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

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Université d'Ottawa · University of Ottawa

Connection Admission Control Methods for Multimedia Communication Systems

Zhaogong Shi

A THESIS

Submitted to the School of Graduate Studies and Research

in Partial Fulfilment of the Requirements

for the Degree of

Master of Systems Science

in

Systems Science

University of Ottawa

OTTAWA, ONTARIO, K1N 6N5

June 1997

Acknowledgments

First of all, I wish to express my profound gratitude to Dr. Tet Yeap, my supervisor, for active discussions and constructive suggestions on the subject of this thesis. This work would not have been possible without his consistent guidance and encouragement. I also thank Dr. Yeap for the financial support given to me during my study in the University of Ottawa.

I would like to thank Dr. Daniel Lane, Director of Systems Science, for his kind offer of Von Bertalanffy Entrance Scholarship and University of Ottawa Admission Scholarship. I also wish to thank Dr. Lane for his valuable suggestions and critical review of this thesis. Finally, I want to express my appreciation for what he has done to help me.

I am grateful to my wife Youchen Lou for her hours of beneficial discussions, her patience and moral support.

I wish to express my thanks to those people who have helped me in the production of this thesis.

Abstract

In this thesis, Quality of Service (QoS) and resource management in multimedia communication systems are briefly reviewed. Then, two Connection Admission Control (CAC) methods, one for a media server with multiple disks supporting the simultaneous retrieval of continuous media streams and the other for Asynchronous Transfer Mode (ATM) networks supporting multimedia traffic under varying traffic conditions and different QoS requirements, are proposed, simulated and evaluated in order to improve system performance with respect to the delivery of multimedia services in multimedia communication systems.

The proposed CAC method for the media server with multiple disks can be applied to the different media stream storage patterns on the disks, which is also well-suited to support retrieval service for Redundant Arrays of Inexpensive Disks (RAID). This CAC method can distribute the workload of applications evenly across the disks in order to maximize the system processing capabilities.

The proposed CAC method for ATM networks can be applied to not only an ATM multiplexer but also an output buffer type switching node. Since cell loss rate and average queuing delay are two main important performance objectives provided by the networks to connections, which is closely related to the buffer sizes in the ATM switching node or the multiplexer, our proposed CAC method with buffer model can improve statistical multiplexing gains due to the effects of different buffer sizes. Also, a two-level CAC method is proposed so that our proposed algorithm could be guaranteed to compute in real-time.

Computer simulations are used to help the performance evaluation for two CAC methods. The simulation results show that the proposed CAC methods can provide very high efficient admission control, guarantee the real-time continuous retrieval of delay sensitive media streams or achieve high utilization efficiency by taking account of statistical multiplexing effects based on different buffer sizes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 QoS and Resource Management

Multimedia applications have raised many new challenges in computing environments, especially in multimedia communication systems. This is due to the complexity of these applications and the diversity of media data manipulated. Multimedia news on-demand, multimedia conferencing systems, and multimedia games are some examples of such applications. Multimedia information, such as audio, video, text, graphics and animation, can appear in any of multimedia applications. Resource management is one of most important issues in multimedia communication systems. As we have known, resources, such as CPU, bandwidth, buffer space, etc., are system entities required by multimedia applications for manipulating multimedia data. All services for multimedia applications need system resources to present multimedia data to the end-users. Hence, the real-time requirements for the processing of continuous media streams, such as audio or video, must be met by every system component along each media stream path.

All of the multimedia application requirements can be expressed by Quality of Service (QoS), such as guaranteed media stream retrieval rate, allowed average queuing delay, required cell loss rate, etc. QoS is becoming a common and convenient basis for expressing as well as supporting the variety of multimedia application requirements. It plays an important role in multimedia communication systems nowadays. QoS is generally expressed by a number of parameters called QoS parameters. These QoS parameters can be mapped or translated among different levels of system components and their values may change at different locations during multimedia application execution. The actual QoS parameters depend on the type of media data and the nature of the applications supported.

Because of the shortage of system resources, even high bandwidth networks, large storage capacities on the disks and huge system processing capabilities, resource management is required

when several multimedia applications may compete for the same system resources. In order to guarantee a certain range of QoS-based end-users' requirements and to monitor the offered QoS-based multimedia services continuously, the QoS-based resource manger may consist of the following components: resource adaptation, admission control, negotiation and renegotiation, resource reservation/allocation and deallocation, as shown in Figure 1.1-1.

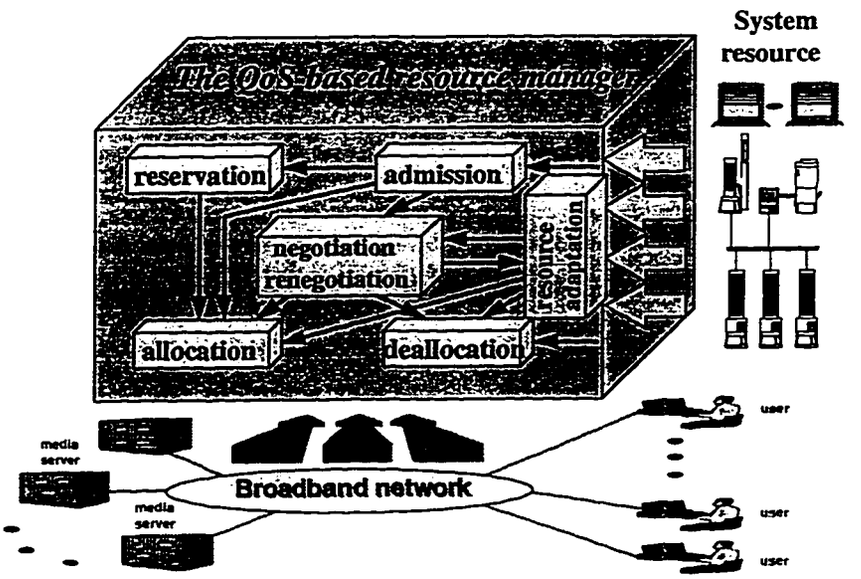


Figure 1.1-1 The QoS-Based Resource Manager

During multimedia call establishment, the QoS-based resource manager may admit, reserve or allocate the required system resources for end-users or multimedia applications, or reject the end-user requests. During multimedia application execution, resulting from end-users' interactions, introduction of a new medium, addition of a new end-user to the application, or occurrence of resource problems, the resource manager will apply resource adaptation for these changes if possible,

or take further actions such as negotiation, renegotiation, etc. for these new QoS-based requirements. Sometimes, the resource manager may notify the applications of QoS degradation if there is lack of system resources. After the execution of a multimedia application, those system resources must be deallocated, which means that CPU, bandwidth and buffer space, etc., must be free for the coming applications. The admission control is an important step for the QoS-based resource manager. It is used to check the availability of shared system resources in a multimedia communication system. If the minimal bound of QoS requirements cannot be satisfied, a reject message will be sent to end-user to reject the application requirements.

1.2 Motivation

1.2.1 Media Server

The low I/O bandwidth of the current disk technology, and the large seek time overhead as well as the slow rotation speed of the current disk drive present a real challenge to some multimedia applications. For example, during the execution of multimedia news on-demand, different continuous media streams may be required to be displayed simultaneously. Sometimes the display of one video stream may require synchronization with the other video or audio stream that is already running. Thus, a real-time continuous retrieval of these media streams from disk storage devices is required. So far, there have been many techniques or methods to support a real-time retrieval and display of multimedia streams. But, some methods are limited to only one single disk storage device to support the real-time retrieval of several media streams simultaneously [2], the others handle only specific media stream storage patterns or use read-ahead and buffering that need huge memory [2][4]. Moreover, very few techniques and methods have been proposed for providing the high efficient performance of multimedia data retrieval supported by the media server with multiple disks or Redundant Arrays of Inexpensive Disks (RAID). Therefore, a new approach will be required to support the real-time retrieval of delay sensitive media streams. It is desirable that the Connection Admission Control (CAC) method can be used for the media server with multiple disks or RAID and support the following four different media stream storage patterns:

- The entire media stream, such as audio or video, is stored on one disk.
- The entire media stream is duplicately stored on several disks in the same media server.
- The media stream is segmented into several substreams, each of which may be stored on different disk in the same media server.
- The media stream is declustered across several disks in the same media server.

Based on the above motivation, a new CAC method will be presented in this thesis. It is used as admission retrieval test during an application call establishment so that the QoS-based resource manager can provide the appropriate resources for the application as long as the desired QoS requirements can be met. In order to propose the new CAC method, which is able to handle the real-time retrieval of media streams served by the media server with multiple disks or RAID, the work reported in this thesis is extended and developed from their experience [2].

1.2.2 ATM networks

ATM traffic control is one of most important and difficult issues in multimedia communication systems, which remains a controversial problem in practice that how to provide network resources required by end-users. One of the reasons that traffic control is difficult in ATM networks is the diversity of traffic characteristics and the different QoS requirements. Moreover, requirements for adaptability, flexibility, and robustness also present new challenges in ATM traffic control. So far, there have been a lot of CAC techniques and research results published or reported. But, methods proposed by many researchers [11] [12] [26] still have certain problems. Some methods cannot be applied to the case where the diversity of traffic sources is supported. The other methods cannot operate in real-time, when various types of connections are multiplexed [11] [12]. Also, their CAC algorithms are basically for ATM multiplexers since their methods are not applied for the actual switching nodes. In some methods [14], they assumed that only a bufferless model be discussed or evaluated, which means that the effects of different buffer sizes in switching nodes or multiplexers were not considered.

During our research about CAC methods for ATM networks, our interest has gradually concentrated on the work reported by [14]. Compared with the other CAC methods, their work is

more complete and effective. Since it deals with not only a multiplexer but also a switching node. As a result, we wish to improve and extend their work so that our proposed method can keep the features developed by their method in [14] and take account of the statistical multiplexing gains based on different buffer sizes. In addition, a two-level CAC method is to be proposed so that our proposed algorithm could be guaranteed to compute in real-time.

Note: The admission control used for a media server is called retrieval admission control while the admission control used for an ATM network is called connection admission control. In this thesis, we call the retrieval/connection admission control as the CAC, which is applied for a media server and an ATM network.

1.3 Main Objectives

The main objectives of this thesis are to propose, simulate and evaluate two CAC methods, one for the media server with multiple disks or RAID supporting the simultaneous retrieval of continuous media streams and the other for ATM networks supporting multimedia traffic under varying traffic conditions and different QoS requirements. The purpose is to improve entire system performance with respect to the delivery of multimedia services in a multimedia communication system.

1.4 Thesis Outline

This thesis is organized as follows:

In chapter 2, background and literature review are presented, which give an overview of some published research results for CAC methods. Also, two CAC methods from [2] and [14], which are fundamental theories in this thesis, are briefly reviewed, respectively.

In chapter 3, the detailed analysis of the media server with multiple disks, which can support the simultaneous retrieval of continuous media streams with different rates, is presented. Then, the proposed CAC algorithm is presented to control the acceptance of new retrieval request from one

of the given four different media stream storage patterns. At the end of this chapter, the simulation results are shown that the CAC algorithm can provide very high efficient admission control and guarantee the real-time continuous retrieval of delay sensitive media streams.

In chapter 4, the preliminaries for the proposed CAC method are first described. Then, the proposed CAC method with buffer model, which can be applied to an ATM multiplexer and an output buffered switching node, is developed, followed by the performance evaluation for the CAC algorithm. Simulation results show that the proposed CAC method can achieve high utilization efficiency by taking account of different buffer sizes. A two-level CAC method is also proposed so that our proposed algorithm could be guaranteed to compute in real-time.

The thesis ends with chapter 5, which concludes on our current research work and points out problems and suggestions for improvements and future work.

Chapter 2

Background and Literature Review

2.1 Media Server

A typical multimedia communication system consists of media servers, workstations and other multimedia devices connected through the broadband network, such as Broadband Integrated Services Digital Network (B-ISDN), based on ATM technologies. Thus, end-users can work at workstations at different locations and share media servers or databases. One of the requirements of a multimedia communication system is to provide the desired QoS retrieval methods for multimedia data or streams. The disks in a media server must have very large storage capacities and fast data transfer rates to guarantee the continuous retrieval of media streams, such as audio and video. The simultaneous retrieval of continuous media streams from disks through broadband networks must be performed according to the appropriate data retrieval rates. In the case of video, real-time deadlines must be met for the display of successive video frames in order to flow smoothly without any gaps or interruptions. The data retrieval rates and the number of media stream retrieval supported by a media server depend on the characteristics of disk storage devices, the media stream storage patterns on the disks, the achieved compression rates and the chosen basic media block size. In order to perform media stream retrieval, transfer and presentation, the new storage model and real-time retrieval techniques need to be designed. Some techniques and methods proposed in the past few years are as follows:

The techniques in [20] were developed for the real-time requirements of the digital audio playback. The real-time requirements are described in terms of the consumption rate of the audio and the nature of the data retrieval rate from the disk storage device. Making use of their techniques, bounds are derived for buffer space requirements for three common retrieval scenarios. They also described and compared the different data placement strategies for multimedia data streams.

According to their paper, the placement strategies of multimedia data can be categorized as: scattered noninterleaved data placement, contiguous noninterleaved data placement, scattered interleaved data placement, and contiguous interleaved data placement. There are various advantages and disadvantages to each strategy. The description about each strategy can be found in [2] [20]. Currently, the data placement strategy used by most of the multimedia applications is contiguous interleaved placement, which is that the media stream is organized on the disk storage devices in terms of media blocks separated by the gaps for the same media stream.

The mechanisms for merging storage patterns of multiple media streams in [4] were developed by filling the gaps between media blocks of one stream with media blocks of other streams. Moreover, both algorithms developed are as follows: An on-line merging algorithm is suitable for merging a new media stream into a set of already stored streams and an off-line merging algorithm can be applied to the storage of a set of media streams before any of them have been stored on the disk. As a consequence of merging, storage patterns of media streams may become perturbed slightly. To compensate this, read-ahead and buffering is required so that continuity of retrieval remains satisfied. The techniques for minimizing both read-ahead and buffering were presented. Their storage model can guarantee the real-time requirements of video retrieval, and can reduce the data copying amounts required during data editions. Using merging, their disk usage can approach 90 percent. However, read-ahead and buffering during the retrieval increases buffer requirements and complicates the computational performance.

A parallel multimedia information system was developed and key technical ideas that enable the system to support real-time display of media streams were proposed in [3]. The techniques for media stream storage are as follows: first, a media stream is declustered across several disk drives, enabling the system to utilize the aggregate bandwidth of multiple disks to retrieve a media stream in real-time. Second, the workload of an application is distributed evenly across the disk drives in order to maximize the processing capability of the system. Moreover, to support simultaneous display of several media streams for different users, two alternative approaches for media stream retrieval were also proposed. The first approach multitasks a disk drive among several requests while the second replicates the data and dedicates resources to each individual request. The tradeoffs associated with each approach using a simulation model were investigated. Their results

demonstrated the superiority of the replication approach.

Some techniques use RAID as a high bandwidth secondary storage device. The central idea behind RAID is to construct an I/O subsystem which consists of a disk controller and an array of disk drives. The controller can collect a file or a multimedia object across all the drives in order to provide a high data rate upon its retrieval. In order to minimize the probability of the data becoming unavailable in the presence of disk failures, RAID has introduced a taxonomy of five different organizations of disk arrays, beginning with mirrored disks and progressing through a variety of alternatives with different performance and reliability characteristics. These arrays offer performance enhancement, which increases linearly with the number of disks, but there is a practical limit of five-drive array [15][18] since large disk arrays are highly vulnerable to disk failures. A disk array with 5 disks is 5 times more likely to fail than a single-disk array.

Streaming RAID - a disk array management system for video files was proposed in [5]. This system manages an array of Winchester disks, and uses a disk access algorithm particularly suitable for video streaming, and is thus referred to as 'streaming RAID'. The system performance was also characterized by determining the number of streams that can be supported for a given memory size and a given start-up latency requirement.

The next-generation multiprocessors that will be deployed as servers in a multimedia environment were suggested in [6]. Current uniprocessor severs cannot handle multimedia traffic internally and effectively for delivering the network's high bandwidth to the processing and storage subsystems. Therefore, three scalable, subsystem-based, multimedia server architectures using ATM to tackle this problem were presented. Here, ATM protocols and technology would be used as the interconnect mechanism for a subsystem-based multimedia server.

An admission control scheme for simultaneous retrieval of multiple continuous media streams, which can guarantee real-time requirements, was proposed by [2]. By combining this admission control scheme with buffering techniques, a dynamic admission control algorithm was developed to control the acceptance of the new retrieval request. But, this admission control scheme is limited to only a single disk storage device to support the media stream retrieval.

However, some models are limited to only a single disk storage device to support the real-time retrieval of several media streams [2] and the others can handle only specific media stream

storage patterns or use read-ahead and buffering that need huge memory [2][4]. Moreover, very few techniques have been proposed for providing the high efficient performance of media stream retrieval supported by a media server with multiple disks or RAID. Therefore, a new approach will be required to support the real-time retrieval of delay sensitive media streams. In order to develop our CAC method, which is able to handle the real-time retrieval of media streams served by a media server with multiple disks or RAID, the work in this thesis is based on the method developed in [2], which is briefly described as follows:

According to the reported work, the simultaneous retrieval of N continuous media streams is done by a transfer process and N consuming processes, which is shown in Figure 2.1-1. Since all media blocks are disk residence and only one disk head is in serving, the transfer process is shared by N retrieving media streams, the consuming processes of different media streams are independent from each other and the consumption rates may be different depending on the application. To guarantee the requirements of continuous retrieval of N media streams, a pair of equal sized buffers is assigned to each retrieving media stream. The buffer size of each media stream can be the same or different depending on its consumption rate. By relating the retrieval buffer size with its consumption rate, the same T_c for all the retrieving media streams can be obtained. The control of the simultaneous retrieval of N continuous media streams can be expressed as

$$\frac{B_i}{R_{ci}} = T_c \qquad (1 \le i \le N) \tag{2.1-1}$$

where the subscript i corresponds to the ith specific media stream. B_i is the buffer size of the ith media stream. R_{ci} is the buffer consumption rate of the ith media stream, and the buffer empty duration, expressed as T_c, is the time required for a consuming process to empty one of the retrieval buffers.

In each buffer empty duration, the transfer process will fill N empty buffers with media blocks from the disk respectively, in which rotational latency is incurred to skip the gaps between two successive media blocks and the average seek time needs to be considered when the disk head switches between the media blocks of two retrieving streams. Then, the transfer process will perform

an idle delay to wait for the beginning of next buffer empty duration.

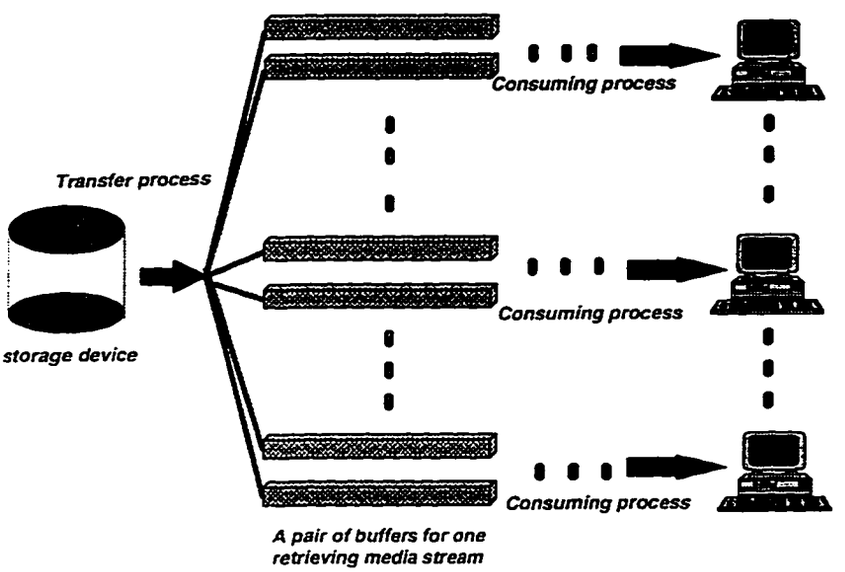


Figure 2.1-1 Retrieval and Playback of *N* Continuous Media Streams

In general, if there are N retrieving media streams, the buffer empty duration T_c can be expressed as

$$T_c = \sum_{i=1}^{N} T_{fi} + NT_s + \sum_{i=1}^{N} T_{ri} + T_{wN} \qquad (2.1\text{-}2)$$

In the above equation, T_{fi} is the total time needed to transfer contiguous media blocks from disk and fill out the retrieval buffer of the ith media stream completely, T_s is the average seek time when the disk head switches between the media blocks of two retrieving media streams, T_{ri} is the

rotational latency occurred while the retrieval buffer of the ith media stream is filled and T_{wN} is the transfer idle period when a single disk head is serving the simultaneous retrieval of N continuous media streams.

From the above equation (2.1-2), the condition that N continuous media streams can be retrieved simultaneously can be determined. It can be expressed as $T_{wN} \geq 0$, or as

$$T_{wN} = T_c - (\sum_{i=1}^{N} T_{fi} + NT_s + \sum_{i=1}^{N} T_{ri}) \geq 0 \tag{2.1-3}$$

If the retrieval buffer of the ith media stream can contain up to K_i media blocks, then

$$B_i = K_i M \tag{2.1-4}$$

$$T_{fi} = \frac{B_i}{R_{dt}} = \frac{K_i M}{R_{dt}} \tag{2.1-5}$$

and

$$T_{ri} = \frac{(K_i - 1)G_i}{R_{dt}} \tag{2.1-6}$$

where media block size M is the number of physical sectors occupied by a media block, logically, M is the basic object of a continuous media stream, gap size G_i is the number of physical sectors between two successive media blocks of the ith media stream on the disk and R_{dt} is defined as the data transfer rate of the disk storage device.

With the details of each term, the equation (2.1-3) can be rewritten as:

$$\frac{B_i}{R_{ci}} - (\sum_{i=1}^{N} \frac{B_i}{R_{dt}} + NT_s + \sum_{i=1}^{N} (K_i - 1)\frac{G_i}{R_{dt}}) \geq 0 \tag{2.1-7}$$

By satisfying this condition, N continuous media streams can be retrieved simultaneously with the required media stream retrieval rates.

2.2 ATM Networks

B-ISDN has received much attention recently due to demand for networks that can support a wide variety of traffic and diverse services. ATM has been chosen as the transport and switching technique for these networks. This is a statistical multiplexing cell switching technique that delivers messages by using the fixed-sized cells of 53 bytes and is able to offer bandwidth on demand based on service requirements. This capability is attractive for networks carrying burst traffic. If statistic multiplexing is used to allocate bandwidth on a network, congestion may occur when many sources become active at once. One aspect of congestion control is CAC where congestion is controlled by limiting the amount of traffic in the ATM networks. When a call arrives, a decision is made to accept or reject it based on the current load in the network. The end-user must provide the desired QoS requirements to the system in terms of required cell loss rate, allowed average queuing delay and traffic descriptors, which can characterize their traffic. Consultative Committee on International Telegraphy and Telephony (CCITT) recommended that traffic descriptors are peak cell rate, average cell rate and average burst length [7]. So far, there have been many techniques and theories published or reported about CAC for ATM networks, which is one of most important and difficult issues in the ATM traffic control. In the following, several published techniques and methods are reviewed.

Different approaches for effective bandwidth allocation in an ATM link were investigated [8]. The linear approximation and nonlinear approximation for the admission control were discussed. Results that are useful in constructing some simple admission models were suggested. A framework for flow control and routing in ATM networks based on methods developed for circuit-switched networks was proposed. In fact, it is shown that, in some cases, the linear approximation may cause significant errors and in general is too optimistic since a mixture of call classes with different link utilization is treated as the homogeneous case. The proposed nonlinear approximation is significantly more accurate. But, it is based on only a two-parameter characterization of statistical properties of call mixtures.

By using a synthesis approach, E. D. Sykas, et al.[9], defined and calculated an 'effective bandwidth' for On-Off ATM sources, which can be used as a connection acceptance criterion in ATM

networks, thus leading to high utilization of ATM network resources. Although this method offers an easy, fast and practical way for calculating the effective bandwidth of an ATM source, the analytical formula is derived based on several plotted figures, which may not be accurate enough to calculate the effective bandwidth of a wide spectrum of ATM traffic characteristics.

The exact analysis of the blocking conditions at the burst level, experienced by different types of traffic in an ATM network was proposed in [10]. A basic assumption is that traffic is generated by a finite set of independent two-state sources that share equally the available bandwidth. When all sources are active, the bandwidth demanded may exceed the capacity of one or more links in the network. The burst blocking probabilities were calculated by means of a recursive relation that considerably simplifies the complexity of the problem. These results supply the analytical basis for an admission control mechanism that guarantees a predefined performance threshold in terms of burst loss probability for the classes of traffic supported.

A CAC method was proposed and evaluated in [14]. The proposed method is suitable for real-time operation even in large diversity of connection types, because the amount of calculation for CAC is reduced remarkably compared with conventional algorithms. Moreover, the amount of calculation for the algorithm does not increase even when the number of connection types increases. The proposed method uses a probability function for the number of cells transferred from multiplexed connections and uses recursive equations in estimating cell loss rates. It is assumed in this method that only a bufferless model was discussed and evaluated, which means that the effect of the buffer in the switching node or ATM multiplexer is not considered.

When various kinds of traffic are multiplexed, some CAC methods cannot operate in real-time, because of their computational complexity [11] [12] and vast diversity of traffic sources supported [9]. Moreover, many methods [13] discussed CAC algorithms, but they did not discuss their methods applied to the actual switching node. Every proposed algorithm is basically for an ATM multiplexer.

During our research of the CAC methods for ATM networks, our interest has gradually concentrated on the work reported by [14]. Compared with other CAC methods, the methods in [14] are more effective. They can deal with not only a multiplexer but also a switching node. Because cell loss rate and average queuing delay are the two main important performance objectives provided by

the network to connections, which is closely related to the buffer sizes in an ATM switching node or a multiplexer. Hence, the CAC methods for ATM networks should consider the effects of the different buffer sizes. We wish to improve and extend the work in [14] so that our proposed method has the features of [14] as well as can improve statistical multiplexing gains due to different buffer sizes in a switching node or an ATM multiplexer. In the following, the main formulas and theories developed in [14] are described.

One unit-time used in the method without buffer model [14] is defined as the time that the multiplexer or switching node transfers one cell to the multiplexed line. V (cells/unit -time) is defined as the cell transmission speed of the multiplexed line. T_1 is defined as the inverse of V_{pl}, here, V_{pl} is defined as the maximum cell transmission speed of low speed connections ($T_1 = 1/V_{pl}$). The low speed connection is defined as the connection whose peak cell transmission speed is smaller than the threshold value ($V_{pl} = V/100$). On the other hand, the high speed connection is defined as the connection whose peak cell transmission speed is larger than V_{pl} (= $V/100$).

In the following, the method without buffer model is assumed. When the number of cells, which are transferred from high speed and low speed connections, are larger than M_{T_1} cells during T_1 period, the overflowed cells are discarded. Here, it is assumed that the multiplexed line can transfer M_{T_1} cells during T_1. In order to evaluate the cell loss rate for this bufferless model, the probability function for the number of transferred cells from the multiplexed connections during T_1 period should be obtained. The probability $G(n)$ that n cells are transferred from the multiplexed connections during T_1 is defined. $G(n)$ is given by the following equation (2.2-1).

$$G(n)=\sum_{k=0}^{n} P(k)F(n-k) \qquad (0\le n\le M_{T_1}) \tag{2.2-1}$$

here, $P(k)$ is the probability that k cells are transferred from low speed connections during T_1. $F(n-k)$ is the probability that $n-k$ cells are transferred from high speed connections during T_1 (for details of $F(k)$ and $P(k)$, see [14] or the following chapter 4). Here, we use the notations such as $F(k)$ and $P(k)$, in order to keep consistency with the notations in [14].

Then, the cell loss rate $CLR(bufferless)$ is given by equation (2.2-2),

$$CLR(bufferless)=\frac{1}{\rho M_{T_1}}\sum_{n=M_{T_1}+1}^{\infty}(n-M_{T_1})G(n) \tag{2.2-2}$$

Equation (2.2-2) can be transformed as equation (2.2-3), which can reduce the summation from $[M_{T_1}+1, \infty]$ to $[0, M_{T_1}]$.

$$CLR(bufferless)=\frac{1}{\rho M_{T_1}}\sum_{n=0}^{M_{T_1}}(M_{T_1}-n)G(n)-\frac{1-\rho}{\rho} \tag{2.2-3}$$

where, ρ ($0 \le \rho \le 1$) is the average offered load to the multiplexed line, as defined in the following:

$$\rho=\sum_{j=1}^{L_l}\frac{N_j V_{aj}}{V}+\sum_{s=1}^{L_h}\frac{N_s V_{as}}{V} \tag{2.2-4}$$

L_l is the number of low speed connection types. The number of type j ($1 \le j \le L_l$) low speed connections is N_j. Peak cell transmission speed of a type j low speed connections is V_{pj} and average cell transmission speed of a type j low speed connections is V_{aj}. On the other hand, L_h is the number of high speed connection types. The number of type s ($1 \le s \le L_h$) high speed connections is N_s. Peak cell transmission speed of a type s high speed connections is V_{ps} and average cell transmission speed of a type s high speed connections is V_{as}.

As a result, the amount of calculation for obtaining a new $\{P(k)\}$, which is the set of probabilities for low speed connections, and the calculation for obtaining a new $\{F(k)\}$, which is for high speed connections, are of only order M_{T_1} when a new connection is established or torn down. This means that the calculation for evaluating cell loss rate is of order $M_{T_1}^2$. Moreover, the amount of calculation is independent of the number of connection types. Therefore, the computation does not increase even when the number of connection types increases. The performance between this method and the proposed method will be presented and evaluated in the following Chapter 4.

Chapter 3

Connection Admission Control for Media Server

3.1 Media Server with Multiple Disks

Multimedia applications involve a large amount of multimedia data that need to be stored, retrieved, manipulated, processed and exchanged between a group of end-users. A simple multimedia system may consist of workstations and media servers (each with a single disk storage device) connected by a broadband network. Several end-users' requests for retrieving continuous media streams may arrive at the media server at the same time, so the media server may try to serve all the requests within a specific deadline. Although some current servers can provide a fast response and method for the media stream retrieval, they may not guarantee continuous retrieval of several media streams simultaneously, especially for media streams requiring high I/O bandwidth. To improve the retrieval performance, the media servers with multiple disks or RAID could be used in a multimedia communication system, as shown in Figure 3.1-1. The total data transfer rates of such system would be improved dramatically to support more retrieval requests and reduce the delays of media storage access.

Currently, the media data placement strategy on the disks used by most of the multimedia applications is contiguous interleaved placement. For example, a sequence of video frames is organized on the disk storage devices in terms of media blocks separated by the same sized gaps for the same video stream. This kind of storage model can guarantee the desired QoS requirements of media stream retrieval, especially video stream retrieval. In this thesis, the same media data placement strategy is used.

In general, if there are a total of S disks in a media server and the real-time retrieval of continuous media streams is being served by each disk H_k, by extending equation (2.1-2), the buffer empty duration $T_c^{H_k}$ can be expressed as:

$$T_c^{H_k} = \sum_{i=1}^{N_k} T_{fi}^{H_k} + \sum_{i=1}^{N_k} T_{si}^{H_k} + \sum_{i=1}^{N_k} T_{ri}^{H_k} + T_{wN_k}^{H_k} \qquad (1 \le k \le S) \tag{3.1-1}$$

where H_k represents the kth disk storage device. N_k is the total number of the media streams being served by disk H_k. $T_{fi}^{H_k}$ is the total time needed to transfer contiguous media blocks from disk H_k and fill the retrieval buffer of the ith media stream ($1 \le i \le N_k$). $T_{si}^{H_k}$ is the seek time when the head of disk H_k switches between the media blocks from i-1th retrieving stream to ith retrieving stream. $T_{ri}^{H_k}$ is the rotational latency occurred while the retrieval buffer of the ith media stream ($1 \le i \le N_k$) is filled by disk H_k and $T_{wN_k}^{H_k}$ is transfer idle period when the head of disk H_k is serving the simultaneous retrieval of N_k continuous media streams.

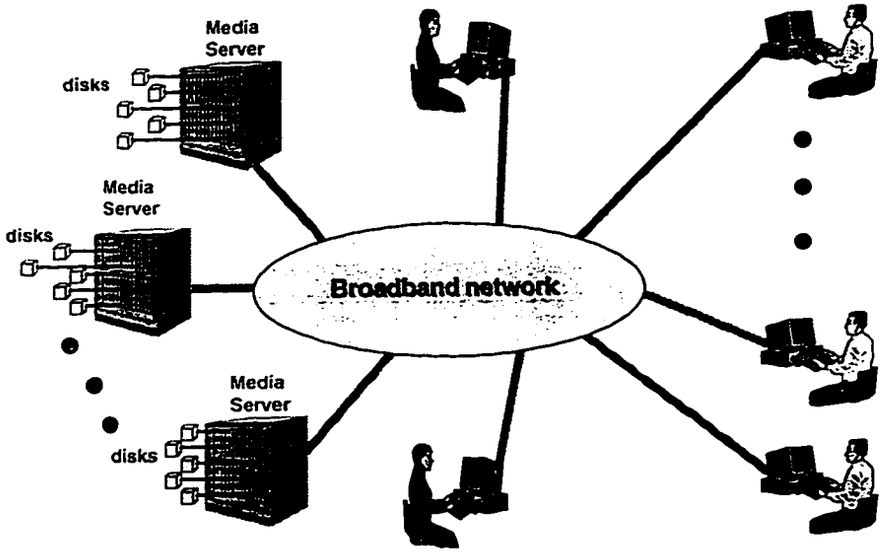


Figure 3.1-1 The Media Server with Multiple Disks Supporting Simultaneous Retrieval of Continuous Media Streams

If $T_{wN_k}^{H_k} \geq 0$, with the details of each term, the above equation (3.1-1) can be rewritten as:

$$\frac{B_i^{H_k}}{R_{ci}^{H_k}} - \left(\sum_{i=1}^{N_k} \frac{B_i^{H_k}}{R_{dt}^{H_k}} + \sum_{i=1}^{N_k} T_{si}^{H_k} + \sum_{i=1}^{N_k} (K_i^{H_k} - 1) \frac{G_i^{H_k}}{R_{dt}^{H_k}}\right) \geq 0 \qquad (1 \leq k \leq S) \qquad (3.1\text{-}2)$$

where $B_i^{H_k}$ is the retrieval buffer size of the ith media stream $(1 \leq i \leq N_k)$ being served by disk H_k. $R_{ci}^{H_k}$ is the buffer consumption rate of the ith media stream $(1 \leq i \leq N_k)$ being served by disk H_k. $K_i^{H_k}$ is the total number of media blocks contained by the retrieval buffer of the ith media stream. $G_i^{H_k}$ is the gap size between two successive media blocks of the ith media stream on disk H_k and $R_{dt}^{H_k}$ is the data transfer rate of disk H_k.

By satisfying the above condition (3.1-2), N_k continuous media streams can be retrieved simultaneously by disk H_k $(1 \leq k \leq S)$ without violating the desired QoS retrieval requirements or with guaranteeing the required media stream retrieval rates.

When all the disks are serving the simultaneous retrieval of continuous media streams, upon reception of a new retrieval request, the CAC has to decide whether this new request can be accepted. Note that the acceptance of a new request must not violate the retrieval requirements of any of the current retrieving streams. The retrieval requirements of this new request also need to be guaranteed during the retrieval. Moreover, there must be the media stream resource being requested on disk H_{k_1}, disk H_{k_2}, ..., or disk H_{k_n} and enough length of their transfer idle period $T_{wN_{k_1}}^{H_{k_1}}$, $T_{wN_{k_2}}^{H_{k_2}}$,...,or $T_{wN_{k_n}}^{H_{k_n}}$ $(1 \leq k_1, k_2, \ldots, k_n \leq S)$ per buffer empty duration. In order to explore some important properties, a simplified retrieval model will be used for our study. Some assumptions of our retrieval model are as follows:

- The disk storage devices are dedicated to only one continuous media stream type (e.g., a video stream or audio stream) with different rates.
- The contiguous interleaved placement is used in terms of media blocks separated by the same sized gap for the same media stream.
- Two equal sized buffers are allocated for each retrieving media stream.
- The round robin method (services in sequence) is chosen to be the service discipline of the

data transfer process when multiple media streams are being retrieved simultaneously by the disk.

- All the disks H_k ($1 \le k \le S$) have the same characteristics, which satisfy the following conditions:
  (1) $R_{dt}^{H_k} = R_{dt}$
  (2) $M^{H_k} = M$
  (3) $T_c^{H_k} = T_c$ and the same starting time point

Since the proposal method is based on the method developed by [2], some of the retrieval assumptions are similar to those in their method.

The proposed media stream storage patterns on the disks, which can be handled by the proposed CAC method, are categorized into the following four types:

Type A: The entire media stream, such as audio or video, is stored on one disk.

Type B: The entire media stream is duplicately stored on several disks in the same media server.

Type C: The media stream is segmented into several substreams, each of which may be stored on different disks in the same media server.

Type D: The media stream is declustered (distributed uniformly) across several disks in the same media server.

Based on the above media stream storage patterns on the disks, the concrete CAC method for solving the different five cases is given as follows:

Case 1 (solving type A):

The media stream resource being requested, such as video or audio, is stored on disk H_k ($1 \le k \le S$), and the given buffer consumption rate is R_c. The transfer idle period is $T_{wN_k}^{H_k}$ when the head of disk H_k is serving the retrieval of N_k continuous media streams simultaneously.

According to the above retrieval conditions, the equations derived from equations (2.1-1) and (2.1-4) can be expressed as:

$$B_{N_k+1}^{H_k} = T_c R_c \qquad (3.1\text{-}3)$$

$$K_{N_k+1}^{H_k} = \frac{B_{N_k+1}^{H_k}}{M} \tag{3.1-4}$$

By satisfying the following condition (3.1-5), the new media stream will be served by disk H_k, otherwise, the retrieval request for this media stream will be rejected.

$$T_{wN_k}^{H_k} \geq \frac{B_{N_k+1}^{H_k}}{R_{dt}} + (K_{N_k+1}^{H_k} - 1)\frac{G_{N_k+1}^{H_k}}{R_{dt}} + T_{sN_k+1}^{H_k} \tag{3.1-5}$$

where $B_{N_k+1}^{H_k}$ is the size of retrieval buffer that may be allocated for the new retrieval request, $B_{N_k+1}^{H_k}$ $= K_{N_k+1}^{H_k} M$ from (3.1-4), which means that the retrieval buffer can contain up to $K_{N_k+1}^{H_k}$ media blocks, and $G_{N_k+1}^{H_k}$ is the gap size of the media stream resource being requested on disk H_k.

When one retrieval is newly established, $T_{wN_k}^{H_k}$ and N_k need to be updated by using the following equations:

$$T_{wN_k}^{H_k} = T_{wN_k}^{H_k} - [\frac{B_{N_k+1}^{H_k}}{R_{dt}} + (K_{N_k+1}^{H_k} - 1)\frac{G_{N_k+1}^{H_k}}{R_{dt}} + T_{sN_k+1}^{H_k}] \tag{3.1-6}$$

and

$$N_k = N_k + 1 \tag{3.1-7}$$

When one retrieval is newly finished, $T_{wN_k}^{H_k}$ and N_k need to be updated by using the following equations:

$$T_{wN_k}^{H_k} = T_{wN_k}^{H_k} + [\frac{B_{N_k}^{H_k}}{R_{dt}} + (K_{N_k}^{H_k} - 1)\frac{G_{N_k}^{H_k}}{R_{dt}} + T_{sN_k}^{H_k}] \quad (3.1\text{-}8)$$

and

$$N_k = N_k - 1 \quad (3.1\text{-}9)$$

here, it is assumed that the N_kth media stream is newly finished.

Case 2 (solving type B):

The entire media stream resource being requested is stored in duplicate on several disks, such as $H_1, \ldots, H_k, \ldots, H_{s_1} (2 \le s_1 \le S)$, and the given buffer consumption rate is R_c. Their transfer idle periods are $T_{wN_1}^{H_1}, \ldots, T_{wN_k}^{H_k}, \ldots, T_{wN_{s_1}}^{H_{s_1}}$, respectively, when the head of disk H_k $(1 \le k \le s_1)$ is serving the retrieval of N_k continuous media streams simultaneously.

According to the above the retrieval conditions, the following equations can be derived from equations (2.1-1), (2.1-4) and (3.1-5):

$$B = B_{N_k+1}^{H_k} = T_c R_c \qquad (1 \le k \le s_1) \quad (3.1\text{-}10)$$

$$K = K_{N_k+1}^{H_k} = \frac{B_{N_k+1}^{H_k}}{M} \qquad (1 \le k \le s_1) \quad (3.1\text{-}11)$$

$$Result_k = T_{wN_k}^{H_k} - [\frac{B_{N_k+1}^{H_k}}{R_{dt}} + (K_{N_k+1}^{H_k} - 1)\frac{G_{N_k+1}^{H_k}}{R_{dt}} + T_{sN_k+1}^{H_k}] \qquad (1 \le k \le s_1) \quad (3.1\text{-}12)$$

By satisfying the following condition (3.1-13), the new media stream will be served by disk

H_g, otherwise, this retrieval request will be solved by the following Case 3:

$$Result_g = \max(Result_1,...,Result_k,...,Result_{s_1}) \ and \ Result_g \geq 0 \qquad (1 \leq g \leq s_1) \qquad (3.1\text{-}13)$$

Case 2 can distribute the workload of the applications evenly across the disks so as to maximize the system processing capabilities.

When one retrieval is newly established, $T_{wN_g}^{H_g}$ and N_g need to be updated by using the following equations:

$$T_{wN_g}^{H_g} = T_{wN_g}^{H_g} - [\frac{B_{N_g+1}^{H_g}}{R_{dt}} + (K_{N_g+1}^{H_g} - 1)\frac{G_{N_g+1}^{H_g}}{R_{dt}} + T_{sN_g+1}^{H_g}] \qquad (3.1\text{-}14)$$

and

$$N_g = N_g + 1 \qquad (3.1\text{-}15)$$

When one retrieval is newly finished, $T_{wN_g}^{H_g}$ and N_g need to be updated by using the following equations:

$$T_{wN_g}^{H_g} = T_{wN_g}^{H_g} + [\frac{B_{N_g}^{H_g}}{R_{dt}} + (K_{N_g}^{H_g} - 1)\frac{G_{N_g}^{H_g}}{R_{dt}} + T_{sN_g}^{H_g}] \qquad (3.1\text{-}16)$$

and

$$N_g = N_g - 1 \qquad (3.1\text{-}17)$$

here, it is assumed that the N_gth media stream is newly finished.

Case 3 (solving type B and all *Result* $_k < 0$ ($1 \le k \le s_1$)):

The entire media stream resource being requested is duplicately stored on several disks, such as H_1,..., H_k ,..., H_{s_1} ($2 \le s_1 \le S$), but all *Result*$_k$ ($1 \le k \le s_1$) from Case 2 are less than zero.

In order that several disks are working together to serve the retrieval of this media stream, the maximum number of media blocks served by each disk should be calculated based on its remaining transfer idle period $T_{wN_k}^{H_k}$ ($1 \le k \le s_1$). Equation (3.1-5) can be changed as the following equation (3.1-18):

$$T_{wN_k}^{H_k} = \frac{K^{H_k}M}{R_{dt}} + (K^{H_k}-1)\frac{G_{N_k+1}^{H_k}}{R_{dt}} + T_{sN_k+1}^{H_k} \qquad (1 \le k \le s_1) \tag{3.1-18}$$

where K^{H_k} is defined as the maximum number of media blocks served by disk H_k based on its remaining transfer idle period $T_{wN_k}^{H_k}$. Further, equation (3.1-18) can be rewritten as equation (3.1-19):

$$K^{H_k} = \left\lfloor \frac{(T_{wN_k}^{H_k} - T_{sN_k+1}^{H_k})R_{dt} + G_{N_k+1}^{H_k}}{M + G_{N_k+1}^{H_k}} \right\rfloor \qquad (1 \le k \le s_1) \tag{3.1-19}$$

here, $\lfloor x \rfloor$ means the largest integer that is not beyond x.

Sort K^{H_1},..., K^{H_k},..., $K^{H_{s_1}}$ into decreasing order, such as $K^{H'_1}$,..., $K^{H'_k}$,..., $K^{H'_{s_1}}$. From equations (3.1-10) and (3.1-11), the total number of media blocks for the media stream being requested within buffer empty duration T_c can be given as follows:

$$K = \frac{T_c R_c}{M} \tag{3.1-20}$$

If the following condition (3.1-21) is satisfied, then the new media stream will be served by at least j disks, such as H'_1,..., H'_j, otherwise, the retrieval request of this media stream will be rejected.

$$K \le \sum_{i=1}^{\min(j)} K^{H'_i} \quad and \quad j \le s_1 \tag{3.1-21}$$

When one retrieval is newly established, $T_{wN'_i}^{H'_i}$ and N_i' $(1 \le i \le j)$ need to be updated by using the following equations:

$$T_{wN'_i}^{H'_i} = T_{wN'_i}^{H'_i} - [\frac{K^{H'_i} M}{R_{dt}} + (K^{H'_i} - 1)\frac{G_{N'_i+1}^{H'_i}}{R_{dt}} + T_{sN'_i+1}^{H'_i}] \tag{3.1-22}$$

and

$$N'_i = N'_i + 1 \tag{3.1-23}$$

When one retrieval is newly finished, $T_{wN'_i}^{H'_i}$ and N'_i $(1 \le i \le j)$ need to be updated by using the following equations:

$$T_{wN'_i}^{H'_i} = T_{wN'_i}^{H'_i} + [\frac{K^{H'_i} M}{R_{dt}} + (K^{H'_i} - 1)\frac{G_{N'_i}^{H'_i}}{R_{dt}} + T_{sN'_i}^{H'_i}] \tag{3.1-24}$$

and

$$N'_i = N'_i - 1 \tag{3.1-25}$$

here, it is assumed that the media stream being served by j disks together is newly finished.

Case 4 (solving type C):

The media stream resource being requested is segmented into several substreams, each of which is stored on the different disk, such as H_1,..., H_k ,..., H_{s_2} $(2 \le s_2 \le S)$, and the given buffer consumption rate is R_c.

Based on the above retrieval conditions, the retrieval request of each substream can be solved by Case 1, but the retrieval of each substream can be assigned higher priority except the first one, in order to serve the real-time retrieval of this media stream continuously.

Case 5 (solving type D):

The media stream resource being requested is declustered across several disks, such as $H_1,...,H_k,...,H_{s_3}$ ($2 \le s_3 \le S$), and the given buffer consumption rate is R_c. Their transfer idle periods are $T_{wN_1}^{H_1},...,T_{wN_k}^{H_k},...,T_{wN_{s_3}}^{H_{s_3}}$ when the head of disk H_k ($1 \le k \le s_3$) is serving the simultaneous retrieval of N_k continuous media streams, respectively.

According to the above retrieval conditions, the following equations derived from equations (2.1-1) and (2.1-4) can be expressed as:

$$B = B_{N_k+1}^{H_k} = T_c \frac{R_c}{s_3} \qquad (1 \le k \le s_3) \tag{3.1-26}$$

$$K = K_{N_k+1}^{H_k} = \frac{B_{N_k+1}^{H_k}}{M} \qquad (1 \le k \le s_3) \tag{3.1-27}$$

$$Result_k = T_{wN_k}^{H_k} - [\frac{B_{N_k+1}^{H_k}}{R_{dt}} + (K_{N_k+1}^{H_k} - 1)\frac{G_{N_k+1}^{H_k}}{R_{dt}} + T_{sN_k+1}^{H_k}] \qquad (1 \le k \le s_3) \tag{3.1-28}$$

If the following condition is satisfied, then the new media stream will be served by s_3 disks, such as $H_1,...,H_k,...,H_{s_3}$, otherwise, the retrieval request of this media stream will be rejected.

$$all\ Result_k \ge 0 \qquad (1 \le k \le s_3) \tag{3.1-29}$$

When one retrieval is newly established, $T_{wN_k}^{H_k}$ and N_k ($1 \le k \le s_3$) need to be updated by using the following equations:

$$T_{wN_k}^{H_k} = T_{wN_k}^{H_k} - [\frac{B_{N_k-1}^{H_k}}{R_{dt}} + (K_{N_k+1}^{H_k} - 1)\frac{G_{N_k-1}^{H_k}}{R_{dt}} + T_{sN_k+1}^{H_k}] \quad (3.1\text{-}30)$$

and

$$N_k = N_k + 1 \quad (3.1\text{-}31)$$

When one retrieval is newly finished, $T_{wN_k}^{H_k}$ and N_k ($1 \le k \le s_3$) need to be updated by using the following equations:

$$T_{wN_k}^{H_k} = T_{wN_k}^{H_k} + [\frac{B_{N_k}^{H_k}}{R_{dt}} + (K_{N_k}^{H_k} - 1)\frac{G_{N_k}^{H_k}}{R_{dt}} + T_{sN_k}^{H_k}] \quad (3.1\text{-}32)$$

and

$$N_k = N_k - 1 \quad (3.1\text{-}33)$$

here, it is assumed that the media stream being served by s_3 disks together is newly finished.

During each buffer empty duration, if the event that at least one retrieval is newly established or finished happens, the seek time of corresponding next retrieving media stream in the round robin sequence need to be updated and transfer idle period need to be little adjusted based on new seek time. The purpose is that the CAC algorithm can perform retrieval admission control more accurately.

When a new retrieval request is accepted, the retrieval start time of this media stream must be controlled so that the buffer empty point of this new media stream will not conflict with the buffer empty point of any other retrieving media streams. Thus, the contention of the storage device can be avoided. Three different cases are discussed as follows:

For the case that the retrieval of a new media stream is served by only one disk (e.g., Case

1 or Case 2), the start time of a new media stream depends on the point in T_c when the retrieval request arrives. It is assumed that once the buffer filling process starts, it has to fill a buffer completely before it can switch to fill one of the other buffers for another retrieving media stream. Therefore, if a request arrival at a point where the time left in the current $T_{wN_k}^{H_k}$ is not enough for the transfer process to fill a buffer completely for this new media stream, then, the retrieval of this new media stream has to be delayed, and wait until the next $T_{wN_k}^{H_k}$ becomes available, which is the same process as [2] for these cases.

For the case that the retrieval of each substream is served by different disk (e.g., Case 4), the starting retrieval of each substream has to wait until the retrieval of previous substream is finished except the retrieval of first one, in order to guarantee the desired QoS requirements for the retrieval of this media stream.

For the case that the retrieval of a new media stream is served by several disks (e.g., Case 3 or Case 5), the retrieval of this new media stream has to be delayed, and waits until the next T_c starts, in order to guarantee enough time for all retrieval from different disks. Once the retrieval buffers are filled, each consuming process in different disks can start to retrieve the data from its corresponding buffer and send it over the network or to the local display device.

3.2 Proposed CAC Algorithm

Through the above analysis, the CAC algorithm is given as follows:

Algorithm input:

The new retrieval request: the requested media stream ID and its buffer consumption rate R_c.

Algorithm output:

Boolean: Accepted or Rejected or Error message.

(1) Beginning of algorithm

(2) Receive the new retrieval request: the requested media stream ID and its buffer consumption rate R_c.

(3) Search the resource database for this media stream being requested and obtain type j (j=A or B or C or D or other) and corresponding disk subset.

(4) If obtained disk subset is empty then return ("Rejected"), otherwise:

(5) switch(type j)

```
{type A: goto case 1;
         Break;
 type B: goto case 2;
         Break;
 type C: goto case 4;
         Break;
 type D: goto case 5;
         Break;
default: return ("Error message");
         Break;
}
```

(6) End of algorithm

By using the above algorithm, the access to disk storage devices can be controlled so that the real-time desired QoS requirements (or the required retrieval rates) of all continuous retrieving media streams can be guaranteed.

3.3 Performance Evaluation for CAC

In a multimedia communication system, the CAC algorithm for the media server with multiple disks is complex. Without simulations, it is very difficult to verify the completeness and consistency of the real-time QoS retrieval requirements. There are several system parameters affecting the total number of end-users who require retrieving services provided by media servers. The main purpose of our simulations is to analyse and evaluate how the number of real-time simultaneous retrieving media streams is affected by system parameters, such as the number of disks

in a media server, disk head seek time, gap size on disks, retrieval buffer size and speed ratio between the disk transfer rate and the buffer consumption rate. From equation (3.1-2), it may be shown that the number of simultaneous retrieving media streams is closely related to these system parameters. In the following sections, the evaluation model and the performance evaluation are presented.

3.3.1 Evaluation Model

The evaluation modelling and simulations for the CAC method have been developed. The Borland C^{++} software package has been used for the modelling construction and the modelling simulation to carry out the performance analysis and evaluation for the CAC method, but our programming is not object-oriented. In order to explore the crucial relationship among the disk characteristics, the buffer size requirements and the number of simultaneous retrieving media streams, four experiments have been performed. Besides the assumptions made in section 3.1, it is also assumed that all retrieval requests from end-users are of video retrieval type with the required rate (30 *FRAMEs* per second), and media streams being requested by end-users are stored in duplicate on all disks in the media server. Because all disks satisfy the condition ($T_c^{H_k} = T_c$) and the buffer consumption rate R_c is 30 *FRAMEs* per second, the same buffer size B for all retrieving media streams can be obtained. The system parameters used in the simulations are shown in the following tables. Some of basic system parameter values we have used for the simulations are the same as in [2] so that the facts can be shown how the media server with multiple disks is able to provide the high performance of retrieval processing.

Parameter	Description	Value and type
SECTOR	Disk sector size	512 *bytes* (constant)
FRAME	Video frame size	600 *SECTORs* (constant)
R_c	Buffer consumption rate	30 *FRAMEs*/s (constant)
M	Media block size	600 *SECTORs* (constant)
RATIO	Speed ratio R_{dt}/R_c	20 (constant)
B	Retrieval buffer size	3600 *SECTORS* (constant)
T_s	Average seek time	9 ms (constant)
T_c	Buffer empty duration	$T_c=B/R_c$ (constant)
R_{dt}	disk data transfer rate	$20R_c$ (constant)
S	maximum number of disk storage devices in a media server	1, 2, 3, 4, or 5 (variable)
$T_{wN_k}^{H_k}$ ($1 \le k \le S$)	The transfer idle period when the retrieval of N_k media streams is being served by disk H_k	0 - T_c (variable)
$T_{si}^{H_k}$ ($1 \le k \le S$)	The head seek time of disk H_k switches from i-1th retrieving stream to ith retrieving stream	Poisson random variable having mean value T_s ms (variable)
$G_i^{H_k}$ ($1 \le k \le S$)	The gap size of the ith media stream on disk H_k	Poisson random variable having mean value (500, ..., 5000) *SECTORS* (variable)
N_k ($1 \le k \le S$)	Number of retrieving media streams being served by disk H_k	The values from simulation results (variable)

Table 3.3.1-1: The System Parameters for Gap Size Simulations

Parameter	Description	Value and type
SECTOR	Disk sector size	512 *bytes* (constant)
FRAME	Video frame size	600 *SECTORs* (constant)
R_c	Buffer consumption rate	30 *FRAMEs*/s (constant)
M	Media block size	600 *SECTORs* (constant)
RATIO	Speed ratio R_{dt}/R_c	20 (constant)
G	Average gap size	900 *SECTORs* (constant)
B	Retrieval buffer size	3600 *SECTORS* (constant)
R_{dt}	disk data transfer rate	$20R_c$ (constant)
T_c	Buffer empty duration	$T_c=B/R_c$ (constant)
S	maximum number of disk storage devices in a media server	1, 2, 3, 4, or 5 (variable)
$T_{wN_k}^{H_k}$ (1≤ k ≤ S)	The transfer idle period when the retrieval of N_k media streams is being served by disk H_k	0 - T_c (variable)
$T_{si}^{H_k}$ (1≤ k ≤ S)	The head seek time of disk H_k switches from *i*-1th retrieving stream to *i*th retrieving stream	Poisson random variable having mean value (0, ..., 20) ms (variable)
$G_i^{H_k}$ (1≤ k ≤ S)	The gap size of the ith media stream on disk H_k	Poisson random variable having mean value *G* *SECTORS* (variable)
N_k (1≤ k ≤ S)	Number of retrieving media streams being served by disk H_k	The values from simulation results (variable)

Table 3.3.1-2: The System Parameters for Seek Time Simulations

Parameter	Description	Value and type
SECTOR	Disk sector size	512 *bytes* (constant)
FRAME	Video frame size	600 *SECTORs* (constant)
R_c	Buffer consumption rate	30 *FRAMEs*/s (constant)
M	Media block size	600 *SECTORs* (constant)
RATIO	Speed ratio R_{dt}/R_c	20 (constant)
G	Average gap size	900 *SECTORs* (constant)
T_s	Average seek time	9 ms (constant)
R_{dt}	disk data transfer rate	$20R_c$ (constant)
B	Retrieval buffer size	600, ..., 6000 *SECTORS* (variable)
T_c	Buffer empty duration	$600/R_c$, ..., $6000/R_c$ $T_c=B/R_c$ (variable)
S	maximum number of disk storage devices in a media server	1, 2, 3, 4, or 5 (variable)
$T_{wN_k}^{H_k}$ ($1 \le k \le S$)	The transfer idle period when the retrieval of N_k media streams is being served by disk H_k	0 - T_c (variable)
$T_{si}^{H_k}$ ($1 \le k \le S$)	The head seek time of disk H_k switches from *i*-1th retrieving stream to *i*th retrieving stream	Poisson random variable having mean value T_s ms (variable)
$G_i^{H_k}$ ($1 \le k \le S$)	The gap size of the ith media stream on disk H_k	Poisson random variable having mean value *G* *SECTORS* (variable)
N_k ($1 \le k \le S$)	Number of retrieving media streams being served by disk H_k	The values from simulation results (variable)

Table 3.3.1-3: The System Parameters for Buffer Size Simulations

Parameter	Description	Value and type
SECTOR	Disk sector size	512 *bytes* (constant)
FRAME	Video frame size	600 *SECTORs* (constant)
R_c	Buffer consumption rate	30 *FRAMEs*/s (constant)
M	Media block size	600 *SECTORs* (constant)
G	Average gap size	900 *SECTORs* (constant)
T_s	Average seek time	9 ms (constant)
B	Retrieval buffer size	3600 *SECTORS* (constant)
T_c	Buffer empty duration	$T_c = B/R_c$ (constant)
RATIO	Speed ratio R_{dt}/R_c	5, ..., 55 (variable)
R_{dt}	disk data transfer rate	$(5, ..., 55)R_c$ (variable)
S	maximum number of disk storage devices in a media server	1, 2, 3, 4, or 5 (variable)
$T_{wN_k}^{H_k}$ $(1 \le k \le S)$	The transfer idle period when the retrieval of N_k media streams is being served by disk H_k	0 - T_c (variable)
$T_{si}^{H_k}$ $(1 \le k \le S)$	The head seek time of disk H_k switches from *i*-1th retrieving stream to *i*th retrieving stream	Poisson random variable having mean value T_s ms (variable)
$G_i^{H_k}$ $(1 \le k \le S)$	The gap size of the ith media stream on disk H_k	Poisson random variable having mean value *G* *SECTORS* (variable)
N_k $(1 \le k \le S)$	Number of retrieving media streams being served by disk H_k	The values from simulation results (variable)

Table 3.3.1-4: The System Parameters for Speed Ratio Simulations

Random numbers are a necessary basic ingredient in the simulations. A sequence of random numbers, R_1, R_2, . . . , must have two important statistical properties, uniformity and independence. In this thesis, the method of random number generation [16], which meets the simulation requirements, have been applied. Also, for making independent replications during the simulations, different seeds for random number generation have been chosen.

The gap size of a media stream is determined by its real-time condition [2] and its media stream storage pattern on the disks, so different streams may have different gap size. The seek time is determined by the current position of the disk head and the disk location of the required media block. They are only two Poisson random variables among the system parameters. In the simulations, the gap size $G_i^{H_k}$ of the *i*th retrieving media stream ($1 \le i \le N_k$) on disk H_k ($1 \le k \le$ S) and the head seek time $T_{si}^{H_k}$ of disk H_k ($1 \le k \le$ S) are modelled as independent Poisson random variables having a certain mean value G representing the average gap size of the retrieving media streams and T_s representing the average head seek time, respectively [2]. We choose Poisson distribution for gap size and seek time due to its characteristics similar to gap size's and seek time's. Since Poisson distribution for gap size and seek time has been specified, the ways must be sought to generate samples from this distribution to be used as input to the simulation model. Because Borland C^{++} software package does not have random-variate generation libraries, we must construct random-variate generation subroutine for Poisson distribution. In order to generate random variates for Poisson distribution, the acceptance-rejection technique is used. The procedure for generating a Poisson random variate, N, is given by the following steps (for details, see [16]):

Step 1: Set $n = 0$, $P = 1$.

Step 2: Generate an uniform random number R_{n+1} [0, 1] and replace P by PR_{n+1}.

Step 3: If $P < e^{-\alpha}$, then accept $N = n$. Otherwise, reject the current n, increase n by one, and return to step 2.

where α is the mean of Poisson distribution, such as G and T_s, and N is Poisson random variate, Which can be gap size or seek time in the simulations.

3.3.2 Performance Evaluation

All model and simulation programming have been done by using Borland C^{++} software package. Four sets of experiments have been performed, namely, the effect of gap size, the effect of disk head seek time, the effect of retrieval buffer size, and the effect of speed ratio. For each simulation experiment, 1000 simulation times have been performed as well as maximum, average, and minimum number of retrieving media streams supported by the media server have been obtained. The performance evaluation of simulation results will be focused on average number of retrieving media streams. They are detailed as follows:

The effect of gap size

The disk head rotational latency is mainly introduced by the gap size between two successive media blocks of a media stream. Through the simulations, the fact can be shown how the gap size and the total number of disks affect the average number of simultaneous retrieving media streams. As discussed in the above sections, the number of disk storage devices is closely related to the number of simultaneous retrieving media streams the media server can support. In fact, the number of retrieving media streams is considerably increased as the number of disks is increased. This can be verified by the following simulation results.

As shown from Figure 3.3.2-1 to Figure 3.3.2-5, for the given speed ratio ($RATIO = 20$), the retrieval buffer size of the media stream (B = 3600 *SECTORs*), the average seek time ($T_s = 9$ ms), and the average gap size (G = from 500 to 5000 *SECTORs*), the more number of disks, the more number of simultaneous retrieving media streams, which means that the media server can support more retrieving media streams simultaneously. 15 retrieving media streams can be served by the disks at the point in the Figure 3.3.2-5 where the total number of disks S is 5 and the average gap size G is 3000 *SECTORs*, although one disk can support only 2 retrieving media streams alone in Figure 3.3.2-1. This is due to the relatively large remaining transfer idle period after each disk supports 2 retrieving media streams, and Case 2 and case 3 can distribute the workload of the applications evenly across the disks so as to maximize the system processing capabilities.

On the other hand, the larger the average gap size is, the fewer number of simultaneous retrieving media streams will be for the given any number of disks. The gap size effect becomes more significant when the gap size reaches 5000 SECTORs. So, the large gap size will reduce the average number of simultaneous retrieving media streams the media server can support.

This results can be explained by equation (3.1-12). The equation (3.1-12) is given again in the following:

$$Result_k = T_{wN_k}^{H_k} - [\frac{B_{N_k+1}^{H_k}}{R_{dt}} + (K_{N_k+1}^{H_k}-1)\frac{G_{N_k+1}^{H_k}}{R_{dt}} + T_{si}^{H_k}] \qquad (1 \le k \le S)$$

$$= T_{wN_k}^{H_k} - \frac{B_{N_k+1}^{H_k}}{R_{dt}} - \frac{K_{N_k+1}^{H_k}-1}{R_{dt}} G_{N_k+1}^{H_k} - T_{si}^{H_k} \qquad (1 \le k \le S)$$

then we can see that $G_{N_k+1}^{H_k}$ is in the negative portion, because $K_{N_k+1}^{H_k}$ is equal to or more than 1 and R_{dt} is more than 0. The larger new gap size is given, the less $Result_k$ will be, which means that the retrieval request of the new media stream will be more likely rejected.

The effect of disk head seek time

From Figure 3.3.2-6 to Figure 3.3.2-10, it is shown that average number of simultaneous retrieving media streams a media server can support is increased when the disk head seek time decreases and the total number of disks increases.

If the figures meets the following conditions, the given speed ratio ($RATIO$ = 20), the retrieval buffer size (B = 3600 *SECTORs*), and the average gap size (G = 900 *SECTORs*), the average number of simultaneous retrieving media streams is shown in these figures for any given number of disks ($1 \le S \le 5$). For example, when T_s is 8 ms, the media server with three disks can support 18 simultaneous retrieving media streams in Figure 3.3.2-8, and when T_s drops to 4 ms, 21 simultaneous retrieving media streams can be supported.

Further, the relationship between the total number of disks and simultaneous retrieving media streams being supported can be shown. Extra 4 retrieving media streams are being served by all disks at least two points where the total number of disks S is 5 and average seek time T_s is 2 ms or 16 ms in Figure 3.3.2-10. The reason is that the remaining transfer idle period of each disk is relatively large at these points, so, five disks can support more extra 4 media streams.

This scenario can be also explained by the equation (3.1-12). The equation (3.1-12) is given again in the following:

$$Result_k = T_{wN_k}^{H_k} - [\frac{B_{N_k+1}^{H_k}}{R_{dt}} + (K_{N_k+1}^{H_k}-1)\frac{G_{N_k+1}^{H_k}}{R_{dt}} + T_{si}^{H_k}] \qquad (1 \le k \le S)$$

$$= T_{wN_k}^{H_k} - \frac{B_{N_k+1}^{H_k}}{R_{dt}} - (K_{N_k+1}^{H_k}-1)\frac{G_{N_k+1}^{H_k}}{R_{dt}} - T_{si}^{H_k} \qquad (1 \le k \le S)$$

We can see that $T_{si}^{H_k}$ is in the negative portion. The larger $T_{si}^{H_k}$ is given, the less $Result_k$ will be, which means that the media server can support fewer media streams.

The effect of retrieval buffer size

If the retrieval buffer size is increased, the buffer empty duration $T_c^{H_k}$, the total time needed to fill out a retrieval buffer $T_{fi}^{H_k}$ and the rotational latency $T_{ri}^{H_k}$ are increased. The overall effects cannot be seen clearly from equation (3.1-12) or other equations.

The simulation results are shown from Figure 3.3.2-11 to Figure 3.3.2-15. For the given system parameters on the top of figures, some facts can be observed that when the retrieval buffer size is between 4200 and 6000 *SECTORs*, with the specified average gap size (G = 900 *SECTORs*), speed ratio (*RATIO* = 20), and the number of disks (S = 1), at most 6 retrieving media streams can be supported, no matter how large the retrieval buffer size is. On the other hand, although it has not

enough transfer idle period to support any more media streams for each disk, the remaining transfer idle period is accumulated while the size of retrieval buffer is increased, which can make five disks be able to support the retrieval of 32 media streams (2 media streams are extra) at the point where the buffer size is 6000 *SECTORs*.

So, when the total disk latency, the speed ratio and the total number of disk storage devices are known in advance, these simulation results can help us to determine the optimal size of the retrieval buffer, which is when the transfer idle period $T_{wN_k}^{H_k}$ is equal to or is slightly large than 0. In order to find the optimal size of retrieval buffer, the mathematical programming equations need to be developed and solved, which is beyond our research scope.

The effect of speed ratio

From Figure 3.3.2-16 to Figure 3.3.2-20, the total number of disks ($1 \le S \le 5$), the retrieval buffer ($B = 3600$ *SECTORs*), the average seek time ($T_s = 9$ ms), and the average gap size ($G = 900$ *SECTORs*) are given in the simulations. With the same number of disks, seek time and gap size, increasing the speed ratio *RATIO* can increase the average number of simultaneous retrieving media streams significantly.

These results can be explained by the equation (3.1-12):

$$Result_k = T_{wN_k}^{H_k} - [\frac{B_{N_k+1}^{H_k}}{R_{dt}} + (K_{N_k+1}^{H_k}-1)\frac{G_{N_k+1}^{H_k}}{R_{dt}} + T_{si}^{H_k}] \qquad (1 \le k \le S)$$

$$= T_{wN_k}^{H_k} - \frac{B_{N_k+1}^{H_k} + (K_{N_k+1}^{H_k}-1)G_{N_k+1}^{H_k}}{R_{dt}} - T_{si}^{H_k} \qquad (1 \le k \le S)$$

We can see that $1/R_{dt}$ is in the negative portion. The larger speed ratio ($RATIO = R_{dt}/R_c$) is given (here, R_c is constant and equal to 30 *FRAMEs* per second), the less $1/R_{dt}$ or the larger $Result_k$ will be, which means that the media server can accept more new retrieval requests.

3.4 Summary

In this chapter, the detailed analysis of the media server with multiple disks supporting the simultaneous retrieval of continuous media streams with different rates is given. Then, the CAC algorithm is presented to control the acceptance of new retrieval request for the given four different types. At the end of this chapter, the simulation results show with the relationships between the maximum, average and minimum number of simultaneous retrieving media streams and system parameters, such as the number of disks, the disk head seek time, the gap size, the retrieval buffer size, and the speed ratio. Moreover, if any disk has already reached the upper limit and cannot accept new retrieval request alone, it is still possible that several disks are working together to accept the new retrieval request with the guaranteed QoS retrieval requirements or the required retrieval rates. So, the CAC algorithm can provide very high efficient admission control and guarantee the real-time continuous retrieval of delay sensitive media streams.

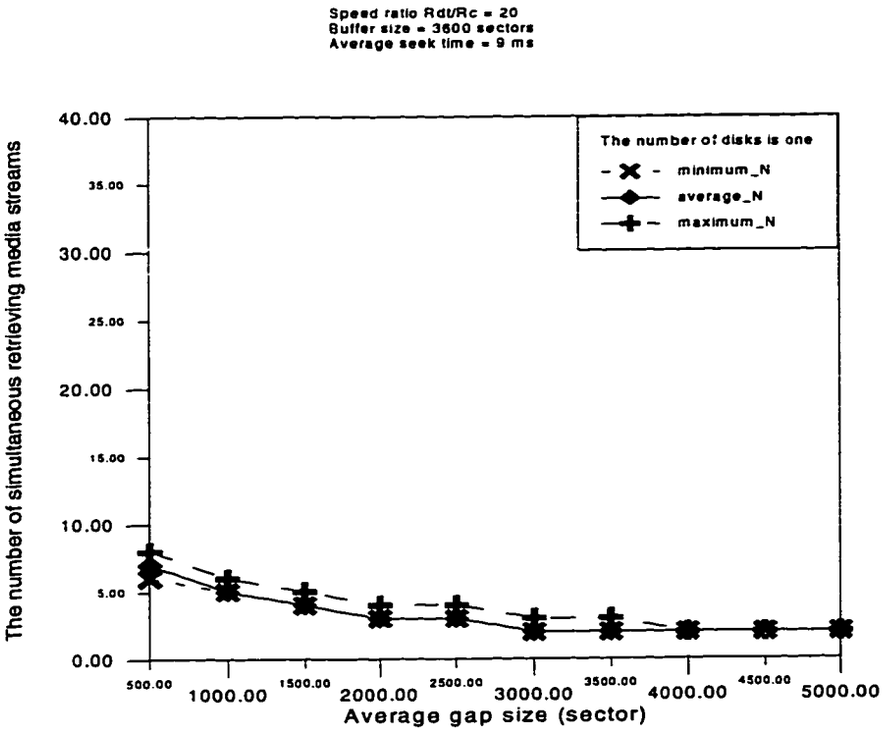


Figure 3.3.2-1 Simulation Results on Gap Size Effect (S=1)

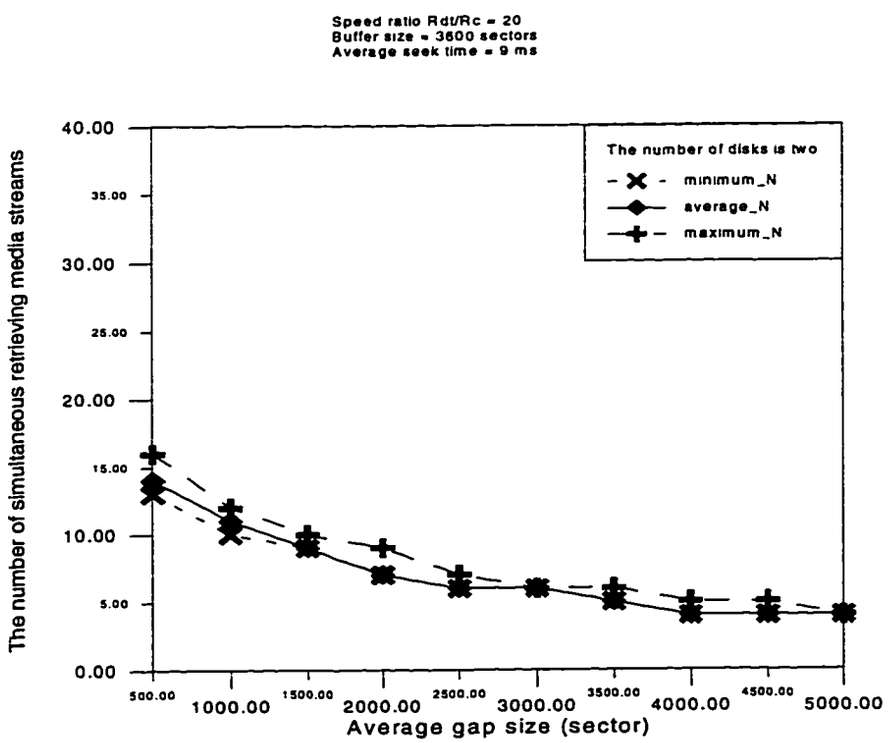


Figure 3.3.2-2 Simulation Results on Gap Size Effect (S=2)

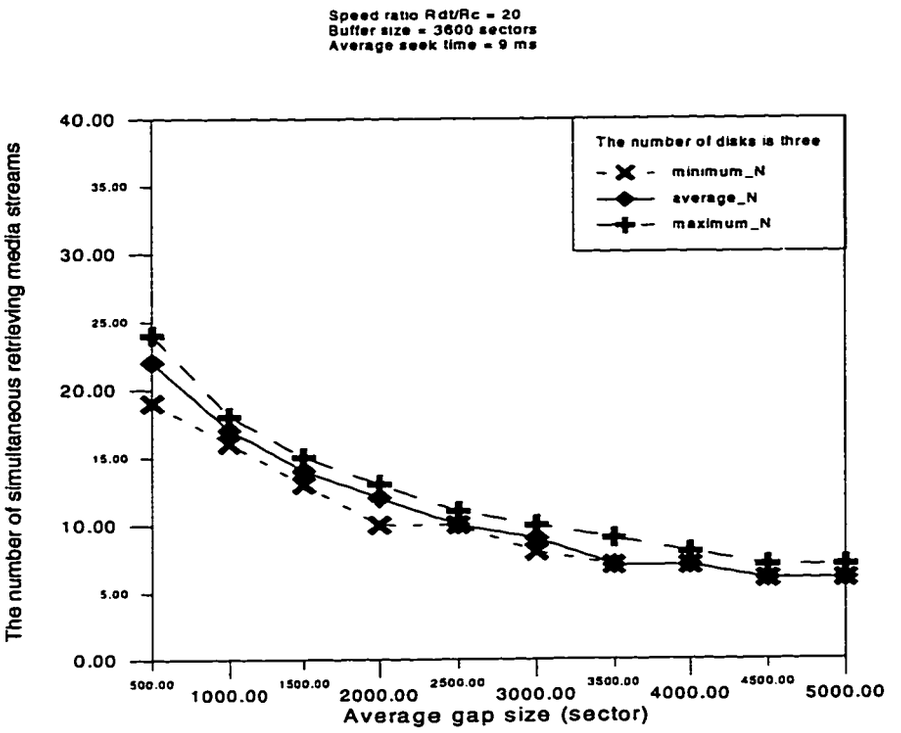


Figure 3.3.2-3 Simulation Results on Gap Size Effect (S=3)

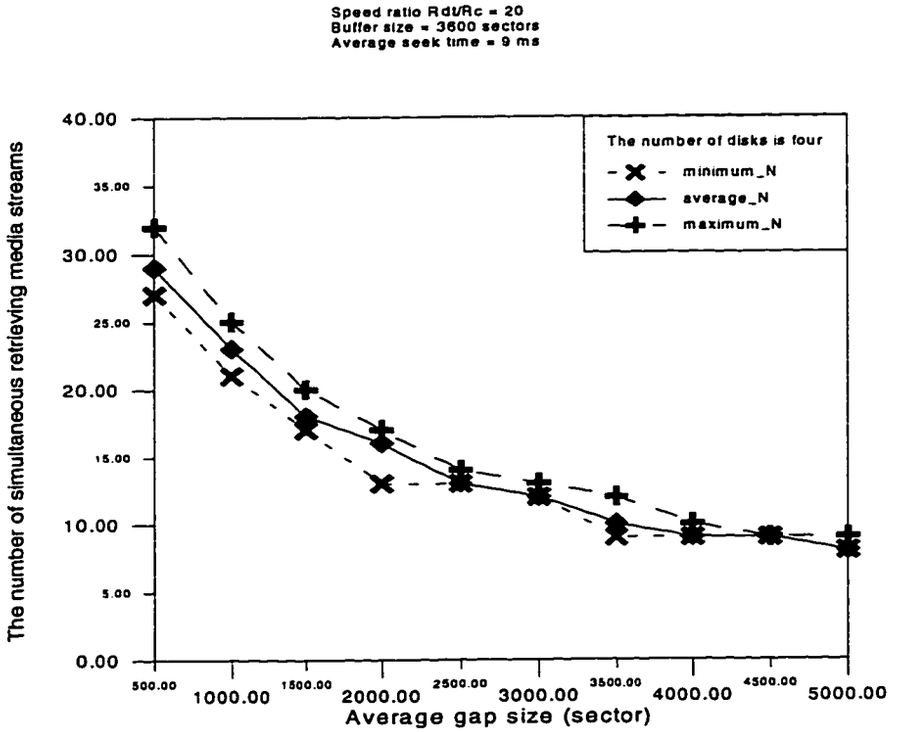


Figure 3.3.2-4 Simulation Results on Gap Size Effect (S=4)

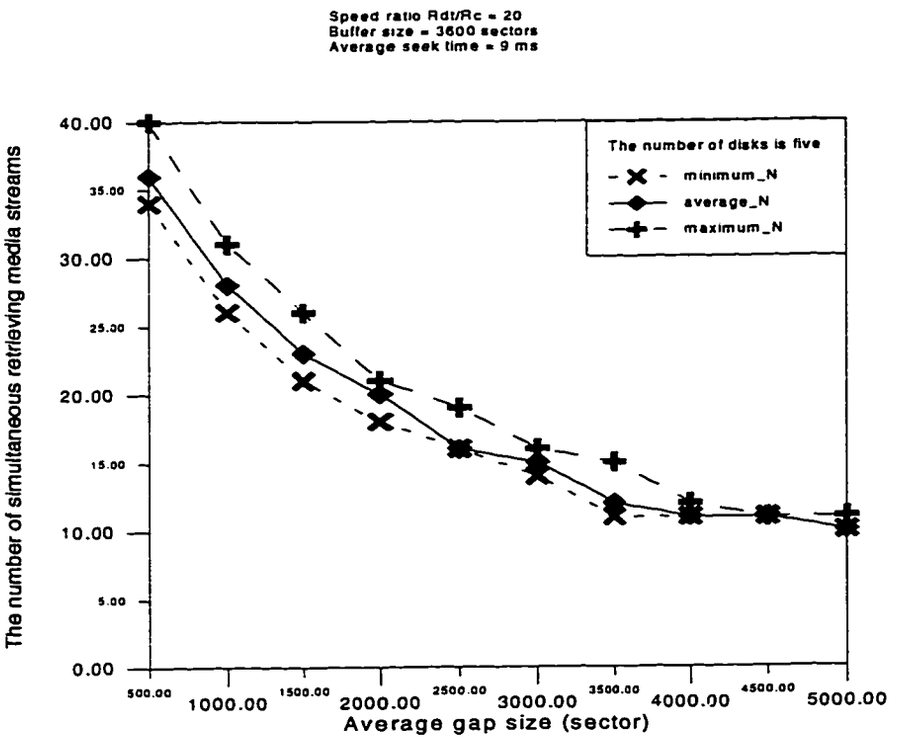


Figure 3.3.2-5 Simulation Results on Gap Size Effect (S=5)

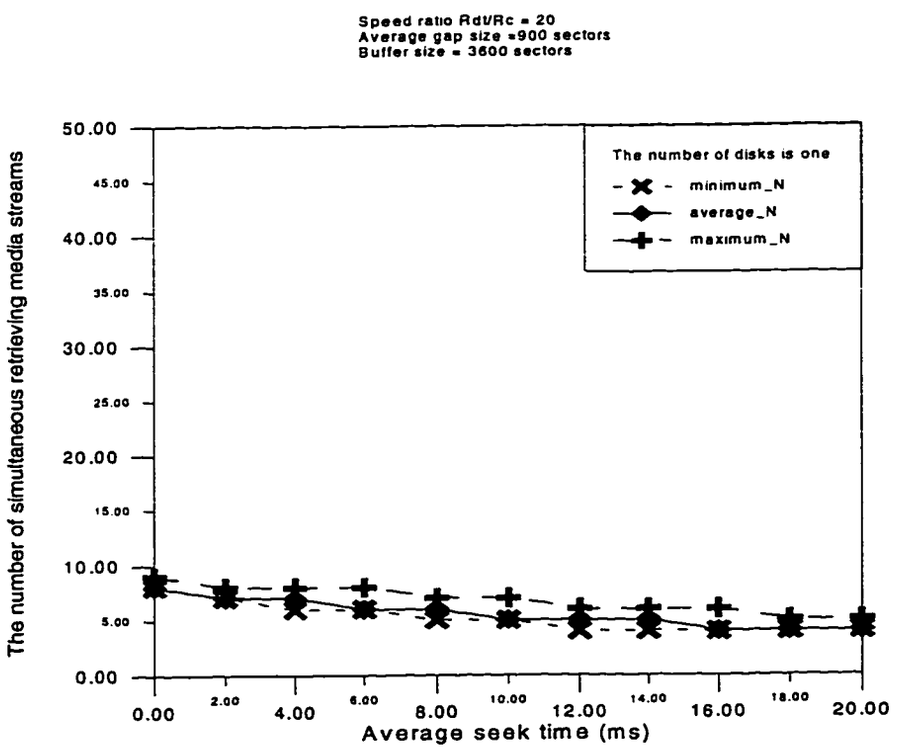


Figure 3.3.2-6 Simulation Results on Seek Time Effect (S=1)

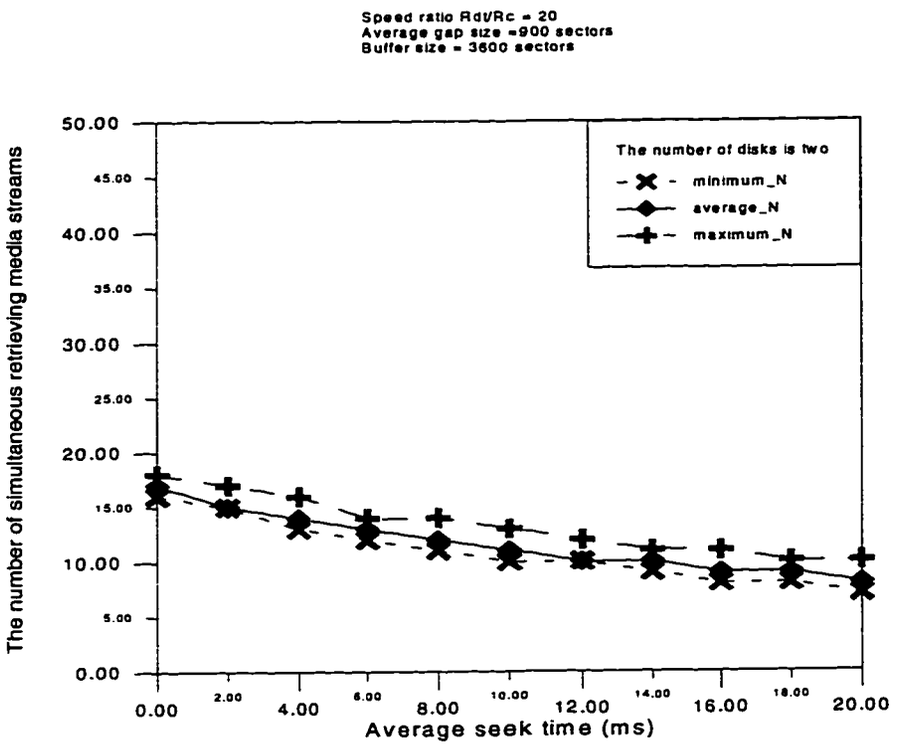


Figure 3.3.2-7 Simulation Results on Seek Time Effect (S=2)

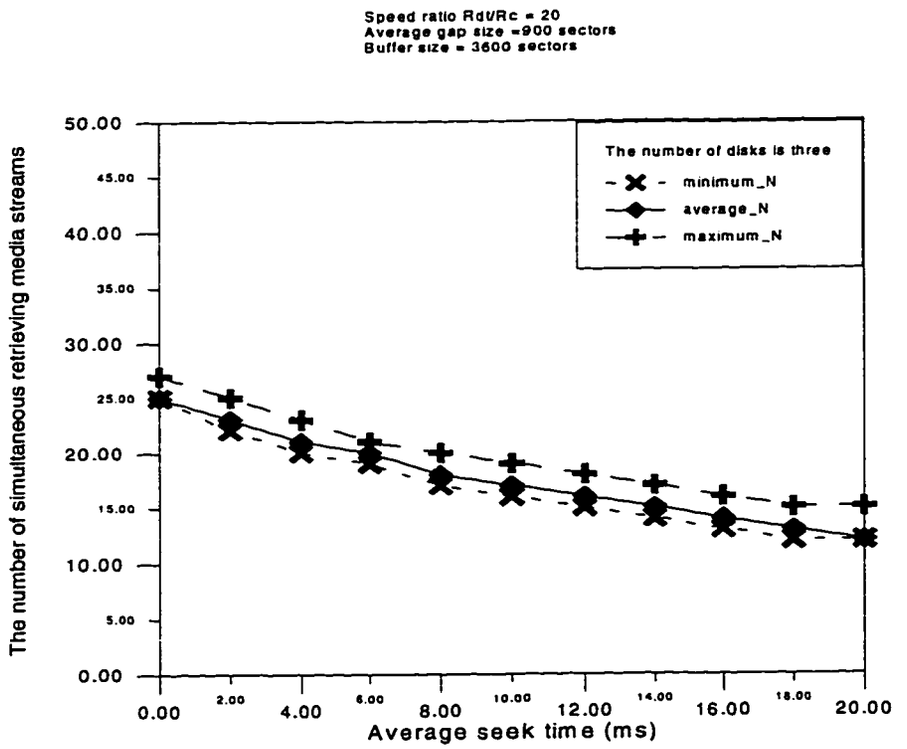


Figure 3.3.2-8 Simulation Results on Seek Time Effect (S=3)

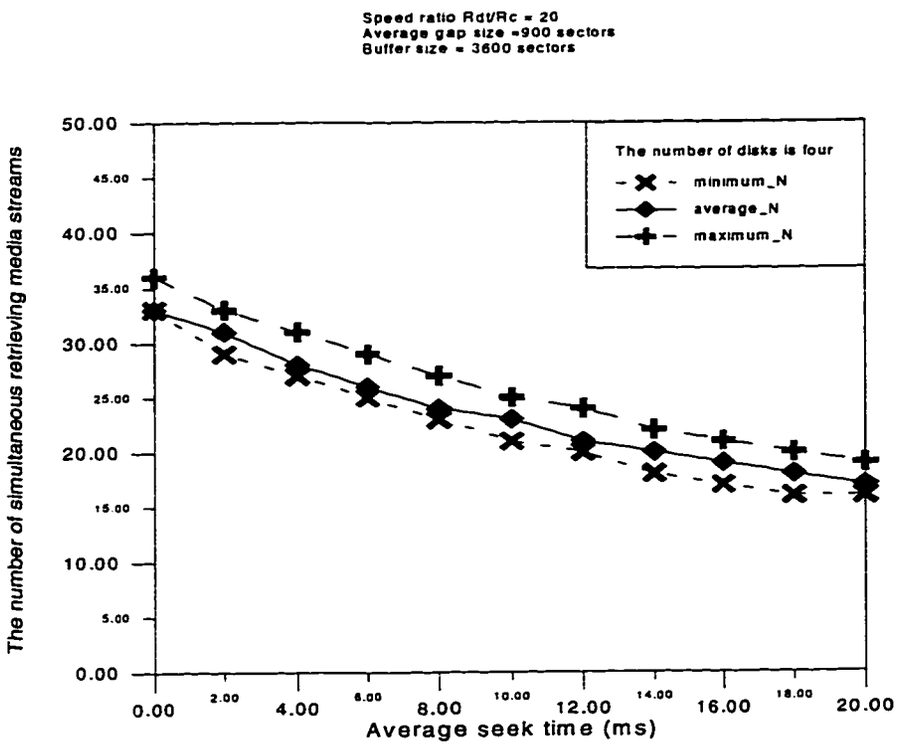


Figure 3.3.2-9 Simulation Results on Seek Time Effect (S=4)

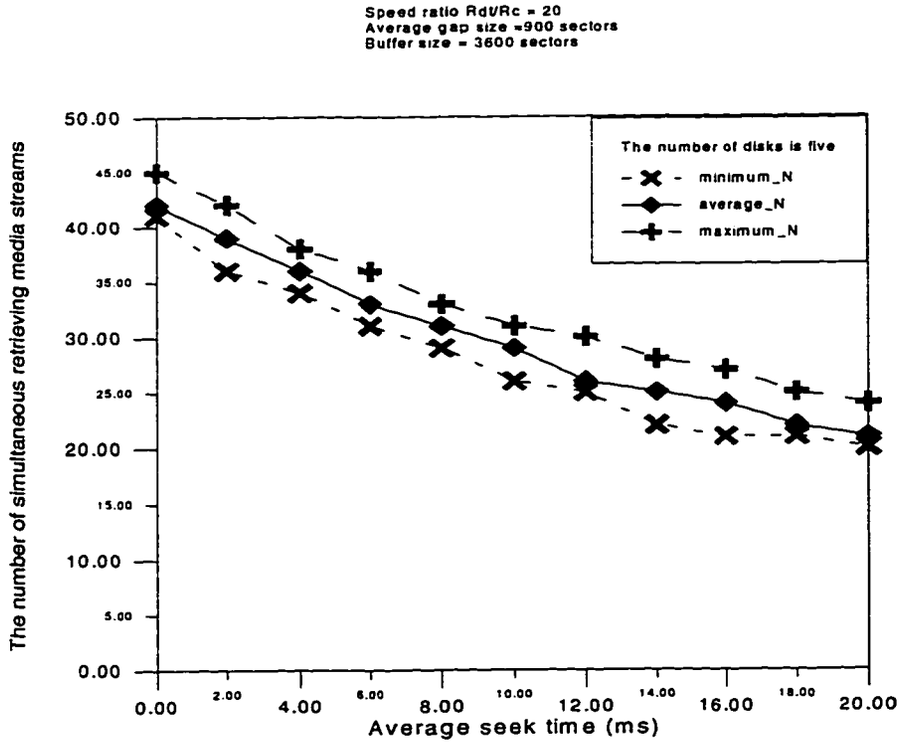


Figure 3.3.2-10 Simulation Results on Seek Time Effect (S=5)

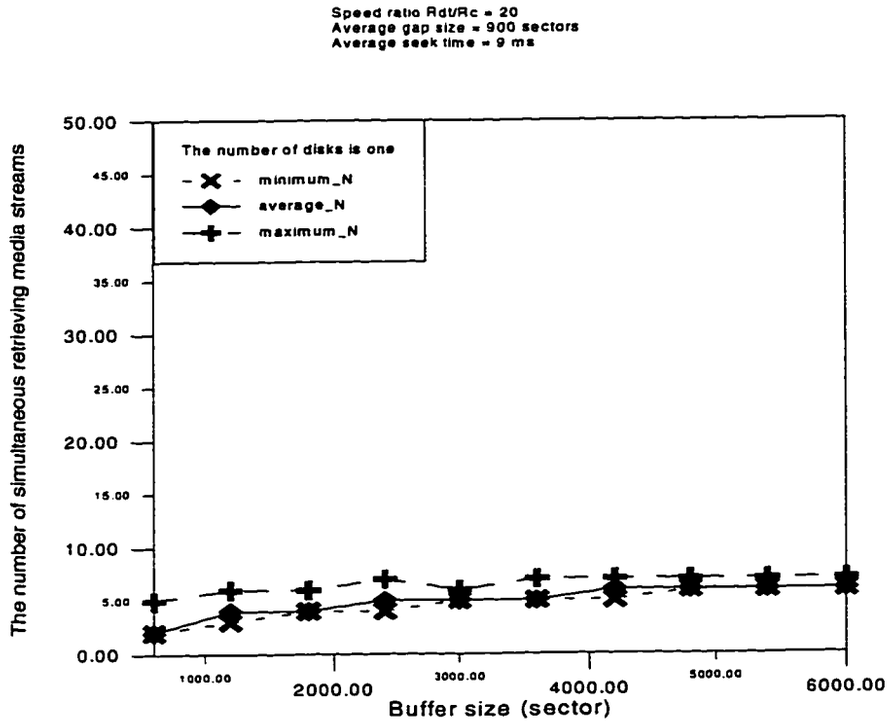


Figure 3.3.2-11 Simulation Results on Buffer Size Effect (S=1)

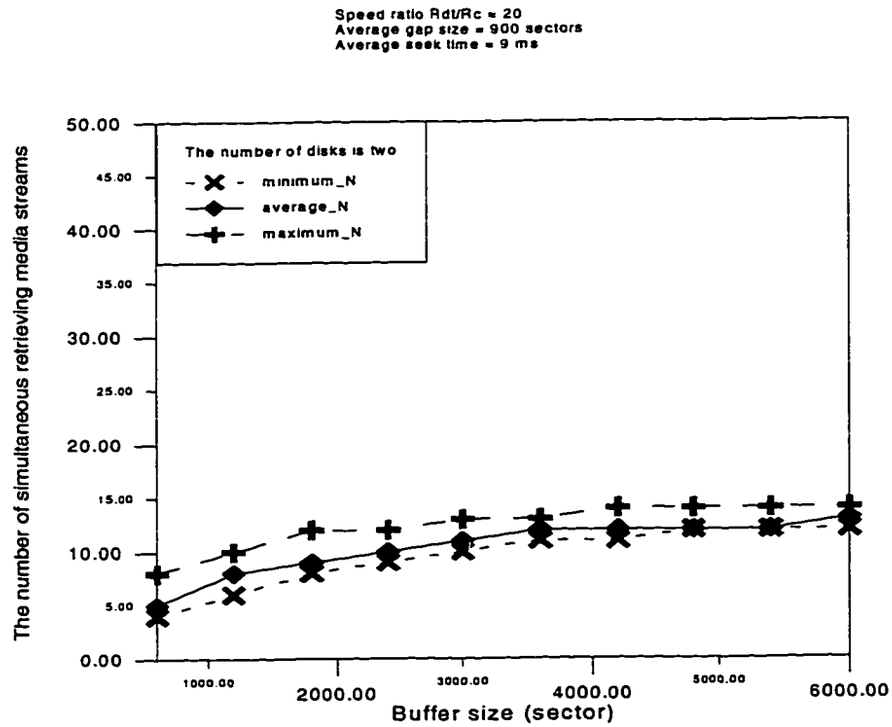


Figure 3.3.2-12 Simulation Results on Buffer Size Effect (S=2)

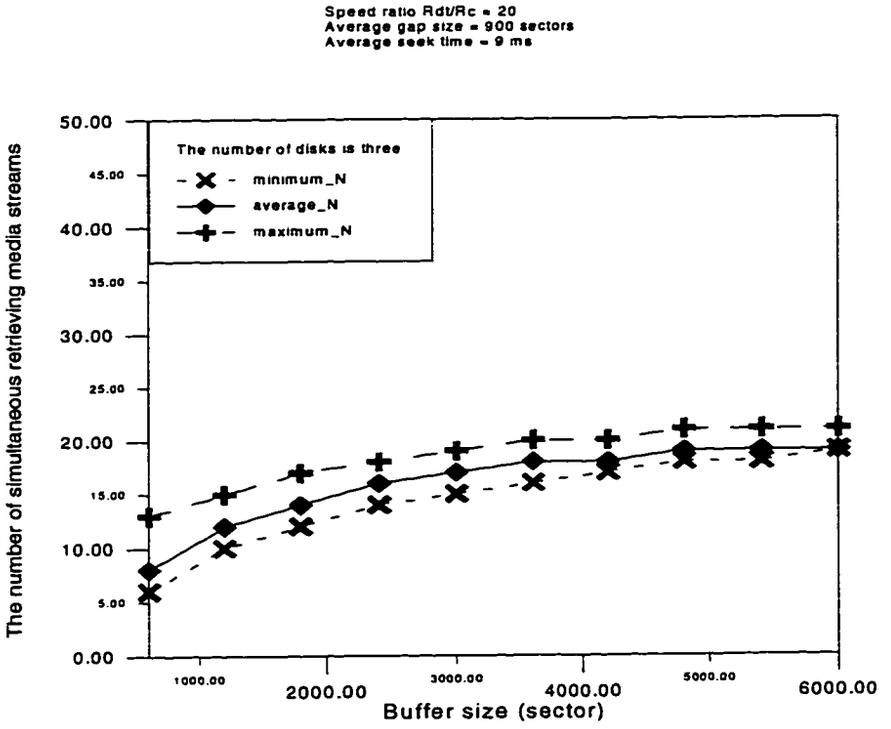


Figure 3.3.2-13 Simulation Results on Buffer Size Effect (S=3)

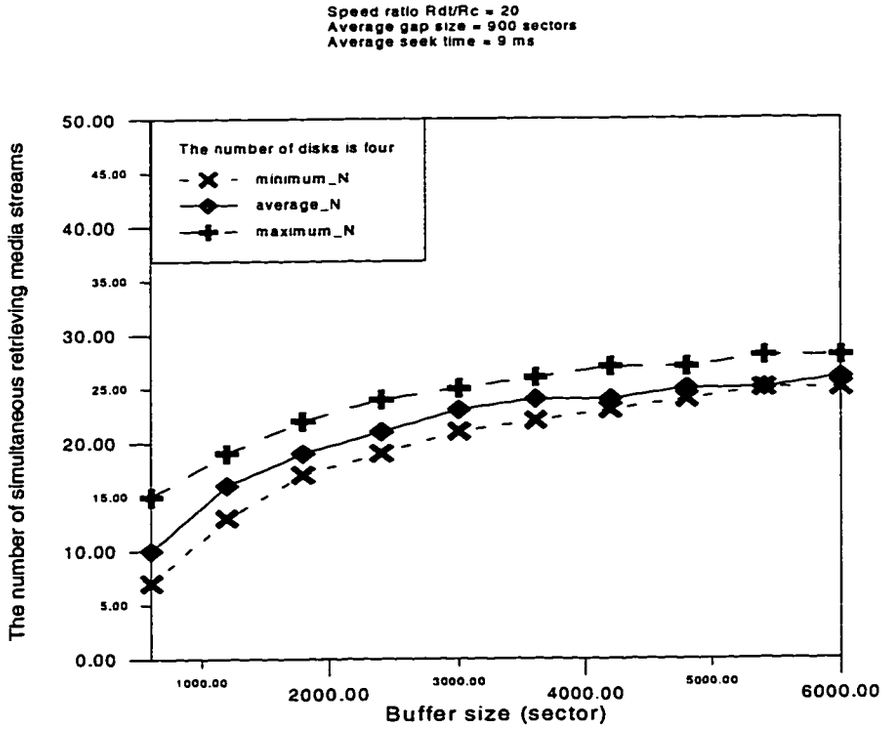


Figure 3.3.2-14 Simulation Results on Buffer Size Effect (S=4)

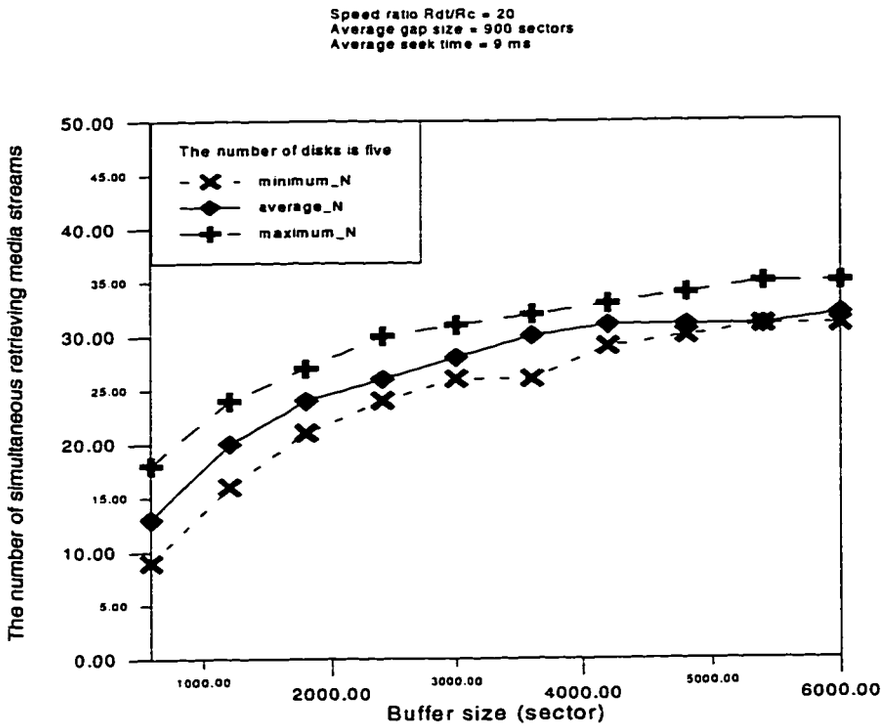


Figure 3.3.2-15 Simulation Results on Buffer Size Effect (S=5)

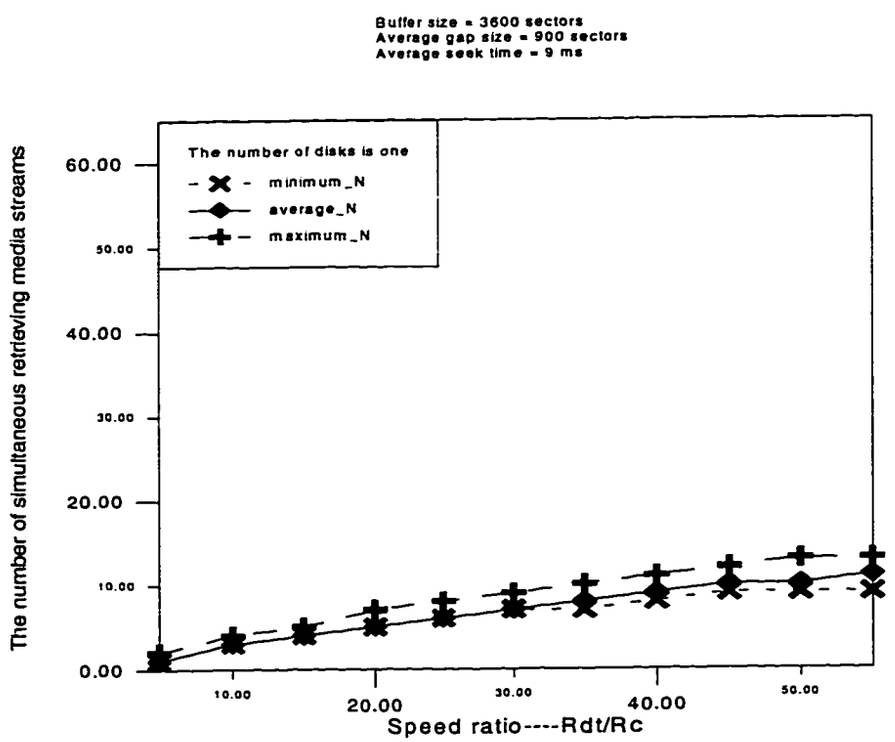


Figure 3.3.2-16 Simulation Results on Speed Ratio Effect (S=1)

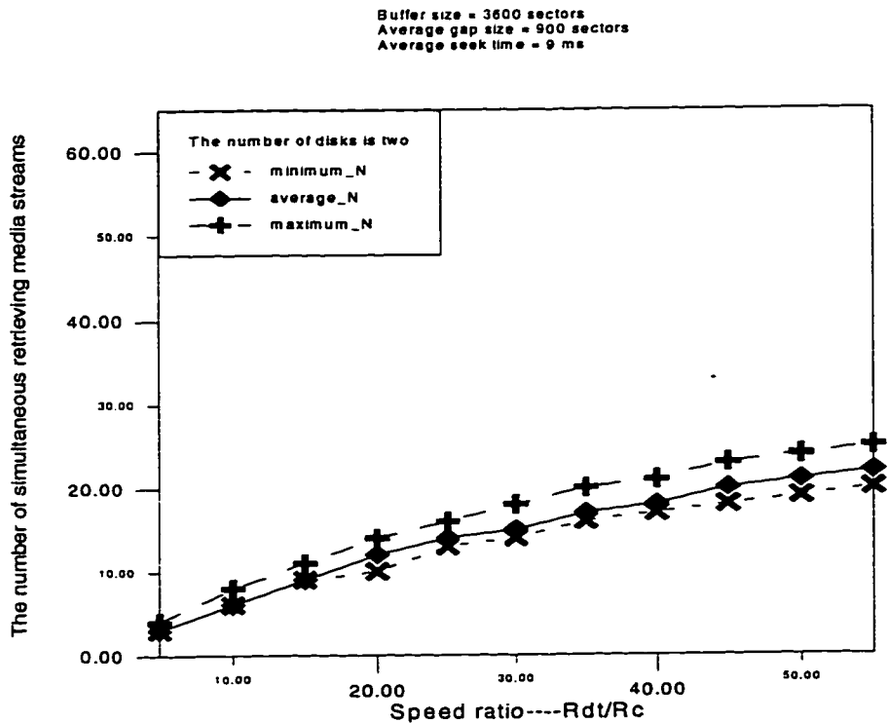


Figure 3.3.2-17 Simulation Results on Speed Ratio Effect (S=2)

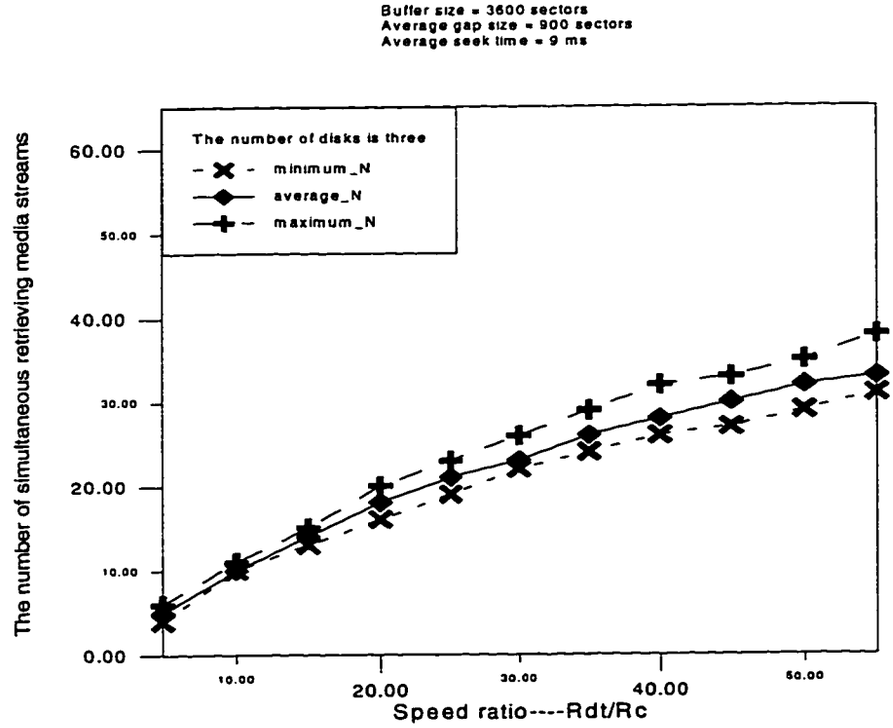


Figure 3.3.2-18 Simulation Results on Speed Ratio Effect (S=3)

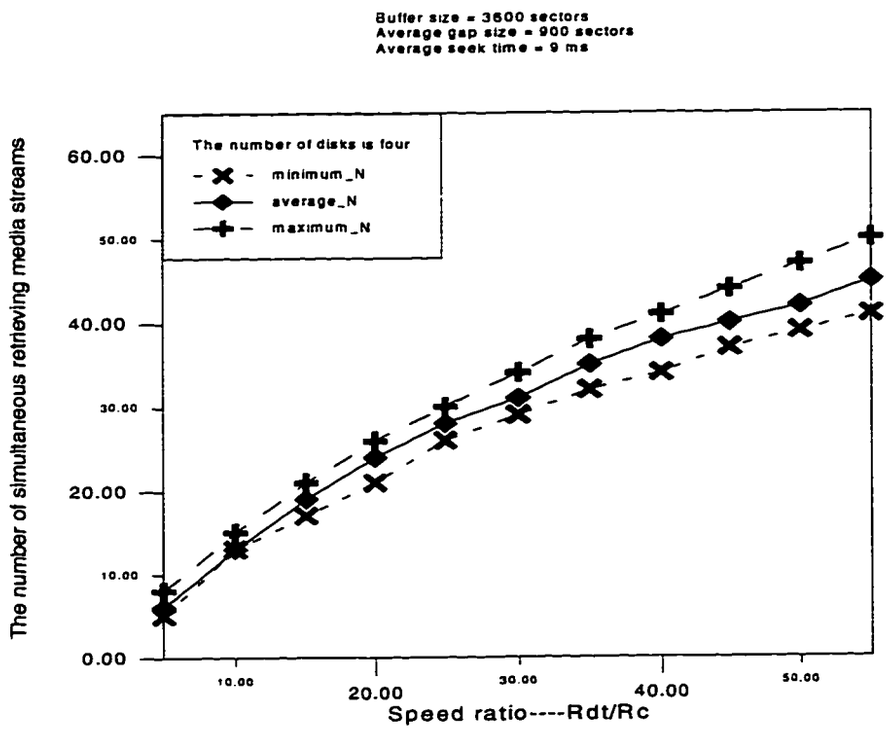


Figure 3.3.2-19 Simulation Results on Speed Ratio Effect (S=4)

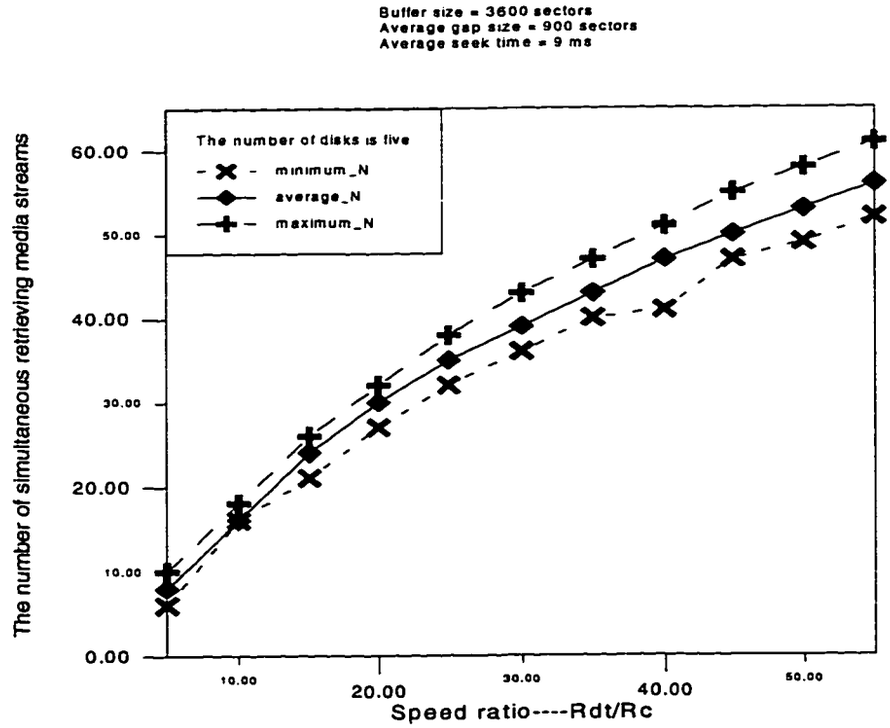


Figure 3.3.2-20 Simulation Results on Speed Ratio Effect (S=5)

Chapter 4

Connection Admission Control for ATM network

4.1 Proposed CAC Method

4.1.1 Preliminaries

Queues are common in an ATM switching node and an ATM multiplexer. Since the preassigned time slot concept disappears in ATM networks, contention problems arise if two or more cells compete for the same time slot. This can be solved by temporarily queuing the arriving cells before sending them out and its primary function is to effectively reduce cell loss rates in ATM networks. From the view point of current Very Large Scale Integration (VLSI) technologies, three or four hundred cells buffer per port is the upper bound to build in the actual switching system [14]. Cell loss rate and average queuing delay are two main important performance objectives provided by the network to connections, which is closely related to the buffer sizes in an ATM switching node or an ATM multiplexer. So, the CAC methods for ATM networks should consider the effects of different buffer sizes. A general ATM node model, which is shown in Figure 4.1.1-1, has been studied in this thesis in order to analyse and evaluate the performance of the (output-buffered) multiplexer or the (non-blocking, output-buffered) switching node to input traffic sources from the multiplexed connections.

First, let $P(B_i^k)$ be the probability that i cells are in the buffer on the kth T_1 period, which can be derived from the probability $G(n)$ that n cells are transferred from multiplexed connections (or entering the buffer), and M_{T_1} that the maximum number of cells transferred by the multiplexed line (or going out the buffer) during T_1 period. The value of T_1 is equal to the inverse of peak cell transmission speed for the connection whose peak cell transmission speed is the maximum value

among the low speed connections [14]. When one connection is newly added to or torn down from the current existing multiplexed connections, the probability function *G(n)* needs to be calculated. The probability function *G(n)* is given again by the following equation (4.1.1-1):

$$G(n)=\sum_{k=0}^{n} P(k)F(n-k) \qquad (4.1.1\text{-}1)$$

where *P(k)* is the probability that *k* cells are transferred from low speed connections during T_1, *F(n-k)* is the probability that n-*k* cells are transferred from high speed connections during T_1 (for details of *P(k)* and *F(k)*, see [14] or the following section 4.1.2).

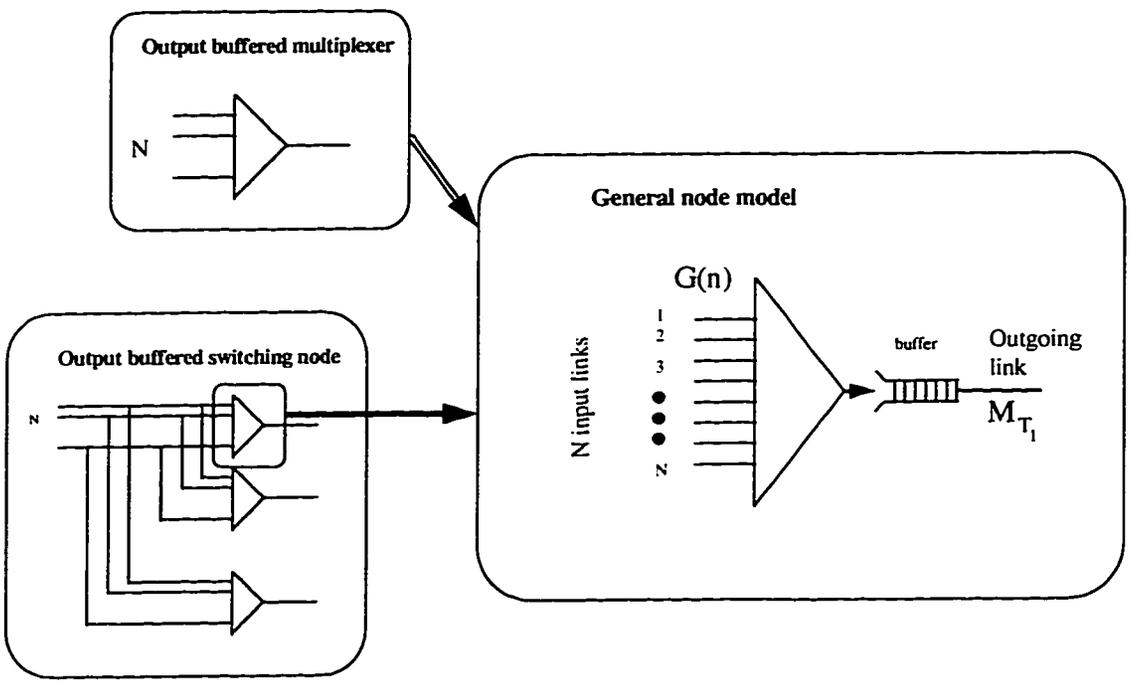


Figure 4.1.1-1 General ATM Node Model

In the following, it is assumed that the number or the types of multiplexed connections are not be changed since the k'th T_1 period ($k > k'$). By *Law of Total Probability Theorem 2.4.2* in [27], the probability $P(B_i^k)$ can be given by equation (4.1.1-2):

$$P(B_i^k) = \sum_{j=0}^{S} P(B_j^{k-1}) P(B_i^k | B_j^{k-1}) \qquad (0 \le i \le S) \tag{4.1.1-2}$$

and

$$\sum_{i=0}^{S} P(B_i^k) = 1 \tag{4.1.1-3}$$

where S is the buffer size, $P(B_i^k)$ is the probability that i cells are in the buffer during the kth T_1 and $P(B_i^k | B_j^{k-1})$ is the conditional probability that i cells are in the buffer during the kth T_1, after the occurring event that j cells are in the buffer during the (k-1)th T_1.

If the event B_j^{k-1} ($0 \le j \le S$) that j cells are in the buffer during the (k-1)th T_1 period occurs, the event B_i^k ($0 < i < S$) happens on the kth T_1 period only when there are $M_{T_1} - j + i$ cells coming into the buffer during the kth T_1, since at most M_{T_1} cells of $M_{T_1} - j + i + j$ cells (j cells in the buffer) can be transferred by the multiplexed line (or going out the buffer). On the other hand, the event B_0^k or B_S^k happens on the kth T_1 period when there are less than $M_{T_1} - j + 1$ cells or larger than $M_{T_1} - j + S - 1$ arriving at the buffer during the kth T_1 period, respectively. So, we obtain the following equations:

$$P(B_i^k | B_j^{k-1}) = \begin{cases} \sum_{m=0}^{M_{T_1}-j} G(m) & (i=0) \\ G(M_{T_1} - j + i) & (1 \le i \le S-1) \\ 1 - \sum_{m=0}^{M_{T_1}-j+S-1} G(m) & (i=S) \end{cases} \tag{4.1.1-4}$$

The probability equations approaching the stationary distribution that the number of cells is in the buffer from the (k-1)th T_1 period to the kth T_1 period are given as follows:

$$\begin{bmatrix} P(B_0^k) \\ P(B_1^k) \\ \vdots \\ P(B_S^k) \end{bmatrix} = \begin{bmatrix} P(B_0^k|B_0^{k-1}) & P(B_0^k|B_1^{k-1}) & \cdots & P(B_0^k|B_S^{k-1}) \\ P(B_1^k|B_0^{k-1}) & P(B_1^k|B_1^{k-1}) & \cdots & P(B_1^k|B_S^{k-1}) \\ \vdots & & & \vdots \\ P(B_S^k|B_0^{k-1}) & P(B_S^k|B_1^{k-1}) & \cdots & P(B_S^k|B_S^{k-1}) \end{bmatrix} \begin{bmatrix} P(B_0^{k-1}) \\ P(B_1^{k-1}) \\ \vdots \\ P(B_S^{k-1}) \end{bmatrix} \quad (4.1.1\text{-}5)$$

In the following, the calculation methods for the stationary probability distribution that the number of cells in the buffer are discussed.

Method 1 (Solving Linear Probability Equations):

If $P(B_i^k)$ $(0 \le i \le S)$ can reach the stationary probability state $P(B_i)$, then $P(B_i^k)$ is equal to $P(B_i^{k-1})$ when $k \ge$ existing k". So, linear probability equations (4.1.1-5) can be changed to the following:

$$\begin{bmatrix} P(B_0^k) \\ P(B_1^k) \\ \vdots \\ P(B_S^k) \end{bmatrix} = \begin{bmatrix} P(B_0^k|B_0^{k-1}) & P(B_0^k|B_1^{k-1}) & \cdots & P(B_0^k|B_S^{k-1}) \\ P(B_1^k|B_0^{k-1}) & P(B_1^k|B_1^{k-1}) & \cdots & P(B_1^k|B_S^{k-1}) \\ \vdots & & & \vdots \\ P(B_S^k|B_0^{k-1}) & P(B_S^k|B_1^{k-1}) & \cdots & P(B_S^k|B_S^{k-1}) \end{bmatrix} \begin{bmatrix} P(B_0^k) \\ P(B_1^k) \\ \vdots \\ P(B_S^k) \end{bmatrix} \quad (4.1.1\text{-}6)$$

The above linear probability equations can be further transformed to,

$$\begin{bmatrix} P(B_0^k|B_0^{k-1})-1 & P(B_0^k|B_1^{k-1}) & \cdots & P(B_0^k|B_S^{k-1}) \\ P(B_1^k|B_0^{k-1}) & P(B_1^k|B_1^{k-1})-1 & \cdots & P(B_1^k|B_S^{k-1}) \\ \vdots & & & \vdots \\ P(B_S^k|B_0^{k-1}) & P(B_S^k|B_1^{k-1}) & \cdots & P(B_S^k|B_S^{k-1})-1 \end{bmatrix} \begin{bmatrix} P(B_0) \\ P(B_1) \\ \vdots \\ P(B_S) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (4.1.1\text{-}7)$$

Since the last row equation depends linearly on the rest, the new linear probability equations have the following forms after this equation is replaced by $\sum_{i=0}^{S} P(B_i)=1$:

$$\begin{bmatrix} P(B_0^k|B_0^{k-1})-1 & P(B_0^k|B_1^{k-1}) & \cdots & P(B_0^k|B_S^{k-1}) \\ P(B_1^k|B_0^{k-1}) & P(B_1^k|B_1^{k-1})-1 & \cdots & P(B_1^k|B_S^{k-1}) \\ \vdots & & & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} P(B_0) \\ P(B_1) \\ \vdots \\ P(B_S) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \qquad (4.1.1\text{-}8)$$

The above probability equations can be easily solved by applying several algorithms, e.g., Gassian-Jordan algorithm, LU algorithm [38], etc. The calculations of these algorithms are of order S^3.

In order to reduce the amount of calculation for solving probability equations, we may solve $P(B_{2i})$ ($0 \le i \le \frac{S}{2}$), and then apply Lagrange Interpolation algorithms to get the interpolating values of $P(B_{2i-1})$ ($1 \le i \le \frac{S}{2}$), which can generate the approximate results of the probability equations. The calculations are of order $(\frac{S}{2})^3$.

Method 2 (Converging Stationary Probability Distribution):

Another approach, which can obtain the stationary probability distribution of the number of cells in the buffer $\{ P(B_i) \}$ ($0 \le i \le S$), is to apply the equation (4.1.1-2) repeatedly. In order to speed up the convergence of the linear equation (4.1.1-2) for on-line use, the initial values of $\{ P(B_i) \}$ ($0 \le i \le S$) can be calculated off-line by solving (4.1.1-8) according to the different number and different types of connections. Based on the predetermined initial values of $\{ P(B_i) \}$ ($0 \le i \le S$), the linear equation (4.1.1-2) can be solved to determine the approximate values of the stationary probability distribution that the number of cells is in the buffer during T_1 period. The total calculations for this method are less than order S^3. In order to determine whether the convergence to reach a certain accuracy or not, the threshold ε for stopping the computations must be provided.

4.1.2 CAC Method with Buffer Model

In this section, the CAC method with buffer model is described. Cell loss rate and average queuing delay are the two main important performance objectives provided by the network to connections, which are closely related to the buffer sizes in an ATM switching node or an ATM multiplexer. Hence, the queuing delay in the buffer is first discussed. Let V be the cell transmission speed of the multiplexed line, S be output buffer size, and D be the maximum admissible queuing delay allocated to the output buffer. Assume that the buffer size S is dimensioned such that the maximum delay in the buffer is less than D under a first-in-first-out (FIFO) discipline [21],

$$D \geq \frac{SL}{V} \tag{4.1.2-1}$$

where L is the cell length. Thus the cell queueing delay objective is always satisfied, and CAC can just concentrate on meeting the desired requirements of cell loss rate. If there are multiple delay requirements, D may be set to the most stringent value.

In the following, we describe and apply the probability $G(n)$ [14] in our proposed methods with buffer model. In order to evaluate the cell loss rate, we must use the probability $G(n)$ that n cells are transferred from the multiplexed connections during T_1. Here, the value of T_1 is equal to the inverse of V_{pl} that is the maximum peak cell transmission speed of low speed connections. That is, $T_1 = 1/V_{pl}$. The low speed connection is defined as the connection whose peak cell transmission speed is smaller than the threshold value ($V_{pl} = V/100$), and the high speed connection is defined as the connection whose peak cell transmission speed is larger than ($V_{pl} = V/100$). If the multiplexed connections consist of low speed connections and high speed connections, $G(n)$ can be written as follows:

$$G(n) = \sum_{k=0}^{n} P(k)F(n-k) \qquad (0 \leq n \leq M_{T_1} + S) \tag{4.1.2-2}$$

where, $P(k)$ is defined as the probability that k cells are transferred from low speed connections during T_1, and $F(n-k)$ is defined as the probability that $n-k$ cells are transferred from high speed connections

during T_1. The low speed connections can be slow video, voice and banking transfer while the high speed connections can be all kinds of high quality video products. For example, the peak rate is 32Kbit/s and mean rate is 11.2Kbit/s for typical voice while the peak rate is 11.6Mbit/s and mean rate is 3.85Mbit/s for typical video [36][39].

By assuming that each low speed connection is modelled as transferring cells at the speed of V_{pl}, which is the maximum peak cell transmission speed of the multiplexed low speed connections ($V_{pl} = V/100$), the number of cells transferred from multiplexed low speed connections is surely larger than the actual number of cells transferred from multiplexed low speed connections in a switching node or a multiplexer. Thus, it guarantees the safety margin for estimating cell loss rate. The probability $P(k)$ is shown as follows:

$$P(k) = \sum_{k=\sum_{i=1}^{L_l} n_i} \prod_{j=1}^{L_l} \binom{N_j}{n_j} \alpha_j^{n_j} (1-\alpha_j)^{N_j - n_j} \tag{4.1.2-3}$$

$$\alpha_j = \frac{V_{aj}}{V_{pl}} \tag{4.1.2-4}$$

$$\rho_l = \sum_{j=1}^{L_l} \frac{N_j V_{aj}}{V} \tag{4.1.2-5}$$

where, L_l is the number of low speed connection types. The number of type j ($1 \le j \le L_l$) low speed connections is N_j. Peak cell transmission speed of a type j low speed connections is V_{pj} and average cell transmission speed of a type j low speed connections is V_{aj}. n_j ($1 \le j \le L_l$) is the number of a type j low speed connections that are in active state. α_j ($1 \le j \le L_l$) is the ratio average cell transmission speed of a type j low speed connection to the maximum peak cell transmission speed of multiplexed low speed connections. α_j corresponds to the probability that cell is generated during T_1 period. ρ_l ($0 \le \rho_l \le 1$) is the average offered load to the multiplexed line from low speed connections.

When one low speed connection of type j is newly established, $P'(k)$ ($0 \le k \le M_{T_1} + S$) and ρ'_l are updated by equations (4.1.2-6) and (4.1.2-7).

$$P'(k)=\begin{cases} (1-\alpha_j)P(0) & (k=0) \\ (1-\alpha_j)P(k)+\alpha_j P(k-1) & (otherwise) \end{cases} \quad (4.1.2\text{-}6)$$

$$\rho'_l=\rho_l+\frac{V_{aj}}{V} \quad (4.1.2\text{-}7)$$

Similarly, when one low speed connection of type j is torn down, $P'(k)$ $(0 \le k \le M_{T_1}+S)$ and ρ'_l are given by equations (4.1.2-8) and (4.1.2-9).

$$P'(k)=\begin{cases} \dfrac{P(0)}{1-\alpha_j} & (k=0) \\ \dfrac{P(k)-\alpha_j P'(k-1)}{1-\alpha_j} & (otherwise) \end{cases} \quad (4.1.2\text{-}8)$$

$$\rho'_l=\rho_l-\frac{V_{aj}}{V} \quad (4.1.2\text{-}9)$$

As a result, both the amount of calculation for the connection establishing and tearing down are of order $(M_{T_1}+S)$.

On the other hand, $F(k)$ is the probability that k cells are transferred from high speed connections during T_1 period. Since T_1 is larger than the inverse value of the peak cell transmission speed for high speed connections, high speed connections can transfer more than one cell during T_1. In order to calculate the cell loss rate that is never smaller than the actual cell loss rate, the worst case cell transmission pattern is assumed to evaluate cell loss rate. When $F(k)$ represents the worst case cell transmission pattern for high speed connections, a high speed connection transfers cells continuously at the speed of its peak cell transmission speed V_{ps} during T_1. Since each connection keeps its average cell transmission speed V_{as}, the probability that a high speed connection type s

transfers with the worst case pattern is V_{as} / V_{ps}. The worst case for $F(k)$ mentioned above is given by the following equations.

$$F(k)=\sum_{k=n_1+n_2+\cdots+n_{L_h}} \prod_{s=1}^{L_h} Q_s(n_s) \tag{4.1.2-10}$$

$$Q_s(n_s)=\begin{cases} \binom{N_s}{m}\alpha_s^m(1-\alpha_s)^{N_s-m} & (\ n_s=m\lceil V_{ps}T_1\rceil\ ) \\ 0 & (otherwise) \end{cases} \tag{4.1.2-11}$$

$$\alpha_s=\frac{V_{as}}{V_{ps}} \tag{4.1.2-12}$$

$$\rho_h=\sum_{s=1}^{L_h}\frac{N_sV_{as}}{V} \tag{4.1.2-13}$$

where, $\lceil x \rceil$ means the smallest integer equal to or greater than x. $Q_s(n_s)$ is the probability that n_s cells are transferred from type s high speed connections during T_1, L_h is the number of high speed connection types and N_s is the number of high speed connections of type s ($1 \le s \le L_h$). ρ_h ($0 \le \rho_h \le 1$) is the average offered load to the multiplexed line from high speed connections.

When one high speed connection of type s is newly established, $F'(k)$ ($0 \le k \le M_{T_1}+S$) and ρ'_h are updated by the following equations.

$$F'(k)=\begin{cases} (1-\alpha_s)F(k)+\alpha_sF(k-m_s) & (m_s \le k) \\ (1-\alpha_s)F(k) & (otherwise) \end{cases} \tag{4.1.2-14}$$

$$\rho'_h=\rho_h+\frac{V_{as}}{V} \tag{4.1.2-15}$$

Similarly, when one high speed connection of type s is torn down, $F'(k)$ $(0\le k \le M_{T_1}+S)$ and ρ'_h are given by the following equations.

$$F'(k)=\begin{cases}\dfrac{F(k)-\alpha_s F'(k-m_s)}{1-\alpha_s} & (m_s\le k)\\ \dfrac{F(k)}{1-\alpha_s} & (otherwise)\end{cases} \tag{4.1.2-16}$$

$$\rho'_h=\rho_h-\frac{V_{as}}{V} \tag{4.1.2-17}$$

where,

$$m_s=\lceil V_{ps}T_1\rceil \tag{4.1.2-18}$$

As a result, both the amount of calculation for the connection establishing and tearing down are of order $(M_{T_1}+S)$.

So, the amount of calculation for obtaining a new $\{P(k)\}$, which is the set of probabilities for low speed connections, and the calculation for obtaining a new $\{F(k)\}$, which is for high speed connections, are only order $(M_{T_1}+S)$ when a new connection is established or torn down. This means that the calculation for $\{G(n)\}$ is of order $(M_{T_1}+S)^2$. Moreover, the amount of calculation is independent of the number of connection types. Therefore, the amount of calculation does not increase even when the number of connection types increases.

Assume that there are i cells $(0\le i \le S)$ in the buffer after previous T_1 period. When the combined number of cells transferred from high speed and low speed connections is larger than $M_{T_1}+S-i$ cells during current T_1 period, the number of cells beyond $M_{T_1}+S-i$ cells are discarded. This is because the multiplexed line can transfer at most M_{T_1} cells during T_1 period and there are $S-i$

empty cell spaces in the buffer. We can derive the mean number of cells arriving into the buffer during T_1 from ρM_{T_1}, and the mean number of cells discarded during T_1 from (4.1.2-19):

$$\sum_{n=M_{T_1}+S-i+1}^{\infty}(n-(M_{T_1}+S-i))G(n) \tag{4.1.2-19}$$

So, if i cells ($0 \le i \le S$) are in the buffer after previous T_1 period, the cell loss probability during current T_1 period is given as follows:

$$\frac{\sum_{n=M_{T_1}+S-i+1}^{\infty}(n-(M_{T_1}+S-i))G(n)}{\rho M_{T_1}} \tag{4.1.2-20}$$

then, the estimated cell loss rate is given by equation (4.1.2-21):

$$CLR(buffer)=\sum_{i=0}^{S}P(B_i)[\frac{1}{\rho M_{T_1}}\sum_{n=M_{T_1}+S-i+1}^{\infty}(n-(M_{T_1}+S-i))G(n)] \tag{4.1.2-21}$$

where,

$$\rho=\rho_l+\rho_h=\sum_{j=1}^{L_l}\frac{N_j V_{aj}}{V}+\sum_{s=1}^{L_h}\frac{N_s V_{as}}{V} \tag{4.1.2-22}$$

here, ρ ($0 \le \rho \le 1$) is the average offered load to the multiplexed line.

Equation (4.1.2-21) can be transformed as equation (4.1.2-23), reducing the summations from $[M_{T_1}+S-i+1, \infty]$ to $[0, M_{T_1}+s-i]$ $(0 \le i \le S)$.

$$CLR(buffer)=\sum_{i=0}^{S}P(B_i)[\frac{1}{\rho M_{T_1}}\sum_{n=0}^{M_{T_1}+S-i}((M_{T_1}+S-i)-n)G(n)+\frac{\rho M_{T_1}-(M_{T_1}+S-i)}{\rho M_{T_1}}] \tag{4.1.2-23}$$

As a result, when a new connection is established or torn down, the amount of calculation for evaluating cell loss rate is of order $(M_{T_1}{}^2 + M_{T_1}S + S^3)$. In order to apply equation (4.1.2-23) to different burst lengths for all connections (the method without buffer model [14] can be applied to the case that all connections have 100 burst length traffic parameter), we introduce a new parameter S_m ($S_m \geq 1$) for equation (4.1.2-23), which is called as the related burst length parameter. Here, burst length of connection is defined as the connection that can generate cells (the number of cells is equal to burst length) continuously at the speed of its peak cell transmission rate. The equation (4.1.2-23) is modified as follows:

$$CLR(buffer) = \sum_{i=0}^{S} P(B_i)\left[\frac{1}{\rho M_{T_1}} \sum_{n=0}^{M_{T_1}-\frac{S-i}{S_m}} \left((M_{T_1}+\frac{S-i}{S_m})-n\right)G(n) + \frac{\rho M_{T_1} - (M_{T_1}+\frac{S-i}{S_m})}{\rho M_{T_1}}\right] \qquad (4.1.2\text{-}24)$$

4.1.3 Two Level CAC Method

This method is a hierarchical CAC strategy by using two different levels, which combines the fast decision of relatively simple CAC algorithm (without buffer model [14]) with the accuracy of the more complex background CAC algorithm (with buffer model) to enable real-time admission control for an ATM switching node and a multiplexer. The two level CAC algorithm is presented as follows:

Step 1: When the network traffic load is low, which means that the calculated cell loss rate is smaller than certain threshold (e.g., threshold = 10^{-30} or less), calls or connection requests can be handled by the first level algorithm without buffer model in real-time.

Step 2: When the cell loss rate calculated by the first level algorithm for adding any call is equal to or larger than certain threshold (it should be much smaller than the requested cell loss rate), the call or connection request can be accepted, then goto step 3.

Step 3: The relatively complex background algorithm with buffer model starts to calculate the stationary probability distribution $\{P(B_i)\}$ $(0 \leq i \leq S)$ for several predicted next calls in advance which

roughly represent current existing number and types of connections carried by ATM network.

Step 4: When next requested call is received, this call is compared with several predicted calls so that one of predicted calls can be chosen, which meets the following two conditions:

(1) The chosen predicted call is the most similar to the required call among several predicted calls, based on traffic characteristics (peak cell transmission speed, average cell transmission speed and average burst length).

(2) The chosen predicted call requires a stricter bandwidth than the required call does.

Then, the complex background algorithm can calculate the cell loss rate for this required call in real-time, based on the pre-prepared stationary probability distribution $\{ P(B_i) \}$ $(0 \leq i \leq S)$ to determine whether this call request can be accepted or not.

Step 5: Finally, if this call is accepted, a complex background algorithm performs a refinement of this decision, which means that it will finish the calculation of the stationary probability distribution $\{ P(B_i) \}$ $(0 \leq i \leq S)$ for current network traffic load already added by new requested call. If many connections are torn down or network traffic load is low, then the calculated cell loss rate may fall below certain threshold and goto step 1. Otherwise, goto step 3.

4.2 Performance Evaluation for CAC

4.2.1 Evaluation Model

In the following sections, the performance of the proposed method with buffer model is simulated and evaluated for a general ATM node model. To evaluate that the proposed method can improve the performance in terms of statistical multiplexing gains, based on the different sizes of queuing buffer, the numerical results of the proposed method and the method without buffer model [14] are compared with simulation results .

The simulator of the general ATM node model has been built, and simulation results have been performed by using Borland C^{++} software package, but our programming is not object-oriented. So far, several different models for the characterization of the ATM connection sources have been

identified by many researchers. These traffic models have to be flexible enough to account for a very large variety of services ranging from constant to variable cell rate. One of these traffic models is two-state Markov chain source (on/off model) [37], which has been applied in this thesis. According to this source model, the cell stream from a single ATM connection source is modelled as follows: active (burst) period where the generation of the cells occurs and silence (idle) period with no cell generation. This source model is depicted in Figure 4.2.1-1. In this figure, the burst period (burst length) where the cell generation occurs is assumed to be geometrically distributed. During this period the cells arrive every T ms ($T = 1/V_p$). After the generation of the cells, a geometrically distributed idle period (idle length) follows. M_a is defined as the average burst length and M_i is defined as the average idle length.

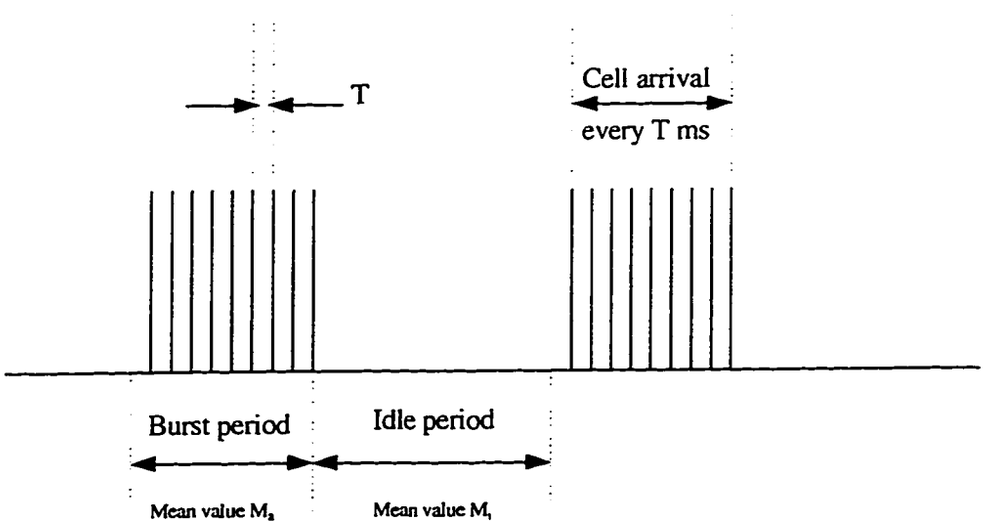


Figure 4.2.1-1 Two-State Markov Chain Source Traffic Model

In the simulations, high speed connections and low speed connections, which can generate cells with two-state Markov chain model, are multiplexed. The burst length m_a and the idle length m_i are modelled as independent Geometrically distributed random variables having a certain mean value M_a and M_i representing the average burst length and the average idle length, respectively. Geometrical random variables are given as follows:

$$P(m_a = n) = q_1 p_1^n \quad (n = 0, 1,...) \tag{4.2.1-1}$$

$$P(m_i = n) = p_2 q_2^n \quad (n = 0, 1,...) \tag{4.2.1-2}$$

where p_1 and p_2 are probabilities that the cell can be generated. q_1 and q_2 are the probabilities that no cell can be generated. The average burst length M_a and the average idle length M_i are equal to p_1/q_1 and q_2/p_2, respectively.

Since the Geometrical distribution for the burst length m_a and the idle length m_i has been specified, the ways are sought to generate samples from this distribution to be used as input to a simulation model. Since C^{++} software package does not have random-variate generation libraries, we must construct random-variate generation subroutine for Geometrical distribution. The equation for generating a Geometrical random variate, n, is given by the following (for details, see [16]):

$$n = \left\lceil \frac{\ln(1-R)}{\ln(1-P)} - 1 \right\rceil \tag{4.2.1-3}$$

here, $\lceil x \rceil$ means the smallest integer equal to or greater than x. R is an uniform random number [0, 1]. P represents q_1 for burst length m_a and represents p_2 for idle length m_i. n is a Geometrical random variate. Moreover, the different sizes of queuing buffer must be provided for the simulations.

On the other hand, the maximum allowable throughputs are numerically calculated by using equation (4.1.2-24) for the proposed method with buffer model and by using equation (2.2-3) for the method without buffer model [14], based on input traffic parameters (peak transmission speed V_p, average transmission speed V_a, average burst length M_a and average idle length M_i).

4.2.2 Performance Evaluation

Figure 4.2.2-1 to Figure 4.2.2-5 show the maximum offered load of low speed connections versus the offered load of high speed connections, when both connections require that the cell loss rate smaller than or equal to 10^{-5} (QoS requirement). The high speed connection is $V_p = V/10$, $V_a = V_p/10$, $M_a = 100$ cells and $M_i = 900$ cells. The low speed connection is $V_p = V/100$, $V_a = V_p/2$ and $M_a = M_i = 100$ cells. In order to evaluate the performance between the proposed method with buffer model and the method without buffer model [14], we have used the same traffic source parameters from [14], and have chosen the related burst length parameter ($S_m = 28.0$), which corresponds to the average burst length ($M_a = 100$ cells) provided by all connections so that the cell loss rate calculated by the proposed method can operate within a safety region. In Figures 4.2.2-1 to 4.2.2-5, the method without buffer model [14] is denoted as "Fundamental method", while the improved method with buffer model developed by us is denoted as "Proposed method".

As shown in Figure 4.2.2-1, the throughput of the proposed method with buffer model, when the queuing buffer size is zero, is the same as that of the method without buffer model [14]. This is because equation (4.1.2-24) is the same as the equation (2.2-3) if the buffer size S is equal to zero. It is shown from Figures 4.2.2-2 to 4.2.2-5 that the throughput of the method without buffer model [14] is not changed, no matter how large the queuing buffer size is. This indicates that the method without buffer model does not consider statistical multiplexing gains due to the different buffer sizes. On the contrary, the proposed method can take into account of statistical multiplexing effect fairly well. It is also shown that the proposed method can operate in safe side. This means that the estimated maximum offered load by using equation (4.1.2-24) is smaller than the actual maximum offered load by using computer simulations, while it takes into account of statistical multiplexing effect. For example, when the buffer size is 300, and the same load from high speed connections is assumed, the proposed CAC method with buffer model can allow more (18 to 20) low speed connections than the CAC method without buffer model, which means that more 9% offered load from low speed connections can be supported by an ATM multiplexer or an ATM switching node. Furthermore, as the buffer size is increased from 0 to 300, some facts can be observed that the more number of high speed connections is, the more different throughputs generated by computer

simulations and calculated by the proposed method. The reasons are: first, the given $F(k)$ is the probability that k cells are transferred from high speed connections during T_1 and it is chosen as the worst case cell transmission pattern for high speed connections in [14], second, $\{ P(B_i) \}$ $(0 \le i \le S)$ is the stationary probability distribution that the number of cells is in the queuing buffer during T_1 and it is derived from equation (4.1.2-2). Here, the equation (4.1.2-2) is given again in the following:

$$G(n) = \sum_{k=0}^{n} P(k)F(n-k) \qquad (0 \le n \le M_{T_1} + S)$$

There is a large safety margin associated with the evaluated cell loss rate in terms of $F(k)$ using the worst case cell transmission pattern, because $\{ P(B_i) \}$ $(0 \le i \le S)$ and $G(n)$ $(0 \le n \le M_{T_1} + S)$ are applied to calculate the cell loss rate within the proposed method.

4.3 Summary

In this chapter, the detailed analysis of the proposed CAC method with buffer model is presented. This proposed method can deal with not only an ATM multiplexer but also an output buffered switching node. Since cell loss rate and average queuing delay are two of main important performance objectives provided by the network to connections, and they are closely related to the buffer sizes in an ATM switching node or an ATM multiplexer, our proposed CAC method can improve the performance in terms of statistical multiplexing gains, based on different buffer sizes. Also, it is shown that the proposed method can handle certain average burst length provided by connection request. The simulation results are shown that the proposed method with buffer model can achieve the satisfactory performance based on the different sizes of queuing buffer as well as they have exposed that the cell loss rate calculated by the proposed method results in a large safety margin due to $F(k)$ chosen as the worst case cell transmission pattern for high speed connections in [14]. Also, a two-level CAC method is proposed so that our proposed algorithm could be guaranteed to compute in real-time.

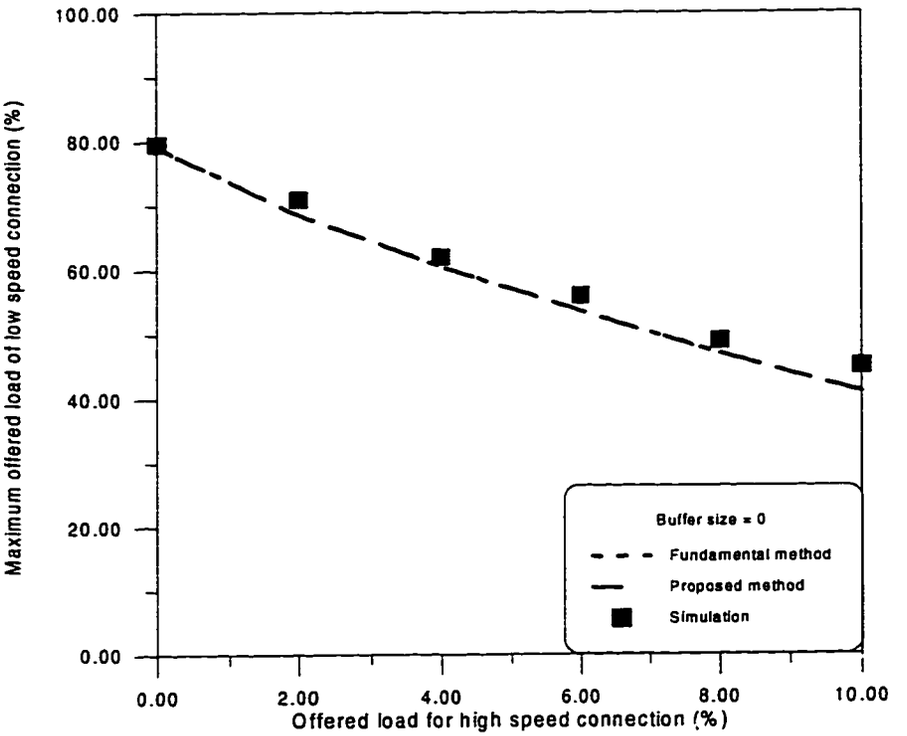


Figure 4.2.2-1 Simulation Results on Multiplexing Connections

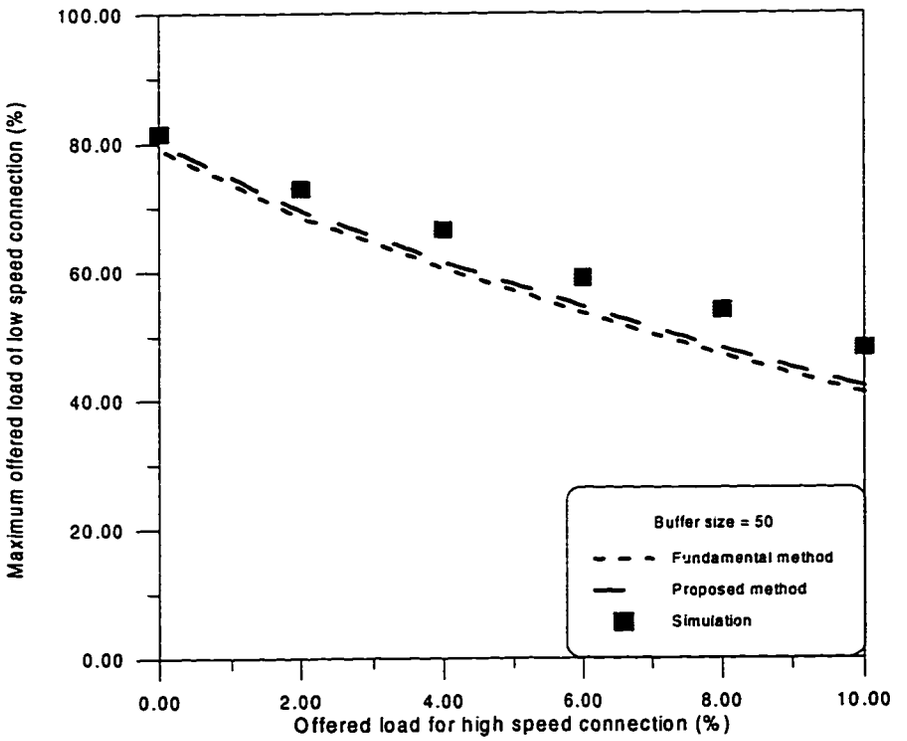


Figure 4.2.2-2 Simulation Results on Multiplexing Connections

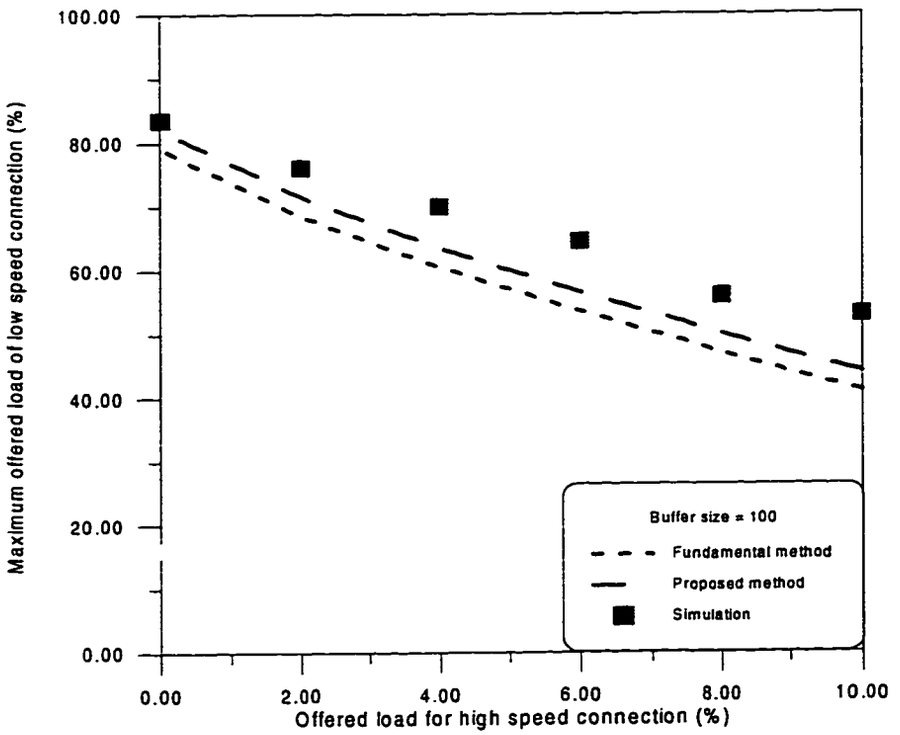


Figure 4.2.2-3 Simulation Results on Multiplexing Connections

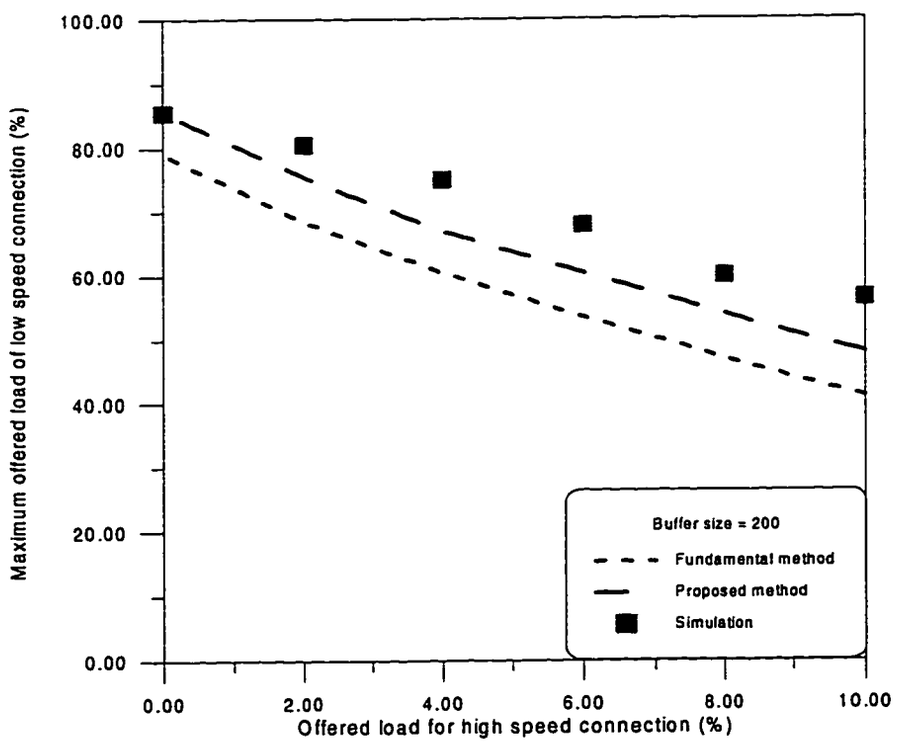


Figure 4.2.2-4 Simulation Results on Multiplexing Connections

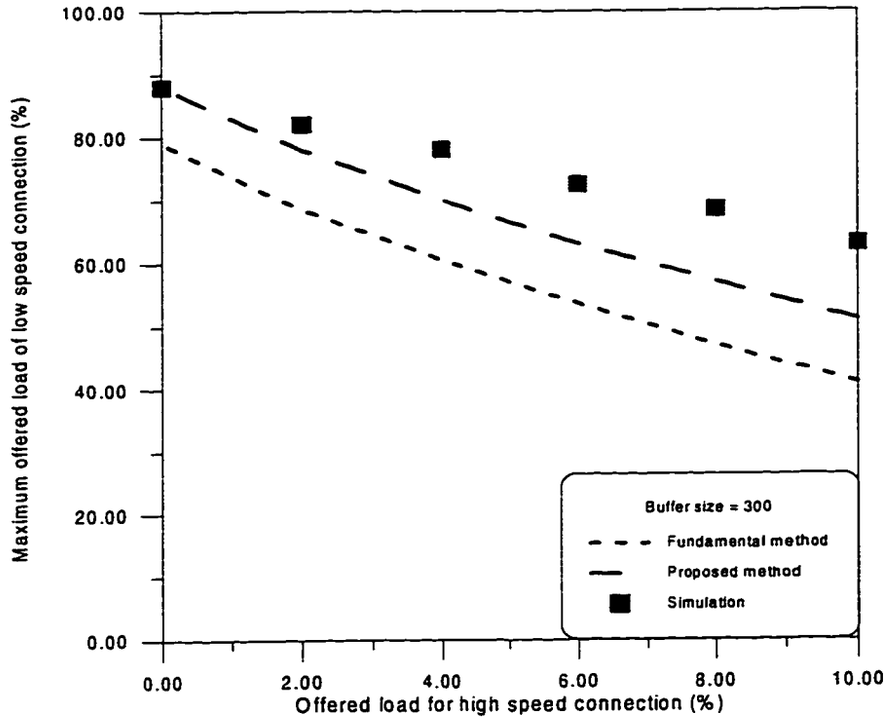


Figure 4.2.2-5 Simulation Results on Multiplexing Connections

Chapter 5

Conclusion and further work

5.1 Conclusion

In this thesis, two proposed CAC methods supporting multimedia traffic under varying traffic conditions and different QoS requirements have been presented, in order to improve the system performance with respect to the delivery of multimedia services in a multimedia communication system. The CAC methods must be developed in order to predict and avoid the potential overload situations, to balance the resource reservation/allocation, and to maximize the resource utilization in a multimedia communication system.

A complete analysis of the CAC method for supporting the simultaneous retrieval of continuous media streams have been presented, which can be used to handle four different media stream storage patterns. Relationships between the maximum, average and minimum number of simultaneous retrieving media streams and system parameters, such as, the number of disks, disk head seek time, gap size on disks, retrieval buffer size and speed ratio, have been demonstrated by using computer simulations. Simulation results have also shown that even if any disk has already reached the upper limit and cannot accept new retrieval request alone, the system is able to redistribute the load over other disks to accept the new retrieval request with the guaranteed QoS requirements or the required retrieval rates.

Finally, we have proposed and evaluated the CAC method with buffer model for ATM network. This CAC method can be applied not only to an ATM multiplexer but also to an output buffer switching node, which can improve the system performance in terms of the statistical multiplexing gains, based on the different queuing buffer sizes. Computer simulations are used to help the performance analysis, and simulation results have shown that this CAC algorithm can provide very high efficient admission control and achieve high utilization efficiency by taking into account of

statistical multiplexing effects in the buffer model. A two-level CAC method has been proposed in this thesis to enable real-time admission control for an ATM switching node and a multiplexer.

5.2 Problems and Suggestions for Future Research

In a multimedia communication system, the complete mapping and handling of the QoS at different levels of resource management is a very important and complex task. Further work and developments, such as QoS mapping, QoS negotiation and QoS dynamic support, are required.

The future works in CAC for media servers may involve developing and improving algorithms to reduce the disk head seek time and rotational latency as well as to handle different multimedia data placement strategies, such as, scattered noninterleaved data placement, contiguous noninterleaved data placement and scattered interleaved data placement.

In the design of CAC method with buffer model for ATM networks, peak cell rate, the average cell rates, and the same burst length for all connections as traffic descriptors are used in this thesis. However, to make the proposed method more useful, it should also handle the different cell burst lengths provided by each connection to determine the cell loss rate. In addition, the probability density function of the diversely multiplexed connections during T_1 period cannot be accurately predicted in the method without buffer model [14], which is one of most difficult issues in CAC techniques. This will become one issue of our future research and work.

Future work may also include the development of two new CAC methods (one for multimedia storage servers and the other for ATM networks) based on sharing a resource database with respect to the delivery of multimedia data objects according to the multimedia data delivery schedules. This data delivery schedule specifies the time when a media object should be delivered to the network. The ATM networks and multimedia storage servers must be able to cooperate for scheduling and controlling the transmission of media data objects to meet the data delivery deadlines and to guarantee the continuity of retrieving multimedia data streams. The CAC algorithms need to be developed for serving the multimedia data delivery schedules based on sharing a resource database.

Bibliography

[1] Wassim Tawbi, Linda Fedaoui and Eric Horlait, " Dynamic QoS Issues in Distributed Multimedia Systems", Broadband islands, 1993.

[2] R. Wang, "Design of a storage and retrieval model for multimedia data", Master thesis, Electrical Engineering, University of Ottawa, 1994.

[3] S. Ghandeharizadeh and L. Ramos, "Continuous Retrieval of Multimedia Data Using Parallelism", IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No. 4, August, 1993.

[4] P.V. Rangan and H.M. Vin, "Efficient Storage Techniques for Digital Continuous Multimedia", IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No. 4, August, 1993.

[5] F.A. Tobagi, J. Pang, R. Baird and M. Gang, "Streaming RAID -- A Disk Array Management System For Video Files", Starlight Networks, Inc. Mountain View, California 94041.

[6] R. Rooholamini, "ATM-based multimedia servers", IEEE Multimedia, 1995.

[7] J.J. Harms, H. Qian, "A call Admission Scheme for ATM Networks Based on Refinement of Traffic Measures", IEEE,1995.

[8] Z. Dziong, J. Chooquette, K.Q. Liao and L. Mason, "Admission control and routing in ATM networks", Computer Networks and ISDN Systems, 1990.

[9] E.D. Sykas, K.M. Vlakos, I.C. Paschalidis and G.K. Mourtzinou, "Congestion avoidance for ATM networks", Computer Communications, Vol. 17, No. 9, Sep. 1994.

[10] G.M. Stamatelos and J.F. Hayes, "Admission-control techniques with application to broadband networks", Computer Communications, Vol. 17, No. 9, Sep. 1994.

[11] T. Murase, H. Suzuki and T. Takeuchi, "A call admission control for ATM networks", ICC91, Vol. 1, 6.5, Jun. 1991.

[12] B. Jabbari and F. Yegenlu, " An upper bound of cell loss probability of burst sources in broadband packet networks". ICC91, Vol. 2, 23.3, Jun. 1991.

[13] Murase, T. Suzuki, H. And Takeuchi, T., "A Call Admission Control for ATM Networks", ICC91, vol.1,6.5, Jan. 1991.

[14] Hiroshi ESAKI, Kazuaki IWAMURA, Toshikazu KODAMA and Takeo FUKUDA,

"Connection Admission Control in ATM Networks",IEICE TRANS. COMMUN., VOL. E77-B, NO.1 Jan. 1994.

[15] Bohdan O. Szuprowicz, "Multimedia Networking", McGraw-hill, Inc., 1995.

[16] Jeey Banks, John S. Carson and Barry L. Nelson, "Discrete-Event System Simulation", Prentice-Hall, Upper Saddle, New Jersey, 1996, pp.289-297, pp.321-352.

[17] Nikolas M. Mitrou, Kimon P. Kontovasilis, "Statistical Multiplexing, Bandwidth Allocation Strategies and Connection Admission Control in ATM Networks", ETT, Vol.5, No.2, Mar.- Apr., 1994.

[18] Peter M. Chen et al.," Use of RAID Technology for Multimedia storage", ACM Computing Surveys, Vol. 26, No. 2, June 1994.

[19] S. Sibbs, D. Tsichritzis, A. Fitas, et al, "MUSE: A multimedia Filing System", IEEE Software, 4(2):4-15, March, 1997.

[20] Jim Genmmell and Stavros Christodoulakis, "Principles of Delay-Sensitive Multimedia Data Storage and Retrieval", ACM Transactions on Information Systems, Vol. 10, No. 1, Jan. 1992, pp. 51-90.

[21] Hiroshi Saito, "Teletraffic Technologies in ATM Network", Artech House Boston, London, 1995.

[22] S. Manthorpe, J.Schormans, J. Pitts and E. Scharf, "A simulation study of buffer occupancy in the ATM access network: Are renewal assumption justified?", Teletraffic and Datatraffic in a Period of Change: ITC-13 Proc., 1991, pp.801-805.

[23] H. Heffes, D. M. Lucantoni, "A Markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance", IEEE J. Select. Areas Comm., Vol. SAC-4, No. 6, Sept. 1986, pp.856-867.

[24] H. Esaki, K. Iwamura and T. Kodama, "Simple and Effective Admission Control Method for an ATM Network", GLOBECOMM90, 300.4, Vol. 1, Dec. 1990.

[25] H. Esaki, " Call Admission Control Method in ATM Network", ICC92, 354.4, Vol.3, Jan. 1992.

[26] P. Joos and W. Verbiest, "A Statistical Bandwidth Allocation and Usage Monitoring Algorithm for ATM Network", ICC89,13.5, Vol.1, pp.415-422, Jun. 1989.

[27] Arnold O. Allen, "Probability, Statistics, and Queueing Theory", ACADEMIC PRESS, INC.,

1990.

[28] J. F. Koegel Buford, "Multimedia system", 1994

[29] G. Sudhakar, A. Karmouch and N.D. Georganas, "Design and performance evaluation consideration of a multimedia medical database", IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No. 5, October, 1993.

[30] T.H. Yeap and A. Karmouch, "Modelling and performance analysis of a distributed multimedia database system over an ATM network", Electrical Engineering, University of Ottawa.

[31] W. Stallings, "Computer organization and architecture", 1990, pp 137 - 142.

[32] H.S. Stone, "Introduction to computer architecture", 1980, pp 251 - 256.

[33] W. Tawbi, F. Horn, E. Horlait and J.B. Stefani, "Video compression standards and Quality of Service", The Computer Journal, Vol. 36, No. 1, 1993.

[34] A. Vogel, B. Kerherve, G. V. Bochmann and J. Gecsei, "Distributed multimedia and QoS: a survey", IEEE Multimedia, April, 1995.

[35] R. Steinmetz and K. Nahrstedt, "Multimedia: Computing, Communications and Applications". Prentince Hall PTR, Upper Saddle River, NJ 07458.

[36] S. Rampal and D.S. Reeves, "Routing and admission control algorithms for multimedia traffic", Computer Communications, Vol. 18, No. 10, Oct. 1995.

[37] E D Sykas, K M Vlakos and M J Hillyard, "Overview of ATM networks: functions and procedures", Butterworth-Heinemann Ltd, 2 July 1991.

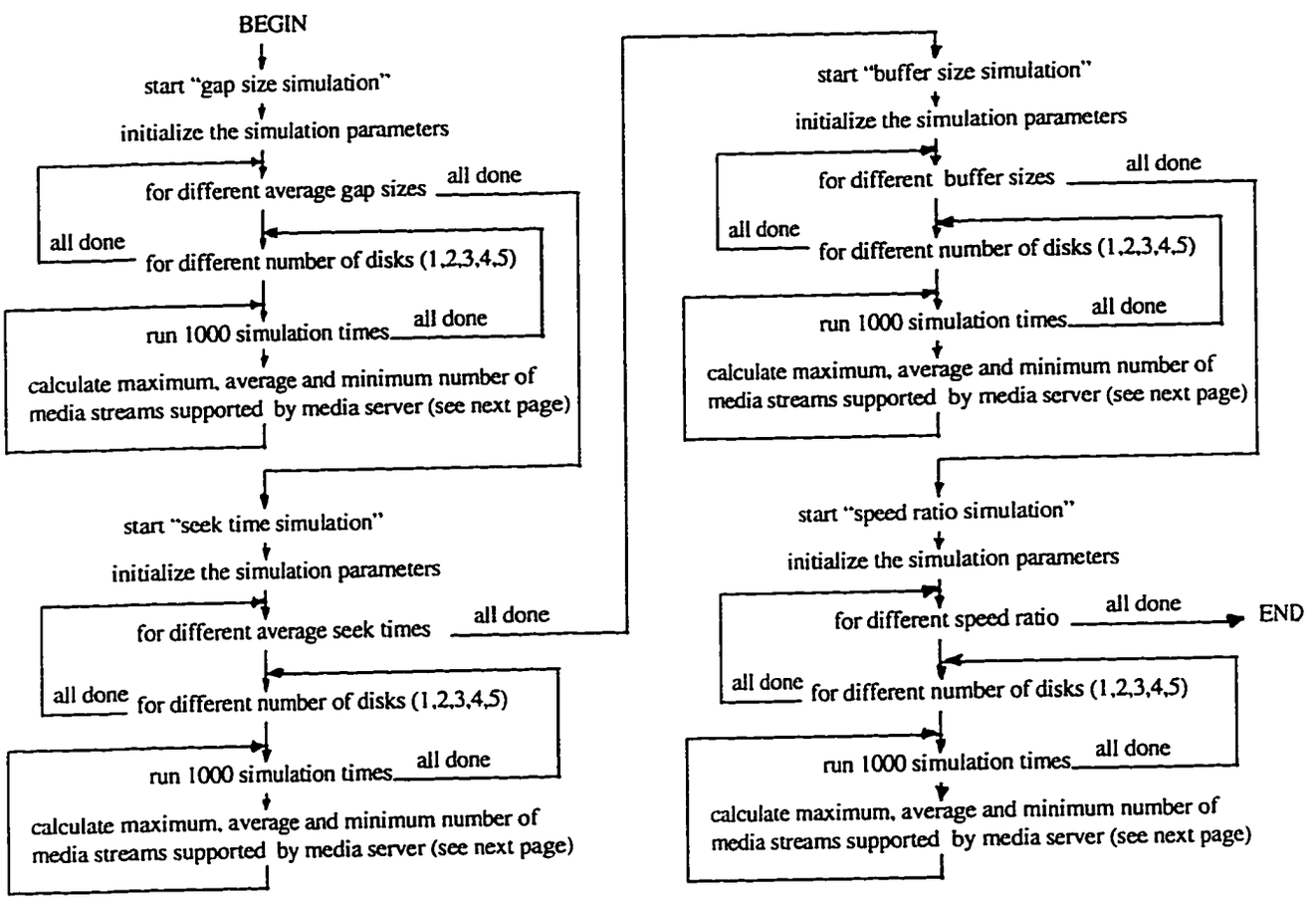
[38] Baipei Feng et al "Computer Algorithm Manual", Shanghai institute of computer technology, China, 1982.

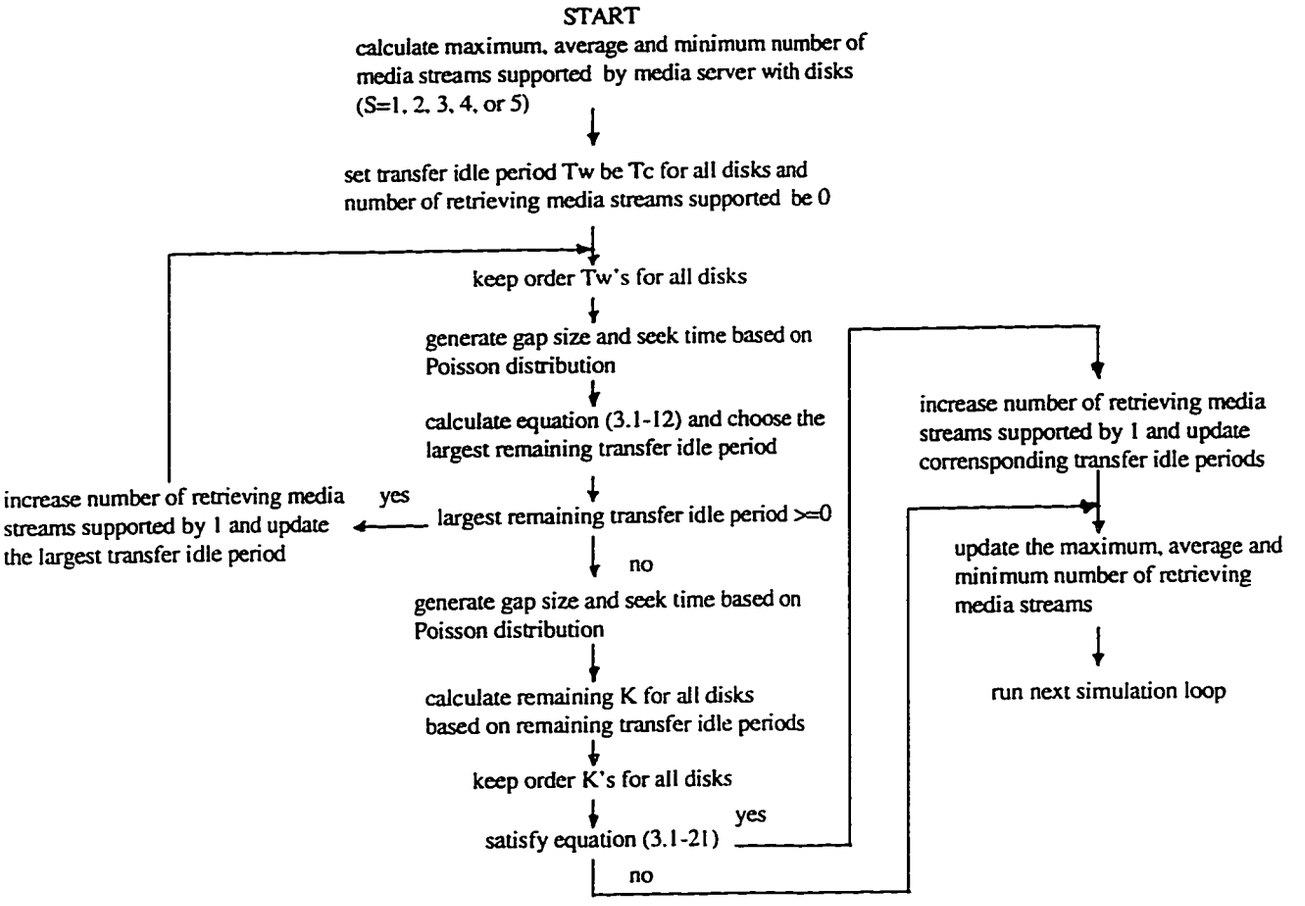
[39] Zhaohui Zheng, Chunhui Teng "Transmission of Video and Audio Over ATM Networks" Project report, University of Ottawa, 1996.

Appendix A

Connection Admission Control for Media Server

A.1 The Flowchart of CAC Simulation Model




START
calculate maximum, average and minimum number of media streams supported by media server with disks (S=1, 2, 3, 4, or 5)
set transfer idle period Tw be Tc for all disks and number of retrieving media streams supported be 0
keep order Tw's for all disks
generate gap size and seek time based on Poisson distribution
calculate equation (3.1-12) and choose the largest remaining transfer idle period
largest remaining transfer idle period >=0
yes
increase number of retrieving media streams supported by 1 and update the largest transfer idle period
no
generate gap size and seek time based on Poisson distribution
calculate remaining K for all disks based on remaining transfer idle periods
keep order K's for all disks
satisfy equation (3.1-21)
yes
increase number of retrieving media streams supported by 1 and update correnspondingtransfer idle periods
no
update the maximum, average and minimum number of retrieving media streams
run next simulation loop

A.2 The Simulation Codes

```
*************************************************************************
*                        CAC for media server with multiple disks       *
* The simulations (average gap size, average seek time, buffer size and speed ratio) are *
* implemented by the following codes                                    *
*************************************************************************

#include <iostream.h>
#include <conio.h>
#include <dos.h>
#include <process.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <bios.h>
#include <ctype.h>

#define YES                  1
#define NO                  -1
#define MAX_DISK_NUM         5
#define MAX_ARRAY_NUM       11

int
seed_num,
SIMULATION_TIMES=1000,
N,S,
final_N[MAX_DISK_NUM+1][MAX_ARRAY_NUM+1],
max_N[MAX_DISK_NUM+1][MAX_ARRAY_NUM+1],
min_N[MAX_DISK_NUM+1][MAX_ARRAY_NUM+1],
remain_K[MAX_DISK_NUM+1],
maximum_N,
minimum_N,
average_N;

long
total_average_N;

double
pre_seed,cur_seed,
```

```
sector,frame_size,
Rdt,Rc,Tc,Ts,Tf,Tr,
G,B,M,K,SR,
gapsize,seektime,
Tw[MAX_DISK_NUM+1],
temp_Tw[MAX_DISK_NUM+1];


FILE
*fp0, *fp1;

void select_seed(int seed_num)
{
 switch(seed_num)
 { case 1: pre_seed=123457.0;
           break;
   case 2: pre_seed=13.0;
           break;
   case 3: pre_seed=199.0;
           break;
   case 4: pre_seed=11111.0;
           break;
   case 5: pre_seed=4567.0;
           break;
   case 6: pre_seed=127345.0;
           break;
   case 7: pre_seed=574674.0;
           break;
   case 8: pre_seed=52346.0;
           break;
   default:pre_seed=7654321.0;
           break;
 }
}

long poisson(double mean)
{  double
   a=16807.0,
   m=2147483647.0,
   p=1.0;

  long
  n=-1;
```

```
        do
        { n++;
          cur_seed=fmod(a*pre_seed,m);
          pre_seed=cur_seed;
          p=p*cur_seed/m;
        } while (p >= exp(-mean));
        return(n);
}

void initialization(void)
{
 sector=512.0;                          //-> bytes
 frame_size=600.0*sector;
 Rc=30.0*frame_size/(1000.0); //-> bytes/ms
 Rdt=20.0*Rc;
 B=3600.0*sector;
 Ts=9.0;                                //-> ms
 M=600.0*sector;
 K=B/M;
 G=900.0*sector;
 SR=Rdt/Rc;
 Tc=B/Rc;
}

void keep_order_Tw(int total_num_disk)
{
 for (int ii=1;ii<=total_num_disk;ii++)
 {
        for (int jj=ii+1;jj<=total_num_disk;jj++)
        { double temp;

            if (Tw[ii]<Tw[jj])
           { temp=Tw[ii];
             Tw[ii]=Tw[jj];
             Tw[jj]=temp;
           }
        }
 }
}

void keep_order_remain_K(int total_num_disk)
{
 for (int ii=1;ii<=total_num_disk;ii++)
```

```
 {
        for (int jj=ii+1;jj<=total_num_disk;jj++)
        { int temp_int;

             if (remain_K[ii]<remain_K[jj])
            { temp_int=remain_K[ii];
              remain_K[ii]=remain_K[jj];
              remain_K[jj]=temp_int;
            }
        }
 }
}

void sub_disk_simulation(int total_num_disk,int index)
{ int
 finish;

 double
 temp_result;

 maximum_N=0,
 minimum_N=100,
 average_N=0;
 total_average_N=0;

 for (int ss=1;ss<=SIMULATION_TIMES;ss++)
 {
        finish=NO;
        for (int ii=1;ii<=total_num_disk;ii++)
        { Tw[ii]=Tc; }
        N=0;
        while (finish != YES)
        { keep_order_Tw(total_num_disk);
          gapsize=poisson(G/(10.0*sector))*10.0*sector;
          seektime=poisson(Ts);
          temp_result=B/Rdt+(K-1.0)*gapsize/Rdt+seektime;
          if (Tw[1]-temp_result>=0.0)
          {  N++;
             Tw[1]=Tw[1]-temp_result;
          }
          else
          {  for (ii=1;ii<=total_num_disk;ii++)
             { if (Tw[ii]>0.0)
```

```
                { gapsize=poisson(G/(10.0*sector))*10.0*sector;
                  seektime=poisson(Ts);
                  remain_K[ii]=floor(((Tw[ii]-seektime)*Rdt+gapsize)/(M+gapsize));
                  if (remain_K[ii]<0) remain_K[ii]=0;
                  if (remain_K[ii]>0) temp_Tw[ii]=0.0;
                  else                temp_Tw[ii]=Tw[ii];
                }
                else
                { remain_K[ii]=0;
                  temp_Tw[ii]=0.0;
                }
              }
              keep_order_remain_K(total_num_disk);
              {   int final=1, j, total_k=0;

                  while ((final<=total_num_disk)&&(total_k<K))
                  {  total_k=total_k+remain_K[final];
                     final++;
                  }
                  if (total_k>=K)
                  { for (j=1;j<=final-1;j++)
                    { Tw[final]=temp_Tw[final]; }
                    N++;
                  }
                  finish=YES;
               }
            }
          }

          if (maximum_N<N) {maximum_N=N;}
          if (minimum_N>N) {minimum_N=N;}
          total_average_N=total_average_N+N;
  }
  average_N=floor(total_average_N/SIMULATION_TIMES);
  final_N[total_num_disk][index]=average_N;
  max_N[total_num_disk][index]=maximum_N;
  min_N[total_num_disk][index]=minimum_N;
}

void open_file_and_write(int simulation_type, int disk_num)
{ char si_char[30],di_char[30],ch[30];

   itoa(simulation_type,si_char,10);
```

```
   itoa(disk_num,di_char,10);
   strcpy(ch,"\\ccccc\\disk");
   strcat(ch,si_char);
   strcat(ch,"_");
   strcat(ch,di_char);
   strcat(ch,".dat");
   if ((fp1=fopen(ch,"w"))==NULL)
   { printf("Cannot open file!"); exit(1); }
 }

void disk_simulation(int index)
{  for (int disk_num=1; disk_num<=MAX_DISK_NUM; disk_num++)
   { sub_disk_simulation(disk_num, index);}
}

main()
{ int index, disk_num, array_num;

  if ((fp0=fopen("\\ccccc\\disk_report.dat","w"))==NULL)
  {printf("Cannot open file!"); exit(1);}

  printf("**********FINAL REPORT********************************\n");
  printf("* The simulation results of gap size, disk head seek time,                *\n");
  printf("* retrieval buffer size and speed ratio                                   *\n");
  printf("*************************************************************************\n");
  printf("The run simulation times is %d\n", SIMULATION_TIMES);
  printf("The constant system parameters are listed as follows:\n");
  printf("Sector is 512 bytes\n");
  printf("Frame size is 600 sectors\n");
  printf("Buffer consumption rate is 30 frames per sec\n");
  printf("buffer size is 3600 sectors for experiment one, two and four\n");
  printf("Speed ratio is 20 for experiment one, two and three\n");
  printf("Average gap size is 900 sectors for experiment two,three and four\n");
  printf("Average seek time is 9 ms for experiment one, three and four\n");
  printf("gap size and seek time are modelled as independent Poisson\n");
  printf("random variables\n\n");

  fprintf(fp0,"**********FINAL REPORT**************************\n");
  fprintf(fp0,"* The simulation results of gap size, disk head seek time,          *\n");
  fprintf(fp0,"* retrieval buffer size and speed ratio                             *\n");
  fprintf(fp0,"*************************************************************************\n");
  fprintf(fp0,"The run simulation times is %d\n", SIMULATION_TIMES);
  fprintf(fp0,"The constant system parameters are listed as follows:\n");
```

```
fprintf(fp0,"Sector is 512 bytes\n");
fprintf(fp0,"Frame size is 600 sectors\n");
fprintf(fp0,"Buffer consumption rate is 30 frames per sec\n");
fprintf(fp0,"buffer size is 3600 sectors for experiment one, two and four\n");
fprintf(fp0,"Speed ratio is 20 for experiment one, two and three\n");
fprintf(fp0,"Average gap size is 900 sectors for experiment two,three and four\n");
fprintf(fp0,"Average seek time is 9 ms for experiment one, three and four\n");
fprintf(fp0,"gap size and seek time are modelled as independent Poisson\n");
fprintf(fp0,"random variables\n\n");

seed_num=1;
select_seed(seed_num);
initialization();
index=1;
for (G=500.0*sector;G<=10.0*500.0*sector;G=G+500.0*sector)
{
   disk_simulation(index);
   index++;
}

for (disk_num=1;disk_num<=MAX_DISK_NUM; disk_num++)
{
   printf("total_disk_num=%d\n",disk_num);
   fprintf(fp0,"total_disk_num=%d\n",disk_num);
  G=500.0;
  printf("gap_size  minimum_num_N average_num_N maximum_num_N\n");
  fprintf(fp0,"gap_size  minimum_num_N average_num_N maximum_num_N\n");
  open_file_and_write(1,disk_num);
  for (array_num=1;array_num<=10;array_num++)
  {
      printf("%f %d %d %d \n", G,min_N[disk_num][array_num],
                final_N[disk_num][array_num], max_N[disk_num][array_num]);
      fprintf(fp0,"%f %d %d %d \n", G,min_N[disk_num][array_num],
                final_N[disk_num][array_num], max_N[disk_num][array_num]);
      fprintf(fp1,"%f %d %d %d \n", G,min_N[disk_num][array_num],
                final_N[disk_num][array_num], max_N[disk_num][array_num]);
      G=G+500.0;
  }
  printf("\n");
  fprintf(fp0,"\n");
  fclose(fp1);
}
```

```
seed_num=2;
select_seed(seed_num);
initialization();
index=1;
for (Ts=0.0;Ts<=20.0;Ts=Ts+2.0)
{
    disk_simulation(index);
    index++;
}
for (disk_num=1;disk_num<=MAX_DISK_NUM; disk_num++)
{
   printf("total_disk_num=%d\n",disk_num);
   fprintf(fp0,"total_disk_num=%d\n",disk_num);
   Ts=0.0;
   printf("seek time minimum_num_N average_num_N maximum_num_N\n");
   fprintf(fp0,"seek_time minimum_num_N average_num_N maximum_num_N\n");
   open_file_and_write(2,disk_num);
   for (array_num=1;array_num<=11;array_num++)
  {
     printf("%f %d %d %d \n", Ts,min_N[disk_num][array_num],
                final_N[disk_num][array_num], max_N[disk_num][array_num]);
     fprintf(fp0,"%f %d %d %d \n", Ts,min_N[disk_num][array_num],
                final_N[disk_num][array_num], max_N[disk_num][array_num]);
     fprintf(fp1,"%f %d %d %d \n", Ts,min_N[disk_num][array_num],
                final_N[disk_num][array_num], max_N[disk_num][array_num]);
     Ts=Ts+2.0;
  }
  printf("\n");
  fprintf(fp0,"\n");
  fclose(fp1);
}

seed_num=3;
select_seed(seed_num);
initialization();
index=1;
for (B=M;B<=10.0*M;B=B+M)
{  K=B/M;
    Tc=B/Rc;
    disk_simulation(index);
    index++;
}
for (disk_num=1;disk_num<=MAX_DISK_NUM; disk_num++)
```

```
{
    printf("total_disk_num=%d\n",disk_num);
    fprintf(fp0,"total_disk_num=%d\n",disk_num);
    B=M;
    printf("buffer_size minimum_num_N average_num_N maximum_num_N\n");
    fprintf(fp0,"buffer_size minimum_num_N average_num_N maximum_num_N\n");
    open_file_and_write(3,disk_num);
    for (array_num=1;array_num<=10;array_num++)
    {
      printf("%f %d %d %d \n", B/sector,min_N[disk_num][array_num],
                final_N[disk_num][array_num], max_N[disk_num][array_num]);
      fprintf(fp0,"%f %d %d %d \n", B/sector,min_N[disk_num][array_num],
                final_N[disk_num][array_num], max_N[disk_num][array_num]);
      fprintf(fp1,"%f %d %d %d \n", B/sector,min_N[disk_num][array_num],
                final_N[disk_num][array_num], max_N[disk_num][array_num]);
      B=B+M;
    }
    printf("\n");
    fprintf(fp0,"\n");
    fclose(fp1);
}

seed_num=4;
select_seed(seed_num);
initialization();
index=1;
for (Rdt=5.0*Rc;Rdt<=55.0*Rc;Rdt=Rdt+5.0*Rc)
{  SR=Rdt/Rc;
   disk_simulation(index);
   index++;
}
for (disk_num=1;disk_num<=MAX_DISK_NUM; disk_num++)
{
    printf("total_disk_num=%d\n",disk_num);
    fprintf(fp0,"total_disk_num=%d\n",disk_num);
    Rdt=5.0*Rc;
    SR=Rdt/Rc;
    printf("speed_ratio minimum_num_N average_num_N maximum_num_N\n");
    fprintf(fp0,"speed_ratio minimum_num_N average_num_N maximum_num_N\n");
    open_file_and_write(4,disk_num);
    for (array_num=1;array_num<=11;array_num++)
    {
        printf("%f %d %d %d \n", SR,min_N[disk_num][array_num],
```

```
                final_N[disk_num][array_num], max_N[disk_num][array_num]);
        fprintf(fp0,"%f %d %d %d \n", SR,min_N[disk_num][array_num],
                final_N[disk_num][array_num], max_N[disk_num][array_num]);
        fprintf(fp1,"%f %d %d %d \n", SR,min_N[disk_num][array_num],
                final_N[disk_num][array_num], max_N[disk_num][array_num]);
        Rdt=Rdt+5.0*Rc;
        SR=Rdt/Rc;
    }
    printf("\n");
    fprintf(fp0,"\n");
    fclose(fp1);
  }
  fclose(fp0);
}
```

A.3 The Sample Output

```
**********FINAL REPORT**********
* The simulation results of gap size, disk head seek time,                    *
* retrieval buffer size and speed ratio                                       *
*****************************************************************************
The run simulation times is 1000
The constant system parameters are listed as follows:
Sector is 512 bytes
Frame size is 600 sectors
Buffer consumption rate is 30 frames per sec
buffer size is 3600 sectors for experiment one, two and four
Speed ratio is 20 for experiment one, two and three
Average gap size is 900 sectors for experiment two, three and four
Average seek time is 9 ms for experiment one, three and four
gap size and seek time are modelled as independent Poisson
random variables

total_disk_num=1
gap_size  minimum_num_N average_num_N maximum_num_N
500.000000 6 7 8
1000.000000 5 5 6
1500.000000 4 4 5
2000.000000 3 3 4
2500.000000 3 3 4
3000.000000 2 2 3
3500.000000 2 2 3
4000.000000 2 2 2
4500.000000 2 2 2
5000.000000 2 2 2

total_disk_num=2
gap_size  minimum_num_N average_num_N maximum_num_N
500.000000 13 14 16
1000.000000 10 11 12
1500.000000 9 9 10
2000.000000 7 7 9
2500.000000 6 6 7
3000.000000 6 6 6
3500.000000 5 5 6
4000.000000 4 4 5
4500.000000 4 4 5
```

5000.000000 4 4 4

total_disk_num=3
gap_size minimum_num_N average_num_N maximum_num_N
500.000000 19 22 24
1000.000000 16 17 18
1500.000000 13 14 15
2000.000000 10 12 13
2500.000000 10 10 11
3000.000000 8 9 10
3500.000000 7 7 9
4000.000000 7 7 8
4500.000000 6 6 7
5000.000000 6 6 7

total_disk_num=4
gap_size minimum_num_N average_num_N maximum_num_N
500.000000 27 29 32
1000.000000 21 23 25
1500.000000 17 18 20
2000.000000 13 16 17
2500.000000 13 13 14
3000.000000 12 12 13
3500.000000 9 10 12
4000.000000 9 9 10
4500.000000 9 9 9
5000.000000 8 8 9

total_disk_num=5
gap_size minimum_num_N average_num_N maximum_num_N
500.000000 34 36 40
1000.000000 26 28 31
1500.000000 21 23 26
2000.000000 18 20 21
2500.000000 16 16 19
3000.000000 14 15 16
3500.000000 11 12 15
4000.000000 11 11 12
4500.000000 11 11 11
5000.000000 10 10 11

total_disk_num=1
seek_time minimum_num_N average_num_N maximum_num_N

0.000000 8 8 9
2.000000 7 7 8
4.000000 6 7 8
6.000000 6 6 8
8.000000 5 6 7
10.000000 5 5 7
12.000000 4 5 6
14.000000 4 5 6
16.000000 4 4 6
18.000000 4 4 5
20.000000 4 4 5

total_disk_num=2
seek_time minimum_num_N average_num_N maximum_num_N
0.000000 16 17 18
2.000000 15 15 17
4.000000 13 14 16
6.000000 12 13 14
8.000000 11 12 14
10.000000 10 11 13
12.000000 10 10 12
14.000000 9 10 11
16.000000 8 9 11
18.000000 8 9 10
20.000000 7 8 10

total_disk_num=3
seek_time minimum_num_N average_num_N maximum_num_N
0.000000 25 25 27
2.000000 22 23 25
4.000000 20 21 23
6.000000 19 20 21
8.000000 17 18 20
10.000000 16 17 19
12.000000 15 16 18
14.000000 14 15 17
16.000000 13 14 16
18.000000 12 13 15
20.000000 12 12 15

total_disk_num=4
seek_time minimum_num_N average_num_N maximum_num_N
0.000000 33 33 36

2.000000 29 31 33
4.000000 27 28 31
6.000000 25 26 29
8.000000 23 24 27
10.000000 21 23 25
12.000000 20 21 24
14.000000 18 20 22
16.000000 17 19 21
18.000000 16 18 20
20.000000 16 17 19

total_disk_num=5
seek_time minimum_num_N average_num_N maximum_num_N
0.000000 41 42 45
2.000000 36 39 42
4.000000 34 36 38
6.000000 31 33 36
8.000000 29 31 33
10.000000 26 29 31
12.000000 25 26 30
14.000000 22 25 28
16.000000 21 24 27
18.000000 21 22 25
20.000000 20 21 24

total_disk_num=1
buffer_size minimum_num_N average_num_N maximum_num_N
600.000000 2 2 5
1200.000000 3 4 6
1800.000000 4 4 6
2400.000000 4 5 7
3000.000000 5 5 6
3600.000000 5 5 7
4200.000000 5 6 7
4800.000000 6 6 7
5400.000000 6 6 7
6000.000000 6 6 7

total_disk_num=2
buffer_size minimum_num_N average_num_N maximum_num_N
600.000000 4 5 8
1200.000000 6 8 10
1800.000000 8 9 12

2400.000000 9 10 12
3000.000000 10 11 13
3600.000000 11 12 13
4200.000000 11 12 14
4800.000000 12 12 14
5400.000000 12 12 14
6000.000000 12 13 14

total_disk_num=3
buffer_size minimum_num_N average_num_N maximum_num_N
600.000000 6 8 13
1200.000000 10 12 15
1800.000000 12 14 17
2400.000000 14 16 18
3000.000000 15 17 19
3600.000000 16 18 20
4200.000000 17 18 20
4800.000000 18 19 21
5400.000000 18 19 21
6000.000000 19 19 21

total_disk_num=4
buffer_size minimum_num_N average_num_N maximum_num_N
600.000000 7 10 15
1200.000000 13 16 19
1800.000000 17 19 22
2400.000000 19 21 24
3000.000000 21 23 25
3600.000000 22 24 26
4200.000000 23 24 27
4800.000000 24 25 27
5400.000000 25 25 28
6000.000000 25 26 28

total_disk_num=5
buffer_size minimum_num_N average_num_N maximum_num_N
600.000000 9 13 18
1200.000000 16 20 24
1800.000000 21 24 27
2400.000000 24 26 30
3000.000000 26 28 31
3600.000000 26 30 32
4200.000000 29 31 33

4800.000000 30 31 34
5400.000000 31 31 35
6000.000000 31 32 35

total_disk_num=1
speed_ratio minimum_num_N average_num_N maximum_num_N
5.000000 1 1 2
10.000000 3 3 4
15.000000 4 4 5
20.000000 5 5 7
25.000000 6 6 8
30.000000 7 7 9
35.000000 7 8 10
40.000000 8 9 11
45.000000 9 10 12
50.000000 9 10 13
55.000000 9 11 13

total_disk_num=2
speed_ratio minimum_num_N average_num_N maximum_num_N
5.000000 3 3 4
10.000000 6 6 8
15.000000 9 9 11
20.000000 10 12 14
25.000000 13 14 16
30.000000 14 15 18
35.000000 16 17 20
40.000000 17 18 21
45.000000 18 20 23
50.000000 19 21 24
55.000000 20 22 25

total_disk_num=3
speed_ratio minimum_num_N average_num_N maximum_num_N
5.000000 4 5 6
10.000000 10 10 11
15.000000 13 14 15
20.000000 16 18 20
25.000000 19 21 23
30.000000 22 23 26
35.000000 24 26 29
40.000000 26 28 32
45.000000 27 30 33

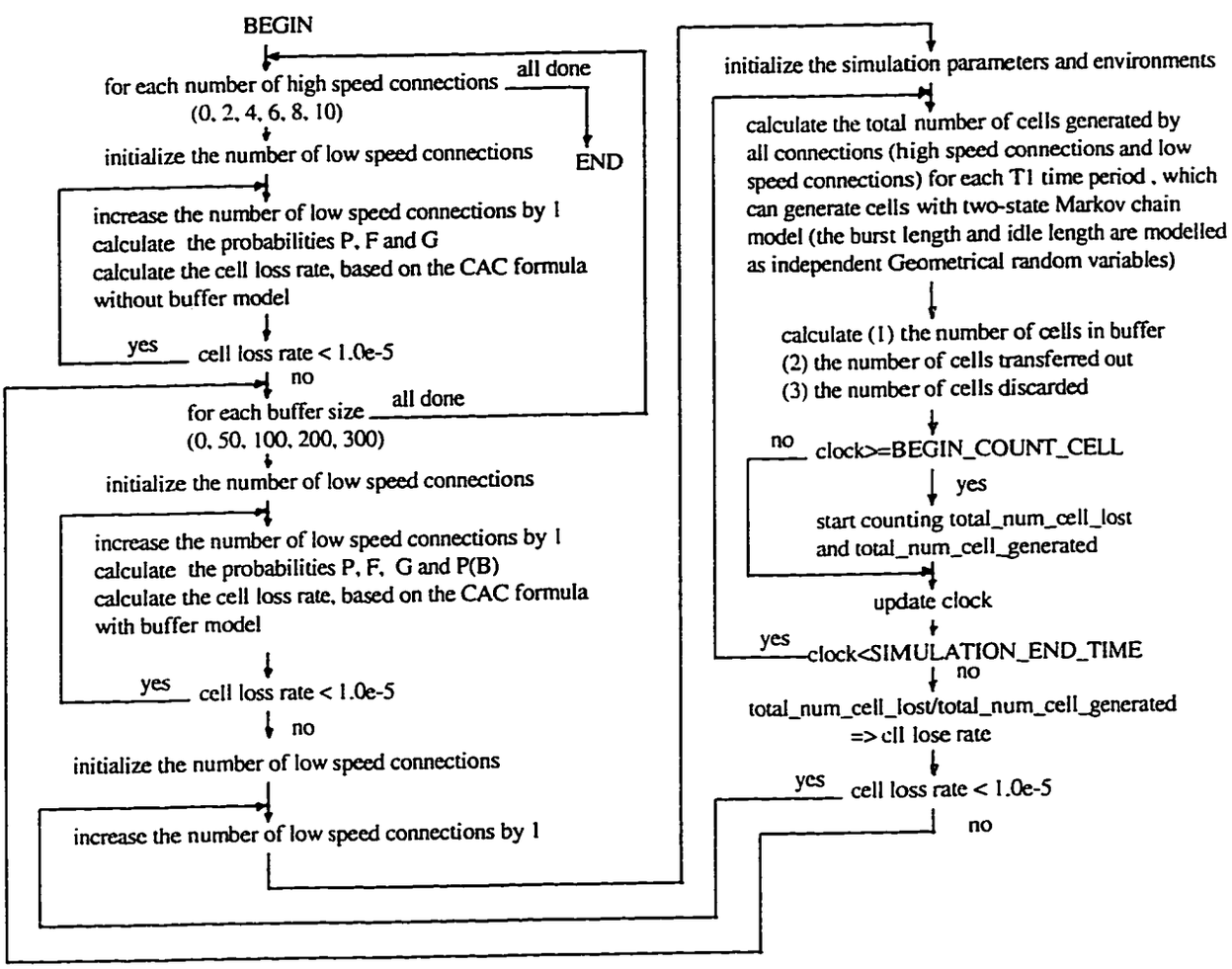
50.000000 29 32 35
55.000000 31 33 38

total_disk_num=4
speed_ratio minimum_num_N average_num_N maximum_num_N
5.000000 5 6 8
10.000000 13 13 15
15.000000 17 19 21
20.000000 21 24 26
25.000000 26 28 30
30.000000 29 31 34
35.000000 32 35 38
40.000000 34 38 41
45.000000 37 40 44
50.000000 39 42 47
55.000000 41 45 50

total_disk_num=5
speed_ratio minimum_num_N average_num_N maximum_num_N
5.000000 6 8 10
10.000000 16 16 18
15.000000 21 24 26
20.000000 27 30 32
25.000000 32 35 38
30.000000 36 39 43
35.000000 40 43 47
40.000000 41 47 51
45.000000 47 50 55
50.000000 49 53 58
55.000000 52 56 61

Appendix B

Connection Admission Control for ATM Network



B.2 The Simulation Codes

```
*************************************************************************
*                          CAC for ATM network                          *
* The CAC method without buffer model, the proposed CAC method with buffer model  *
* and  CAC simulations are implemented by the following codes           *
*************************************************************************

#include <iostream.h>
#include <conio.h>
#include <dos.h>
#include <process.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

#define YES                     1
#define NO                      -1
#define MAX_SOURCE_NUM          2
#define MAX_NUM_SAME_SOURCE     200
#define MAX_PRO_NUM             1000
#define ARRAY_NUM               6
#define BUFFER_NUM              5
#define MAX_BUFFER_SIZE         400

double
V = 100.0,
PEAK_CELL_RATE_1 = V/10.0,
AVERAGE_CELL_RATE_1 = PEAK_CELL_RATE_1/10.0,
PEAK_CELL_RATE_2 = V/100.0,
AVERAGE_CELL_RATE_2 = PEAK_CELL_RATE_2/2.0,
MAX_RATE = V/100.0,
T = 1.0/MAX_RATE,

ACTIVE_MEAN_1 = 100.0,
SILENCE_MEAN_1 = 900.0,

ACTIVE_MEAN_2 = 100.0,
SILENCE_MEAN_2 = 100.0;
```

```
int
BUFFER_SIZE,NUM_SOURCE_1,NUM_SOURCE_2;

double
count1,count2,
pre_seed, cur_seed,
total_num_cell_lost, total_num_cell,
clock, ru, related_parameter=28.0,
aa=0.0, bb=0.0;

double
SIMULATION_END_TIME,
BEGIN_COUNT_CELL,
b1[MAX_BUFFER_SIZE+1],
b2[MAX_BUFFER_SIZE+1],
fro[MAX_PRO_NUM+1],
pro[MAX_PRO_NUM+1],
gro[MAX_PRO_NUM+1];

int
mt, seed_num=1, cell_in_buffer,
connection_num[MAX_SOURCE_NUM+1];

struct cell_table_type
{ int sign;    //1 -> active, 0 -> silence
  double t;
  double next_start_time;
  double end_time;
};

struct cell_table_type
        cell_table[MAX_SOURCE_NUM+1][MAX_NUM_SAME_SOURCE+1];

struct source_type
{ double peak_cell_rate;
  double average_cell_rate;
  double active_mean;  //number of cells
  double silence_mean;
};

struct source_type
        source[MAX_SOURCE_NUM+1];
```

```
void select_seed(int seed_num)
{
 switch(seed_num)
 { case 1: pre_seed=123457.0;
           break;
   case 2: pre_seed=13.0;
           break;
   case 3: pre_seed=199.0;
           break;
   case 4: pre_seed=11111.0;
           break;
   case 5: pre_seed=4567.0;
           break;
   case 6: pre_seed=127345.0;
           break;
   case 7: pre_seed=574674.0;
           break;
   case 8: pre_seed=52346.0;
           break;
   default: pre_seed=7654321.0;
           break;
 }
}

int geometric(double mean)
{ int
  X;
 double
 a=16807.0,m=2147483647.0;

 cur_seed=fmod(a*pre_seed,m);
 pre_seed=cur_seed;
 X=ceil(log(1.0-cur_seed/m)/log(1.0-1.0/(mean+1.0))-1.0);
 aa++;bb=bb+X;
 return(X);
}

long double order(int n)
{ long double
  sum;

 if (n==0) return(1.0);
 sum=1.0;
```

```
 for (int i=1;i<=n;i++)
 { sum=sum*i;}
  return(sum);
}

double C_order(int n,int k)
{ double
  sum;

  sum=order(n)/(order(k)*order(n-k));
  return(sum);
}

double P(int k,int N,double arfa)
{double
 sum=1.0;

 if (k>N) return (0.0);
 for (int i=1;i<=k;i++)
 { sum=sum*arfa;}
 for (i=1;i<=N-k;i++)
 { sum=sum*(1-arfa);}
 sum=C_order(N,k)*sum;
 return(sum);
}

void initialization_pro_fro_gro(void)
{
 for (int i=0;i<=MAX_PRO_NUM;i++)
 {pro[i]=0.0;fro[i]=0.0;gro[i]=0.0;}

 for (i=0;i<=NUM_SOURCE_2;i++)
 {pro[i]=P(i,NUM_SOURCE_2,AVERAGE_CELL_RATE_2/MAX_RATE);}

 for (i=0;i<=NUM_SOURCE_1;i++)
 {int d;

  d=i*ceil(PEAK_CELL_RATE_1*T);
  fro[d]=P(i,NUM_SOURCE_1,AVERAGE_CELL_RATE_1/PEAK_CELL_RATE_1);
  }

  for (int n=0;n<=MAX_PRO_NUM;n++)
  {gro[n]=0.0;
```

```
    for (int k=0;k<=n;k++)
    {gro[n]=gro[n]+pro[k]*fro[n-k];}
   }
     ru=(NUM_SOURCE_1*AVERAGE_CELL_RATE_1+NUM_SOURCE_2*
AVERAGE_CELL_RATE_2)/V;
}

double cell_loss_rate(int w)
{ double sum;

  sum=0.0;
  for (int k=0;k<=w;k++)
  {  sum=sum+(mt-k)*gro[k];}
  sum=sum/(ru*mt)-(1.0-ru)/ru;
  return(sum);
}

void initialization(int num_1,int num_2)
{
 count1=0.0;count2=0.0;
 cell_in_buffer=0;
 total_num_cell_lost=0.0;
 total_num_cell=0.0;

 connection_num[1]=num_1;
 source[1].peak_cell_rate=PEAK_CELL_RATE_1;
 source[1].average_cell_rate=AVERAGE_CELL_RATE_1;
 source[1].active_mean=ACTIVE_MEAN_1;  //the number of cells
 source[1].silence_mean=SILENCE_MEAN_1;

 connection_num[2]=num_2;
 source[2].peak_cell_rate=PEAK_CELL_RATE_2;
 source[2].average_cell_rate=AVERAGE_CELL_RATE_2;
 source[2].active_mean=ACTIVE_MEAN_2;  //the number of cells
 source[2].silence_mean=SILENCE_MEAN_2;

 for (int source_type=1; source_type<=MAX_SOURCE_NUM;source_type++)
 {  for (int i=1; i<=floor(connection_num[source_type]/2.0);i++)
    {cell_table[source_type][i].next_start_time=0.0;
     cell_table[source_type][i].t=1.0/source[source_type].peak_cell_rate;
     cell_table[source_type][i].sign=1;
     cell_table[source_type][i].end_time=
          cell_table[source_type][i].next_start_time+
```

```
            cell_table[source_type][i].t*source[source_type].active_mean;
      }

      for (i=floor(connection_num[source_type]/2.0); i<=connection_num[source_type];i++)
     {cell_table[source_type][i].next_start_time=0.0;
      cell_table[source_type][i].t=1.0/source[source_type].peak_cell_rate;
      cell_table[source_type][i].sign=0;
      cell_table[source_type][i].end_time=
           cell_table[source_type][i].next_start_time+
           cell_table[source_type][i].t*source[source_type].silence_mean;
     }
  }
  clock=0.0;
}

void update_table(int source_type,int i)
{
   if (cell_table[source_type][i].sign==1)
  {  cell_table[source_type][i].end_time=
           cell_table[source_type][i].next_start_time+
           cell_table[source_type][i].t*geometric(source[source_type].silence_mean);
     cell_table[source_type][i].sign=0;
  }
  else
  {  cell_table[source_type][i].end_time=
           cell_table[source_type][i].next_start_time+
           cell_table[source_type][i].t*geometric(source[source_type].active_mean);
     cell_table[source_type][i].sign=1;
  }
}

void read_table(void)
{ int beyond_clock=NO, cell_num=0;
  while(beyond_clock==NO)
  { beyond_clock=YES;
    for (int source_type=1; source_type<=MAX_SOURCE_NUM;source_type++)
    {  for (int i=1; i<=connection_num[source_type];i++)
       {  if (cell_table[source_type][i].next_start_time<=clock)
          {  beyond_clock=NO;
             cell_num=cell_num+cell_table[source_type][i].sign;
             cell_table[source_type][i].next_start_time=
                    cell_table[source_type][i].next_start_time+
                    cell_table[source_type][i].t;
```

```
            if (cell_table[source_type][i].next_start_time>=cell_table[source_type][i].end_time)
            { update_table(source_type,i);}
          }
        }
      }
    }
    { int
      sum_cell,cell_sent_to_link,cell_lost;

      sum_cell=cell_in_buffer+cell_num;
      if (sum_cell<=mt)
      { cell_sent_to_link=sum_cell;
        cell_lost=0;
        cell_in_buffer=0;
      }
      else
      { cell_sent_to_link=mt;
        sum_cell=sum_cell-mt;
        if (sum_cell>BUFFER_SIZE)
        { cell_lost=sum_cell-BUFFER_SIZE;
          cell_in_buffer=BUFFER_SIZE;
        }
        else
        { cell_lost=0;
          cell_in_buffer=sum_cell;
        }
      }
      if (clock>=BEGIN_COUNT_CELL)
      {  total_num_cell_lost=total_num_cell_lost+cell_lost;
         total_num_cell=total_num_cell+cell_num;
         if (cell_num<NUM_SOURCE_2/2.0+NUM_SOURCE_1) count1=count1+cell_num;
         if (cell_num>NUM_SOURCE_2/2.0+NUM_SOURCE_1) count2=count2+cell_num;
      }
      clock=clock+T;
    }
}

void initiation_b1(void)
{
  b1[0]=1.0;
  for (int i=1;i<=BUFFER_SIZE;i++)
  b1[i]=0.0;
}
```

```
void b2_to_b1(void)
{
 for (int i=0;i<=BUFFER_SIZE;i++)
 b1[i]=b2[i];
}

double conditionl_p(int mt,int i,int j)
{int k;
 double aa;

 if (BUFFER_SIZE==0) {aa=1.0;}
  else
 { if (i==0)
    { aa=0.0;
      for (k=0;k<=mt-j;k++)
      {aa=aa+gro[k];}
    }
    else
    if (i==BUFFER_SIZE)
   { aa=1.0;
     for (k=0;k<=mt+BUFFER_SIZE-(j+1);k++)
     {aa=aa-gro[k];}
   }
   else
   { if (mt+i-j>=0)
     {aa=gro[mt+i-j];}
     else
     {aa=0.0;}
   }
 }
 return(aa);
}

void calculation_b2(void)
{ double sum;

 for (int i=0;i<=BUFFER_SIZE;i++)
 { b2[i]=0.0;
   for (int j=0;j<=BUFFER_SIZE;j++)
   { b2[i]=b2[i]+b1[j]*conditional_p(floor(mt),i,j);}
 }
}
```

```
double cell_lost_ratio_update(double epsilon)
{ double result,sum=1.0;
  int i=0;

  initiation_b1();
  while (sum>epsilon)
  { i++;
   calculation_b2();
   sum=0.0;
   for (int i=0; i<=BUFFER_SIZE; i++)
   {sum=sum+fabs(b2[i]-b1[i]);}
   b2_to_b1();
  }
  result=0.0;
  for (i=0;i<=BUFFER_SIZE;i++)
  {result=result+b2[i]*cell_loss_rate(floor(mt)+(BUFFER_SIZE-i)/related_parameter); }
  return(result);
}

double simulation(void)
{
  while (clock<SIMULATION_END_TIME)
  { read_table();}
  return(total_num_cell_lost/total_num_cell);
}

void pro_call(void)
{ double sum=0.0;

  for (int y=0;y<=NUM_SOURCE_1+NUM_SOURCE_2;y++)
  { printf("pro[%d]=%f ",y,pro[y]);
    sum=sum+pro[y];
  }
  printf("sum=%f\n",sum);
}

void fro_call(void)
{ double sum=0.0;

  for (int y=0;y<=NUM_SOURCE_1+NUM_SOURCE_2;y++)
  { printf("fro[%d]=%f ",y,fro[y]);
    sum=sum+fro[y];
  }
```

```
  printf("sum=%f\n",sum);
}

void gro_call(void)
{ double sum=0.0;

  for (int y=0;y<=NUM_SOURCE_1+NUM_SOURCE_2;y++)
  { printf("gro[%d]=%f ",y,gro[y]);
    sum=sum+gro[y];
  }
  printf("sum=%f\n",sum);
}

void b1_call(void)
{double sum=0.0;

 for (int y=0;y<=BUFFER_SIZE;y++)
 { printf("b1[%d]=%20.19f ",y,b1[y]);
   sum=sum+b1[y];
 }
 printf("sum=%f\n",sum);
}

void main(void)
{  int
  high[ARRAY_NUM+1],
  low_fundamental[ARRAY_NUM+1],
  low_propasal[BUFFER_NUM+1][ARRAY_NUM+1],
  low_simulation[BUFFER_NUM+1][ARRAY_NUM+1];
  double
  result,result_f,pre_result,eps=1.0e-5;
  int
  pre_low_num,i,j,ii,jj;
  FILE
  *fp,*fp1,*fp2,*fp3,*fp4,*fp5;

  if ((fp=fopen("\\shi_c\\report.dat","w"))==NULL)
  {  printf("Cannot open file!"); exit(1);}
  if ((fp1=fopen("\\shi_c\\sheet1.dat","w"))==NULL)
  {  printf("Cannot open file!"); exit(1);}
 if ((fp2=fopen("\\shi_c\\sheet2.dat","w"))==NULL)
 {  printf("Cannot open file!"); exit(1);}
 if ((fp3=fopen("\\shi_c\\sheet3.dat","w"))==NULL)
```

```
{  printf("Cannot open file!"); exit(1);}
if ((fp4=fopen("\\shi_c\\sheet4.dat","w"))==NULL)
{  printf("Cannot open file!"); exit(1);}
if ((fp5=fopen("\\shi_c\\sheet5.dat","w"))==NULL)
{  printf("Cannot open file!"); exit(1);}

printf("**********FINAL REPORT**********\n");
printf("The results of fundamental method, proposal method and simulation\n");
printf("**********\n");
printf("The cell loss rate required is %f.\n",eps);
printf("V is defined as the cell transmission speed of the multiplexed line.\n");
printf("Vp is peak cell rate and Va is average cell rate.\n");
printf("Ma is average burst length and Mi is average idle length.\n");
printf("The high speed connection Vp=V/10(%f),Va=Vp/10(%f),Ma=%f cells and Mi=%f cells\n",
 PEAK_CELL_RATE_1,AVERAGE_CELL_RATE_1,ACTIVE_MEAN_1,SILENCE_MEAN_1);
printf("The low speed connection Vp=V/100(%f),Va=Vp/2(%f),Ma=%f cells and Mi=%f cells\n",
 PEAK_CELL_RATE_2,AVERAGE_CELL_RATE_2,ACTIVE_MEAN_2,SILENCE_MEAN_2);
printf("The related_parameter is %f\n",related_parameter);
printf("method:  buffer_size high_num low_num cell_loss_ratio\n");

fprintf(fp,"**********FINAL_REPORT**********\n");
 fprintf(fp,"The results of fundamental method, proposal method and simulation\n");
 fprintf(fp,"**********\n");
fprintf(fp,"The cell loss rate required is %f.\n",eps);
fprintf(fp,"V is defined as the cell transmission speed of the multiplexed line.\n");
fprintf(fp,"Vp is peak cell rate and Va is average cell rate.\n");
fprintf(fp,"Ma is average burst length and Mi is average idle length.\n");
fprintf(fp,"The high speed connection Vp=V/10(%f),Va=Vp/10(%f),Ma=%f cells and Mi=%f cells\n",
PEAK_CELL_RATE_1,AVERAGE_CELL_RATE_1,ACTIVE_MEAN_1,SILENCE_MEAN_1);
fprintf(fp,"The low speed connection Vp=V/100(%f),Va=Vp/2(%f),Ma=%f cells and Mi=%f
cells\n",PEAK_CELL_RATE_2,AVERAGE_CELL_RATE_2,ACTIVE_MEAN_2,SILENCE_M
EAN_2);
fprintf(fp,"The related_parameter is %f\n",related_parameter);
fprintf(fp,"method:  buffer_size high_num low_num cell_loss_ratio\n");

mt=floor(V*T);
select_seed(seed_num);
for (i=1;i<=ARRAY_NUM;i++)
{ high[i]=(i-1)*2;
  low_fundamental[i]=80;
  for (j=1;j<=BUFFER_NUM;j++)
  { low_propasal[j][i]=80;
    low_simulation[j][i]=80;
```

```
  }
}
for (jj=ARRAY_NUM;jj>=1;jj--)
{  if (jj!=ARRAY_NUM)
   {low_fundamental[jj]=low_fundamental[jj+1];}
   result_f=0.0;
   while ( result_f<eps)
   { low_fundamental[jj]++;
     NUM_SOURCE_1=high[jj];
     NUM_SOURCE_2=low_fundamental[jj];
     initialization_pro_fro_gro( );
     result_f=cell_loss_rate(floor(mt));
   }
   for (ii=1;ii<=BUFFER_NUM;ii++)
   { switch (ii)
     { case 1:BUFFER_SIZE=0;break;
       case 2:BUFFER_SIZE=50;break;
       case 3:BUFFER_SIZE=100;break;
       case 4:BUFFER_SIZE=200;break;
       case 5:BUFFER_SIZE=300;break;
       default:BUFFER_SIZE=400;break;
     }
     printf("fundamental:%d      %d       %d      %20.19f\n",
                                  BUFFER_SIZE,high[jj],low_fundamental[jj],result_f);
     fprintf(fp,"fundamental:%d      %d       %d      %20.19f\n",
                                    BUFFER_SIZE,high[jj],low_fundamental[jj],result_f);

     if (ii!=1)
     { low_propasal[ii][jj]=low_propasal[ii-1][jj];}
     else
     { low_propasal[ii][jj]=low_fundamental[jj]-1;}
     result=0.0;
     while (result<eps)
     { low_propasal[ii][jj]++;
       NUM_SOURCE_1=high[jj];
       NUM_SOURCE_2=low_propasal[ii][jj];
       initialization_pro_fro_gro();
       result=cell_lost_ratio_update(1.0e-5);
      }
      printf("  proposal:%d      %d       %d      %20.19f\n",
                                  BUFFER_SIZE,high[jj],low_propasal[ii][jj],result);
      fprintf(fp,"  proposal:%d      %d       %d      %20.19f\n",
                                    BUFFER_SIZE,high[jj],low_propasal[ii][jj],result);
```

```
  if (ii!=1)
  { low_simulation[ii][jj]=low_simulation[ii-1][jj];}
  else
  { low_simulation[ii][jj]=low_fundamental[jj]-10;}
   result=0.0;
   while ( result<eps)
   { low_simulation[ii][jj]++;
     NUM_SOURCE_1=high[jj];
     NUM_SOURCE_2=low_simulation[ii][jj];
     pre_result=result;
     pre_low_num=NUM_SOURCE_2;
     if (result==0.0)
     { SIMULATION_END_TIME=50000.0;
       BEGIN_COUNT_CELL=10000.0;
     }
     else
     { SIMULATION_END_TIME=120000.0;
       BEGIN_COUNT_CELL=20000.0;
     }
     initialization(NUM_SOURCE_1,NUM_SOURCE_2);
     result=simulation();
   }
   if (result>=eps*10.0&& pre_result!=0.0)
   { result=pre_result;
     low_simulation[ii][jj]=pre_low_num;
   }
   printf(" simulation:%d      %d        %d       %20.19f\n",
                                  BUFFER_SIZE,high[jj],low_simulation[ii][jj],result);
   fprintf(fp," simulation:%d      %d        %d       %20.19f\n",
                                  BUFFER_SIZE,high[jj],low_simulation[ii][jj],result);
   printf("\n");
   fprintf(fp,"\n");
  }
}
fclose(fp);

for (jj=ARRAY_NUM;jj>=1;jj--)
{
    for (ii=1;ii<=BUFFER_NUM;ii++)
    { switch (ii)
      { case 1:fprintf(fp1,"%f %f %f %f\n", high[jj]*1.0,(low_fundamental[jj]-1.0)/2.0,
                          (low_propasal[ii][jj]-1.0)/2.0, (low_simulation[ii][jj]-1.0)/2.0);
               break;
```

```
            case 2:fprintf(fp2,"%f %f %f %f\n",high[jj]*1.0,(low_fundamental[jj]-1.0)/2.0,
                                (low_propasal[ii][jj]-1.0)/2.0, (low_simulation[ii][jj]-1.0)/2.0);
                    break;
            case 3:fprintf(fp3,"%f %f %f %f\n",high[jj]*1.0,(low_fundamental[jj]-1.0)/2.0,
                                (low_propasal[ii][jj]-1.0)/2.0, (low_simulation[ii][jj]-1.0)/2.0);
                    break;
            case 4:fprintf(fp4,"%f %f %f %f\n",high[jj]*1.0,(low_fundamental[jj]-1.0)/2.0,
                                (low_propasal[ii][jj]-1.0)/2.0, (low_simulation[ii][jj]-1.0)/2.0);
                    break;
            case 5:fprintf(fp5,"%f %f %f %f\n",high[jj]*1.0,(low_fundamental[jj]-1.0)/2.0,
                                (low_propasal[ii][jj]-1.0)/2.0, (low_simulation[ii][jj]-1.0)/2.0);
                    break;
            default:break;
           }
        }
    }
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
    fclose(fp4);
    fclose(fp5);
}
```

B.3 The Sample Output

```
**********FINAL REPORT**********
The results of fundamental method, proposal method and simulation
************************************************************
The cell loss rate required is 0.000010.
V is defined as the cell transmission speed of the multiplexed line.
Vp is peak cell rate and Va is average cell rate.
Ma is average burst length and Mi is average idle length.
The high speed connection Vp=V/10(10.000000),Va=Vp/10(1.000000),Ma=100.000000 cells and
Mi=900.000000 cells
The low speed connection Vp=V/100(1.000000),Va=Vp/2(0.500000),Ma=100.000000 cells and
Mi=100.000000 cells
The related_parameter is 28.000000
method:  buffer_size high_num low_num cell_loss_ratio
fundamental:0       10       83      0.0000107598152501633
proposal:   0       10       83      0.0000107598152501633
simulation: 0       10       87      0.0000177814717064818

fundamental:50      10       83      0.0000107598152501633
proposal:   50      10       85      0.0000107197825165720
simulation: 50      10       97      0.0000426038628922484

fundamental:100     10       83      0.0000107598152501633
proposal:   100     10       89      0.0000106524116347589
simulation: 100     10       107     0.0000322066504376570

fundamental:200     10       83      0.0000107598152501633
proposal:   200     10       97      0.0000105622195800664
simulation: 200     10       114     0.0000435924338438243

fundamental:300     10       83      0.0000107598152501633
proposal:   300     10       103     0.0000105248762224868
simulation: 300     10       127     0.0000115556046177827

fundamental:0       8        95      0.0000108712904860673
proposal:   0       8        95      0.0000108712904860673
simulation: 0       8        100     0.0000145637218659235

fundamental:50      8        95      0.0000108712904860673
proposal:   50      8        97      0.0000108727125672190
simulation: 50      8        109     0.0000188783085035581
```

fundamental:	100	8	95	0.0000108712904860673
proposal:	100	8	101	0.0000108836021225251
simulation:	100	8	113	0.0000297831554208824
fundamental:	200	8	95	0.0000108712904860673
proposal:	200	8	109	0.0000109367429323501
simulation:	200	8	121	0.0000111906206715311
fundamental:	300	8	95	0.0000108712904860673
proposal:	300	8	115	0.0000109987548202715
simulation:	300	8	138	0.0000752061523819280
fundamental:	0	6	108	0.0000111455508757365
proposal:	0	6	108	0.0000111455508757365
simulation:	0	6	115	0.0000103495167539165
fundamental:	50	6	108	0.0000111455508757365
proposal:	50	6	110	0.0000112112600274047
simulation:	50	6	119	0.0000587686018949609
fundamental:	100	6	108	0.0000111455508757365
proposal:	100	6	114	0.0000113473845243301
simulation:	100	6	130	0.0000152049790295051
fundamental:	200	6	108	0.0000111455508757365
proposal:	200	6	122	0.0000116403550505074
simulation:	200	6	137	0.0000203255096722505
fundamental:	300	6	108	0.0000111455508757365
proposal:	300	6	127	0.0000101737309082121
simulation:	300	6	145	0.0000227919715761643
fundamental:	0	4	122	0.0000109712535121492
proposal:	0	4	122	0.0000109712535121492
simulation:	0	4	124	0.0000145474588620089
fundamental:	50	4	122	0.0000109712535121492
proposal:	50	4	124	0.0000111280779554721
simulation:	50	4	134	0.0000233211477994600
fundamental:	100	4	122	0.0000109712535121492
proposal:	100	4	128	0.0000114469309472590

simulation: 100 4 141 0.0000206960852319979

fundamental:200 4 122 0.0000109712535121492
proposal: 200 4 135 0.0000101017829308828
simulation: 200 4 152 0.0000680079381325977

fundamental:300 4 122 0.0000109712535121492
proposal: 300 4 141 0.0000105502777039500
simulation: 300 4 157 0.0000231353484504464

fundamental:0 2 138 0.0000101401203274646
proposal: 0 2 138 0.0000101401203274646
simulation: 0 2 142 0.0000123120230402722

fundamental:50 2 138 0.0000101401203274646
proposal: 50 2 140 0.0000104764422162247
simulation: 50 2 147 0.0000178610824543641

fundamental:100 2 138 0.0000101401203274646
proposal: 100 2 144 0.0000111615195702014
simulation: 100 2 153 0.0000240743169040633

fundamental:200 2 138 0.0000101401203274646
proposal: 200 2 152 0.0000125901374002345
simulation: 200 2 162 0.0000379532590553916

fundamental:300 2 138 0.0000101401203274646
proposal: 300 2 157 0.0000109133974107948
simulation: 300 2 165 0.0000551242092751569

fundamental:0 0 159 0.0000110021970441394
proposal: 0 0 159 0.0000110021970441394
simulation: 0 0 159 0.0000141265566100887

fundamental:50 0 159 0.0000110021970441394
proposal: 50 0 161 0.0000118330969697107
simulation: 50 0 164 0.0000160056536893647

fundamental:100 0 159 0.0000110021970441394
proposal: 100 0 165 0.0000136140679860620
simulation: 100 0 168 0.0000165958852162195

fundamental:200 0 159 0.0000110021970441394

proposal:	200	0	172	0.00001287894769 10770
simulation:	200	0	173	0.0000286732870100452
fundamental:	300	0	159	0.0000110021970441394
proposal:	300	0	177	0.0000113484737455467
simulation:	300	0	178	0.0000279823728359818

Note: In these cases, N high speed connections offer N % load while M low speed connections offer M/2 % load.