SIMULATION OF ASYNCHRONOUS TRANSFER MODE (ATM) NETWORKS FOR MULTIRATE ATM TRAFFIC APPLICATIONS

A DISSERTATION

Submitted in partial fulfilment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

NEELESH YADAV

ER & DCI
NOIDA

IIT Roorkee-ER&DCI, Noida
C-56/1, "Anusandhan Bhawan"
Sector 62, Noida-201 307

FEBRUARY, 2003

621.380285
YAD

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this dissertation " **SIMULATION OF ASYNCHRONOUS TRANSFER MODE (ATM) NETWORKS FOR MULTIRATE ATM TRAFFIC APPLICATIONS** ", in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Information Technology**, submitted in **IIT Roorkee–ER&DCI Campus, NOIDA,** is an authentic record of my own work carried out during the period from August, 2002 to February, 2003 under the guidance of **Dr. Moinuddin**, Professor, Electrical Engineering Dept. at Faculty of Engineering, Jamia Millia Islamia University–New Delhi.

I have not submitted the matter embodied in this dissertation for award of any other degree of diploma.

Date: 24-02-2003
Place: ...NOIDA..

[illegible]
(Neelesh Yadav)

CERTIFICATAE

This is to certify that the above statements made by the candidate are correct to the best of my knowledge and belief.

Co-Guide [illegible] 24-02-03
(Mr. Munish Kumar)
Project Engineer,
ER&DCI-NOIDA

Guide [illegible]
(Dr. Moinuddin)
Professor, Elect. Engg. Dept.,
Faculty of Engg. & Tech.,
Jamia Millia Islamia,
New Delhi-110 025

ACKNOWLEDGEMENT

It gives me great pleasure to take this opportunity to thank and express my deep sense of gratitude to Prof. Prem Vrat, Director, IIT Roorkee, Shri R.K.Verma, Executive Director, ER&DCI, Noida for providing me with excellent acedemic environment to undergo my dissertation. I would like to extend my thanks to Prof. A.K.Awasthi, Dean PGS&R and Prof. R.P.Agarwal, Course Coordinator M.Tech.(IT), IIT Roorkee for providing all the required facilities for my dissertation.

The work presented in this report would not have been completed without the excellent guidance, flexibility and consistent support given by Mr. V.N. Shukla, Course coordinator M.Tech. (IT) program.

At the submission of this dissertation work, I take the opportunity to express my deep sense of gratitude and indebtedness to my guide, Prof. Moinuddin, Electrical Engg. Department, Faculty of Engg., Jamia Millia Islamia University, New Delhi for his invaluable, tireless guidance and constructive criticism throughout this dissertation. It is only due to his constant motivation and moral support, I was able to pull through many difficult phases of the work and bring it to a successful completion.

I am also grateful to Dr. P.R. Gupta, Reader and my Co-guide Mr. Munish Kumar, Project Engineer, ER&DCI, Noida for the cooperation extended by them in the successful completion of this report.

I would like to thank all my friends and well wishers, who helped me directly or indirectly in the process and contributed towards this dissertation work.

Finally, I express my regards to my parents and my elder brother, who have been a constant source of inspiration to me and provided me a perfect environment for my studies and supported me throughout.

[illegible]

(Neelesh Yadav)

Enrolment No. - 019028

CONTENTS

ABSTRACT

The introduction of the Asynchronous Transfer Mode (ATM) is currently being propelled by the need for fast data communication in public and private networks. ATM is the switching and multiplexing technique employed for ISDN. In this dissertation a simulation tool is developed for Asynchronous Transfer Mode (ATM) networks to analyze the behavior of ATM networks. The developed tool is a GUI based that gives the user an interactive modeling environment. With this tool the user may create different network topologies, control component parameters, measure network activity, and log data from simulation runs. In this simulator we include some components which are necessary to formulate an ATM network like an ATM switch, a Broadband Terminal Equipment (BTE), (may be an ISDN node like Host computer, Workstation etc.), Some ATM traffic Applications like CBR (Constant Bit Rate), VBR (Variable Bit Rate), ABR (Available Bit Rate) and TCP/IP based traffic etc., and Physical connection link to connect various components like switch to BTE, switch to switch etc., this may be Optical Fiber, Coaxial Cable or Copper wired.

This dissertation aims to study of the various kinds of network configurations with ATM as a backbone and the results of various parameters of network components like throughput, delay, link utilization at various network configuration, size of various ATM application queues, and some congestion related parameters also.

Chapter 1

INTRODUCTION

1.1 Overview

In recent years we can watch a great deployment of computer networks, that are widely used to transmit various kinds of information in many corporations, institutions, organizations. Users and customers require faster, more robust and more reliable services. Therefore network planners and engineers work hardly on reducing network construction cost, while providing an acceptable level of survivability. Loss of services and traffic in high-speed fiber systems due to failures of network elements could cause a lot of damages and significant revenue loss. The increase of bandwidth in optical transmission means that even a failure of a single link will impact many services [16]. Restoration methods providing survivability has to be self-healing. Self-healing means, that the network has the ability to reconfigure itself around failures such, that as little as possible of traffic is lost [6,16].

ATM is one of the most promising networking technologies, that provides high performance, the ability to carry many types of services (data, voice, video), the ability to carry traffic over all kinds of networks, and Quality of Service (QoS) guarantees, which facilitate new classes of applications such as multimedia.

1.2 Objective

The objective of this dissertation is to design and develop an Asynchronous Transfer Mode network simulator to provide a means for researchers and network planners to analyze the behavior of ATM and other networks without the expense of building a real network. This ATM simulator gives the user an interactive modeling environment with a graphical user interface. It helps the user to create different network topologies, control component parameters, measure network activity, and log data from simulation runs.

The system developed in this dissertation is capable to simulate the behavior of an ATM network under the presence of excessive load to the network through its all nodes, each representing an autonomous switch with specific arrival random interval intensity. A switch takes time to manage a call request in order to send it to the appropriate neighbor node. This time delay is insignificant if the network link has simulated distance in terms of time required for a call to pass through the link [5,6,11].

1.2 Scope

ATM network simulator is a tool to analyze the behavior of ATM networks without the expense of building a real network. There are two major uses for the simulator: as a tool for ATM network planning and as a tool for ATM protocol performance analysis. As a planning tool, we can run this simulator with various network configurations and traffic loads to obtain data/parameters such as channel utilization and throughput. It can diagnose about congestion control parameters with channel utilization of network. Response are displayed directly on the screen or logged in a data file for further processing.

As a protocol analysis tool, a researcher or protocol designer could study the total system effect of a particular protocol. For example, one could investigate the effectiveness of various flow control mechanisms for ATM networks and addresses such issues as: mechanisms for fair bandwidth allocation protocol overhead, bandwidth utilization, etc. In this simulation the tool has designed in such a way that modules simulating components of an ATM network can be easily changed, added, or deleted. Through this simulation the activities can be recorded on a cell by cell basis for subsequent analysis.

1.4 Organization of Dissertation

Including this introductory chapter, which gives a brief description about the objective and scope of this dissertation work, the dissertation report is organized as follows:
The second chapter discusses about the literature survey of traffic management of ATM network and congestion control in general. In this chapter we also discuss the various ATM service class like VBR (Variable Bit Rate), CBR (Constant Bit Rate), ABR (Available Bit Rate) etc.

The third chapter discusses about the concept of simulation in which we explain the flow of simulation with necessary flow chart.

The fourth chapter explains the various implementation strategies of simulation. In this chapter the operational feature and data structure used in various components in the simulation software explained in detailed.

The fifth chapter discusses the various results of simulation with some screenshots and some graphs of significant parameters of network performance.

The sixth and last chapter discusses the conclusion of this dissertation with future scope of this simulation.

Chapter 2

LITERATURE SURVEY OF ATM NETWORKS

Due to the increased demand for communication services for voice, data, video and multimedia application Broadband Integrated Services Digital Networks (B-ISDN) have received increased attention in the past few years. It is because ITU, formerly CCITT, modestly defines B-ISDN as a service requiring transmission channels capable of supporting rates greater than the primary rate (1.544 Mbps) of Integrated Services Digital Network.

2.1 Broadband Application

B-ISDN is conceived as an all-purpose digital network. This means B-ISDN is required to support traffic requiring bandwidth ranging from a few Kbps to several hundred Mbps second. Some traffic, such as interactive data and video, are highly bursty while some traffic, such as file transfer, is rather continuous. Besides, B-ISDN should support services with both constant and variable bit rates, as well as connection-oriented and connectionless transfers. It is also required to meet diverse services and performance requirements of multimedia traffic. For instance, real-time voice requires real-time delivery through the network, but the loss of small amounts of voice information is tolerable. On the other hand, high throughput and strict error control are of primary requirements in many data applications, while reasonable amount of delay is acceptable. Some services, such as real-time video communications, require error-free transmission as well as rapid transfer. Further more, B-ISDN should not only support multipurpose applications that we know of, but should also provide the framework to support future applications that we do not fully understand, or even know of, today. For example, some of the expected future services include High-Definition Television and video/document retrieval services [12].

In order to achieve the successful deployment of B-ISDN, existing technologies, such as circuit switching and packet switching, cannot be used. It is because in circuit switching, the bandwidth consists of fixed-size channels and the channel size cannot be changed easily. As B-ISDN is required to support a wide variety of traffic with different speed, the scheduling of time slots becomes intractable. In addition, the use of circuit switching results in the inefficient utilization of network resources because the entire end-to-end circuit remains dedicated to the user throughout the connection time, even the user does not use it all the time. On the other hand, packet switching employs complex procedures like window flow control and recovery of errored or lost packets. These complex procedures usually implemented by using software, makes high-speed communication difficult [14].

2.1.1 Asynchronous Transfer Mode (ATM)

Asynchronous Transfer Mode (ATM) has been proposed [CCITT-I.121] as a target technology for overcoming the difficulties in the existing methods. It is a particular packet-oriented and connection oriented transfer mode that uses asynchronous time division multiplexing techniques.

2.1.1.1 What is ATM

ATM stands for "Asynchronous Transfer Mode". ITU-T[ITUT I.113] defines ATM as : "A transfer mode in which the information is organized into cells; it is asynchronous in the sense that the recurrence of cells containing information is not periodic." The above definition encompasses three basic terms, viz. transfer mode; cell-based and asynchronous transfer [16].

a) Transfer Mode

Transfer mode refers to the techniques used to transmit, switch and multiplex information. In other words, transfer mode is the means of packaging, sending and receiving information on the network. In current literature, Asynchronous Transfer Mode (ATM) and Synchronous Transfer Mode (STM) are the only two possible transfer modes. For others, circuit switching and packet switching are the two basic transfer modes. In circuit switching information is sent as bit streams, while in packet switching, information is sent as large frames. ATM fits in between these two extremes because it uses a small-sized frame (53 bytes). By using a small size frame (precisely, a cell!), ATM retains the speed of circuit switching while still offering the flexibility of packet switching. This is why ATM is also referred to as fast packet switching technology [16].

b) Cell-based Transfer

Information in ATM is "organized into cells", which means that lowest unit of information in ATM is a cell. A cell is a fixed size of frame of 53 bytes, with 5 bytes of header and 48 bytes of payload. The header carries information require to switch cells, while payload contains the actual information to be exchanged [16].

c) Asynchronous Transfer

ATM is an Asynchronous Transfer Mode. There is considerable confusion regarding the term "asynchronous". Usually, the terms synchronous and asynchronous refer to the way data is transmitted. In synchronous mode, the transmitter and receiver clocks are synchronized and frames are sent/received periodically. In asynchronous mode, timing information is derived from the data itself, and that transmitter is not compelled to send data periodically.

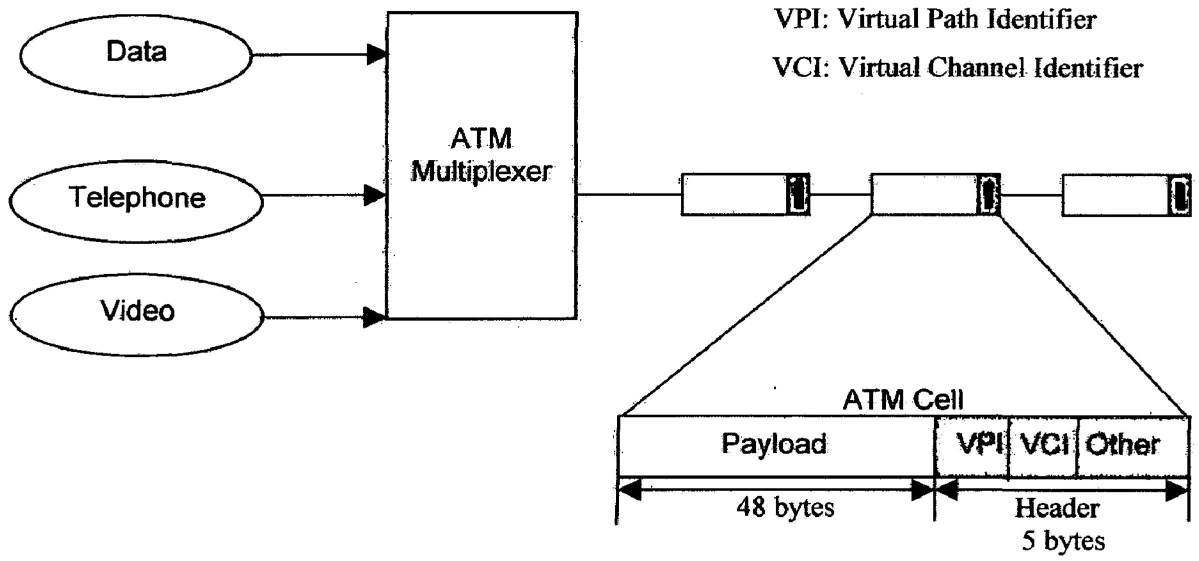


Fig. 2.1 Cell-based Transfer

2.1.2 Basic Principles of ATM

- Information is sent in short fixed-length blocks called cells. Transmitting the necessary number of cells per unit time provides the flexibility needed to support variable transmission rates. Fixed-length cells can help simplify the processing overhead at network switches. Using short cells can reduce overhead and delay in packetization.

- ATM is a connection-oriented technique based on virtual circuit, in which a path has to be established between the users before information can be exchanged. This is done by the connection set-up procedure at the start and by a clear-down procedure at the end. The order in which cells arrive is guaranteed to be the same as the order in which they are transmitted. Each connection has the assigned capacity, allocated according to the user's request, subject to available capacity. The cell is switched according to the label in the ATM header, called a virtual path identifier/virtual channel identifier (VPI/VCI). These identifiers denote the routing address and are used in multiplexing. At an ATM switch, the VPI/VCI value of an input is translated into another set of VPI/VCI before the cell is transmitted to the down stream node.

- The information field is transported transparently by the ATM network. Flow control and error recoveries are performed on an end-to-end basis. Only header error control in the cell header by cyclic redundancy check code is done inside the network.

- Unlike traditional networks, ATM must provide guarantee to meet the user's Quality of Service (QoS) in terms of cell loss ratio (CLR), cell transfer delay (CTD) or/and cell delay variance (CDV).

ATM has the advantages of both circuit switching and packet switching. Firstly, the advantage of self-routing switch in circuit switching allows high-speed communication after connection setup. The simplification of functions within the core network reduces the number of software-driven procedures. This overcomes the disadvantage of packet switching but still allows an arbitrary amount of bandwidth to be allocated to a connection. Network resources are consumed only when cells are generated (i.e., when there is information to be transferred). Therefore, network resources can be used efficiently. Owing to the flexibility mentioned above, ATM has been accepted by ITU as the integrated access method for B-ISDN. Additional bandwidth efficiency enhancement can be achieved by proper buffering and statistically multiplexing at the cost of introducing cell loss and cell delay [16].

2.1.3 ATM Cells Format

The ATM cell consists of a 5-octet header and a 48-octet information field immediately following the header. Two cell formats, one for the user-network interface (UNI) and the other for the network-network interface (NNI), have been proposed. The UNI format is used between the user installation and the first ATM exchange as well as within the users' owns network. The NNI format is used between the ATM exchanges in the ATM network.

Cell (53 Byte)						
Header Information						
GFC	VCI/VPI Field	Payload Type Identifier	RES	Cell Loss Priority	Header Checksum	Payload
4 bits	24 bits	2 bits	1 bit	1 bit	8 bits	48 bytes

Fig. 2.2-ATM cell format

- Generic Flow Control (GFC) field contains 4 bits at the UNI and can provide flow control information towards the network.

- 24 bits of routing field (VPI/VCI) are available for routing at the UNI and 28 bits at the NNI.
- 3 bits of payload type (PT) field are used to provide an indication of whether the cell payload contains user information or network information. According to [CCITT-I.361], for example, PT = 000 represents user data cell without experiencing congestion; while PT = 110 represents resource management cell.
- Cell loss priority (CLP) field may be set by the user or service provider to indicate lower-priority cells. Cells with the CLP bit set are at risk of being discarded, during congestion.
- Header error control (HEC) field consists of 8 bits and is processed by the physical layer to detect errors in the header. The error control covers the entire cell header. The code used for this function is capable of either single-bit error- correction, or multiple-bit error-detection.
- Payload field contains 48 bytes of user information. This may include additional overhead bits added by the higher layers.

2.1.4 ATM protocol reference model

The ATM protocol reference model is based on standards developed by the International Telecommunication Union (ITU). The protocol reference model for ATM is divided into three layers: the physical layers, the ATM layer and the ATM adaptation layer (AAL). The ATM layered network architecture is as shown in following figure.

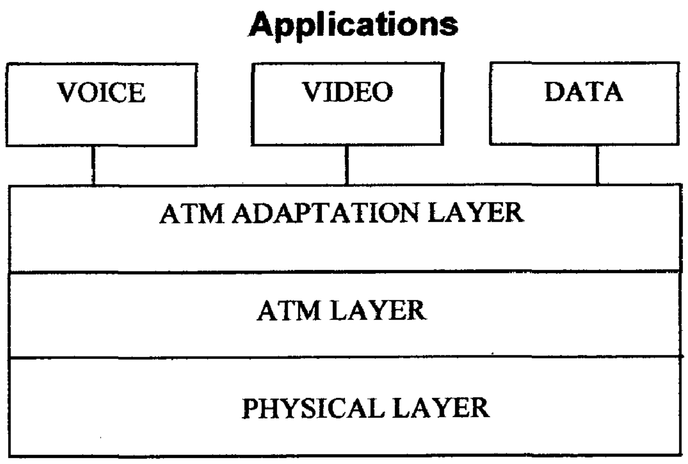


Fig. 2.3 Simple Layered Architecture of ATM network

Physical Layer: This layer defines a transport method for ATM cells between two ATM entities. It has a medium-dependent sublayer (responsible for the correct transmission and reception of bits on the physical medium) and a transmission convergence sublayer (responsible for the mapping of the ATM cell to the transmission system used).

ATM Layer: The ATM layer is a unique layer that carries all the different classes of services supported by B-ISDN within a 53-byte cell. It mainly performs switching and multiplexing functions.

ATM Adaptation Layer: The purpose of the ATM adaptation layer is to provide a link between the services required by higher network layers and the generic ATM cells used by the ATM layer. The AAL consists of a sublayer that provides cell segmentation and reassembly to interface to the ATM layer and also a more service specific convergence function to interface to the bearer services being carried.

ATM networks are assumed to be capable of being flexible and doing efficient allocation of bandwidth to a wide variety of traffic classes. The description of various traffic classes (means ATM applications in my simulation strategy) is as follows;

2.2 Overview of ATM Service or Traffic Classes

As mentioned previously, ATM is a networking protocol with the potential to support applications with distinct tolerances for delay, jitter, and cell loss and distinct requirements for bandwidth or throughput. To address these diversities of needs, the ATM Forum has defined a family of service categories.

2.2.1Constant Bit Rate (CBR)

The CBR service class is intended for real-time applications, i.e., those requiring tightly constrained delay and delay variation. This would be appropriate for voice and video applications. With this type of service, the ATM network contracts to deliver the requested bandwidth with low transfer delay, low delay variation and low cell loss.The consistent availability of a fixed quantity of bandwidth is considered appropriate for CBR service. In establishment of a CBR connection, the user has to specify its required CLR, PCR, CDV and PCR. Note that CLR may not be specified for cells with CLP = 1 [16].

2.2.2 Real-time Variable Bit Rate (VBR)

The real time VBR service class is intended for real-time applications, i.e., those requiring tightly constrained delay and delay variation, as would be appropriate for voice and video applications. Sources are expected to transmit at a rate, which varies with time. Equivalent the source can be described as bursty. Real time VBR service may support statistical multiplexing of real-time sources, or may provide a consistently guaranteed QoS. To establish a connection for real-time VBR traffic, the user has to specify the required CLR, PCR, CTD, CDV, SCR and BT [15,16].

2.2.3 Non-real-time Variable Bit Rate (VBR)

This service class is intended for non-real time applications, which have bursty traffic characteristics. The application expects a bound on CTD for all the cells. Non-real time VBR service supports statistical multiplexing of connections. Similar to real-time VBR, non-real time VBR requires PCR, CLR, CTD, SCR and BT. Note that CDV is unspecified and CLR is only specified for CLP = 1 cells.

2.2.4 Unspecified Bit Rate (UBR)

UBR is a best effort service intended for delay-tolerant or non-real time applications, i.e., those which do not require tightly constrained delay and delay variation, ATM attributes required in ABR connections PCR (Peak Cell Rate), MCR (Minimum Cell Rate), CLR (Cell Loss Ratio) such as traditional computer communications applications. Sources are expected to transmit non-continuous bursts of cells. Because of this nature, UBR service can support a high degree of statistical multiplexing among different sources. For each UBR virtual circuit, PCR is specified, and while the network makes its best effort to deliver traffic under that peak rate, no assurances are given as to delay or reliability [16].

2.2.5 Available Bit Rate (ABR)

The ABR traffic is used to fill in the bandwidth slack left by the scheduled traffic that has guaranteed bandwidth and latency. The ABR service thus eliminates the need of contract negotiation between the connectionless traffic sources and the network. Certain applications are willing to live with the unreserved bandwidth. Therefore, deterministic traffic parameters are not necessary. These applications can reduce their rates of transmission when the network is congested and asks them to do so. When the congestion is over, those applications can increase their information transfer rates because there is extra bandwidth available within the network. To support traffic from such sources in an ATM network will require facilities different from those for peak cell rate or sustainable cell rate traffic . ABR is an ATM Layer service. Generally, the resources available for ABR connections may change subsequent to connection establishment. It is expected that a user that adapts its traffic rate according to the control information will experience a low cell loss ratio. Cell Delay Variation is not controlled in this service. Therefore, ABR service is not intended to support real-time applications. Besides, ABR traffic should have a rapid access to unused network bandwidth at up to PCR, whenever the network bandwidth is available. PCR is the maximum cell rate that a source can transmit and it is negotiated during call establishment. MCR is a rate negotiated between the source and the network(s) such that the actual cell rate of the ABR source can never be less than MCR. However, there is no obligation that the sources must transmit at a rate is made as to the cell loss ratio experienced by a connection, or as to the cell transfer delay experienced by cells in the connection. Therefore, UBR is only suitable for those applications which do not require tightly constrained delay and delay variation. On the other hand, ABR can be used for any application for which the user wants a more reliable service and critical data transfer. The ABR service is inherently closed loop. The source performs traffic dynamic shaping based on the feedback it receives from the network. The network using dynamic rate enforcement may enforce this behavior. A MCR is negotiated with the network. If the MCR is non-zero, then it is assumed that resources are reserved in network nodes to ensure that the feedback never causes the available cell rate to fall below the MCR, and that the CLR guarantee is met of at least MCR. CLR is minimized for sources that adjust cell flow in response to control information. Note that the delay is not specified. The objective is that the network should not excessively delay the admitted cells. On the establishment of an ABR connection, the user has to specify to the network both a maximum required bandwidth and a minimum usable bandwidth. The minimum value it the user requested may be specified as zero. The bandwidth available from the network may vary, but shall not become less than MCR. Notice that ABR is different from the UBR. The UBR service class is also intended for delay-tolerant or non-real-time applications. However, UBR service class offers no traffic related service guarantees. Cells in the connection make as to the cell loss ratio experienced by a connection, or as to the cell transfer delay no numeric commitment experienced. Therefore, UBR is only suitable for those applications, which do not require tightly constrained delay and delay variation. On the other hand, ABR can be used for any application for which the user wants a more reliable service and critical data transfer [11,12,13,16].

2.3 Traffic Management and ATM Networks [11,16]

Traffic management in communication networks deals with the controlled use of network resources to prevent the network from becoming a bottleneck. In particular, when network resources are allocated more connections/traffic than they can effectively support, network performance for users degrades. Therefore it is necessary to allocate and control the traffic so that the network can operate at acceptable levels even at times when the offered load to the network exceeds its capacity.

In circuit-switched networks, each connection is allocated a fixed amount of bandwidth and a constant data rate in the network is provided to communicating entities throughout the duration of the connection. A simple call admission procedure is sufficient to control congestion in circuit-switched networks since the dedicated bandwidth is always available for a connection and there is no contention for resources once a channel is allocated.

On the other hand , traffic control is much more complex task in packet-switched networks. In these networks, in simple terms there is a queue associated with each link at every switching node in the network. As the arrival rate at a link approaches its transmission rate, buffers start to overflow (i.e. packets that arrive at a time the buffer is full are discarded). Dropped packets are eventually retransmitted by an upstream node causing the traffic to further increase. As the number of retransmission increases, more nodes become congested and more packets are dropped. Eventually, the network can reach a catastrophic state in which most of the packets in the network are retransmission.

In general, two types of traffic control are used in packet switched networks, namely, flow control and congestion control. Flow control is concerned with the regulation of the rate the sender transmits packets in order to control the rate at which the destination receives data, so that it is not overwhelmed. Congestion control on the other hand is the control of the network traffic in global view of the network. Fundamentally, congestion occurs when the users of the network collectively demand more resources than the network has to offer.

2.4 Components Description

The following are brief descriptions of the major building blocks of the simulated network.

2.4.1 Switch

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

2.4.2 Broadband Terminal Equipment (BTE)

This is a component to simulate a broadband ISDN node, e.g., host computer, workstation, etc. A BTE component has one or more ATM Applications on one side and a physical link on different standard rates. The user also specifies the length of the link. The output parameter reported by the simulator is link utilization in terms of the other side. Cells received from the Application side are forwarded to the physical link; if the link is busy the cells go into a queue. The user can specify the maximum output queue size. The parameters that can be monitored are the number of cells in an output queue and the number of cells dropped.

2.4.3 ATM Application

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

2.4.4 Physical Link

This component simulates the physical medium (Twisted Pair, Coaxial Cable, Optical Fiber or Wireless) on which cells are transmitted. The user may choose the link speed from a list of several bit rate (Mbits/s). The Distance is important factor for selecting the particular physical link. So distance is one of the parameters for physical link creation.

The data rate of a particular link may be different so we have to define the different data rates of various links. Some standard carrier links (those can be used in ATM network) speed as follows [11,15,16].

LINK NAME	**DATA TRANSMISSION SPEED/RATE**
1.STS-1	51.840 Mbps
2.STS-3C	155.520 Mbps
3.STS-12C	622.080 Mbps
4.STS-24C	1244.160 Mbps
5.DS-3	44.736 Mbps

SIMULATION CONCEPT

In order to simulate a system, it is needed to study the system in question and to set an appropriate model of it. There are a variety of methods and mechanisms to classify a system in order to create a model of it in an appropriate way. This chapter will describe why we have chosen an event driven simulation and how it works.

3.1 Simulation Model

In this ATM network simulation we are using five main components, those perform and create the environment of an ATM network. Here we are giving brief of every component's concept regarding its work in the simulator. At first we are explaining the concept of simulation model with necessary flow diagram.

3.1.1 Discrete Event Simulation

A discrete system is one for which the state variables change instantaneously at separated points of time. A continuous system is one for which the state variables change continuously with respect to time. So a network system is obviously a discrete system, the incoming calls variates at separated point of time.

At separated points in time arrive a number of calls to the system. These points in time are the ones at which an *Event* occurs, where an event is defined as an instantaneous occurrence that may change the state of the system. But an event might be used to schedule an other event or the end of a simulation run at a particular time or to schedule a decision about the system's operation at a particular time and might not actually result in a change in the state of the system. This is why an event might change the state of the system [9,17].

3.1.2 Time-Advance Mechanism

Because of the dynamic nature of discrete-event simulation models, we must keep track of the current value of simulated time as the simulation proceeds, and we need also a mechanism to advance simulated time from one value to another. We call the variable in a simulation model that gives the current value of simulated time the simulation clock. The unit of time for the simulation clock is never stated explicitly for a given model, it's assumed to be in the same units as the input parameters. That is as we set a dimension for the network each incoming call have a given time unit when it occurs, how long time it will takes in the network system, etc. Also, a relationship between simulated time and the time needed to run a simulation on the computer does not exist [9,17].

3.1.3 Components and Organization of a Discrete-Event Simulation Model

There are a number of components and a logical organization for these components that are common to a typical discrete-event simulation which uses the next-event time-advance approach:

System state: The collection of state variables necessary to describe the system at a particular time.

Simulation clock: A variable giving the current value of simulated time.

Event list: A list ordered according to event time stamps i.e. when each type of event will occur.

Initialization routine: A subprogram to initialize the simulation model at time zero.

Timing routine: A subprogram that determines the next event from the event list and then advances the simulation clock to the time when that event is to occur.

Event routine: A subprogram that updates the system state when a event of particular type occurs (there is one event routine for each event type).

Library routines: A set of subprograms used to generate random observations from probability distributions that were determined as part of the simulation model.

Report generator: A subprogram that computes estimates of the desired measures of performance and produces a report when the simulation ends.

Main program: A subprogram that invokes the timing routine to determine the next event and then transfers control to the corresponding event routine to update the system state appropriately. The main program may also check for termination and invoke the report generator when the simulation is over.

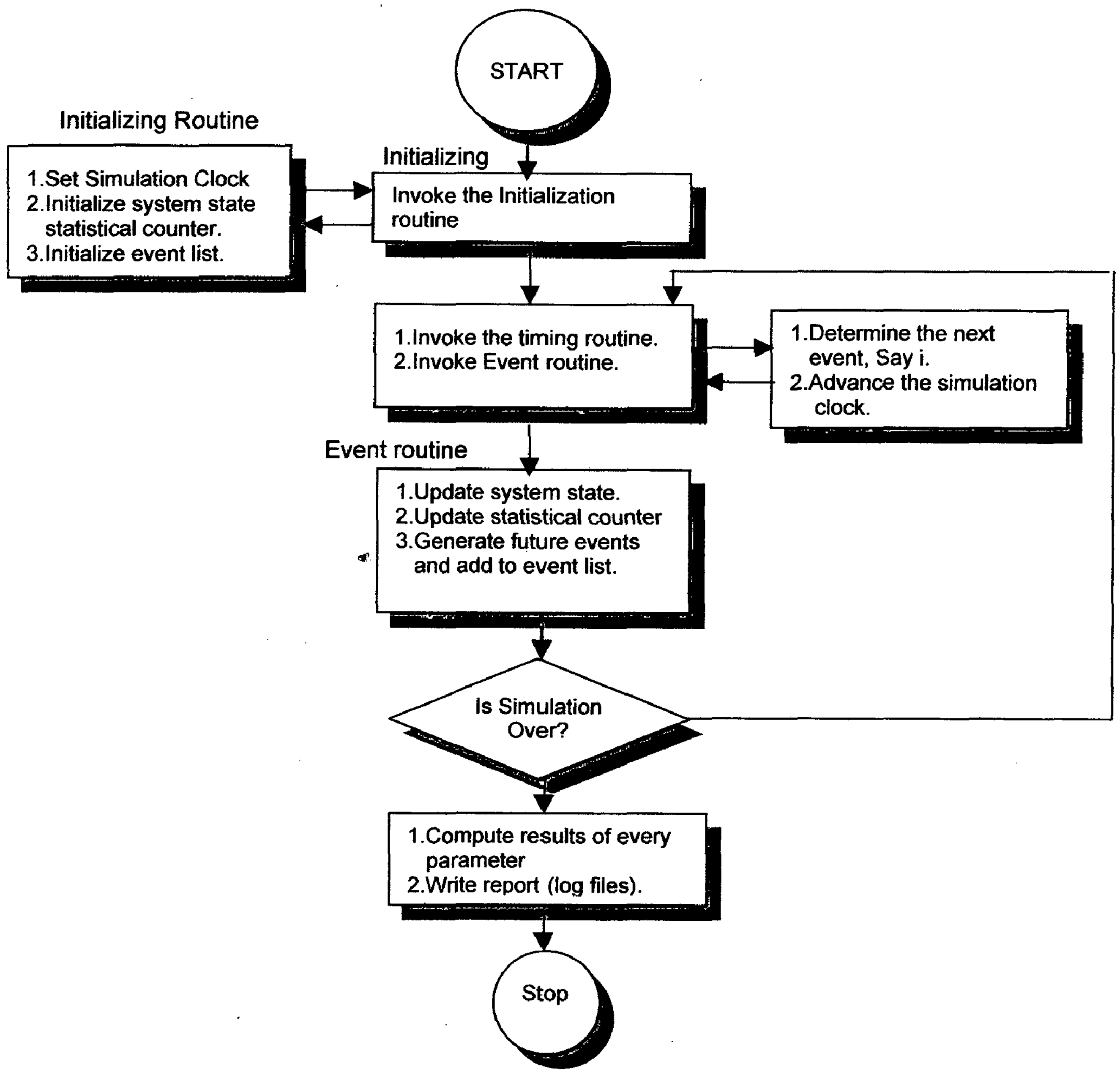

START
Initializing Routine
1.Set Simulation Clock
2.Initialize system state statistical counter.
3.Initialize event list.
Initializing
Invoke the Initialization routine
1.Invoke the timing routine.
2.Invoke Event routine.
1.Determine the next event, Say i.
2.Advance the simulation clock.
Event routine
1.Update system state.
2.Update statistical counter
3.Generate future events and add to event list.
Is Simulation Over?
1.Compute results of every parameter
2.Write report (log files).
Stop

Fig 3.1-Flow diagram of simulation program

3.2 Description of Simulation Components

Now we are explaining the various components of simulation tool, which we simulated through C programming language.

3.2.1 Simulation Clock

The simulator is *event* driven. Components send each other events in order to communicate and send cells through the network. The event manager in units of ticks maintains simulator time. The time is maintained as an unsigned 32-bit value. The simulator time represented by one tick can be changed by software modification. For the present simulation, a tick represents 10 nanoseconds.

3.2.2 ATM Switch

The switch is the component that switches or routes cells over several virtual channel links. A local routing table is provided for each switch. This table contains a route number (that is read from 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. Let's consider a cell arriving at the switch from a physical link. At the next switching slot time, after some delay, 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, i.e., it is only when the CBR/VBR queue is empty that the ABR traffic is sent, and cells in the ABR queue have priority over cells in the UBR queue. If either queue exceeds a High Threshold value set by the user, 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 (set by the user) determines the available buffer space for each type of queue (CBR/VBR, ABR, or UBR). If any queue exceeds the set limit, cells are dropped and this is recorded as a percentage of the total number of cells received by the switch. Also, there is a per port cell drop parameter recorded for each queue.

	Name:
	Delay to process a cell (sec):
	Slot time (Mbits/sec):
	Output queue size (cells):
	High Threshold, Q cong. flag (cells):
	Low Threshold, Q cong. flag (cells):
	Logging every (n ticks) (e.g., 1, 100):
	Cells received:
	Cells in VBR Q to link n:
	Cells dropped in VBR Q to link n:
	Cells in ABR Q to link n:
	Cells dropped in ABR Q to link n:

Table 3.2.2.1 Switch Information Window

Description of Input/output parameters of Switch information window shown in Table 3.2.2.1

In above information window the shaded cells are the output parameters

a) Name: Name of switch

b) Delay to process a cell: An increment of time after the arrival of a cell at the switch places the cell on the outgoing link.

c) Slot time: The rate at which cells are switched from an input port to an output port. The program calculates the cell slot time from the bit rate entered. The actual switching of a cell from input port to output port occurs at the beginning of a slot period.

d) Output queue size: Available buffer space for a queue; the same value is used for every queue in the switch. When a cell is ready for transmission but a slot on that link is not available it waits in a queue at that port.

e) High Threshold, Q congestion flag: If the number of cells in any queue exceeds this value the congestion flag is set.

f) Low Threshold, Q cong. flag (cells): The congestion flag is cleared when the number of cells in all queues fall below this value.
g) Logging every (n clock ticks): If n is set to 1, data will be logged for a parameter anytime there is a change in its value. Potentially, this could occur at every tick. Since this may result in an extremely large data file, it may be desirable to set n to a larger number. For example, if n = 100, logging will occur only if a change occurred and 100 ticks had gone by since the last logging activity.
h) Cells received: Total number of cells received by the switch.
i) Cells in xBR Q to link n: Cells awaiting transmission in a given priority queue. There are two types of queues for each port - a CBR/VBR queue and an ABR queue. Cells in the CBR/VBR queue have top priority; a cell from the ABR queue will be sent only if the CBR/VBR queue is empty.
j) Cells dropped in xBR Q to link n: Cells dropped at a port when a queue exceeds its maximum size.

3.2.3 Broadband Terminal Equipment (BTE)

The BTE component simulates a Broadband ISDN node, e.g., a host computer, workstation, etc. 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. If no slot is available for immediate transmission a cell queued in one of three queues, a VBR/CBR queue, an ABR queue, or a UBR queue. The user can specify the maximum output queue size; if either queue exceeds this limit cells will be dropped. The parameters that can be monitored for a B-TE are the number of cells in an output queue and the number of cells dropped at each queue. Also, the total number of cells received from the network may be monitored. In this case, the cells are enqueued in a special queue (called Input Queue) to control their transmission on the network.

	Name:
	Max. o/p Queue size(cells):
	Logging every (ticks) (e.g.,100,1000):
	Cells received :
	Cells in xBR Q to link n:
	Cells dropped in xBR Q to link n:

Table 3.2.3.1 BTE parameters information window

Description of input/output parameters of Broadband Terminal Equipment:

a) Maximum Output Queue Size. Available buffer space for each type of queue.
b) Logging every n ticks. same as switch concept.
c) Cells Received. Total number of cells received by the B-TE.
d) Cells in xBR Q to link n. Cells awaiting transmission in a given priority queue. There are two types of queues - a CBR/VBR queue and an ABR queue. Cells in the CBR/VBR queue have top priority; a cell from the ABR queue will be sent only if the CBR/VBR queue is empty.
e) Cells dropped in xBR Q to link n. Cells dropped at the network port when a queue exceeds its maximum size.

3.2.4 ATM Applications

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

(a) A constant rate traffic where the user specifies the bit rate. Cells will be generated at the specified rate for the duration of the simulation.

(b) Variable Bit Rate -. This type of traffic has an ON-OFF source. Both the burst period (ON) and the silence period (OFF) are drawn from an exponential distribution. The user specifies the mean burst length, the mean interval between bursts, and the bit rate at which cells are generated during the ON period.

Another ATM Application type that can be simulated is a TCP/IP application. This application can be used with either the ABR or UBR service.

(i) CBR (Constant Bit Rate)

	Name:
	Bit Rate (Mbits/sec):
	Start time (usec):
	Number of Mbits to be sent:
	Other End Connection: Name

Table 3.2.4.1 CBR parameters information window

a) Bit Rate: The desired bit rate, which you want from CBR application.
b) Start time: This is the number of microseconds after the program starts that the application will begin generating cells.
c) No. of Mbits to be sent: User will specify how many no. of Mbits he want to sent.
d) Other end connection: Name of ATM application at other end.

(ii) VBR (Variable Bit Rate)

	Name
	Bit Rate (Mbits/s):
	Mean Burst Length (secs):
	Mean Interval Between Bursts(secs):
	Start Time (secs):
	Number of Mbits to be sent:
	Other End Connection: Name

Table 3.4.2.2 VBR parameters information window

In this VBR application traffic is generated as an ON-OFF source. Cells are generated at the specified bit rate during a burst. Mean burst length and mean interval between bursts are user specified, but the actual periods of both are drawn from an exponential distribution.

(iii) ABR (Available Bit Rate)

	Name
	Bit Rate: 0
	Start time (sec): 0
	Number of Mbits to be sent: 0
	Other End Connection: Name

Table 3.4.2.3 ABR parameters information window

All parameters of ABR are same as CBR/VBR applications, but this differ from CBR/VBR in only the priority of application.

c) **TCP/IP Application:** The TCP/IP Application sends data in large packets. These packets must be segmented to fit into the ATM cell structure before being put on the network. A rather extensive set of parameters is provided that gives the user flexibility in controlling and monitoring this type of application.

	Name:
	Bit Rate (Mbits/sec):
	Buffer Size (bytes):
	Transmitter's State: FALSE
	Start Time (sec):
	Start Random Period (sec):
	Transmission Size (bytes):
	Number of bytes unsent:
	Sender sequence number logging
	Sender ACK sequence number logging
	Receiver sequence number logging
	Mean packet processing time (sec):
	Packet processing time variation (sec):
	TCP open time (sec):
	TCP close time (sec):
	Connection Busy: FALSE
	Packet input queue has 0 pkts
	Max segment size (octets):
	Peer Receive Window size (octets):
	Tahoe (0), Reno (1), standard (2): 0
	Timer granu. in us (e.g. 100000,500000):
	RTT (secs):
	Forward Trip Time FTT (sec):
	Cwnd in bytes:
	Average throughput (bytes/sec):
	RTO (current):
	Retransmission percentage:
	Other end connection: Name

Table 3.4.2.4 TCP/IP parameters information window

Input Parameters

Bit Rate: The bit rate for the cells on the ATM route.

Buffer Size: The size of the user's buffer, large enough to hold many packets, but a fraction of the total transmission size.

Transmitter's State: A TRUE/FALSE control. If FALSE, no transmission will take place, but the Application can still receive.

Random Start Period: If Start Time is negative, the value entered here is the mean for a random start time.

Transmission Size: The total number of data bytes (payload) to be sent.

Mean Packet Processing Time: The mean delay to process the packet.

Packet Processing Time Variation: A computation based on a random perturbation in the processing delay in the range [-Packet Processing Time Variation, +Packet Processing Time Variation].

Maximum Segment Size: The maximum size of the TCP/IP packet, whether it is being sent or being received.

My Receive Window Size: This number determines how many packets are going to be sent without waiting for an acknowledgment.

Timer granu: The TCP time granularity is the minimum time separating the release of a TCP packet and the expiry of the associated timer. Most TCP implementations use a coarse grained timer of 500 ms.

Tahoe, Reno, Standard: Version of the TCP congestion avoidance and control algorithm. Three possible values can be specified.

-0 corresponds to TCP-Tahoe

-1 corresponds to TCP-Reno

-2 corresponds to Standard TCP

Output Parameters

Number of Bytes Unsent: This is the number of bytes remaining from the total specified under transmission size.

Packet input queue has n packets: This queue contains packets waiting for TCP processing, both for transmission (before segmentation) and for reception (after reassembly).

Peer Receive Window Size: This is the other-end companion to My Receive Window Size.

RTT: Round Trip Time - time from packet sent to ACK received. This is set to a default value at the beginning.

Transmission Size: Every TCP/IP packet has a sequence number, including the ACK packets: The following three parameters let the user enable the logging of these numbers as the packets are sent or received. Note that only the logging function applies, no metering is possible.

Sender Sequence Number Logging

Sender ACK Sequence Number Logging

Receiver Sequence Number Logging.

TCP Open Time: The time that the first TCP packet was sent.

TCP Close Time: The end of the TCP transmission (all bytes have been sent).

Connection Busy: Activity flag for TCP processing; TRUE = busy, FALSE = not busy.

3.2.5 Link Components

This component simulates the physical medium (copper wire or optical fiber) on which cells are transmitted. The user may choose the link speed from a list of several different standard rates. The user also specifies the length of the link. The output parameter reported by the simulator is link utilization in terms of bit rate (Mbits/s). The measurement of link rate is averaged over a period of 10 cells.

In general this simulator can simulate anything that can be modeled by a network of components that send messages to one another. The components schedule events for one another to cause things to happen. The model being simulated and the action of the components are entirely determined by the code controlling the components. In this simulation if want to add more components then we can easily add new component due to the easy architecture of my software data structure.

The simulator program includes a graphical user interface which provides the user with a means to display the topology of the network, define the parameters and connectivity of the network, log data, and to save and load the network configuration. In addition to the user interface, the simulator has an event manager, I/O routines, and various tools that can be used to build components.

Link component Information window:

	Name:
	Link Speed (Mbits/sec): 0
	STS-1 51.840 Mbits/sec
	STS-3C 155.520 Mbits/sec
	STS-12C 622.080 Mbits/sec
	STS-24C 1244.160 Mbits/sec
	DS-3 44.736 Mbits/sec
	Distance (km):
	Link Rate (Mbits/sec) to Switch n:
	Link Rate (Mbits/sec) to B-TE n:

Table 3.2.5.1 Parameters information window of Link component

There are only two input parameters for a Physical link, link speed and distance. The link speeds shown in the window are not selectable with the mouse; the desired speed (in Mbits/s) must be typed into the text window. However, the bit rate typed in need not be exact; the software will accept a round number near the standard rate. The bit rates shown include overhead bits. The simulator maps the entry into the correct payload rate when doing calculations for bits transmitted. The link output parameter is link utilization (in each direction) in terms of bit rate (Mbits/s).

Chapter 4

IMPLEMENTATION DETAILS

In this work, simulation software is developed to simulate an ATM network for study of various kinds of network performance parameters. In this chapter the implementation details with the architecture of simulation software will discuss. Data structure and routines will also discuss with necessary structure of simulated software program.

The simulation model used is the discrete event simulation method. So simply this is a event driven simulation. Components send each other events in order to communicate and send cells through the network. My simulator software contains an event manager who provides a general facility to schedule and send or "fire" an event. An event queue is maintained in which events are kept sorted by time. To fire an event, the first event in the queue is removed, the global time is set to the time of that event and any action scheduled to take place is undertaken. Events can be scheduled at the current time in the future.

For the discrete event simulation of ATM network the following are simulated:

a) ATM Switch
b) BTE component
c) ATM Application
d) Physical link

4.1 Simulation Software Architecture

The simulation program written in C Language (with X WINDOW SYSTEM) and run under LINUX (version 7.3) operating system. For the development of Graphical User Interface (GUI), I used X WINDOW SYSTEM on C language.

4.1.1 Display of Simulation software

The display is composed of three major parts:

- A network window to display ATM network configurations. This window is used both while creating the configurations and to show network activity while the simulation is running.
- A text window for messages that will prompt the user, and to provide a place for the user to input text or parameter values.
- A control panel that consists of a clock and several control buttons, such as START, QUIT, PAUSE etc.

4.1.1.1 The Network Window

The entire area not otherwise occupied by clock and control buttons is the Network Window. If the program is started with no configfile this area is blank. The network is represented as a collection of components connected to each other in the desired configurations. ATM switches and broadband terminals (BTEs), are represented by rectangular boxes while ATM Applications are represented by ellipses; both shapes contain the name of the component. ATM switches and BTEs are interconnected by Physical Links. The Links are also considered components and are identified by name, but they are represented on the figure by straight lines. The connection between a B-TE and an ATM Application is also represented by a line but is not considered a component, i.e., it is not a physical entity and has no associated parameters.

Other information is displayed in the Network Window as required. When creating or modifying a component an information window appears beside its symbol, displaying the component's parameters. When a virtual connection is established between ATM applications a dotted line appears denoting the path of the information flow. When a simulation is running, one or more information window may appear on the screen to display information about selected parameters.

4.1.1.2 The Text Window

The text window appears as a bar at the bottom of the screen. The text window allows the program to present various messages to the user. In addition, any keyboard input is displayed in the text window. The cursor does not need to be in the text window when entering information with the keyboard. When entering information using the keyboard, pressing "Enter" key without entering any text will tell the program to accept a default value or to abort that operation.

4.1.1.3 The Control Panel

The control panel appears on the right hand portion of the screen. It contains an analog clock, a digital clock, and an array of control buttons.

(a) Analog Clock

The analog clock indicates the passage of simulator time in a graphic style. The intent is not a precise timer but to give the user an indication of how busy the simulator is. A tick is a movement of 6 degrees around the circle. Each tick of the big hand represents 1 millisecond. Each tick of the small hand represents one revolution of the big hand (60 milliseconds).

(a) Digital Clock

The digital clock provides a display of current simulator time accurate to the nearest 10 nanoseconds.

(a) Control Buttons

The following is a description of the functions of each control button. All of the functions are initiated by clicking with the **middle** mouse button.

START	Clicking on this button will start the simulation with simulated time initialized to zero. The simulation can be restarted as many times as we wishes; each click on the button will initialized the simulation.
PAUSE/RUN	This button toggles between two modes. When the simulation is running the word PAUSE will be displayed. Clicking on the button will then stop all activity with all parameter and time information held in

	place. With the simulation stopped, the button label will change to RUN; clicking on it will cause the simulation to resume running with current settings.
DELAY	This button allows the user to slow down the simulation by setting a delay between each event firing. The text window will appear asking for the desired delay(in microseconds).
KILL	This button may be used to stop a simulation in progress or to eliminate components. Clicking on the KILL button while a simulation is in progress stops all activity. If a simulation is not running, clicking on a component after KILL has been clicked will delete that component.
LOAD	This button allows the user to load a network configuration. The text window appears asking for the name of the file to be loaded. Note that this erases whatever configuration was being displayed on the screen at the time.
SAVE	The SAVE control button allows the user to save the present configuration in a specially formatted text file which is readable by the simulator at LOAD time. The text window appears asking for a filename under which to save the configuration.
SNAP	This is similar to SAVE, but in addition it saves the present arrangement of information windows on the display. The text window appears asking for a filename under which to save the configuration. Present values of the component's parameters are saved.
PRINT	Prints out the network topology into a postscript file.
QUIT	This is the normal exit from the simulator program. Note that clicking on the QUIT button while in KILL

mode merely causes an exit from that mode; it does not cause an exit from the program.

4.2 Brief Description of Data structure and routines used

- **Components**

The component is the basic building block of the simulator. There are different classes of components; examples are switches, physical links, terminal equipment, and ATM applications. Some classes allow different types within the class in order to accommodate the simulation of a variety of implementations. For example, an ATM application may generate traffic at a constant bit rate, or a variable bit rate that is governed by some particular distribution function. Every component consists of an action routine and a data structure. All components of the same type share the same action routine; this routine is called for each event that happens to a component. Each instance of a component has its own data structure that is used to store information that characterizes the component plus some standard information required by the simulator for every component.

- **Classes and Types**

Some component like ATM Application has a class and a type. A particular class of component may contain several different types of components. The following are the different classes of components currently defined and, in parentheses, the way the names appear in the source file comptypes.h:

- ATM Applications (CONNECTION_CLASS)

For now, the Link, Switch, and B-TE classes contain only one type each. Respectively, they are (as defined in comptypes.c and comptypes.h):

- Physical Link (ATMLINK)
- ATM Switch (SWITCH)

The ATM Applications class contains many types; these are defined as follows:

- Constant Bit Rate- (CBRCONNECTION)

- Variable Bit Rate - (VBRCONNECTION)
- Available Bit Rate - (ABRCONNECTION)
- TCP/IP (UBR) - (TCPCONNECTION)

4.2.1 Component Data Structures

Each instance of a component has a data structure that is used to store any information needed by the component, as well as standard information needed by the simulator for every component. Component structures are kept in a list; the order of the list depends on the order of creation of the component. Each different *type* of component has its own structure which is defined in the header (.h) file for that type, but the beginning of every component structure is the same. This generic structure is as follows:

```
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 */
COMP_OBJECT         co_picture;             /* Graphics object to be displayed on screen */
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;
```

- **Parameters**

Any information about a component that needs to be displayed on the screen, logged to disk, or saved in a configuration file must be stored in a *parameter*. A parameter is a data structure that (besides storing a value) stores information needed to display, save, or load the parameter. 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 Param structure is defined in **component.h**; it is listed below.

```
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 */
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 */
GRAF_OBJECT p_my_picture;               /* The graphics object to display this */
int                    p_log;           /* Integer associated with this param for logging */
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 (if needed) */
caddr_t p;
int vpi;
int i;
} pi;
tick_t sample;         /* Keeps track of time parameter value was updated */
} u;                   /* This structure is used and maintained by the simulator */
} Param
```

A component may have as many parameters as needed. They are stored in a doubly linked list pointed to by **co_params**. The I/O routines iterate through this list to display the parameters as described below. The action routine may reference the parameters any way it wants. In addition to the linked parameter list, there is a set of pointers in the component that point to the individual parameters. As the parameter is initialized and added to the list, the pointer is set to point to it. Then the action routine can use a named variable to refer to the parameter rather than trying to search through the list.

The actual value of a parameter is stored in a structure at the end of the Param structure. Currently, the structure has room for an integer, a double, or a pointer. A new value type can be added just by changing the definition of the structure. This value is not used by any part of the simulator except for the action routine of the component that contains the parameter. The I/O routines read and change the value only by calling one of the functions pointed to in the parameter structure. A parameter is initialized by calling **param_init()** with arguments containing values for various fields in the parameter structure. The values for the arguments calc_val, make_text, make_short_text, and input are pointers to predefined functions in **subr.c**, which consists of a set of routines that calculate the parameter's value, display it, etc., for a variety of types of the parameter, such as int, double, boolean and more. The following is a listing of the **param_init()** routine.

```
Param *
param_init(c, name, calc_val, make_text, make_short_text, input, display_type, flags, scale)
Component *c;                        /* Pointer to the component */
char *name;                          /* Name of parameter */
PFD calc_val;                        /* Function to update the parameter value for display */
PFP make_text,make_short_text;       /* Function to convert value to a string */
PFI input;                           /* Function to read input string and convert into param
value. */
int flags;                           /* How to display -- look below for details */
```

The names of arguments listed below correspond to fields in the parameter, which in most cases have the same name, beginning with the prefix p_. For example, the argument calc_val is for p_calc_val, flags is for p_flags, etc.

p_make_text Used to generate text for parameter display, this element returns a pointer to a string. The string is expected to contain some meaningful, humanreadable representation (i.e., with some sort of label) of the value of the parameter.

p_make_short_text Also returns a pointer to a string, but the string contains only the value of the parameter (no labels). Used primarily for logging data to disk.

p_input Points to a function that will read an input string from either the keyboard or from a file. This routine will convert the string to an appropriate value and store it into the parameter. This is used for the initialization of values that affect the operation of the component, and that can vary from one instance of the component to another. For example, hosts have a "Processing delay" parameter that is the time needed to process a cell.

p_flags Contains flags that control the display. The constants (masks) are defined in the file **simx.h** with the following names:

InputMask When set, the simulator will call the function pointed to by **p_input**. Parameters that have this flag set will also have their values saved (using the **p_make_short_text** routine) when the configuration of the simulator is saved.

CanHaveLogMask If the parameter has this flag set, the user can cause the parameter values to be written to a file on the disk as the values change. To update screen displays or to cause data to be logged to a disk file, the action routine for the parameter must call **log_param(c,p)** every time the value changes. The variables **c** and **p** are pointers to the component and parameter, respectively. (The **log_param()** function is found in the **log.c** file.)

- **Neighbors**

Neighbors are stored as a list of **Neighbor** structures; this list is pointed to from component structures. Each neighbor structure contains a pointer to the neighboring component, a queue in which to store cells (if needed), a busy flag, and a pointer to a parameter to display anything that might be associated with the neighbor. The definition of the Neighbor structure is listed below; it can be also be found in **component.h**.

```
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
caddr_t         n_data;
} Neighbor;
```

When a neighbor is added, the component must create and initialize a neighbor structure, and put it on its neighbors list. If there is some piece of information associated with the neighbor that must be displayed, a parameter structure must be allocated, initialized appropriately, and added to the queue of parameters in the component structure. See the function **b_neighbor()** in **bte.c** for an example of usage. The following is defined in **subr.c** and can be used when writing a new routine to give it the capability to add neighbors.

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

4.2.2 Relationship of Data Structures

As stated in the preceding sections, the component data structure contains the doubly-linked parameter list and a set of pointers that point to the individual components. When a neighbor is added, the component creates a neighbor structure and puts it on its neighbors list. Each neighbor structure then contains a pointer to a neighboring component. When all of the components in the network are created and linked together then "list_of_components" will be completed and will include all elements in a network topology, e.g., link1, sw11, switch2, ABR2, etc.,

♦ **Events**

The simulator is event driven. 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. When the user clicks on the START button, each component is sent a reset command followed by a start command, then the simulator enters a loop. The loop processes any X events, updates the display, then fires all the events at the head of the event queue that have the same time.

- **Command Set (EV_CLASS_CMD)**

  All components must accept the following commands. The component need not actually use the command but should respond in an orderly and predictable way when the command is received. When used in an action routine, the action routine should return NULL if an error occurs during a command, and something that is non-NULL otherwise.
- **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. The simulator as specified in the Parameters section of this document will initialize them.
- **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 (Broadband-ISDN Terminal).
- **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_LEGAL_NEXT_HOPS** **arg** points to an *l list* that contains a virtual channel connection being constructed (not including **comp**). The list contains only the components in the path so far, **comp** is the component being considered as the next step in the connection. The action routine must return a new list of the components that are legal in the path after **comp**. The X routines know that a component of type ATM Application must be at the beginning of a virtual channel. When the user picks an ATM Application, the X routine calls that component action routine with this command to find out which components are allowed to be next on the path. As the user picks more components, the process continues until he picks another ATM Application to end the path.
- **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.

4.2.3 ATM Network-Related Issues

a) ATM Cell Definition

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

The cell data structure is not constrained to be any particular format. If we are only modifying some existing components we should not remove any elements from the structure, but if we are writing a set of components from scratch, a cell can contain anything.

The following is a simple example of a cell structure:

```
typedef  struct_Cell {          /* Define cell structure */
struct _Cell *cell_next;        /* Pointer for use by the queue the cells will be stored in */
VPI vpi;                        /* Route number (virtual path identifier) */
PTI pti;                        /* Payload type identifier */
struct cell_payload {           /* Structure for the payload portion */
Packet *tcp_ip_info;            /* The payload will */
AAL5_Trailer len;               /* be any one of */
RM rm;                          /* these three types */
} u;                            /* Structure */
} Cell
```

An event may include a cell, and most simulation do so. Normally, cells are transmitted from one component to another by having the transmitting component call a routine (**ev_enqueue**) which creates a new event and places it in a queue to be fired at the appropriate time. The receiving component must be able to process the event in order to receive the cell. This process is explained in more detail in the Events section of this document.

A module to handle the allocation and deallocation of cells is provided in the package. The module keeps track of all the cells, so that when the simulator is reset all cells can be freed in one step. **cell_alloc()** returns a new cell, **cell_free()** frees a cell, and **cell_free_all()** frees all cells. The simulator obeys the convention that all components must dispose of all cells that they receive in one way or another. In other words, a component that receives a cell must either call **cell_free()** on the cell or send the cell to someone else, but not both. Furthermore, a component that sends a cell to someone else should no longer refer to that cell. If it wants to save the cell for some reason (if the cell might be retransmitted, for example), the component must call **cell_alloc()** and make a copy of the cell.

b) **Setting Up the ATM Virtual Channel**

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. When the user clicks on an ATM Application that is at the other end of the virtual channel (this is done while making the route), the routing table at each ATM Switch is updated and information about the next link and the next ATM Switch found on the path is stored. The file route.c contains a couple of functions to manipulate connections. To determine where to route a cell next, the function

```
Route_info *
rt_lookup(some arguments)
/* ..*/
```

can be called from an ATM Switch action routine; this should return the next link and next switch. The routing process starts when **io()** within **IO.c** calls **make_route_event_handler(bevent)** which is found in **routes.c**. The routing process involves creating a **route_list**, which is a list of components. When finally the user clicks on a component of type **ATM APPLICATION** which is at the end of a route, all switches found in **route_list** call their respective **action_routines** to update their local routing modules.

c) **Tools**

Lists and Queues: Lists (doubly-linked lists) and queues (singly-linked lists) are used extensively throughout the simulator. Lists and queues contain variables to store the current, maximum and minimum length of the list/queue. A list has the following 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;
```

An element in the list, **l_elt**, has the following structure:

```
typedef struct l_elt {              /* list element */
struct l_elt *le_next, *le_prev;    /* Links */
caddr_t le_data;
} l_elt;
```

Because both lists that were efficient and lists that were flexible were needed, there are two kinds of lists. One kind requires that the item being placed on the list contain the pointers needed to link it into the list. This means that no extra memory is needed to put the new item into the list. However, this also means that the item being placed on the list must include room for one or two pointers at the beginning, and it can only be on one list at a time. Since the item itself contains the pointers, the pointers for the first list will be overwritten when it is placed on a second list. This type of list we have chosen to call an le list (or a qe queue). The le stands for list element, and it means that the items being placed in the list already have the pointers for a list element built in. As an example, the global list of components is an le list and the component structure contains two pointers (the structure elements **co_next** and **co_prev**).

The other kind of list allocates a small area of memory in which to store the pointers every time a new element is added to a list. This means that adding and removing items from the list is slower, but any type of data structure (even ones that don't have pointers at the beginning) can be placed on any number of lists any number of times. This type of list is called an l list (or a q queue).

Functions (and macros) that start with **le_** and **qe_** are the faster routines, and the ones that start with **l_** and **q_** are the more general ones. (With one exception: **l_create()** serves both types of list.) In the arguments to the functions, **l** and **q** indicate a list and a queue, respectively, and any other arguments are elements on which to operate. Here is a summary of the available list and queue commands:

l_create()	Create a new, empty list and return it. Returns NULL on error.
l{e}_addh(l, elt)	Add **elt** to the head of the list **l**. The **le** version does not return anything (it is a macro); the **l** type returns NULL on error (couldn't allocate memory to hold the pointers), non-NULL otherwise.
l{e}_addt(l, elt)	As above, but add **elt** to the tail of the list l.
l{e}_remh(l)	Remove the item at the head of the list and return it.
l{e}_remt(l)	Remove the item at the tail of the list and return it.
l{e}_adda(l, prev, new)	Add **new** to the list after **prev**, which must already be in the list. Again, the **le** is a macro that doesn't return anything, and **l_adda()** returns NULL on error.
l{e}_del(l, elt)	Delete **elt** from the list l.
l_find(l, elt)	Search for elt in the list. Returns a pointer to the l_elt that contains elt. An l_elt is the structure that contains

	the pointers used to add something to an l list. See list.h for the definition.
lq_delete(l)	This function works for both lists and queues. It also works for both flavors of each, although the effect is slightly different. For **le** lists, **lq_delete()** frees the list and the elements that were stored in the list. For **l** lists, the function does not free the items stored in the list, just the list and associated extra garbage.
lq_clear(l)	As with **lq_delete()**, this function works for both lists and queues.This function removes all the items from a list or queue. If it is a **le** list or **qe** queue, the memory for the items is also freed, otherwise they are merely removed from the list or queue.

The following functions perform the same actions on queues as the similarly-named functions for lists:

q_create()

q{e}_addh(q, elt)

q{e}_addt(q, elt)

q{e}_adda(q, prev, new)

q{e}_del(q, elt)

q_find(q, elt)

Finally, queues have the following operations of their own:

q{e}_deq(q) Removes and returns the item at the head of the queue.

qe_find(q, qe) Looks for the item **qe** in the queue. Returns **qe** if found, NULL otherwise.

qe_dela(q, prev) Removes the element after **prev** in q.

RESULTS AND DISCUSSION

So far the simulation model, working procedure and implementation details of the ATM network with various ATM traffic application have been discussed. After running the simulation we examine the data-log files of various network configurations (or topology) in the ATM network.

In this chapter, performance of various parameters in various ATM traffic applications is studied. Also the propagation delay, buffer size, and traffic characteristics such as average arrival rate, utilization of link and throughput (in case of UBR TCP/IP application) studied through simulation. Moreover, a performance comparison between the different ATM traffic application w.r.t. time and other parameters mentioned is drawn.

After running the simulation program we analyzed so may parameters of various components of the network through their log files, and we draw some graphs between some parameters, like end-to-end delay in a different traffic scheme w.r.t. time, then we saw that the various ATM traffic scheme have different delay for same kind of network topology and same configuration.

The screen snapshots of simulation software at runtime are also included in this chapter.

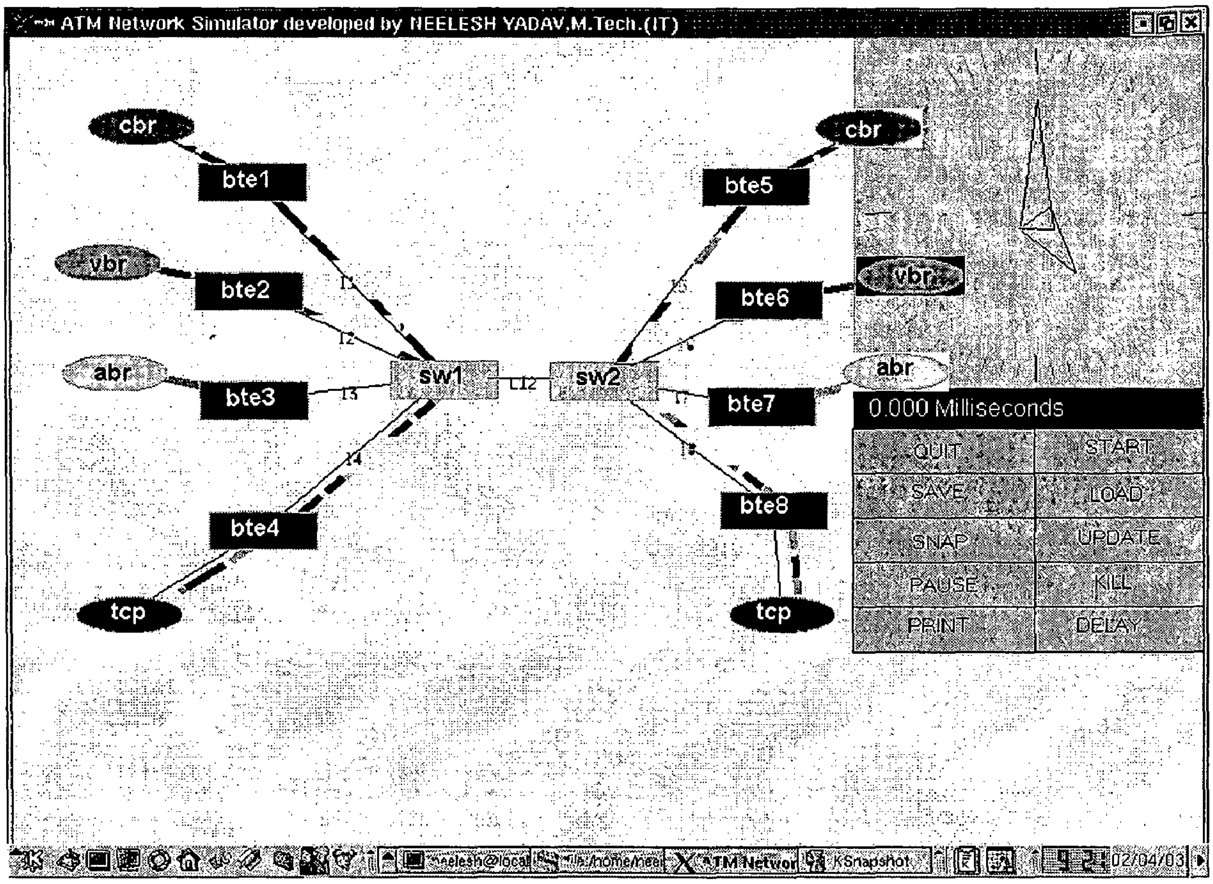
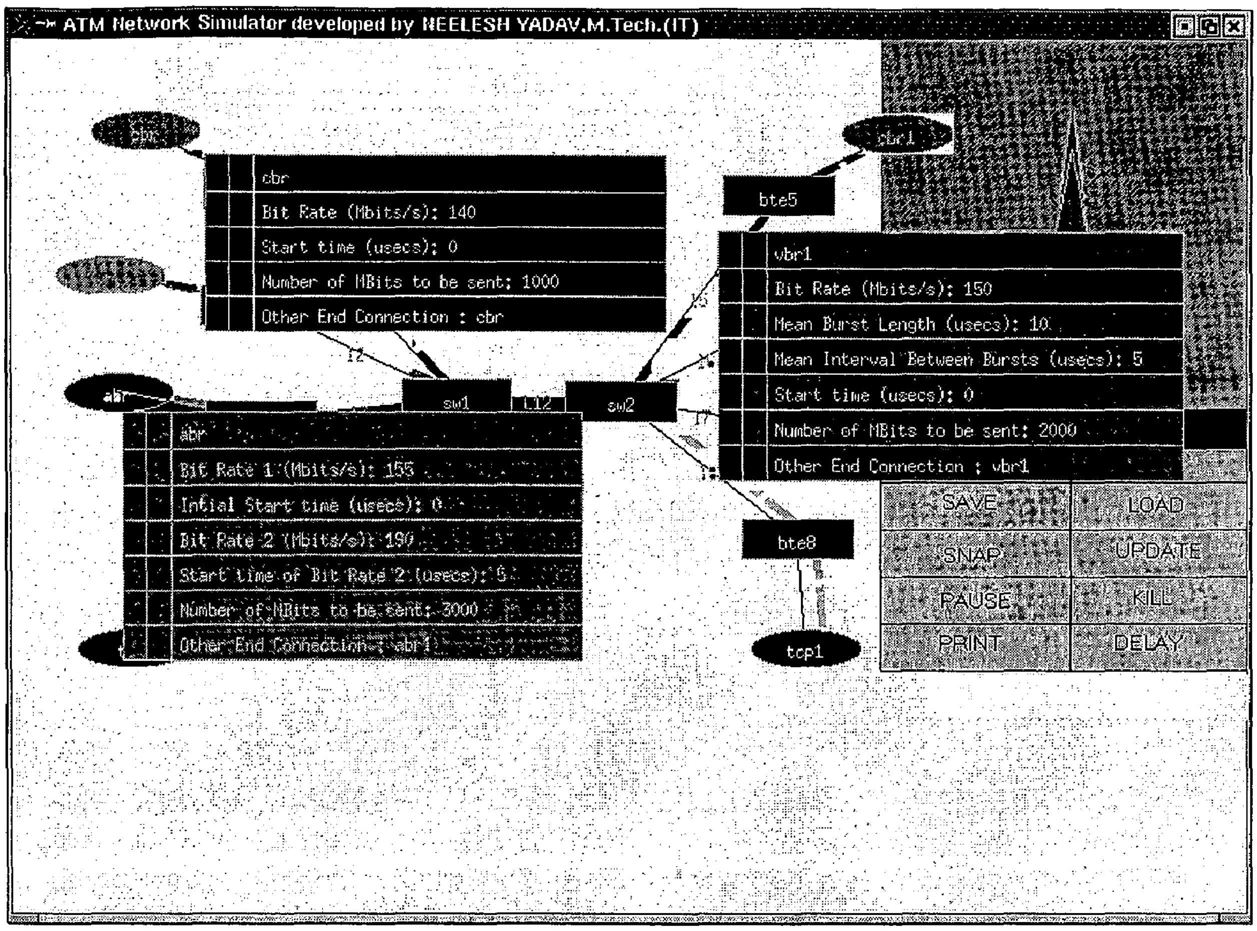


Fig. 5.1 Screenshot of the simulation program with an ATM n/w configuration

Description of Screenshot: In this screen shot the ellipse shaped components are the ATM traffic Applications like CBR, VBR, ABR, TCP/IP etc. The dark rectangle shaped boxes are the broadband terminal equipment, and light color rectangle are the switch .The lines drawn between ATM application to BTE components and BTE component to ATM switch are the Physical Links component. The every link connected from a B-TE to a switch has a name associated to it. In right hand corner the analog clock is there and below that the digital clock is also there.

The light and dark dashed thick lines are the ROUTES of one ATM application to another ATM application. In right hand lower corner there are a control toolbox for operating the simulation software with different kinds of operation as we already explain in chapter four.



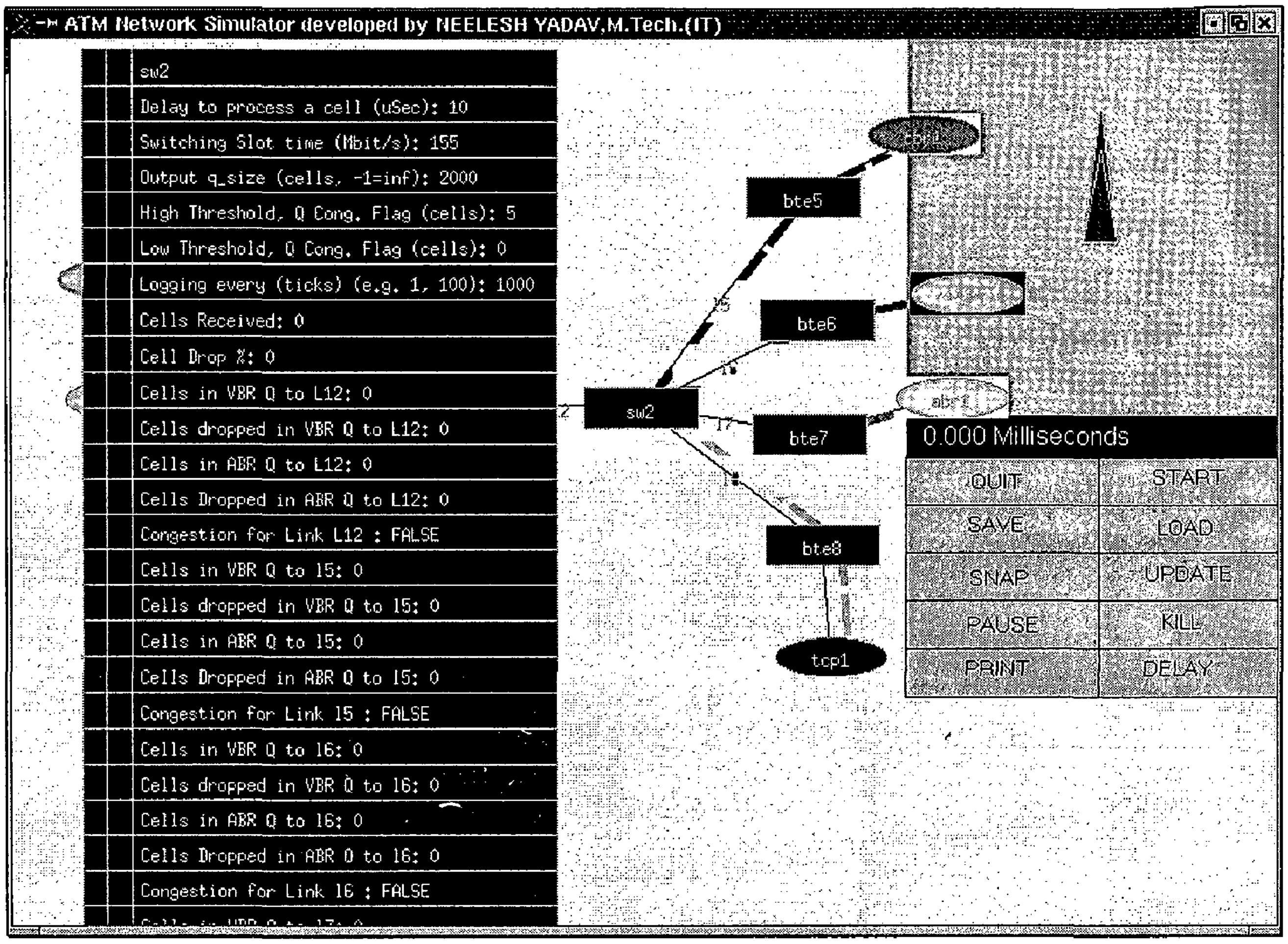
```
cbr
Bit Rate (Mbits/s): 140
Start time (usecs): 0
Number of MBits to be sent: 1000
Other End Connection : cbr
```

```
vbr1
Bit Rate (Mbits/s): 150
Mean Burst Length (usecs): 10
Mean Interval Between Bursts (usecs): 5
Start time (usecs): 0
Number of MBits to be sent: 2000
Other End Connection : vbr1
```

```
abr
Bit Rate 1 (Mbits/s): 155
Intial Start time (usecs): 0
Bit Rate 2 (Mbits/s): 190
Start time of Bit Rate 2 (usecs): 5
Number of MBits to be sent: 3000
Other End Connection : abr1
```

Fig. 5.2 Screenshot of previous network with information window of some component.

In this screenshot the input/output information window of CBR ATM application, ABR ATM application and BTE5 component is displaying.



```
sw2
Delay to process a cell (uSec): 10
Switching Slot time (Mbit/s): 155
Output q_size (cells, -1=inf): 2000
High Threshold, Q Cong. Flag (cells): 5
Low Threshold, Q Cong. Flag (cells): 0
Logging every (ticks) (e.g. 1, 100): 1000
Cells Received: 0
Cell Drop %: 0
Cells in VBR Q to L12: 0
Cells dropped in VBR Q to L12: 0
Cells in ABR Q to L12: 0
Cells Dropped in ABR Q to L12: 0
Congestion for Link L12 : FALSE
Cells in VBR Q to 15: 0
Cells dropped in VBR Q to 15: 0
Cells in ABR Q to 15: 0
Cells Dropped in ABR Q to 15: 0
Congestion for Link 15 : FALSE
Cells in VBR Q to 16: 0
Cells dropped in VBR Q to 16: 0
Cells in ABR Q to 16: 0
Cells Dropped in ABR Q to 16: 0
Congestion for Link 16 : FALSE
```

Figure 5.3-Screenshot of network with SWITCH information window

In this screenshot the information window of the switch2 is present .In which we can see so many parameters of the switch2 with its current value. We can see some parameters of this switch like the delay to process a cell, switching slot time, output queue size, High and low threshold, cell received in so many queues like VBR/CBR and ABR etc.

- **Topology of two simulated network configuration**

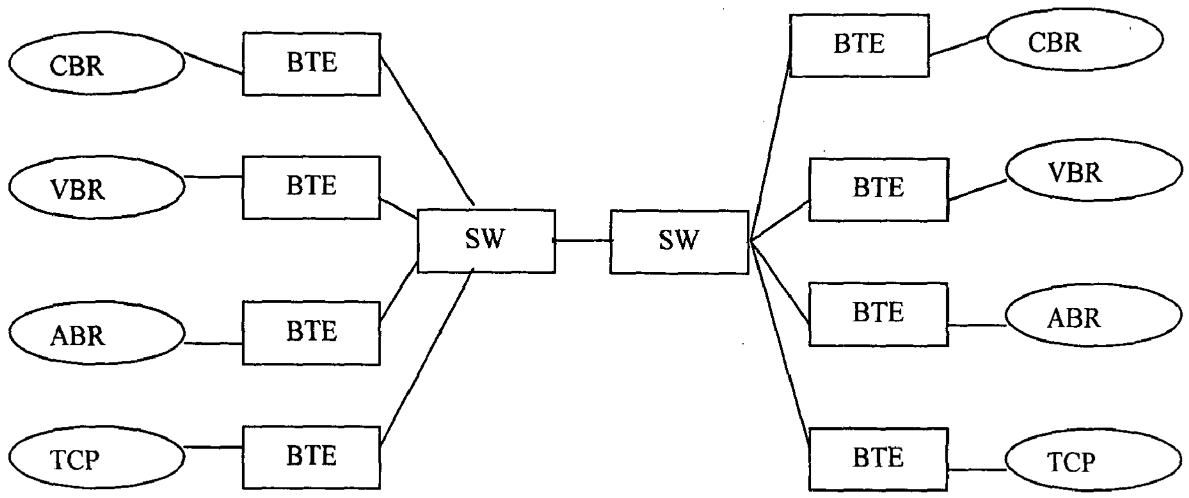


Fig.5.4 A simple ATM network configuration with all application.

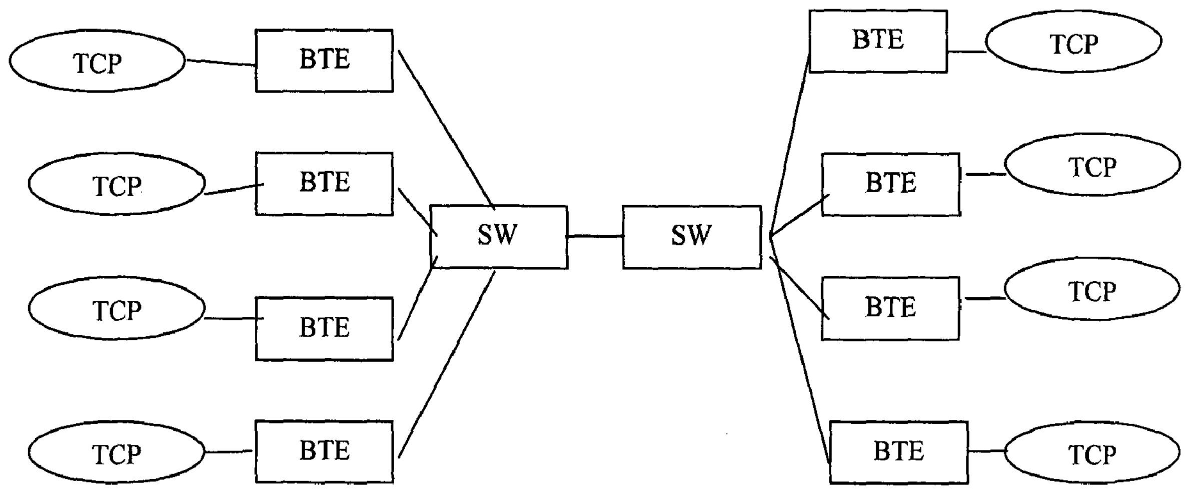


Fig. 5.5 A UBR TCP/IP based network configuration

End to End Delay Vs Offered load

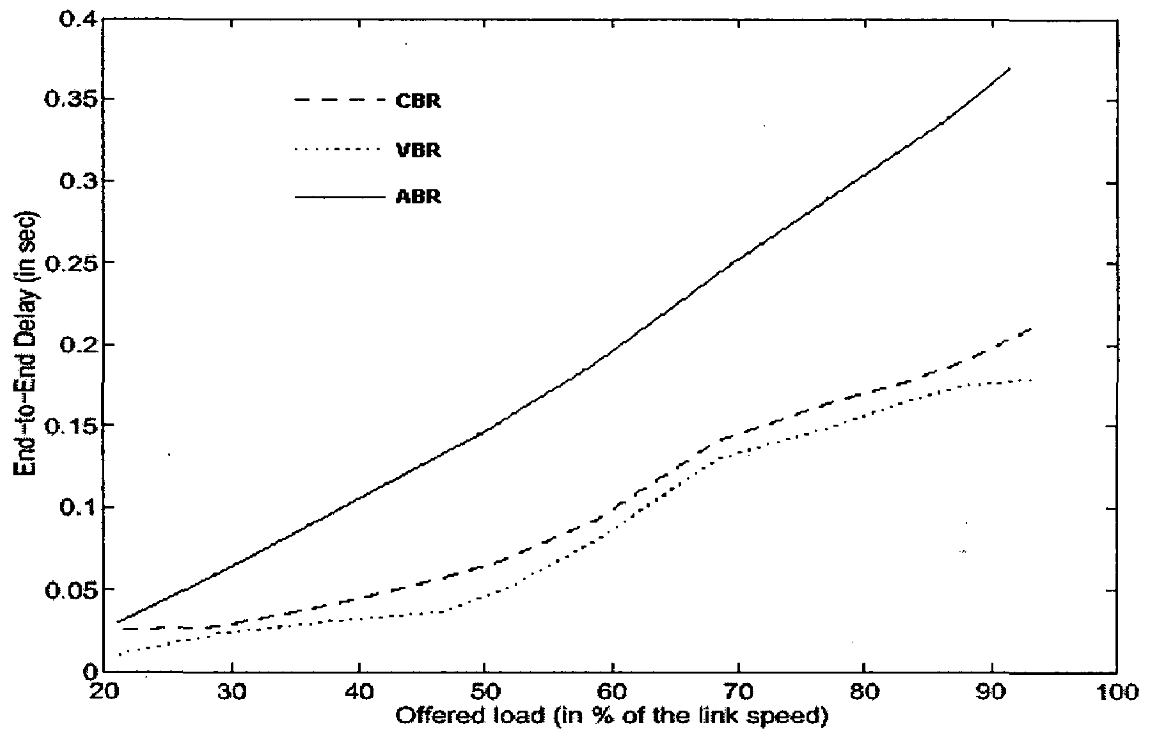


Fig. 5.6 End to End Delay Vs Offered load for CBR, VBR, ABR traffic

The above graph shows the end-to-end delay of CBR, VBR and ABR ATM services/traffic application for the same offered load in a particular ATM network configuration. Here we saw that the delay is maximum in case of ABR traffic. But in case of VBR traffic this is minimum. This indicate that if our application is real time and we want minimum delay in cell transmission then VBR is suitable for those kind of application so that the Cells can be generated at the specified bit rate during a burst. The CBR traffic delay is nearer to VBR delay but in case of CBR the delay is mainly depend on the specified bit rate. For some bit rate the delay may be less in CBR traffic, may be high in VBR traffic

- **Some results regarding UBR-TCP/IP application with congestion related issue.**

  One of the main problem protocols such as TCP, that use packet loss as a congestion indication is that congestion is only handled when it is already too late. The results depicted in following fig.5.6 and 5.7 show the link utilization achieved with Tahoe-TCP, Reno-TCP and Standard-TCP over a time period of one minute.

Link Utilization for TCP Tahoe, Reno AND Standard

(with buffer size 128 kbyte and 256 kbyte)

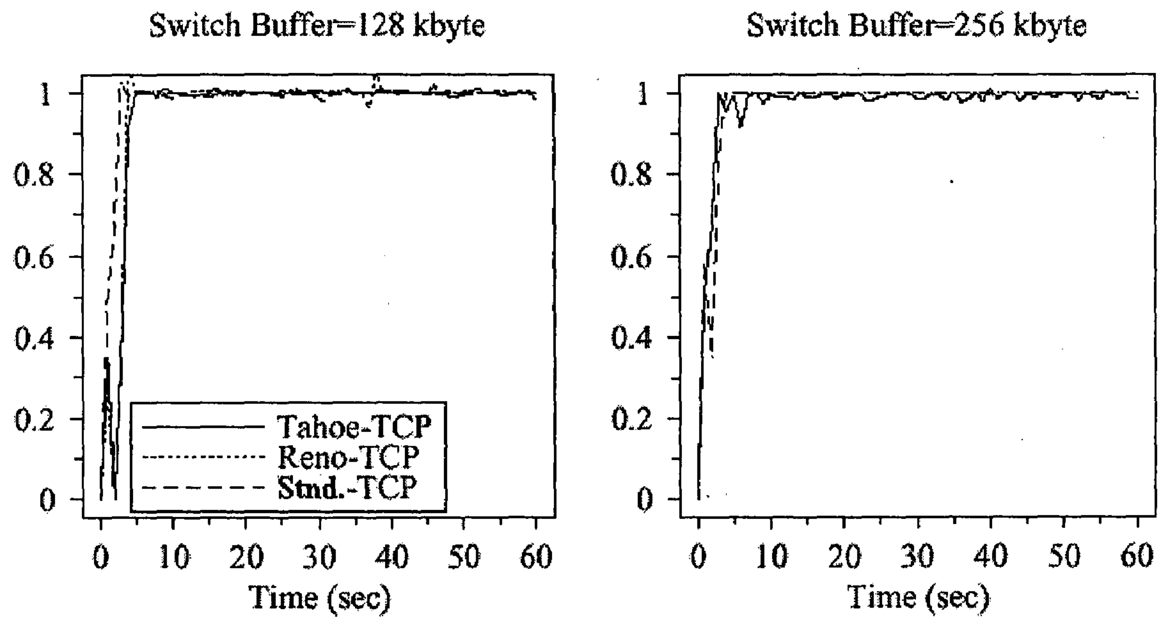


Fig.-5.7 Link utilization for Tahoe-TCP, Reno-TCP and Standard-TCP with a propagation delay of 40 msec.

Link Utilization for TCP Tahoe, Reno AND Standard

(with buffer size 512 kbyte and 1024 kbyte)

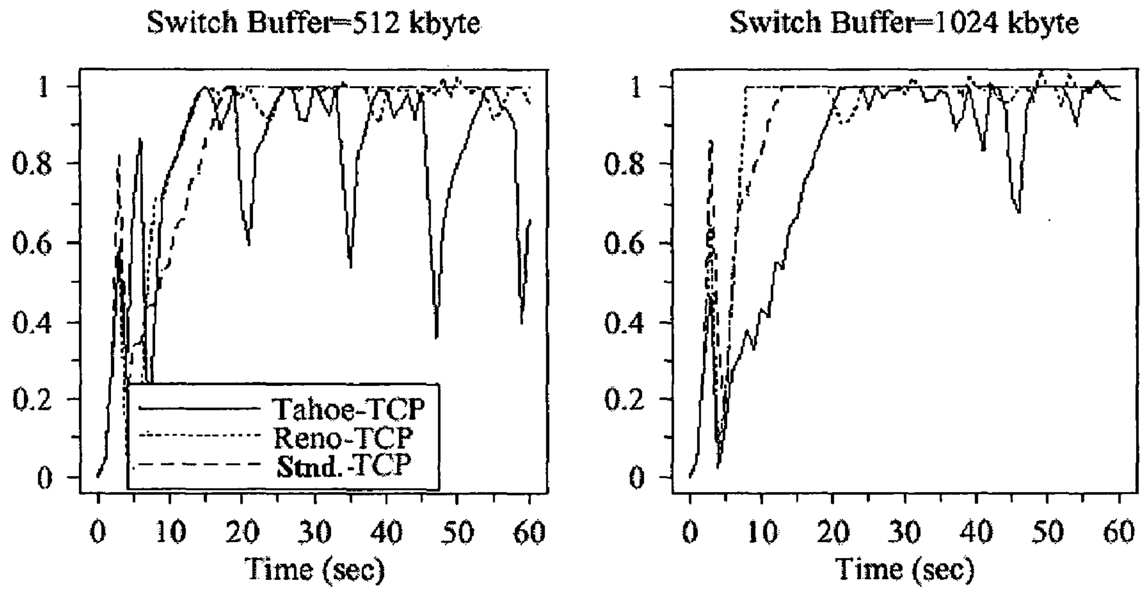


Fig.-5.8 Link utilization for Tahoe-TCP, Reno-TCP and Standard-TCP with a propagation delay of 220 msec.

Propagation delay	Buffer length	Tahoe TCP	Reno TCP	Stnd. TCP
40 mesc	128 kbytes	94%	96%	98%
	256 kbytes	97%	97%	98%
220 mesc	512 kbytes	80%	87%	88%
	1024 kbytes	81%	92%	93%

Table 5.1- Link utilization reached using Tahoe, Reno, Standard TCP with different switch buffer sizes and different round trip delays.

As expected, using Reno-TCP, results in general in higher throughput than that achieved with Tahoe-TCP. The result achieved with Standard TCP resembles to a great extent those achieved for Reno-TCP. The throughput improvements are especially evident for the WAN case. Due to the long round trip delay the recovery from packet losses takes more time increasing thereby the idle time of the sources and leading to the oscillating behavior that can be seen in Fig. 5.7.

Effective Throughput for 10 VCs , Buffer=10000 cells

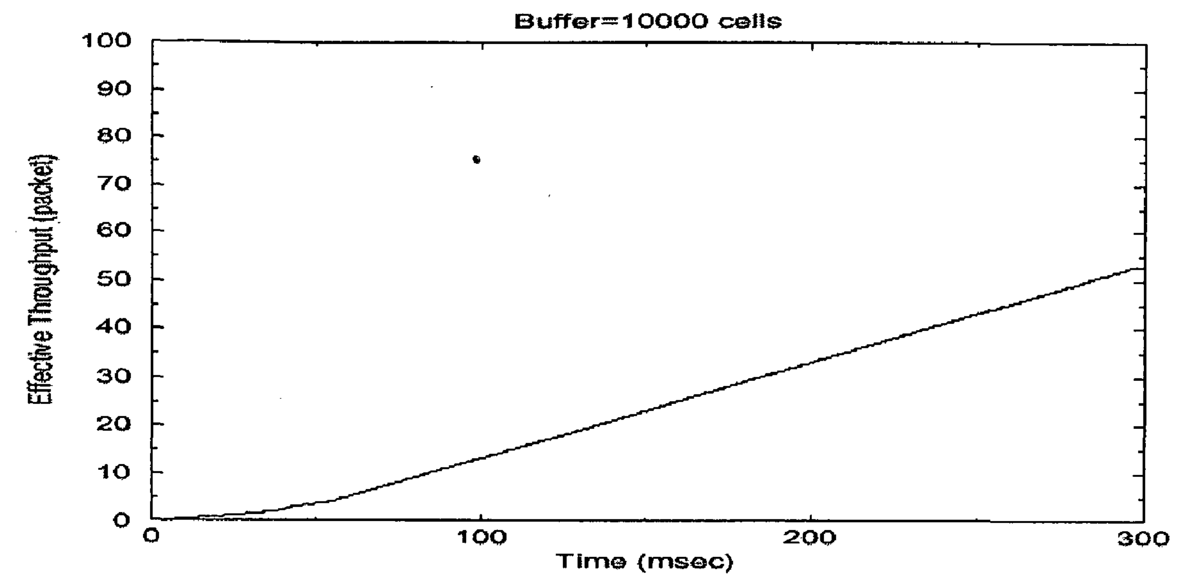


Fig.5.9 Effective throughput of Virtual Circuit in network configuration of Fig.5.6

Effective Throughput for 10 VCs Buffer = 10000 cells

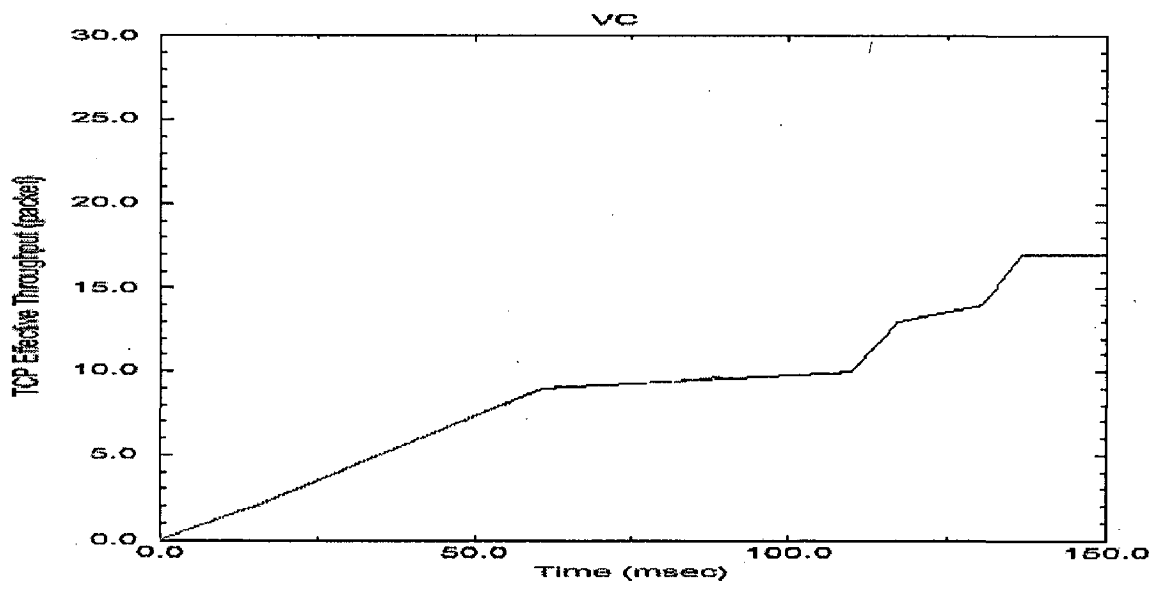


Fig. 5.10 TCP effective throughput of Virtual circuit in Network configuration of Fig.5.7

Above two graphs shows the effective throughput of VCs (virtual circuits) in simulated ATM network configurations of fig. 5.6 and 5.7 respectively. The throughput of fig.-5.10 indicates that the rise in throughput is continuos incremental due to the less no. of cell loss. In this graph the no. of virtual circuits is less as compared to next graph shown in fig.-5.10. We observed that with per VC accounting, increasing throughput so that each VC gets a little more than its fair buffer share can actually overall fairness. The reason is that when the cells from the faster VCs are discarded, the cells from slower VCs have more opportunities to occupy the buffer.

As a concluding remarks of results parameters related to TCP UBR with other traffic like CBR, VBR and ABR the following two results observed.

- When congestion is moderate and no buffer overflow occurs at the ATM switches, TCP/IP UBR has similar performance as ABR. However, compared with ABR, the good performance of TCP/IP UBR comes at the expanse of larger switch buffer size.
- TCP over UBR may not be able to provide fair bandwidth allocations among TCP sessions even with simple peer-to-peer configuration in a LAN environment, especially when the networks are very congested. Furthermore, TCP over UBR may experience the "beat-down" problem while TCP over ABR does not.

CONCLUSION AND FUTURE WORKS

In this dissertation efforts have been put on developing a simulation tool of ATM network with different traffic or service schemes like CBR, VBR, ABR and TCP/IP UBR based traffic. The performance of throughput, delay, link utilization, bandwidth (buffer) utilization etc have been shown by some graphs and discussed. After analyzing the various results of network we saw that the some significant parameters like delay, throughput, and buffer utilization is in permissible limit of a standard ATM network.

As a future expansion of this ATM simulation tool we can add so many other components like; different ATM application, like CBR MPEG connection, ABR TCP/IP connection, ABR Self similar traffic connection etc. And if we want more and more precise results of various network component parameters like delay, link utilization, buffer allocation, throughput rate of virtual circuits, protocol overhead etc, then we can use Rate based ATM Switch and Rate based Broadband Terminal Equipment (BTE), in which we can add so many parameters like various congestion control schemes, Peak Cell Rate (PCR), Minimum Cell Rate (MCR), Tagged Cell Rate, Available Cell Rate (ACR), collision detection and avoidance algorithm based more parameters. We also can considered the ABR and UBR service with early packet discard (EPD) schemes for supporting data application in ATM networks. In addition we can also add some mechanism for hybrid fiber coaxial network to support this ATM network simulation tool, so that we can also study the various other activity of hybrid fiber network that is generally used for cable TV distribution.

REFERENCES

1. F.Bonomi and K.W.Fendick. “The rate-based flow control framework for the available bit rate ATM service”. IEEE Network,9(2):25-39,March-April 1995.
2. Hongqing Li, Kai-Yeung Siu, Hong-Yi Tzeng. “TCP Performance over ABR and UBR Service in ATM”. IPCCC’96 March 1996.
3. R. Sanchez,“Virtual ATM Switch Driver for ATM on Linux”.http://www.ittc.ukans.edu/ rsanchez/software/vswitch.html.
4. Sean B House, Shyam Murthy, Douglas Niehaus, “Proportional Time Simulation of ATM Networks”. Information & Telecommunication Technology Department of Electrical Engineering and Computer University of Kansas.
5. Dorgham Sisalem, Henning Schulzrinne. “Congestion control in TCP: Performance of binary congestion notification enhanced TCP Compared to Reno & Tahoe TCP”.Oct.1995,CEC R2116 TOMQAT.
6. Krzysztof Walkowiak , Andrzej Kasprzak, Wladyslaw Budynkiewicz. “Simulation of virtual path routing in survivable ATM Network”. Proceedings of ASIS 1998, Ostrava, 1998, p.p. 229-234.
7. Jorg Abarca pereda. “Simulation of an ATM based PNNI switch system”. Master’s thesis report, Royal institute of technology, Sweden, September 1997.
8. George S. Fishman, “Principles of Discrete event simulation”: Wiley Interscience publication, 1988.
9. Youlu Zheng, Shakil akhatar, “Networks for computer scientists and engineers”. Oxford University press, 2002.
10. Nabajyoti Barakakati, “X Window system programming”, second edition. Prentice hall of India, 2001.
11. Rainer Handel, Manfred N Huber, “ATM networks, concepts protocols applications”. Pearson education Asia, 2002.

12. A.S.Tanenbaum, "Computer Networks" 3rd edition, PHI India, December 1998.
13. Martin, J., and J.Leben, "Asynchronous transfer mode: ATM Architecture and Implementation". Englewood Cliffs, NJ: Prentice Hall, 2000.
14. Stalling William, "Data and computer communications", PHI, Fourth edition, 1994 Alberto Leon-Garcia, Indra Widjaja, "Communication Network", Tata McGraw Hill, 2000.
15. Alberto Leon-Garcia, Indra Widjaja, "Communication Network", Tata McGraw Hill, 2000p.p. .
16. Sumit Kasera, Pankaj Sethi, "ATM Networks-Concepts and Protocols", Tata McGraw Hill, 2001.
17. Simulation Modeling & Analysis, Averill M. Law, W. David Kelton, McGraw Hill Int., second edition, 1991.
18. Website of ATM FORUM, www.atmforum.com.

Appendix A:

Format of SAVE file.

The SAVE file conserves information about the components, including their screen position, the values of their input parameters, their interconnection with neighboring components, and the established routes (virtual circuits). Note that it does not preserve values of output parameters or data logging instructions.

The listing below is an example of a SAVE file. There are three distinct types of information in the file - component descriptions, linkages, and route definitions. The component descriptions come first. The first line of each description 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. The lines immediately following are a listing of the input parameters and their values. Any text on a line after a pound sign (#) is a comment; the comment identifies the parameter.

Following all the component descriptions are the linkages. Each line of this group begins with the keyword *neighbor*1, followed by a component's name in single quotes, and then 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'. The last group of lines in the file is the route listing. Each line begins with the keyword *route*1, 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.

Sample SAVE file:

```
component 'switch1' SWITCH 417 341
param 'switch1' # switch
param 0        # Delay to process a cell (Sec): 0
param 155      # Switching Slot time (Mbit/s): 155
param 10000    # Output q_size : 10000
param 550      # High Threshold for Q Congestion Flag: 550
param 450      # Low Threshold for Q Congestion Flag: 450
param 1        # Logging every (ticks) (e.g., 1, 100): 1
component 'host1' BTE 331 452
param 'host1' # host1param 50 # Max Output Queue Size: 50
```

Appendix B:

Format of SNAP file

The SNAP file contains all the configuration information of a SAVE file plus additional information that reveals the status of the simulated network at a particular point in time, i.e., when a "snapshot" of the simulation has been made. At the top of the file are two lines starting with the pound sign (#). The first line records the seed used for that particular simulation run. (This line will not be loaded or used if the file is used as a configuration file.) The second line records the time (in ticks) when the snapshot was taken.

```
# 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 (Sec): 0
param 155 12 0   # Switching Slot time (Mbit/s): 155
param 10000 12 0  # Output q_size : 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  4 #Cells Received: 0
pflags  4#Cell Drop %: 0
pflags  4#Cells in VBR Q to link1: 0
pflags  4 #Cells dropped in VBR Q to link1: 0
pflags  4 #Cells in ABR Q to link1: 0
pflags  4#Cells Dropped in ABR Q to link1: 0
pflags  1 #Congestion for Link link1: FALSE
pflags  4#Cells in VBR Q to link2: 0
pflags  4#Cells dropped in VBR Q to link2: 0
component 'host2' BTE 562 460
param 'host2' 32 0 # host2
param 50 12 0 # Max Output Queue Size: 50
param 1 12 0 # Logging every (ticks) (e.g. 1, 100): 1
pflags  4 562 424 159 93 #Cells Received: 0
pflags  4 #Cells in VBR Q to link2: 0
pflags  4 #Cells dropped in VBR Q to link2: 0
```