INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Cell Loss Estimation in an ATM Multiplexer Loaded with Heterogeneous On-Off Sources

Charlie Kawwas

A Thesis

in

The Department

of

Electrical & Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montreal, Quebec, Canada

December 1998

ABSTRACT

Cell Loss Estimation in an ATM Multiplexer Loaded with
Heterogeneous On-Off Sources

Charlie Kawwas

In an ATM network, estimating cell loss probability is a crucial factor in congestion control and traffic management. This thesis proposes a method to estimate the cell loss probability in an ATM multiplexer with heterogeneous traffic sources. We consider an ATM Multiplexer with a finite buffer of size K (cells) and a single link with capacity C (cells/second). Unlike previous work, we approximate the heterogeneous arrival process using a Markov Modulated Deterministic Process (MMDP). We model the ATM multiplexer as MMDP/D/1/K queueing system, then we derive the cell loss probability formula for an ATM multiplexer, and we provide an algorithm to compute it. We consider several classes of traffic sources: voice, low-speed data, high-speed data and image-retrieval. Then, we compute the cell loss probability based on the analysis derived. By simulating the same network, we calculate the cell loss probability. Comparing the analysis and simulation figures proves the accuracy of our results. After examining these results, we observe that a dominant class is present in the heterogeneous ATM network, and it significantly affects the cell loss probability when extra connections are accepted from this class. Moreover, we demonstrate that our model performs consistently well for large buffer sizes where the burst level congestion is dominant. Therefore, this model is suitable for use in preventing burst level congestion from ATM heterogeneous networks.

ACKNOWLEDGMENTS

I would like to express my profound gratitude and respect to my supervisor Prof. M.R. Soleymani for his guidance, support and encouragement that made this research and work possible. He has always been available to advise and direct me throughout the course of this work. Also, he has contributed a great deal of his time, effort and ideas to the work presented in this thesis.

I am also very thankful to Nortel (Northern Telecom) for funding my studies and giving me the opportunity to further my education and broaden my knowledge.

Most of all, I would like to thank my parents, my wife, my sister and my brother for their tremendous love, help and support without which this work could never be accomplished.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 ATM

Asynchronous Transfer Mode (ATM) is based on asynchronous time division multiplexing and the use of fixed length cells. Each cell consists of a header and an information field. The cell header is used to identify cells belonging to the same virtual channel within the asynchronous time division multiplex. It is also used to carry out the appropriate cell-routing functions. The ATM cell information field is transparently routed throughout the ATM network. No error control checking or processing is performed on the information field inside the network [1] & [2].

ATM is capable of carrying out various services such as: voice, data and video. In order to accommodate these services, several types of ATM Adaptation Layers (AALs) have been defined to accommodate the various services. The AAL specific information is included in the information field of the ATM cell.

An ATM network consists of ATM switches that are connected to each other using an interface called Network-Node Interface (NNI), while terminals are connected to the network using User-Network Interface (UNI). There are two types of UNI interfaces: a public UNI that defines the interface between a public service ATM network and a private ATM switch, and a private UNI which defines an ATM interface between an end-user and a private ATM switch [3] & [4].

ATM is a connection-oriented protocol. Connections are established either as Permanent Virtual Circuits (PVC) when a user is provisioned to the network, or as Switched Virtual Calls (SVC) when a user enters into a session with a network as a connection-on-demand. The virtual channel numbers assigned in the cell header are used for the complete duration of the connection. These values are translated when switched from one section to another. An ATM connection is called a virtual connection. There are two types of virtual connections: Virtual Path Connections (VPC) and Virtual Channel Connections (VCC). A VPC is considered an aggregate of VCCs. Cell switching is done on the VPC, then on the VCC as shown in Figure 1-1 [1], [5] & [6].

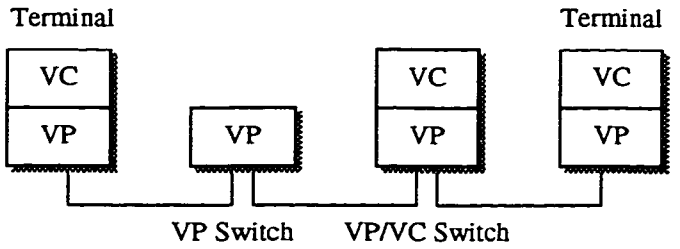

Figure 1-1: VC/VP ATM Connections.

1.2 Broadband ISDN

The International Telecommunications Union (ITU) – Telecommunication Standardization Sector (ITU-T), formerly known as the CCITT, describes the Broadband Integrated Services Digital Network (B-ISDN) as a network built on the concepts of the ISDN model, and as a network which is implemented using ATM and Synchronous Optical Network (SONET) technologies. The ITU-T presents these two technologies as complementary technologies as illustrated in Figure 1-2 where the ATM switches act as the UNI with the user's Customer Premise Equipment (CPE). As a result, the user can deploy a wide variety of services at an ATM node that relays the traffic to SONET ports or local UNI ports. Therefore, SONET and ATM form an alliance for B-ISDN where SONET provides the transport, operations and maintenance services, while ATM provides direct services to the users [5].

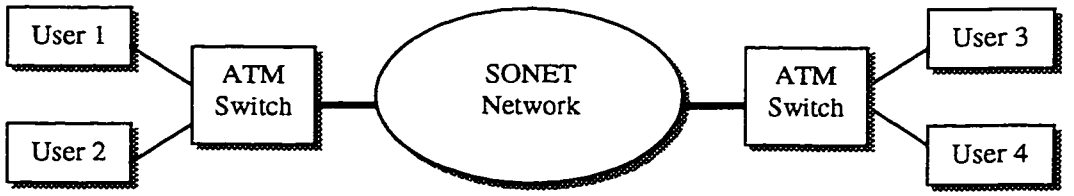

Figure 1-2: ATM & SONET.

1.3 ATM Traffic Classes

Traffic is classified into four different classes in ATM. Figure 1-3 illustrates the relationship of each class in terms of timing, bit-rate and connection mode requirements. The four classes of ATM traffic are:

- *Class A:* Constant Bit Rate (CBR) traffic such as telephony, circuit emulation of narrow-band ISDN services and video conferencing [7].
- *Class B:* Variable Bit Rate (VBR) is divided into two categories depending on sensitivity to cell delay variations: *real-time VBR (rt-VBR)* such as interactive compressed video, and *non real-time VBR (nrt-VBR)* such as multimedia e-mail [8] & [9].
- *Class C:* Available Bit Rate (ABR) traffic that constitutes most of the remaining bandwidth and is used for normal data traffic such as file transfer. Applications using this class require fair access to this bandwidth with a minimum of cell loss [8].
- *Class D:* Unspecified Bit Rate (UBR) traffic is intended for applications and services that use what is left of the bandwidth and are insensitive to cell delay or loss. Examples of UBR can be specific application of news groups [10] & [11].

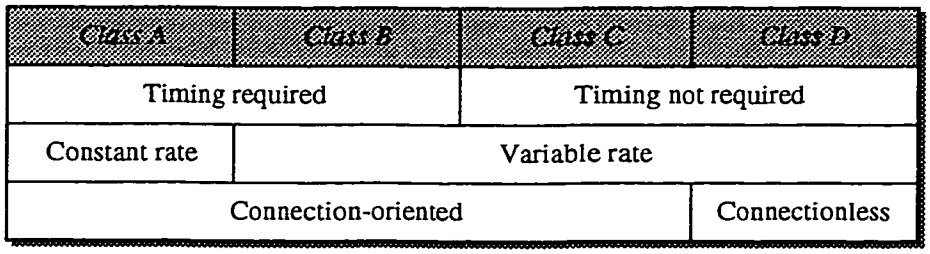

Figure 1-3: Traffic Classes.

1.4 ATM Source Characterization

An ATM source can be characterized at three different levels: connection, burst and cell levels as shown in Figure 1-4 [12] & [13]. The connection level activity

describes the traffic of a source on a virtual connection from the connection setup until the disconnection. The length of a connection might be from tens of seconds until minutes or hours.

Each connection is composed of an alternating silence and burst periods. During the silence period the source is off, and no cells are generated. While during the burst periods, the source is on, and cell arrivals can be approximated in terms of the mean rate. Typical length of a burst period can be from a few seconds to tens of seconds.

The lowest level that an ATM source can be characterized at is the cell level, which describes the behavior of the source traffic at the cell level. The cell level can be calculated from the source transmission rate and the cell length (53 bytes). Typically the cell level is in the microseconds.

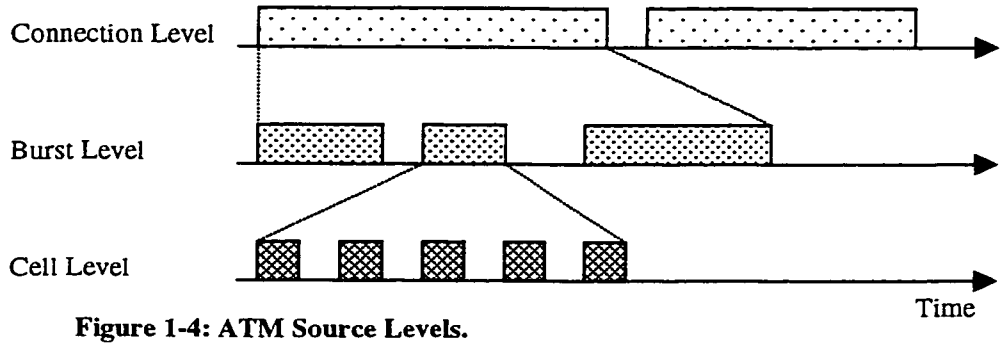

Figure 1-4: ATM Source Levels.

1.5 Congestion Control in ATM

The use of statistical multiplexing in ATM allows many virtual connections to share network resources, since each connection needs these resources for a fraction of its time. Congestion might occur while ATM cells are competing for the network resources. During congestion periods, cells are stored in a buffer with a finite size. In an ATM network, cells carrying traffic are susceptible to loss as a result of congestion. Therefore, it is important to have proper congestion control to ensure efficient and fair operation of the network in spite of varying demand. When the network is under a high load, the use of statistical multiplexing in ATM allows an efficient use of the bandwidth [14] & [15].

One of the techniques in congestion control is based on the estimation of the cell loss probability in an ATM multiplexer. This method allows us to obtain a good estimate of the Quality of Service (QoS) [16], [17] & [18] and the ability to accept or refuse a connection request to avoid long term congestion. Figure 1-5 illustrates the relationship between the cell loss probability, buffer size and source levels. For small buffer sizes, cell loss probability is caused by cell level fluctuations. However, for larger buffer sizes, cell loss is more attributed to burst level traffic. Connection level traffic is also considered as burst level with larger periods, thus connection level traffic affects cell loss probability at large buffer sizes [13] & [19].

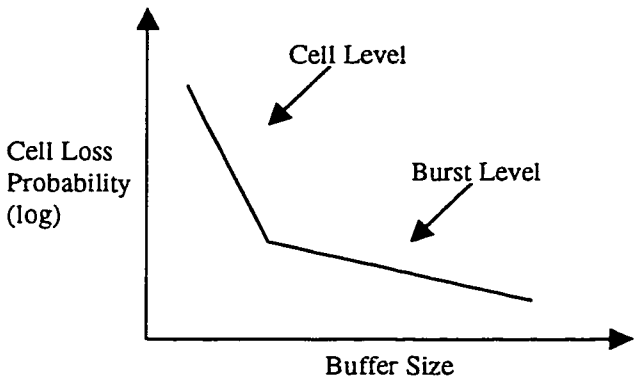

Figure 1-5: Cell Loss Probability and Source Levels.

1.6 ATM Traffic Modeling

To address the estimation of cell loss probability in an ATM multiplexer, ATM traffic sources must be modeled to characterize their behavior. Several techniques are used for traffic modeling, such as: Bernoulli process, On-Off source model, Markov Modulated Poisson Process (MMPP), Fluid-Flow (FF) approximation and Pareto Modulated Poisson Process (PMPP). In this thesis, the On-Off source model is used. In the On-Off model, a source can be in two states: an *on* state where cells are generated at a constant rate, or an *off* state where no cells are generated. We approximate the actual arrival process using a Markov Modulated Deterministic Process (MMDP). Also, we model the ATM multiplexer by a superposition of several classes of independent On-Off

sources, where sources within a class are identical. The multiplexer is modeled as a single server with a service rate C cells per second and a buffer size of K cells. Figure 2 illustrates our ATM multiplexer model as an MMDP/D/1/K queueing system [20].

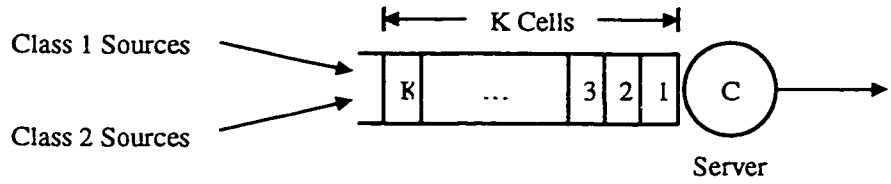

Figure 1-6: ATM Multiplexer Model.

1.7 Cell Loss Estimation & Previous Research

One of the most important papers and research conducted in the field of queueing is the paper by Anick, Mitra and Sondhi [21]. They considered a physical model in which a buffer receives messages from a finite number of independent and identical sources that alternate between on and off states. In the on state, sources generate messages at a uniform rate. The buffered messages are processed through an output channel with a given maximum rate of transmission. The paper presents the equilibrium buffer distribution by a set of differential equations which are analyzed. The main result provides the system's eigenvalues. Also, a simple expression is provided for the asymptotic behavior of buffer content. Finally, the paper presents numerical results for a broad range of system parameters specifically the buffer overflow probability. The concepts and mathematical analysis presented by Anick, Mitra and Sondhi are applicable with certain changes to the ATM multiplexer model. For example, the output rate should be constant since the messages in ATM have a fixed length of 53 bytes, thus a deterministic service time. Moreover, the type of sources can be increased to include a heterogeneous traffic mix as opposed to a homogeneous one.

Many researchers have studied various models of the ATM multiplexer and its cell loss probability. Table 1-1 presents a summary and comparison of several of these researches. The first one considers the multiplexer as a discrete-time system and models

it as a two-dimensional discrete-time Markov chain [22] & [23]. The cell loss probability is expressed in terms of the Markov chain's limiting probabilities. Even though this approach results in an accurate estimate of the cell loss probability, it is computationally heavy.

The second approach estimates the arrival process by a two-state Markov Modulated Poisson Process (MMPP) and the system by an MMPP/D/1/K queueing system [24] & [25]. The cell loss probability is also expressed in terms of the limiting probabilities. This approach is computationally more efficient; nevertheless, its accuracy may not be reliable, as it depends on how the parameters of the two-state MMPP are estimated.

Table 1-1: Models of ATM Multiplexer & Cell Loss Estimation.

Model	*Summary*
2D discrete-time Markov chain	• Accurate estimate of cell loss probability • Computationally heavy
2-state MMPP	• MMPP/D/1/K queueing system • Computationally more efficient • Reliability depends on how MMPP parameters are estimated
SFF	• Accurate estimate of cell loss probability • Can breakdown when solving large size problem
Homogeneous MMDP	• Similar to SFF but discrete • Only one class of traffic considered • Accurate estimate of cell loss probability • Numerically stable and efficient
Heterogeneous MMDP	• More than one class of traffic considered • Realistic queueing system • Accurate estimate of cell loss probability • MMDP/D/1/K queueing system • Numerically stable with two classes in the network • Increasing the number of classes results in exponentially increasing the problem size

The third approach approximates the arrival process by a Stochastic Fluid-Flow (SFF) [26], [27], [28], [29] & [30]. The cell loss probability is calculated by several

approaches to the limiting probability of the buffer. This approach is accurate, but the major drawback is its potential to breakdown when solving a large size problem.

The fourth approach, which is similar to the SFF approach, is based on the approximation of the arrival process by a Markov Modulated Deterministic Process (MMDP) [31]. The major difference is that MMDP treats cells as discrete units, while SFF treats them as continuous fluid flow. This difference leads to a completely different solution that is as accurate as the SFF approach, but it is numerically stable and efficient.

Our approach is based on the MMDP proposal; however, the major difference is that the past work considered one class of traffic, i.e. a homogeneous case, while we present a more realistic scenario with a combination of two classes of traffic, i.e. a heterogeneous case. We limited the number of classes to two, as increasing the number of classes results in exponentially increasing the problem size; therefore, it becomes computationally heavy and unstable [20].

1.8 Scope of Thesis

In this thesis, chapter 1 provides a brief overview of the ATM and its synergy with SONET in the B-ISDN model. Then, it presents the various ATM traffic classes and their relationship with respect to timing, bit rate and connection mode. ATM source levels are discussed, and their effect on traffic congestion and cell loss probability is illustrated. The need for traffic modeling is presented, the model used in this thesis is discussed. Finally, the different models adopted by different researchers are compared, and the queueing model used in this research is highlighted.

Chapter 2 presents the ATM standard. The ATM cell structure and the value of each field of the ATM cell are given. Then, the B-ISDN/ATM reference model and the functionality of each layer are explained. Traffic attributes and different techniques in traffic control and congestion feedback are discussed and compared. Afterwards, ATM signaling and the end-to-end ATM network are presented.

In chapter 3, the main statistical features of traffic sources are given. Then, we provide various traffic source models and their application to different types of ATM traffic. We also present the selected source model as well as the traffic parameters used

to approximate the arrival process. Subsequently, we define the queueing system and derive the mathematical analysis. Based on the analysis, we provide a derivation of a new method to calculate the cell loss probability in an ATM multiplexer with heterogeneous traffic sources.

Chapter 4 presents a method to estimate the cell loss probability derived earlier. Moreover, an algorithm to compute the cell loss probability for a heterogeneous ATM network is discussed. Finally, we illustrate the design methodology used to implement the algorithm, thus providing a new technique to estimate the cell loss.

The numerical results and simulation are presented in chapter 5. Several types of traffic in ATM networks are considered, and the traffic parameters for each traffic type are provided. Then, we describe the design methodology and implementation used to simulate the ATM multiplexer with heterogeneous incoming traffic. Afterwards, we provide the performance analysis results for the same networks considered in the simulation. Finally, we compare the simulation and analysis results.

In chapter 6, the concluding remarks are discussed. We present a summary of the thesis and the research presented. Moreover, future research directions are discussed and provided.

2. ASYNCHRONOUS TRANSFER MODE

2.1 Introduction

Asynchronous Transfer Mode (ATM) provides a high-speed with low-delay switching and multiplexing network to properly transport all types of applications such as: voice, video and data. ATM segments and multiplexes traffic into fixed length packets called cells. Each cell has a header containing a virtual circuit identifier that is used to relay traffic through high-speed switches in the network. ATM does not provide any error detection or retransmissions on the cell payload, since it assumes a reliable end-to-end network. As a result, ATM can be used to implement networks carrying traffic at gigabit rates [5], [32] & [33].

In this chapter, we present the ATM standard. We start with the ATM cell structure, and we discuss the use and value of each field of the ATM cell. The B-ISDN/ATM protocol reference model layers are provided as well as the functionality of each layer. Then, we discuss ATM traffic parameters, as well as their use in guaranteeing Quality of Service (QoS) and their relationship to each of the ATM traffic classes. Subsequently, we present different techniques in ATM traffic control and congestion. Finally, we describe ATM signaling and the end-to-end ATM network.

2.2 The ATM Cell

Some of the design objectives used to define ATM are the integration of multimedia traffic, minimization of the switching complexity and reduction of buffering at intermediate nodes. These objectives were met by defining a short and fixed packet length called the ATM cell as the fundamental unit of ATM transmission. An ATM cell has 53 bytes: 5 bytes for header and 48 bytes for the payload or the information field, as shown in Figure 2-1 [3], [5], [7] & [34].

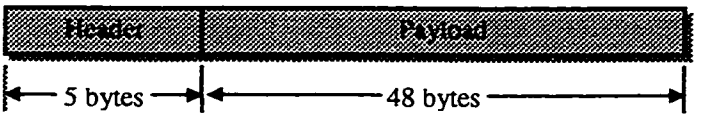

Figure 2-1: ATM Cell Structure.

The ATM cell header has six fields: virtual path identifier (VPI), virtual channel identifier (VCI), payload type (PT), cell loss priority (CLP), header error check (HEC) and generic flow control (GFC). The first five fields are used for the Network-to-Network Interface (NNI) as shown in Figure 2-2, while all six are used for the User-to-Network Interface (UNI) as in Figure 2-3 [3] & [5].

2.2.1 Virtual Path Identifier (VPI)

The VPI is a routing field for the ATM network, which is used for routing cells in a virtual path. Each virtual path is composed of 65K virtual channels. The VPI field in the UNI ATM cell contains eight bits for routing; therefore, allowing 256 virtual paths. The VPI in the NNI ATM cell consists of the first 12 bits of the cell header, which results in providing enhanced routing capabilities of 4096 virtual paths.

2.2.2 Virtual Channel Identifier (VCI)

The VCI is another routing field in the ATM cells, which is used for routing ATM cells in a virtual channel. Thus, the routing information of an ATM cell is included in the two routing fields of the header: VCI and VPI. A VCI consists of 16 bits that allow for 65K virtual channels.

2.2.3 Payload Type (PT)

The payload type identifier is a three-bit field that indicates the type of traffic in the information field. The cell can contain user, management or control traffic. This field can also be used for congestion notification operations.

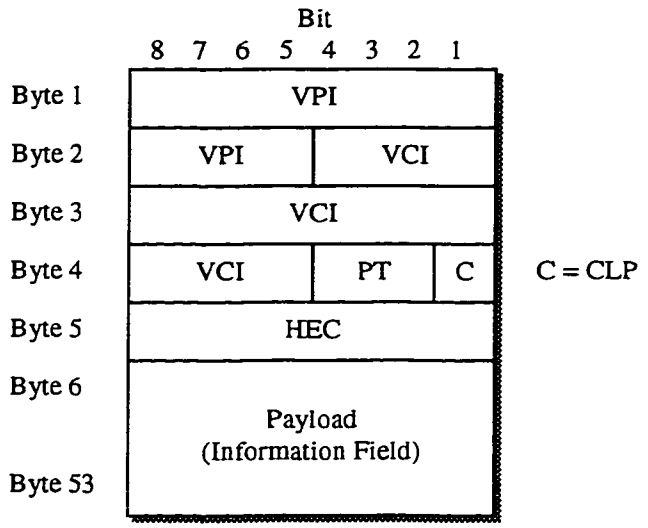

Figure 2-2: ATM Cell Structure for NNI.

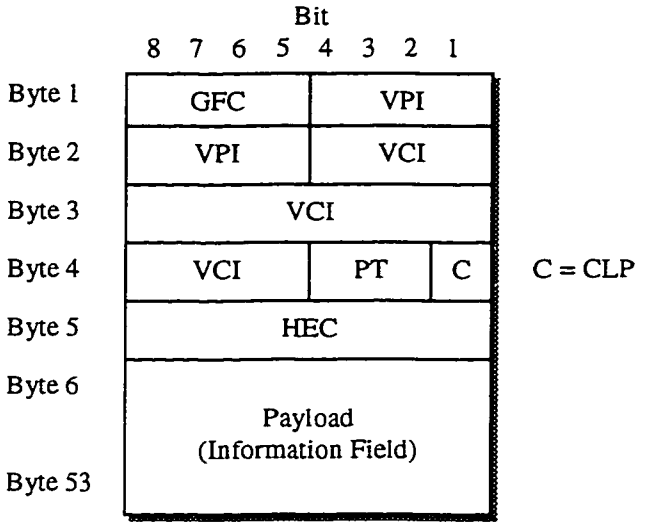

Figure 2-3: ATM Cell Structure for UNI.

2.2.4 Cell Loss Priority (CLP)

The CLP field is composed of one bit that is used to indicate the cell loss priority. If the CLP bit is set to 1, the cell is subject to being discarded by the network. Otherwise, the CLP is set to 0 that indicates a higher priority cell in the network.

2.2.5 Header Error Control (HEC)

The HEC is an error check field, which can correct a one-bit error. This field is computed based on the five-byte header in an ATM cell only, and not the 48-byte user information field.

2.2.6 Generic Flow Control (GFC)

The GFC appears only in the UNI ATM cells as shown in Figure 2-3. The GFC is a four-bit field that provides a framework for flow control and fairness to the UNI traffic. The use of this field has not been specified yet and is set to zeros in the UNI ATM cells.

2.3 B-ISDN Layered Model & ATM

The B-ISDN Protocol Reference Model (PRM) is based on the OSI model that uses the concept of separate planes to differentiate between user, control and management functions. Figure 2-4 illustrates the B-ISDN PRM for ATM. This model consists of three planes: User Plane (U-Plane), Control Plane (C-Plane) and Management Plane (M-Plane). The U-Plane provides the transport of the user information. The C-Plane is responsible for setting up and managing network connections. The M-Plane is composed of two layers: Plane Management and Layer Management. Plane Management coordinates and manages the different layers. Layer Management performs Operation, Administration and Maintenance (OAM) functions and services [1], [5] & [35].

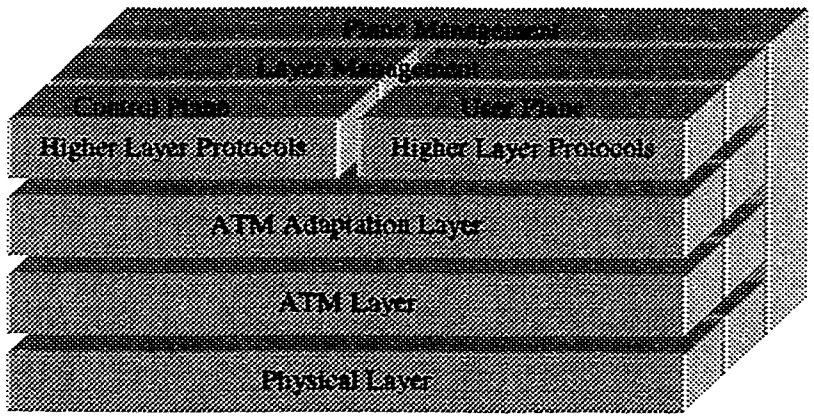

Figure 2-4: B-ISDN PRM and ATM.

For each plane, a layered approach is used with independence between the layers. There are three main layers: the Physical Layer, the ATM Layer, and the ATM Adaptation Layer (AAL). These layers are divided into sub-layers as shown in Figure 2-5 [1], [3] & [5]. Each sub-layer has a number of functions that are addressed in the next sections.

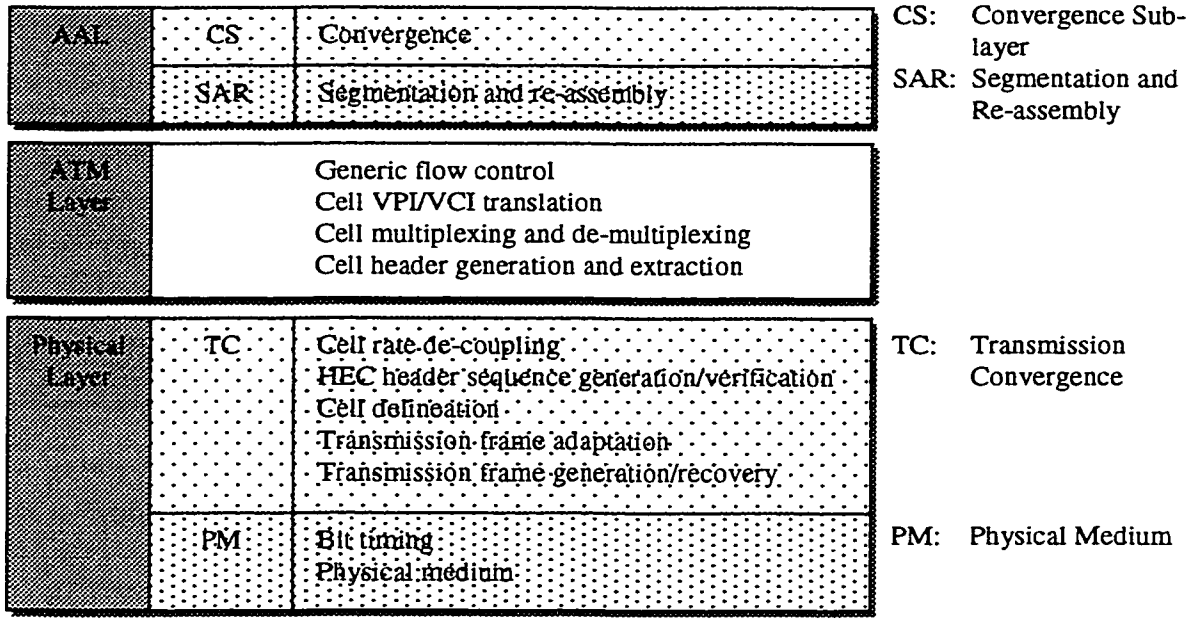

Figure 2-5: PRM Sub-layers and Functions.

On top of these three layers, we have the Higher Layer Protocols, as shown in Figure 2-4. This layer provides connectivity between the ATM network and higher protocols such as: Internet Protocol (IP), LAN Emulation (LANE), Multi-Protocol Over ATM (MPOA) and WAN service inter-working.

2.4 Physical Layer (PL)

The physical layer handles transmission operations such as: packaging of cells for transmission, sending data across a transmission medium, recovering data, and determining cell boundaries within a bit-stream. One of the key advantages of ATM is that it is independent of the transmission medium and rate. ATM is implemented over many physical layers such as: fiber, copper and wireless. Originally, ATM was

envisioned to use Synchronous Optical Network (SONET) or Synchronous Digital Hierarchy (SDH) at speeds of 25Mbps, 155Mbps (OC-3), 622 Mbps (OC-12) and 2.5 Gbps (OC-48). However, the ATM Forum and other ATM organizations defined other speed and media options for the UNI such as: DS-3 at 45 Mbps, Unshielded Twisted Pair category 3 (UTP-3) at 51 Mbps, and UTP-5 using TP-DDI at 100 Mbps. The physical layer functions can be divided in two sub-layers: Physical Medium (PM) sub-layer and Transmission Convergence (TC) sub-layer as shown in Figure 2-6 [1], [3] & [5].

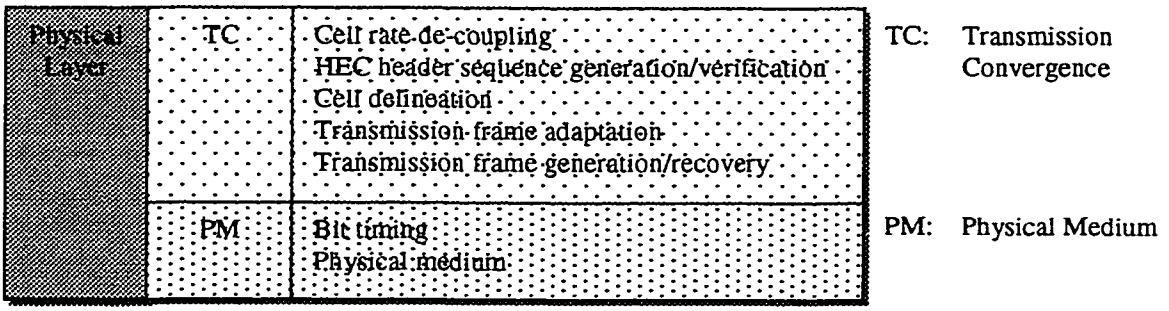

Figure 2-6: Physical Layer Sub-layers.

2.4.1 Physical Medium (PM)

The PM sub-layer provides the correct transmission and reception of bits on the physical medium. It also performs line coding as well as media conversion, if necessary. It includes bit-timing functions such as the generation and reception of wave forms suitable for the medium and also insertion and extraction of bit timing information. Since this sub-layer is at the lowest level, it is dependent on the physical medium.

2.4.2 Transmission Convergence (TC)

The TC sub-layer has five main functions as illustrated in Figure 2-6. The first function is generation and recovery of the transmission frame. The second function is transmission frame adaptation. The cells are fit within the transmission system according to a standardized mapping.

The third function, cell delineation function allows the receiver to recover the cell boundaries. Scrambling and de-scrambling are to be done in the information field of a cell

before the transmission and reception, respectively, to protect the cell delineation mechanism.

The fourth function, the HEC sequence generation is done in the transmit direction and its value is recalculated and compared with the received value and thus used in correcting the header errors. If the header errors can not be corrected, the cell will be discarded.

The last function, cell rate de-coupling inserts the idle cells in the transmitting direction in order to adapt the rate of the ATM cells to the payload capacity of the transmission system. It suppresses all idle cells in the receiving direction. Only assigned and unassigned cells are passed to the ATM layer.

2.5 The ATM layer

The ATM layer, which is above the physical layer, has four main functions: cell header generation/extraction, cell multiplexing & de-multiplexing, VPI/VCI translation and generic flow control, as shown in Figure 2-7 [1], [3] & [5].

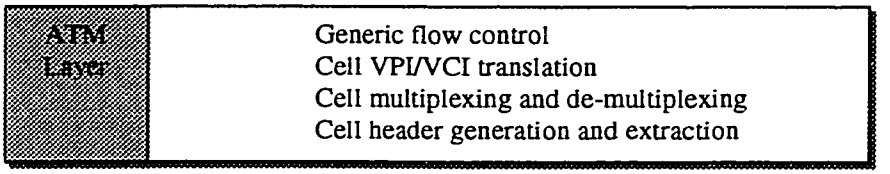

Figure 2-7: ATM Layer Sub-layers.

2.5.1 Generic Flow Control (GFC)

The GFC bits in the cell header support this function that is used to implement a flow control mechanism on the ATM traffic for the User-to-Network Interface (UNI).

2.5.2 VPI / VCI Translation

The translation of the cell identifier is performed when switching a cell from one physical link to another in an ATM switch or cross-connect. At a VC switch, the values

of VPI and VCI of incoming cells are translated into new values. At the VP switch, the VPI value is translated.

2.5.3 Cell Multiplexing & De-multiplexing

This function performs the multiplexing and de-multiplexing of cells from different connections that are identified by different VPI/VCI values onto a single cell stream.

2.5.4 Cell Header Generation and Extraction

This function is applied at the end points of the ATM layer. In the transmit direction, it adds the appropriate ATM cell header without the HEC value to the received cell payload from the AAL. In the receive direction, cell payload field is passed to the AAL after removing the cell header.

2.6 ATM Adaptation Layer (AAL)

The AAL enhances the service provided by the ATM Layer to a level required by the Higher Layer Protocols. It carries out functions for the user, control and management planes. Furthermore, it supports the mapping between the ATM layer and the higher layers [1], [3], [5] & [36].

The AAL function is divided into two sub-layers: *segmentation and re-assembly (SAR) sub-layer* and *convergence sub-layer (CS)*, as shown in Figure 2-8. The SAR sub-layer performs segmentation of a higher layer information into a size suitable for the payload of the ATM cells. At the receive side, the SAR reassembles the contents of the cells of a virtual connection into data units to be delivered to a higher layer.

The CS sub-layer performs functions such as message identification and time/clock recovery. This layer is further divided into Common Part Convergence Sub-layer (CPCS) and a Service Specific Convergence Sub-layer (SSCS) to support data transport over ATM. AAL service data units are transported from one AAL service access point (SAP) to one or more others through the ATM network. The AAL users can select a

given AAL-SAP associated with the quality of service required to transport the AAL service data unit (AAL-SDU).

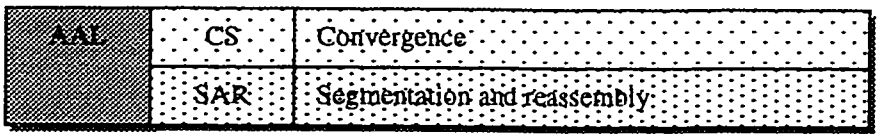

Figure 2-8: AAL Sub-layers.

2.6.1 AAL and Traffic Classes

Based on ATM traffic classes, the ITU has recommended four types of AAL protocols namely: AAL 1, AAL 2, AAL 3/4, and AAL 5. Each of these AAL protocols is described in the following sections.

2.6.2 AAL 1

AAL1 is intended for Class A CBR service that requires information to be transferred between the source and destination at a constant bit rate. The services provided by AAL 1 are:

- Transferring timing information between source and destination.
- Transferring service data units at constant bit rate and delivering them at the same bit rate.
- Transferring data structure information.
- Indication of lost or error information.

Typical applications of AAL 1 are voice and high quality constant bit rate audio and video.

2.6.3 AAL 2

AAL 2 is employed for Class B VBR services where timing relationship is required between the source and destination. This layer offers the following functions:

- Inserting and extracting time information for clock recovery.
- Handling of lost or misdelivered cells.

- Forward error correction for audio and video services.

Typical applications of AAL 2 are VBR audio or video.

2.6.4 AAL 3/4

AAL 3/4 is used to transfer data that is sensitive to loss, but not to delay. This AAL can provide both connection oriented and connectionless VBR services with no timing requirement. AAL 3/4 offers class C and class D services. It allows multiplexing at cell level between AAL connections using the same ATM-VCC and provides error detection on individual cells. AAL 3/4 can be used for applications such as large file transfers and data applications in general.

2.6.5 AAL 5

AAL 5 was recommended to offer a service with less overhead and better error detection below the CPCS layer. The service provided by AAL 5 is the same service provided by the CPCS of AAL 3/4 without multiplexing. Should multiplexing be required at AAL 5, it will be done in the SSCS layer. AAL 5 offers no error checking and supports only a single data stream per channel but devotes the entire cell data field to user data. The AAL 5 is used with Class C services that have variable bit rate sources without timing relationship between source and destination.

2.7 Traffic Control in ATM

In today's high-speed networks, a tremendous amount of information can be lost due to congestion on a link. As a result, it is very crucial to have proper congestion control to ensure efficient and fair operation of networks in spite of varying demand. Furthermore, during congestion periods, the information is buffered and delayed. If this information arrives at the destination after a certain delay threshold, then it is too late to be retransmitted or used. Many applications and services are delay sensitive, such as real-time services. This is the reason why traffic control is an important issue for the standard bodies such as the International Telecommunication Union (ITU), ATM Forum and Internet Engineering Task Force (IETF).

In current data networks, such as Ethernet and X.25, the user cannot explicitly indicate the quality of service and traffic profile required. Instead, the network is expected to provide a fair share of the available bandwidth to each user. This is not the case in an ATM network, since users adjust their transmission rates according to the feedback from the networks. In the following sections, we will discuss ATM traffic attributes and services, traffic congestion techniques, and signaling in ATM networks [3] & [37].

2.8 ATM Traffic Attributes

One of the technical challenges of ATM is the guarantee of Quality of Service (QoS), as it is very strenuous to dynamically allocate network resources to ensure the required QoS. The ATM Forum Technical Committee (TC) approaches this problem by requiring the terminal to precisely specify the traffic characteristics before establishing an ATM connection. There are several parameters that can be specified concerning traffic characteristics and Quality of Service (QoS). Table 2-1 provides a list of those parameters as well as their description [38]. These parameters can be classified into two categories as shown in Figure 2-9.

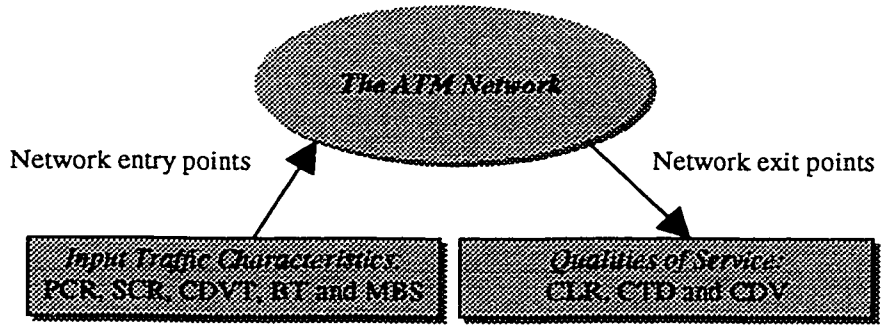

Figure 2-9: Traffic Parameters.

Table 2-1: ATM Traffic Attributes.

Parameter	*Description*
Peak Cell Rate (PCR)	Maximum cell rate of transmission.
Sustained Cell Rate (SCR)	Average cell transmission rate over a long period.
Cell Loss Ratio (CLR)	Ratio of lost cells to those transmitted due to error and congestion.
Cell Transfer Delay (CTD)	The cell delay between network entry and exit points.
Cell Delay Variation (CDV)	The variance of CTD.
CDV Tolerance (CDVT)	The slight variation in the inter-cell transmission time allowed.
Maximum Burst Size (MBS)	Maximum number of back-to-back cells allowed for transmission at PCR without violating SCR.
Burst Tolerance (BT)	The inter-cell burst time variation allowed at PCR without violating SCRT. $BT = (MBS-1)*(1/SCR - 1/PCR)$
Minimum Cell Rate (MCR)	Minimum cell transmission rate desired by the user.

2.8.1 Input Traffic Characteristic Parameters

Input traffic characteristics are those parameters that are enforced by the network at network entry points:

- Peak Cell Rate (PCR)
- Sustained Cell Rate (SCR)
- Cell Delay Variation Tolerance (CDVT)
- Burst Tolerance (BT)
- Maximum Burst Size (MBS)

2.8.2 QoS Parameters

Quality of Service (QoS) are those parameters that are provided by the network and measured at network exit points:

- Cell Loss Ratio (CLR)
- Cell Transfer Delay (CTD)

- Cell Delay Variation(CDV)

2.8.3 Traffic Classes and Traffic Attributes

Traffic attributes can also be specified for each of the four classes of traffic as described below and summarized in Table 2-2, note that "✓" means that a parameters is specified, "✘" means that it is unspecified, and "-" means that it is not applicable.

- Class A (CBR) related attributes are PCR, CDVT, CDV, CTD and CLR.
- Class B (VBR) has two categories: rt-VBR and nrt-VBR. The mean CTD is the important attribute for rt-VBR, while Maximum CTD and peak-to-peak CDV are the important attributes for nrt-VBR. Attributes related to both categories are PCR, CDVT, SCR, MBS, and CLR.
- Class C (ABR) related attributes are Feedback, CLR, MCR, PCR and CDVT.
- Class D (UBR) related attributes are PCR and CDVT.

Table 2-2: ATM Traffic Classes & Their Attributes.

Attribute	*CBR*	*rt-VBR*	*nrt-VBR*	*UBR*	*ABR*
PCR, CDVT	✓	✓	✓	✓	✓
SCR, MBS, CDVT	-	✓	✓	-	-
MCR	-	-	-	-	✓
Peak-to-Peak CDV	✓	✘	✓	✘	✘
Mean CTD	✘	✘	✓	✘	✘
Maximum CTD	✓	✓	✘	✘	✘
CLR	✓	✓	✓	✘	✓
Feedback	✘	✘	✘	✘	✓

2.9 Congestion Feedback & Traffic Shaping

2.9.1 Traffic Congestion

Congestion occurs when the total input rate is larger than the available link capacity. Depending on the duration of the congestion, a specific congestion technique is selected. For instance, for short congestion periods buffering, link-by-link feedback or even end-to-end feedback is used. On the other hand, for long congestion periods, capacity and network design or dynamic routing is considered [38].

Congestion control methods can be either preventive or reactive. On one hand, reactive methods help in controlling the congestion in the network. These methods are suited for low speed networks. On the other hand, preventive methods help avoid congestion by utilizing bandwidth enforcement methods. These methods are suited for high speed networks. In the next sections, we will present three congestion control methods: Connection Admission Control (CAC), traffic shaping using the "leaky bucket" algorithm and feedback mechanisms [34].

2.9.2 Connection Admission Control (CAC)

One of the methods to control medium duration congestion is the CAC. It is defined as the set of actions taken by the network during the call set-up phase to establish whether a VC/VP connection can be made. A connection request for a given call is only accepted if there are sufficient network resources to establish the end-to-end connections, while maintaining its required QoS attributes without affecting the QoS attributes of other connections [3].

CAC is based on routing cells according to load level of links and rejects any new connections if all paths are highly loaded. The "busy" tone in the telephone network is an example of CAC. There are two types of traffic attributes to be considered with CAC: the first is source traffic parameters such as PCR, SCR, BT and MBS. The second type is QoS required by the source such as CTD, CLR, CDV and CDVT.

2.9.3 The Leaky Bucket

The Leaky Bucket algorithm is a traffic-shaping technique that changes the traffic characteristics of a stream of cells on a VPC or a VCC by properly spacing the cells of individual ATM connections to reduce CDV and decrease PCR attributes. Any traffic shaping technique should preserve the cell sequence of an ATM connection.

This algorithm is also known as Generalized Cell Rate Algorithm (GCRA) which is used to determine if the variation in the inter-cell times is acceptable. All arriving cells are placed in a bucket, then they are drained at a specific rate. If too many cells arrive at the same time, the bucket may overflow. If the overflowing cells are admitted to the network, their CLP bit is set, so that in case of overload these cells will be the first to be dropped. This algorithm has two parameters: the inverse of the peak cell rate, and the second is the allowed variation on the inter-cell time, GCRA (1/PCR, CDVT) [38].

2.9.4 Feedback Mechanisms

There are two feedback mechanisms specified in the UNI: Generalized Flow Control (GFC) and Explicit Forward Congestion Indication (EFCI). GFC uses the first four bits of the cell header for the user-network interface (UNI). At the network-network interface (NNI), these bits are used for the VP field instead. This approach has been dropped from the current specification of the UNI. Instead, an explicit approach is used.

EFCI uses the payload type field in the cell header to send congestion information in a binary format. When the first bit is set to zero, the second bit is treated as EFCI bit. For all cells leaving their sources the EFCI bit is set to zero. However, switches in the connection path may set the EFCI bit in case of congestion. When the destination receives a cell with the EFCI bit set, it will demand from the source to adjust its transmission rate accordingly [38].

2.10 Flow-Control Techniques: Credit-Based vs. Rate-Based

For the past three years, the Traffic Management Working Group of the ATM Forum has been working on a way to implement the flow-control feedback for ABR services. Throughout the initial phases, several schools of thought were presented

regarding flow-control implementation. However, they were funneled down to two approaches: one is a "credit-based" scheme, and the other is a "rate-based" scheme. In the following sections, we will present both approaches with their advantages and disadvantages [37].

2.10.1 Credit-Based Approach

Credit-based flow-control makes use of a feedback loop on a hop-by-hop basis as shown in Figure 2-10. Each link in the network runs a flow-control loop for each virtual circuit independently. As traffic moves through the network, it moves through a series of hop-by-hop feedback loops. The receiving end of each link maintains a separate queue, the receiver checks the length of each queue and determines the number of cells the source is allowed to send which is called "credits". The credit is sent to the transmitting end indicating the number of cells the transmitter is allowed for a specific VC. The main source transmits only as many cells as allowed by the credit received [39].

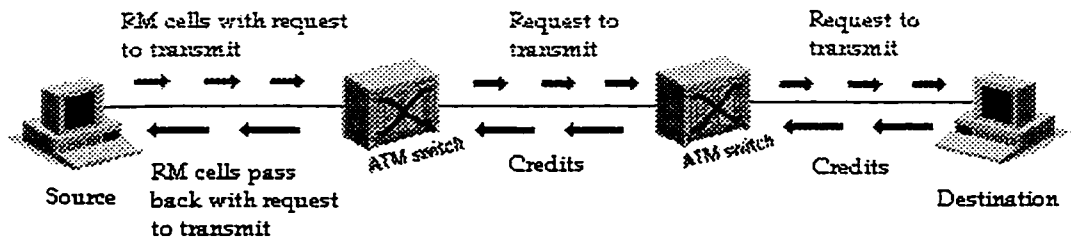

Figure 2-10: Credit-Based Feedback Loop.

2.10.2 Rate-Based Approach

Rate-based flow-control scheme is an end-to-end feedback mechanism as shown in Figure 2-11. There is only one source and one destination station for each feedback loop. When congestion starts, the destination alerts the source to slow cell transmission. The ATM switches between the source and destination simply pass and augment the flow-control information [37].

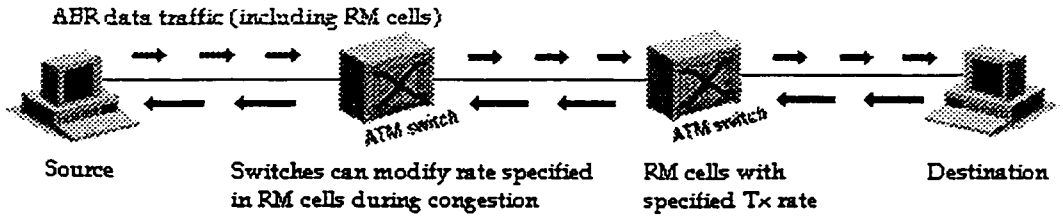

Figure 2-11: Rate-Based Feedback Loop.

In the rate-based scheme, the source of a VC indicates the desired rate of transmission in a Resource Management (RM) cell that is an ATM cell that contains flow-control information. This RM cell travels all the way on the VC to the destination end, then the destination re-sends the RM cell marking it returning in the reverse direction. The ATM switches in the path may decrease the rate allocated for the VC in the reverse RM cell. When the source receives this cell, it will contain the allowed rate by the network.

An extension of the rate-based approach can be used for multi-cast VCs, i.e. a point-to-multipoint connections or broadcasting. As shown in Figure 2-12, the forward RM cell is copied to each branch. Then, the reverse RM cell has its own cell on each branch, where at convergence points the minimum of the converging reverse RM cells is chosen. This procedure is carried on until the source receives the minimum RM cell.

2.10.3 The Great Debate

The debate between credit-based hop-by-hop congestion-control vs. rate-based end-to-end congestion-control resulted in considerable enhancement of the overall quality of the ATM congestion-control techniques for the ABR services. In Table 2-3, we provide a comparison between both techniques [37] & [38].

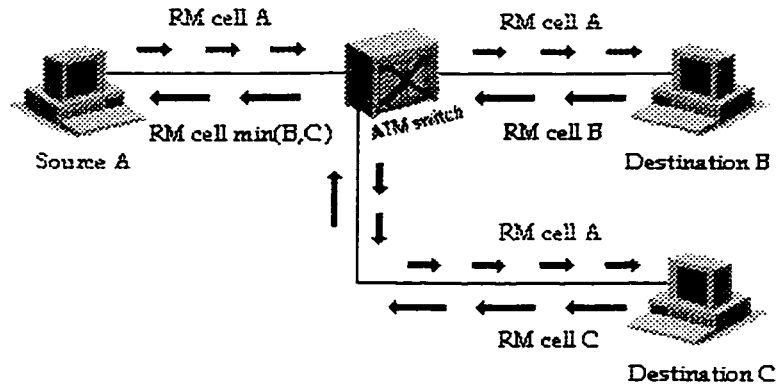

Figure 2-12: Multicasting VCs.

Under the credit-based approach, each VC must maintain a separate queue. For ATM switches supporting a large number of VCs, the per-VC queueing causes a huge complexity on the switch. This was the biggest objection against credit-based especially by switch manufacturers. On the other hand, rate-based approach does not require any queueing for each VC, where the choice is left up to the designers.

The memory requirements for the credit-based approach are tremendous when it comes to the WAN switches, since the per-VC buffering is proportional to the link delay. While in the rate-based, the buffering is proportional to the end-to-end delay.

Table 2-3: Credit-Based vs. Rate-Based Congestion Control.

Credit-based	*Rate-based*
Queueing for each VC is a must and is not scaleable for very large switches.	Queueing for each VC is optional.
Buffer size is proportional to the link delay.	Buffer size is proportional to the end-to-end delay.
Large buffering required of WAN deployment.	Acceptable buffering for both LAN and WAN.
Guarantees zero cell loss	Cannot guarantee zero cell loss
Fast ramp-up time for cell transmission rate.	Could take several round-trip delays to ramp up.

During the debate, many other proposals were truckled, such as a mix and match option "Credit on the LAN, Rate on the WAN", as shown in Figure 2-13. However, such proposals provided a stumbling block to ATM providing a seamless integration between the LAN and WAN environments [37].

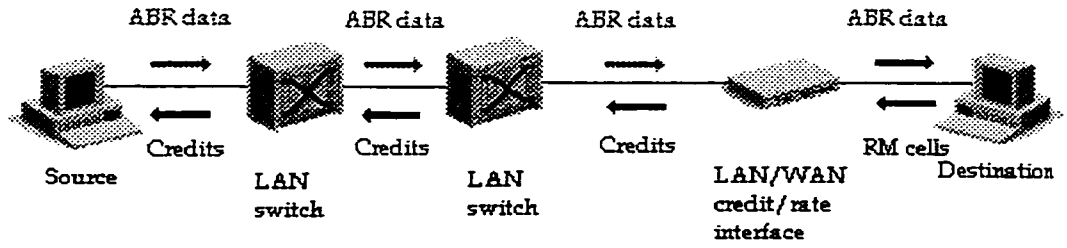

Figure 2-13: Credit on the LAN, Rate on the WAN.

Ultimately, the ATM Forum decided to adopt a rate-only specification for flow control by a vote of 7 to 104. Some of the reasons are: the simplicity provided on both the LAN and WAN by using only one flow control technique, since the credit-based was not practical on the WAN side, it had to be the rate-based. Secondly, less buffering is required by the rate-based especially on the WAN side. Finally, the most important reason for selecting rate-based was the unwillingness to accept a per-VC requirement on the ATM switch [38].

2.11 ATM Signaling

The signaling protocol specifies the sequence of messages that must be exchanged to establish virtual connections. The ATM Forum and the ITU have been working on a standard signaling protocol for ATM. The first standard for signaling specification was drafted by the ITU called Q.93B, then was changed to Q.2931. The ATM Forum's UNI versions 3.1 and 4.0 signaling protocols, which are derived from the ITU specifications, define several messages the source and destination must exchange during the call setup procedure. Some of these messages are shown in Table 2-4. These messages have to traverse the network very quickly and efficiently in order to minimize the call set-up

times. The ATM network uses ATM cells to send these messages. However, to allow the nodes to distinguish between data cells and control message cells, the network uses a protocol discriminator in the header of the cell [36] & [40].

Table 2-4: Signaling Messages.

Message	*Description*
Setup	Sent by the source to the network and by the network to the destination to setup a connection.
Call Processing	Sent by the destination to the network or by the network to the source, to indicate that the connection has been initiated but no further connection information can be accepted.
Connect	Sent by the destination to the network and by the network to the source indicating that an SVC has been established.
Connect Acknowledge	Sent by the source to the network and then to the destination after the source has received the Connect message.
Status Inquiry	Sent by the source, destination or a network switch to inquire about the status of a connection.
Status	Sent in response to a status inquiry message or at any time to report errors.
Release	Sent by the source, destination or a network-switch to clear an end-to-end connection.
Release Complete	Sent by the source, destination or a network switch to indicate that a virtual channel has been released and is available for reuse.

2.11.1 Point-to-Point Connection

To establish a point-to-point connection, the network establishes both forward and backward virtual channels between the source and destination, since each switched virtual channel is a bi-directional link as shown in Figure 2-14.

To establish a simple point-to-point connection, the source sends a setup message that includes the address of the destination. Once the network receives this message, it returns a call proceeding message and sends the setup message to the destination. When the destination receives the setup message, it returns a connect message which is forwarded to the source by the network. Then, the source sends to the network a connect

acknowledge message. When the network receives the connect message it returns to the destination a connect acknowledge message. At this point, the connection is up. When either station is ready to terminate the connection, it sends a release message to the network that returns a release complete across the network terminating the connection [3] & [40].

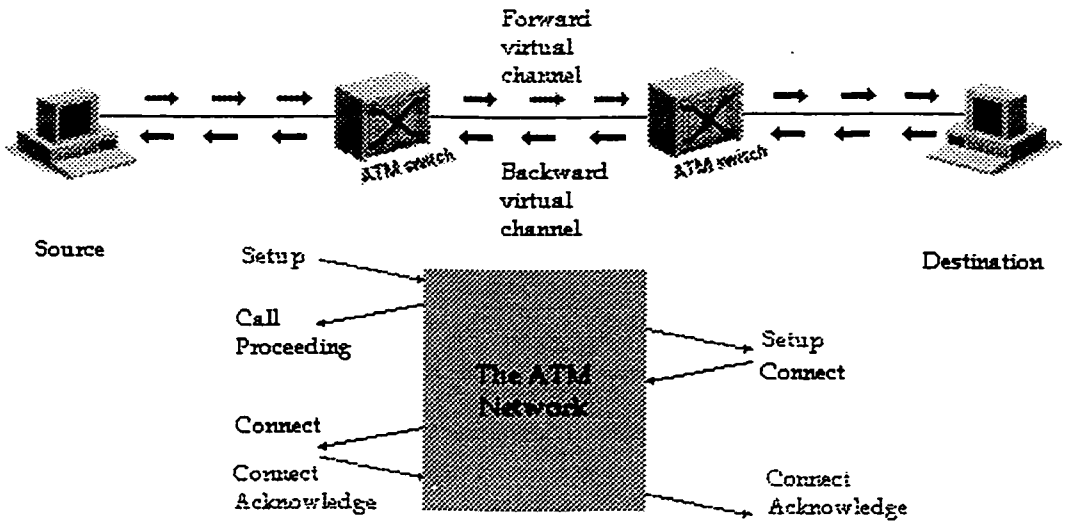

Figure 2-14: Point-to-Point Connection.

2.11.2 Point-to-Multipoint Connection

Point-to-Multipoint is also known as broadcasting or multi-casting, where there is a source station and several destination stations, as shown in Figure 2-15. The source establishes a point-to-point connection with one of the end stations, as described in the previous section. As soon as the source sends the connect acknowledge to the network for the first destination, it signals the second destination using an add-party message. When the switch receives this message, it translates it into a setup message for the second destination. When the second destination receives the setup message, it responds with a connect message. The switch then translates the connect message into an add-party acknowledge message to the source. This process is repeated with the rest of the destinations. Any node can initiate a release message to its connection to the source. If

the entire multi-cast connection is to be terminated at once, the source sends a release message, and the entire VC is released [3] & [40].

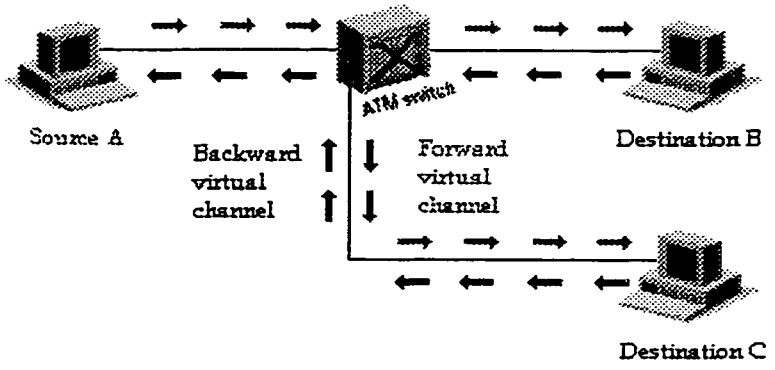

Figure 2-15: Point-to-Multipoint Connection.

2.12 ATM End-to-End Network

This chapter presented the ATM standard. We described the ATM cell structure and the value of each field of the ATM cell. The B-ISDN/ATM reference model was provided. The main functions of each of the three main layers were detailed. The Physical Layer and ATM Layer provide the facilities for the connection-oriented transport of cells. These two protocol layers must be implemented in every ATM device, including end-user hosts as shown in Figure 2-16. Moreover, the AAL layer was presented, and the four AAL protocols were defined and explained.

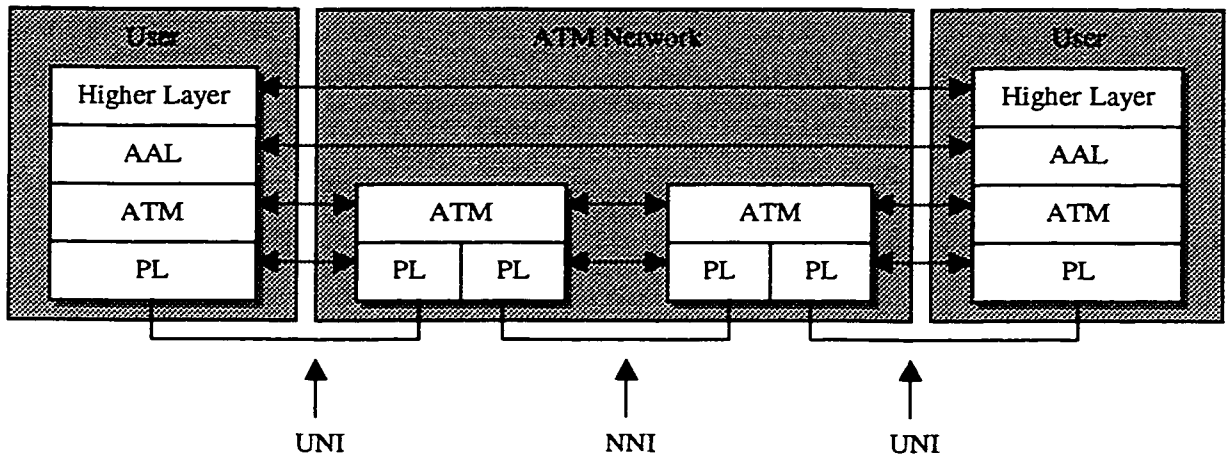

Figure 2-16: The ATM Network.

ATM traffic attributes were presented. We also discussed how these attributes can be specified for each of the four ATM traffic classes. Then, the value for congestion control was explained. Afterwards, the various techniques in traffic control and congestion feedback were provided and compared. Finally, ATM signaling was presented to illustrate connection setup and disconnection for point-to-point and point-to-multipoint connections. Therefore, we presented the ATM network from the cell to the connection, as well as from the source to the destination throughout the B-ISDN/ATM protocol layers to the ATM switches. In the next chapter, we will consider traffic modeling and the queueing analysis necessary to calculate the cell loss probability in an ATM network with heterogeneous traffic.

3. THE QUEUEING SYSTEM

3.1 Introduction

ATM networks are expected to support a wide range of communication services, such as: voice, data and video. In order to evaluate the performance of such networks, we need to accurately model the various services supported [41]. In this chapter, we present the main statistical features of voice, data and video sources. Then, we describe a number of relevant source models. After selecting a source model, we present and analyze a queueing system for a network consisting of an ATM multiplexer with a finite buffer and heterogeneous incoming traffic. Based on the analysis, we derive a formula for calculating the cell loss probability in such a network.

3.2 Traffic Statistical Characteristics

In designing ATM network resources or evaluating control protocols or performance, we need to know relevant statistical characteristics. Generally, traffic corresponds to one of the three main categories: voice, data and video traffic. In this section, we present to pertinent statistical features for each of the traffic categories.

3.2.1 Voice Traffic

Voice traffic is very sensitive to delay; however, it can tolerate moderate cell loss. Cell streams from a voice source can be modeled as alternating talk and silent periods. When a source is in a talk period, it constantly generates cells at a peak rate. However, no cells are generated in a silent period. This model is used for Pulse Code Modulation (PCM), Differential PCM (DPCM) and Adaptive DPCM (ADPCM) [32], [41] & [42].

3.2.2 Data Traffic

Typical data applications generate traffic from a burst state with a continuous cell stream at a peak rate to an idle state with no cells generated. These applications are very sensitive to cell loss; however, they can tolerate certain cell delay [32] & [43].

3.2.3 Video Traffic

Video sources require large bandwidth and generate correlated cell arrivals. The statistical characteristics of traffic generated by a video source is dependent on several aspects: correlation of a line in an image with the next line, correlation of a frame with another frame from the next image, scene correlation and quality of service provided [32] & [41].

3.3 Source Models

Source modeling is required for a precise definition of the behavior of a traffic source. Several traffic source models are used to evaluate connection acceptance or access performance characteristics. Some of these sources models that are relevant to ATM are described below, also Table 3-1 illustrates how these models can be applied to ATM traffic [42] & [44].

3.3.1 Bernoulli Process

The Bernoulli Process, also known as Geometric Process, is a discrete-time process where the probability of a cell arrival is r, and the probability of no cell arrival is $1-r$. This process is memoryless, i.e. a cell arrival is statistically independent of previous arrivals. Also, the inter-arrival time of two successive cells has a geometric distribution [44], [45] & [46].

3.3.2 Renewal Process

The renewal process has inter-arrival times that are statistically independent and identically distributed. Since there is no correlation between inter-arrival times, their

distribution completely defines the process. It is worthwhile stating that the Bernoulli Process is a special case of a renewal process where the inter-arrival times are geometrically distributed [44].

3.3.3 General Modulated Deterministic Process (GMDP)

GMDP is based on a finite state machine with N states, where the inter-arrival time is constant d_i (where i is the state). GDMP has an N x N transition matrix that governs the state changes. Figure 3-1 shows an example of GMDP with $N = 3$. In each state, cells are generated at a constant inter-arrival time. GMDP might also include silent states where no cells are generated. The state changes are covered by an N x N transition matrix $\boldsymbol{P} = [P(i,j)]$, where $P(i,j)$ is the probability that at the end of state i the source moves to state j, $i \neq j$ [44] & [47].

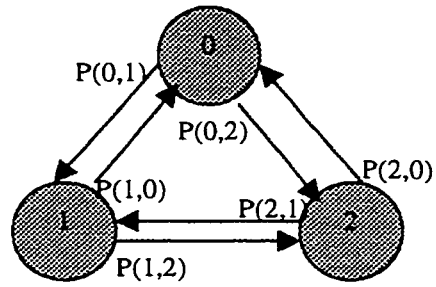

Figure 3-1: Three-State GMDP.

3.3.4 On-Off Model

It is also known as Talkspurt-Silence model or Burst-Silence model. It consists of a two-state Markov model that alternates between an *on* period where cells are generated at a fixed rate, and an *off* period where no cells are generated. The on and off periods are distributed as independent exponential random variables for the continuous case, and as a geometric distribution for the discrete case. Note that the On-Off model is a special case of the GMDP model [31], [44] & [48].

3.3.5 Markov-Modulated Poisson Process (MMPP)

MMPP is a Poisson process where the arrival rate is modulated by the state of a continuous-time discrete-state Markov chain. As shown in Figure 3-2, a two-state MMPP is governed by four parameters: the arrival rates λ_1 and λ_2 for states 1 and 2 respectively, and the remaining time at state 1 and 2 that is exponentially distributed with rates r_1 and r_2, respectively [44], [49] & [50].

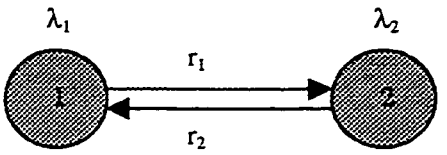

Figure 3-2: Two-State MMPP.

3.3.6 Fluid-Flow (FF) Approximation Method

This method approximates a discrete cell stream by a continuous flow of information. This abstraction covers only the long-term variations of the arrival process being modeled. Thus, modeling short-term queueing increases are not applicable using this method. The fluid-flow approximation method is successfully used in many ATM studies, such as: the Fluid-Flow version of the On-Off source model and the Multi-mini source model [44].

3.3.7 Pareto Modulated Poisson Process (PMPP)

PMPP is a special class of doubly stochastic Poisson process. The model consists of a Poisson process switching between two rates λ_1 and λ_2, as shown in Figure 3-3. The sojourn times in these two states are independent and identically distributed with a Pareto distribution. This model is important in capturing long-term dependence and self-similarity; therefore, PMPP may be used to model self-similar data traffic. The two states of the switched Poisson process correspond to the short and long burst rates of the data traffic [34].

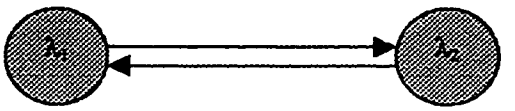

Figure 3-3: PMPP Model.

Table 3-1: Traffic Source Models.

Source Model	*Application to ATM Traffic*
Bernoulli Process	It is a good approximation for modeling short-term and cell level traffic behavior.
Renewal Process	It is characterized by the distribution of the interarrival times; however, it neglects the correlation structure of the original process. This process is used for modeling CBR and video conferencing traffic.
GMDP	GMDP can be used for interactive data, LAN and retrieval of still images. It is unable to model any correlation structures such as non-buffered video codecs.
On-Off Model	It can be used for interactive data, LAN and retrieval of still images.
MMPP	It is useful to approximate the cell loss probability for small buffers only.
FF Approximation	The Multi-mini source model is used to approximate video and HDTV traffic. While the FF version of the On-Off source can be used for interactive data and LAN traffic.
PMPP	It is important in capturing long-term dependence and self-similarity that make it suitable to model self-similar data traffic.

3.3.8 Which Source Model should be selected?

We chose the On-Off source model to approximate the actual process since it is suitable for highly correlated traffic such as data, LAN and image retrieval traffic. Even though, this model is not suitable for cell-level congestion for which good cell loss approximation is presented in [51] and [52], the On-Off source model provides a very good approximation for burst-level congestion [31].

Our work is based on an ATM multiplexer with a finite buffer of size K (cells) and a single link with capacity C (cells/second). Incoming traffic to the ATM multiplexer is generated from M On-Off sources that consist of two classes of traffic sources: the first class has M_1 identical sources, and the second has M_2 identical sources. Each On-Off source can be in two states: on state in which cells are generated at a fixed rate, or off state where no cells are generated. The on and off periods are distributed as independent exponential random variables with parameters μ_j and λ_j respectively, where j is the traffic class. Our traffic model parameters are summarized in Table 3-2 and illustrated in Figure 3-4. Moreover, the relationship amongst these parameters is shown below.

$$\begin{aligned} b_j &= 1+\mu_j / \lambda_j \\ \phi_j &= \Delta_j / b_j \\ L_j &= \Delta_j / \mu_j \end{aligned} \qquad (3\text{-}1)$$

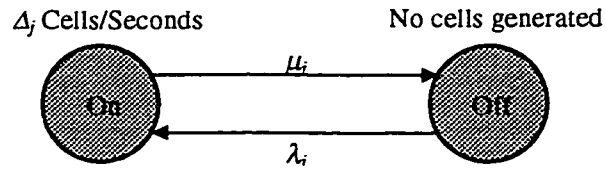

Figure 3-4: On-Off Source Model for Class *j*.

Table 3-2: On-Off Source Model Parameters.

Parameter	Unit	Description
Δ_j	cell/second	Peak rate is a fixed rate at which cells are generated from a source of class j during an on period.
$1/\mu_j$	Second	Mean duration of an on period of a source of class j.
$1/\lambda_j$	Second	Mean duration of an off period of a source of class j.
Φ_j	cell/second	Mean rate is the overall number of cells generated per second from a source of class j.
b_j	None	Burstiness is the ratio between the peak rate and the mean rate, where j is the traffic class.
L_j	cell	Burst length is the average number of cells generated from a source of class j during an on period.

3.4 Queueing Analysis

Our queueing system is an MMDP/D/1/K which consists of a single server with a deterministic service rate of C (cells/second) and a queue with a maximum length of K cells, as shown in Figure 3-5. The actual cell arrival process is approximated using a Markov Modulated Deterministic Process (MMDP). We consider two classes of traffic sources: M_1 class 1 sources and M_2 class 2 sources. All sources are independent, and sources within a class are identical. Each traffic source is modeled as an On-Off source model with parameters Δ_1, μ_1, λ_1 for class 1 and Δ_2, μ_2, λ_2 for class 2, as described in the previous section.

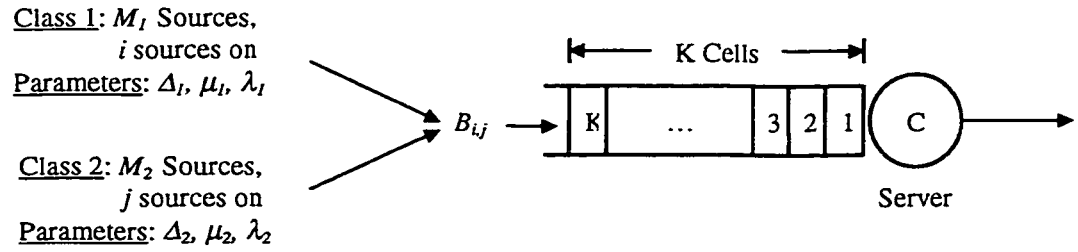

Figure 3-5: MMDP/D/1/K Queueing System for an ATM MUX with Heterogeneous Traffic.

3.4.1 MMDP Arrival Process

Let $X(t)$ be the state of the modulating Markov process at time t, where $t \geq 0$, also E_n ($n = 0, 1, 2, \ldots$) as the n^{th} transition epoch of $X(t)$ [31], as shown in Figure 3-6. We define an M x M probability transition matrix $\boldsymbol{P}$, where $M = (m_1 + 1)(m_2 + 1)$. Let (i, j) be the current state of class 1 and 2 sources, respectively, while, (i',j') indicates the next state of class 1 and 2 sources, respectively. Thus, the probability transition matrix of $X(t)$ can be written as $\boldsymbol{P} = [p_{(i,j),(i',j')}]$, where $0 \leq i,i' \leq m_1$ and $0 \leq j,j' \leq m_2$. Let $1/\gamma_{i,j}$ be the mean sojourn time for state (i,j), and define a random variable $U_{i,j} = E_{n+1} - E_n$ that is the interval $[E_n, E_{n+1})$ during which $X(t)$ is in state (i,j). This random variable is exponentially distributed with parameter $\gamma_{i,j}$, i.e. $\Pr\{ U_{i,j} \} = \gamma_{i,j} \exp(-U_{i,j} \gamma_{i,j})$. Also, let $B_{i,j}$ be a fixed cell arrival rate (cells/second) to the buffer when the $X(t)$ is in state (i,j). Let $X_n = X(t)$, where $t \in [E_n, E_{n+1})$. If $X_n = (i,j)$, then during the interval $[E_n, E_{n+1})$, we know that $X(t)$ is in state (i,j) where the cells arrive to the buffer at fixed rate of $B_{i,j}$. Note that the first arrival will occur at $E_n + 1/B_{i,j}$ due to the deterministic process assumed. Thus, we completely defined our MMDP arrival process by the matrices $\boldsymbol{P} = [p_{(i,j),(i',j')}]$, $\Gamma = [\gamma_{i,j}]$ and $\boldsymbol{B} = [B_{i,j}]$.

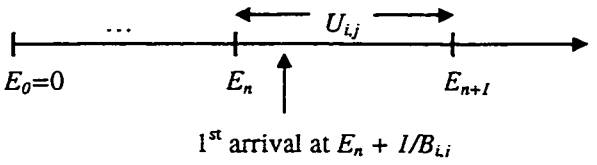

Figure 3-6: $X(t)$ epoch transitions during state (i,j).

3.4.2 The Embedded Process

Let $Y(t)$ be the number of cells in the buffer at time t (including the cell being served). Assume that $Y_n = Y(t = E_n)$, i.e. Y_n is the number of cells in the buffer including the one being served at the beginning of state (i,j), also assume Y_{n+1} to be the number of cells in the buffer including the one being served at the end of state (i,j). To simplify the analysis for the embedded process $\{(X_n, Y_n), n \geq 0\}$, we make two assumptions which are

suitable for ATM applications. The first is called **service renewal assumption**: if E_n occurs while a cell is still receiving service, then the entire cell is to be retransmitted after E_n. As a result, the embedded process becomes a Markov chain. The second assumption is the **continuity assumption** in which we relate Y_{n+1} and Y_n by assuming that both the cell inter-arrival times and the cell service times are extremely small in comparison to the length of the interval $E_{n+1} - E_n$ [31]. Let us consider these assumptions with the following two cases:

- Let $B_{i,j} > C$ (i.e. the arrival rate in state (i,j) > the service rate), this also indicates that the queue size will not decrease during the interval $[E_n, E_{n+1})$:

$$Y_{n+1} = \min\{(\text{full buffer size}) \text{ OR } (\text{no. of cells in the buffer at the end of state } (i,j))\}$$

$$Y_{n+1} = \min\{K, \text{ no. of cells in the buffer at the end of state } (i,j)\}$$

$$Y_{n+1} = \min\{K, Y_n + (\text{no. of arrivals - no. of departures during state } (i,j))\}$$

$$Y_{n+1} = \min\{K, Y_n + \lfloor \text{the buffer size rate of change} \times \text{the time interval of state } (i,j) \rfloor\}$$

$$Y_{n+1} = \min\{K, Y_n + \lfloor (B_{i,j} - C)\, U_{i,j} \rfloor\}$$

Using the continuity assumption, $Y_{n+1} = \min\{K, Y_n + \lfloor (B_{i,j} - C)\, U_{i,j} \rfloor\}$

where $\lfloor w \rfloor$ denotes the largest integer smaller than or equal to w.

- Let $B_{i,j} < C$ (i.e. the arrival rate in state (i,j) < the service rate), this also indicates that the queue size will not increase during the interval $[E_n, E_{n+1})$:

$$Y_{n+1} = \max\{(\text{empty buffer}) \text{ OR } (\text{no. of cells in the buffer at the end of state } (i,j))\}$$

$$Y_{n+1} = \max\{0, Y_n - \lfloor (C - B_{i,j})\, U_{i,j} \rfloor\}$$

Using the continuity assumption, $Y_{n+1} = \max\{0, Y_n - \lfloor (C - B_{i,j})\, U_{i,j} \rfloor\}$

Using these assumptions, the solution for the transition probabilities of the embedded Markov chain $\{(X_n, Y_n), n \geq 0\}$ can be simplified, since the distributions of $\lfloor (B_{i,j} - C)\, U_{i,j} \rfloor$ and $\lfloor (C - B_{i,j})\, U_{i,j} \rfloor$ are geometric and can be easily calculated, as shown below:

- For $B_{i,j} > C$, since the buffer size will not decrease in state (i,j), we define $\rho_{i,j}$ as the probability of a cell arriving and remaining in the buffer in state (i,j), i.e. $(1 - \rho_{i,j})$ is the probability that the arriving cells will not remain in the buffer in state (i,j). Now, consider $\lfloor (B_{i,j} - C)\, U_{i,j} \rfloor$:

$$\Pr\{ (B_{i,j} - C)\, U_{i,j} < 1\} = \Pr\{ U_{i,j} < 1/(B_{i,j} - C) \}$$

But, $U_{i,j}$ is exponentially distributed with parameter $\gamma_{i,j}$, i.e. PDF for $U_{i,j}$ is $\gamma_{i,j} \exp(-U_{i,j}\gamma_{i,j})$

$$\therefore \Pr\left\{U_{i,j} < 1/\left(B_{i,j} - C\right)\right\} = \int_0^{1/(B_{i,j}-C)} \gamma_{i,j} e^{-\gamma_{i,j} x} dx = 1 - \rho_{i,j} \qquad (3\text{-}2)$$

Also, $\Pr\{ U_{i,j} \geq 1/(B_{i,j} - C) \} = \rho_{i,j}$, where $\rho_{i,j} = \exp\{- \gamma_{i,j} / (B_{i,j} - C)\}$

$\Pr\{ \lfloor (B_{i,j} - C)\, U_{i,j} \rfloor = k\} = \Pr\{$(no. of arrivals - no. of departures in state (i,j))$=k\}$

Thus, $\Pr\{ \lfloor (B_{i,j} - C)\, U_{i,j} \rfloor = k\} = \rho^k_{i,j} \cdot (1 - \rho_{i,j})$ for $k = 0, 1, \ldots$

- For $B_{i,j} < C$, since the buffer size will not increase in state (i,j), we define $\rho_{i,j}$ as the probability of an arriving cell will not remain in the buffer in state (i,j), i.e. $(1 - \rho_{i,j})$ is the probability that the arriving cells remain in the buffer in state (i,j). Let us consider $\lfloor (C - B_{i,j})\, U_{i,j} \rfloor$:

$\Pr\{ \lfloor (C - B_{i,j})\, U_{i,j} \rfloor = k\} = \Pr\{$(no. of departures - no. of arrivals during state (i,j))$=k\}$

Similarly, $\Pr\{ \lfloor (C - B_{i,j})\, U_{i,j} \rfloor = k\} = \rho^k_{i,j} \cdot (1 - \rho_{i,j})$ for $k = 0, 1, \ldots$

where $\rho_{i,j} = \exp\{- \gamma_{i,j} / (C - B_{i,j})\}$, since $U_{i,j}$ is exponentially distributed with parameter $\gamma_{i,j}$.

Therefore, the transition probability matrix of the embedded Markov chain can be expressed as $\boldsymbol{Q} = [q_{(i,j,k),(i',j',k')}]$, where $q_{(i,j,k),(i',j',k')}$ is the transition probability from state (i,j,k) to state (i',j',k'), $0 \leq i,i' \leq m_1$, $0 \leq j,j' \leq m_2$, $0 \leq k,k' \leq K$, k is the number of cells in the buffer (including the one being served) at the beginning of state (i,j) and k' is the number of cells in the buffer (including the one being served) at the end of state (i,j). Then,

$q_{(i,j,k),(i',j',k')} = \Pr\{Y_{n+1} = k'$ given that $(X_n = (i,j)$ and $Y_n = k)\}$ and $p_{(i,j),(i',j')}$

$q_{(i,j,k),(i',j',k')} = \Pr\{Y_{n+1} = k' \mid (X_n = (i,j),\ Y_n = k)\}\ p_{(i,j),(i',j')}$

$$q_{(i,j,k),(i',j',k')} = a^{(i,j)}_{(k,k')}\ p_{(i,j),(i',j')} \qquad \text{(3-3)}$$

where $p_{(i,j),(i',j')}$ *is an element of the transition matrix* $\boldsymbol{P}$

and $a^{(i,j)}_{(k,k')} = \Pr\{Y_{n+1} = k' \mid (X_n = (i,j),\ Y_n = k)\}$

3.4.3 Buffer Occupancy

Let $\boldsymbol{A}_{i,j}$ be the conditional probability transition matrix for the buffer occupancy during state (i,j) with $(K + 1)$ x $(K + 1)$ elements. Thus, $\boldsymbol{A}_{i,j} = [a^{(i,j)}_{(k,k')}]$, where $a^{(i,j)}_{(k,k')}$ is the conditional probability that the buffer size changes from k to k' at the beginning and end of state (i,j), respectively. Using $\boldsymbol{A}_{i,j}$, we can partition $\boldsymbol{Q}$ into blocks of $(K + 1)$ x $(K + 1)$. As a result, $\boldsymbol{Q}$ becomes an M x M matrix, where $\boldsymbol{Q} = [p_{(i,j),(i',j')}\ \boldsymbol{A}_{i,j}]$. To calculate $a^{(i,j)}_{(k,k')}$, we consider the following three cases:

- For $B_{i,j} = C$, we consider two cases: when the buffer is not empty at the beginning of state (i,j), the buffer size will not change at the end of that state. The second case, when the buffer is empty at the beginning of state (i,j), it will either be empty or contain only one cell at the end of that state due to the service renewal assumption.

$$a^{i,j}_{k,k'} = \begin{cases} 1 - e^{-\gamma_{i,j}/C}, & k = k' = 0 \\ e^{-\gamma_{i,j}/C}, & k = 0, k' = 1 \\ 1, & k = k' > 0 \\ 0, & \textit{otherwise} \end{cases} \qquad \text{(3-4)}$$

- For $B_{i,j} < C$, we know that the buffer size will not increase in state (i,j), i.e. $\Pr\{k<k'\}=0$. We are left with two cases: the buffer is not empty ($k' \geq 1$), and the buffer is empty ($k' = 0$) at the end of state (i,j).

$$a_{k,k'}^{i,j} = \begin{cases} \Pr\{k < k'\} = 0, & k < k' \\ \Pr\left\{\left\lfloor \left(C - B_{i,j}\right) U_{i,j} \right\rfloor = k - k'\right\} = \rho_{i,j}^{k-k'}\left(1-\rho_{i,j}\right), & k \geq k' \geq 1 \\ \Pr\left\{\left\lfloor \left(C - B_{i,j}\right) U_{i,j} \right\rfloor \geq k\right\} = \sum_{w=k}^{\infty} \rho_{i,j}^{w}\left(1-\rho_{i,j}\right) = \rho_{i,j}^{k}, & k \geq k' = 0 \end{cases} \quad (3\text{-}5)$$

- For $B_{i,j} > C$, we know that the buffer size will not decrease in state (i,j), i.e. $P\{k>k'\}=0$. We are left with two cases: the buffer is not full ($k' < K$), and the buffer is full ($k' = K$) at the end of state (i,j).

$$a_{k,k'}^{i,j} = \begin{cases} \Pr\{k > k'\} = 0, & k > k' \\ \Pr\left\{\left\lfloor \left(B_{i,j} - C\right) U_{i,j} \right\rfloor = k' - k\right\} = \rho_{i,j}^{k'-k}\left(1-\rho_{i,j}\right), & k \leq k' < K \\ \Pr\left\{\left\lfloor \left(B_{i,j} - C\right) U_{i,j} \right\rfloor \geq K - k\right\} = \sum_{w=K-k}^{\infty} \rho_{i,j}^{w}\left(1-\rho_{i,j}\right) = \rho_{i,j}^{K-k}, & k \leq k' = K \end{cases} \quad (3\text{-}6)$$

Let the limiting probability $\pi_{i,j,k} = \lim_{n\to\infty} \Pr\{X_n = (i,j),\ Y_n = k\}$, also let $\boldsymbol{\pi}_{i,j} = [\pi_{i,j,0}, \pi_{i,j,1}, \ldots, \pi_{i,j,k}, \ldots, \pi_{i,j,K}]$ and $\boldsymbol{\Pi} = [\boldsymbol{\pi}_{i,j}]$. Note that $\boldsymbol{\pi}_{i,j}$ has $K+1$ elements, and $\boldsymbol{\Pi}$ has M vectors. Since the embedded process $\{(X_n, Y_n),\ n \geq 0\}$ is a Markov chain, then $\boldsymbol{\Pi} = \boldsymbol{\Pi}\boldsymbol{Q}$. By solving the set of equations, we can compute $\boldsymbol{\pi}_{i,j}$:

$$\pi_{i,j} = \sum_{i \neq i'} \sum_{j \neq j'} p_{(i,j),(i',j')} \pi_{i,j} A_{i,j} \text{ and } \sum_{i=0}^{M_1} \sum_{j=0}^{M_2} \pi_{i,j} e^T = 1 \quad (3\text{-}7)$$

where e is the unity vector with $(K + 1)$ elements.

3.4.4 Cell Loss Probability Derivation

To compute the cell loss probability p, we need to define two more random variables. First, let $N_{i,j,k}$ be the total number of cells arrived during the interval $[E_n, E_{n+1})$ given that $X_n = (i,j)$ and $Y_n = k$.

$$N_{i,j,k} = \lfloor \text{arrival rate x length of the interval } [E_n, E_{n+1}) \rfloor = \lfloor B_{i,j}\ U_{i,j} \rfloor$$

$$\Pr\{ B_{i,j}\ U_{i,j} > 1 \} = \Pr\{ U_{i,j} > 1 / B_{i,j} \} = 1 - \exp\{-\gamma_{i,j} / B_{i,j}\},$$

$$\text{and } \Pr\{ B_{i,j}\ U_{i,j} \geq 1 \} = \exp\{-\gamma_{i,j} / B_{i,j}\}, \text{ for } B_{i,j} > 0$$

since $U_{i,j}$ is exponentially distributed with parameter $\gamma_{i,j}$, i.e. $\Pr\{U_{i,j}\} = \gamma_{i,j} \exp(-U_{i,j} \gamma_{i,j})$

$$\Pr\{ \lfloor B_{i,j}\ U_{i,j} \rfloor = x \} = \exp^{x}\{-\gamma_{i,j} / B_{i,j}\} (1 - \exp\{-\gamma_{i,j} / B_{i,j}\}) \text{ for } x = 0, 1, \ldots$$

i.e. $N_{i,j,k}$ is geometrically distributed with paramter $\exp\{-\gamma_{i,j} / B_{i,j}\}$.

$$\text{Therefore, the mean} = E[N_{i,j,k}] = \exp\{-\gamma_{i,j} / B_{i,j}\} / (1 - \exp\{-\gamma_{i,j} / B_{i,j}\})$$

The second random variable is $R_{i,j,k}$ defined as the total number of cells lost during the interval $[E_n, E_{n+1})$ given that $X_n = (i,j)$ and $Y_n = k$. For $B_{i,j} \leq C$, $\Pr\{R_{i,j,k} = 0\} = 1$. Thus, we only consider the case $B_{i,j} > C$, applying the continuity approximation:

$$R_{i,j,k} = \max\{0, \lfloor (B_{i,j} - C)\ U_{i,j} \rfloor - K + k\}$$

$$\Pr\{ R_{i,j,k} = x \} = \Pr\{ \lfloor (B_{i,j} - C)\ U_{i,j} \rfloor = K - k + x \}$$

$$\text{Thus, } \Pr\{ R_{i,j,k} = x \} = \rho_{i,j}^{(K-k+x)} (1 - \rho_{i,j}) \text{ which is geometrically distributed}$$

$$\text{where } \rho_{i,j} = \exp\{- \gamma_{i,j} / (B_{i,j} - C)\}, \text{ for } B_{i,j} > C$$

$$\text{Therefore, the mean} = E[R_{i,j,k}] = \rho_{i,j}^{(K-k+1)} / (1 - \rho_{i,j})$$

Now, we are able to estimate the cell loss p in terms of R, N and π.

p = (mean cell loss in a period) / (mean cell arrivals in a period)

$$p = \frac{\sum_{(i,j) \in V} \sum_{k=0}^{K} E\left[R_{i,j,k}\right] \pi_{i,j,k}}{\sum_{i=0}^{M_1} \sum_{j=0}^{M_2} \sum_{k=0}^{K} E\left[N_{i,j,k}\right] \pi_{i,j,k}} \qquad \text{(3-8)}$$

$$\text{where } V = \left\{(i,j) : B_{i,j} > C\right\}$$

3.5 Conclusion

In this chapter, we presented several source models for characterizing traffic sources. Then, we selected the On-Off source model, as it is suitable for correlated traffic such as voice, image retrieval and data. Afterwards, we analyzed an ATM multiplexer

with a finite buffer and a single output link with incoming traffic generated from heterogeneous On-Off sources. We also detailed our analysis of the arrival process, embedded process and buffer occupancy. Based on our analysis, we provided a derivation of a new method to calculate the cell loss probability of the ATM multiplexer loaded with heterogeneous traffic.

4. CELL LOSS PROBABILITY ESTIMATION

4.1 Introduction

In the previous chapter, we provided a new technique to obtain the cell loss probability of an ATM multiplexer with a finite buffer and loaded with heterogeneous traffic. In this chapter, we present a method to approximate the analysis and results presented earlier. Furthermore, we provide an algorithm to compute the cell loss probability efficiently. Finally, we discuss the design methodology used in the implementation of this algorithm.

4.2 Cell Loss Estimation

In order to be able to calculate the cell loss probability, we need to approximate the arrival the Markov process $X(t)$ by a finite birth-death process with a birth rate of $\lambda_{i,j}$ and a death rate of $\mu_{i,j}$, where i and j are the number of on sources from class 1 and class 2, respectively. Thus, $m_1 = M_1$ and $m_2 = M_2$. In the next sections, we will present the transition state diagram for the finite birth-death process. Based on this approximation, we will simplify the cell loss probability formula presented in the previous chapter.

4.2.1 Transition States

We define the birth and death probabilities for class 1 as $\alpha_{1,i,j}$ and $\beta_{1,i,j}$, respectively, and the birth and death probabilities for class 2 as $\alpha_{2,i,j}$ and $\beta_{2,i,j}$, respectively. Figure 4-1 shows the transition state diagram for the approximate arrival process, and it also provides the relationship among the birth and death probabilities of this process. The following equations demonstrate how the birth and death processes are related to the On-Off source model parameters of each class:

$$\lambda_{i,j} = (M_1 - i)\,\lambda_1 + (M_2 - j)\,\lambda_2$$
$$\mu_{i,j} = i\,\mu_1 + j\,\mu_2$$
$$\gamma_{i,j} = \lambda_{i,j} + \mu_{i,j}$$
$$B_{i,j} = i\,\Delta_1 + j\,\Delta_2$$

$$\alpha_{1,i,j} = (M_1 - i)\,\lambda_1 / \gamma_{i,j} \qquad \textbf{(4-1)}$$
$$\alpha_{2,i,j} = (M_2 - i)\,\lambda_2 / \gamma_{i,j}$$
$$\beta_{1,i,j} = i\,\mu_1 / \gamma_{i,j}$$
$$\beta_{2,i,j} = j\,\mu_2 / \gamma_{i,j}$$

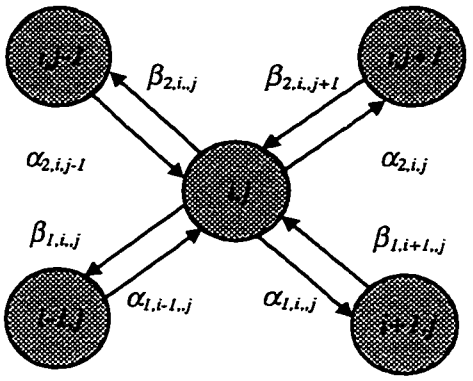

Figure 4-1: *X(t)* Transition State Diagram.

Based on these parameters, we can calculate the transition probability matrix $\boldsymbol{P} = [p_{(i,j),(i',j')}]$ and the marginal probability $p_{i,j} = \lim_{n\to\infty} \Pr\{X_n = (i,j)\}$. Note that $p_{i,j}$ is equal to $\pi_{i,j}\, e^T$.

$$p_{(i,j),(i',j')} = \begin{cases} \alpha_{1,i,j}, & i = 0,1,\ldots, M_1 - 1 \quad i' = i'+1 \quad j = j' \\ \beta_{1,i,j}, & i = 1,2,\ldots, M_1 \quad i' = i - 1 \quad j = j' \\ \alpha_{2,i,j}, & j = 0,1,\ldots, M_2 - 1 \quad j' = j + 1 \quad i = i' \\ \beta_{2,i,j}, & j = 1,2,\ldots, M_2 \quad j' = j - 1 \quad i = i' \\ 0, & otherwise \end{cases} \qquad \textbf{(4-2)}$$

$$p_{i,j} = p_{0,0} \frac{\left(\prod_{l=0}^{j-1} \alpha_{2,i,l}\right)\left(\prod_{l=0}^{i-1} \alpha_{1,l,0}\right)}{\left(\prod_{l=1}^{j} \beta_{2,i,l}\right)\left(\prod_{l=1}^{i} \beta_{1,l,0}\right)} \tag{4-3}$$

$$\text{where } p_{0,0} = \left(1 + \sum_{i=0}^{M_1} \sum_{j=1}^{M_2} \frac{\left(\prod_{l=0}^{j-1} \alpha_{2,i,l}\right)\left(\prod_{l=0}^{i-1} \alpha_{1,l,0}\right)}{\left(\prod_{l=1}^{j} \beta_{2,i,l}\right)\left(\prod_{l=1}^{i} \beta_{1,l,0}\right)} + \sum_{i=1}^{i=M_1} \frac{\left(\prod_{l=0}^{i-1} \alpha_{1,l,0}\right)}{\left(\prod_{l=1}^{i} \beta_{1,l,0}\right)}\right)^{-1} \tag{4-4}$$

$$\text{since } \sum_{i=0}^{M_1} \sum_{j=0}^{M_2} p_{i,j} = 1 \tag{4-5}$$

4.2.2 Solving for the Embedded Process

At this stage, we are able to compute all marginal and transitional probabilities for the Markov process $X(t)$ by using μ_1, λ_1, μ_2 and λ_2. Now, let us consider the embedded process $\{(X_n, Y_n), n \geq 0\}$ which is a Markov chain with $\Pi = \Pi Q$, as defined in the previous section. Since $Q = [p_{(i,j),(i',j')} A_{i,j}]$,

$$Q_{M \times M} = \begin{bmatrix} & (0,0) & (1,0) & (2,0) & \cdots & (M_1, M_2) \\ (0,0) & 0 & p_{(0,0),(1,0)} A_{0,0} & 0 & & \\ (1,0) & p_{(1,0),(0,0)} A_{1,0} & 0 & p_{(1,0),(2,0)} A_{1,0} & & \\ (2,0) & 0 & p_{(2,0),(1,0)} A_{2,0} & 0 & & \\ \cdots & & & & & \\ (M_1, M_2) & & & & & \end{bmatrix} \tag{4-6}$$

$$Q_{M\times M} = \begin{bmatrix} & (0,0) & (1,0) & (2,0) & \dots & (M_1, M_2) \\ (0,0) & 0 & \alpha_{1,0,0} A_{0,0} & 0 & & \\ (1,0) & \beta_{1,1,0} A_{1,0} & 0 & \alpha_{1,1,0} A_{1,0} & & \\ (2,0) & 0 & \beta_{1,2,0} A_{2,0} & 0 & & \\ \dots & & & & & \\ (M_1, M_2) & & & & & \end{bmatrix} \quad (4\text{-}7)$$

We can solve for $\Pi = [\pi_{i,j}]$ from $\Pi = \Pi Q$, as shown below:

$$\pi_{0,0} = \beta_{1,1,0}\, \pi_{1,0}\, A_{1,0} + \beta_{2,0,1}\, \pi_{0,1}\, A_{0,1}$$

$$\pi_{1,0} = \alpha_{10,0}\, \pi_{0,0}\, A_{0,0} + \beta_{1,1,2}\, \pi_{1,2}\, A_{1,2} + \beta_{2,1,1}\, \pi_{1,1} A_{1,1}$$

...

$$\pi_{0,1} = \beta_{1,1,1}\, \pi_{1,1}\, A_{1,1} + \alpha_{2,0,0}\, \pi_{0,0} A_{0,0} + \beta_{2,0,2}\, \pi_{0,2} A_{0,2}$$

$$\pi_{1,1} = \alpha_{1,1,0}\, \pi_{1,0}\, A_{1,0} + \beta_{1,2,1}\, \pi_{2,1}\, A_{2,1} + \alpha_{2,1,0}\, \pi_{1,0} A_{1,0} + \beta_{2,1,2}\, \pi_{1,2} A_{1,2}$$

...

$$\therefore \pi_{i,j} = \alpha_{1,i-1,j}\, \pi_{i-1,j}\, A_{i-1,j} + \beta_{1,i+1,j}\, \pi_{i+1,j} A_{i+1,j} + \alpha_{2,i,j-1}\, \pi_{i,j-1} A_{i,j-1} + \beta_{2,i,j+1}\, \pi_{i,j+1} A_{i,j+1} \quad (4\text{-}8)$$

where $\alpha_{1,i,j} = \alpha_{2,i,j} = 0$, if $i < 0$ or $j < 0$

and $\beta_{1,i,j} = \beta_{2,i,j} = 0$, if $i > M_1$ or $j > M_2$

4.2.3 The Linear Transformation

To simplify the calculation of $\pi_{i,j}$, we introduce the linear transformation $y_{i,j} = \pi_{i,j} \cdot A_{i,j}$, where $y_{i,j} = [y_{i,j,0}, y_{i,j,1}, \dots, y_{i,j,K}]$. Note that, $y_{i,j,k} = \sum_{l=0}^{K} \pi_{i,j,l} \alpha_{i,j}^{l,k}$. By multiplying the equation derived for $\pi_{i,j}$ by $A_{i,j}$, we obtain:

$$\pi_{i,j} A_{i,j} = (\alpha_{1,i-1,j}\, \pi_{i-1,j}\, A_{i-1,j} + \beta_{1,i+1,j}\, \pi_{i+1,j} A_{i+1,j} + \alpha_{2,i,j-1}\, \pi_{i,j-1} A_{i,j-1} + \beta_{2,i,j+1}\, \pi_{i,j+1} A_{i,j+1})\, A_{i,j}$$

$$y_{i,j} = (\alpha_{1,i-1,j}\, y_{i-1,j} + \beta_{1,i+1,j}\, y_{i+1,j} + \alpha_{2,i,j-1}\, y_{i,j-1} + \beta_{2,i,j+1}\, y_{i,j+1})\, A_{i,j} \quad (4\text{-}9)$$

Note that $y_{i,j}\, e^T = \pi_{i,j}\, A_{i,j}\, e^T = \pi_{i,j}\, e^T = p_{i,j}$

Now, we can calculate the cell loss probability p based on N, R and the marginal probabilities for the embedded process:

$$p = \frac{\sum_{(i,j)\in V}\sum_{k=0}^{K}\frac{\rho_{i,j}^{K-k+1}}{1-\rho_{i,j}}\pi_{i,j,k}}{\left(\sum_{i=1}^{M_1}\sum_{j=0}^{M_2}\sum_{k=0}^{K}\frac{e^{-\gamma_{i,j}/B_{i,j}}}{1-e^{-\gamma_{i,j}/B_{i,j}}}\pi_{i,j,k}\right)+\left(\sum_{j=1}^{M_2}\sum_{k=0}^{K}\frac{e^{-\gamma_{0,j}/B_{0,j}}}{1-e^{-\gamma_{0,j}/B_{0,j}}}\pi_{0,j,k}\right)} \quad \textbf{(4-10)}$$

$$p = \frac{\sum_{(i,j)\in V}\frac{\rho_{i,j}}{1-\rho_{i,j}}\sum_{k=0}^{K}\rho_{i,j}^{K-k}\pi_{i,j,k}}{\left(\sum_{i=1}^{M_1}\sum_{j=0}^{M_2}\frac{e^{-\gamma_{i,j}/B_{i,j}}}{1-e^{-\gamma_{i,j}/B_{i,j}}}\sum_{k=0}^{K}\pi_{i,j,k}\right)+\left(\sum_{j=1}^{M_2}\frac{e^{-\gamma_{0,j}/B_{0,j}}}{1-e^{-\gamma_{0,j}/B_{0,j}}}\sum_{k=0}^{K}\pi_{0,j,k}\right)} \quad \textbf{(4-11)}$$

$$\text{But, } \sum_{k=0}^{K}\pi_{i,j,k} = \pi_{i,j}e^{T} = p_{i,j}$$

$$\text{Also for } B_{i,j} > C,\ \sum_{k=0}^{K}\rho_{i,j}^{K-k}\pi_{i,j,k} = \sum_{k=0}^{K}a_{i,j}^{k,K}\pi_{i,j,k} = y_{i,j,k}$$

Therefore,

$$p = \frac{\sum_{(i,j)\in V}\frac{\rho_{i,j}}{1-\rho_{i,j}}y_{i,j,K}}{\left(\sum_{i=1}^{M_1}\sum_{j=0}^{M_2}\frac{e^{-\gamma_{i,j}/B_{i,j}}}{1-e^{-\gamma_{i,j}/B_{i,j}}}p_{i,j}\right)+\left(\sum_{j=1}^{M_2}\frac{e^{-\gamma_{0,j}/B_{0,j}}}{1-e^{-\gamma_{0,j}/B_{0,j}}}p_{0,j}\right)} \quad \textbf{(4-12)}$$

4.3 Calculating Cell Loss Probability

To determine the cell loss probability p, we need to calculate $y_{i,j,K}$. We will use a modified version of the algorithm presented in [31] to find the value $y_{i,j,K}$. Finally, the cell loss probability, p, can be obtained as all the unknowns are now available.

4.3.1 Proposed Algorithm

To calculate $y_{i,j,K}$ and p, we proposed the algorithm which is based on the aggregation-disaggregation algorithm for large Markov chains as shown in Figure 4-2

[53]. Initially, `Y(0)` is uniformly distributed, `k` is equal to zero and `delta` is a defined error tolerance value. The algorithm consists of three loops. The middle and inner for-loops iterate through all of the possible states of class 1 and class 2 sources in order to calculate the probability `y(i,j,k+1)`. Then, we perform a local normalization to ensure that the calculated probabilities are accurate. The outer while-loop iterates through the calculation of `y(i,j,k+1)` and `p_current` until either k is equal to `k_max` or `p_current` has convereged.

```
k = 0;
p_previous = 0;
initialize_Y();                          //Y(0) is uniformly distributed
get_transitional_probabilities();        //Calculate a=alpha, b=beta
get_marginal_probabilities();            //Calculate p(i,j)
get_buffer_occupancy();                  //Calculate A(i,j)
while (k < k_max)
{
   for (int i=0;i<=M1;i++)
        for (int j=0;j<=M2;j++)
        {
                if (i = 0)
                  y(i,j,k+1) = b(1,i+1,j) y(i+1,j,k);
                elseif (i = M1)
                  y(i,j,k+1) = a(1,i-1,j) y(i-1,j,k);
                else
                  y(i,j,k+1) = a(1,i-1,j) y(i-1,j,k+1) + b(1,i+1,j) y(i+1,j,k);

                if (j = 0)
                  y(i,j,k+1) =+ b(2,i,j+1) y(i,j+1,k);
                 elseif (j = M1)
                  y(i,j,k+1) =+ a(2,i,j-1) y(i,j-1,k+1);
                else
                  y(i,j,k+1) =+ a(2,i,j-1) y(i,j-1,k+1) + b(2,i,j+1) y(i,j+1,k);

                // Normalize y(i,j,k+1)
                y(i,j,k+1) = p(i,j) y(i,j,k+1) / sum(y(i,j,k+1));
        }

    p_current = calculate_cell_loss();
    if fabs(p_current-p_previous)/p_current > delta return p_current;
    k = k + 1;
    p_previous = p_current;
}
```

Figure 4-2: Algorithm to Calculate the Cell Loss Probability.

4.3.2 Implementation

Our main process - the Network Analysis Process (NAP) - is a state-machine that obtains the required parameters for several ATM networks from an input file. These inputs specify the traffic and ATM multiplexer parameters for the ATM network to be analyzed. Based on the algorithm presented earlier, NAP estimates the cell loss probability for a defined ATM network, then it stores the results in an output file.

Afterwards, the main process reads the parameters of the next network to be analyzed, and the process it repeated until all the networks in the input file have been analyzed with their results stored in the output file as shown in Figure 4-3.

Figure 4-3: Network Analysis Process (NAP).

We implemented NAP using C++ as shown in APPENDIX A: MAIN.CXX. The supporting matrix operations required are implemented using a matrix class. The class specification is provided in APPENDIX B: MATRIX.H, and the implementation is included in APPENDIX C: MATRIX.CXX. We selected C++ due to its object-oriented nature, speed, precision, and portability. Moreover, the matrix class allowed us to simplify the implementation of all the complex matrix operations required. This implementation was tested on several platforms: UNIX, MS Windows 95 and MS Windows NT. The results obtained for the different platforms were identical. We analyzed several types of ATM networks with various flavors of traffic classes. These results of the analysis are presented in the next chapter and compared to the simulation results.

One of the ATM networks analyzed in the next chapter is composed of 10 Fast Ethernet workstations, 50 Ethernet workstations, a multiplexer with output rate of an OC-12c/STS-12c (622Mbps) and a buffer of size 50 cells. Figure 4-4 shows the required input parameters sources for class 1 (Fast Ethernet) and class 2 (Ethernet): peak rate, mean rate, cell burst length and number of class sources. It also describes buffer size and the link speed of the ATM multiplexer. Moreover, any of the above parameters can be varied with a constant value. In this network, we decided to vary the number of Fast Ethernet workstations from 10 to 60. Afterwards, NAP will analyze the new network and calculate the cell loss probability and store the results in the output file. Figure 4-5 shows

the screen output for the networks described, and Figure 4-6 provides the output file which can be used to graph the cell loss probability for the networks analyzed.

```
#-------------------------------------------------;
#class1=high-speed data and class2=low-speed data;
#-------------------------------------------------;
#peak1,peak2,mean1,mean2,L1,L2,M1,M2,K,C;
#col 0,1    ,2    ,3    ,4 ,5 ,6 ,7,8,9;
100,10,10,1,3390,339,10,50,50,622;
#-------------------------------------------------;
#Calculate CLP for M1 10 to 60;
#col, end val, step val;
%6,60,10;
```

Figure 4-4: NAP Input File MMDP.ATM .

```
************************************************************
      Calculating Cell Loss Probability for an ATM Mux
************************************************************
<> Software version 0.07
<> Results are stored in result.txt file.
------------------------------------------------------------
*> M1=10 M2=50 K=50 C=622
*> Peak1=100 Peak2=10 Mean1=10 Mean2=1 Burst1=3390 Burst2=339
=> Cell loss Prob. (1) = 3.42097e-005
------------------------------------------------------------
*> M1=20 M2=50 K=50 C=622
*> Peak1=100 Peak2=10 Mean1=10 Mean2=1 Burst1=3390 Burst2=339
=> Cell loss Prob. (1) = 0.00242185
------------------------------------------------------------
*> M1=30 M2=50 K=50 C=622
*> Peak1=100 Peak2=10 Mean1=10 Mean2=1 Burst1=3390 Burst2=339
=> Cell loss Prob. (2) = 0.0162423
------------------------------------------------------------
*> M1=40 M2=50 K=50 C=622
*> Peak1=100 Peak2=10 Mean1=10 Mean2=1 Burst1=3390 Burst2=339
=> Cell loss Prob. (2) = 0.0493393
------------------------------------------------------------
*> M1=50 M2=50 K=50 C=622
*> Peak1=100 Peak2=10 Mean1=10 Mean2=1 Burst1=3390 Burst2=339
=> Cell loss Prob. (2) = 0.101127
------------------------------------------------------------
************************************************************
              Program Execution completed
************************************************************
```

Figure 4-5: NAP Screen Output.

M1	M2	K	C	Peak1	Peak2	Mean1	Mean2	Burst1	Burst2	CLP
10	50	50	622	100	10	10	1	3390	339	3.42097e-005
20	50	50	622	100	10	10	1	3390	339	0.00242185
30	50	50	622	100	10	10	1	3390	339	0.0162423
40	50	50	622	100	10	10	1	3390	339	0.0493393
50	50	50	622	100	10	10	1	3390	339	0.101127

Figure 4-6: NAP Output File RESULT.TXT.

4.4 Conclusion

In this chapter, we approximated the arrival process with a finite birth-death process. Then, we derived the transitional and marginal probabilities that we used in the cell loss probability formula. Also, a linear transformation was introduced to simplify the calculation of cell loss probability. Then, an algorithm was presented and implemented using C++ to estimate the cell loss probability for several ATM networks. In the next chapter, we will present our simulation model, and we will provide and compare the analysis and simulation results for various types of traffic.

5. SIMULATION & RESULTS

5.1 Introduction

In this chapter, we consider several types of traffic in the ATM network. Also, we present our design methodology and implementation used to simulate various ATM networks with heterogeneous traffic sources. Then, we provide the performance analysis results for the algorithm implemented in the previous chapter for the same ATM networks. Finally, we compare the analysis and simulation results for each of the ATM networks considered.

5.2 ATM Network Simulation

We simulate an ATM network with two classes of traffic: class 1 with M_1 identical sources and class 2 with M_2 identical sources. All sources are either on or off. When a source is on, it generates cells at a constant rate of Δ_1 and Δ_2 cells/second for class 1 and class 2, respectively. No cells are generated when a source is off. Using the traffic mean rate Φ_l and burst length L_l characteristics, we determine the mean on-time of a source as $1/\mu_l$, and the mean off-time of a source as $1/\lambda_l$, where l indicates class 1 or 2. Based on these mean times, we calculate the probability that a source of class l is on, as shown in Figure 5-1:

$$\Pr\{\text{a source of class } l \text{ is on}\} = \frac{1/\mu_l}{1/\mu_l + 1/\lambda_l} \qquad (5\text{-}1)$$

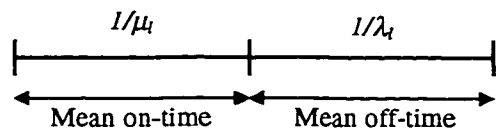

Figure 5-1: Mean On/Off Times of a Source of Class *l*.

Each period consists of MAX_SLOT slots. During each slot, the state of a source is determined using a random number generator and the on-time probability presented above. The cell loss probability is calculated at the end of each period. We stop the simulation when the cell loss probability calculated during a period is within ϵ% of the average cell loss over all pervious periods.

5.3 Traffic Parameters

In our research, we consider four types of ATM network traffic: voice, Low-Speed Data (LSD), High-Speed Data (HSD) and image retrieval. Note that more traffic types can be introduced; however, we need to find their respective On-Off source model parameters. The parameters for the four types of traffic used here are shown in Table 5-1 [31].

Throughout the analysis and the simulation, we assume that the cell size is 53 bytes, i.e. we consider the information is in the payload and the header, as opposed to considering only the information in the payload, which is a more realistic case. We have adopted this assumption to simplify the analysis and the simulation.

Table 5-1: Various Traffic Parameters.

Parameter	*Units*	*Voice*	*LSD*	*HSD*	*Image*
Peak-Rate (Δ)	Mbits/sec	0.064	10	100	2
Mean-Rate (Φ)	Mbits/sec	0.022	1	10	0.087
Burst Length (L)	Cells	58	339	3390	2064

5.4 Implementation

The main process - Network Simulation Process (NSP) - is a state-machine that obtains the required parameters for several ATM networks from an input file. This input file is the same file used with the analysis program, which specifies traffic and ATM

multiplexer parameters for network simulation. Then, the NSP begins the simulation by starting M_1 traffic generators (sources) from class 1 and M_2 traffic generators (sources) from class 2 for a period of `MAX_SLOT` slots. During each slot, any of the traffic generators is either on or off. When a source of class l is on, cells are generated at a peak rate Δ_l, where l is class 1 or 2. Then, the NSP processes the cells if the server is idle; otherwise, the cells are stored in the buffer. When the buffer is full, all incoming cells are lost. The number of lost cells is accumulated in a variable called `lost_cells`. Similarly, the number of all incoming cells is stored in `total_cells` variable. At the end of a period, the cell loss probability is calculated as follows:

```
cell_loss_probability = lost_cells / total_cells
```

Finally, the NSP repeats this process until the cell loss probability converges as specified earlier. Afterwards, the NSP stores the result in an output file. Then, the NSP reads the parameters of the next network to be simulated, and the process it repeated until all the networks in the input file have been simulated with their results stored in the output file, as shown in Figure 5-2.

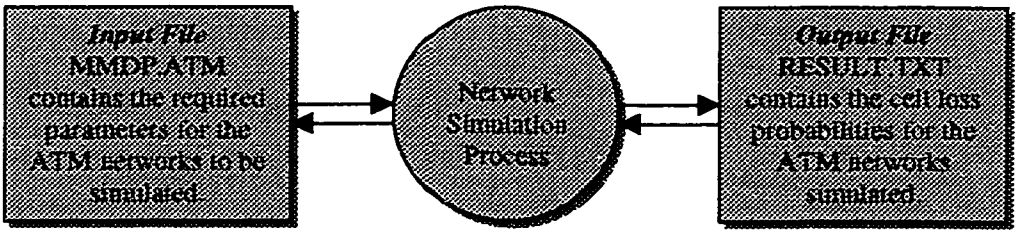

Figure 5-2: Network Simulation Process (NSP).

Let us consider the same ATM network analyzed in the previous chapter. The network is composed of Fast Ethernet workstations (10 to 60), 50 Ethernet workstations, a multiplexer with output rate of an OC-12c/STS-12c (622Mbps) and a buffer of size 50 cells. Figure 5-3 shows the required input parameters, this file is the same exact file used with the Network Analysis Process (NAP). NSP will analyze the networks in the input file and calculate their respective cell loss probabilities, the screen output is provides in Figure 5-4. Then, it will store the results in the output file that is shown in Figure 5-5.

```
#-------------------------------------------------;
#class1=high-speed data and class2=low-speed data;
#-------------------------------------------------;
#peak1,peak2,mean1,mean2,L1,L2,M1,M2,K,C;
#col 0,1    ,2    ,3    ,4 ,5 ,6 ,7,8,9;
100,10,10,1,3390,339,10,50,50,622;
#-------------------------------------------------;
#Calculate CLP for M1 10 to 60;
#col, end val, step val;
%6,60,10;
```

Figure 5-3: NSP Input File MMDP.ATM.

```
************************************************************
            Simulating Cell Loss in an ATM Mux
************************************************************
<> Software version 0.04
<> Results are stored in result.txt file.
------------------------------------------------------------
*> M1=10 M2=50 K=50 C=622
*> Peak1=100    Mean1=10        Burst1=3390
*> Peak2=10     Mean2=1         Burst2=339
+> Prob_on1 = 0.1
+> Prob_on2 = 0.1
-> Period       33      Cell loss prob.        4.76882e-005
------------------------------------------------------------
*> M1=20 M2=50 K=50 C=622
*> Peak1=100    Mean1=10        Burst1=3390
*> Peak2=10     Mean2=1         Burst2=339
+> Prob_on1 = 0.1
+> Prob_on2 = 0.1
-> Period       9       Cell loss prob.        0.00216308
------------------------------------------------------------
*> M1=30 M2=50 K=50 C=622
*> Peak1=100    Mean1=10        Burst1=3390
*> Peak2=10     Mean2=1         Burst2=339
+> Prob_on1 = 0.1
+> Prob_on2 = 0.1
-> Period       6       Cell loss prob.        0.017513
------------------------------------------------------------
*> M1=40 M2=50 K=50 C=622
*> Peak1=100    Mean1=10        Burst1=3390
*> Peak2=10     Mean2=1         Burst2=339
+> Prob_on1 = 0.1
+> Prob_on2 = 0.1
-> Period       1       Cell loss prob.        0.0458761
------------------------------------------------------------
*> M1=50 M2=50 K=50 C=622
*> Peak1=100    Mean1=10        Burst1=3390
*> Peak2=10     Mean2=1         Burst2=339
+> Prob_on1 = 0.1
+> Prob_on2 = 0.1
-> Period       1       Cell loss prob.        0.101089
************************************************************
                      Simulation Completed
************************************************************
```

Figure 5-4: NSP Screen Output.

M1	M2	K	C	Peak1	Peak2	Mean1	Mean2	Burst1	Burst2	CLP Sim.
10	50	50	622	100	10	10	1	3390	339	4.76882e-005
20	50	50	622	100	10	10	1	3390	339	0.00216308
30	50	50	622	100	10	10	1	3390	339	0.017513
40	50	50	622	100	10	10	1	3390	339	0.0458761
50	50	50	622	100	10	10	1	3390	339	0.101089

Figure 5-5: NSP Output File RESULT.TXT.

5.5 Results

In our research, we analyzed and simulated several ATM networks with many flavors. First, we performed various combinations of traffic classes in the network: voice with LSD, HSD with LSD and image retrieval with LSD. We have kept the LSD as the common factor amongst our studies, since Ethernet traffic is a major component of today's networks especially with the exponential growth of the Internet. Second, we varied the number of sources of one class at a time in the network, and we compared the cell loss probabilities obtained from the analysis and the simulation results. Finally, we simulated and analyzed the same networks with the buffer size changing, while maintaining the rest of the network parameter fixed. As a result, we will see the effect of the buffer size on the cell loss probability and how well our approximation is at different congestion periods.

5.5.1 Voice and LSD

We analyzed an ATM network with voice and LSD traffic as shown in Figure 5-6 and Figure 5-7. Based on these figures, we observe the accuracy of the simulation and analysis results. The network considered initially contains 100 voice sources and 40 LSD sources (such as a computer or a workstation on an Ethernet LAN). Also, the incoming traffic is fed into an ATM MUX that has a buffer size of 50 and a service rate of 155 Mbps (STS/OC-3 link). As shown in Figure 5-6, we increase the number of voice sources from 100 to 300 while maintaining the number of LSD sources constant at 40. We notice that the CLP slightly increases as we add more voice sources to the network. However in Figure 5-7, we increase the number of LSD sources from 40 to 110 while maintaining the number of voice sources constant at 100. We notice that the CLP increases dramatically as the number of LSD sources increases.

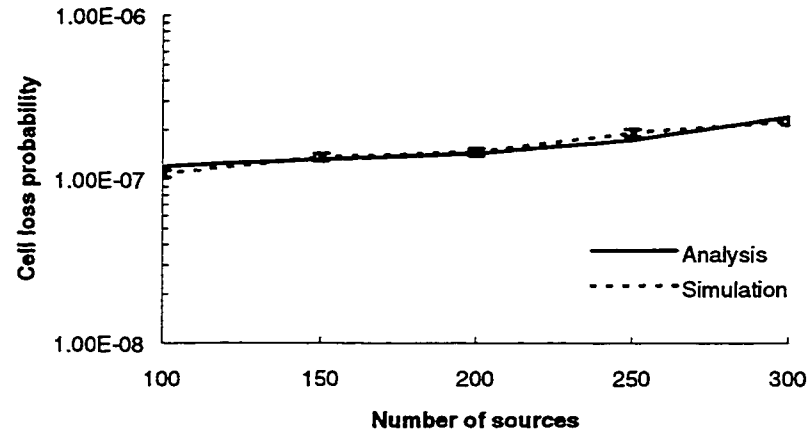

Figure 5-6: Voice (100-300) vs. LSD (40) {C = 155 Mbps, K = 50}.

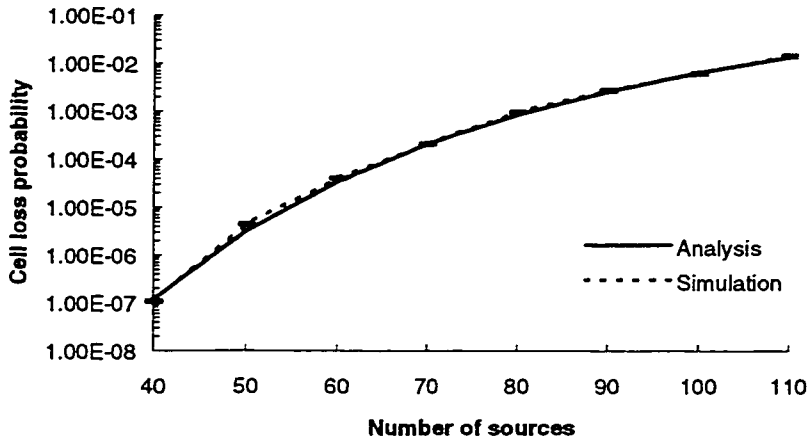

Figure 5-7: Voice (100) vs. LSD (40-110) {C = 155 Mbps, K = 50}.

In Figure 5-8, we plot the cell loss probability for an ATM network with 100-300 voice sources and 50 LSD sources. Then, we plot the cell loss probability for the same network, but this time we vary the LSD sources from 40 to 240 and maintain 100 voice sources. As a result, we observe that in a voice and LSD network, the LSD traffic is the dominant class in the network; therefore, any new connections from the dominant class will significantly affect the CLP.

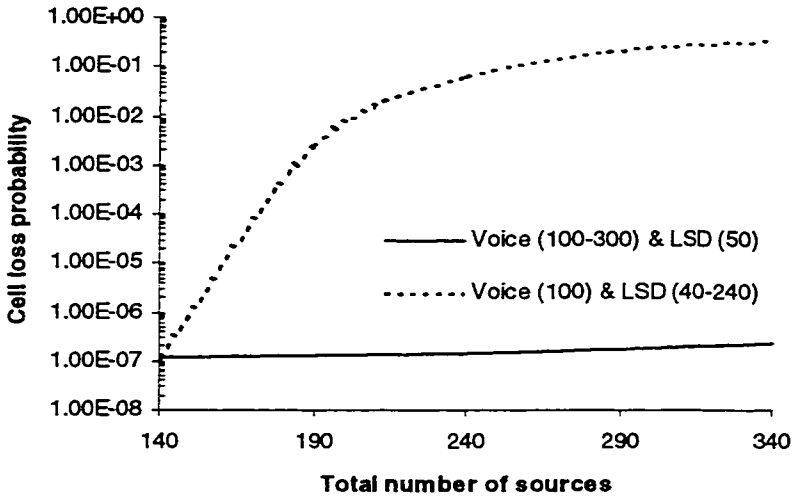

Figure 5-8: Voice vs. LSD (C = 155 Mbps, K = 50.

5.5.2 HSD and LSD

In Figure 5-9 and Figure 5-10, we consider a network that consists of 10 HSD sources (such as a workstation or a PC on a Fast Ethernet LAN) and 50 LSD sources. The network also has an ATM MUX that has a buffer size of 50 and a service rate of 622 Mbps (STS/OC-12 link). In Figure 5-9, we increase the number of HSD sources from 10 to 50 while maintaining the number of LSD sources constant at 50. As a result, the CLP increases from 10^{-5} to 10^{-1}. In Figure 5-10, we increase the number of LSD sources from 50 to 250 while maintaining the number of HSD sources constant at 10. In this case, the CLP increases from 10^{-5} to 10^{-3}. Based on these figures, we notice the accuracy of our analysis and simulation.

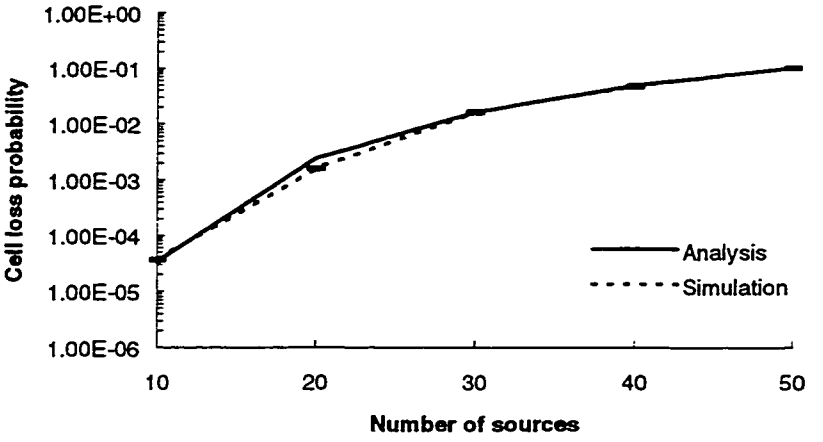

Figure 5-9: HSD (10-50) vs. LSD (50) {C = 622 Mbps, K = 50}.

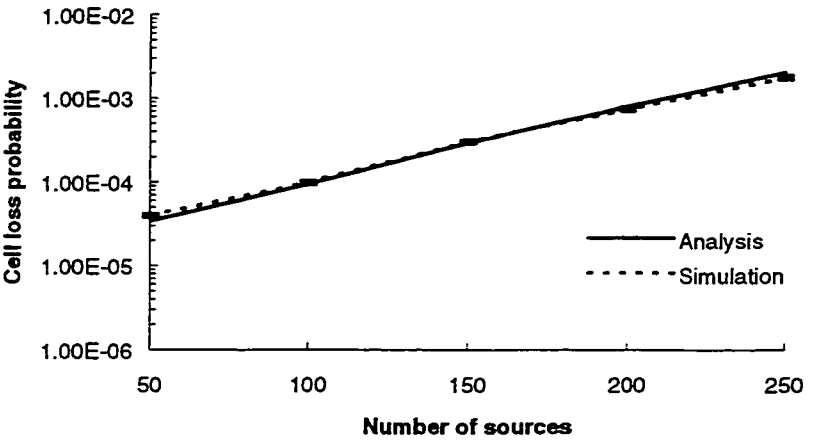

Figure 5-10: HSD (10) vs. LSD (50-250) {C = 622 Mbps, K = 50}.

Figure 5-11 compares the cell loss probability for an ATM network with 10-110 HSD sources and 50 LSD sources, to the cell loss probability for the same network with increasing LDS sources from 50 to 150 and maintaining 10 HSD sources. Consequently, we observe that the HSD dominates LSD traffic in this network. Thus, any new HSD sources added in this network would greatly impact the network CLP.

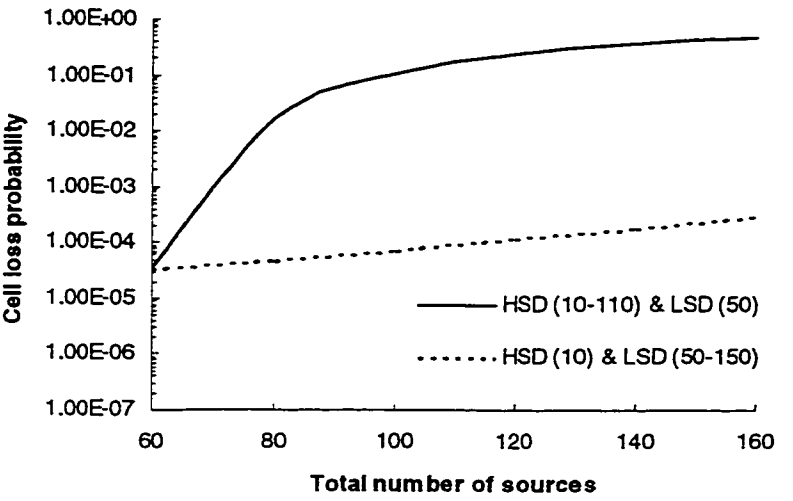

Figure 5-11: HSD vs. LSD {C = 622 Mbps, K = 50}.

5.5.3 Image Retrieval and LSD

We also analyzed a network of image retrieval and LSD sources, as shown in Figure 5-12 and Figure 5-13. Initially, our network contains 10 image-retrieval sources and 50 LSD sources with an ATM MUX that has a buffer size of 50 and a service rate of 155 Mbps (STS/OC-3 link). In Figure 5-12, we increase the number of image-retrieval sources from 10 to 190 while maintaining the number of LSD sources constant at 50. As shown, adding image sources in the network increases the CLP from 10^{-5} to 10^{-4}. In Figure 5-13, we increase the number of LSD sources from 50 to 120 while maintaining the number of image-retrieval sources constant at 100. In this case, the CLP increases tremendously from 10^{-5} to 10^{-1}. Moreover, the simulation and analysis results are accurate.

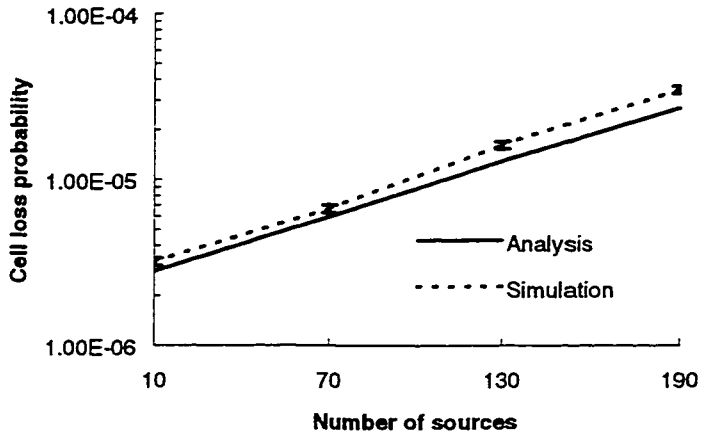

Figure 5-12: Image Retrieval (10-190) vs. LSD (50) {C = 155 Mbps, K = 50}.

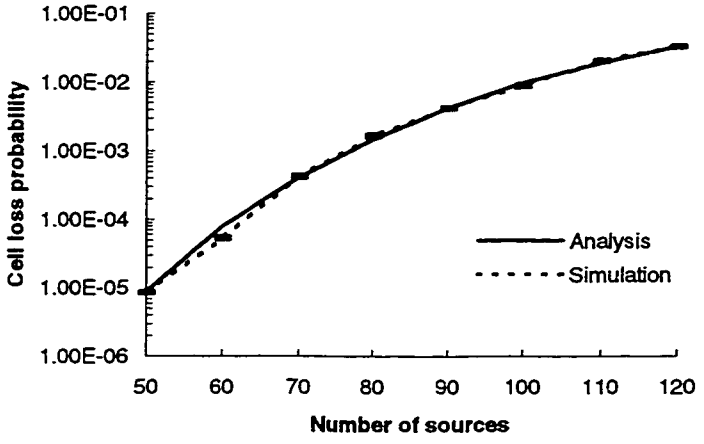

Figure 5-13: Image Retrieval (100) vs. LSD (50-120) {C = 155 Mbps, K = 50}.

In Figure 5-14, the cell loss probability is compared for an ATM network with 90-140 image retrieval sources and 50 LSD sources to the same network but with 40-90 LDS sources and 100 image retrieval sources. As observed in the previous networks considered, data is the dominant class in the network; therefore, accepting new LSD connections will tremendously impact the CLP.

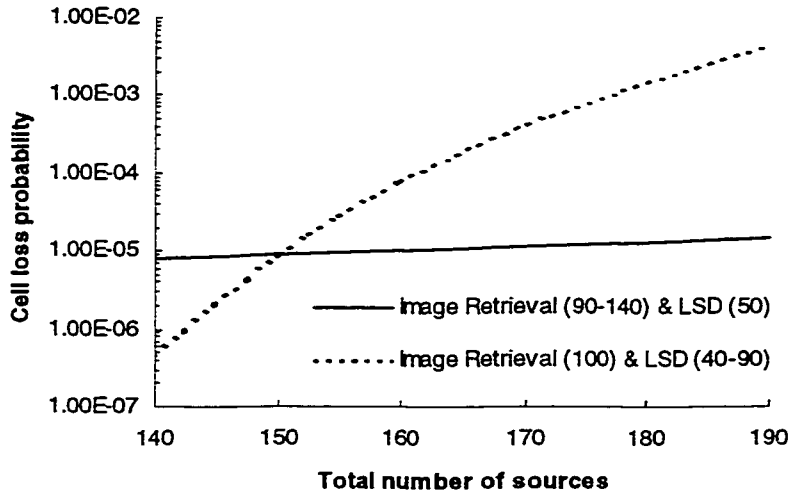

Figure 5-14: Image Retrieval vs. LSD {C = 155 Mbps, K = 50}.

5.5.4 Buffer Size and Cell Loss Probability

In Figure 5-15, we analyze and simulate an ATM network of 50 LSD workstations and 100 voice sources. The ATM multiplexer has a service rate of 155Mbps (STS/OC-3c link). With the buffer size changing from 1 to 71, we plot the cell loss probability for the simulation and the analysis. Similarly, Figure 5-16 shows another network with 15 image retrieval sources and 5 HSD workstations. The multiplexer has an OC12c output link (622Mbps). We also vary the buffer size from 1 to 71, and we graph the cell loss probability for the analysis and the simulation program.

In both figures, the simulation follows the graph shown in Figure 1-5. Thus, we clearly see the effect of cell level and burst level congestion. Therefore, our model does not approximate the cell loss probability well for small buffer sizes where the cell level congestion is dominant. However, our model performs consistently well for large buffer sizes where the burst level congestion dominates.

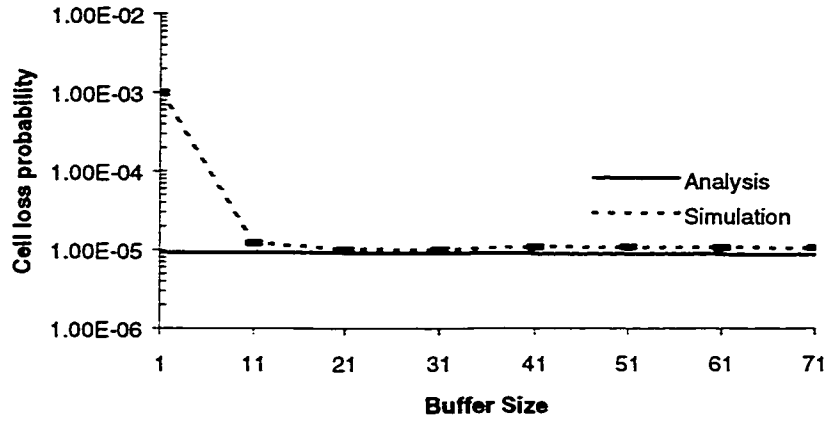

Figure 5-15: Voice (100) and LSD (50) {C=155Mbps}.

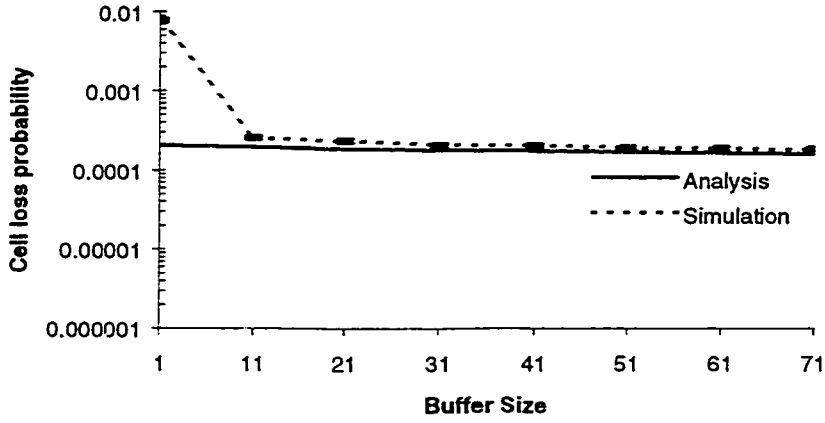

Figure 5-16: Image Retrieval (15) and HSD (5) {C=622Mbps}.

5.6 Conclusion

In all cases, we observe that the cell loss probability (CLP) increases as the number of sources increases irrespective of the source class, as shown in the analysis and the simulation results. Furthermore, the analysis and simulation graphs indicate the accuracy of our results. After examining these figures, we notice that a dominant class is present (the data traffic sources) in the heterogeneous ATM network, and it significantly affects the CLP when extra connections are accepted from this class. Therefore, when

accepting new connections from the dominant class, the CLP increases significantly, thus affecting the Quality of Service (QoS) in the network. Furthermore, we confirmed the relationship between the buffer size and the cell loss probability through our simulation. Also, we demonstrated that our model performs consistently well for large buffer sizes where the burst level congestion is dominant. Thus, this model can be used to prevent burst level congestion from ATM heterogeneous networks by estimating the CLP of adding new connections from the various traffic classes in the network.

6. CONCLUSION

6.1 Thesis Summary

In chapter 1, we presented a brief overview of ATM and its synergy with SONET in the B-ISDN model. Then, we provided the four ATM traffic classes and their relationship with respect to timing, bit rate and connection mode. ATM source levels were discussed, and their effect on traffic congestion and cell loss probability was given. Afterwards, we discussed traffic modeling and presented the model used in this thesis. Finally, the various models adopted by different researchers were compared, and the queueing model used here was presented.

Chapter 2 discussed the ATM standard. The ATM cell structure and the value of each field of the ATM cell were given. The B-ISDN/ATM reference model and the functionality of each layer were provided. Traffic attributes and different techniques in traffic control and congestion feedback were compared. Lastly, ATM signaling and the end-to-end ATM network were presented.

We presented the queueing system in chapter 3. We started with the main statistical features of traffic sources. Then, we provided numerous traffic source models and their application to different types of ATM traffic. We also explained the selected source model as well as the traffic parameters required to approximate the arrival process. Afterwards, we defined the queueing system and derived a new method to calculate the cell loss probability in an ATM multiplexer with heterogeneous traffic sources.

In chapter 4, we presented a technique to estimate the cell loss probability from the formula derived earlier. We provided an algorithm to compute the cell loss probability for a heterogeneous ATM network. Then, we discussed and demonstrated the design methodology used to implement the algorithm.

Chapter 5 contained the numerical results and simulation. Four types of traffic in ATM networks were provided along with their traffic parameters. These four classes are low-speed data such as Ethernet, high-speed data like Fast Ethernet, voice and image retrieval. We explained the design methodology and implementation used to simulate the

ATM multiplexer. Then, we provided the performance analysis results for the same networks considered in the simulation. Finally, we compared both the simulation and analysis results.

6.2 Conclusions

This thesis addressed a new technique in estimating the cell loss probability in an ATM multiplexer with heterogeneous network. The actual arrival process is approximated using a Markov Modulated Deterministic Process (MMDP). Furthermore, the ATM multiplexer has a single server with a rate of C cells/second and a finite buffer with K cells. The multiplexer is considered to serve a superposition of several classes of independent On-Off sources, where sources within a class are identical. Therefore, the multiplexer is modeled as an MMDP/D/1/K queueing system.

After carrying out the analysis for the queueing system, the cell loss probability formula for a two-class ATM network was derived. Afterwards, we estimated the cell loss probability and provided an algorithm to compute it. We considered four different classes, namely: voice, low-speed data, high-speed data and images retrieval. By considering two classes at a time, we calculated the cell loss probability based on the analysis derived. Then, we compared the analysis results to simulation results for the same network, this proved the accuracy of the analysis. In all cases, we observe that the cell loss probability (CLP) increases as the number of sources increases irrespective of the source class. After examining these results, we observed that a dominant class (data) is present in the heterogeneous ATM network, and it significantly affects the CLP when extra connections are accepted from this class. Therefore, when accepting new connections from a dominant class, the CLP increases significantly, thus affecting the Quality of Service (QoS) in the network. Moreover, we demonstrated that our model captures the burst level congestion in an ATM network, as it performs consistently well for large buffer sizes. Therefore, this model is well suited for preventing burst level congestion from ATM heterogeneous networks.

6.3 Future Research

Future research can be conducted in several directions. First, a closed form solution for the two-class ATM network can be derived which is analytically easier to calculate, but less accurate in estimating the CLP. Another future research area can be the extension of the heterogeneous network analysis derived in this paper for a two-class ATM network to an m-class network. However, increasing the number of classes results in exponentially increasing the problem size, which makes it computationally heavy and unstable.

APPENDIX A: MAIN.CXX

```
/*
============================================================
File:           main.cxx
                main.cxx implements the algorithm for
                        calculating the cell loss probability
                        for a two-class-traffic ATM MUX, based
                        on the model of MMDP.
                        Mi sources of class i, i=1,2
                        K  MUX queue sixe
                        C  service rate
                        peak and mean rates
                        cell burst length
Dependencies:   math.h, fstream.h, matrix.h
Author:         Charlie B. Kawwas
Version:
                0.07    1998/01/07      CK
                - add fabs in the cell_loss_algorithm function
                  to take the floating point absolute value.
                0.06    1997/02/26      CK
                - fix calculate_lambda with the right
                  formula = mu / (1-burstiness).
                0.05    1997/02/24      CK
                - fix calculate_current_y for M1=0
                  and/or M2=0.
                0.04    1996/12/22      CK
                - implement output results to result.txt
                - correct the calculation of the cell_loss_
                  const.
                0.03    1996/12/14      CK
                - Implement in main() to be able to read
                  from "mmdp.atm" file the parameters
                  required using fstream.h.
                0.02    1996/12/14      CK
                - Commented debugging info.
                - cCreate cell_loss_algorithm procedure
                  from main().
                - Introduced i_count to track the number
                  needed to coverge for the cell loss
                  prob. in cell_loss_algorithm().
                - Defined a temp_matrix in calculate_
                  state_prob() to resize state_prob
                  accordingly.
                - Defined a temp_matrix in cond_prob()
                  to resize A matrix when queue size
                  changes.
                0.01    1996/12/14      CK
                - First release
============================================================
*/

#include <math.h>
#include <fstream.h>
#include "matrix.h"

// functions prototypes

void calculate_mu();
void calculate_lambda();
void calculate_state_prob();

void initialize_y();
void calculate_current_y();

double gamma(int i, int j);
double alpha(int iclass,int i,int j);
double beta(int iclass ,int i,int j);

double arrival_rate(int i, int j);
Cmatrix& cond_prob(int i, int j);
```

```
double cell_loss_const();
double calculate_cell_loss();
double cell_loss_algorithm();

void get_input_params();

// global constants and variables

const double d_VERSION = 0.07;                //Software version

const int i_TRAFFIC_CLASSES = 2;              //Max 2 traffic classes are used

int i_max_classA_sources;
int i_max_classB_sources;
int i_max_queue_size;
double d_service_rate;                        //Mbps

Cmatrix  state_prob;
Cmatrix  mu(i_TRAFFIC_CLASSES);
Cmatrix  lambda(i_TRAFFIC_CLASSES);

Cmatrix  peak_rate(i_TRAFFIC_CLASSES);       //Mbps
Cmatrix  mean_rate(i_TRAFFIC_CLASSES);       //Mbps
Cmatrix  burst_length(i_TRAFFIC_CLASSES);    //Cells

// A pointer to a matrix - used for a matrix of vectors
Cmatrix * y_previous;
Cmatrix * y_current;

//Input parameters array
int const MAX_IN=10;
int const MAX_LOOP=3;
double in_params[MAX_IN];
double in_loop[MAX_LOOP];

// ============================================================================
// MAIN PROGRAM
// ============================================================================

void main()
{

const int PEAK1=0, PEAK2=1, MEAN1=2,MEAN2=3, CELL1=4,CELL2=5;
const int SOURCES1=6, SOURCES2=7, BUFFER_SIZE=8, SERVICE_RATE=9;
const int COL_NUM=0, END_VALUE=1, STEP_VALUE=2;

get_input_params();

cout << '*****************************************************************'<<endl;
cout << '      Calculating Cell Loss Probability for an ATM Mux' << endl;
cout << '*****************************************************************'<<endl;

cout << '<> Software version ' << d_VERSION << endl;

ofstream f_out('result.txt');
if (!f_out)
        cout << '<> Error: unable to open result.txt for output!'<< endl;
else
        cout << '<> Results are stored in result.txt file.'<< endl;

cout << '-----------------------------------------------------------------'<<endl;

f_out << 'M1\tM2\tK\tC\t';
f_out << 'Peak1\tPeak2\tMean1\tMean2\tBurst1\tBurst2\t';
f_out << 'Cell loss Probability' <<endl;

int start_val = in_loop[COL_NUM];
for (double i=in_params[start_val];i<in_loop[END_VALUE];
                i=i+in_loop[STEP_VALUE])
        {
        in_params[start_val]=i;

        // values are in Mbps for rates and cells for lengths
        peak_rate.Set(in_params[PEAK1],0);
        peak_rate.Set(in_params[PEAK2],1);
        mean_rate.Set(in_params[MEAN1],0);
```

```
        mean_rate.Set(in_params[MEAN2],1);
        burst_length.Set(in_params[CELL1],0);
        burst_length.Set(in_params[CELL2],1);

        i_max_classA_sources =in_params[SOURCES1];
        i_max_classB_sources =in_params[SOURCES2];
        i_max_queue_size =in_params[BUFFER_SIZE];
        d_service_rate = in_params[SERVICE_RATE];

        f_out << i_max_classA_sources <<"\t";
        f_out << i_max_classB_sources <<"\t";
        f_out << i_max_queue_size <<"\t";
        f_out << d_service_rate <<"\t";
        f_out << peak_rate.Get(0) <<"\t"<< peak_rate.Get(1) <<"\t";
        f_out << mean_rate.Get(0) <<"\t"<< mean_rate.Get(1) <<"\t";
        f_out << burst_length.Get(0) <<"\t"<< burst_length.Get(1)<<"\t";

        f_out << cell_loss_algorithm() << endl;
        }

f_out.close();

cout << "*****************************************************************"<<endl;
cout << "*                  Program Execution completed"<<endl;
cout << "*****************************************************************"<<endl;

}

// ============================================================================
// LOCAL ROUTINES
// ============================================================================

// ----------------------------------------------------------------------------
void calculate_mu()
{
for (int i=0;i<i_TRAFFIC_CLASSES;i++)
        mu.Set((1000000.0 * peak_rate.Get(i)) / (53 * burst_length.Get(i)),i);
}

// ----------------------------------------------------------------------------
void calculate_lambda()
{
for (int i=0;i<i_TRAFFIC_CLASSES;i++)
        lambda.Set(((1000000.0 * peak_rate.Get(i)) / (53 * burst_length.Get(i)))/
                                ((peak_rate.Get(i) / mean_rate.Get(i)) - 1),i);
}

// ----------------------------------------------------------------------------
double gamma(int i, int j)
{
        return ((i_max_classA_sources - i)* lambda.Get(0) + i* mu.Get(0) +
                        (i_max_classB_sources - j)* lambda.Get(1) + j* mu.Get(1));
}

// ----------------------------------------------------------------------------
double alpha(int iclass, int i, int j)
{
        if ((i<0)||(j<0)||(i>i_max_classA_sources)||(j>i_max_classB_sources))
                return 0;
        else if (iclass == 1)
                return (i_max_classA_sources - i) * lambda.Get(0) /
                                gamma(i,j);
        else
                return (i_max_classB_sources - j) * lambda.Get(1) /
                                gamma(i,j);
}

// ----------------------------------------------------------------------------
double beta(int iclass, int i, int j)
{
        if ((i<0)||(j<0)||(i>i_max_classA_sources)||(j>i_max_classB_sources))
                return 0;
        else if (iclass == 1)
                return i * mu.Get(0) / gamma(i,j);
        else
                return j * mu.Get(1) / gamma(i,j);
```

```
}

// ------------------------------------------------------------------------
void calculate_state_prob()
{

        Cmatrix  temp_matrix(i_max_classA_sources+1,i_max_classB_sources+1);
        state_prob = temp_matrix;

        // calculate P(0,0) =
        //    inverse( 1 + sum((i=0 to M1)(j=1 to M2)) + sum(i=1 to M1)(j=0))

        double initial_prob_1=0;
        for (int i=0;i<=i_max_classA_sources;i++)
                for (int j=1;j<=i_max_classB_sources;j++)
                {
                        double init_value=1;
                        for (int jj=j-1;jj>=0;jj--)
                                init_value *= (alpha(2,i,jj)/beta(2,i,jj+1));
                        for (int ii=i-1;ii>=0;ii--)
                                init_value *= (alpha(1,ii,0)/beta(1,ii+1,0));
                        //cout << "(" <<i <<"," << j<<")=" << init_value <<endl;
                        initial_prob_1+= init_value;
                }

        double initial_prob_2=0;
        for (int ij=1;ij<=i_max_classA_sources;ij++)
                {
                        double init_value=1;
                        for (int ii=ij-1;ii>=0;ii--)
                                init_value *= (alpha(1,ii,0)/beta(1,ii+1,0));
                        //cout << "(" <<ij <<",0)=" << init_value <<endl;
                        initial_prob_2 += init_value;
                }


        state_prob.Set(1/(1+initial_prob_1+initial_prob_2),0,0);

        // calculate P(r,c) = P(0,0) * (alphas)/(betas)
        double d_total_prob=state_prob.Get(0,0);
        for (int r=0;r<=i_max_classA_sources;r++)
                for (int c=0;c<=i_max_classB_sources;c++)
                        if (!((r == 0)&& (c == 0)))
                        {
                                double init_value=state_prob.Get(0,0);
                                for (int jj=c-1;jj>=0;jj--)
                                        init_value *= (alpha(2,r,jj)/beta(2,r,jj+1));
                                for (int ii=r-1;ii>=0;ii--)
                                        init_value *= (alpha(1,ii,0)/beta(1,ii+1,0));
                                state_prob.Set(init_value,r, c);
                                d_total_prob+=init_value;
                        }
        if ((d_total_prob >= 1.00000001) || (d_total_prob <= 0.999999999))
                cout << ":> Calculating state probabilities ... Total Prob. = "
                     << d_total_prob << endl;
}


// ------------------------------------------------------------------------
double arrival_rate(int i, int j)
{
        return (i * peak_rate.Get(0) + j * peak_rate.Get(1));
}

// ------------------------------------------------------------------------
Cmatrix& cond_prob(int i, int j)
{
        // A is static and is needed for Cmatrix& <-- ref
        // and concatenation.
        static Cmatrix A;
        Cmatrix temp_matrix(i_max_queue_size+1,i_max_queue_size+1);
        A = temp_matrix;

        double d_arrival_rate = arrival_rate(i,j);
        double d_gamma = gamma(i,j);
        double d_ro;
```

```
        if (d_arrival_rate == d_service_rate)
        {
                d_ro = exp(-1*d_gamma/(1000000.0*d_service_rate));
                for(int k=0;k<=i_max_queue_size;k++)
                        for (int h=0;h<=i_max_queue_size;h++)
                        {
                                if ((k == h) && (h > 0))
                                        A.Set(1,k,h);
                                else
                                        A.Set(0,k,h);
                        }
                A.Set(1-d_ro ,0,0);
                A.Set(d_ro ,0,1);
        }
        else if (d_arrival_rate < d_service_rate)
        {
                d_ro = exp(-1*d_gamma/(1000000.0*(d_service_rate-d_arrival_rate)));
                for(int k=0;k<=i_max_queue_size;k++)
                        for (int h=0;h<=i_max_queue_size;h++)
                        {
                                if (k < h)
                                        A.Set(0,k,h);
                                else if ((k >= h) && (h >= 1))
                                        A.Set(pow(d_ro,k-h) * (1- d_ro),k,h);
                                else
                                        A.Set(pow(d_ro,k),k,h);
                        }
        }
        else
        {
                d_ro = exp(-1*d_gamma/(1000000.0*(d_arrival_rate-d_service_rate)));
                for(int k=0;k<=i_max_queue_size;k++)
                        for (int h=0;h<=i_max_queue_size;h++)
                        {
                                if (h < k)
                                        A.Set(0,k,h);
                                else if ((k <= h) && (h < i_max_queue_size))
                                        A.Set(pow(d_ro,h-k) * (1- d_ro),k,h);
                                else
                                        A.Set(pow(d_ro,i_max_queue_size-k),k,h);
                        }
        }

/*      cout << "Conditional Probability Transition Matrix "
                 << "for buffer occupancy in a given state " << i << "," << j
                 << " -> " << endl;
        A.Display();
*/
        return A;
}


// ------------------------------------------------------------------------
void initialize_y()
{
        // algorithm steps 1 and 2
        Cmatrix y(1,i_max_queue_size+1);

        //initialize with uniform distribution
        double d_uniform = 1.0/((i_max_classA_sources+1)*(i_max_classB_sources+1)
                                                        *(i_max_queue_size+1));
        for (int k1=0; k1<=i_max_queue_size;k1++)
                y.Set(d_uniform,0,k1);

        //cout << "+> y_initial:" <<endl;
        //y.Display();

        //initialize current and previous y vectors
        for (int ii=0; ii<=i_max_classA_sources;ii++)
                for(int jj=0;jj<=i_max_classB_sources;jj++)
                {
                        y_previous[ii*(i_max_classB_sources+1)+jj] = y;
                        y_current[ii*(i_max_classB_sources+1)+jj] = y;
                }
}
```

```
// ------------------------------------------------------------------------
void calculate_current_y()
{
        for (int i=0; i<=i_max_classA_sources;i++)
                for(int j=0;j<=i_max_classB_sources;j++)
                {
                if ((i == 0) && (i_max_classA_sources > 0 ))
                        y_current[i*(i_max_classB_sources+1)+j] =
                                beta(1,i+1,j)*y_previous[(i+1)*(i_max_classB_sources+1)+j];

                else if ((i == i_max_classA_sources) && (i_max_classA_sources > 0 ))
                        y_current[i*(i_max_classB_sources+1)+j] =
                                alpha(1,i-1,j)*y_current[(i-1)*(i_max_classB_sources+1)+j];

                else if  (i_max_classA_sources > 0 )
                        y_current[i*(i_max_classB_sources+1)+j] =
                                alpha(1,i-1,j)*y_current[(i-1)*(i_max_classB_sources+1)+j] +
                                beta(1,i+1,j)*y_previous[(i+1)*(i_max_classB_sources+1)+j];

                //Debug
                //cout << "Debug ... " << i << ", " << j << endl;


                if ((j == 0) && (i_max_classB_sources > 0 ))
                        y_current[i*(i_max_classB_sources+1)+j] =
                                y_current[i*(i_max_classB_sources+1)+j] +
                                beta(2,i,j+1)*y_previous[i*(i_max_classB_sources+1)+(j+1)];

                else if ((j == i_max_classB_sources) && (i_max_classB_sources > 0 ))
                        y_current[i*(i_max_classB_sources+1)+j] =
                                y_current[i*(i_max_classB_sources+1)+j] +
                                alpha(2,i,j-1)*y_current[i*(i_max_classB_sources+1)+(j-1)];

                else if (i_max_classB_sources > 0 )
                        y_current[i*(i_max_classB_sources+1)+j] =
                                y_current[i*(i_max_classB_sources+1)+j] +
                                alpha(2,i,j-1)*y_current[i*(i_max_classB_sources+1)+(j-1)] +
                                beta(2,i,j+1)*y_previous[i*(i_max_classB_sources+1)+(j+1)];

                //Debug
                //cout << "Debug ... y_current"<<endl;
                //y_current[i*(i_max_classB_sources+1)+j].Display();

                //multiply by A (conditional buffer occupancy prob. matrix)
                y_current[i*(i_max_classB_sources+1)+j] =
                        y_current[i*(i_max_classB_sources+1)+j] * cond_prob(i,j);

                //Debug
                //cout << "Debug ... y_current * A ="<<endl;
                //y_current[i*(i_max_classB_sources+1)+j].Display();

                //normalize
                y_current[i*(i_max_classB_sources+1)+j] =
                        y_current[i*(i_max_classB_sources+1)+j] *
                        (state_prob.Get(i,j) /
y_current[i*(i_max_classB_sources+1)+j].Sum());

                //Debug
                //cout << "-> y_previous("<<i<<","<<j<<") ="<<endl;
                //y_previous[i*(i_max_classB_sources+1)+j].Display();
                //cout << "-> y_current("<<i<<","<<j<<") ="<<endl;
                //y_current[i*(i_max_classB_sources+1)+j].Display();

                }

        //y_previous = y_current
        double d_pre_sum=0;
        double d_curr_sum=0;
        for (int ii=0; ii<=i_max_classA_sources;ii++)
                for(int jj=0;jj<=i_max_classB_sources;jj++)
                {
                        d_pre_sum = d_pre_sum +
y_previous[ii*(i_max_classB_sources+1)+jj].Sum();
                        d_curr_sum = d_curr_sum +
y_current[ii*(i_max_classB_sources+1)+jj].Sum();
```

```
                        y_previous[ii*(i_max_classB_sources+1)+jj] =
                        y_current[ii*(i_max_classB_sources+1)+jj];
                }
        cout << "-> y_previous(sum) ="<<d_pre_sum<<endl;
        cout << "-> y_current(sum)  ="<<d_pre_sum<<endl;

}

// ---------------------------------------------------------------------------
double cell_loss_const()
{
        double temp_value=0;
        for (int i=1;i<=i_max_classA_sources;i++)
                for (int j=0;j<=i_max_classB_sources;j++)
                        temp_value = temp_value + state_prob.Get(i,j) *
                                                        exp(-
1*gamma(i,j)/(1000000.0*arrival_rate(i,j))) /
                                                        (1 - exp(-
1*gamma(i,j)/(1000000.0*arrival_rate(i,j))));
        for (int j=1;j<=i_max_classB_sources;j++)
                temp_value = temp_value + state_prob.Get(0,j) *
                                                exp(-
1*gamma(0,j)/(1000000.0*arrival_rate(0,j))) /
                                                (1 - exp(-
1*gamma(0,j)/(1000000.0*arrival_rate(0,j))));
        //Debug
        cout << "+> Cell loss constant = sum() = " << temp_value << endl;
        return temp_value;
}

// ---------------------------------------------------------------------------
double calculate_cell_loss()
{
        double d_ro_temp;
        double temp_value=0;

        for (int i=0; i<=i_max_classA_sources;i++)
                for(int j=0;j<=i_max_classB_sources;j++)
                        if (arrival_rate(i,j) > d_service_rate)
                        {
                                //d_ro = exp(-1.0*gamma(i,j)/(1000000.0*(arrival_rate(i,j)-
d_service_rate)));
                                d_ro_temp =-1.0*gamma(i,j)/(1000000.0*(arrival_rate(i,j)-
d_service_rate));
                                d_ro_temp = exp(d_ro_temp);
                                temp_value = temp_value +

y_current[i*(i_max_classB_sources+1)+j].Get(0,i_max_queue_size)
                                        *d_ro_temp/(1.0-d_ro_temp);

                                //Debug
                                if ((i > 225)&(j > 49)){
                                        cout << "\tTemp value ("<<i<<","<<j<<") = "<<
temp_value  <<endl;
                                        cout << "-> gamma("<<i<<","<<j<<") = "<< gamma(i,j)
<<endl;
                                        cout << "-> b("<<i<<","<<j<<") - C = "<<
arrival_rate(i,j)-d_service_rate<<endl;
                                        cout << "-> ro("<<i<<","<<j<<") = "<< d_ro_temp
<<endl;
                                        cout << "-> cal ro"<<exp(-
1.0*gamma(i,j)/(1000000.0*(arrival_rate(i,j)-d_service_rate)))  <<endl;
                                        cout << "-> y("<<i<<","<<j<<") = "<<

        y_current[i*(i_max_classB_sources+1)+j].Get(0,i_max_queue_size)<<endl;
                                }
                        }
        //Debug
        cout << "-> sum(ro*y/(1-ro)) = " << temp_value << endl;
        return  temp_value/cell_loss_const();
}

// ---------------------------------------------------------------------------
double cell_loss_algorithm()
{
```

```
cout << "*>";
cout << " M1=" <<i_max_classA_sources;
cout << " M2=" <<i_max_classB_sources;
cout << " K=" <<i_max_queue_size;
cout << " C=" <<d_service_rate <<endl;

cout << "*>";
cout << " Peak1="<<peak_rate.Get(0);
cout << " Peak2="<<peak_rate.Get(1);
cout << " Mean1="<<mean_rate.Get(0);
cout << " Mean2="<<mean_rate.Get(1);
cout << " Burst1="<<burst_length.Get(0);
cout << " Burst2="<<burst_length.Get(1)<<endl;

calculate_mu();
calculate_lambda();
calculate_state_prob();

/*
cout<<"*** mu -> "<< endl;          mu.Display();
cout<<"*** lambda -> "<< endl;      lambda.Display();
cout<<"*** state_prob -> "<< endl; state_prob.Display();
*/

// memory allocation of matrix of vectors
y_previous = new Cmatrix [(i_max_classA_sources+1)*(i_max_classB_sources+1)];
y_current = new Cmatrix [(i_max_classA_sources+1)*(i_max_classB_sources+1)];

double d_cell_loss_prob_previous=0;
double d_cell_loss_prob_current;

initialize_y();

int i_found=0;
int i_count=-1;
const MAX_TRY = 20;

while ((i_found == 0) && (i_count < MAX_TRY))
        {
//  cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"<<endl;
        i_count++;
        calculate_current_y();
        d_cell_loss_prob_current = calculate_cell_loss();
//      cout <<  "=> Cell loss Prob. ("<< i_count << ") = "
//               << d_cell_loss_prob_current << endl;
        if ((fabs(d_cell_loss_prob_current-d_cell_loss_prob_previous)/
              d_cell_loss_prob_current)  < 0.000001) i_found=MAX_TRY;
        else    d_cell_loss_prob_previous = d_cell_loss_prob_current;
        }

cout  <<  "=> Cell loss Prob. ("<< i_count << ") = ";
cout << d_cell_loss_prob_current << endl;


delete [] y_previous;
delete [] y_current;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"<<endl;

return d_cell_loss_prob_current;

}

// ---------------------------------------------------------------------------
void get_input_params()
{
        const int STATE_READ=0;
        const int STATE_COMMENT=1;
        const int STATE_LOOP=2;
        const int STATE_EXIT=10;

        int i_state=STATE_READ;
        char c, c_value[256];
        int count=0, i_count_char=0;

        c_value[0]=0;
```

```
        ifstream f_in("mmdp.atm");
        if (!f_in)
        {
                cout << "Error: unable to open input file" <<endl;
                i_state = STATE_EXIT;
        }
        else
        {
          while (f_in.get(c))
                switch (c)
                {
                case '#':
                        i_state = STATE_COMMENT;
                        break;
                case ';':
                        if (i_state == STATE_READ)
                        {
                                c_value[i_count_char]=0;
                                if (count < MAX_IN)
                                        in_params[count]=atof(c_value);
                                count++;
                                i_count_char = 0;
                        }
                        if (i_state == STATE_LOOP)
                        {
                                c_value[i_count_char]=0;
                                if (count < MAX_LOOP)
                                        in_loop[count]=atof(c_value);
                                count++;
                                i_count_char = 0;
                        }
                        i_state = STATE_READ;
                        count=0;
                        i_count_char=0;
                        break;
                case '%':
                        i_state = STATE_LOOP;
                        break;
                case ',':
                        if (i_state == STATE_READ)
                        {
                                c_value[i_count_char]=0;
                                if (count < MAX_IN)
                                        in_params[count]=atof(c_value);
                                count++;
                                i_count_char = 0;
                        }
                        if (i_state == STATE_LOOP)
                        {
                                c_value[i_count_char]=0;
                                if (count < MAX_LOOP)
                                        in_loop[count]=atof(c_value);
                                count++;
                                i_count_char = 0;
                        }
                        break;
                default:
                        if ((c > 32) && (c<128))
                        {
                                c_value[i_count_char]=c;
                                i_count_char++;
                                c_value[i_count_char]=0;
                        }
                        break;
                }
        f_in.close();
        }
}
```

APPENDIX B: MATRIX.H

```
/*
============================================================
File:           matrix.h
                defines required prototypes for a matrix
                class
Dependencies:   iostream.h
Author:         Charlie B. Kawwas
Version:
                0.01    1996/12/14
                - First release by CK.
============================================================
*/

// Matrix class declaration
#include <iostream.h>
class Cmatrix
{
private:
        int row;
        int col;
        double * matrixP;
public:
        Cmatrix(int irow = 1,int icol = 1);          //Constructor
        virtual ~Cmatrix();                          //Destructor
        void Display(void) const;
        void Set(double d_value, int r= 0, int c = 0);
        double Get(int r = 0 , int c = 0);
        double Sum();

        Cmatrix& operator =(const Cmatrix& A);              //Override the assignment
        Cmatrix& operator +(const Cmatrix& B);
        Cmatrix& operator ^(const int power);               // NOT IMPLEMENTED YET!
        Cmatrix& operator *(const Cmatrix& B);
        Cmatrix& operator*(const double coeff);

        friend Cmatrix& operator *(const double coeff, const Cmatrix& A);
};
```

APPENDIX C: MATRIX.CXX

```
/*
============================================================
File:           matrix.cxx
                implements a matrix class.
Dependencies:   matrix.h
Author:         Charlie B. Kawwas
Version:
                0.01    1996/12/14
                - First release by CK.
============================================================
*/

// Matrix class implementation

#include "matrix.h"

//Constructor
Cmatrix::Cmatrix(int irow,int icol)
{
        row = irow;
        col = icol;
        matrixP = new double [row*col];
        if (matrixP != 0) {
                //Intialize Matrix
                for (int r=0;r<row;r++)
                        for (int c=0;c<col;c++)
                                matrixP[(r*col)+c] = 1.0;
#ifdef DEBUG
                cout << "Allocated memory for matrix: Row = " << row
                          << " and Col = " << col<< endl;
#endif
        }else
                cout << "ERROR! Insufficient memory for matrix: Row = "
                          << row << " and Col = " << col<<endl;
}

//Destructor
Cmatrix::~Cmatrix()
{
        delete [] matrixP;
#ifdef DEBUG
        cout << "Deallocated memory for matrix: Row = " << row
                  << " and Col = " << col<< endl;
#endif
}

//Override the assignment of matrix class
Cmatrix& Cmatrix::operator =(const Cmatrix& A)
{
        // check for A!=A i.e. no self-assignment
        if (this != &A)
        {
#ifdef DEBUG
                cout << "Deallocating " << row << "x" << col;
#endif
                row = A.row;
                col = A.col;
                delete [] matrixP;  // delete memory allocated to current object
                matrixP = new double [row*col];       //assign new dynamic mem.
                //copy matrrix A elements to current object memory
                for (int r=0;r<row;r++)
                        for (int c=0;c<col;c++)
                                matrixP[(r*col)+c] = A.matrixP[(r*col)+c];
#ifdef DEBUG
                cout << ", and allocating " << row << "x" << col << endl;
#endif
        }
        return *this;
}
```

```
// Display matrix
void Cmatrix::Display(void) const
{
        cout << '-------- Matrix Display '<< row << 'x' << col << ' --------\n';
        for (int irow=0;irow<row;irow++)
        {
                cout << 'Row ' << (irow+1) <<': ';
                for (int icol=0;icol<col;icol++)
                        cout << matrixP[(irow*col)+icol] << ' ';
                cout << endl;
        }
}

// Set element value
void Cmatrix::Set(double d_value, int r, int c)
{
        matrixP[(r)*col+ c] = d_value;
}

// Get element value
double Cmatrix::Get(int r, int c)
{
        if ((r<0) || (c<0) || (r>=row) || (c>=col))
                return 0;
        else
                return matrixP[(r)*col+ c];
}

// Sum all elements
double Cmatrix::Sum()
{
        double sum_value=0;
        for (int irow=0;irow<row;irow++)
                for (int icol=0;icol<col;icol++)
                        sum_value =sum_value + matrixP[(irow)*col+ icol];
        return sum_value;
}

//Adding two matrices
Cmatrix& Cmatrix::operator +(const Cmatrix& B)
{
        Cmatrix C(row,col);
        static Cmatrix result;
        if ((row == B.row) && (col == B.col))
        {
                for (int r=0;r<row;r++)
                        for (int c=0;c<col;c++)
                        {
                                C.matrixP[(r*col)+c] = matrixP[(r*col)+c] +
                                  B.matrixP[(r*col)+c];
                        }
        }
        result = C;
        return result;
}

//Override multiply M * coeff
Cmatrix& Cmatrix::operator*(const double coeff)
{
        Cmatrix C(row,col);
        static Cmatrix result;
        for (int r=0;r<row;r++)
                for (int c=0;c<col;c++)
                        C.matrixP[(r*col)+c] = coeff * matrixP[(r*col)+c];
        result = C;
        return result;
}

//Overload matrix multiply M1 * M2
Cmatrix& Cmatrix::operator *(const Cmatrix& B)
{
        Cmatrix C(row,B.col);
        static Cmatrix result;
        if (col == B.row)
        {
```

```
                for (int r=0;r<C.row;r++)
                        for (int c=0;c<C.col;c++)
                        {
                                C.matrixP[(r*C.col)+c] = 0;
                                for (int count=0;count<col;count++)
                                        C.matrixP[(r*C.col)+c] = C.matrixP[(r*C.col)+c] +
                                         (matrixP[(r*col)+count]*
                                         B.matrixP[(count*B.col)+c]);
                        }
        }
        else
                cout << "!> Invalid Matrix Multiplication! "
                     << row << "x" << col << " * "
                         << B.row<< "x" << B.col <<endl;
        result = C;
        return result;
}

//Friend function multiply coeff * M
Cmatrix& operator*(const double coeff,const Cmatrix& A)
{
        Cmatrix C(A.row,A.col);
        static Cmatrix result;
        for (int r=0;r<A.row;r++)
                for (int c=0;c<A.col;c++)
                        C.matrixP[(r*A.col)+c] = coeff * A.matrixP[(r*A.col)+c];
        result = C;
        return result;
}
```

REFERENCES

[1] M. De Prycker, "*Asynchronous Transfer Mode: Solution For Broadband ISDN*", 2nd Edition, Ellis Horwood, 1995.

[2] P. Veitch and D. Johnson, "*ATM Network Resilience*", IEEE Network, pp. 26-33, September/October 1997.

[3] R. Onvural, "*Asynchronous Transfer Mode Networks - Performance Issues*", 2nd Edition, Artech House, 1995.

[4] H. Peeters, et al., "*The UNI Protocol Architecture in the Belgian Broadband Experiment*", ISS'92, Yokohama, October 1992.

[5] U. Black, "*ATM: Foundation for Broadband Networks*", Prentice Hall, 1995.

[6] C. Ozveren, et al, "*Reliable and Efficient Hop-by-Hop Flow Control,*", IEEE Journal on Selected Areas in Communications, Vol. 13, No. 4, pp. 642-650, May 1995.

[7] M. Jeffrey, "*Asynchronous Transfer Mode: the Ultimate Broadband Solution?*", Electronics & Communication Engineering Journal, pp. 143-151, June 1994.

[8] S. Kalyanaraman, et al, "*Performance of TCP over ABR on ATM Backbone and with Various VBR Background Traffic Patterns*", ICC'97, Montreal, Canada, Vol. 2, pp. 1035-1041, June 1997.

[9] S. Xu, "*A Theoretic Analysis Model for VBR Video Traffic in ATM Networks*", ICC '97, Montreal, Canada, Vol. 2, pp. 844-848, June 1997.

[10] E. Guarene, et al, "*IP and ATM Integration Perspectives*", IEEE Communications Magazine, pp. 74-80, January 1998.

[11] R. Goyal, et al, "*UBR+: Improving Performance of TCP over ATM-UBR Service*", ICC '97, Montreal, Canada, Vol. 2, pp. 1042-1048, June 1997.

[12] J. Y. Hui, "*Resource Allocation for Broadband Networks*", IEEE. JSAC, Vol. 6, pp. 1598-1608, 1988.

[13] F. Kamoun, "*A New Approach in the Transient Analysis of ATM Multiplexers with Bursty Sources*", Ph.D. Thesis, Concordia University, 1995.

[14] C. Courcoubetis, et al, "*Admission Control and Routing in ATM Networks Using Inferences From Measured Buffer Occupancy*", IEEE Transactions on Communications, Vol. 43, pp. 1778-1784, 1995.

[15] T. Takine, et al, "*Cell Loss and Output Process Analyses of a Finite-Buffer Discrete-Time ATM Queueing System with Correlated Arrivals*", IEEE Transactions on Communications, Vol. 43, No. 2/3/4, pp. 1022-1037, 1995.

[16] T. Egawa, et al, "*QoS Restoration for Dependable Networks*", NOMS '98, New Orleans, LA, Vol. 2, pp. 503-512, February 1998.

[17] T. Huang, "*Performance Analysis of a New Cell Discarding Scheme in ATM Networks*", ICC '97, Montreal, Canada, Vol. 1, pp. 205-209, June 1997.

[18] K. Kawahara, et al, "*Performance Analysis for Reactive Congestion Control for ATM Networks*", IEEE Journal on Selected Areas in Communications, Vol. 13, No. 4, pp. 651-661, May 1995.

[19] H. Bruneel and B. G. Kim, "*Discrete-Time Models for Communications Systems Including ATM*", Kluwer Academic Publishing, 1993.

[20] C. B. Kawwas and M. R. Soleymani, "*Estimating Cell Loss Probability in an ATM Multiplexer with Heterogeneous Traffic*", CCBR'98, pp. 66-77, Ottawa, Canada, June 1998.

[21] D. Anick, D. Mitra and M. M. Sondhi, "*Stochastic Theory of a Data-Handling Sysyem with Multiple Sources*", The Bell System Technical Journal, vol. 61, pp. 1871-1894, October 1982.

[22] T. C. Hou and H. K. Wong, "*Queueing Analysis for ATM Switching of Mixed Continuous-bit-rate and Bursty Traffic*", Proc. INFOCOM '90, San Francisco, CA, pp. 660-667, June 1990.

[23] S. Q. Li, "*A New Performance Measurement for Voice Transmission in Bursty and Packet Switching*", IEEE Trans. Comm., Vol. 35, pp. 1083-1094, Oct. 1987.

[24] A. Baiocchi, et al, "*Loss Performance Analysis of an ATM Multiplexer Loaded with High Speed On-Off Sources*", IEEE J. Selected Areas Comm., Vol. 9, pp.388-393, April 1991.

[25] R. Nargarajan, et al, "*Approximation Techniques for Computing Packet Loss in Finite-Buffered Voice Multiplexers*", IEEE Journal Selected Areas Comm., Vol. 9, pp. 368-377, April 1991.

[26] I. Cidon, et al, "*Bandwidth Management and Congestion Control in PlaNET*", IEEE Communication Magazine, pp.54-64, October 1991.

[27] A. I. Elwalid and D. Mitra, "*Analysis and Design of Rate-Based Congestion Control of High Speed Networks, I: Stochastic Fluid Models, Access Regulation*", Queueing System Theory Application, vol. 9, pp. 29-64, 1991.

[28] R. Guerin, et al, "*Equivalent Capacity and its Application to Bandwidth Allocation in High-Speed Networks*", IEEE Journal Selected Areas Comm., Vol. 9, pp. 968-981, September 1991.

[29] P. Jelenkovic and A. Lazar, "*Multiplexing On-Off Sources with Sub-exponential On Periods: Part I*", IEEE InfoCom '97, vol.1, pp. 187-195, Kobe, Japan, April 1997.

[30] A. Baiocchi, et al, "*On the Significant Parameters for the Characterization of the Cell Loss Behavior in an ATM Multiplexer*", ICC'92, pp. 682-688, 1992.

[31] T. Yang and D. H. Tsang, "*A Novel Approach to Estimation the Cell Loss Probability in an ATM Multiplexer Loaded with Homogeneous On-Off Sources*", IEEE Transactions on Communications, vol. 43, pp. 117-126, 1995.

[32] M. Talla, "*Throughput and Delay Performance of Transport User in Congestion Controlled Hybrid ATM/TDMA Networks*", Concordia University, Montreal, Canada, October 1995.

[33] M. Peyravian and A. Kshemkalyani, "*Connection Preemption: Issues, Algorithms and a Simulation Study*", ", IEEE InfoCom '97, vol.1, pp. 143-151, Kobe, Japan, April 1997.

[34] S. Subramanian, "*Traffic Modeling an Multi-Media Environment*", Concordia University, Montreal, Canada, 1996.

[35] M. N. Huber, et al, "*Proposed Evolutionary Paths for BISDN Signaling*", ISS'92, Yokohama, October 1992.

[36] B. Klessig, "*The Status of ATM Data networking Interoperability Specifications*", 3TECH, Vol. 6, No. 3, July 1995.

[37] K. K. Ramakrishan and P. Newman, "*ATM Flow Control: Inside the Great Debate*", Data Communications, pp. 111-120, Vol. 24, No. 8, June 1995.

[38] R. Jain, "*Congestion Control and Traffic Management in ATM Networks*", Ohio State University, October 1995.

[39] H. Hansen, "*Connection Management Functions of a Private Wireless ATM Network*", Helsinki University of Technology, 1996.

[40] N. Shelef, "*SVC Signaling: Calling All Nodes*", Data Communications, pp. 123-130, Vol. 24, No. 8, June 1995.

[41] N. Zhou, "*Congestion Control Techniques in Hybrid ATM/CDMA Network*", Concordia University, Montreal, Canada, April 1996.

[42] G. D. Stamoulis, et al, "*Traffic Source Models for ATM Networks: A Survey*", J. Computer Communications, Vol. 17, No. 6, pp. 428-438, June 1994.

[43] T. Saadawi, et al, "*Fundamentals of Telecommunications Networks*", Wiley Back, 1994.

[44] J. Cosmas, et al, "*A Review of Voice, Data and Video Traffic Models for ATM*", European Transactions on Telecommunications and Related Technologies, Vol. 5, pp. 11-26, 1994.

[45] G. H. Petit, E. M. Desmet, "*Performance Evaluation of Shared Buffer Multi-server Output Queue Switches used in ATM*", Proceeding of the 7th ITC Specialist Seminar, Paper 9.1, October 1990.

[46] L. Zhang, "*Statistics of Cell Loss and its Application for Forward Error Recovery in ATM Network*", ICC'92, pp. 694-698, 1992.

[47] B. Steyaert and H. Bruneel, "*On the Performance of Multiplexers with Three-State Bursty Sources: Analytical Results*", IEEE Transactions on Communications, Vol. 43, No. 2/3/4, pp. 1299-1303, 1995.

[48] K. Sriram, W. Whitt, "*Characterization Superposition Arrival Processes in Packet Multiplexers for Voice and Data*", IEEE Journal on Selected Areas in Communications", Vol. 7, pp. 833-846, September 1986.

[49] E. Fituri and H. T. Mouftah, "*Resource Estimation and Connection Admission Control in Wireless ATM Networks*", CCBR '97, Ottawa, Canada, pp. 119-124, 1997.

[50] H. Heffes and D. M. Lucantoni, "*A Markov Modulated Characterization of Packetized Voice and Data Traffic and Related Statistical Multiplexer Performance*", IEEE J. Selected Areas in Comm., Vol. SAC-4, No. 6, pp. 856-868, September 1986.

[51] I. Norros, et al, "*The Superposition of Variable Bit Rate Sources in an ATM Multiplexer*", IEEE J. Selected Areas in Comm., vol. 9, pp. 378-387, Apr. 1991.

[52] J. W. Roberts and A.Gravey, "*Recent Results on B-ISDN/ATM Traffic Modeling and Performance Analysis*", Proc. IEEE GLOBECOM '91, pp. 1325-1330, 1991.

[53] P. J. Schweiter, *"A Survey of Aggregation-Disaggregation in Large Markov-Chains"*, Numerical Solution of Markov Chains, W. J. Stewart, New York, Marcel Dekker, 1991.