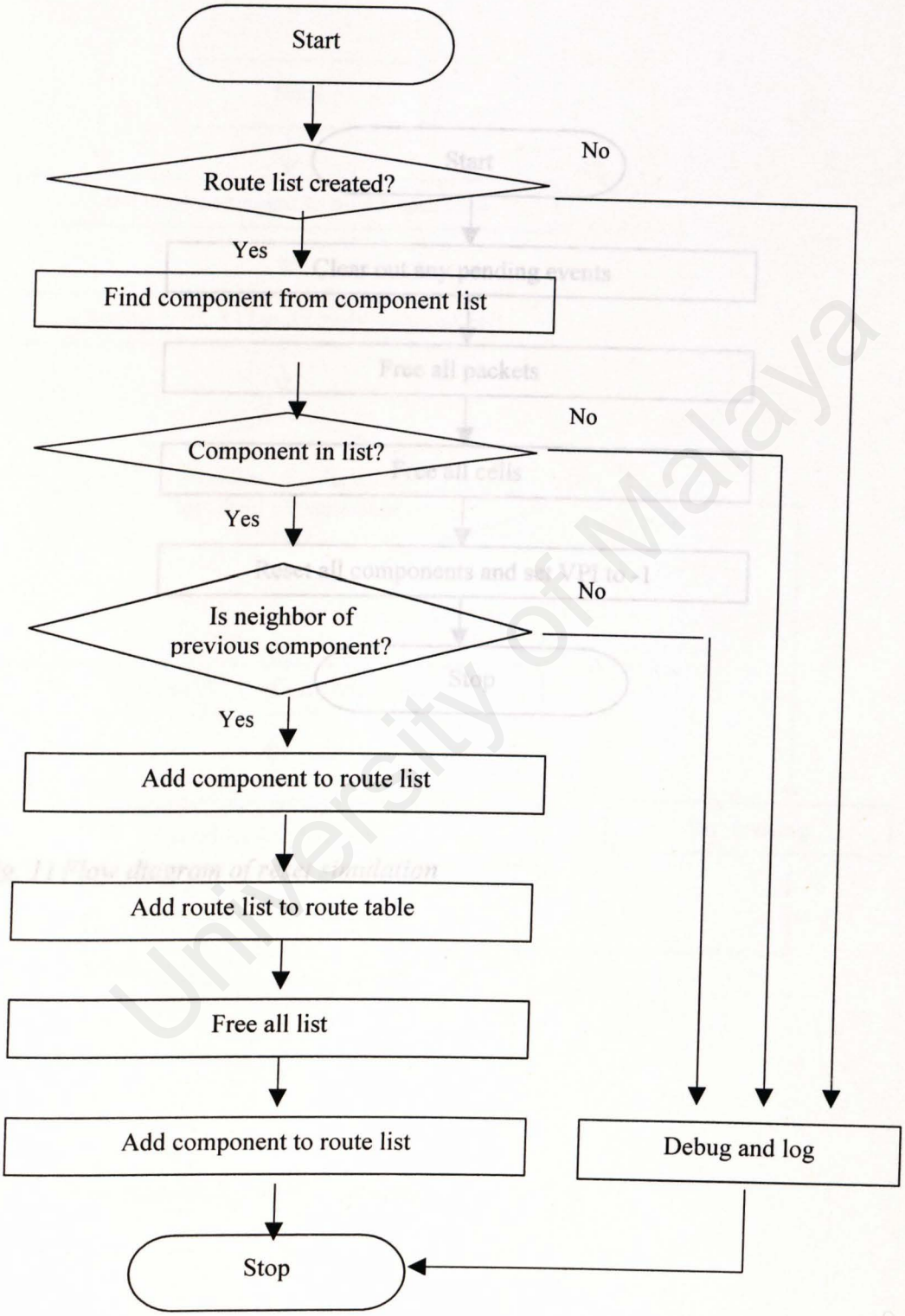


Fig. 10 Flow diagram of create route

3.3.6 Reset simulation

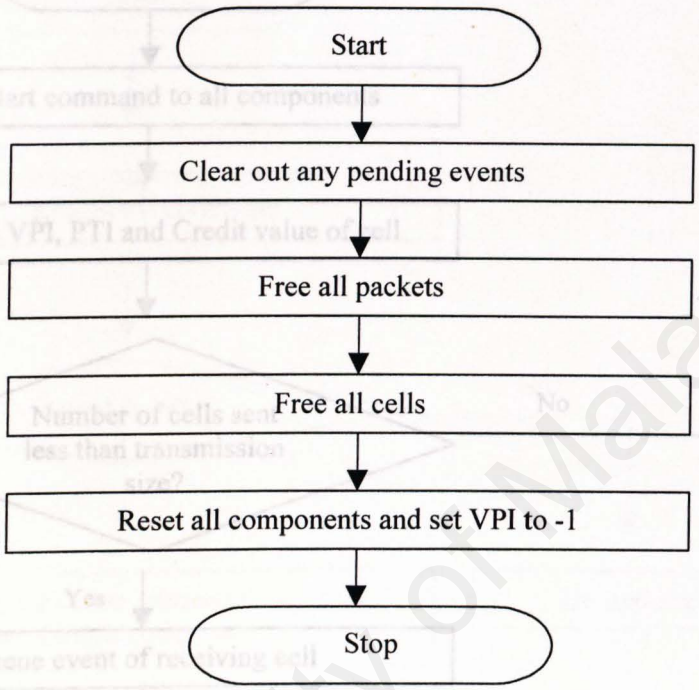


Fig. 11 Flow diagram of reset simulation

3.3.7 Start simulation

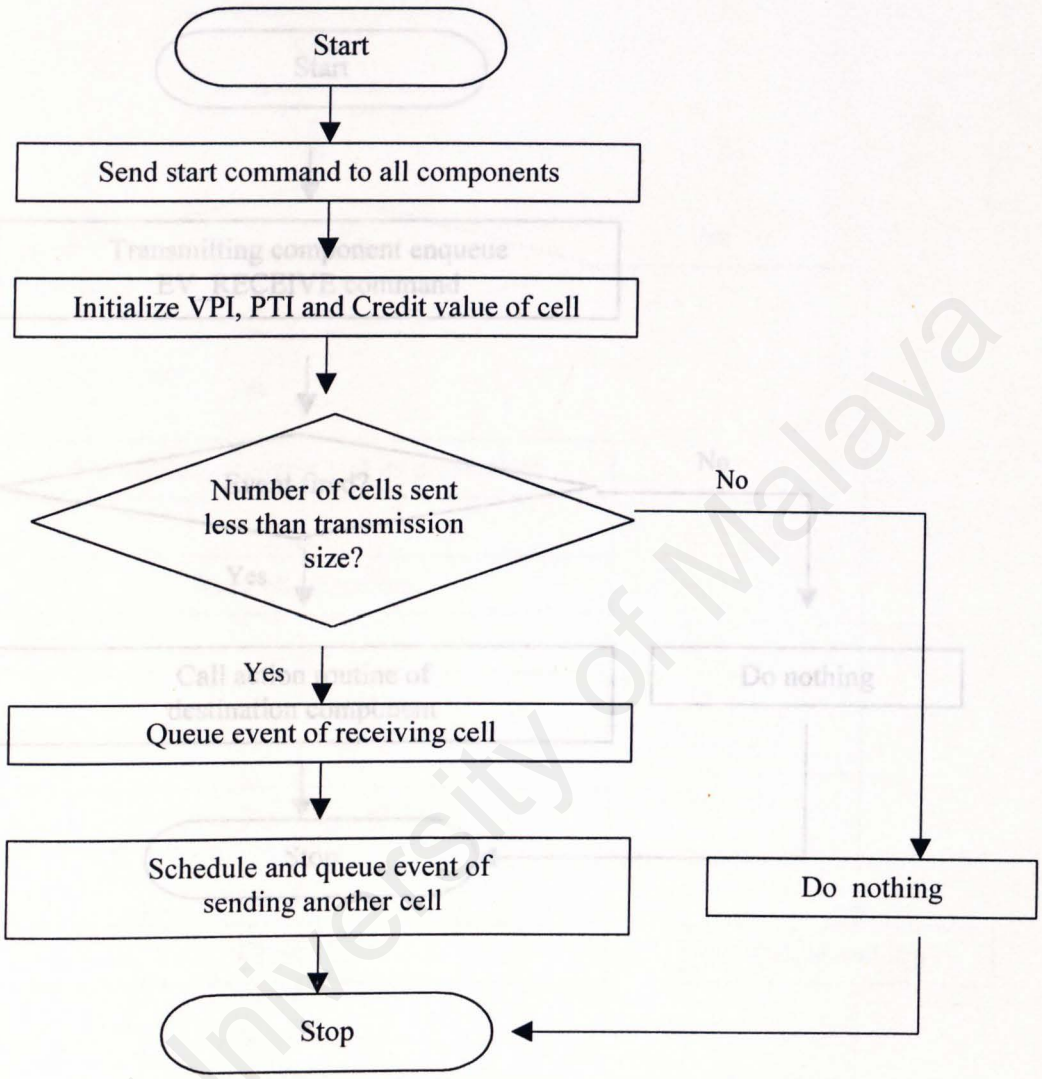


Fig. 12 Flow diagram of start simulation

3.3.8 Transmission of cells

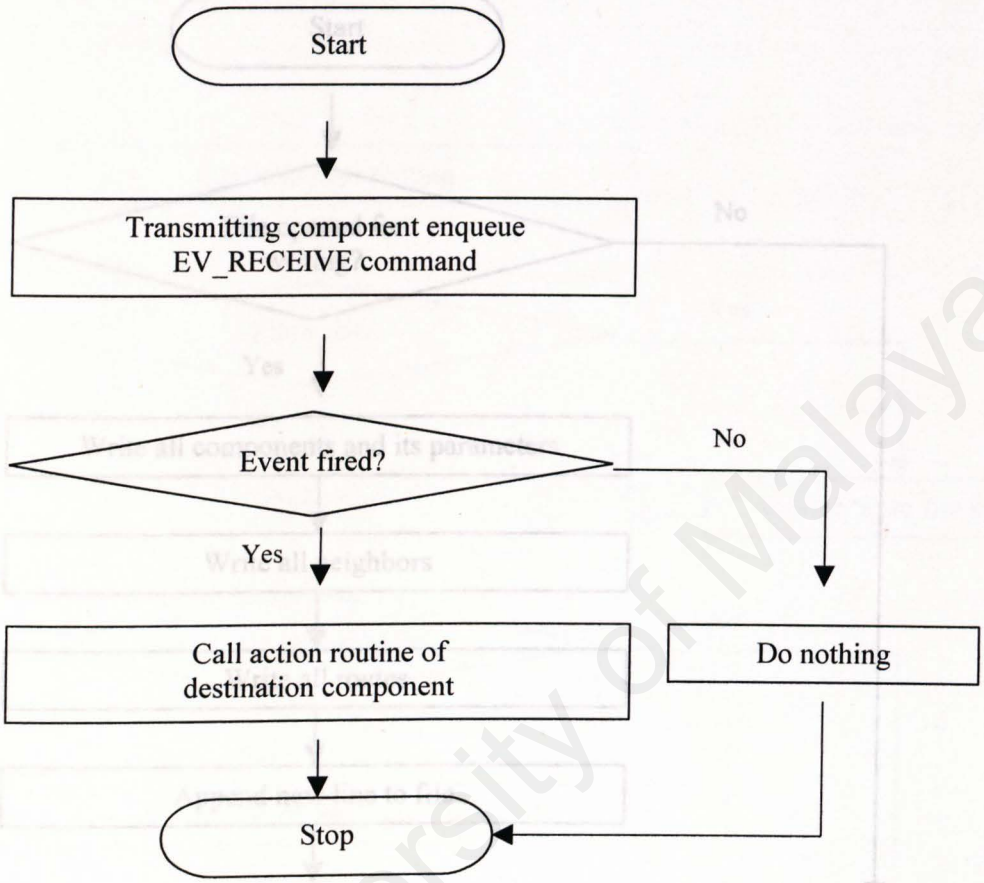


Fig. 13 Flow diagram of cell transmission

3.3.9 *File writing (snap file)*

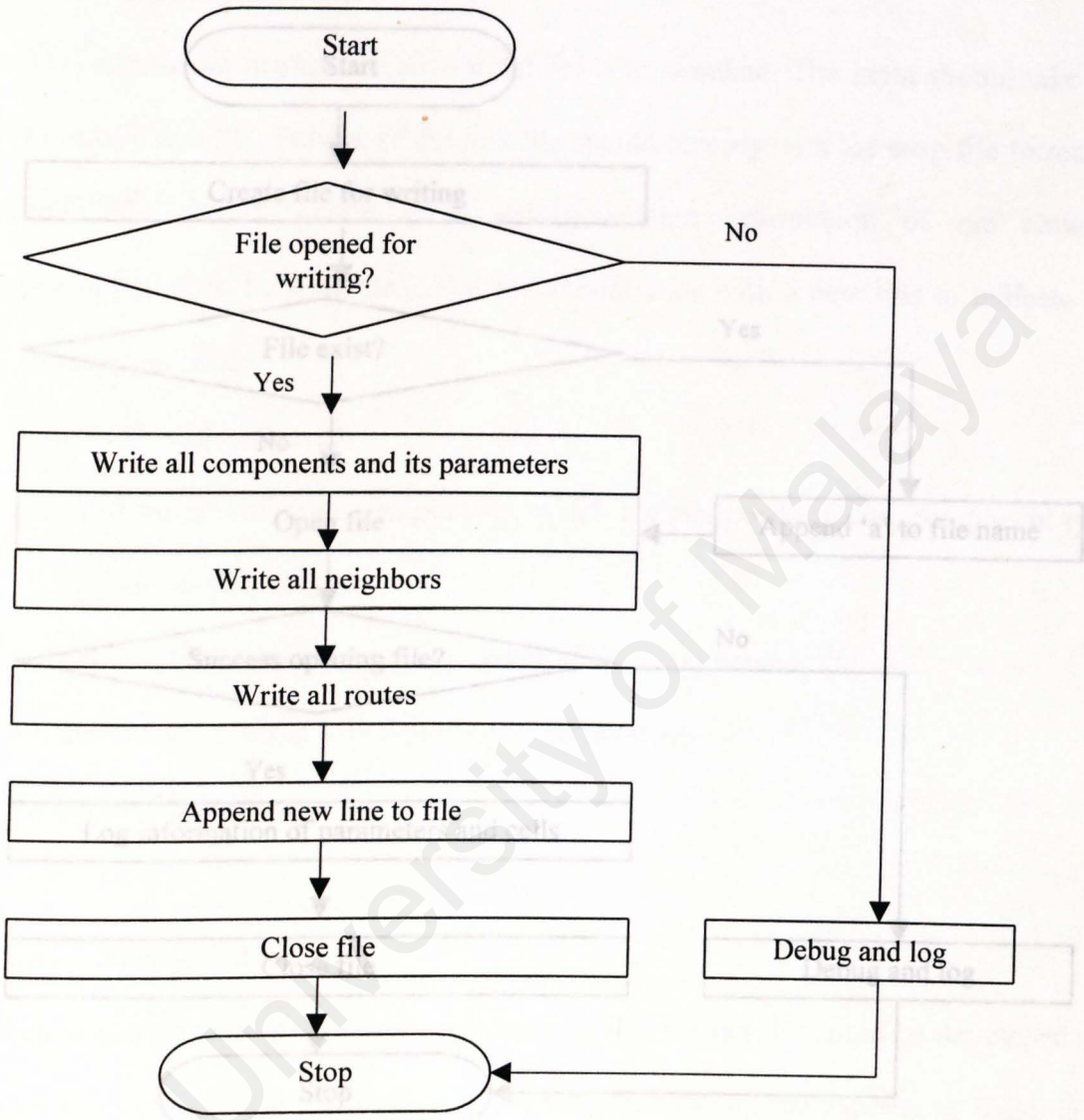


Fig. 14 Flow diagram of file writing

3.3.10 Data logging – log file

.

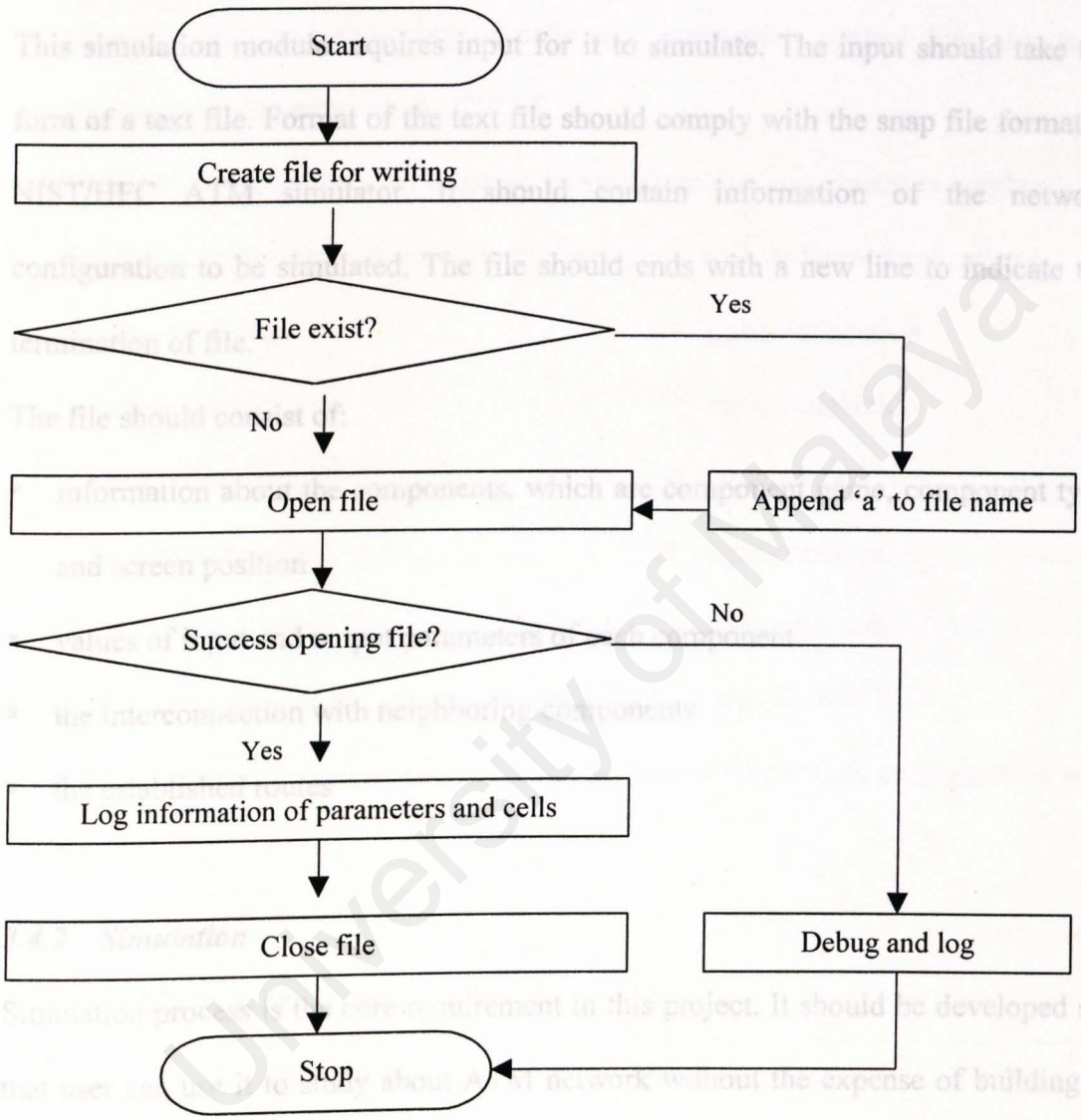


Fig. 15 Flow diagram of data logging

3.4 Requirements

3.4.1 Input of simulation

This simulation module requires input for it to simulate. The input should take the form of a text file. Format of the text file should comply with the snap file format in NIST/HFC ATM simulator. It should contain information of the network configuration to be simulated. The file should ends with a new line to indicate the termination of file.

The file should consist of:

- information about the components, which are component name, component type and screen position
- values of input and output parameters of each component
- the interconnection with neighboring components
- the established routes

3.4.2 Simulation

Simulation process is the core requirement in this project. It should be developed so that user can use it to study about ATM network without the expense of building a real one. The simulation should be designed to meet the conventions of the real ATM network as defined in CCITT.

This module should be able to:

- read network configuration from a text file as input to the simulation module
- create a virtual network configuration with parameters and other needed values as specified in the input file

- generate cells according to ATM conventions and send cells across the route created
- allow queuing and switching of cells with congestion control function using mechanisms suggested by the CCITT
- "fire" events and allows messages to be sent from one component to another
- simulate various types of network configurations with different applications
  - ⇨ abrconnection
  - ⇨ cbrconnetion
  - ⇨ atcpconnection
  - ⇨ batchconnection
  - ⇨ cbatchconnection
  - ⇨ ggbarconnection
  - ⇨ mpegconnection
  - ⇨ ssconnection
  - ⇨ tcpconnection
  - ⇨ vbrconnection
- produce an output of the simulation result, in the form of a text file
- produce another output of text file with the simulated network configuration and the seed used

3.4.3 Output of simulation

There should be two outputs at the end of the simulation:

Log file

Log file is needed because it should contain data of the simulation. Users should be able to study the performance and throughput of the network they created using the information in log file. However, the log file has to be processed or filtered before any analysis can be done. Graphs or charts can be plotted using the filtered file for analyzing purpose.

The log file should consist of:

- name of the component and the parameter logged
- a unique ID number for every parameter logged so that it can be identified throughout the file
- actual data recorded during the simulation, which are the time (in ticks) and the value of parameter at that time

Snap file

This output is used to identify the network configuration that has been simulated. It contains information about the seed (time in ticks that is used to seed the random number generator) that is used in the simulation. The rest of the format of this snap file should be the same as the input file used in the simulation.

3.4.4 Support inter platform simulation

This module should be able to support inter platform simulation. It should be able to simulate various network configurations in Windows environment as well as in Unix environment. Thus, the simulation module should be developed as a Dos-based program.

3.4.5 Programming language

C is used as the programming language for the simulation program. This is to make use of the data structure in C language. Structure is widely used in the program. Every basic component and module in the program has its own structure. Since lists and queues are extensively used in the program, C is the programming language that

able to support it. Microsoft Visual C++ 6.0 provides an integrated development environment for C and C++ applications, supporting multi-platform and cross-platform development. The debugging tool in Microsoft Visual C++ 6.0 is also better and user-friendlier compared to Turbo C. Thus, the simulation module is developed using C language in the Microsoft Visual C++ 6.0.

CHAPTER 4

SYSTEM DESIGN

Chapter 4 SYSTEM DESIGN

4.1 Introduction

During system design, the information collected during the system analysis and requirements phase is used to design the network simulation system. The simulation part of the system is designed to allow various network topologies being simulated. It is designed to simulate the network topologies created in the user interface part. The output of the simulation is also designed allow the integration of system.

4.2 Techniques Used

4.2.1 Modular decomposition

In this technique, the construction of the system is based on assigning functions to components. The design of this system begins with a high-level description of the functions to be implemented. Then, lower-level explanation of how each component will be organized and related to each component is produced.

4.2.2 Event-oriented decomposition

This technique allows the system to be designed base on events that the system must handle. Several events are identified and analyzed during the system analysis. Thus, design is based on the analysis done.

4.3 Overall System Design

The system is designed to simulate various network configurations. It should produce some useful results for the user to analyze and study the network. The simulator should consist of several basic components in a network. These include ATM Switches, Broadband Terminal Equipment (B-TE), Hybrid Fiber Coax (HFC) network, ATM Applications and Physical Link that interconnects Switches and B-TE.

The ATM Applications are logical entities that run on B-TE (hosts). The Applications may be considered as traffic generators that are capable of emulating variable or constant bit rate traffic sources. ATM Applications are connected to each other over a route that uses a selected list of adjacent components to form an end-to-end virtual connection.

4.4 Input Design

Input design is done according to requirement. The input is a text file. The file consists of information about the components, including values of input and output parameters of each component. There should be lists of neighboring components and lists of routes in the file. The input file should ends with a new line. Any information that is not needed for the simulation, which is meant for user's understanding, should starts with a # sign. This is because anything after # will be ignored when the file is read.

Fig. 16 Input design

```
component 'switch1' SWITCH 417 341
param 0 12 0 # Delay to process a cell (uSec): 0
param 155 12 0 # Switching Slot time (Mbit/s): 155
param 10000 12 0 # Output q_size (cells, -1=inf): 10000
param 550 12 0 # High Threshold for Q Congestion Flag: 550
param 450 12 0 # Low Threshold for Q Congestion Flag: 450
param 1 12 0 # Logging every (ticks) (e.g. 1, 100): 1
pflags 2a 4 #Cells Received: 0
pflags 2a 4 #Cell Drop %: 0
pflags 2a 4 #Cells in VBR Q to link1: 0
pflags 2a 4 #Cells dropped in VBR Q to link1: 0
.
.

component 'host2' BTE 562 460
param 'host2' 32 0 # host2
param 50 12 0 # Max Output Queue Size(-1=inf): 50
param 1 12 0 # Logging every (ticks) (e.g. 1, 100): 1
pflags 2b 4 562 424 159 93 #Cells Received: 0
pflags 2a 4 #Cells in VBR Q to link2: 0
.
.
.
neighbor1 'switch1' 'link1'
neighbor1 'switch1' 'link2'
neighbor1 'host1' 'tcp1'
.
.
.
route1 'tcp1' 'host1' 'link1' 'switch1' 'link2' 'host2' 'tcp2'
```

4.5 System Functionality Design

4.3.1 Cell

The generic structure of a cell contains the route number. During switching or routing, an ATM switch reads off the route number found in the cell. The switch then looks up its routing table to forward the cell via the next link to the next switch. If it is at the end of a connection, it will be forwarded to the next B-TE.

The cell structure should conform to the basic ATM cell structure. It will consist of:

- Header information, to recognize an ATM cell on a physical transmission medium
  - ⇨ VPI -Virtual Path Identifier, which is the route number
  - ⇨ PTI -Payload Type Identifier
  - ⇨ CREDIT -For credit based applications

- Cell payload information
  - ⇨ Packet or -for TCP/IP connection
  - ⇨ AAL5 trailer or -for interconnect schemes like LAN emulation and TCP/IP over ATM
  - ⇨ RM -Resource Management cell, for ABR service
  - ⇨ QFC -for QFC (Quantum Flow Control) connection

- ⇨ MPEG - for MPEG application
- ⇨ CREDIT_AMT - for credit based connection

Cells are transmitted from one component to another. When a component wants to transmit a cell, that is, it passes an EV_RECEIVE event to another component. The transmitting component calls ev_enqueue (EV_RECEIVE, src, dest, time, rtn, ce, arg) which has as one of its parameters a pointer to the cell, ce. When the resulting event, after being queued in the event list, gets "fired," the action routine of the destination component is called and the pointer to the cell structure is passed as an argument in that call. The destination action routine executes the portion of the code that describes the behavior of the destination component when it receives a cell. The receiving component must be able to process the event in order to receive the cell.

In convention, all components must dispose of all cells that they receive in one way or another. Thus, a component that receives a cell must either call cell_free() on the cell or send the cell to someone else. There is a module to handle the allocation and deallocation of cells, where cell_alloc() returns a new cell, cell_free() frees a cell, and cell_free_all() frees all cells.

4.3.2 *Route*

The simulator implements static connections. An ATM channel begins and ends with a component of the type ATM APPLICATION. A particular Application can have a route to only one other Application. The routing table at each ATM Switch is updated and information about the next link and next ATM Switch found on the path is stored during the creation of route.

Route is created before the actual simulation starts. This is done by calling the EV_MAKE_ROUTE in each component. The function is defined in each component.c file. There should also be a function to determine where to route a cell next.

4.3.3 *Components*

The simulator is designed to simulate anything that can be modeled by a network of components that send messages to one another.

The basic components in a simulator are:

- ATM Switches
- Broadband Terminal Equipment (B-TE)
- Hybrid Fiber Coax (HFC) network
- ATM Applications
- Physical Link

Each instance of a component has its own data structure. This data structure is used to store information that characterizes the component with some standard information required for every component. The structure of each type of component is defined in the header (.h) file of that type.

The components schedule events for one another to cause things to happen. The model being simulated and the action of the components are determined by the code controlling the components. Every component consists of an action routine and a data structure. Components of the same type share the same action routine, which is called for each event that happens to the component. Component can have its own action routine.

4.3.4 Parameters

Parameter is a data structure that holds information about a component that needs to be displayed on the screen, logged to disk, or saved in a configuration file. The information stored includes pointers to functions to convert the parameter to and from a string, the name of the parameter, and flags describing how to save and/or display the parameter. The Parameter structure is defined in component.h.

The Parameter structure should include:

- Pointer to previous and next parameter so that it can be put in a queue
- Structure describing parameter value

- Structure for storing data
- A variable to keep track of time parameter

There can be more than one parameter for a component, where the parameters will be stored in a doubly linked list pointed to by co_params. The parameters can be displayed by the I/O routines iterating through the list of parameters by using a named variable by the action routine. The actual value of a parameter is stored in a structure at the end of the Param structure. A parameter is initialized by calling param_init() with arguments containing values for various fields in the parameter structure.

4.3.5 Neighbors

Neighbors are stored as a list of Neighbor structures, pointed to from component structures. Each neighbor structure contains a pointer to the neighboring component, a queue in which to store cells, a busy flag, and a pointer to a parameter to display anything that might be associated with the neighbor. The Neighbor structure can be found in component.h.

The structure of neighbor should include:

- Pointer to previous and next data
- Pointer to the neighboring component
- An instance of queue structure for queuing the packets to be sent
- An instance of parameter structure to store parameter information

- An instance of list structure for list of parameters related to vpi number of the different routes

Neighbor must be created and put in a list before a route can be made and before the actual simulation starts. This is done by calling EV_NEIGHBOR in each component. The function is defined in each component.c file. The X_neighbor() function for each component has to determine if the next component can be a neighbor of it. For example, an ATM Application cannot be a neighbor of a Switch.

4.3.6 Action routine

Every component in the simulator (ATM Switches, Broadband Terminal Equipment (B-TE), Hybrid Fiber Coax (HFC), ATM Applications and Physical Link) contains an action routine. This routine is called for each event that happens to a component. The action routine is called to execute a set of commands that will give the component its unique behavior. Components can send any type of events to one another.

The action routine contains a large switch statement with a "case" for every type of event to which the component or the connection type is expected to respond. The events for component creation, routing, and initialization, as well as the basic function of giving the component the ability to pass cells.

4.3.7 *Events*

There are three classes of events: commands, regular events, and private events. Commands and regular events are defined in eventdefs.h. Commands are those events which perform some action such as reset, start and create. Regular events are those directly involved in the actual running of the simulation such as receive, ready and busy. Regular events have to be understood by all components in order to facilitate global communication within the simulator. Private events are events that components send to themselves, therefore they are defined in the source files of the components and not in eventdefs.h.

The event queue is a queue of events kept sorted by time. To fire an event, the first event in the queue is removed, the global time is set to the time of that event, and the action routine pointed to in the event structure is called. Since the simulator is event driven, when the simulation starts, each component is sent a reset command, EV_RESET, followed by a start command, EV_START. All the events have to be queued before they are fired. This is done by the ev_enqueue() defined in event.c.

Command Set (EV_CLASS_CMD)

EV_CREATE

Create a new instance of a component. The comp variable must be NULL, arg points to the name of the new component, and the action routine returns either a pointer to a new data structure or NULL for error. The action routine must allocate the correct amount of memory for the new component's data structure, create its (empty) neighbor list, create the queue of parameters, create any cell queues, etc. This command must also initialize all the private data in the component as necessary. The only information that need not be initialized are any parameters with the InputMask flag set. They will be initialized by the simulator as specified in the Parameters section of this document.

EV_DEL

Delete an instance of component. This command will detach the component from any neighbors it has, free any storage associated with the component, including its data structure, and perform any other necessary clean-up.

EV_RESET — Reset the state of the component and clear out any cell queues, forget about any cells being processed, etc. When the START button of the simulator is hit, EV_RESET is called first for all components and then EV_START.

EV_START — Start operations for example, start a cell generator sending cells. For many components, this will be a no-op.

EV_NEIGHBOR — Attach another component as a new neighbor. The component to be made a neighbor is pointed to by arg. A component should only allow legal neighbors. For example, an ATM Application will not allow an ATM Switch to be attached as a neighbor the ATM Application can only be connected to a B-TE or a HFC.

EV_UNEIGHBOR — Remove the neighbor pointed to by arg from the list of neighbors, and free any memory used to keep track of the neighbor (such as a cell queue

	and the neighbor structure itself). If there is a parameter associated with this neighbor, it must be removed from the queue of parameters and freed. This is a no-op if the component is not a neighbor.
EV_MAKE_ROUTE	This command is a no-op for some components like physical links. ATM Applications and B-TEs use it to store the route number in the VCI field of their component structures. The ATM Switch component creates a local routing table and stores the previous and next component and the VCI number of the route.

Regular Events (EV_CLASS_EVENT)

EV_RECEIVE	Receive a cell event.
EV_READY	Component ready signal.
EV_BUSY	Component busy signal.

Private Events

Private events are events that have only local significance. They are defined within action routine for use by that routine only. Private events are the means by which an action routine can send events to itself.

4.3.8 List and queue

In this simulation module, various lists and queues have to be created throughout the simulation. Since they will be used at vast, there is a need to create a structure for list and queue. The structure should be able to store the current, maximum and minimum length of the list or queue. It should have pointer to the first element and the last element in the list or queue. The list and queue should have functions to:

- create a new and empty list/queue
- add an element to the head/tail of list/queue
- remove an element from the head of list/queue and return it
- delete an element from the list/queue
- search for an element in the list/queue
- free the list (and the elements)
- remove element after the previous element in queue

The lists and queues that will be used in the simulation module includes:

- Component list - list of all the components in the network configuration
- Param_info list - list of parameters in each component
- Neighbor list - list of neighbors, each component (except Application) should have 2 neighbors (used during file reading only)

- Route list - list of components in the route. The components in the list should be neighbor of each other
- Parameter queue - queue of the param_info list
- Event queue - queue of the events to be fired during the simulation
- Cell queue - queue of cells at the component, there will be an input queue and an output queue

4.4.1 Log File

The simulator includes data logging, a function that is used to record the values of a parameter while the simulation is running. When a parameter is to be logged during the simulation, every new value of the parameter with a corresponding time stamp will be saved in a log file. Every entry of the log data will consist of parameter number, time tick and parameter value at that tick. The parameter number will be identified by its name in the file header. The log file will be named sim_log.xxxx where xxxx is the process ID of the simulator.

Fig. 17 Sample sim_log file

```
# 1 'switch3' 'Name'
# 2 'switch2' 'Cells in VBR Q to link22'
# 3 'switch2' 'Cells dropped in VBR Q to link22'
2 3003 1
2 3003 2
2 3043 3
1 3277 switch3 link22 4
2 4095 3
3 4175 1
```

Lines at the head of the file starting with pound sign (#) are a listing of all of the parameters that were marked for data logging when the simulator was running. The number immediately following the # 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.

Lines after the ones marked with # are the actual data recorded during the simulation. The first column is the parameter ID, the second column is the time (in ticks), and the third column is the value of the parameter at that time. A slightly different format is used for the case where the data logged represents cell arrival at a switch or B-TE component. (This is the logging enabled with the box on the component's name line.) In this case the third column is the name of the component on which the data is collected (switch3 in the example). The fourth column is the name of the link from where the cell arrived (link 22), and the fifth column is the route number.

4.4.2 Snap file

The snap file contains information of the network configurations. There are three distinct types of information, which are component descriptions, linkages and route definitions. The file begins with two lines starting with the pound sign (#). The first line records the seed used for that particular simulation run. The second line records the time (in ticks) when the snapshot was taken.

Component descriptions

The group of lines after the two lines is listing of the components and other information. The first line of each group begins with the keyword component, followed by the component's name in single quotes, then the component type in capital letters, and finally the x and y coordinates of the screen position of the component. If component had an open information window when the snapshot was taken, the keyword infowindow appears immediately after the line with the component keyword.

The lines immediately following are a listing of the input parameters, output parameters and their values. Any text on a line after a pound sign (#) is a comment, which identifies the parameter. Each line of input parameters begins with the word param and followed immediately with a number indicating the parameter's value. Following the value are two other numbers. The first number indicates whether the log box is active (0 - 41 = inactive, 42 - 77 = active). The final number on these lines is unused and is always a zero.

Each line of output parameters begins with the keyword pflags. Following the pflags keyword is the number 2 with the letter 'a' or 'b' appended. This is a code that reveals whether the output parameter has its data logging box in the active mode.

2a = the log box is not active

2b = the log box is active

Keyword '6e' is also used to indicate that the parameter is to be logged to the log file.

Linkages

Next group of lines is the linkages. Each line of this group begins with the keyword neighbor1, followed by a component's name in single quotes and either a physical link name or another component name in single quotes. In the example, 'switch1' has two physical links attached, while the B-TE named 'host1' is connected to the ATM Application named 'tcp1'.

Route definitions

The last group of lines is the route listing. Each line begins with the keyword route1, which is followed by the names of all components in the route. Each component name is in quotes. The component list always begins and ends with an ATM Application component.

Fig. 18 Sample snap file

```
# Seed 776093072
# Time of snapshot (ticks) 0
component 'switch1' SWITCH 417 341
infowindowparam 'switch1' 32 0 # switch1
param 0 12 0 # Delay to process a cell (uSec): 0
param 155 12 0 # Switching Slot time (Mbit/s): 155
param 10000 12 0 # Output q_size (cells, -1=inf): 10000
param 550 12 0 # High Threshold for Q Congestion Flag: 550
.
.
.

component 'host2' BTE 562 460
param 'host2' 32 0 # host2
param 50 12 0 # Max Output Queue Size(-1=inf): 50
param 1 12 0 # Logging every (ticks) (e.g. 1, 100): 1
pflags 2b 4 562 424 159 93 #Cells Received: 0
pflags 2a 4 #Cells in VBR Q to link2: 0
.
.
.

neighbor1 'switch1' 'link1'
neighbor1 'switch1' 'link2'
neighbor1 'host1' 'tcp1'
.
.
.
route1 'tcp1' 'host1' 'link1' 'switch1' 'link2' 'host2' 'tcp2'
```

CHAPTER 5

PROGRAM CODING

Chapter 5 ROGRAM CODING

Fig. 19 Cell structure

```
typedef struct Cell{
struct Cell *cell_next;      /* Pointer for use by the queue the cells will
                             be stored */
  VPI vpi;                   /* Route number (virtual path identifier) */
  PTI pti;                   /* Payload type identifier */
  CREDIT cr;
  unsigned int scfq;         /*OE added for fair queueing tag */
  tick_t stamp1;             /* Time stamps are used for data
                             collection */
  int station;               /* Used by the new HFC component */
  int priority;              /* Used by the new HFC component */
  int traffic_class;         /* Used by the new HFC component */
  int size;                  /* Used by the new HFC component */
  struct cell_payload{       /* Structure for the payload portion */
        Packet *tcp_ip_info;
         AAL5_Trailer len;
         RM rm;
         QFC qfc;
         MPEG mpeg;
         CREDIT_AMT cr_amt;
        } u;
```

Fig. 20 Route creation function

```
static caddr_t
cn_route(cn,route_list,vpi)
      register Componentype *cn;
      list *route_list;
   VPI vpi;
{
        /* function of creating route for the ComponenType
}
```

Fig. 21 Component structure

```
typedef struct _Component {
struct _Component *co_next, *co_prev;       /* Links to other
                                components in list */
     short co_class;                  /* Class of component */
     short co_type;                   /* Type of component */
     char co_name[40]                 /* Name to appear on screen */
     PFP co_action                    /* Main function, called with each
event */
     list *co_neighbors;              /* Points to a list of neighbors of
     this component */
          /* Parameters -- data that will be displayed on the screen */
     short co_menu_up;                /* If true, then text window is up */
     queue *co_params;                /* Variable-length queue of
parameters */
          /* Any other info that a component needs to keep will vary */
} Component;
```

Fig. 22 Parameter structure

```
typedef struct _Param {
struct _Param *p_next, *p_prev;      /* So that these can be put in a queue */
     char p_name[40];          /* Name of this parameter for display */
     PFD p_calc_val;           /* Computes a value to be displayed in a meter */
     PFP p_make_text;          /* Makes a string containing the current value */
     PFP p_make_short_text;          /* As above, but only the value, no text */
     PFI p_input;                    /* Routine to input this parameter */
     int p_log;                /* Integer associated with this param for logging */
     double p_scale;           /* Scale to use for meters */
     struct {                        /* Structure to store data in */
          int i;                     /* Commonly used value types */
          int vpi;                   /* Only need to use one of these types */
          double d;
          caddr_t p;
          struct {                   /* Structure describing parameter value */
               caddr_t p;
               int vpi;
               int i;
          } pi;
          tick_t sample;        /* Keeps track of time parameter */
     } u;          /* This structure is used and maintained by the simulator */
} Param
```

Fig. 23 Neighbour structure

```
typedef struct_Neighbor {
struct_Neighbor
    *n_next, *n_prev;              /* Links for the list */
    Component *n_c;                /* Pointer to the neighboring
component */
        /* The next values will vary from network to network, and from
        component to component. For example, only switches and hosts
        have queues in the current application. */
    queue *n_pq;                   /* Queue of packets to be sent */
    short n_busy;                  /* True if neighbor is busy */
    double n_prev_sample;  /* Previous sample time used for utilization
    calculation in links */
    Param *n_p;            /* Index of parameter to display whatever */
    Param *n_pp;           /* Index of parameter to display whatever */
    Param *n_ppp;
    list *n_vpi;                    /* List of parameters related to vpi number
                                    of the different routes */
    caddr_t n_data;        /* If a component wants to store arbitrary
                                    data for each neighbor, put it here. */
} Neighbor;
```

Fig. 24 Create neighbor function (eg. BTE):

```
static caddr_t
b_neighbor(b,c)
    register BTet *b;
    register Component *c;
{
    /* function for determining if the next component can be a neighbor
     and add it as neighbour */
}
```

The b_neighbor() function will eventually call the add_neighbor() function defined in subr.c.

Fig. 25 Function of add_neighbor

```
Neighbor *
add_neighbor(c, neighc, max_num_neighbors, num_classes, va_alist)
    register Component *c;          /* Comp to add neighbor to */
    register Component *neighc;     /* New neighbor */
    int max_num_neighbors;          /* Max number neighbors allowed
                                    0=infinite)*/
    int num_classes;                /* How many classes follow  */
    va_dcl
{
        /* function to add neighbor */
}
```

Fig. 26 Generic action routine and commands for each component

```
caddr_t
action(src, comp, type, cell, vpi, arg)
Component *src;      /* Component that sent this event. Null for cmds. */
Component *comp;     /* Component which this event/cmd applies. */
int type;                    /* Type of event or cmd that is happening. */
Cell *cell;                  /* A cell. */
VPI *vpi;                    /* VPI number of data cell wherever it is
                             applicable. */
caddr_t arg;                 /* Whatever */
{
        /* Usually a large switch statement on the event type */
}
```

Fig. 27 Switch function in action routine

```
switch (type) {

        case EV_RESET:      /* Case for receiving the command EV_RESET*/
                result = cn_reset(cn); /* Call the routine "cn_reset" */
                break;

        case EV_CREATE:     /* Case for receiving the command
                            EV_CREATE */
                result = cn_create((char*)arg);         /* Call the routine
                                                        "cn_create" */
                break;

        case EV_DEL:
                comp_delete((Component *)cn);       /* Delete the componet */
                result = (caddr_t)cn;
                break;          /* case of an event received continues for all */

        case EV_NEIGHBOR:
                result = cn_neighbor(cn, (Component *)arg);
                break;          /* called, the portion of the code that defines */

        case EV_UNEIGHBOR:
                result = cn_uneighbor(b, (Component *)arg);
                break;

        case EV_MAKE_ROUTE:
                result = cn_route(cn, (list *)arg, vpi);
                break;

        case EV_START:
                result = cn_send(cn);
                break;

        case MY_SENDCELL:
                result = cn_send(cn);
                break;

        case EV_RECEIVE:
                result = cn_receive(cn, cell);
                break;
        default:
                break;
} /* end switch statement */
```

Fig. 28 Enqueue events function

```
Event *
ev_enqueue(type, src, dest, time, rtn, ce, vpi, arg)
int type;           /* Type of event -- e.g EV_RECEIVE,EV_CREATE
                    etc */
Component *src;     /* Component which issues this command */
Component *dest;    /* Component on which command applies */
tick_t time;        /* Time at which the event should be scheduled */
PFP rtn;            /* The action routine of the destination component */
Cell *ce;           /* Pointer to a cell*/
VPI vpi;            /* Route number if a cell is passed */
caddr_t arg;        /* Can be anything */
```

Fig. 29 List structure

```
typedef struct list {       /* list header */
l_elt *l_head;              /* first element in list */
l_elt *l_tail;              /* last element in list */
int l_len;                  /* number of elements in queue */
int l_max;                  /* maximum length */
int l_min;                  /* minimum length */
} list;
```

Queue structure will be the same as list structure.

CHAPTER 6

SYSTEM IMPLEMENTATION AND TESTING

Chapter 6 SYSTEM IMPLEMENTATION AND TESTING

6.1 Introduction

The implementation of this system includes programming, testing and debugging of the system. Since the whole project is divided into two main modules, which are the interface module and the simulation module, implementation of this system also includes integration of these two modules.

6.2 Programming

Basically, the codes can be divided into a few categories:

6.2.1 Main

Table 2 Function of main code files

FILE	FUNCTION
Sim.h	⇨ Define things for all simulator routines
Main.c	⇨ Contains the main routine of the simulator, including starting and resetting the simulation
file.c	⇨ Disk I/O routines, including read & write world file (network configuration file)

6.2.2 *Components*

Table 3 Function of component files

FILE	FUNCTION
component.h	⇨ Define the generic structure of component, parameter structure and neighbor structure
Comptypes.h Comptypes.c	⇨ Define the different types of components ⇨ Keep the component types into array to map the strings contained in the world file to component types (constant) and to the action routines used by each component
Every component	⇨ Define own action routine (with private event of that component)

6.2.3 *Program structure and Events*

Table 4 Function of basic files

FILE	FUNCTION
cell.c cell.h	⇨ Routines for allocating and freeing cells ⇨ Define cell structure
eventdefs.h	⇨ Define types of events
event.h event.c	⇨ Define event structure ⇨ Queue the events ⇨ Fire the events ⇨ Return the simulator's clock value
list.h list.c q.h q.c	⇨ Define list/queue structure ⇨ General-purpose list/queue manipulation routines ⇨ Create list/queue, add element, delete element, find element, free storage
log.h	⇨ Describe structures used in the log cluster

log.c	⇨ Contain 'Cluster' to write happenings to a log file ⇨ Contain functions for debugging log and parameter log
route.h route.c	⇨ Create, initialize and destroy the routing table ⇨ Insert, delete or find the route

6.2.4 Miscellaneous

Table 5 Function of miscellaneous files

FILE	FUNCTION
gamma_dist.c	⇨ For Gamma-Based connection, called by gbar.c ⇨ Generate Random Variates in (0,INFINITY) from a gamma distribution with density
hash.h hash.c	⇨ Make and destroy the table ⇨ Insert, delete and find a record
heap.h heap.c	⇨ Create, free a heap ⇨ Insert/delete an element in a heap ⇨ Return (and delete) minimum element of heap
ifft.h ifft.c	⇨ In-place radix 2 decimation in time inverse FFT
mempool.h mempool.c	⇨ Storage manager for large quantities of objects that are going to be allocated and freed many times
random.c	⇨ Random number generation with a special state info interface
subr.c	⇨ Contains random functions that don't belong anywhere else
ssgen.h ssgen.c	⇨ Sscore, self-similar traffic generator

6.3 Integration

6.3.1 Module integration

In the simulation module, components of the system are developed separately. Thus, these components have to be integrated so that message can be sent to each other during the simulation. The basic functions in the project, for example, creating lists and queues of components and events have to be integrated. This is because they are extensively used throughout the simulation. Module integration also includes the integration of data logging function with the component codes.

6.3.2 System integration

The project is into two main modules. System integration of these two modules needs to be done so that a complete simulator is developed. This integration is done after each module is tested to work correctly.

System integration of this project is to integrate the output of interface module with the simulation module. The interface module will generate a savefile (or snapfile) of the network configuration created by the user. This output will contain the information of components and its parameters, neighbors and the routes, which are needed for the simulation module to start simulation. The simulation module cannot be executed successfully without correct output for the interface module. Thus, both the format of creating (interface module) and reading (simulation module) of the file must be integrated.

6.4 Testing

Modules are coded separately for the interface and the simulation part. First, unit testing is carried out. Then, module testing is done for the modules separately. After the modules are integrated, integration testing is done. System testing for the entire network simulation system is carried out at the end of the testing process. Some test cases are used to test whether the input is properly converted to the desired output.

6.4.1 Unit testing

Unit testing is done to find faults in the components of program since it is developed using modular decomposition technique. Process of unit testing is divided into two parts: manually test the codes and test using the compile and debug tool in Microsoft Visual C++ 6.0.

Manually test

During this testing process, the codes are examined by reading through them. The faults of algorithm, data and syntax are checked manually. Some tests like statement testing, branch testing and path testing are done to test the thoroughness of the codes. Statements are read through to check for syntax error. Manual test is done before the codes are compiled and when debugging cannot be done with the debug tool.

Compile using program

Testing with debug tool is more emphasized during unit testing of this project. The codes are compiled to eliminate the remaining syntax faults that are missed during the manual testing. During this testing process, the debugging tool of the compiler is

made full use of. This includes inserting break points to break at certain functions or statements for debugging purpose, using the step into to step into the function, using watch to check the output and variables.

Test cases:

- fprintf statement

  this test case is most widely used in this testing. Basically, there are usage of this statement.

  ⇨ To test whether the function block is entered. For example, putting inside if statement

```
if (!strcmp(l->l_head->le_data, "COMPONENT")) {
        fprintf(stderr, "got here....file.c.....if(!strcmp.. COMPONENT)\n");
/ * function */
}
```

  If the condition of the if statement is correct, the function block will be entered and the message of getting there will be printed out.

  ⇨ To test and check the input or output of a function call. This is to make sure that the correctness of the input or output.

```
c1 = find_comp(clist, le->le_data);
        fprintf(stderr, "got here....file.c..route.....route_c1: %s\n",
    c1->co_name);
```

  In this case, name of component found by the function call will be printed.

- manually assign data
  - ⇨ Instead of using current time as seed, a value is assigned to seed to test if the random number generator is functioning.

6.4.2 Module testing

The testing of the module is carried out together with the system development. Module testing is done to test the integration between tested units to form a module. Testing is also done to test if the simulation produces the desired output.

Test cases:

- manually assigned data
  - ⇨ Enter different values of stop time to check if the statement

    `if (ev_now() > stoptime)`

    can be executed properly to stop the simulation

- condition statement
  - ⇨ Statement of condition is inserted before certain function call to test the conditions needed to execute the function. For example, `if(routel)` is added before this function call,

    `l_addt(route_table,(list *)routel);`

    to make sure that there is a route list to be inserted to the route table. This is done because there is always an access violation error when executing this function call.

- comment out certain function
  - ⇨ Functions call error that cannot be debugged during the module testing is commented out temporarily to test if the module can continue without the function call. For example,

    ```
    l_addt(route_table,(list *)routel);
    ```

  - ⇨ This function is commented out during the module testing because an item can only be placed in a list at a time. Without executing the whole module, there will be an error.

- simple snapfile
  - ⇨ Some basic network configuration files is created to test the format of snapfile needed for this simulation module.

- Faulty input
  - ⇨ Change the sequence of components in route list of the snapfile created. This is to test if the function of route creation is according done to the design. The creation of route should fail is the component is not neighbor of each other.

6.4.3 Integration testing

This testing is done after each module is successfully tested to work properly and correctly. Integration testing tests whether the two separate modules can be integrated into a complete system. The approach used in this testing is the bottom-up integration.

6.4.4 System testing

System testing is carried out to validate that the system is conformed to the requirements. It is also tested to ensure that the integration is correctly and successfully done.

In system testing, some test cases are generated to be implemented on the system. It is designed to provide a thorough test on the use of network simulator. A few people are invited to test out the simulator.

The test cases for system testing are mainly created to test whether the system can simulate different network configurations and produce desired output, which is the correct and useful logged data. Test cases are also used to check if the same output is obtained for certain input.

Test cases:

- different components
  - ⇨ Various components are tested to test whether network configuration with that component can be simulated.

- faulty input
  - ⇨ Wrong file name (input) is given to test how the system responds to the error.
  - ⇨ The input (which is the save file of network configuration) is modified to test whether the simulator can respond to the fault. For example,
    - Change the COMPONENT_TYPE, which is not defined in comptypes

- Add a different component to the neighbor list in the file, which is not a component in the same file.
- Add a different component to the route list in the file, which is not a component in the same file.

6.4.5 Performance testing

Reliability of the system is tested. This test is done to ensure that the developed system is reliable, which means that the system can operate without failure under given conditions for a given time interval. The reliability test is measured over the execution time. During performance testing, test cases that contain bugs or errors are used to test whether the system can handle the exceptions properly.

Test cases:

- exclude stop time
  - ⇨ This is done to test if the system can respond to infinite simulation. The simulation must be stopped after a certain period if stop time is not specified.

CHAPTER 7

SYSTEM EVALUATION AND CONCLUSION

Chapter 7 SYSTEM EVALUATION AND CONCLUSION

7.1 Review Of Goals

The major goal of this project is to develop a PC Based simulator to simulate the ATM networks. The proposed simulator is to provide a ground for studying the queuing delays, throughput and buffer occupancies at various nodes in the network. The simulator should be able to run under Windows platform with a use friendly graphical user interface (GUI). Besides, it should be able to support the Unix platform. It is also proposed that the simulator should have discrete event schedule. Finally, the simulator should have an integrated analysis tool to analyze the output of the simulation.

7.2 System Strengths

There are several strengths in this simulator as compared to other existing simulators.

7.2.1 Support Multi-platform

Most network simulators available currently have to be run under Unix platform. The simulator developed in this project is able to run under Windows NT 4.0, Windows 95 or Windows 98. Since Windows platform is more common among normal users, this encourages more users to use the simulator as a tool to study the network. This is because users do not have to change to Unix platform when they want to use a simulator. However, this simulation module is developed as a Dos-based program. The design of this module not only allows simulation to run under Windows, it can

also be run under Unix environment. Users of NIST/HFC ATM simulator can easily use this simulation module to either simulate their network configurations in Windows or in Unix. This is because the features of this simulation module follow the convention of NIST/HFC ATM simulator.

7.2.2 Designed for ATM Network

As ATM is the trend in networking world, there is a need to study about the ATM network without the expense to develop a real one. This simulator is specially developed to simulate ATM networks. Conventions of the ATM network design are used in this simulator. For example, the ATM cell, the routing protocols and the flow control. Besides, TCP/IP connections can also be simulated using this simulator.

7.2.3 Adding new component or application

New component or application can be easily added to the simulation module. This is because the codes are very modular. The addition requires the user to create new component structure with the common structure that every component must have. The component should have its own component.c and component.h files. Reuse of the existing code is highly recommended when creating the new component. The new component should contain an action routine with various commands that must be performed in the simulation. However, the new component can have its own extra events, which is called private events. An important step when creating new component is to modify the comptypes.c and comptypes.h files. The modification includes:

- Adding a new constant for the new type of component in comptypes.c
- Adding a new entry in the component_types[] array in comptypes.c
- Adding a declaration of the action routine in comptypes.h

7.2.4 User friendly GUI

The simulator is developed with a user friendly Graphical User Interface (GUI) in another module, which is the interface module. Users can create network topologies and configurations with button-click commands.

7.3 Drawbacks And Limitations

7.3.1 Data analysis tool

The simulator is not integrated with an analysis tool to analyze the logged data. For the time being, the logged data in the sim_log file has to be filtered with another module of program. The filtered data can then be plotted using Windows-based GNUPlot. Only filtered data can be analyzed and studied.

7.3.2 Meters

Meter to display information about a parameter during the simulation is unavailable now. Users are unable to view and study the changing value of parameters in a graphical form.

7.3.3 Unable to simulate identical results

This simulator uses a seed (random number generator) to starts the simulation. The seed of the simulator is the current time in ticks. The user is not prompted to input any seed number, thus each simulation may generate different results for the same network configuration.

7.4 Problems Encountered

7.4.1 Integration of project

The main problem encountered during the project development is to integrate the interface module with the simulation module. The integration must be done so as the network configuration created by the user in the interface part can successfully simulated using the simulation module. For integration propose of this project, a savefile (or snapfile of the network configuration) must be produced in the interface module. The format of snap file to be read in the simulation module must conform to the format defined in the simulation module.

7.4.2 Out of virtual memory

The computer may run out of virtual memory if the simulation is carried out for a certain period of time. Thus, the stoptime of the simulation has to be set so that the simulation will not be infinite and it will stop after the specified time.

7.5 Future Enhancements

7.5.1 Integrate with data analysis tool

The simulator can be enhanced by integrating it with a data analysis tool. With the tool, data logged can be filtered automatically and plotted with a program. After integration, users should be able to obtain analyzed data at the end of simulation. The enhancement will make the simulator more useful and hassle free to study the outcome of simulation result.

7.5.2 Display meter during simulation

Meters are useful when the user wants to view the changing data of a parameter during the simulation. The simulator can be enhanced by adding meters to display the information in a graphical form. There are several types of meter for displaying the information. For example, Binary meter, Bar Graph, Time History meter and Histogram meter.

7.5.3 Identical simulation result

Enhancement on this is important to allow user obtain identical simulation results for a certain network configurations. This can be done by allowing the user to set the seed for the simulation.

7.5.4 Parallel simulation

Further enhancement of the simulator can be done to allow parallel simulation. For the time being, only a single network configuration can be simulated using this

simulator. The simulator can be enhanced to let the user simulate a few network configurations at the same time.

7.6 Conclusion

This project has provided a great learning opportunity for ATM network. Basic concept about ATM has to be acquired before the simulator can be developed. A simulator with prime features has been developed according to the design. This simulator is able to simulate various network configurations. User is able to study the performance of the network with logged data after the simulation. However, there are certain constrains that still cannot be solved at the end of the project. Further enhancements can be done in these fields as proposed above.

REFERENCE

REFERENCE

[1] "ATM Development", May. 1995,
http://ece.wpi.edu/courses/ee535/hwk11cd95/bkh/devel.html

[2] L. Fang, "ATM Congestion Control", Survey Paper, Oct. 1995,
http://cclab.chungnam.ac.kr/~ylee/congestion/index.html

[3] William Stallings, "High-Speed Networks TCP/IP ATM Design Principles", Prentice-Hall, Inc., 1998.

[4] Walter J. Goralski, "Introduction to ATM Networking", McGraw-Hill, Inc., 1995.

[5] http://tebbit.eng.umd.edu/Classes/ENEE728/Simulation/

[6] I.Y. Chong, A. Makowski, P. Narayan, J. Kim, T. Christofili, G. Charleston, S. Rao, R. Gupta. "OPNET Simulation Model of the ALAX" Technical Research Report, Dec. 1995,
http://www.isr.umd.edu/TechReports/CSHCN/1995/CSHCN_TR_95-16/CSHCN_TR_95-16.phtml

[7] "OPNET and OMNeT++",
http://www.hit.bme.hu/phd/vargaa/opnet.htm

[8] "OMNeT++",
http://www.hit.bme.hu/phd/vargaa/whatis.htm

[9] Y. Song, D. Cypher, D. Su, "Manual: The NIST ATM PNNI Routing Protocol Simulator (APRoPS), Operation and Programming Guide, Version 1.0," Jan. 1999,
http://www.hsnt.nist.gov/misc/hsnt/

[10] K. Peter, "dlpsim: A Data Link Protocol Simulator", Feb. 1998, http://www.cee.hw.ac.uk/~pjbk/dlpsim/user.html

[11] N. Golmie, F. Mouveaux, L. Hester, Y. Saintillan, A. Koenig, D. Su, "Manual: The NIST ATM/HFC Network Simulator, Operation and Programming Guide, Version 4.0," Dec. 1998.

[12] X.Cai, "The Performance of TCP Over ATM ABR and UBR Services", Jul. 1999, http://www.netlab.ohio-state.edu/~jain/cis788-97/tcp_over_atm/index.htm

[13] Mark Wiman, "Optimizing Buffer Sizes In A Simulated ATM Multicast Network Using A Self-Similar Traffic Generator", 1999, http://www.cs.fsu.edu/~wiman/project/index.html

[14] Windows, Unix and X, http://www.worldserver.pipex.com/home/bill/myweb/writing/pcnetwk/unix.html

[15] Brian Robinson, "Competition between Unix and Windows NT reshapes the technical workstation market", Tech Briefing, Jul. 1998, http://www.fcw.com/pubs/fcw//1998/0706/fcw-feature-7-6-1998.html

[16] Deloitte, Touche, "Technical Workstsations Running Microsoft Windows NT Workstation 37% Less Expensive to Own than Traditional UNIX Offerings", Market Bulletin, Oct. 1997, http://www.microsoft.com/windows/platform/info/nt-unix_tco_study.htm

APPENDIX

APPENDIX A SNAP FILE

```
# Seed 10
# Time of snapshot (ticks) 0

component 'cbr1' CBRCONNECTION 0 0
param 'cbr1' 32 0 # cbr1
param 100 12 0    # Bit Rate (Mbits/s): 100
param 0 12 0      # Start time (usecs): 0
param 50000 12 0  # Number of MBits to be sent: 50000
pflags 22 0

component 'abr1' ABRCONNECTION1 0 0
param 'abr1' 32 0 # abr1
param 500 12 0    # Bit Rate 1 (Mbits/s): 500
param 0 12 0      # Intial Start time (usecs): 0
param 200 12 0    # Bit Rate 2 (Mbits/s): 200
param 0 12 0      # Start time of Bit Rate 2 (usecs): 0
param 100000 12 0 # Number of MBits to be sent: 100000
pflags 22 0

component 'bte1' BTE 0 0
param 'bte1' 32 0 # bte1
param 50 12 0     # Max Output Queue Size(-1=inf): 50
param 100 12 0    # Max Input Queue Size(-1=inf): 100
pflags 2e 4  #Cells Received: 0
param 10 12 0     # Logging every (ticks) (e.g. 1, 100): 10
param 0 12 0      # Stop sending at (usecs): 0
param 149.76 12 0 # Peak Cell Rate of abr1: 149.76
param 32 12 0     # Nrm abr1: 32
param 7.49 12 0   # ICR of abr1: 7.49
param 1.49 12 0   # MCR of abr1: 1.49
param 0.0625 12 0 # RIF of abr1: 0.0625
param 0.0625 12 0 # RDF of abr1: 0.0625
param 32 12 0     # CRM (Cells) of abr1: 32
param 0.0625 12 0 # CDF of abr1: 0.0625
param 2 12 0      # MRM of abr1: 2
param 100000 12 0 # TRM (in us) of abr1: 100000
param 0.00424 12 0     # TCR (in Mbit/s) of abr1: 0.00424
param 500000 12 0 # ADTF (in usecs) of abr1: 500000
pflags 2e 4  #ACR (in Mbit/s) for ABR abr1: 7.49
pflags 2a 4  #Received count (in cells) for ABR abr1: 0
pflags 6e 4  #Cells Dropped ABR Input Q from abr1: 811
pflags 2e 4  #Cells in ABR Input Q from abr1: 100
pflags 2e 4  #Cells in VBR Q to ln1: 0
pflags 2e 4  #Cells dropped in VBR Q toln1: 0
pflags 2e 4  #Cells in ABR Q to ln1: 34
pflags 6e 4  #Cells Dropped in ABR Q to ln1: 0
pflags 2e 4  #Cells in UBR Q to ln1: 0
pflags 2e 4  #Cells Dropped in UBR Q to ln1: 0

component 'bte2' BTE 0 0
param 'bte2' 32 0 # bte2
param 30 12 0     # Max Output Queue Size(-1=inf): 30
param 100 12 0    # Max Input Queue Size(-1=inf): 100
```

```
pflags 2e 4  #Cells Received: 0
param 10 12 0      # Logging every (ticks) (e.g. 1, 100): 10
param 0 12 0       # Stop sending at (usecs): 0
pflags 2e 4  #Cells in VBR Q to ln2: 30
pflags 6e 4  #Cells dropped in VBR Q toln2: 443
pflags 2e 4  #Cells in ABR Q to ln2: 0
pflags 2e 4  #Cells Dropped in ABR Q to ln2: 0
pflags 2e 4  #Cells in UBR Q to ln2: 0
pflags 2e 4  #Cells Dropped in UBR Q to ln2: 0

component 'sw1' SWITCH 0 0
infowindow
param 'sw1' 32 0   # sw1
param 0 12 0       # Delay to process a cell (uSec): 0
param 155 12 0     # Slot time (Mbit/s): 155
param 50 12 0      # Output q_size (cells, -1=inf): 50
param 30 12 0      # High Threshold (Cells): 30
param 5 12 0       # Low Threshold (Cells): 5
param 0 12 0       # buf mgmt: EPD (0), FBA(1), RED(2), FQ(3): 0
param 0 12 0       # EPD Threshold (Cells): 0
param 0 12 0       # MIN threshold for RED: 0
param 0 12 0       # linear scale factor Z for FBA: 0
param 10 12 0      # Logging every (ticks) (e.g. 1, 100): 10
param 2 12 0       # 1(EFCI), 2(NIST), 3(EPRCA), 4(ERICA), 5 : 2
param 0.5 12 0     # MACR Additive Increase Rate (Mbits/s): 0.5
param 142 12 0     # Target Rate (Mbits/s): 142
param 0.9375 12 0  # Explicit Reduction Factor (ERF): 0.9375
param 100 12 0     # Measurement Interval in cells (N): 100
param 0 12 0       # Congestion Tolerance in cells (tau): 0
param 0.875 12 0   # VC Separator: 0.875
param 0.0625 12 0  # Average Factor (AV): 0.0625
param 2000 12 0    # DQT (Cells): 2000
param 0.95 12 0    # Major Reduction Factor (MRF): 0.95
param 0.875 12 0   # Down Pressure Factor (DPF): 0.875
pflags 2a 1  #Congestion (True/False): 0
pflags 2e 4  #% cell drop: 0
pflags 2a 1  #HFC Backward RM Cells (True/False): 0
param 0 12 0       # Max BRM rate (cells/sec): 0
param 0.1 12 0     # Delta [0.05..1]: 0.1
param 1500 12 0    # T0 (in usecs): 1500
param 1.15 12 0    # Factor a: 1.15
param 1 12 0       # Factor b: 1
param 0.5 12 0     # Queue Drain Limit Factor (QDLF): 0.5
param 0.9 12 0     # Decay Factor: 0.9
param 0.8 12 0     # Alpha: 0.8
param 5000 12 0    # Averaging Interval in usecs (AI): 5000
param 500 12 0     # Phantom Tau: 500
param 0.0625 12 0  # Phantom Alpha: 0.0625
param 0.125 12 0   # Beta: 0.125
param 0.75 12 0    # Decreasing Factor: 0.75
param 0.0625 12 0  # h: 0.0625
param 1 12 0       # Utilization_factor: 1
param 0.1 12 0     # Initial Portion: 0.1
pflags 2e 4  #Cells in VBR Q to ln1: 0
pflags 2e 4  #Cells dropped in VBR Q toln1: 0
pflags 2e 4  #Cells in UBR Q to ln1: 0
pflags 2e 4  #Cells dropped in UBR Q toln1: 0
pflags 2e 4  #Cells in ABR Q toln1: 0
```

```
pflags 2e 4  #Cells Dropped in ABR Q toln1: 0
pflags 2e 4  #ERS for Q toln1: 155
pflags 2e 4  #MACR rate for Q toln1: 155
pflags 2e 4  #Phantom MACR rate for Q toln1: 15.5
pflags 2e 4  #Delta for Q toln1: 155
pflags 2e 4  #Alpha Inc for Q toln1: 0
pflags 2e 4  #Alpha Dec for Q toln1: 0
pflags 2e 4  #Fast MACR rate for Q toln1: 15.5
pflags 2e 4  #ABR Traffic for Q toln1: 0
pflags 2a 4  #ABRVBR Traffic for Q toln1: 0
pflags 2a 4  #VBR Traffic for Q toln1: 0
pflags 2a 4  #ERR for Q toln1: 0
pflags 2e 4  #Sigma Neg for Q toln1: 0
pflags 2e 4  #Sigma Pos for Q toln1: 0
pflags 2a 4  #Ratio for Q toln1: 0
pflags 2e 4  #Cells in VBR Q to ln2: 0
pflags 2e 4  #Cells dropped in VBR Q toln2: 0
pflags 2e 4  #Cells in UBR Q to ln2: 0
pflags 2e 4  #Cells dropped in UBR Q toln2: 0
pflags 2e 4  #Cells in ABR Q toln2: 0
pflags 2e 4  #Cells Dropped in ABR Q toln2: 0
pflags 2e 4  #ERS for Q toln2: 155
pflags 2e 4  #MACR rate for Q toln2: 155
pflags 2e 4  #Phantom MACR rate for Q toln2: 15.5
pflags 2e 4  #Delta for Q toln2: 155
pflags 2e 4  #Alpha Inc for Q toln2: 0
pflags 2e 4  #Alpha Dec for Q toln2: 0
pflags 2e 4  #Fast MACR rate for Q toln2: 15.5
pflags 2e 4  #ABR Traffic for Q toln2: 0
pflags 2a 4  #ABRVBR Traffic for Q toln2: 0
pflags 2a 4  #VBR Traffic for Q toln2: 0
pflags 2a 4  #ERR for Q toln2: 0
pflags 2e 4  #Sigma Neg for Q toln2: 0
pflags 2e 4  #Sigma Pos for Q toln2: 0
pflags 2a 4  #Ratio for Q toln2: 0

component 'ln1' LINK 0 0
param 'ln1' 32 0    # ln1
param 155 12 0      # Link Speed (MBits/sec): 155
pflags 2 0
pflags 2 0
pflags 2 0
pflags 2 0
pflags 2 0
pflags 2 0
param 10 12 0       # Distance (Km): 10
pflags 2e 4  #Link rate (Mbit/s) to bte1: 0
pflags 2e 4  #Link rate (Mbit/s) to sw1: 0

component 'ln2' LINK 0 0
param 'ln2' 32 0    # ln2
param 155 12 0      # Link Speed (MBits/sec): 155
pflags 2 0
pflags 2 0
pflags 2 0
pflags 2 0
pflags 2 0
pflags 2 0
```

```
param 8 12 0       # Distance (Km): 8
pflags 2e 4  #Link rate (Mbit/s) to sw1: 0
pflags 2e 4  #Link rate (Mbit/s) to bte2: 0

neighbor1 'cbr1' 'bte2'
neighbor1 'abr1' 'bte1'
neighbor1 'bte1' 'abr1'
neighbor1 'bte1' 'ln1'
neighbor1 'bte2' 'ln2'
neighbor1 'bte2' 'cbr1'
neighbor1 'sw1' 'ln1'
neighbor1 'sw1' 'ln2'
neighbor1 'ln1' 'bte1'
neighbor1 'ln1' 'sw1'
neighbor1 'ln2' 'sw1'
neighbor1 'ln2' 'bte2'

route1 'abr1' 'bte1' 'ln1' 'sw1' 'ln2' 'bte2' 'cbr1'
```

APPENDIX B LOG FILE

```
# 1 'bte1' 'Cells Dropped ABR Input Q from abr1'
# 2 'bte1' 'Cells Dropped in ABR Q to ln1'
# 3 'bte2' 'Cells dropped in VBR Q toln2'
3 12720 1
3 13144 2
3 13568 3
3 13992 4
3 14416 5
3 14840 6
3 15264 7
3 15688 8
3 16112 9
3 16536 10
3 16960 11
3 17384 12
3 17808 13
1 21836 1
3 22048 23
1 22048 2
1 22260 3
3 22472 24
1 22472 4
3 22896 25
1 22896 5
1 23108 6
3 23320 26
1 23320 7
1 23532 8
1 23744 9
3 23744 27
1 23956 10
3 24168 28
1 24168 11
1 24380 12
1 24592 13
3 24592 29
1 24804 14
3 25016 30
1 25016 15
1 25228 16
```

GLOSSARY

GLOSSARY

AAL

ATM Adapation Layer: The standards layer that allows multiple applications to have data converted to and from the ATM cell. A protocol used that translates higher layer services into the size and format of an ATM cell.

ABR

Available Bit Rate: 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 which 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. Cell delay variation is not controlled in this service, although admitted cells are not delayed unnecessarily.

ATM

Asynchronous Transfer Mode: A transfer mode in which the information is organized into cells. It is asynchronous in the sense that the recurrence of cells containing information from an individual user is not necessarily periodic.

BTE

Broadband Terminal Equipment: An equipment category for B-ISDN, which includes terminal adapters and terminals.

CBR

Constant Bit Rate: An ATM service category which supports a constant or guaranteed rate to transport services such as video or voice as well as circuit emulation which requires rigorous timing control and performance parameters.

Cell

A unit of transmission in ATM. A fixed-size frame consisting of a 5-octet header and a 48-octet payload.

CRC

Cyclic Redundancy Check: A mathematical algorithm that computes a numerical value based on the bits in a block of data. This number is transmitted with the data and the receiver uses this information and the same algorithm to insure the accurate delivery of data by comparing the results of algorithm and the number received. If a mismatch occurs, an error in transmission is presumed.

GFC

Generic Flow Control: GFC is a field in the ATM header which can be used to provide local functions (e.g., flow control). It has local significance only and the value encoded in the field is not carried end-to-end.

Link

An entity that defines a topological relationship (including available transport capacity) between two nodes in different subnetworks. Multiple links may exist between a pair of subnetworks. Synonymous with logical link

MPOA

Multiprotocol over ATM: An effort taking place in the ATM Forum to standardize protocols for the purpose of running multiple network layer protocols over ATM

Physical Link

A real link which attaches two switching systems

RM

Resource Management cell, used in ABR service

QoS

Quality of Service: Quality of Service is defined on an end-to-end basis in terms of the following attributes of the end-to-end ATM connection: - Cell Loss Ratio, Cell Transfer Delay, and Cell Delay Variation

UBR

Unspecified Bit Rate: UBR is an ATM service category, which does not specify traffic related service guarantees. Specifically, UBR does not include the notion of a per-connection negotiated bandwidth. No numerical commitments are made with respect to the cell loss ratio experienced by a UBR connection, or as to the cell transfer delay experienced by cells on the connection.

VBR

Variable Bit Rate: An ATM Forum defined service category which supports variable bit rate data traffic with average and peak traffic parameters.

VCI

Virtual Channel Identifier: A unique numerical tag as defined by a 16 bit field in the ATM cell header that identifies a virtual channel, over which the cell is to travel.

VPI

Virtual Path Identifier: An eight-bit field in the ATM cell header, which indicates the virtual path over which the cell should be routed.