ATMSIM: A Simulator for ATM Networks

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

Master of Technology

by

Somanchi Brij Vinod

under the guidance of

Dr. Dheeraj Sanghi

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

April 1995

Certificate

It is certified that the work contained in this thesis entitled **ATMSIM : A Simulator for ATM Networks** has been carried out by Somanchi Brij Vinod (Roll No: 9311125) under my supervision, and that this work has not been submitted elsewhere for a degree.

Dr. Dheeraj Sanghi
Assistant Professor,
Department of Computer Science and Engineering
IIT, Kanpur

April 1995

Abstract

The existing simulators cannot provide an environment for simulating various protocols related to the ATM networks. Setting up an ATM network, which involves fiber optic cable and switches, is costly. The performance analysis of the ATM networks for different kinds of resource reservation and congestion control schemes, is required to decide whether to implement a particular scheme in reality. Thus, the need to develop a simulator for ATM networks has been felt. Discrete-event simulation technique offers the greatest degree of flexibility in modeling a simulator for an ATM network. Due to this reason ATMSIM, the simulator for ATM networks, is developed using discrete-event simulation technique. ATMSIM allows the user to define a network configuration consisting of a physical network and specifications of the components of the network. ATMSIM provides both steady-state and instantaneous performance measures to facilitate the study of complex dynamics that arises in ATM networks. ATMSIM has been used as a test bed for emulating a virtual reality model for ATM networks and to study the above mentioned schemes.

Contents

List of Figures

Chapter 1

Introduction

1.1 Simulation

Simulation plays an important role in computer-aided analysis and design of any system. Simulation technique has often been used to model and design communication networks. Before we build a system, it is important to evaluate the performance of various designs. In case of complex systems, it is difficult to build an analytical model that is mathematically tractable. In such cases, simulation of the system is often the only technique for performance evaluation.

The simulation of large, complex, high-speed networks presents unique problems, such as setting up networks of different topologies and evaluating their performance parameters[16]. These problems can be addressed using statistical simulation techniques, such as importance sampling, distributed computing techniques. Thus, simulation has been playing an increasingly important role in the modeling and design of communication networks.

1.1.1 Advantages and drawbacks of simulation

The advantage of a simulator, having modeling constructs closely related to the components of communication networks, is that development time of a target network becomes considerably small.

The major drawback of basic network simulators is that they are limited to modeling only network configurations allowed by the package's standard building blocks. Thus, if a communication system has some specific features which are difficult to model, they have to be approximated to preserve its behaviour. However, there are several simulators that allow existing modeling constructs to be modified or new constructs to be created, with increased modeling flexibility.

1.1.2 Types of simulation techniques

In this section we discuss about various simulation techniques available.

Process emulation Every component of the target system, i.e., the computer network to be studied, is represented by a separate process. Each process executes code corresponding to the code of its component. An advantage of process emulation is that all components of the target system can be tested. However, in order to faithfully capture the dynamics of the target system, it is necessary to periodically synchronize all the processes. The overhead involved in this process synchronization becomes a serious disadvantage.

Discrete-event simulation In discrete-event simulation, the target system is modeled by a set of variables and a set of events[11][17]. Associated with each event is a routine that can update the set of variables. A simulation of the target system corresponds to a succession of event occurrences of time, i.e., a sequence of the form (e_1, t_1), (e_2, t_2), ..., (e_n, e_n), where (e_i, t_i) denotes the execution of event e_i's routine at time t_i. The simulator executes a loop, with one event occurrence simulated in each iteration. It maintains the following two variables : *simulated time*, indicating the time of the last event simulated; and *event list*, consisting of a set of (e, t) pairs, where e refers to an event and t refers to the time instant when e is scheduled. In each iteration, the simulator removes an (e, t) pair with the earliest time from the event list, updates the simulated time to t, and executes the routine corresponding to event e. The simulation of event e can schedule new events, i.e., add new $(e^{'}, t^{'})$ pair(s) into the event list.

Rare-event simulation This technique is used to measure quality-of-service (QoS) [9]. The most common example is cell loss in ATM networks. As it becomes computationally costly to use conventional simulation techniques for finding the occurrence of a very rare event, this technique has been formulated. Important sampling[13] technique is a rare-event simulation technique. In this the probabilities that govern the outcomes of the simulation are modified in such a way that the original low-probability events, governed by the reference probability density function, occur more frequently under biased probability density function, i.e., instead of simulating low-probability events under the reference probability density function, relatively high-probability events are simulated under appropriately biased probability density function, and the results are weighted to compensate for the bias.

1.1.3 Desired features of a simulator

A simulator should have the following features.

- Flexibility in modeling the target system, ease in development of model parameters and fast model execution speed.
- Addition of new schemes, algorithms, and components of the target system should be facilitated depending on the requirement and so, a simulator should be modular in structure.
- Since all communication networks exhibit some sort of random behavior, a simulator must contain good statistical capabilities, such as random-number generator with multiple random-number streams and a standard probability distribution.
- The user should be able to specify various kinds of topological and network dependent parametric changes in the network and observe the system performance.
- It should provide standard performance measures such as average delay, average throughput, average load, as well as more specialized measures such as

number of voice cells traveled through a link, and number of times a request for connection setup has failed, etc. It should also provide both average and instantaneous measures. Instantaneous measures (e.g., link utilization during the last 5 msecs) are needed to observe the system's adaptability to radical changes (e.g., buffer overflow)

1.1.4 The output of a simulator

It is desirable for simulators to generate standard reports for commonly occurring performance statistics, such as throughput, utilization, end-to-end delays, and number of buffers allocated. The software should allow tailored reports to be developed easily, including user-defined statistics that are specific to a particular application. Ideally, a simulator should provide a graphical interface to enable the user to obtain a variety of graphical plots of his/her simulation data, which have been stored in a database.

1.2 Existing Network Simulation tools

Several tools are already available for simulating a network. Some of these tools have been built by researchers to simulate a specific class of network that they are interested in studying, while others are general purpose tools that can simulate a wide variety of networks. We give a glossary of available tools below.

MaRS This has been developed at University of Maryland[6]. It provides a flexible platform for the evaluation and comparison of routing algoritms.

NetSim This has been developed at MIT. It is designed for studying various flow control mechanisms.

COMNET II.5 This is a commercial package similar to NetSim, developed by CACI Products Company.

DeNet (Discrete Event Network) This has been developed at the University of Wisconsin-Madison. It is a general purpose simulator that simulates a set of

communicating objects.

Real Real has been developed by S.Keshav of University of California. It is a general purpose simulator which takes source and router parameters, network topology and link speeds as input and generates congestion window sizes, throughput, queue lengths and link utilization and generates a number of packet drops and retransmissions.

Vince Vince is a software developed at the Naval Research Laboratory. Its primary function is to perform ATM signaling and virtual connection (VC) management tasks. Vince is a testbed for ATM using TCP/IP connections, over which multiple hosts can communicate with each other.

BONeS DESIGNER It is a graphically-oriented, general-purpose simulation language that contains many features for modeling communications networks[12]. The major building blocks are data structures and block diagrams. In this simulator, after a simulation has been executed, a user may develop tailored output reports that are displayed in the form of graphical plots.

BONeS PlanNet It is a simulator for LANs whose major building blocks are LAN segments, traffic models, interconnects, and a WAN mesh. PlanNet produces a standard output report that is displayed in the form of graphical plots; however, numerical results can also be obtained. The PlanNet modules can be modified using BONeS DESIGNER, provided that the user has the Hierarchical Version of the PlanNet modules.

COMNET III It is an object-oriented simulator for LANs and WANs. The major building blocks(objects) are nodes, links, protocols, and traffic generators. COMNET produces standard output reports, which include such statistics as average end-to-end delay, throughput, and utilizations.

L.NET II.5 Formerly called LANNET II.5, it is a simulator for LANs. The major building blocks are LANs, stations, and gateways(a generic device for connecting LANs). After a simulation run has been executed, L.NET provides the user with a variety of standard reports that include such information as messages transmitted, number of collisions, utilizations, and average end-to-end delay.

NETWORK II.5 It is a simulator for computer systems and networks whose major building blocks are hardware devices and software modules. Hardware devices include processing elements, transfer devices, and storage devices. It provides the user with a set of standard output reports, which include device utilizations, average end-to-end delay, and information on the execution of software modules.

OPNET Modeler It is a communications-oriented simulation language that uses Network, Node, and Process Editors to build a simulation model. The Network Editor is used to specify graphically the network topology consisting of nodes and fixed-position links. The Node Editor is used to describe graphically the data flow between modules (hardware and software systems) in a node. Module types include processors, queues, and traffic generators. OPNET can produce tailored output reports that include a wide variety of numerical results and graphical plots.

SES/workbench It is a graphically oriented general-purpose simulation language that contains features for modeling computer systems and communications networks. The major building blocks are nodes, arcs and transactions. To build a model in a workbench, one defines a transaction that corresponds to a message. The user then develops a directed graph consisting of nodes and arcs, which describes how transactions flow through the network. It can display simulation results in both numerical and graphical formats, either during the simulation or after it has been completed.

1.3 Overview of ATMSIM

In the context of ATM networks, none of the above mentioned simulators can be used because ATM networks have certain unique features when compared to the other networks[7][14][1][5]. The features that make the ATM networks different from other networks are given below.

- ATM is a connection oriented network and deals with the concept of virtual connections.

- In ATM networks routing at the ATM switches is done using routing table which have been set up statistically at the time of setting up of the network or dynamically depending on the virtual connections going through the ATM switches.

- In ATM networks, depending on the application, traffic shaping may be needed to be done at every switch of a virtual connection.

- ATM networks support different classes of traffic. So, the simulator for ATM networks needs to provide traffic modeling features.

The performance analysis of ATM networks, for various resource reservation and congestion control schemes[4][8][2] is required to decide whether or not to implement a particular scheme in reality. Setting up an ATM network, which involves fibre optic cable and switches, is costly. Thus, the need for development of a simulator for ATM networks has been felt. The unique features related to ATM networks as well as the features desirable by a simulator are provided by ATMSIM. ATMSIM takes network related parameters as input to set up ATM networks with different topologies, to measure the performance parameters and to output these parameters periodically.

The features of *vince* are very close to those required by a simulator for ATM networks. In *vince* source modeling has not been done. Source modeling is very necessary to generate cells to do performance analysis on a variety of above mentioned schemes on ATM networks. *Vince* uses process emulation to perform signalling and virtual connection management. As discussed in Section 1.1.2, time synchronization becomes a problem in process emulation technique. Discrete-event simulation technique offers the greatest flexibility in modeling a simulator for ATM network. Due to this reason ATMSIM is developed using discrete-event simulation method.

Entities defined by ATMSIM include switches, links, virtual connections and cells. These entities are represented internally by data structures, which are dependent on the specifications of the schemes using ATMSIM. For example, a data structure representing a cell would contain the elements required by the ATM networks along with the simulation specific information. A cell-creation time stamp, for example, can be used to collect statistical information. ATMSIM contains a user interface module to initialize the data structures, an initialization mechanism

to establish the initial system state, network processing routines and statistical collection routines to obtain measurements. It also contains a coordinating program to control the event list and initiation and termination of the simulation.

1.4 Organization of the Thesis

- Chapter 2 discusses the design aspects of ATMSIM. Some of the algorithms that are used during implementation are also discussed.
- Chapter 3 discusses different aspects of implementation of ATMSIM in detail.
- Chapter 4 portrays the accomplishments of the thesis and proposed enhancements.

Chapter 2

Design of ATMSIM

2.1 Introduction

ATMSIM is a simulator for ATM networks. So, it should contain the features to model ATM networks as well as the desirable features of a simulator. ATMSIM has been designed to support all these features. It takes the input related to an ATM network and sets it up. It simulates the unique features of ATM and monitors performance parameters periodically.

ATMSIM is organized into three modules. The User Interface module reads network information from an input file. The Initialization module initializes the event queue and other relevant data structures. The Simulator module deals with the features related to ATM networks and gathers the performance parameters for evaluation.

2.2 User Interface Module

This module reads the network dependent parameters from a user-defined input file and puts them into the corresponding data structures. The user has to specify the input file in a format described in Appendix A. The user defined input file includes a description of the network to be simulated. It consists of three parts. The first part is a list of all components with their parameter values. The second part indicates the

interconnection of these components. The third part lists the parameters which have been monitored, the periodicity of monitoring them, and the files into which their values have to be stored. Depending on input requirements of the protocol being developed on ATMSIM, the information regarding the network can be changed. The input file is parsed to read the network related parameters. Parsing is done using the grammer rules which describe the input file.

2.3 Initialization Module

In this module, we deal with the following three aspects.

1. Data structures are initialized, using the values read from the input file.
2. The database to store the values of the parameters to be monitored periodically is initialized.
3. The events needed to start the simulation process are put in the event queue. The algorithm for this process is given in Figure 1.

In ATM networks, the transmission of information payload starts with the formation of cells. In order to reflect the features of ATM networks, simulation process should also start with generation of cells at each source. For each source, the generation of cells event is put in the event queue to initiate the process of simulation. This is scheme dependent. For example, the scheme discussed in [3] requires the virtual connections to be setup before generation of cells. In such schemes, connection setup event should be put in the event queue to initiate the simulation process and only on successful setup of connection, the generation of cells starts. Along with the above mentioned events which are used to reflect the features of ATM networks in the simulator, the events used to monitor performance parameters periodically, are also put in the event queue in this module. In the next section, we will discuss about the way in which events have been discretized.

```
InitEvQ()
{
    for each source in the network
    {
        get information about the source and the start switch of
            the virtual connection corresponding to the source;
        get the time at which source can start generating cells;
        put cell-generation event timestamped with the above mentioned time,
            along with the info. regarding the source and
            the start switch, into the event queue;
    }
    for each switch
    {
        get the switch information and put in
            the switch output event structure;
        put this event structure into the event queue with
            the start time of the simulator as timestamp;
    }
    for each link
    {
        get the link information and put in
            the link output event structure;
        put this event structure into the event queue with
            the start time of the simulator as timestamp;
    }
    for each virtual connection
    {
        get the virtual connection information and put in
            the virtual connection event structure;
        put this event structure into the event queue with
            the start time of the simulator as timestamp;
    }
}
```

Figure 1: Algorithm for initialization of event queue

2.4 Simulator Module

In this module, events are removed from the event queue, till either the simulation time expires or the event queue becomes empty. The events are simulated which might result in adding more events to the event queue. This process continues. The event data structure contains a timestamp, event type, a routine corresponding to the event and the information that has to be passed to the routine. At the source switch, ATM layer takes the information payload from the upper layers, adds header fields including switching information to the payload, and sends the cells to the physical layer[5]. A source generates cells and an ATM switching element updates the cell header values and puts them into the network. It is assumed in this design that an ATM switch can act as a source. So, two basic events are used to reflect the functionality of ATM networks.

1. Generation of cells
2. Sending cells (for switching at the switches).

In order to monitor the performance parameters and to collect statistical results, some more events are required. Three kinds of events are recommended for monitoring the performance parameters periodically at each *switch*, *link* and *virtual connection* (VC).

The three performance monitoring events are

1. Output information related to each switch.
2. Output information related to each link.
3. Output information related to each VC.

2.4.1 Generation of cells

The algorithm for generation of cells is given in Figure 2. A given source is checked to see whether or not it is alive. If the source is alive, cells are generated depending on the source characteristics. The simulation specific information, such

```
GenCells()
{
    get the information about source and switch;

    if source is alive
    {
        generate cells based on the source characteristics;
        if cells are generated
        {
            find the VC buffer corresponding to the source;
            put the generated cells in the VC buffer;
            find the outgoing link for the VC;
            put the values of the current switch info. and
                the outgoing link info. in the event structure;
            put a sendcells event, with the current time
                as timestamp, into the event queue;
            put a generation of cells event, with the next generation time
                as timestamp and the source info. and the current switch info.,
                into the event queue as event info.;
        }
    }
}
```

Figure 2: Algorithm for generation of cells

as cell-generation time and queuing delay experienced by a cell, is initialized in the generated cells before being put into the corresponding VC buffer. In case the source generates cells, a send-cell event for switching these cells from the current switch to the next switch is put into the event queue, timestamped with the current time. A send-cell event needs information about the link through which the cells are to be sent. So, the link information is also specified in the event. A cell-generation event is also put in the event queue timestamped with next generation time of the source. Along with the timing information, information regarding the current source and the corresponding start switch is also put in the event.

2.4.2 Send cells

The routine corresponding to this event switches cells. The algorithm for doing this is given in Figure 3. It is checked whether a given link is capable of sending cells. If the link cannot send cells, since it might be transmitting cells at that time, the time at which the link becomes free to transmit cells is obtained. A send-cell event is put in the event queue with the next send time as timestamp, along with information regarding the current link.

Duration for which the cells in the virtual connection buffers can be sent is calculated. A send-cell event is put the event queue due to the transmission of cells through a virtual connection. For speeding up the simulation process, instead of sending cells of only that virtual connection because of which the present send-cell event is put into the event queue, all the virtual connections passing through the link are taken into consideration. Bandwidth is allocated to each virtual connection through the link, depending on the bandwidth allocation policy[10][19][18].

The cells of each virtual connection are put into the VC buffer in the next switch. Simulation related information in the cells is updated while being put into the next switch. The outgoing link of the next switch, through which the cells are to be sent, is determined. A send-cell event is put in the event queue with the sum of current simulator time and propagation value of the current link as timestamp and information regarding the next switch's outgoing link is put into the event which is used as input to the corresponding routine.

```
SendCells()
{
    /* switching of cells */
    get the information about switch and
        the outgoing link on which cells are to be sent;

    if outgoing link can not send cells at the current time{
        put the values of the current switch info. and
            the outgoing link info. in the event structure;
        put a sendcells event, with the time
            at which link can send cells next
            as timestamp, into the event queue;
    }
    calculate duration for which the cells can be sent;
    allocate band width depending on the band width allocation policy;
    for each VC{
        update the simulation related infromation
            in the cells being sent;
        put the cells to be sent in the corresponding
            VC buffer in next switch;
        if next switch is not the destination{
            put the next switch info. and its outgoing
                link's info. in the event structure;
            put a send cells event in the event queue timestamped
                with summation of propagation delay of the
                present outgoing link and the current time
        }
        if some cells left in the VC buffer to be sent{
            put the present switch info. and the outgoing
                link's info. in the event structure;
            put a send cells event in the event queue
                timestamped with summation of
                the send duration and the current time
        }
    }
    allocate buffer for the cells left in the VC buffers;
    drop those cells that could not be buffered;
    update the switch output parameter information;
    update the link output parameter information;
    update the simulation specific link information;
}
```

Figure 3: Algorithm for sending cells

If the cells of a given virtual connection that could not be sent due to bandwidth allocation problems, a send-cell event is put in the event queue with the sum of current time and send duration as timestamp. Such cells are buffered in VC buffers according to the buffer allocation policy used by the current switch. The cells, that could not be buffered, are dropped according to the cell drop policy of the current switch[15]. The performance parameters pertaining to the switch and the link are updated. Simulation related information of the link is also updated.

2.4.3 Output switch information

Depending on the performance parameters to be stored in the output file, the average and instantaneous values (values obtained for a period of time, between time the parameters have been stored last time and current time) are stored in the output file for a given switch along with the name of the switch and the current time. Output of performance parameters is done periodically as specified by the user through input file. The next time at which the switch has to output the performance parameters, is calculated. A switch output event is put in the event queue, with the next output time as timestamp, and the current switch information which is needed by the event routine.

2.4.4 Output link information

Gathering of the performance parameters for links is similar to that of switches, except that the switch information is replaced by the link information.

2.4.5 Output virtual connection information

In this module, the cells received by the destination switch of a virtual connection are processed and performance parameters of the virtual connection are measured. Processing at destination switch of the virtual connection is application dependent. When an application provides the existence of higher layers, cells reaching the destination have to be sent to the upper layers for further processing[7]. In such cases,

processing of cells at the destination and updating information of the virtual connection can be seperated from this module. But, in this design such features have not been considered. Simulation specific parameters, like queueing delay contained in each cell, are used to calculate average and instantaneous values. The average and instantaneous values of throughput for the given virtual connection are calculated using the number of cells reaching the destination. Other performance parameters like maximum and minimum queuing delays of the cells are updated. Depending on the performance parameters to be monitored, the corresponding values are stored in the output file. Finally, a virtual connection output event is put in the event queue with the next monitoring time of the virtual connection as timestamp and the information regarding the current virtual connection as input for the routine corresponding to output VC event.

2.5 Conclusion

In this chapter we described the different modules of ATMSIM. It can be used to simulate a variety of schemes for finding out the performance of ATM networks, given the network related information. In the next chapter, we will be describing the implemetation aspects of ATMSIM.

Chapter 3

Implementation of ATMSIM

3.1 Introduction

The implementation of simulator can be broadly divided into three modules, namely, user interface module, initialization module, and simulation module.

Data structures used in ATMSIM are discussed in Section 3.2. Implementation details of the user interface module are discussed in Section 3.3. The implementation aspects of initialization module are discussed in Section 3.4. Various implementation aspects of the simulator module are discussed in Section 3.5. We conclude this chapter in Section 3.6.

3.2 Data Structures Used in ATMSIM

In this section, we discuss some important data structures.

Event Queue : As the number of events that are going to be in the event queue are not known at the beginning of the simulation, the event queue is represented as a linked list of event structures. The pointer to the first event of the event queue is maintained globally.

The data structure of an event is given below.

```
typedef struct eventqueue {
  /* keeps the events in the ascending order of time */
  void         *evInfo;
  /* info. for the event specified in the eventType */
  event_e      eventType;
  /* type of the event */
  void         (*Event)();
  /* this is the function that is going to be called
  /* depending on the type of the event */
  time_t       *time;
  /* time at which the event occurs */
  struct eventqueue *nextEv;
} eventQ_t;
```

Set of switches : The set of switches is represented as an array of switch structures. Each switch is identified by the index number in the array. Memory for this array is dynamically allocated depending on the number of switches in the network. The data structure of a switch is given below.

```
typedef struct switches {   /* info. about each switch */
  char           *swName;
  int            noOfLinks; /* no. of out going links */
  outLink_t      *outLink;  /* info. of outgoing links */
  bwPolicy_t     *bwPolicy; /* bandwidth allocation policy */
  baPolicy_t     *baPolicy; /* buffer allocation policy */
  int            noOfSrcs;  /* no of sources for this switch */
  source_t       **source;  /* information of source */
  lookUpTable_t  *luTable;  /* lookup table for vpi and vci */
  swOp_t         *swOp;     /* Info. of switch for output */
} switch_t;
```

Set of Outgoing links of each switch : These are represented as an array of outgoing link structures. Depending on the number of outgoing links of each switch the array is dynamically allocated. The link information field of the outgoing link structure is a pointer to a link structure. The *bufferQ* field of the outgoing link's data structure contains an array of virtual connections, memory for which has been allocated dynamically using the number of virtual

connections going through the outgoing link. The data structure for representing outgoing link is given below.

```
typedef struct outlink {/* info. regarding out link */
  linkBuffer_t *bufferQ;/* buffer queue of the outgoing link */
  link_t       *link;   /* this points to the corresponding
                         * link in the array of links */
} outLink_t;
```

Set of sources in a switch : This is represented as an array of pointers, each pointing to the corresponding source structure. It is allocated dynamically using the number of routes. The data structure of the source is given below.

```
typedef struct source {
  /* info. for the source if the switch has a source */
  int         sourceId;   /* the source identifier */
  time_t      *liveTime;  /* amount of time
                           * the source can be alive */
  time_t      *startTime; /* time at which
                           * it can start generating cells */
  int         vpi :12;    /* virt. path identifier for
                           * the out link */
  int         vci :16;    /* virt. conn. identifier for
                           * the out link */
  int         linkNo;     /* out going link no.
   * corresponding to vpi,vci */
  sourceChar_t *sChar;    /* characteristics of the source and
   * the virtual connection */
  struct route *route;    /* route for this source */
} source_t;
```

Cells : The cells in the VC buffers are always represented as a list of cell structures. In order to decrease the frequency of memory allocation for cell structures during the simulation, these cells are maintained as an available pool of cell structures. The data structure used for representation of cells is given below.

```
typedef struct buffercell { /* info. for each buffer cell */
  int  PT  : 3; /* payload type (data, audio or video) */
  int  CLP : 1; /* if CLP is 1 cell can be dropped */
  int  sTmsec;
  int  sTpsec;  /* time when cell started from its source*/
```

```
  int  rTmsec;
  int  rTpsec;  /* time when it has to received by switch */
  int  qDmsec;
  int  qDpsec;  /* the amount of time the cells is queued */
  struct buffercell *next;
} bufferCell_t;
```

Set of links : The set of links is used in the form of an array of link structures. These are the structures to which the pointers in the outgoing link structures point to.

```
typedef struct link {
  /* info. regarding each link in the network */
  int     linkId;    /* link identifier */
  int     headSw;    /*head switchNo. of the connection*/
  int     tailSw;    /*tail switchNo. of the connection*/
  int     bufSize;   /* no. of buffers allocated to the link */
  float   bandWidth; /* the band width of the link */
  float   allowedBW; /* bandwidth that a link can allocate */
  time_t  *propDelay;/* propagation delay of the link */
  time_t  *idleTime; /* idle time of the link */
  time_t  *lastSent; /* time the last cell was sent */
  time_t  *nextSend; /* used in sendCells event time at which
                      * next chunk of cells have to be sent */
  linkUtil_t *linkUtil; /* to gather performance parameters */
} link_t;
```

Events : Various event structures are distinguished by the information content stored in the event information field. So, an array of list of available events of various event types is constructed in the beginning of the simulation. In the next section, we will discuss about the data structure used to provide information to the event routines through *evInfo* field for the event types cell-generation, output switch information, output link information, and output virtual connection. The data structure used to provide information to the event routine corresponding to send-cell event is given below.

```
typedef struct sendcell {
  int switchNo; /* switch on which event is performed */
  int linkNo;   /* no. of links thru which cells are sent */
} sendCell_t;
```

VC output information : VC output information is represented in the form of array of vcOpInfo_t structures. It is given below.

```
typedef struct vcInfo {/* info. about the VC */
  /* VC is identified by the start switch, source
   * no. in that switch and the dest. switch */
  int    QoS;    /* quality of service for VC */
  int    vcInd;  /* virtual connection identifier */
  int    stSw;   /* start switch of VC */
  int    destSw; /* destination switch of VC */
  int    totalCells;  /* total no. of cells that reached
                       * destination upto a particular moment*/
  time_t *avge2eDelay;/* avg end to end delay of connection */
  time_t *mine2eDelay;/* minimum of all the e2e delays */
  time_t *maxe2eDelay;/* maximum of all the e2e delays */
  time_t *avgQDelay; /* average queuing delay */
  time_t *minQDelay; /* minimum of all the queuing delays */
  time_t *maxQDelay; /* maximum of all the queuing delays */
  thruput_t *thruput;/* thruput of the VC */
  int    alive;      /* this flag specifies whether the source
                       /* corresponding to VC is alive or not */
} vcInfo_t;
```

There is a correspondence between a VC, a route, and a source. So, they are identified by a VC index.

Time : In a format similar to events, time information is maintained as a pool of time data structures represented in the form of an dynamic array of pointers to the time structure. Time values of the simulator are stored in the data structure given below.

```
typedef struct {       /* timing info. in msec and picosec */
       int     msec;
       int     picosec;
} time_t;
```

The reason for dividing the time value into milliseconds and picoseconds is to get better accuracy.

Output file information : The output files for each switch, link and VC are represented in the form of an array of file pointers.

3.3 User Interface Module

In the user interface module the general purpose routines compatible to *lex* and *yacc* for parsing an input file for a certain parameter value are implemented. So, this module is specific to the parsing rules of the input file format. But, changing the parsing rules will make it useful for various input file formats. A recursive descent parser is implemented using the grammar rules specifying the input file format. The grammar rules regarding the user defined input file are given in Appendix A. Depending on the grammar rules, it reads the description about the network to be simulated. The read values are put into the corresponding fields of the data structures. This parser can be used to read any data from a given input file by changing the grammar rules of the input file.

3.4 Initialization Module

This section deals with the following implementation aspects.

3.4.1 Initialize output files

For storing the performance parameters that are monitored periodically, the output files are opened in the routine OpenOutputFiles().

This routine takes the output files for each switch, link and virtual connection and opens them using fopen(), a standard C function to open an ASCII file, and puts them at the corresponding position indexed by the identification number in the array of file pointers. While opening the output files it checks for duplication. The output files are not opened if they have already been opened, for storing the information of another switch or link or virtual connection. After opening the output files this routine puts the information that enhances readability and interpretation of the parameters to be monitored.

3.4.2 Initialize event queue

The process of initializing the event queue is done using the following routines.

InitEvQ() is the routine which initializes the event queue. The events with which the event queue has to be initialized, are dependent on the protocol being implemented. In ATMSIM the events with which the event queue is initialized are generation of cells, output switch information, output link information, and output VC information.

An available pool of events for each event type is implemented in the simulator. GetAvailEvent() takes the event type as input and gives the corresponding event structure.

A generation of cells event is put in the event queue, for each source, timestamped with its start time. For ease of processing identification number of the switch, in which the source is present, and the source identification number are put in the event information field of the event structure. A pointer variable of type ¨**void** *ïs used in the event structure is used for this purpose as pointer as any structure can be pointed to by this field and information regarding a specific structure can be retrieved by casting this field to a pointer to the same. This makes the implementation easier as information needed by the event routines can easily be put into the event structure. By doing this the underlying aspects of implementation are abstracted. The data structure used to provide information for the routine corresponding to cell-generation event is given below.

```
typedef struct gencell {
  int switchNo;/* switch on which event is performed */
  int sourceNo;/* to know about the characteristics of the source
                * for which cells are to be generated */
} genCell_t;
```

For each switch, an output event is put in the event queue, time stamped with the start time of the simulation, as the performance parameters have to be monitored from the beginning of the simulation in ATMSIM. The time, at which monitoring starts, can be changed depending on the scheme. The information needed by the corresponding routine for this event is the identification number of the switch for

which output has to be monitored. The identification of the switch is the index number of the corresponding switch structure in the array of switches. The data structure containing this information is given below.

```
typedef struct swEvInfo {
        int switchNo;
        /* identifier of the switch on which event is performed */
} swEvInfo_t;
```

For each link and VC, the process is same as described above except that the information regarding a switch is replaced by that of the link and VC. The data structures containing this information are given below.

```
typedef struct linkEvInfo {
        int linkInd;
        /* identifier of the link on which event is performed */
} linkEvInfo_t;

typedef struct vcEvInfo {
        int vcInd;
        /* identifier of the VC on which event is performed */
} vcEvInfo_t;
```

The routine putInEvQ() puts the event given as input, into the event queue in the increasing chronological order. This routine checks for duplication of events. Duplication of event in the event queue is checked by comparing the time values and the information contents of the *evInfo* field of the events. In case a duplication does occur, it is not put into the event queue. Thus, InitEvQ() updates both the event queue and the variable pointing to its first event structure.

3.5 Simulator Module

Switchprocessing() routine contains a basic simulator loop in which processing of the cells and monitoring of performance parameters are done. This takes an event from the event queue using GetEvent(), which returns the first event of the event queue and updates the global variable pointing to the first event of the event queue,

if the event queue is not empty and *NULL* otherwise. Throughout the discussion, the event returned by GetEvent() routine is referred to as the current event, and its timestamp as the current time. The simulation is continued till GetEvent() returns *NULL* or time information extracted from the event structure is greater than the simulation end-time.

The routine corresponding to the event type is represented as a pointer to function. After execution of the routine, the current time value is printed to enable the end user to know about the processing of the simulator. In the following subsections we discuss about the implementation aspects of the event routines used in the simulator module.

3.5.1 Generation of cells

The routine corresponding to the cell-generation event is GenCells(). The information needed by the routine is extracted from *evInfo* field of event structure. CheckEndTime() routine compares the life time of the source with the current time of the simulator. If the life time is greater than or equal to the current time, the source generates cells. The *SwitchWithCells* field in sourceChar_t structure is used to point to the routine to be executed depending on the source type. The data structure corresponding to sourceChar_t is given below.

```
typedef struct characteristics {
  /* Characteristics of virt. conn. and switch */
  float     peakRate;       /* peak rate of the source */
  float     avgRate;        /* average rate of the source */
  int       burstSize;
  time_t    *burstDur;      /* Burst Duration */
  time_t    *interBurstGap;
  float     burstTol;       /* tolerance in the size of the burst */
  time_t    *delayTol;      /* Delay tolerance in micro second*/
  float     *(*SourceWithCells)();/* depending on the source type
     /* pointer to the corresponding function is set*/
  source_e  sourceType;
  union {
     video_t           *videoSrc;
     singleVoiceSrc_t  *sVoiceSrc;
     data_t            *dataSrc;
```

```
  } source_u;
} sourceChar_t;
```

The input values that the routine, pointed to by *SwitchWithCells*, takes are the source characteristics. It returns the time at which the source can generate next burst of cells and an array of intercell gap values. The design and implementation aspects of source modeling have been discussed in [3]. The generation time of each cell is put into the cells using the array of intercell gap values and the current time. This is done in the routine MakeCells() which takes an array of intercell gap values as input and returns a linked list of updated cell structures. This routine also initializes queuing delay of the cells to zero. The VC buffer, to which generated cells belong, is obtained and the cells are put into it. The initial values of the VCI, VPI and incoming link identification number are used to determine the VC buffer. The reason for not using VCI, VPI and link identity number in the cell header is discussed in the next section.

The *nextArrivalTime* field of the VC buffer data structure is set to the next generation time. Identification number of the outgoing link through which the cells have to be sent is determined. This value along with identification number of the current switch is put in *evInfo* field of the event structure. Depending on whether or not cells are generated in the source, a send-cell event has to be put into the event queue with the current time as timestamp. A generation of cells event is put into the event queue with the next generation time as event timestamp and the source identification number and the corresponding switch identification number as information to the GenCells() routine.

3.5.2 Send cells

The process of switching(sending) cells is done in the SendCells() routine.

From *evInfo* field of event structure the information needed by the routine is extracted. In the link structure *nextSendTime* field specifies the time at which it can send cells next time. This time is compared with the current time. The current link identification number and the current switch identification number are put into the send-cell event structure which is used as input to the routine corresponding to

send-cell event. If *nextSendTime* is greater than the current time, a send-cell event is put in the event queue with *nextSendTime* as timestamp.

The amount of time for which cells can be sent through the link is calculated using GetSendDur() routine. The *nextArrivalTime* field in the VC buffer structure specifies the time at which next chunk of cells will be received by the VC buffer. The *nextArrivalTime* of each VC buffer in the link, is used in the calculation of send duration. This is used to speed up the process of simulation. At the source switch, next generation time and at intermediate switches, propagation delay of the incoming link of a switch for a particular virtual connection and the duration for which cells have been sent through the link, is used to set the *nextArrivalTime* of the virtual connection. The minimum value of next arrival time of all the virtual connections is taken as send duration.

Bandwidth is allocated for each virtual connection going through the link depending on the bandwidth allocation policy of the current switch[10]. Depending on the bandwidth allocation policy of the switch, the corresponding routine is put in the associated policy structure. This routine takes the buffer queue structure of the outgoing link, and the link identification number as input. It updates the amount of time cells can be sent (send duration) and sets the *sendTail* field of each VC buffer structure. It returns the send duration and the total number of cells to be sent on the link. From the VC buffer structure, the incoming VCI, VPI and link identification numbers are obtained. The reason for maintaining these values in the VC buffer structure is to lessen the complexity of the simulator. All the cells being put into the VC buffer have the similar incoming VCI, VPI and link identifier values. This makes the processing easy in a way that the cells need not be updated whenever they are switched from the current switch to the next. The VC buffer and the corresponding outgoing link identifier of the next switch of the virtual connection, are found using the look up tables of the current switch, and the next switch. The cells in each VC buffer of the current switch are put into the corresponding VC buffer of the next switch. The time at which each cell is going to be received by the next switch, and the queuing delay it has suffered till then, are calculated and the corresponding fields are updated in each cell structure. The next arrival time of the corresponding VC buffer structure of the next switch is set to the sum of the

current time and the propagation delay of the outgoing link.

In ATMSIM, processing at the destination is done while outputting the performance parameters of the virtual connection. In a destination the outgoing link is given the identification number as -1. This is used to find out whether the next switch is the destination of a specific virtual connection. If the next switch is the destination, the send cells event is not put in the event queue. Otherwise the same event is put into the event queue, with the sum of current time and the propagation delay of the outgoing link as timestamp and the next switch identification number and the outgoing link identifier as event information. At least one cell is left in the buffer, if the receive time of the cell pointed to by the head field of the VC buffer is less than the sum of the current time and the send duration. In that case buffers are allocated to all the cells, that should have been sent before the send duration but could not be transmitted due to bandwidth allocation problems, depending on the buffer allocation policy of the switch. The cells that could not be buffered are either dropped or the CLP value in the header is set to 1 depending on the drop policy being used by the switch. And also, a send cells event is put in the event queue timestamped with the nextSendTime of the link along with the current switch and the outgoing link identifiers.

For gathering the performance parameters, such as *buffer uttilization* and *number of cells dropped*, to be output swOp field in switch structure is used. The swOp field is represented by data structure given below.

```
typedef struct outputinfo {
       /* contains output info. of a switch */
       bufferUtil_t   *bufferUtil;
       /* buffer utilization of each buffer queue
        * in the switch */
       lossOfCells_t  *cellsLost;
       /* cell losses in the switch */
} swOp_t;
```

The two fields are again used to store the information about the buffer utilization in the switch and the number of cells dropped in the switch respectively. They give the information about the cumulative and instantaneous number of cells allocated or dropped along with the time at which the information has to be updated in case

of instantaneous cells.

For a link, link utilization is represented in terms of the cumulative and instantaneous number of cells transmitted through the link. The corresponding field in the link structure contains information about the time information as to when the output of the parameters given above is going to be performed.

The number of buffers used in the switch is calculated by adding the number of buffers allocated to the cells of each virtual connection. The cumulative and instantaneous parameters are updated using the calculated values. The number of cells transmitted through the link is returned by the bandwidth allocation routine. This value is used to update the cumulative and instantaneous parameter of link utilization.

The *nextSendTime* field of the link structure is updated with the summation of the current time and the send duration. The fields referred to as *lastSendTime* and *idleTime* of link structure are also updated. SendCells() routine acts as the backbone during the performance evaluation of ATM networks on ATMSIM.

3.5.3 Output switch information

The routine that stores the information required is SwOpEvent(). The file pointers of the files, where the monitored performance parameters are stored, are represented in the form of arrays of file pointers. The file pointer indexed by the identification number of the switch, is opened for appending the new information regarding the performance parameters. The flags in the switch output structure are checked to find the parameters to be stored in the output file. The instantaneous value fields in the corresponding data structures are set to zero in order to obtain the values till the next output time. The data structure containing information about the flags and the output file name is given below.

```
typedef struct swOpInfo { /* output info. for each switch */
  int   switchNo;/* switch # of which info. is given */
  int   lossFlag;/* specifies whether or not cell losses
                  * have to be given as output */
  int   bufFlag; /* specifies whether or not buffer
                  * utilization has to be stored in the file */
```

```
  time_t *freq;   /* frequency at which the output is given in other
                   * words time interval between two outputs */
  char  *swFile; /* file into which output has to be printed */
  char  *swName; /* User's name to the switch being dealt with */
} swOpInfo_t;
```

The *frequency* field is used in finding out the time at which the storing of the parameter information is to be done. This value is added to the current time to give the next output time. With this time as timestamp, an switch output information event is put in the event queue along with the identification number of the switch.

3.5.4 Output link information

LinkOpEvent() is the routine performing the task of storing the information required. The process of output is similar to that of a switch except that information about the link is used in place of that of the switch. The parameters output in this routine are link utilization and number of cells cells sent through the link for each kind of traffic. The data structure containing the information about the flags and the name of the output file is given below.

```
typedef struct linkOpInfo {  /* output info. for each link */
  int    linkId;    /* link Identifier of the link */
  char   *linkName; /* user's name for that link */
  char   *linkFile; /* name of the file for output */
  time_t *freq;/* time interval between two successive outputs */
  int    utilFlag;/* whether or not to output link utilization */
} linkOpInfo_t;
```

3.5.5 Output VC information

The routine for storing the performance parameters is VcOpEvent(). The parameters being monitored for each virtual connection are throughput, maximum queueing delay experienced by the cells, cells dropped and end to end delay for each virtual connection. The cells received at the destination of the current virtual connection are processed. To perform this the following actions are taken.

- The queuing delay information of each cell received is obtained from the cell structure to calculate the total queuing delay and to check for the maximum and the minimum queuing delays.
- The cells are either discarded or processed depending on the application being implemented on top of the simulator. In this implementation, the cells are discarded after finding the queuing delay information.
- The number of cells received at the destination are counted. The queuing delay and cell count information is used to update the information in the VC output structure given in Section 3.2. The *throughput* field of VC output structure contains information about the cumulative and instantaneous cells reaching the destination.

The output process is similar to that of switch except that the information about the VC is used in the place of that of the switch. The data containing the information about the flags and the name of the output file is given below.

```
typedef struct vcOpInfo {/* output info. for each virt. conn. */
  int    vcInd;
  /* source identifier for a source */
  char   *srcName;
  /* source name given by the user */
  char   *vcFile;
  /* the name of the file into which
  /* output has to be stored */
  time_t *freq;
  /* frequency at which the VC output is stored */
  int    e2eFlag;
  /* flag to show whether or not
  /* the end to end delay can be output */
  int    qFlag;
  /* flag to show whether or not
  /* the queuing delay can be output */
  int    thruputFlag;
  /* flag to show whether or not
  /* the thruput can be output */
} vcOpInfo_t;
```

3.6 Conclusions

In this implementation the special features of interest are

- The abstraction of the event routines and the information passed to them.
- Ease in the way of getting various routines executed depending on the source type, buffer allocation policy, etc.
- The implementation aspects of the resursive descent parser.

We will discuss about the accomplishments of this thesis work and the proposed enhancements in the next chapter.

Chapter 4

Conclusions

In this chapter, we summarize the accomplishments of this thesis, and propose future enhancements to ATMSIM.

4.1 Accomplishments

The need for a simulator for ATM networks has been felt, due to the fact that the simulators available can not be used as ATM network simulators. For finding the feasibility of various schemes on ATM networks, using a simulator is cost effective. Therefore, ATMSIM, a simulator has been designed and implemented with the desirable features of a simulation software with ATM networks as target system. The features like generation of cells at the source and switching of cells from one port to the other depending on the routing tables at the intermediate switches are the most common ones in the context of ATM networks. In order to incorporate these features along with the aspects related to performance monitoring, ATMSIM has been designed using discrete-event simulation technique. In ATMSIM, a user interface has been provided using which user specified input file is read and the network is set up. ATMSIM also provides some primitive routines which can be used for implementing different congestion control and resource reservation schemes on ATM networks. The performance parameters stored in the database can be used to decide whether the scheme being implemented is feasible in reality. ATMSIM provides a database of statistical results which can be converted into the form needed by the

graphics packages to give the graphical output.

4.2 Future Enhancements

The features that can be added to ATMSIM are as follows.

- Features such as connection set up, source modeling, schemes for buffer allocation and bandwidth allocation, required by various congestion control and resource reservation scheme have to be added to ATMSIM, so that it can be used as a tool for implementing different applications of ATM networks.
- A graphical user interface has to be provided to the end user to setup the network and to modify the network related parameters dynamically.
- ATMSIM deals only with features of ATM layer such as switching and cell generation. It can be extended to incorporate the features of higher layers so that applications related to other network protocols can also be implemented.

Bibliography

[1] A Alles. Tutorial: ATM in private networking. Hughes Lan System, 1993.

[2] J J Bae and T Suda. Survey of traffic control schemes and protocols in ATM networks. *Proceedings of the IEEE*, 79(2):170–189, February 1991.

[3] Uzzal Baruah. A framework for congestion control in ATM networks. Master's thesis, CSE Dept, I I T, Kanpur, India, March 1995.

[4] A W Berger, A E Eckberg, T C Hou, and D M Lucantoni. Performance characterizations of traffic monitoring and associated control mechanisms for broadband packet networks. In *IEEE INFOCOMM*, pages 350–354, 1990.

[5] J-Y Le Boudec. The Asynchronous Transfer Mode : A Tutorial. *Computer Networks and ISDN Systems*, 24:279–309, 1992.

[6] Klaudia Dussa-Zieger Cengiz Alaettinoglu, A.Udaya Shankar and Ibrahim Matta. Design and implementation of MaRS : A routing testbed. Technical Report 2687, University of Maryland,College Park, jun 1993.

[7] Prycker M de. *Asynchronous Transfer Mode Solution for Broadband ISDN*. Ellis Horwood, 1993.

[8] Eckberg A E. BISDN/ATM traffic and congestion control. *IEEE Network*, pages 28–37, September 1992.

[9] V. Frost and B. Melamed. Traffic modeling for telecommunications networks. *IEEE Communicaqtions Magazine*, pages 70–80, March 1994.

[10] G Gallassi, G Rigolio, and L Fratta. ATM : Bandwidth assignment and bandwidth enforcement policies. In *IEEE GLOBECOM*, 1989.

[11] Naim A. Kheir, editor. *Systems Modeling and Computer Simulation*, volume 46. Electrical Engineering and Electronics, second edition edition.

[12] Averill M. Law and Michael G. McComas. Simulation software for communications networks : The State of the Art. *IEEE Communications Magazine*, pages 44–50, March 1994.

[13] Q.Wang and V. Frost. Efficient estimation of cell blocking probability for ATM systems. *IEEE Trans. on Networking*, April 1993.

[14] M N Ransom and D Spears. Applications of public gigabit networks. *IEEE Network*, March 1992.

[15] E Rathgreb. Modeling and performance comparison of policing mechanisms for ATM networks. *IEEE Journal on Selected Areas in Communications*, SAC-9(3), April 1991.

[16] Victor S.Frost. Computer-aided modeling and simulation for communications networks. *IEEE Communications Magazine*, page 42, March 1994.

[17] S.Schoemaker. *Computer Networks and Simulation.* Second edition edition, 1968.

[18] J S Turner. Managing bandwidth in ATM networks with bustry traffic. *IEEE Network*, pages 50–58, September 1992.

[19] W Wang and T N Saadawi. Bandwidth allocation for ATM networks. In *IEEE ICC*, pages 439–432, 1990.

Appendix A

Input File

A.1 Input File Format

The input file comprises the following declarations (not necessary in the same order), each of them terminated by a semicolon. The declarations which comprises of blocks (where a block is a series of statements delimited by opening and closing brace) should not be terminated by a semicolon. Also white space characters are ignored and can be used to improve readablity. A line with hash '#' as the first non-white character is treated as a comment. The declarations are:

- Number of Switches
- Names of Switches
- Number of Links
- Names of the Links
- Number of Sources
- Routes
- Link Information
- Source Information
- Buffer Allocation Policy Information

- Bandwidth Allocation Policy Information
- Switch Output Information
- Link Output Information
- Source Output Information
- Accuracy Desired
- Simulation Duration

These declarations need not appear exactly in the order shown above but certain declarations must appear before others. The number of switches must be declared before the names of the switches, the buffer allocation policy information, the bandwidth allocation policy information and the switch output information. . Similary the number of links has to be declared before the number of links, link information, link output information. Also, the number of sources must be specified before any declaration related to source is done.

Each identifier (i.e., the name of the switch and route) must be a sequence of alphanumeric characters that begins from a alphabetic character. Thus *A*, *src1* are valid identifiers. values gives in '[ ]' are optional.

The declaration syntax is specified below:

numofswitches = {*number of switches*} ;
nameofswitches : {*comma separated identifier list*} ;
nooflinks = {*number of links*} ;
linksare : {*comma separated list of links*} ;
# Each link is two switch names separated by a dash.
noofsources = {*number of sources*} ;
routesare {
    # each route is a dash separated list of identifiers
    {*source name 1*} {*route 1*} ;
    {*source name 2*} {*route 2*} ;
    ...
    {*source name n*} {*route n*} ;
}
linkinfo {

```
        # a link name of default will set the given
        # values as default
        {linkname 1}  {
                bandwidth = {bandwidth}  {unit} ;
                buffer = {buffer size} ;
                propdelay = {propagation delay}  {unit} ;
        }
        {linkname 2}  {
                bandwidth = {bandwidth}  {unit} ;
                buffer = {buffer size} ;
                propdelay = {propagation delay}  {unit} ;
        }
        ...
        {linkname n}  {
                bandwidth = {bandwidth}  {units} ;
                buffer = {buffer size} ;
                propdelay = {propagation delay}  {units} ;
        }
}
sourceinfo {
        # if source name is default that will specify
        # default values for all sources
        {source 1}  {
                starttime = {start time}  {units} ;
                livetime = {live time}  {units} ;
                peakrate = {peak rate}  {unit} ;
                averagerate = {average rate}  {unit} ;
                burstsize = {burst size} ;
                RESERVEDBANDWIDTH = {reserved bandwidth}  {unit} ;
                burstduration = {burst duration}  {unit} ;
                bursttolerance = {burst tolerance}  {unit} ;
                celllossprobability = {the probability}  {unit} ;
                video;
                interburstgap = {inter burst gap}  {unit} ;
                delaytolerance = {delay tolerance}  {unit} ;
```

```
        }
        ...
        # other sources follow
}
bapolicyinfo {
        # if switch name is default that will specify
        # default values for all switches
        {switch name 1}  {
                {drop policy type} ;
                {buffer size} ;
                {buffer allocation policy type} ;
        }
        ...
        # other switches follow
}
bwpolicyinfo {
        # if switch name is default that will specify
        # default values for all switches
        {switch name 1}  {
                {band width allocation policy type} ;
        }
        ...
        # other switches follow
}
switchoutput {
        # switches name can be default
        {switch name 1}  {
                [lossofcells;]
                [bufferutilization;]
                frequency = {the frequency}  {unit} ;
                file = {the file} ;
        }
        ...
        # more switches follow
}
```

linkoutput {
    # links name can be *default*
    {*link name 1*} {
        [linkutilization;]
        frequency = {*the frequency*} {*unit*} ;
        file = {*the file*} ;
    }
    ...
    # more links follow
}
sourceoutput {
    # {*source name*} can be *default.*
    {*source name 1*} {
        [endtoenddelay;]
        [qdelay;]
        [throughput;]
        frequency = {*the frequence*} {*unit*} ;
        file = {*the file*} ;
    }
    ...
    # others sources follow
}
accuracy = {*accuray*} ;
simulationduration = {*simulation duration*} ;

A.2 Sample User Input File

```
noofswitches = 5;
nameofswitches : A, B, C, D, E;
nooflinks = 4;
linksare : A-B, B-C, B-D, B-E;
noofsources = 3;
```

```
routesare {
src1 A-B-C;
src2 A-B-D;
src3 A-B-E;
}

linkinfo {

      default {
         bandwidth = 55442 KB ;
         buffer    = 3000 cells;
         propdelay = 500 micro;
      }
      A-B {
         bandwidth = 155442 KB ;
         buffer    = 3000 cells;
         propdelay = 500 micro;
      }
}

sourceinfo {
       default {
          starttime   = 1 msec;
          livetime    = 1000 sec;
          peakrate = 24392.04545 kb;
          averagerate = 6048.37 kb;
          burstsize = 1000 cells;
          RESERVEDBANDWIDTH = 10 MB;
          burstduration = 1 msec;
          bursttolerance = 100 million;
          celllossprobability = 1 inmillion;
          video;
          interburstgap = 10 micro;
          delaytolerance = 1 msec;
       }
```

```
       src1 {
          starttime   = 1 msec;
          livetime    = 1000 sec;
          peakrate = 345.04545 kb;
          averagerate = 60.37 kb;
          burstsize = 100 cells;
          RESERVEDBANDWIDTH = 0.2 MB;
          burstduration = 1 msec;
          bursttolerance = 150 million;
          celllossprobability = 1 inmillion;
          data;
          interburstgap = 10 micro;
          delaytolerance = 1 msec;
       }
       src2 {
          starttime   = 1 msec;
          livetime    = 1000 sec;
          peakrate = 64.04545 kb;
          averagerate = 25.37 kb;
          burstsize = 100 cells;
          RESERVEDBANDWIDTH = 0.2 MB;
          burstduration = 1 msec;
          bursttolerance = 30 million;
          celllossprobability = 1 inmillion;
          singlevoice;
          interburstgap = 10 micro;
          delaytolerance = 1 msec;
      }
}

bapolicyinfo {
       default {
   LIFD;
           infinite;
           distributed;
```

```
        }
        A {
   CLP1;
            fixed;
            shared;
        }
}

bwpolicyinfo {
        default {
            reserved;
        }
}

switchoutput {

        default {
            LOSSOFCELLS;
            BUFFERUTILIZATION;
            frequency = 1 msec;
            file = swds10;
        }
        B {
            BUFFERUTILIZATION;
            frequency = 3 msec;
            file = swds12;
        }
}

linkoutput {
        default {
        linkutilization;
        frequency = 1 msec;
        file = linkds10;
        }
```

```
}

sourceoutput {
        default {
        endtoenddelay;
        qdelay;
        throughput;
        frequency = 1 msec;
        file = srcds10;
        }
        src2 {
        endtoenddelay;
        throughput;
        frequency = 4 msec;
        file = srcds11;
        }
}


accuracy  = 0.02727 ;
simulationduration  = 200;
```