University of Alberta

On The Design of High-Speed Switching Networks

by

Xiaotao Guo ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Spring 2001

0-612-60436-5

University of Alberta

Library Release Form

Name of Author: Xiaotao Guo

Title of Thesis: On The Design of High-Speed Switching Networks

Degree: Master of Science

Year this Degree Granted: 2000

Xiaotao Guo

Xiaotao Guo
Department of Computing Science
University of Alberta
Edmonton, AB
Canada, T6G 2E1

Date: Nov. 8, 2000

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **On The Design of High-Speed Switching Networks** submitted by Xiaotao Guo in partial fulfillment of the requirements for the degree of **Master of Science**.

[illegible]

Professor Ehab S. Elmallah

[illegible]

Professor Xian Liu

[illegible]

Professor Ioanis Nikolaidis

Date: Nov. 8, 2000

Abstract

Along with the growing number of network users and the emerging of new multimedia applications, today's communication networks are required to not only have higher bandwidth to satisfy the large amount of users, but also have the ability to provide integrated services in the same underlying backbone. Based on optical technologies, a revolution in telecommunication networks, optical networks provide higher capacity and reduce cost for new applications. As a dominating transfer mode in optical networks, *circuit-switching* deserves more attentions. On the other hand, *ATM* was designed as a major technical contribution for B-ISDN, and to have the ability of supporting a wide variety of services in a seamless manner. This thesis investigates the design issues of switching networks and the two important transfer modes: *circuit-switching* and *ATM*, with the goals of studying the blocking behavior for multicasting in circuit-switching environment and of measuring the performance of the existing switch designs and developing new methods to improve them.

Acknowledgements

I would like to express my gratitude to my supervisor Professor Ehab S. Elmallah for his support and guidance during these years.

I am also grateful to Professor Xian Liu and Professor Ioanis Nikolaidis for serving on my examine committee and for helping me to expend the scope of my research.

I am especially thanking my parents and my husband for their love and totally support over these years. Also, I'd like to give my thanks to all the people who helped me to make this thesis come true.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today's network applications require communication networks with increasingly higher bandwidth and the ability to provide integrated services in the same underlying backbone for heterogeneous traffic. The effort of supporting a combination of analog and digital traffic in the public switched telephone systems can be dated to 1960s. A significant outgrowth for this trend is the notion of *integrated services digital network* (ISDN). The ISDN is intended to be a worldwide public telecommunication network to replace the existing public telephone networks and deliver a wide variety of services. ISDN has already evolved from its first generation — narrowband ISDN — to its second generation — Broadband ISDN (B-ISDN).

In communication networks, switches form an important part that provides a facility for moving data from node to node until they reach their destinations. The basic function of a switch is to forward the data arriving at its inputs to the corresponding outputs while supporting control and management functions.

Transfer modes play a key role in switch designs. There are different transfer modes, ranging from the most static conventional circuit-switching to the most flexible packet switching. Each of them has its merits and disadvantages. In this section, we review the relevant characteristics of some important transfer modes.

In the early 1980s, a revolution in telecommunications networks began by the use of a relatively unassuming technology, fiber-optic cable. Based on optical technologies, optical networks provide higher capacity and reduce cost for new applications. The *circuit-switching* transfer mode is served as a dominating transfer mode in optical networks. Here, we first introduce the conventional *circuit-switching* and its two enhancements.

The conventional or pure *circuit-switching* has long been used in telephone networks. This transfer mode is based on TDM (time division multiplexing) principle. When there is a call request, a circuit is established for the complete duration of the connection. The information is transferred in the time slot assigned to this connection and will always use the same slot in the frame during the complete duration of the connection. Pure *circuit-switching* is the most static transfer mode and is unsuitable for different kinds of services.

Two enhancements of pure circuit-switching are *multirate circuit-switching* and *fast circuit-switching*. In *multirate circuit-switching*, a connection requests an integer multiple of some basic channel rate. Unlike pure circuit-switching, the technology of multirate circuit-switching is used to support applications with arbitrary rates, and different connections can share the capacity of the same link if the total bandwidth of these connections doesn't exceed the link capacity. Pure circuit-switching is a special case of 1-rate circuit-switching.

Multirate circuit-switching has some disadvantages. First, it makes the switching systems based on it more complex than those of pure circuit-switching. Another problem is the selection of the basic rate. In addition, multirate circuit-switching can not cope efficiently with sources with a bursty character.

In fast circuit-switching, the resources are only allocated when information is sent, and released when no information is sent. At call set-up, users request a connection with a bandwidth equal to some integer multiple of basic rate. The system will not allocate the resources, but store the information on the required bandwidth. When the source starts sending information, the switch allocates the necessary resources immediately.

The next transfer mode talked here is the *Asynchronous Transfer Mode* (ATM). ATM is designed for B-ISDN and potentially capable of supporting all classes of traffic in one transmission and switching fabric. It has the potential to subsume both the Internet and the telephone network, creating a unified infrastructure that carries voice, video, and data. Many applications, such as digital medical imaging, benefit from the flexibility in switching and high speed that ATM provides. The detailed review of the characteristics of ATM is in Chapter 5.

The most flexible transfer mode is *packet switching*. In packet switching network, user information is encapsulated in packets which contain additional information (in the header) used inside the network for routing, error correction, flow control, etc. This transfer mode is used in the Internet. It has some disadvantages such as delay insensitive and packet out-of-order. In order to offer an acceptable end-to-end performance on each link of the network, complex protocols are necessary. Along with the rapid growth of Internet applications, supporting real-time continuous traffic (e.g., digital audio and video) over the Internet is a fast growing direction. Four protocols used in the Internet are introduced below.

ST-II and *RSVP* are internetworking protocols for flow establishment. Both of them can support multicasting between multiple senders and heterogeneous receivers. They allow the receivers to *subscribe* or *unsubscribe* to *broadcast groups* dynamically. Also rate negotiation is very dynamic between the sender and the multiple receivers. One difference between these two protocols is that ST-II requires the sender to manage the flow, but RSVP allows each receiver to make resource reservation tailored to its needs, which is more effective for heterogeneous receivers. For these two protocols, routers not only take part in flow establishment, but also in monitoring and adjusting flow transmission. Also routers can communicate with each other and dynamically reroute flows and reload information about flows.

RTP and *RTCP* are used over other protocols to support continuous media applications in a transport-independent way. They don't assume (or expect) any sophisticated support from the network (especially, support from large-scale switches). As such, the use of such protocols emphasizes the need for the applications to adaptively change their transmission speeds.

1.1 Thesis Contributions and Organization

This thesis examines some architectures and routing algorithms that are used in the design of large-scale switching networks for high speed transport modes.

In one direction, because circuit-switching is the dominating transfer mode in optical networks, we examine the problem of blocking behavior for switching networks serving in circuit-switching environment. Our emphasis is on analyzing routing algorithms that support in-switch broadcast operations. To this end, we review the existing methods for computing end-to-end blocking in 3-stage Clos networks in Chapter 3. And in Chapter 4, we analyze the 2-call blocking behavior for 3-stage Clos networks in circuit-switching environment, under two in-switch multicast schemes: *second-stage* multicasting and *first-stage* multicasting. We also provide a preliminary comparison of these two schemes.

In another direction, we examine switch designs for supporting ATM cell switching. Some typical ATM switch designs are reviewed first in Chapter 5. The focus of this thesis is the AT&T growable ATM switch design. In Chapter 6, the worst-case scenario of this switch design is studied. The pattern for a class of *bad* traffic matrices is found and the algorithms used to generate this class of traffic matrices are proposed. This can help us to build benchmarks that are suitable for testing newly proposed improvements. Finally, some methods to improve the performance of this switch design are discussed.

The organization of this thesis is as follows. In Chapter 2, we provide the background knowledge about some important switching architectures. A generic switching architecture is first presented. Then a survey of typical switching architectures are followed. In Chapter 3, we introduce some previous works on the point-to-point call blocking behavior of Clos network. The focus of this chapter is Yang's models ([15]). In Chapter 4, based on Yang's models, we extend the studying of blocking behavior for Clos network to the 2-call blocking, where each 2-call has one source input module and two distinct destination output modules. Two schemes to multicast a 2-call are used in the analysis. The 2-call blocking models for the two schemes are presented. In Chapter 5, some background knowledge about ATM and ATM switch designs are given. First this chapter reviews the main characteristics of ATM. Then there is a survey of three typical ATM switch designs. In Chapter 6, we study the worst-case scenario of AT&T growable ATM switch design. This switch design uses a fast but sub-optimal routing algorithm: *straight matching*. Chapter 6 provides the results for the traffic type which has significant cell loss by using the straight matching algorithm, and also two algorithms to help generate benchmarks for worst-case traffic type. The last chapter, Chapter 7, concludes the work of this thesis and also discusses areas of future work.

Chapter 2

Background for Switching Architectures

In this chapter, we provide the background knowledge for switching architectures. First, a generic switching architecture is introduced in Section 2.1 to give an overall understanding of the architecture of a switch. Then a survey of typical switching architectures is provided in Section 2.2.

2.1 A Generic Switch Architecture

The basic function of a switch is to connect calls, route cells, or route packets from its input ports to its output ports. Figure 2.1 illustrates a generic architecture which consists of the following parts:

1. The **Switch Interface Unit**: The switch interface includes input modules and output modules. For ATM switches, the modules convert between SONET signals and ATM cell streams.

2. The **Switch Fabric Unit**: The switch fabric is primarily responsible for routing of calls or cells and possibly signaling and managing information as well.

3. The **Connection Admission Control Unit** (CAC): This part is in charge of establishing, modifying and terminating connections.

4. The **Switch Management Unit**: This part handles a variety of management functions, such as fault management, performance management, and so on. Also it collects and administrates management information, communicates with users and network managers, and supervises and coordinates all management activities.

When designing switches, some factors are taken into consideration: *complexity, scalability, management* or *control, speed* and *performance.*

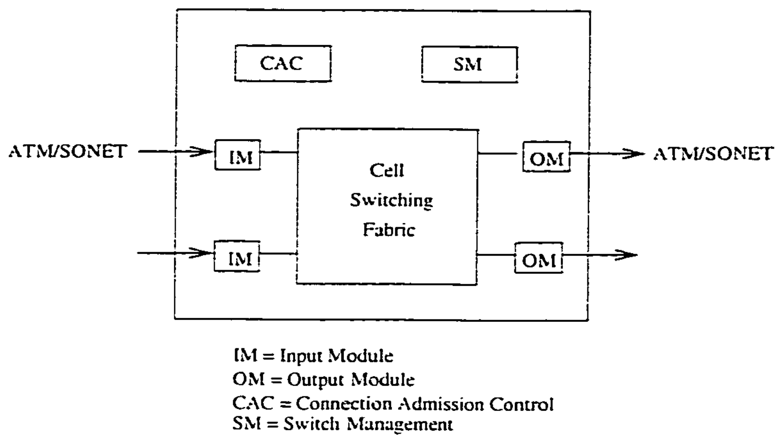


Figure 2.1: A Generic Switching Architecture

2.2 A Survey of Typical Switching Architectures

This section provides a survey of some typical switching architectures. There are different kinds of switching architectures. Each of them has its merits and shortcomings. Some of them are limited in scalability and suitable for small switching modules. such as *Shared-Memory*. *Shared-Medium* and *Crossbar*. For *multistage* fabric. it is easy to scale and can be used to construct large-scale networks.

In Section 2.2.1. the architectures of *Shared-Memory*. *Shared-Medium* and *Crossbar* are introduced. Then. different MINs (multistage interconnection networks) will be introduced in Section 2.2.2.

2.2.1 Some Small-Size Architectures

The architectures introduced here are *Shared-Memory*. *Shared-Medium* and *Crossbar*. They are limited in scalability and suitable for small size networks or switching modules.

- **Shared-Memory**: In this architecture. all the incoming data are converted from serial to parallel form and written sequentially to a common buffer pool. A memory controller controls the order in which data are read out of the memory according to the internal routing tags in the headers of the data.

  The main drawback of this approach is that it is not scalable due to the limitation on the memory access time. The speed requirement is also put on the processing speed of the memory controller. Three examples of this architecture are CNET's *Prelude* switch. Hitachi's shared buffer switch and AT&T's *GCNS-2000*.

- **Shared-Medium**: In this architecture. data are routed through a shared medium such as ring or bus. For example. in ATM environment. a cell arriving at an input port is broadcast to all output ports attached to the same medium. Each output port checks the incoming cell to decide if this cell is to it.

An attractive feature of this approach is that multicast and broadcast can be easily supported. But the speed requirement is placed on both the shared medium and the filters and buffers in the output ports. This places a physical limitation on the scalability. Some typical switches using this architecture are IBM's *PARIS* (Packetized Automated Routing Integrated System) and *plaNET*, NEC's *ATOM* (ATM Output Buffer Modular Switch), and Fore Systems' *ForeRunner ASX-100*.

Sometimes, multiple rings and multiple buses are used in a single or multiple hierarchical structure to increase the capacity. An experimental *SCPS* (Synchronous Composite Packet Switch) uses multiple rings to interconnect the switching modules.

- *Crossbar Switching Architecture*: Crossbar is the simplest fabric and can be implemented using multiple buses. Here, each input port is connected to each output port through a *crosspoint*. A crossbar is internally self-routing and nonblocking. However, the complexity and the cost grow quadratically with the number of ports. This drawback makes it not suitable for large switches. Two famous examples are *Knockout* Switch and *Gauss* Switch.

  For *Knockout* switch ([16]), there is a $L \times N$ concentrator in each output port, where N is the number of ports. If there are more than L cells destined to the same output port in one time cycle (slot), only L cells can be put into the buffer associated with that output port. This is called the *Knockout Principle*. Multicasting is easy to be implemented in *Knockout* switch, but significantly impacts the switch performance and increases the complexity of buffer control.

2.2.2 A Survey of Some Multistage Fabrics

Here, a number of multistage architectures are introduced. Large switches are usually built from smaller *modules* which can be constructed using the architectures introduced in the previous section.

- *The 3-Stage Clos Architecture*

  A Clos network has 3 stages (see Figure 2.2), and can be described by the parameters $N(m, n_1, n_2, k)$. The first stage contains k $(n_1 \times m)$ switch modules. Each of the m outgoing links in a first-stage module is connected to a switch module in the middle stage. The second stage contains m $(k \times k)$ switch modules. Each of the k outgoing links of a middle module is connected to a third stage switch module. The third stage contains k $(m \times n_2)$ modules. Each pair of input and output module has m paths. For symmetric architecture where $n_1 = n_2$, the Clos network can be represented as $N(m, n, k)$.

  Clos network has been widely used for data communications and parallel computing systems. It is also the focus of this thesis. In circuit-switching environment, the study

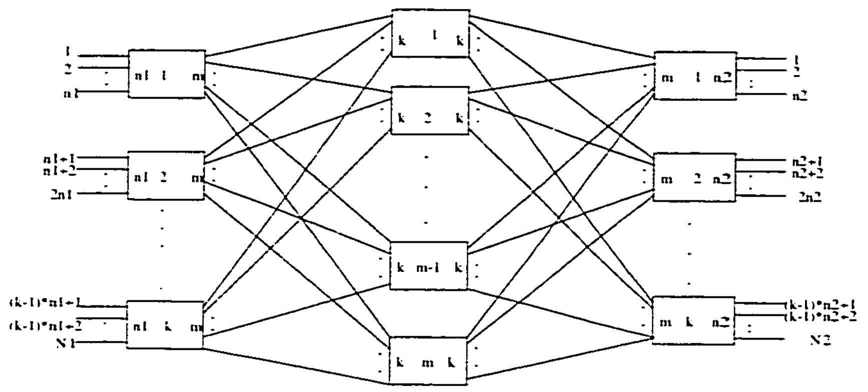


Figure 2.2: A 3-stage Clos Network

of Clos networks focuses on the blocking behavior. In the ATM environment, the study of Clos networks includes routing, performance evaluation and multicasting. Examples of ATM switch designs based on it include: the *AT&T growable switch* and the *Cross-path switch* (to be discussed later).

- *Batcher-Banyan Architecture*

  A Banyan network is *internal blocking*. In a Banyan switch, there is exactly one path from any input to any output. Self-routing can be implemented (in a distributed manner) by adding the destination output port number to the packet and letting each switch element to decide which way to go. Lee has shown that if the destination addresses of all the incoming packets are in strictly increasing or decreasing order, the internal blocking can be avoided ([5]).

  A Banyan network has less crosspoints than a Crossbar switch with the same input-output size. In a Batcher-Banyan network, a Batcher sorting network is used in front of a Banyan network to make all the packets in strictly increasing or decreasing order. Batcher-Banyan networks of large size are physically limited by the possible circuit density and number of input/output pins of the integrated circuits. Also supporting multicasting brings more complexity to the switch control. Examples of switch designs based on this fabric include: *Starlite* ([1]) and Bellcore's *Sunshine* ([9]).

- *The Benes and The Cantor Networks*

  The number of crosspoints in a 3-stage Clos network can be reduced by recursively replacing each module with a 3-stage subnetwork until all switch elements are (2×2). For $N = 2^n$, if we apply the above factorization, we get a $(2log_2N - 1)$-stage rearrangeable network, called the *Benes* network. The total number of crosspoints in a $N \times N$ Benes network is about $4Nlog_2N$ which is roughly the minimum number required for rearrangeable nonblocking networks ([5]). In a Benes network, there are multiple paths from one input to one output. Self-routing for a subset of all $N!$

permutations can be implemented on a Benes network.

For a *strictly nonblocking* network of size $N = 2^n$, if $F(2^n)$ is its crosspoint complexity, then $F(2^n) < N(log_2N)^{2.58}F(2)$ ([5]).

The Cantor network is strictly nonblocking with the complexity of roughly $4N(log_2N)^2$. A $N \times N$ Cantor network contains log_2N planes of $N \times N$ Benes networks. Each input link is connected to the log_2N Benes network planes through a $1 \times log_2N$ demultiplexer, while each output link is connected to the log_2N Benes planes through a $log_2N \times 1$ multiplexer.

Although the Benes and the Cantor networks are attractive in the crosspoint complexity, it is not easy to find fast routing and control algorithms for them.

- *Augmented Data Manipulator (ADM) Networks*

  ADM networks are blocking networks which have more permutation power than Banyan networks, with a slight increase in cost.

  In an ADM network with size $N = 2^m$, there are m stages and N switching elements (SEs) in each stage. Stages are numbered (from left to right) as $m-1, m-2, ..., 0$. Each SE has three input links and three output links. For a SE j at stage i, its first output is connected to node ($j - 2^i$ mod N) of stage $i-1$ (up), its second output to node j of stage $i-1$ (straight), and the third output to node ($j + 2^i$ mod N) of stage $i-1$ (down). For multicast and broadcast cells, a SE can connect its input link to two or all of its output links.

  If the stages of the ADM network are traversed in reverse order, the resulting network is called the IADM network. Both the ADM and IADM networks have multiple paths between any single source and any single destination, except that there is only one path between a source S and destination D when $S = D$. One important feature of ADM and IADM networks is that self-routing can be implemented in them for both unicasting and limited multicasting.

2.3 Concluding Remarks

In this chapter, we present the background knowledge of switching architectures. A generic architecture of a switch is introduced first. Then a survey of typical switch architectures are provided. Some of them are limited in scalability and suitable for small switching modules, such as *Shared-Memory*, *shared-medium* and *Crossbar*. On the other hand, the group of *multistage* architectures, including *Batcher-Banyan*, *Clos*, *Benes*, *Cantor*, *ADM*, are easy to scale and can be used to construct large-scale networks. Especially, *Clos* network is the main focus of this thesis.

Chapter 3

The Blocking Behavior for Unicast Calls in Clos Networks

In the first part of this thesis (Chapter 3 and Chapter 4), we consider approximation methods for computing blocking probability in unicast and multicast circuit switching. We are motivated by

- the importance of the circuit switching transfer mode in high-speed networking (e.g., for its use in optical networks),
- the importance of 3-stage Clos architecture in constructing large-scale switching systems, and
- the growing demand of supporting broadcast inside the switch.

The Clos networks have been widely used and extensively studied. In the circuit-switching environment, call blocking is the primary measure of main focus. The work in this direction can be generally categorized into two classes: deterministic and probabilistic.

In deterministic analysis, the main focus is on finding the minimum number of middle stage modules, given the relevant parameters of the first stage modules, such that the network is nonblocking for arbitrary connection requests. Clos showed ([2]) that a 3-stage Clos network is *strictly non-blocking* if $m \geq n_1 + n_2 - 1$, where n_1 is the number of input links per input module, and n_2 is the number of output links per output module. For a symmetric 3-stage Clos network, it is *strictly non-blocking* if $m \geq 2n - 1$, where $n_1 = n_2 = n$. If the network satisfies this condition, any middle switch module may be chosen arbitrarily to make a connection without re-arranging the existing connections. Furthermore, Benes proved that the network is *rearrangeable nonblocking* if $m \geq n_1$ ([14]). If the network satisfies this condition, any call can be connected by re-arranging the existing connections.

In the probabilistic study, the main focus is on analyzing the blocking behavior of Clos network that is not strictly nonblocking. Unfortunately, exact computation of blocking appears to be a very hard problem. Therefore, extensive studies in the literature appears on approximating the blocking probability measure. Of these studies, several previous analytical models have been proposed on approximating the blocking probability of Clos network for unicast or point-to-point calls. In particular, some approximation methods can not get a zero blocking probability even if the strictly nonblocking condition is satisfied.

Our goal in this chapter is to give an overview of the techniques used for approximating the end-to-end blocking. We emphasize a recent algorithm introduced in [15]: in this paper, Yang and Kessler presented an analytical model for the unicast call blocking probability of the Clos network that can more accurately describe the blocking behavior and is consistent with the deterministic nonblocking condition.

The organization of this chapter is as the follows. Section 3.1 reviews the first pioneering model — Lee's model. Yang's model is described in Section 3.2.

3.1 Lee's Blocking Model

In this section, we give a brief description of Lee's model. As noted by many authors, the problem of computing blocking probability is very difficult, hence the approximation algorithms are needed to be developed. Lee pioneered the first approximation model in [3]. Lee's model is simple. However, the blocking probability can not reach zero when the strictly nonblocking condition is satisfied, by using this model.

Before describing Lee's model, let's first introduce some concepts and assumptions which are used both here and in the rest of the thesis:

- *traffic assumption*: A frequently used circuit-switching model assumes that connection requests arrive according to an exponential distribution with average interarrival time of $\frac{1}{\lambda}$, and that each request has an exponential duration with average $\frac{1}{\mu}$. Let a be the probability that a typical input link is busy (that is just the parameter $\frac{\lambda}{\lambda+\mu}$), and assume a homogeneous system where each input link is busy with the same probability a. In the case where the switch has the same number of input and output links, this then means that a typical output link is busy with the same probability a.

- *Random routing*: The switch is controlled by a routing algorithm that chooses any available path randomly.

- *interstage link*: we call a link between the first stage and the middle stage an *input-middle* interstage link, and a link between the middle stage and the third stage a *middle-output* interstage link.

- *link independence*: the events that individual links are busy are assumed to be independent.

- *network configuration*: The 3-stage Clos network considered here is $N(n, m, k)$. There are k $n \times m$ (or $m \times n$) input (or output) modules, and m $k \times k$ middle stage modules.

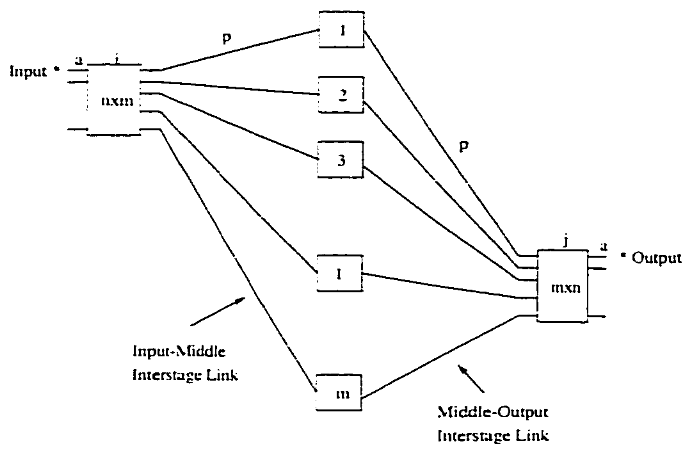


Figure 3.1: The paths between a given input and output pair in the Clos Network

Let's consider a point-to-point call request from an input link of input module i to an output link of output module j (see Figure 3.1). Clearly, at most $n-1$ input-middle interstage links from input module i can be already busy, and at most $n-1$ middle-output interstage links to output module j can be already busy. Also there are m disjoint paths connecting input module i and output module j. Assume that the incoming traffic is uniformly distributed over the m interstage links, then the probability that an interstage link is busy is

$$p = \frac{an}{m}.$$

Also, denote the probability that an interstage link is idle by $q = 1 - p$.

If a path can be used to connect the new call request, then the input-middle interstage link and the middle-output interstage link in this path should be both idle. Thus the probability that a path can not be used is $(1-q^2)$. If all the m paths can not be used, then the call is blocked. With the link independence assumption, the probability that a call is blocked is approximated by

$$P_B = (1-q^2)^m.$$

This is Lee's model for approximation P_B. In some cases, it may not be very accurate. For example, we know that at most n out of m output links of an input module can be busy. Lee's model doesn't account for this. Therefore, Lee's model does not yield a zero blocking probability for strictly nonblocking networks. For example, if we set $n = 32$, $m = 2n - 1 = 63$, and $a = 1$, Lee's model yields

$$P_B = 2.5 \times 10^{-8}.$$

A more accurate model was provided by C. Jacobaeus ([6]), in which, the input limitation is taken into account automatically, and the blocking probability of the three-stage Clos network is calculated by the following formula:

$$P_B = \frac{(n!)^2(2-a)^{2n-m}a^m}{m!(2n-m)!}.$$

Jacobaeus' formula is still an approximation to the blocking probability. If we use the above example, where $n = 32$, $m = 2n - 1 = 63$, and $a = 1$, Jacobaeus' model yields

$P_B = 3.4 \times 10^{-20}$.

3.2 Yang's Blocking Model

In [15], Yang and Kessler proposed an improved analytical model for approximating the unicast blocking probability of Clos networks. For networks where $m > n$, for example, Yang's model takes into account the fact that at most n out of the m links incident to each module can be busy. This model can be used for random routing strategy, and also can be easily extended to packing strategy.

3.2.1 The Blocking Model for Random Strategy

In this section, Yang's blocking model for random routing strategy is introduced. Similar to Lee's model, we consider a call request from input module i to output module j (see Figure 3.1). Assume n_1 input-middle interstage links from input module i are busy and n_2 middle-output interstage links to output module j are busy, where $0 \le n_1, n_2 \le n-1$. Let $\mathbf{n_1}$ denote the event that there are n_1 busy input-middle interstage links from input module i, and $\mathbf{n_2}$ denote the event that there are n_2 busy middle-output interstage links to output module j. If a busy input-middle interstage link and a busy middle-output interstage link share the same middle stage module, then this pair of links is said to be *overlapping*. An important counting argument that is used here, and also in Chapter 4 is:

Lemma 3.1 ([15]): Given events $\mathbf{n_1}$ and $\mathbf{n_2}$, the probability that k pairs of interstage links are overlapping in Clos network is given by

$$\Pr\{k \text{ pairs of links overlapping } |\mathbf{n_1}, \mathbf{n_2}\} = \frac{\binom{n_1}{k}\binom{m-n_1}{n_2-k}}{\binom{m}{n_2}} = \frac{\binom{n_2}{k}\binom{m-n_2}{n_1-k}}{\binom{m}{n_1}}$$

Proof: As we know, there are a total of $\binom{m}{n_1}\binom{m}{n_2}$ ways to choose n_1 busy input-middle interstage links and n_2 busy middle-output interstage links. Assume k pairs of interstage links are overlapping. Now we consider the following construction:

1. there are $\binom{m}{n_1}$ ways to choose n_1 busy input-middle interstage links;

2. then $\binom{n_1}{k}$ ways to choose k input-middle interstage links which are overlapping with k middle-output interstage links;

3. finally, the rest of $n_2 - k$ busy middle-output interstage links in output module j correspond to the $m - n_1$ non-busy input-middle interstage links from input module i, and there are $\binom{m-n_1}{n_2-k}$ ways to choose them.

Thus, the probability that k pairs of links are overlapping is

$$\frac{\binom{m}{n_1}\binom{n_1}{k}\binom{m-n_1}{n_2-k}}{\binom{m}{n_1}\binom{m}{n_2}} = \frac{\binom{n_1}{k}\binom{m-n_1}{n_2-k}}{\binom{m}{n_2}}$$

Symmetrically, the probability can also be

$$\frac{\binom{n_2}{k}\binom{m-n_2}{n_1-k}}{\binom{m}{n_1}}$$

■

A connection request under consideration is not blocked if and only if there exists one path in which both the input-middle interstage link and the middle-output interstage link are not busy, that is, if and only if

$$n_1 + n_2 - k < m,$$

which implies

$$k \geq max\{0, n_1 + n_2 - m + 1\}.$$

On the other hand, $k \leq min\{n_1, n_2\}$. Thus,

$$\Pr\{\text{connection not blocked } |\mathbf{n_1}, \mathbf{n_2}\}$$
$$= \sum_{k=max\{0,n_1+n_2-m+1\}}^{min\{n_1,n_2\}} \Pr\{k \text{ pairs overlapping } |\mathbf{n_1}, \mathbf{n_2}\}$$
$$= \sum_{k=max\{0,n_1+n_2-m+1\}}^{min\{n_1,n_2\}} \frac{\binom{n_1}{k}\binom{m-n_1}{n_2-k}}{\binom{m}{n_2}}$$

On the other hand, under the assumption of link independence, we know that the events $\mathbf{n_1}$ and $\mathbf{n_2}$ are independent, which gives

$$\Pr\{\mathbf{n_1}, \mathbf{n_2}\} = \Pr\{\mathbf{n_1}\}\Pr\{\mathbf{n_2}\}$$

To compute $\Pr\{\mathbf{n_1}\}$ and $\Pr\{\mathbf{n_2}\}$, we use $p = \frac{an}{m}$, the probability that an interstage link is busy, and $q = 1 - p$, the probability that an interstage link is idle (as in Lee's model).
By the independence assumption, the number of busy input-middle interstage links (or middle-output interstage links) follows the binomial distribution, that is, the probability that n_1 input-middle interstage links are busy is $\binom{m}{n_1} p^{n_1} q^{m-n_1}$. Given the fact that there are at most $n - 1$ busy input-middle interstage links from input module i, we get

$$\Pr\{\mathbf{n_1}\} = \frac{\binom{m}{n_1} p^{n_1} q^{m-n_1}}{\sum_{j=0}^{n-1}\binom{m}{j} p^j q^{m-j}}, \text{ and, } \Pr\{\mathbf{n_2}\} = \frac{\binom{m}{n_2} p^{n_2} q^{m-n_2}}{\sum_{j=0}^{n-1}\binom{m}{j} p^j q^{m-j}}.$$

Thus, the probability that a connection request is not blocked is given by

$$\Pr\{\text{connection not blocked}\}$$
$$= \sum_{n_1=0}^{n-1}\sum_{n_2=0}^{n-1}\Pr\{\text{connection not blocked } |\mathbf{n_1}, \mathbf{n_2}\}\Pr\{\mathbf{n_1}\}\Pr\{\mathbf{n_2}\}$$
$$= \frac{\sum_{n_1=0}^{n-1}\sum_{n_2=0}^{n-1}\sum_{k=max\{0,n_1+n_2-m+1\}}^{min\{n_1,n_2\}}\binom{m}{n_1}\binom{n_1}{k}\binom{m-n_1}{n_2-k} p^{n_1+n_2} q^{2m-n_1-n_2}}{\left(\sum_{j=0}^{n-1}\binom{m}{j} p^j q^{m-j}\right)^2}$$

and the blocking probability is

$$P_B = 1 - \Pr\{\text{connection not blocked}\}$$

The above P_B reaches zero when the number of middle stage modules $m \geq 2n - 1$. The complete proof for this latter property contains some complex technical details (omitted in this outline) and can be found in [15].

3.2.2 Concluding Remarks

In [15], the blocking model has also been extended to routing using a *packing strategy*, whereby an empty middle stage module is not used for routing a new request unless there is no partially filled middle stage module that can satisfy that particular connection request. It is generally believed (from numerical results, using different traffic patterns) that packing can lower the blocking probability of the network.

The analysis for the packing strategy is generally similar to the random strategy. But there are some differences between them:

- It is assumed that a certain number d ($d \leq n - 1$) of middle stage modules are fully-packed, that is, all interstage links to or from these middle stage modules are busy. The incoming traffic is thus uniformly distributed over the $m - d$ interstage links to those middle stage modules which are not fully-packed. This results in the probability that an interstage link is busy is
$$p = \begin{cases} 1, & \text{for the } d \text{ links to the fully-packed middle stage modules} \\ \frac{an-d}{m-d}, & \text{for the } m-d \text{ links to the non fully-packed middle stage modules} \end{cases}$$
- Instead of using events $\mathbf{n_1}$ and $\mathbf{n_2}$, here events $\mathbf{n'_1}$ and $\mathbf{n'_2}$ are used to denote that there are $n'_1 = n_1 - d$ busy input-middle interstage links to the non fully-packed middle stage modules from input module i and $n'_2 = n_2 - d$ busy middle-output interstage links from the non fully-packed middle stage modules to output module j.

In [15], both analytical and simulation results are presented. The models for the random routing strategy and packing strategy are both compared with Lee's model and Jacobaeus' model. The simulation results for the random strategy yield a lower blocking probability than the analytical model.

Chapter 4

The Blocking Behavior for 2-calls in Clos Networks

In Chapter 3, we have reviewed some previous work on approximating point-to-point call blocking probability of Clos networks. In this chapter, we extend this direction further by studying multicast call blocking behavior of Clos networks.
The organization of this chapter is as the follows. In Section 4.1, some previous work about the multicast call blocking is reviewed. In this section, three routing schemes for multicasting are introduced. Also, some notations and assumptions which are used in the later of this chapter are presented here. This thesis focuses on two routing schemes. So, in Section 4.2 and Section 4.3, the blocking models for these two schemes are presented separately. Some simulation and analytical results are provided in Section 4.4.

4.1 Previous Works

In [4], a multicast call from an input module to k distinct output modules is termed a k-call. In particular, a point-to-point call is also known as a 1-call. In this work ([4]), the deterministic non-blocking and rearrangeable non-blocking conditions for a k-call are derived. This chapter focuses on the studying of 2-call blocking behavior in Clos networks. As we know, a Clos network consists of switching modules. If a Clos network can route multicast calls, then some or all of the switching modules may have the *fan-out* feature, i.e., each of the input links on one module can be connected to any number of this module's output links. So a *multicast* switch module is a module that has *fan-out* feature, while a switch module is called an *unicast* switch module if an input link can be connected to only one of its output links at one time. Generally, three basic routing schemes are used for connecting multicast calls in a 3-stage Clos network, according to the placement of *multicast* switch modules:

- **first-stage multicast**: This scheme assumes that all modules in the input stage are multicast modules. But the middle stage contains only unicast modules. So a multicast call can be broadcast through the input stage. For example, for a 2-call from input module i to two output module j and k, two paths should be set up, where one is: input module i — a specified middle stage module — output module j; and the other is: input module i — another specified middle stage module — output module k. Due to the unicast feature of middle stage modules, we can not use one middle stage module to connect one input module to two different output modules. So here in the above example, two different middle stage modules are used to route this 2-call. Figure 4.1 is an example of this multicasting scheme.

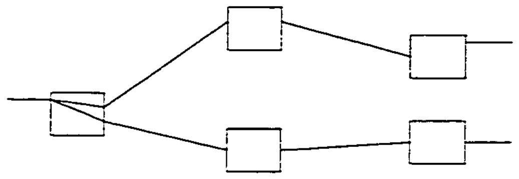


Figure 4.1: First-Stage Multicasting Scheme

- **second-stage multicast**: This scheme assumes that all the switch modules in the middle stage are multicast modules. But the first stage contains only unicast modules and doesn't have multicast feature. So we can multicast a call through the middle stage. Here, let's use the above example. So there are two paths from input module i to output module j and output module k. One path is: input module i — a specified middle stage module — output module j; while the another one is: input module i — the same middle stage module — output module k. Since the first stage can not make multicast, only one middle stage module can be used. So the two paths share the same middle stage module. Figure 4.2 is an example of this multicasting scheme.

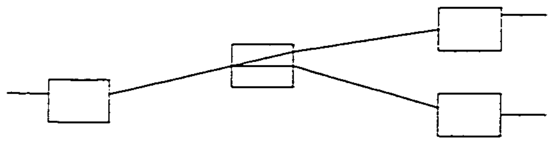


Figure 4.2: Second-Stage Multicasting Scheme

- **first-and-second-stage multicast**: The third scheme assumes that both the first and the second stage can do broadcasting. This can split a multicast call at the first stage into sub-calls with lower fan-out. It is very flexible, but the efficiency of this method depends on the splitting algorithm. Figure 4.3 is an example of this multicasting scheme. This scheme is not discussed in this chapter.

In this chapter, we will study the 2-call blocking behavior in a Clos network, under the first two routing schemes: **first-stage multicast** and **second-stage multicast**. The following are some assumptions used in the rest of this chapter:

- In this chapter, the Clos network has d input and output modules, m middle stage modules, and n input or output links on each input or output module.

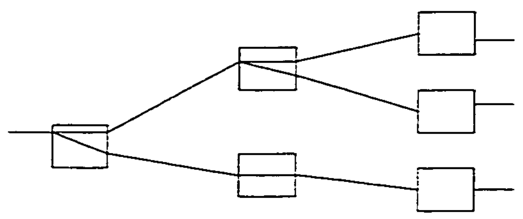


Figure 4.3: First-and-Second-Stage Multicasting Scheme

- The link independence assumption used in Chapter 3 is also used here. Also, same as Chapter 3, the probability that a typical input link of one input module is busy is a, and the probability that a typical output link of one output module is busy is also a. p is used to denote the probability that an interstage link is busy.

- Throughout the thesis, we assume that a 2-call is from one input module to two distinct output modules. For a call from one input module to two output links on the same output module, we assume that the output module can broadcast the call. So only one path is needed to be set up. This kind of calls are treated as unicast calls in this thesis.

- For an input module i, we use a vector A_i to represent the status of all the input-middle interstage links from it. Similarly, for an output module j, a vector B_j is used to represent the status of all the middle-output interstage links to it. Each element in a status vector corresponds to an interstage link. If one interstage link is busy, then the corresponding element in the vector is 1; otherwise, it is 0. All the status vectors have the same size m, which is the number of middle stage modules.

- As we know from Chapter 3, an available path from an input module to an output module consists of two interstage links which share the same middle stage module. So in the two vectors for the input module and the output module, the elements corresponding to the idle input-middle interstage link and the idle middle-output interstage link in this available path are both 0. We call this a *matching* between the two vectors. See Figure 4.4. Each matching corresponds to an available path. So in Figure 4.4, we can see, between input module i and output module j, there is an available path through middle stage module 1 and another available path through middle stage module $m - 1$.

  A connection or a path between an input module and an output module can not be set up if there is no matching between their status vectors.

- In the rest of this chapter, we only consider random routing strategy. If there are more than one middle stage module can be selected for one connection, one of them is selected randomly.

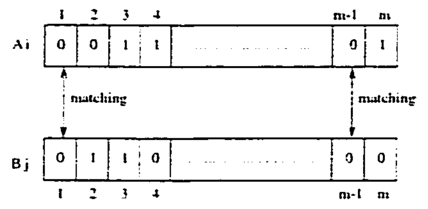


Figure 4.4: Matchings between Input Module i and Output Module j

4.2 The Blocking Behavior for Second-Stage Multicast Scheme

In this section, we will analyze the 2-call blocking behavior under the *second-stage multicast* scheme. As we know, this scheme doesn't take advantage of the fan-out feature in the first stage. So for each 2-call, only one middle stage module is used to connect the source and the two destinations.

Assume a 2-call from input module i to output modules j and k. Let's use A_i, B_j and B_k to represent the status of the three modules. If there is a middle stage l which can connect this 2-call, then the following three interstage links must be idle:

- the input-middle interstage link between input module i and middle stage module l, which is corresponding to $A_i[l]$.
- the middle-output interstage link between middle stage module l and output module j, which is corresponding to $B_j[l]$.
- the middle-output interstage link between middle stage module l and output module k, which is corresponding to $B_k[l]$.

So we can get $A_i[l] = B_j[l] = B_k[l] = 0$. We call this a matching *among* these three vectors. If there is a matching *among* the source vector and the two destination vectors, that means there is an available middle stage module which can be used to connect the input module to the two output modules. Figure 4.5 shows two matchings among A_i, B_j and B_k. Therefore, a 2-call from input module i to output modules j and k can be connected through either middle stage module 3 or $m-1$.

4.2.1 The Probability for Matchings

Before we provide the 2-call blocking probability, we first try to get the probability of matchings *among* three vectors.

Assume there is a 2-call from input module i to output modules j and k. The network state is shown in Figure 4.6, in which r input-middle interstage links from input module i are busy, s_1 middle-output interstage links to output module j are busy, and s_2 middle-output interstage links to output module k are busy.

If we use A_i, B_j and B_k to represent the state of input module i and output modules j and k, we can get that there are r 1's in A_i, s_1 1's in B_j and s_2 1's in B_k, where $r, s_1, s_2 \leq n-1$.

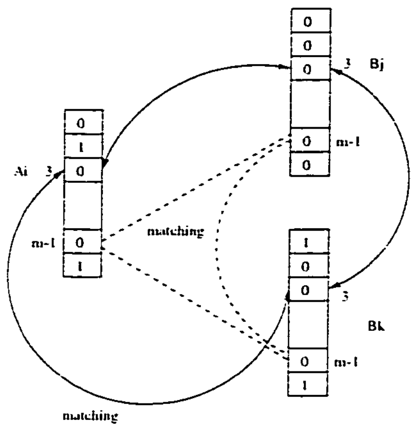


Figure 4.5: Matchings among three vectors

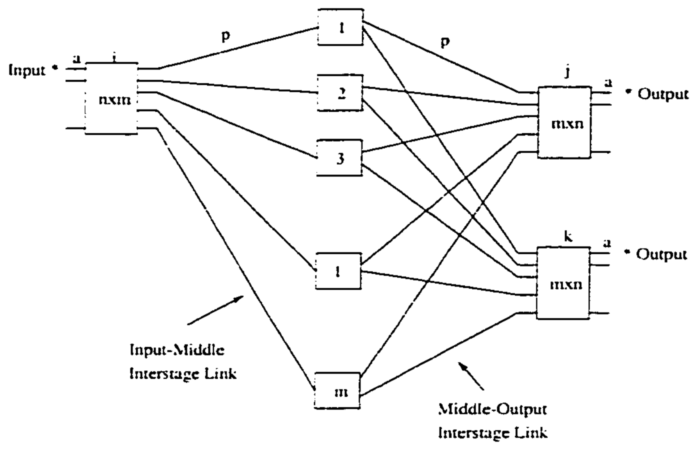


Figure 4.6: The state of the network with a call request from input module i to output modules j and k

Let $\mathbf{r}$ denote the event that there are r 1's in A_i, $\mathbf{s_1}$ denote the event that there are s_1 1's in B_j, and $\mathbf{s_2}$ denote the event that there are s_2 1's in B_k.

The analysis here uses the counting argument in Lemma 3.1 in Section 3.2.1, with the following difference: in Lemma 3.1, the analysis focuses on the probability of fully-used paths (both input-middle interstage link and middle-output interstage link are busy), while here, the analysis is about the probability of matchings or available paths (both input-middle interstage link and middle-output interstage link are idle). So we get the following Corollary 4.1:

Corollary 4.1: Let $\mathbf{x}$ denote the event that there are x 1's in one source vector A, and $\mathbf{y}$ denote the event that there are y 1's in one destination vector B. Given the events $\mathbf{x}, \mathbf{y}$, the probability that A and B have c matchings is

$$\Pr\{c \text{ matchings } |\mathbf{x},\mathbf{y}\} = \frac{\binom{m-x}{c}\binom{x}{m-y-c}}{\binom{m}{m-y}}.$$

On the other hand, let $\mathbf{x}'$ denote the event that there are x' 0's in one source vector A, and $\mathbf{y}'$ denote the event that there are y' 0's in one destination vector B. Given the events $\mathbf{x}', \mathbf{y}'$, the probability that A and B have c matchings is

$$\Pr\{c \text{ matchings } |\mathbf{x}',\mathbf{y}'\} = \frac{\binom{x'}{c}\binom{m-x'}{y'-c}}{\binom{m}{y'}}. \qquad \blacksquare$$

Now, given the events $\mathbf{r}, \mathbf{s_1}, \mathbf{s_2}$, we present the method of computing the nonblocking probability $\Pr\{\text{connection not blocked } |\mathbf{r}, \mathbf{s_1}, \mathbf{s_2}\}$:

- First, we assume that there are c_1 matchings among B_j and B_k, given $\mathbf{s_1}$ and $\mathbf{s_2}$. Based on Corollary 4.1, we get

$$\Pr\{c_1 \text{ matchings } |\mathbf{s_1},\mathbf{s_2}\} = \frac{\binom{m-s_1}{c_1}\binom{s_1}{m-s_2-c_1}}{\binom{m}{m-s_2}}.$$

- Then, by using the bitwise OR operation to the vector B_j and vector B_k, we get a new vector D. The method of combining is: for any $1 \le l \le m$, if both $B_j[l]$ and $B_k[l]$ equal to 0, then $D[l]$ will be 0; otherwise, $D[l]$ is 1. So D will have c_1 0's. Figure 4.7 shows this combining B_j and B_k into D.

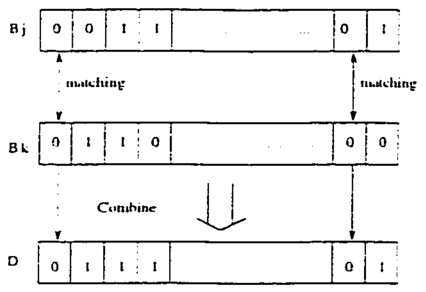


Figure 4.7: Combining B_j and B_k to get D

- Let $\mathbf{c_1}$ denote the event that there are c_1 0's in D. Then, we assume that there are c_2 matchings between A_i and D. These c_2 matchings are also matchings among the three vectors. Given events $\mathbf{r}$ and $\mathbf{c_1}$, from Corollary 4.1, the probability that A_i and D have c_2 matchings is

$$\Pr\{c_2 \text{ matchings } |\mathbf{r},\mathbf{c_1}\} = \frac{\binom{m-r}{c_2}\binom{r}{c_1-c_2}}{\binom{m}{c_1}}.$$

From the above analysis, we will extract the 2-call blocking probability for the second-stage multicasting scheme in the next section.

4.2.2 Blocking Probability

As we can see, the blocking of a 2-call happens when there is no matching among the three modules. Given events $\mathbf{r}$, $\mathbf{s_1}$ and $\mathbf{s_2}$, the 2-call under consideration is not blocked if and only if there exists at least one matching among A_i, B_j and B_k.

As in Section 4.2.1, assume that there are c_1 matchings between B_j and B_k and there are c_2 matchings between A_i and the vector D by combining B_j and B_k. So that means there are c_2 matchings among the three vectors. In order to make the call non-blocking, $c_1, c_2 \geq 1$. When (s_1+s_2) is less than m, there will be at least $(m-s_1-s_2)$ matchings between B_j and B_k. So $c_1 \geq \max\{1, m-s_1-s_2\}$. On the other hand, obviously the number of matchings cannot exceed either $(m-s_1)$ or $(m-s_2)$. So the condition for c_1 to make the call not blocked is $\max\{1, m-s_1-s_2\} \leq c_1 \leq \min\{m-s_1, m-s_2\}$. Similarly, we can get the conditions for c_2 : $\max\{1, c_1-r\} \leq c_2 \leq \min\{m-r, c_1\}$.

Therefore, the probability that the connection is not blocked, given events $\mathbf{r}$, $\mathbf{s_1}$ and $\mathbf{s_2}$, is given by

$$\Pr\{\text{connection not blocked} \mid \mathbf{r}, \mathbf{s_1}, \mathbf{s_2}\}$$
$$= \sum_{c_1=max\{1,m-s_1-s_2\}}^{min\{m-s_1,m-s_2\}} \Pr\{c_1 \text{ matchings} \mid \mathbf{s_1}, \mathbf{s_2}\} \sum_{c_2=max\{1,c_1-r\}}^{min\{m-r,c_1\}} \Pr\{c_2 \text{ matchings} \mid \mathbf{r}, \mathbf{c_1}\}$$
$$= \sum_{c_1=max\{1,m-s_1-s_2\}}^{min\{m-s_1,m-s_2\}} \frac{\binom{m-s_1}{c_1}\binom{s_1}{m-s_2-c_1}}{\binom{m}{m-s_2}} \sum_{c_2=max\{1,c_1-r\}}^{min\{m-r,c_1\}} \frac{\binom{m-r}{c_2}\binom{r}{c_1-c_2}}{\binom{m}{c_1}}.$$

Under the assumption of link independence, we know that the events $\mathbf{r}$, $\mathbf{s_1}$ and $\mathbf{s_2}$ are independent, which gives

$$\Pr\{\mathbf{r}, \mathbf{s_1}, \mathbf{s_2}\} = \Pr\{\mathbf{r}\}\Pr\{\mathbf{s_1}\}\Pr\{\mathbf{s_2}\}.$$

Because the first stage does not do multicasting, the calculation of $\Pr\{\mathbf{r}\}$ is the same as the calculation for $\Pr\{\mathbf{n_1}\}$ and $\Pr\{\mathbf{n_2}\}$ in Section 3.2.1. p is the probability that an interstage link is busy, while $q = 1-p$. Since we only consider random selection of available paths, the calculation of p is $\frac{an}{m}$ (as what in Chapter 3). So we get

$$\Pr\{\mathbf{r}\} = \frac{\binom{m}{r} p^r q^{m-r}}{\sum_{j=0}^{n-1} \binom{m}{j} p^j q^{m-j}}.$$

Furthermore, the calculation of $\Pr\{\mathbf{s_1}\}$ and $\Pr\{\mathbf{s_2}\}$ can be obtained as

$$\Pr\{\mathbf{s_1}\} = \frac{\binom{m}{s_1} p^{s_1} q^{m-s_1}}{\sum_{j=0}^{n-1} \binom{m}{j} p^j q^{m-j}}, \text{ and}$$

$$\Pr\{\mathbf{s_2}\} = \frac{\binom{m}{s_2} p^{s_2} q^{m-s_2}}{\sum_{j=0}^{n-1} \binom{m}{j} p^j q^{m-j}}.$$

Thus, the probability that a 2-call connection request is not blocked under second-stage multicast scheme is given by

$\Pr\{\text{connection not blocked}\}$
$= \sum_{r=0}^{n-1}\sum_{s_1=0}^{n-1}\sum_{s_2=0}^{n-1} \Pr\{\text{connection not blocked } |\mathbf{r},\mathbf{s_1},\mathbf{s_2}\}\Pr\{\mathbf{r}\}\ \Pr\{\mathbf{s_1}\}\Pr\{\mathbf{s_2}\}$

and the blocking probability is

$P_B = 1-\Pr\{\text{connection not blocked}\}$

4.3 The Blocking Behavior for First-Stage Multicast Scheme

In this section, we will analyze the 2-call blocking behavior under the *first-stage multicast* scheme. As we know, this scheme utilizes first stage multicasting without second stage multicasting. So if a 2-call can be routed, there must be two paths through two different middle stage modules.

Assume a 2-call from input module i to output modules j and k. Let's use A_i, B_j and B_k to represent the status of the three modules. If there are two distinct middle stage modules l_1 and l_2 which can connect this 2-call, then the following interstage links must be idle:

- the input-middle interstage link between input module i and middle stage module l_1, which is corresponding to $A_i[l_1]$.
- the middle-output interstage link between middle stage module l_1 and output module j, which is corresponding to $B_j[l_1]$.
- the input-middle interstage link between input module i and middle stage module l_2, which is corresponding to $A_i[l_2]$.
- the middle-output interstage link between middle stage module l_2 and output module k, which is corresponding to $B_k[l_2]$.

So we can get $A_i[l_1] = B_j[l_1] = A_i[l_2] = B_k[l_2] = 0$. That means, there must be at least one matching not only between A_i and B_j, but also at least one different matching between A_i and B_k.

Assume we have the same network state as in Figure 4.6: there are r 1's in A_i, s_1 1's in B_j and s_2 1's in B_k, where $s_1, s_2 \le n-1$ and $r \le m-2$. Due to the first-stage multicasting, in order to connect a new call, at least two input-middle interstage links from the input module should be idle, so $r \le m-2$.

Let $\mathbf{r}$ denote the event that there are r 1's in A_i, $\mathbf{s_1}$ denote the event that there are s_1 1's in B_j, and $\mathbf{s_2}$ denote the event that there are s_2 1's in B_k.

From Corollary 4.1, given events $\mathbf{r}$ and $\mathbf{s_1}$, we can get the probability that A_i and B_j have c_1 matchings:

$$\Pr\{c_1 \text{ matchings } |\mathbf{r},\mathbf{s_1}\} = \frac{\binom{m-r}{c_1}\binom{r}{m-s_1-c_1}}{\binom{m}{m-s_1}}$$

Similarly, given events $\mathbf{r}$ and $\mathbf{s_2}$, the probability that A_i and B_k have c_2 matchings is given by

$$\Pr\{c_2 \text{ matchings } |\mathbf{r}, \mathbf{s_2}\} = \frac{\binom{m-r}{c_2}\binom{r}{m-s_2-c_2}}{\binom{m}{m-s_2}}$$

As stated before, for the first-stage multicasting scheme, there is no multicasting in the second stage. So we cannot use one middle stage module to connect a 2-call. If there is only one matching between A_i and B_j and one matching between A_i and B_k, and both matchings correspond to the same middle stage module, then this 2-call will be blocked. Using the concept in Section 4.2, we know that for this case, there is only one matching *among* the three vectors. From Corollary 4.1 and the analysis in Section 4.2.1, we can get the probability for this scenario from the following steps, given events $\mathbf{r}$, $\mathbf{s_1}$ and $\mathbf{s_2}$:

- Assume there are c_1' matchings between A_i and B_j. For this case, $c_1' = 1$. The probability that there is only one matching between A_i and B_j is given by

$$\Pr\{\text{one matching } |\mathbf{r}, \mathbf{s_1}\} = (m-r)\frac{\binom{r}{m-s_1-1}}{\binom{m}{m-s_1}}$$

- Similarly, assume there are c_2' matchings between A_i and B_k. For this case, $c_2' = 1$. The probability that there is only one matching between A_i and B_k is given by

$$\Pr\{\text{one matching } |\mathbf{r}, \mathbf{s_2}\} = (m-r)\frac{\binom{r}{m-s_2-1}}{\binom{m}{m-s_2}}$$

- Then, we use the bitwise OR operation introduced in Section 4.2.1 to combine A_i and B_j into a new vector D_1, and A_i and B_k into a new vector D_2. So there are c_1' 0's in D_1 and c_2' 0's in D_2. Let $\mathbf{c_1'}$ denote the event that there are c_1' 0's in D_1, and $\mathbf{c_2'}$ denote the event that there are c_2' 0's in D_2. Given $\mathbf{c_1'}$ and $\mathbf{c_2'}$, the probability that there are c matchings between D_1 and D_2 is given by

$$\Pr\{c \text{ matchings } |\mathbf{c_1'}, \mathbf{c_2'}\} = \frac{\binom{c_1'}{c}\binom{m-c_1'}{c_2'-c}}{\binom{m}{c_2'}}.$$

  For this scenario, $c_1' = c_2' = c = 1$. So the probability that there is only one matching between D_1 and D_2 is

$$\Pr\{\text{one matching } |\mathbf{c_1'}, \mathbf{c_2'}\} = \tfrac{1}{m}.$$

So the probability for this scenario is the following

$$\begin{aligned}&\Pr\{\text{this scenario happens}|\mathbf{r}, \mathbf{s_1}, \mathbf{s_2}\}\\ =&\Pr\{\text{one matching } |\mathbf{r}, \mathbf{s_1}\}\Pr\{\text{one matching } |\mathbf{r}, \mathbf{s_2}\}\Pr\{\text{one matching } |\mathbf{c_1'}, \mathbf{c_2'}\}\end{aligned}$$

$$= \frac{(m-r)^2}{m} \frac{\binom{r}{m-s_1-1}}{\binom{m}{m-s_1}} \frac{\binom{r}{m-s_2-1}}{\binom{m}{m-s_2}}$$

Here, this scenario doesn't happen for any set of $\{r, s_1, s_2\}$. For example, if $m - r - s_1 \geq 2$, then there will be at least two matchings between A_i and B_j. So if there is at least one matching between A_i and B_k, this scenario doesn't happen. So we only consider this scenario, when $r \geq m - s_1 - 1$ and $r \geq m - s_2 - 1$.

Given events **r**, **s₁** and **s₂**, the 2-call under consideration is not blocked if and only if

- There exists at least one matching between A_i and B_j. Assume there are c_1 such matchings. So the following condition must be satisfied:

  $$\max\{\ 1, m-r-s_1\ \} \leq c_1 \leq \min\ \{\ m-r, m-s_1\}.$$

- There exists at least one matching between A_i and B_k. Assume there are c_2 such matchings. So the following condition must be satisfied:

  $$\max\{\ 1, m-r-s_2\ \} \leq c_2 \leq \min\ \{\ m-r, m-s_2\}.$$

- The above one matching scenario doesn't happen.

Therefore, the probability that the connection is not blocked, given events **r**, **s₁** and **s₂**, is given by

- $r \geq m - s_1 - 1$ and $r \geq m - s_2 - 1$:

  $$\Pr\{\text{connection not blocked } |\mathbf{r}, \mathbf{s_1}, \mathbf{s_2}\}$$
  $$= \sum_{c_1=max\{1,m-r-s_1\}}^{min\{m-r,m-s_1\}} \sum_{c_2=max\{1,m-r-s_2\}}^{min\{m-r,m-s_2\}} \Pr\{c_1 \text{ matchings } |\mathbf{r}, \mathbf{s_1}\} \Pr\{c_2 \text{ matchings } |\mathbf{r}, \mathbf{s_2}\}$$
  $$- \frac{(m-r)^2}{m} \frac{\binom{r}{m-s_1-1}}{\binom{m}{m-s_1}} \frac{\binom{r}{m-s_2-1}}{\binom{m}{m-s_2}}$$

- Or, for other r, s_1 and s_2, the one matching scenario doesn't happen, so

  $$\Pr\{\text{connection not blocked } |\mathbf{r}, \mathbf{s_1}, \mathbf{s_2}\}$$
  $$= \sum_{c_1=max\{1,m-r-s_1\}}^{min\{m-r,m-s_1\}} \sum_{c_2=max\{1,m-r-s_2\}}^{min\{m-r,m-s_2\}} \Pr\{c_1 \text{ matchings } |\mathbf{r}, \mathbf{s_1}\} \Pr\{c_2 \text{ matchings } |\mathbf{r}, \mathbf{s_2}\}$$

Under the assumption of link independence, we know that the events **r**, **s₁** and **s₂** are independent, which gives

$$\Pr\{\mathbf{r}, \mathbf{s_1}, \mathbf{s_2}\} = \Pr\{\mathbf{r}\}\Pr\{\mathbf{s_1}\}\Pr\{\mathbf{s_2}\}$$

We still use $p = \frac{an}{m}$ (as in Section 4.2) to represent the probability that an interstage link is busy, while $q = 1 - p$. The calculation of $\Pr\{\mathbf{s_1}\}$ and $\Pr\{\mathbf{s_2}\}$ is the same as in Section 4.2, which is

$$\Pr\{\mathbf{s_1}\} = \frac{\binom{m}{s_1} p^{s_1} q^{m-s_1}}{\sum_{j=0}^{n-1} \binom{m}{j} p^j q^{m-j}}, \text{ and } \Pr\{\mathbf{s_2}\} = \frac{\binom{m}{s_2} p^{s_2} q^{m-s_2}}{\sum_{j=0}^{n-1} \binom{m}{j} p^j q^{m-j}}.$$

The calculation for Pr{r} is different. Due to first stage multicasting, the number of busy input-middle interstage links from input module i is not bound by $n-1$ but $m-2$. So Pr{r} is given by

$$\Pr\{r\} = \frac{\binom{m}{r} p^r q^{m-r}}{\sum_{j=0}^{m-2} \binom{m}{j} p^j q^{m-j}}$$

Thus, the probability that a 2-call connection request is not blocked under first-stage multicast scheme is given by

$$\Pr\{\text{connection not blocked}\}$$
$$= \sum_{r=0}^{m-2} \sum_{s_1=0}^{n-1} \sum_{s_2=0}^{n-1} \Pr\{\text{connection not blocked } | r, s_1, s_2\} \Pr\{r\} \Pr\{s_1\} \Pr\{s_2\}$$

and the blocking probability is

$$P_B = 1 - \Pr\{\text{connection not blocked}\}$$

4.4 Numerical Results for The Two Multicast Schemes

In this section, numerical results are provided to examine the quality of the approximations for the blocking probability obtained from the analytical models. The results enable us to make a preliminary comparison of the two multicast schemes.

The network configuration we examined has 8 input (output) modules and 8 input (output) links on each input (output) module. The number of middle stage modules is changed from 8 to 16. We use a high call arrival probability, which is 0.8. In the simulation, each time we generate 10000 2-calls. Given a, the probability that one interstage link is busy can be computed by $p = \frac{an}{m}$. In the simulation, the status (idle or busy) of each interstage link is generated by a Bernoulli process with parameter p. For each 2-call, we generate the status for $3m$ interstage links, where m interstage links from source module, $2m$ interstage links to two destination modules. After that, we apply the two multicasting schemes to check if the current call is blocked based on the status of the $3m$ interstage links.

Figure 4.8 shows the relationship between the analytical models and the the simulation results for the two multicast schemes. The curve "analytical-1" is for the analytical model by using the second-stage multicast scheme, and the curve "analytical-2" is for the analytical model by using the first-stage multicast scheme. Also, the curve "simulation-1" is for the simulation results by using the second-stage multicast scheme, and the curve "simulation-2" is for the simulation by using the first-stage multicast scheme.

In Figure 4.8, the number of middle stage modules m is increasing from 8 to 16, and $a = 0.8$. As we can see, for each multicast scheme, the curves for the analytical model and the simulation results exhibit approximately the same shape. For both the second-stage scheme and the first-stage scheme, the analytical models yield a lower blocking probability than the simulation results. In overall, the first-stage multicasting scheme has a lower blocking probability than the second-stage scheme.

The two models proposed in this chapter provide lower bounds for the simulation results. In

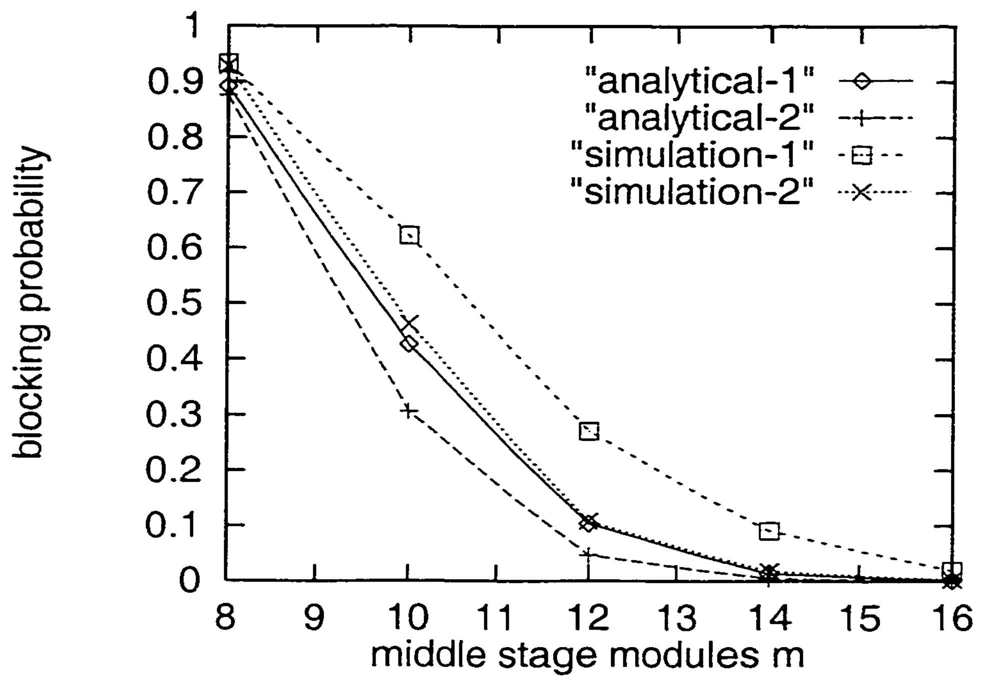


Figure 4.8: analytical models versus simulation results, $a = 0.8$

the analysis of the two models, we assume the events that there are matchings between any two vectors are independent with each other. For example, consider the event that there are c_1 matchings between A_i and B_j and the event that there are c_2 matchings between A_i and B_k. Sharing the same r, these two events are actually not independent with each other. So the two models may over-count the real probability.

Each of the two multicast schemes considered in this chapter has its merits and disadvantages. Based on the results we obtained from the analytical models and the simulations, we can make a preliminary comparison of them:

- In overall, the first-stage multicast scheme has a lower blocking probability than the second-stage multicast scheme. This is due to the flexibility of the first-stage multicast scheme in selecting available paths.

- Although the second-stage multicast scheme has a higher blocking probability, it uses less resources than the first-stage multicast scheme by putting the multicasting as near to the output stage as possible. For example, if we use interstage links as a kind of resources, three interstage links are used to connected a 2-call under the second-stage multicast scheme, while four interstage links are used under the first-stage multicast scheme.

4.5 Concluding Remarks

In this chapter, we analyze the 2-call blocking behavior for Clos networks under two different multicasting strategies. From the numerical results, we can see that the analytical models are consistent with the simulation results and can be good approximations of the actual blocking probability. We believe that based on the models presented in this chapter, more accurate analytical models for other routing strategies can be obtained as well, and the models can be extended to general k-calls. The models in this chapter provide lower bounds for the simulation results. The more useful upper bounds can be a good direction for the future works. Furthermore, the preliminary comparison of the two multicast schemes here can give a hand to the future works on the multicasting strategies in Clos networks.

Chapter 5

ATM and ATM Switch Designs

In this chapter, we first review the characteristics of ATM and ATM switches. Then we provide a survey of some typical ATM switch designs.
The organization of this chapter is as the follows. Section 5.1 introduces the background knowledge about ATM and ATM switches. It includes a review of the important characteristics of ATM and ATM protocols. It also introduces how an ATM switch works and some issues for ATM switch designs. Section 5.2 provides a survey of three ATM switch designs: the AT&T growable ATM switch, the *cross-path* switch and the *I-Cubeout* switch.

5.1 ATM

The *Asynchronous Transfer Mode* (ATM) is an important technology for B-ISDN, and is proposed to transport a wide variety of services in a seamless manner. ATM has the flexibility of packet switching and guarantees QoS for each connection as circuit-switching. ATM is based on a fixed-size virtual circuit-oriented cell-switching methodology. Each cell is 53-byte long, where 5 bytes are for header and 48 bytes are for the data. Before data transmission, a virtual circuit is set up. A virtual path is defined between each pair of endpoints in the communication network. All virtual circuits between the same source and destination should take the same virtual path. This can simplify routing and keep the cell ordering. In the cell header, there are a virtual path identifier (VPI) and a virtual circuit identifier (VCI). The VPI and VCI may vary from link to link.
When there is a new request for connection setup, a *setup message* which contains the information about the new connection is routed through the network. If there is enough bandwidth left in the virtual path between the source and destination of the new connection, the network can accept this request; otherwise, this request is rejected.
The ATM protocol *reference model* is based on standards developed by the ITU. There are three layers in this model: the *physical layer*, the *ATM layer* and the *ATM Adaption Layer* (AAL). The physical layer is defined to transport ATM cells between two ATM entities. The ATM layer provides an interface between the AAL and the physical layer. The AAL layer is on the top of the ATM layer and interfaces the higher layer protocols to ATM layer. When

Service Class	AAL	Quality of Service Parameter	Technical Parameters
constant bit rate (CBR)	AAL1	The cell rate is constant with time. CBR applications are quite sensitive to cell-delay variation. Examples are telephone traffic, video conferencing and television.	CLR, CTD, CDV, PCR
variable bit rate (VBR)	AAL2	The rate can vary with time but require a bounded delay. Examples are compressed packetized voice or video.	CLR, CTD, CDV, PCR, SCR, BT@PCR
available bit rate (ABR)	AAL3/4 or AAL5	This service has variable bit rate and doesn't require bounded delay. But it is desirable for switches to minimize delay and cell loss as much as possible. Examples are file transfer and e-mail.	CLR, flow control
unspecified bit rate (UBR)	AAL3/4 or AAL5	No connection is set up before data transmission. Examples are datagram traffic and data network applications.	PCR

- CLR = cell loss ratio CTD = cell transfer delay
- CDV = cell delay variation PCR = peak cell rate
- SCR = sustained cell rate BT = burst tolerance

Table 5.1: ATM Service Classes

relaying the data received from the higher layers to the ATM layer, the AAL segments the data into ATM cells; conversely, when relaying the data received from the ATM layer to higher layers, the AAL reassembles back the payload. Four AALs are proposed to support four service classes defined for ATM. The service classes are summarized in Table 5.1.

5.2 A Survey of Some ATM Switch Designs

Many large-scale ($\geq$ 1024 inputs/outputs) switch designs have been proposed in the literature for B-ISDN. Of these designs, only a handful are of the multistage fabric and appear to be suitable for ATM. In this section, we survey some designs in the latter category whose performance is justified either analytically or numerically, or by both techniques. Table 5.2 summarizes the important features of them.

5.2.1 The AT&T Growable Architecture

The ATM switching network proposed by [10] and [11] is based on the symmetric 3-stage Clos networks $N(n, m, k)$, where there are k input (output) modules, m middle stage modules and n input (output) links on each input (output module). We call a pair of input module i and output module j an input-output pair (i, j). So there are m disjoint paths for each input-output pair (i, j) through the m middle stage modules. This design adopts buffered Knockout output modules in order to maintain the best possible delay/throughput

Switching Network	Cell Re-ordering	Delay	Cell Loss (a sample result)
AT&T Growable ATM Switching Network ([10], [11])	No	1 or 2 time slots	Switch size $N = \infty$, $n = 16$ Expansion $m/n = 47/16$, 100% load, cell loss below 10^{-9}
Cross-Path Switching Network ([13])	No	Variable, at most one frame long	Switch size $N = 1024$, $n = 16$ Expansion $m/n = 48/16$, 80% load, cell loss about 10^{-7}
I-Cubeout Network ([12])	Yes	variable, at most m slots, m: the number of stages	Switch size $N = 512$, 15 stages of 8×16 SEs, cell loss is 10^{-6}

Table 5.2: A Sample of Architectures

performance. For each output, if there are more than m cells destined to it at the same time, only m cells can be routed to it.

The transmission time is divided into time *slots*. Then the switching fabric operates synchronously with the arriving ATM cells in each time slot. Paths are assigned on a slot-by-slot basis by using a routing algorithm called *straight matching* repeatedly in each time slot. At the end of each time slot, the cells which cannot be assigned paths are dropped.

The *straight matching* routing algorithm plays an important role in the original switch design, and also in Chapter 6 of this thesis. Here, we will describe how this algorithm works. As we know from the above introduction, this algorithm is applied in each time slot. Given a group of incoming cells in one time slot, it dynamically computes the connection patterns in the middle stage modules to make path assignment. Furthermore, this algorithm divides one time slots into k minislots. In each minislot, every input module is given permission to schedule its cells to a particular output module. For example, in the first minislot, input module i tries to schedule its cells destined to output module j and reserves as many free paths as required for these cells. In the hth minislot, input module i attempts to schedule its cells destined to output module $((j + h - 1) \bmod k)$. At the end of one time slot, each input module can schedule its cells to all the output modules.

Here, we introduce a concept *base* which is used in the *straight matching* algorithm and in Chapter 6. For a given slot, assume in the first minislot, input module i $(0 \leq i \leq k - 1)$ tries to schedule its cells destined to output module j $(0 \leq j \leq k - 1)$. We define the *base* to be: $(j - i) \bmod k$, which is the distance from the output module to the input module. This *base* is the same for all the input-output pairs which are given permission to schedule cells in the first minislot. So, if we let *base* to be 0, then in the first minislot, input module 0 schedules its cells to output module 0, and input module 1 to output module 1, etc.

To implement the *straight matching* routing scheme, two sets of vectors A and B are used to represent the status of input and output modules. For example, in one minislot, the status of the input module i can be represented by vector A_i, and the status of the output module j can be represented by vector B_j. Each element in a vector corresponds to an outgoing or ingoing link. If this link is occupied, then that element has a 1, otherwise has a 0. The concepts of the A and B vectors are similar to those in Chapter 4. Each input module

keeps its own A_i; the B vectors are circulated in the first stage as routing control messages. In each minislot, an input module receives some vector B_j from the above input module, compares it with its own vector, and reserves some links by marking the corresponding elements 1 in both of these two vectors, then passes this vector B_j down to the next input module at the end of this minislot. At the end of one time slot, each input module meets all the B vectors, so it can schedule its cells to all the output modules.
The *straight matching* algorithm is fast but suboptimal. The algorithm doesn't consider any advanced knowledge of the incoming traffic. Also it doesn't give priority to heavy loaded input or output modules. As we can see, within each time slot, the input-output pairs scheduled in the later minislots encounter more reserved links so they suffer from more cell loss. For an input module i, if most of its cells request output module j, but the input-output pair (i, j) is scheduled in the last minislot, then the most loss will happen to that input-output pair. On the other hand, assume the input module i has the cells arriving in continuous time slots to output module j (in a burst). If the input-output pair (i, j) happens to be scheduled in the last minislot in every time slot, then most cells from input module i to output module j will be lost.
In [11], cell loss analysis for unicasting was provided. Two kinds of cell loss are considered: *Knockout loss*, and *additional loss* due to the suboptimal algorithm. An upper bound on the cell loss probability for arbitrary patterns of independent cell arrivals is drawn from the worst case: the last scheduled input-output pairs. Also a tighter bound for uniform arrival traffic is shown.
Although the straight matching algorithm is suboptimal and may not be able to schedule all the cells even if an optimal algorithm can do, it is quite fast, simple and also has good overall performance.

5.2.2 The Cross-Path Architecture

In [13], a large-scale ATM switch called *Cross-path* switch is proposed. The architecture is based on 3-stage Clos network. Input buffering and output buffering are both used in the fabric. The switch uses a distributed, quasi-static routing algorithm called *path switching*. Unlike the straight matching algorithm which rearranges the middle stage connection patterns in each time slot, the path switching algorithm takes the QoS requirement of the VCs into consideration and uses predetermined connection patterns for the center stage in a round-robin way to satisfy the bandwidth requirements of the virtual connections. Let f be a set of predetermined connection patterns which are used repeatedly every f time slots, then a *frame* has f time slots. Figure 5.1 is an example for the predetermined connection patterns when $f = 2$.
The predetermined connection patterns can be made by coloring a bipartite capacity graph which is generated from traffic statistics, and are stored in each input module. In one time slot, the input modules use a subset of predetermined connection patterns to schedule

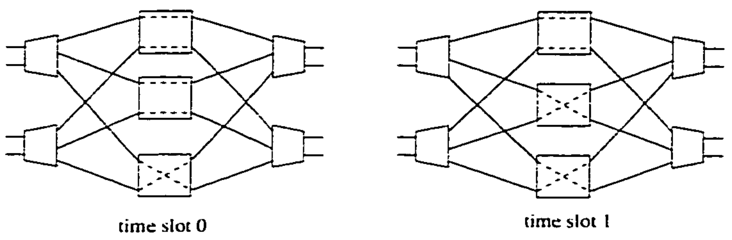


Figure 5.1: Connection Patterns in the middle stage for routing scheduling

their cells. Because each input module has the routing information. there is no need of communications among the input modules. If the traffic doesn't change significantly. using path switching can get good overall performance.
This design has some drawbacks. As the number of VCs requiring high bandwidth increases. the number of connection patterns increases: this may result in an increase in the delay incurred by VCs that require low bandwidth. The predetermined connection patterns are saved in each input module. which requires extra memory in input modules. Moreover. these patterns are obtained from traffic statistics. but it is not easy to decide what traffic statistics can be used.
In [13]. numerical results about cell loss probability are given. Due to the input buffering. cell loss only happens in the third stage. The cell loss probability here is a straightforward extension of the loss probability analysis for Knockout switch. By assuming 100% throughput in the third stage. [13] also provides results for the throughput limitation on input modules.

5.2.3 The I-Cubeout Architecture

In [12]. an ATM switch design called *I-Cubeout* was proposed. This design consists of repeated copies of multiple stages of $b \times 2b$ switching elements. which are interconnected according to the indirect n-cube connection style. The output buffers are separated from the multistage structure. Each $b \times 2b$ switching element has b outlets to the switching elements in the next stage and b outlets to separate output buffers. Figure 5.2 illustrates the architecture of the I-Cubeout switch.
The I-Cubeout employs distributed self-routing. The routing tags are computed at the primary inputs. If a cell reaches a switching element connected to a destination output buffer. it is sent to the output buffer. When two cells conflict at one switching element. only one cell is moved forward and the other is *deflection-routed.* The cell with a small distance to its destination is given priority. where the distance is defined as the minimum number of stages a cell has to travel before getting to its destination. This conflict resolution allows the cells to reach their destinations as soon as possible and free resources for other cells. If two conflicting cells have the same priority. a random one is selected. If a cell cannot reach the destination output buffer in the first copy. it can continue to traverse in

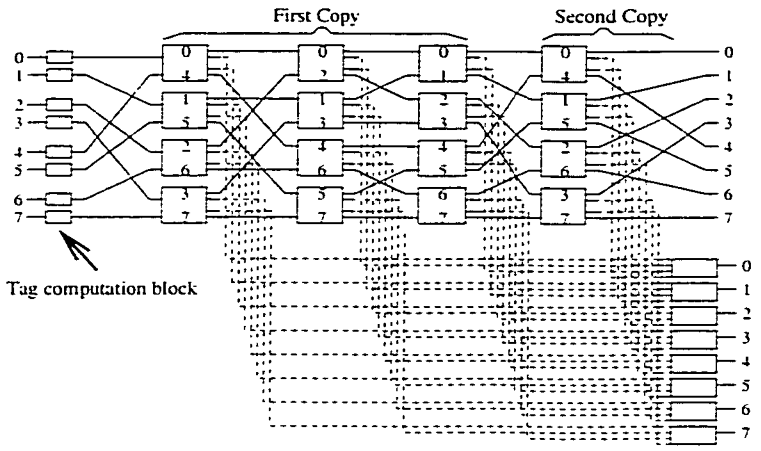


Figure 5.2: I-Cubeout switch with $N = 8$, $b = 2$, four stages

the later copies.

The important idea behind this design is that the cell loss can be reduced by introducing more stages. Moreover, it is scalable and has an acceptably low cell loss. However, the cell delay varies from cell to cell, so cell ordering may not be kept.

5.3 Concluding Remarks

In this chapter, we present a survey of ATM technology and three typical ATM switch designs. Among these three designs, the growable ATM switch from AT&T is very simple and competitive, and hence deserves a serious considerations to study its bottleneck and try to seek improvement for it. The *straight matching* routing algorithm is used in this switch design and makes path assignment dynamically in each time slot. It is fast but sub-optimal. The further studying of this algorithm will be provided in the next chapter.

Chapter 6

Worst Case Scenario of AT&T Growable ATM Switch

This chapter provides new results about some worst case scenarios for the *straight matching* routing algorithm for the ATM switching network proposed by [10] and [11].
The straight matching algorithm is fast and effective. However, it is sub-optimal: for a given traffic matrix in one time slot, there may still be cell loss even without any Knockout loss. The work of [11] focused on the performance evaluation of this algorithm in general and its application in multicasting networks under random traffic pattern. To better understand the behavior of the original algorithm, we focus in this chapter on worst case scenarios. We ask questions like: if we are given the network parameters (*n*,*k*,*m*, and *load*), how much is the maximum cell loss incurred by using this algorithm and what kind of traffic matrices can cause such cell loss? How to improve the basic algorithm to reduce cell loss for these *bad* matrices? This chapter focuses on these problems.
The organization of this chapter is as follows. Section 6.1 provides some experimental results and observations. It contains some sample traffic matrices which lose more cells than other matrices with the same network parameters (*n*,*k*,*m*, and *load*). Section 6.2 introduces two algorithms for generating a special type of *bad* matrices. *Algorithm 1* is presented in Section 6.2.2 and is for the fully-loaded case when *load*$=1$. *Algorithm 2* (Section 6.2.3) is an extension of *Algorithm 1* for non-fully-loaded cases when *load*<1. Both *Algorithm 1* and *Algorithm 2* base on the assumption that *base*$=0$ (The concept of *base* is in Section 5.2.1). In Section 6.2.4, these two algorithms are extended for the cases when *base*$\neq 0$. Section 6.3 introduces a class of matrix transformations that can preserve cell loss value. Section 6.4 describes some specialized preprocessing methods to improve the original *straight matching* algorithm. These methods can change the behavior of the original algorithm by re-arranging the input traffic matrices or changing the scheduling order of the algorithm before applying the algorithm.

6.1 Experimental Results and Observations

In this section, some experimental results and observations from the simulation are provided. First, here are some assumptions used in the simulation:

- The size of a network is decided by three parameters: n is the number of input (output) links on one input (output) module, m is the number of middle stage modules, and k is the number of input (output) modules.

- The cell arriving to one input link is generated by a Bernoulli process with parameter ρ, where ρ is the probability that there is a cell arriving. The network is *fully-loaded* if $\rho = 1$, where each input link has a cell arriving.

- The traffic in each time slot is represented by a $k \times k$ matrix T. Each element $T[i, j]$ is the number of cells from input module i to output module j in a particular time slot. The total cells from one input module can not exceed n, so the sum of all the numbers in one row of T is no more than n. In order to focus on the loss incurred by routing, we eliminate the Knockout loss so that the sum of one column of T is no more than m.

- For the straight matching algorithm, scheduling order for all the input modules are decided at the beginning of each time slot. The scheduling order indicates to which output module an input module should schedule its cells in one special minislot. For instance, the cell scheduling for a network with k input and output modules is in Table 6.1 (IM for input module, OM for output module).

minislot	IM 0	IM 1	IM 2	...	IM k-1
0	OM 0	OM 1	OM 2	...	OM k-1
1	OM 1	OM 2	OM 3	...	OM 0
2	OM 2	OM 3	OM 4	...	OM 1
...	...	...	...	...	...
k-1	OM k-1	OM 0	OM 1	...	OM k-2

Table 6.1: An Example of the Scheduling Order

A parameter *base* is used in the simulation to control the scheduling order. This parameter uses the concept *base* introduced in Chapter 5 and is defined as follows: in each minislot, input module i is scheduled to access output module j, such that *base*= $(j - i) \bmod k$. Table 6.1 shows the scheduling order when *base*= 0. In this chapter, we assume *base*= 0, unless otherwise specified.

- When there are two, or more available straight matchings for one cell, instead of using the random selection method (as used in the original straight matching algorithm), a packing strategy is used to select the available top-most middle stage module.

From the simulation, we can get the following examples of *bad* traffic matrices which cause about 30% cell loss for some *fully-loaded* networks with different sizes, by using *base*= 0:

- **Case 1**: $m = n = k = 4$. The maximum cell loss observed by randomly generating traffic matrices is 6 cells out of total 16 cells in one time slot. Here are four examples of traffic matrices which can cause such cell loss:

$$T_1 = \begin{pmatrix} 0* & 1 & 1 & 2 \\ 2 & 2* & 0 & 0 \\ 1 & 1 & 2* & 0 \\ 1 & 0 & 1 & 2* \end{pmatrix} \quad T_2 = \begin{pmatrix} 2* & 1 & 0 & 1 \\ 2 & 0* & 1 & 1 \\ 0 & 2 & 2* & 0 \\ 0 & 1 & 1 & 2* \end{pmatrix}$$

$$T_3 = \begin{pmatrix} 2* & 0 & 1 & 1 \\ 1 & 2* & 1 & 0 \\ 1 & 2 & 0* & 1 \\ 0 & 0 & 2 & 2* \end{pmatrix} \quad T_4 = \begin{pmatrix} 2* & 0 & 0 & 2 \\ 1 & 2* & 0 & 1 \\ 0 & 1 & 2* & 1 \\ 1 & 1 & 2 & 0* \end{pmatrix}$$

- **Case 2**: $m = n = 6$, $k = 4$. The maximum cell loss observed is 8 cells out of 24 cells. A sample matrix is

$$\begin{pmatrix} 0* & 2 & 2 & 2 \\ 2 & 4* & 0 & 0 \\ 2 & 0 & 4* & 0 \\ 2 & 0 & 0 & 4* \end{pmatrix}$$

- **Case 3**: $m = n = 4$, $k = 6$. The maximum cell loss observed is 8 cells out of 24 cells. A sample matrix is

$$\begin{pmatrix} 0* & 0* & 1 & 1 & 1 & 1 \\ 0* & 0* & 1 & 1 & 1 & 1 \\ 1 & 1 & 2* & 0 & 0 & 0 \\ 1 & 1 & 0 & 2* & 0 & 0 \\ 1 & 1 & 0 & 0 & 2* & 0 \\ 1 & 1 & 0 & 0 & 0 & 2* \end{pmatrix}$$

- **Case 4**: $m = n = k = 6$. The maximum cell loss observed is 12 cells out of 36 cells. A sample matrix is

$$\begin{pmatrix} 0* & 0* & 3 & 1 & 1 & 1 \\ 0* & 0* & 0 & 2 & 2 & 2 \\ 0 & 3 & 3* & 0 & 0 & 0 \\ 1 & 2 & 0 & 3* & 0 & 0 \\ 2 & 1 & 0 & 0 & 3* & 0 \\ 3 & 0 & 0 & 0 & 0 & 3* \end{pmatrix}$$

- **Case 5**: $m = n = k = 8$. The maximum cell loss observed is 20 cells out of 64 cells. A sample matrix is

$$\begin{pmatrix} 0* & 0* & 2 & 0 & 2 & 2 & 0 & 2 \\ 0* & 0* & 1 & 2 & 1 & 1 & 2 & 1 \\ 2 & 1 & 5* & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 5* & 0 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 & 5* & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 5* & 0 & 0 \\ 0 & 2 & 0 & 1 & 0 & 0 & 5* & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 5* \end{pmatrix}$$

From the simulation results, we can find a way to generate some special structure of *bad* traffic matrices. The algorithms and theorems for the generation of such *bad* matrices are introduced in the next section.

6.2 Generation of Bad Matrices

In this section, we first introduce a general method to construct a class of traffic matrices which can lose asymptotically $\frac{kn}{3}$ cells for any network size kn, under the assumption that $\rho = 1, m = n$ and *base*$= 0$. Then we extend this method to the cases when $\frac{1}{2} < \rho < 1$ and the cases when *base*$\neq 0$.

6.2.1 Partition of the input traffic

First, we introduce the main idea behind this method. As we know, a traffic matrix $T_{k,k}$ can be converted to a bipartite graph G which contains k input nodes and k output nodes. There are $T[i,j]$ edges between input node i and output node j. So the problem of routing a given traffic matrix through m middle stage modules can be converted to the m-coloring problem in the corresponding bipartite graph. The m-coloring problem means coloring the edges in a graph using m colors such that no two edges sharing the same node receive the same color. Figure 6.1 is an example of this conversion, assuming $m = n = 6, k = 4$. So, there are total 24 cells and 6 colors.

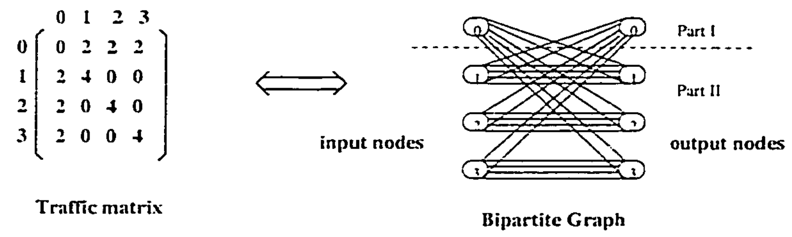


Figure 6.1: The traffic matrix and its corresponding bipartite graph

We first outline the construction method for the small case in Figure 6.1. We can divide the bipartite graph in Figure 6.1 into two parts, where the first part contains input node 0 and output node 0, and the second part contains all other nodes. As we can see, there is no edge between input node 0 and output node 0. In the second part, there are two edges between each pair of input node i and output node i ($i = 1, 2, 3$).

We now show that at least 8 cells will be lost by the original straight matching algorithm. Since *base*=0, the cells along the main diagonal in matrix $T_{k,k}$ will be scheduled first. So in Figure 6.1, all the edges between any input node i and output node i in the bipartite graph will be colored first. After the first minislot coloring, at least 4 colors are used and can't be used in the later minislots. So there are only two colors left for the later rounds. For the edges from input node 0 and the edges to output node 0 in the first part, because these edges are colored after the first minislot, only two colors can be used for them. But input node 0 (output node 0) has 6 outgoing (ingoing) edges. Thus, at least 8 edges can't be colored, and there can be at least 8 cells lost.

We now generalize the above construction method for the fully-loaded case. Here, we refer to input node i and output node i as *same-index input-output pair*. For a given matrix and a given pair (i, j) of integers (i counts for the number of some input modules hence $0 \leq i \leq k$, and j counts for edges of the bipartite graph hence $0 \leq j \leq nk$), if the corresponding bipartite graph of this matrix can be divided into the following two parts:

1. The first part contains i same-index input-output pairs. There is no edge between any input and output node in this part.
2. The second part contains the other $k - i$ same-index input-output pairs. There are at least j edges between the nodes in each same-index input-output pair.

Then we can calculate the lower bound of cell loss for this given matrix quickly without applying the matching algorithm to the whole matrix, which is $2ij$. Furthermore, for a given set of parameters (m, n, k, ρ), if we can find a pair (i, j) which can be calculated from (m, n, k, ρ) and maximize the lower bound of cell loss $2ij$, then we can construct a group of traffic matrices which can lose at least $2ij$ cells. We can apply this partition idea to *Algorithm 1* in the next section.

6.2.2 The Generation Algorithm for Fully-loaded Case

By using the idea of partition in the last section, we can get the following *Algorithm 1* to generate the bad matrices which can cause about $\frac{kn}{3}$ cell loss, under the assumption that $\rho = 1$, $m = n$ and *base*= 0.

Algorithm 1 has two phases:

1. In the first phase, we compute the values of the pair (i, j) which can be utilized to generate a complete traffic matrix with certain lower bound of cell loss.
2. In the second phase, we can construct a complete traffic matrix (using the pair (i, j)) that causes a zero Knockout loss (the assumptions in Section 6.1 about the summation for each row or column in the traffic matrix).

First, we describe the first phase:

- There are two parameters: i and j. Assume there are i same-index input-output pairs in group 1. Moreover, there are at least j edges between each same-index input-output pair in the second group. Assume we divide the bipartite graph into four parts:

  1. Part A contains the i input nodes in group 1 and all the edges from these nodes.
  2. Part A' contains the $k - i$ input nodes in group 2 and all the edges from these nodes.
  3. Part B contains the i output nodes in group 1 and all the edges to these nodes.
  4. Part B' contains the $k - i$ output nodes in group 2 and all the edges to these nodes.

  We use $E_{AB'}$ to denote the set of edges from Part A to Part B'. Since all the input nodes in Part A only have edges to the output nodes in Part B', $E_{AB'}$ contains $i * n$ edges, where $|E_{AB'}| = i * n$. Also, we use $E_{A'B'}$ to denote the set of edges from Part A' to Part B'. $E_{A'B'}$ contains two subsets of edges. The first subset, $E_{A'B'_1}$, consists of edges between each same-index input-output pair in the second group, where $|E_{A'B'_1}| \geq j * (k - i)$. For each edge in the second subset $E_{A'B'_2}$, its input node and output node do not have the same index number. And, $|E_{A'B'_2}| \geq 0$.

  Then, we use $E_{B'}$ for all the edges to Part B', where $|E_{B'}| = (k - i) * n$. Since

  $$|E_{B'}| = |E_{AB'}| + |E_{A'B'}| = |E_{AB'}| + |E_{A'B'_1}| + |E_{A'B'_2}|,$$

  we can get $(k-i)*n \geq i*n + j*(k-i)$. So $j \leq n - \frac{i*n}{k-i}$. Also, since $j \geq 1$, $n - \frac{i*n}{k-i} \geq 1$. Then we get $i \leq \frac{(n-1)k}{2n-1}$.

- The algorithm searches in the space $1 \leq i \leq \lfloor \frac{(n-1)k}{2n-1} \rfloor$. For each particular value of i, let $j = \lfloor n - \frac{i*n}{k-i} \rfloor$. Then compute the new lower bound of maximum cell loss, which is $2ij$. After all the iterations, a pair (i, j) can be found to maximize $2ij$. Let the values of this pair to be (max_i, max_j).

The second phase of *Algorithm 1* is to construct a complete traffic matrix based on (max_i, max_j) and the zero Knockout assumptions. Here is the description of the second phase:

- Randomly select max_i same-index input-output pairs and put them in group 1. In the traffic matrix, for all the elements $T[x, y]$ where x and y are both in group 1, set them to 0;

- The other nodes are in group 2. In the traffic matrix, for all the elements $T[x, x]$ where x is in group 2, set them to be at least max_j;

- Fill the other positions of the traffic matrix according to the fully-loaded condition, which means that the sum of each row is n and the sum of each column is m $(m = n)$.

Here, we provide the first theorem which proves *Algorithm 1* can lose around $\frac{kn}{3}$ cells:

Theorem 6.1: The matrices generated by the above algorithm can cause asymptotically $\frac{kn}{3}$ cell loss, under the assumption that $\rho = 1$, $m = n$ and *base*$= 0$.

Proof: Algorithm 1 divides the input and output nodes into two groups. Assume there are i same-index input-output pairs in the first group and $k - i$ same-index input-output pairs in the second group. Also assume at least j edges between each same-index input-output pair in the second group.

After the first minislot, at least j colors are used. Since $m = n$, $n - j$ colors are left for the later rounds. Because $\rho = 1$, each input (output) node has n outgoing (ingoing) edges. So for each node in the first group, at least j edges can't be colored. That means for each node in the first group, at least j cells are lost. Totally, at least $2ij$ cells would be lost.

Let's use a function $f = 2ij$. As we know, in *Algorithm 1*, $1 \leq i \leq \frac{(n-1)k}{(2n-1)}$ and $j = \lfloor n - \frac{n*i}{(k-i)} \rfloor$. Substituting i and j in $f = 2ij$, we get

$$f = 2n\frac{ki-2i^2}{k-i}$$

In order to maximize f, we first find the derivative of function f:

$$f' = 2n\frac{2i^2-4ik+k^2}{(k-i)^2}$$

Let $f' = 0$. Then $2i^2 - 4ik + k^2 = 0$. From this equation, $i = k + \frac{\sqrt{2}}{2}k$, or $i = k - \frac{\sqrt{2}}{2}k$. Since $i < k$, we set $i = k - \frac{\sqrt{2}}{2}k$. So

$$f = 2ij = 2in\frac{k-2i}{k-i} = (2-\sqrt{2})^2 nk$$
$$= (6 - 4\sqrt{2})nk \approx 0.34nk \approx \frac{nk}{3}. \qquad \blacksquare$$

6.2.3 The Generation Algorithm for Non-fully-loaded Cases

We now extend *Algorithm 1* to non-fully-loaded cases to generate a class of *bad* matrices, when $\rho < 1$. That means each input module has ρn active links or ρn cells. Let $n' = \rho n$. So an original $n \times n$ ($m = n$) input module with ρ can be represented by a $n' \times n$ input module. If $\rho \leq 0.5$, then $n \geq 2n'$, and the network behaves as a strictly non-blocking network. So we just consider the case when $\rho > 0.5$. Same as *Algorithm 1*, we assume $m = n$ and *base*$= 0$. Similar to *Algorithm 1*, *Algorithm 2* contains two phases: the first phase makes the partition: the second phase constructs the complete traffic matrix. There is one difference from *Algorithm 1*. For each output module, it can have at most m cells to it. To be simple, we assume that each output module in the first group has x cells to it, where $0 < x < n$.

We first introduce the first phase of *Algorithm 2*:

- There are three parameters i, j and x: i is the number of same-index input-output pairs in group 1, j is the minimum number of edges between each same-index input-output pair in the second group, and x is the number of edges to each output module in group 1.

- Similar to *Algorithm 1*, we divide the bipartite graph into four parts: Part A, Part B, Part A', Part B'. The division of these four parts is the same as in Section 6.2.2. We use E for all the edges in the graph, $E_{AB'}$ for the edges from Part A to Part B', $E_{A'B}$ for the edges from Part A' to Part B, $E_{A'B'}$ for the edges from Part A' to Part B'. Also, $E_{A'B'}$ consists of two subsets: $E_{A'B'_1}$ (edges between same-index input-output pairs) and $E_{A'B'_2}$ (edges not between same-index input-output pairs).

From the above assumptions, given x and i, we can get the following equations and inequalities:

$$|E| = |E_{AB'}| + |E_{A'B}| + |E_{A'B'}| = |E_{AB'}| + |E_{A'B}| + |E_{A'B'_1}| + |E_{A'B'_2}|$$

$$|E| = \rho * n * k$$

$$|E_{AB'}| = \rho * i * n$$

$$|E_{A'B}| = i * x$$

$$|E_{A'B'_1}| \geq (k-i) * j$$

$$|E_{A'B'_2}| \geq 0$$

From these, we can get

$$\rho * n * k \geq \rho * i * n + x * i + (k-i) * j$$

So here is the first inequality for j: $j \leq \rho * n - \frac{i*x}{k-i}$.

On the other hand, for all the $k-i$ output nodes in Part B', there can be at most $(k-i) * n$ edges to them. Let $E_{B'}$ for all the edges to Part B':

$$|E_{B'}| = |E_{AB'}| + |E_{A'B'_1}| + |E_{A'B'_2}| \leq (k-i) * n$$

So

$$\rho * i * n + (k-i) * j \leq |E_{AB'}| + |E_{A'B'_1}| + |E_{A'B'_2}| \leq (k-i) * n$$

Then we can get $(k-i) * n - (k-i) * j - \rho * i * n \geq 0$. So here is the second inequality for j: $j \leq n - \frac{\rho*i*n}{k-i}$. Combining the two inequalities for j, we get

$$j \leq min(\rho * n - \frac{i*x}{k-i}, n - \frac{\rho*i*n}{k-i})$$

- The algorithm searches in the space $1 \leq i \leq k-1$ and $0 \leq x \leq n$. For each particular value of i and each particular value of x, $j = \lfloor min(\rho * n - \frac{i*x}{k-i}, n - \frac{\rho*i*n}{k-i}) \rfloor$. Then compute the new lower bound of maximum cell loss.

  As we know, if there are at least j edges between each same-index input-output pair in the second group, then j colors are used and cannot be used later ($n-j$ colors left for group 1). On the other hand, each input node in group 1 has $\rho * n$ edges, so if $\rho * n - n + j > 0$, then $\rho * n - n + j$ edges from one input node in the first group can't be colored. Similarly, for each output node in the first group, it has x edges. So if $x - n + j > 0$, then $x - n + j$ edges to one output node in the first group can't be colored.

In contrast with *Algorithm 1*, when we compute the new lower bound of maximum cell loss in the loop for a given x, i and the corresponding j, we consider the following four cases:

1. $\rho * n - n + j > 0$ and $x - n + j > 0$: For each input node in group 1, at least $\rho * n - n + j$ edges can't be colored. And for each output node in group 1, at least $x - n + j$ edges can't be colored. So the total loss can be $i * (\rho * n + x - 2n + 2j)$. So the new lower bound is $i * (\rho * n + x - 2n + 2j)$.
2. $\rho * n - n + j > 0$ and $x - n + j \leq 0$: Since $x - n + j \leq 0$, the number of colors left $(n - j)$ is larger than the number of edges to an output node in the first group. So we can't decide if some edge to an output node in the first group can't be colored. But we are sure that $\rho * n - n + j$ edges from one input node in the first group can't be colored. So the total loss can be at least $i(\rho * n - n + j)$. Then the new lower bound is $i(\rho * n - n + j)$.
3. $\rho * n - n + j \leq 0$ and $x - n + j > 0$: Similar to the second case, we can't decide if some edges from an input node in the first group can't be colored. But we are sure that $x - n + j$ edges to one output node in the first group can't be colored. So the total loss is at least $i * (x - n + j)$. The new lower bound is $i * (x - n + j)$.
4. $\rho * n - n + j \leq 0$ and $x - n + j \leq 0$: for all the input and output nodes in the first group, we can't decide if some edges from an input node or to an output node can't be colored. Then we cannot use this method to decide the loss. So we do nothing and skip this case.

After all the iterations, a set of (i, j, x) can be found to maximize the lower bound of maximum cell loss. Let the values of this set to be (max_i, max_j, max_x).

The second phase of *Algorithm 2* is to construct a complete traffic matrix based on (max_i, max_j, max_x) and the assumptions mentioned at the beginning of this section. Here is the description of the second phase:

- Randomly select max_i same-index input-output pairs and put them in group 1. In the traffic matrix, for all elements $T[x, y]$ where x and y are both in group 1, set them to 0;
- Place the remaining nodes in group 2. In the traffic matrix, for all the elements $T[x, x]$ where x is in group 2, set them to be at least max_j;
- Fill the other positions of the traffic matrix according to the non-fully-loaded condition. In the traffic matrix, for each column corresponding to an output module in group 1, the sum of this column should be max_x. For other columns corresponding to output modules in group 2, the sum of each column can not exceed n. Also, the sum of each row is $\rho * n$.

By using the above algorithm, we can generate the following sample matrix for the case that $n = m = k = 6$, $\rho = \frac{2}{3}$ and *base*= 0:

$$\begin{pmatrix} 0* & 1 & 1 & 1 & 1 & 0 \\ 1 & 3* & 0 & 0 & 0 & 0 \\ 1 & 0 & 3* & 0 & 0 & 0 \\ 1 & 0 & 0 & 3* & 0 & 0 \\ 1 & 0 & 0 & 0 & 3* & 0 \\ 1 & 0 & 0 & 0 & 0 & 3* \end{pmatrix}$$

In this matrix, each input module has 4 (ρn) cells, and $x = 5$, $i = 1$, $j = 3$. This matrix loses 3 cells, which is much for the case $n = m = k = 6$, $\rho = \frac{2}{3}$ and *base*= 0.

6.2.4 Bad Matrices for Non-Zero *base*

In the last two sections, we have introduced *Algorithm 1* and *Algorithm 2* for *base*= 0. In this section, we will show that these two algorithms can be easily extended to generate *bad* matrices when *base*$\neq$ 0.

First, we introduce another concept: *b-skewed input-output pair*. In the last sections, we use a same-index input-output pair to indicate an input module (node) and an output module (node) which have the same index number, for example, input module i and output module i form a same-index input-output pair. For a given b, where $0 < b \leq k - 1$, a *b-skewed input-output pair* consists of input module i and output module $(i + b \bmod k)$ for any $0 \leq i \leq k - 1$.

Now, we explain the idea of extending *Algorithm 1* and *Algorithm 2* to generate *bad* matrices when *base*= b, where $0 < b \leq k - 1$. The algorithms here also contain two phases: partition and construction.

In *Algorithm 1* and *Algorithm 2*, the first phase is to divide the same-index input-output pairs into two groups. Here, instead of using the same-index input-output pairs, we use *b-skewed input-output pairs* when *base*= b. We can use the same method as in *Algorithm 1* and *Algorithm 2* to divide all the *b-skewed input-output pairs* into two groups:

- The first group contains i *b-skewed input-output pairs*, and no edge between any input node and output node in this group.
- The second group contains other *b-skewed input-output pairs*, and at least j edges between the input node and output node in each *b-skewed input-output pair* in this group.

For the non-fully-loaded cases, a value for x is also needed.

Then we can complete the construction of a traffic matrix in the second phase:

- Randomly select i *b-skewed input-output pairs* in the first group. In the traffic matrix, for all the elements $T[x, y]$ where x and y are both in group 1, set them to 0;
- In the traffic matrix, for all the elements $T[x, (x + b) \bmod k]$ where x is in the second group, set them to be at least j;

- Fill the other positions of the traffic matrix according to the fully-loaded or non-fully-loaded condition.

We use an example to explain the two phases. Assume *base*$= 1$, $m = n = 4$, $k = 6$, and $\rho = 1$. Instead of using the regular same-index input-output pairs, we use the following *1-skewed input-output pairs*:

pair	0	1	2	3	4	5
input node	0	1	2	3	4	5
output node	1	2	3	4	5	0

Similar to *Algorithm 1*, the first phase decides 2 *1-skewed input-output pairs* in the first group and at least 2 edges between the input node and output node in each *b-skewed input-output pair* in the second group. Then the second phase constructs the whole traffic matrix. If we select Pair 0 and Pair 1 in the above table and put them in the first group, we can get the following traffic matrix which can lose 8 cells when *base*$=$ 1:

$$\begin{pmatrix} 1 & 0* & 0* & 1 & 1 & 1 \\ 1 & 0* & 0* & 1 & 1 & 1 \\ 0 & 1 & 1 & 2* & 0 & 0 \\ 0 & 1 & 1 & 0 & 2* & 0 \\ 0 & 1 & 1 & 0 & 0 & 2* \\ 2* & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Here is another example for non-fully-loaded cases. Assume *base*$= 1$, $m = n = k = 6$, and $\rho = \frac{2}{3}$. If we set $x = 5, i = 1, j = 3$, we can get the following traffic matrix which can lose 3 cells when *base*$=$ 1:

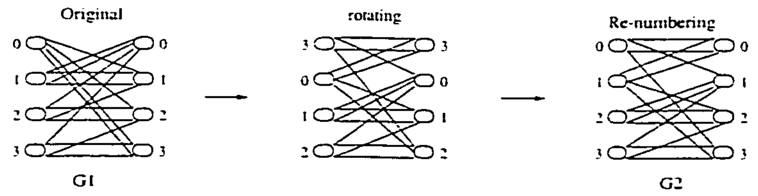
$$\begin{pmatrix} 0 & 0* & 1 & 1 & 1 & 1 \\ 0 & 1 & 3* & 0 & 0 & 0 \\ 0 & 1 & 0 & 3* & 0 & 0 \\ 0 & 1 & 0 & 0 & 3* & 0 \\ 0 & 1 & 0 & 0 & 0 & 3* \\ 3* & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

6.3 Cell Loss Preserving Transformations

This section introduces a matrix transformation that preserves the cell loss. The motivation for studying such a matrix transformation is three-fold:

- The results obtained in this direction are useful in improving the performance since they allow us to classify a number of apparently different matrices into a small number of patterns, and then designing solutions for each pattern.
- This direction gives us a way of generating worst-case benchmarks of arbitrary large size traffic matrices without replicating the exact structure of certain matrices.
- Experimentally, we observed that many *bad* matrices are actually transformations of each other.

First, we introduce some transformation of traffic matrices and their corresponding bipartite graphs. For a given matrix, if we rotate the bottom same-index input-output pair in its bipartite graph to the top and re-number the input and output nodes from top to bottom, we can get a new bipartite graph and its corresponding traffic matrix. Figure 6.2 shows how this rotating and re-numbering works. In Figure 6.2, the graph G_1 and the traffic matrix T_1 are the original graph and matrix, while the graph G_2 and the traffic matrix T_2 are the graph and matrix after transformation (one same-index input-output pair rotating and re-numbering). T_2 is obtained from rotating T_1 one column right and one row down. T_1 and T_2 have the same cell loss under the same *base*.



$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 1 & 2 \\ 1 & 2 & 2 & 0 & 0 \\ 2 & 1 & 1 & 2 & 0 \\ 3 & 1 & 0 & 1 & 2 \end{array} \longrightarrow \begin{array}{c|cccc} & 3 & 0 & 1 & 2 \\ \hline 3 & 2 & 1 & 0 & 1 \\ 0 & 2 & 0 & 1 & 1 \\ 1 & 0 & 2 & 2 & 0 \\ 2 & 0 & 1 & 1 & 2 \end{array} \longrightarrow \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 2 & 1 & 0 & 1 \\ 1 & 2 & 0 & 1 & 1 \\ 2 & 0 & 2 & 2 & 0 \\ 3 & 0 & 1 & 1 & 2 \end{array}$$

T1 T2

Figure 6.2: The rotation and re-numbering of bipartite graph

As we can see, this transformation only changes the index number of input and output nodes and doesn't change the connection patterns between each pair of input and output node. So after the transformation, for any input (output) node i in the original graph, where $0 \le i \le k-1$, it corresponds to the input (output) node ($i+1 \bmod k$). We call this *one input-output pair transformation*. Here is the Lemma 6.1:

Lemma 6.1: The original bipartite graph and the graph obtained after rotating one input-output pair and re-numbering, have the same cell loss behavior, under the same *base*.

Proof: Let I_i (O_i) represent any input (output) node i in the original graph and I'_i (O'_i) to be any input (output) node i in the graph after transformation. So for one input-output pair transformation, I_i (O_i) corresponds to $I'_{(i+1)\text{mod } k}$ ($O'_{(i+1)\text{mod } k}$). And the number of edges between I_i and O_i is the same as the number of edges between $I'_{(i+1)\text{mod } k}$ and $O'_{(j+1)\text{mod } k}$. The transformation preserves the connection patterns.
Assume *base*$= b$. So in the first minislot (minislot 0), I_i schedules its cells to $O_{(i+b)\text{mod } k}$. On the other hand, in minislot 0, $I'_{(i+1)\text{mod } k}$ schedules its cells to $O'_{(i+b+1)\text{mod } k}$. $O'_{(i+b+1)\text{mod } k}$ is corresponding to $O_{(i+b)\text{mod } k}$. Since the number of edges between I_i and $O_{(i+b)\text{mod } k}$ is the same as the number of edges between $I'_{(i+1)\text{mod } k}$ and $O'_{(i+b+1)\text{mod } k}$, and the packing strategy is used (not randomly selection), cell loss is the same for I_i and $I'_{(i+1)\text{mod } k}$ in minislot 0.

As to minislot j, I_i schedules its cells to $O_{(i+b+j) \bmod k}$ and $I'_{(i+1) \bmod k}$ schedules its cells to $O'_{(i+b+j+1) \bmod k}$. $O_{(i+b+j) \bmod k}$ is corresponding to $O'_{(i+b+j+1) \bmod k}$. The number of edges between I_i and $O_{(i+b+j) \bmod k}$ is the same as the number of edges between $I'_{(i+1) \bmod k}$ and $O'_{(i+b+j+1) \bmod k}$. So cell loss is the same for I_i and $I'_{(i+1) \bmod k}$ in minislot j. Through all the time slot, I_i and $I'_{(i+1) \bmod k}$ have the same cell loss behavior. Since the matching between any I_i and $I'_{(i+1) \bmod k}$ is one-to-one, all the input nodes in the original graph have the same cell loss as the input nodes in the graph after transformation. So the cell loss is preserved after the transformation: one input-output pair rotating and re-numbering. ■

This lemma can be extended to Lemma 6.2 for *multiple input-output pair transformation*, where multiple same-index input-output pairs are rotated and then re-numbering. Here is the second lemma:

Lemma 6.2: The original bipartite graph and the graph after one *multiple input-output pair transformation*, have the same cell loss behavior, under the same *base*.

Proof: This lemma can be proved easily by repeating the proof process in Lemma 6.1. For example, G_2 is one input-output pair transformation of the original graph G_1. If we make one input-output pair transformation on G_2 to get G_3, G_3 is the graph after two input-output pair transformation of G_1. G_3 has the same cell loss as G_2, so as G_1. ■

The above lemmas describe the suggested transformation on the bipartite graphs representing traffic distributions. We now formalize the transformation on the corresponding traffic matrices. We first introduce the concept of the diagonal matrix $D_{k,k}$ associated with a traffic matrix $T_{k,k}$: the ith row of $D_{k,k}$ is obtained from the ith diagonal of $T_{k,k}$: that is, for any i ($i = 0, 1, ...k-1$), use the elements of T ($T[0, i], T[1, (i+1) \bmod k], ..., T[k-1, (i+k-1) \bmod k]$) to construct the ith row of its diagonal matrix D ($D[i, 0], D[i, 1], ..., D[i, k-1]$). From the construction method, we can get $D[i, j] = T[j, (i+j) \bmod k]$. On the other hand, a traffic matrix T can be generated from a diagonal matrix D from $T[i, j] = D[(j-i) \bmod k, i]$.

For example, if $T_1 = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 2 & 2 & 0 & 0 \\ 1 & 1 & 2 & 0 \\ 1 & 0 & 1 & 2 \end{pmatrix}$. Then $D_1 = \begin{pmatrix} 0 & 2 & 2 & 2 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 2 & 1 & 1 \end{pmatrix}$.

If we rotate D_1 one column right, we can get another diagonal matrix D_2 and its corresponding traffic matrix T_2,

$D_2 = \begin{pmatrix} 2 & 0 & 2 & 2 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 2 & 2 & 1 \end{pmatrix}$ and $T_2 = \begin{pmatrix} 2 & 1 & 0 & 1 \\ 2 & 0 & 1 & 1 \\ 0 & 2 & 2 & 0 \\ 0 & 1 & 1 & 2 \end{pmatrix}$

T_1 and T_2 have the same cell loss under the same *base*. Similarly, if we rotate D_1 two or three columns right, we can get other two diagonal matrices. From these diagonal matrices, the generated traffic matrices have the same cell loss as T_1 and T_2 under the same *base*.

Here is the second theorem:

Theorem 6.2: Given a traffic matrix T_1, by column-rotating its diagonal matrix D_1, we can get another diagonal matrix D_2 and generate the corresponding traffic matrix T_2. T_2 has the same cell loss as T_1, under the same *base*.

Proof: First, we consider one column rotating right of diagonal matrix. So given T_1 and D_1, we can get

$D_1[x, y] = D_2[x, (y+1) \bmod k]$.

For any $T_1[i, j]$, since $T_1[i, j] = D_1[(j-i) \bmod k, i]$,

$T_1[i, j] = D_2[(j-i) \bmod k, (i+1) \bmod k]$.

On the other hand, $T_2[x, y] = D_2[(y-x) \bmod k, x]$. Let $x = (i+1) \bmod k$ and $y = (j+1) \bmod k$.

$$\begin{aligned}
&T_2[(i+1) \bmod k, (j+1) \bmod k] \\
&= D_2[(((j+1) \bmod k) - ((i+1) \bmod k)) \bmod k, (i+1) \bmod k] \\
&= D_2[(j-i) \bmod k, (i+1) \bmod k] \\
&= T_1[i, j]
\end{aligned}$$

So, we know that input module I_i (output module O_i) for T_1 is corresponding to input module $I_{(i+1) \bmod k}$ (output module $O_{(i+1) \bmod k}$) for T_2. If we consider the bipartite graph G_1 for T_1 and the bipartite graph G_2 for T_2, we can get that G_2 is *one input-output pair transformation* of G_1. From Lemma 6.1, G_1 and G_2 have the same cell loss, so do T_1 and T_2.

The proof for one-column rotating right can be easily extended to the case one-column rotating left. For example, D_1 is one-column rotating left of D_2. Also the proof can be extended to multiple column rotation by applying Lemma 6.2. ■

6.4 Improvement of the Straight Matching Algorithm

In this section, some preprocessing methods are introduced which may improve the original straight matching algorithm for the *bad* matrices. In the following sections, we seek improvements based on the following methods:

- **Reorganization of the incoming traffic:** A *bad* matrix contains some permutation pattern which may cause much cell loss. So the first way can be reorganization of the incoming traffic.

- **Randomization of the scheduling order:** The cell loss of a traffic matrix also depends on the scheduling order of the routing algorithm. For example, for an input module, scheduling its cells to output module i first may cause less cell loss than scheduling its cells to output module i later. So another way is to modify the scheduling order, for example, changing the *base* parameter.

6.4.1 Sorting

The idea of using a sorting network as part of a large-scale switch has been first used in Batcher-Banyan networks. The first method introduced here *sorts* all the incoming cells in a non-decreasing order according to their destinations, then *distributes* the sorted list of cells to the input modules. So this method can reorganize the incoming traffic. For a given traffic matrix, this method can combine some non-zero elements in the matrix and concentrate all the non-zero elements along the main diagonal. For example, after sorting and re-distribution, the traffic matrix $\begin{pmatrix} 0 & 1 & 1 & 2 \\ 2 & 2 & 0 & 0 \\ 1 & 1 & 2 & 0 \\ 1 & 0 & 1 & 2 \end{pmatrix}$ becomes to the matrix $\begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$.

The later matrix has no cell loss, where the first one has 6 cells lost, if using the same *base*. To implement this method, a sorting network needs to be introduced before the routing network. After the sorting, the sorted list of cells will be distributed to the input stage of the routing network. There are two different ways to distribute the cells to the routing network:

- **Scheme 1 - concentration**: The cells are concentrated to the input modules from top to bottom. This method fills the top-most module first, then to the modules below it. If the number of arriving cells is less than the number of total input links in the input stage of the routing network, then the input modules at the bottom may not be used or fully utilized.

- **Scheme 2 - even distribution**: The cells are evenly distributed to the input modules. If there are total l cells and k input modules, then each module may have $\lfloor \frac{l}{k} \rfloor$ or $\lceil \frac{l}{k} \rceil$ cells.

Figures 6.3 to 6.6 show a comparison of the original network without sorting and the networks with sorting. In the simulation, the number of input (output) links on one input (output) module in the routing network is 16 ($n = 16$), and the number of input (output) modules is also 16 ($k = 16$). m is the number of middle stage modules in the routing network. ρ is the traffic load. We set *base*$= 0$.

Figure 6.3 shows the relationship of m and the cell loss for the three methods under the fully-loaded condition $\rho = 1$. As we can see, when $m = n$, the network with sorting can achieve better improvement over the network without sorting. For this case, each input module in the routing network has n cells and each output module also has n cells to it ($m = n$). So after sorting, the n cells on input module i in the routing network are all destined to output module i. For instance, there are n cells to output module 0. After sorting, these cells are on the top of the sorting list and only distributed to the input module 0. On the other hand, input module 0 is fully loaded by the cells to output module 0. So in the traffic matrix after sorting, all the non-zero elements are in the main diagonal. Thus there is no cell loss for the network with sorting.

In Figure 6.3, the two curves for the two schemes are identical; this is explained as follows. For the fully-loaded case, each input module has the same number of cells n, so does each output modules (n cells to it). So, given a particular sorted list of cells (by destination output modules), the two distribution schemes identically distribute the cells to input modules. That is, the bipartite graphs considered in the straight matching algorithm are identical. Then we can get the same cell loss for them.

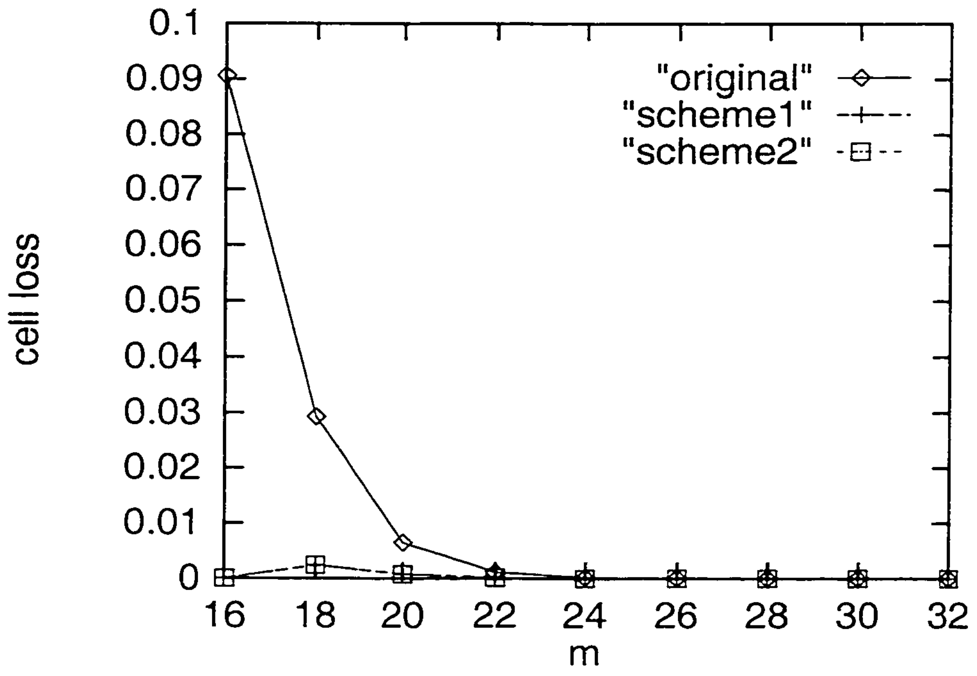


Figure 6.3: original versus scheme 1 and scheme 2, $\rho = 1$

Figure 6.4 shows the relationship of m and the cell loss for the three methods under the non-fully-loaded condition when $\rho = 0.8$. Similar to Figure 6.3, when $m = n$, the network with sorting improves over the network without sorting. Here, the two curves for the two schemes with sorting are not the same. As we know, for the non-fully-loaded case, the number of cells in the sorting list is less than the number of input links in the routing network. So the two schemes make different distribution of cells to the routing network. That is the reason they have different cell loss.

Figure 6.5 shows the relationship of ρ and the cell loss for the three methods when $m = n = 16$. Here, $m = n$, the best improvement of the network with sorting over the network without sorting can be achieved when $\rho = 1$. The same thing happens in Figure 6.6, where $m = 22$. The network with sorting achieves the best improvement over the network without sorting when $\rho = 1$.

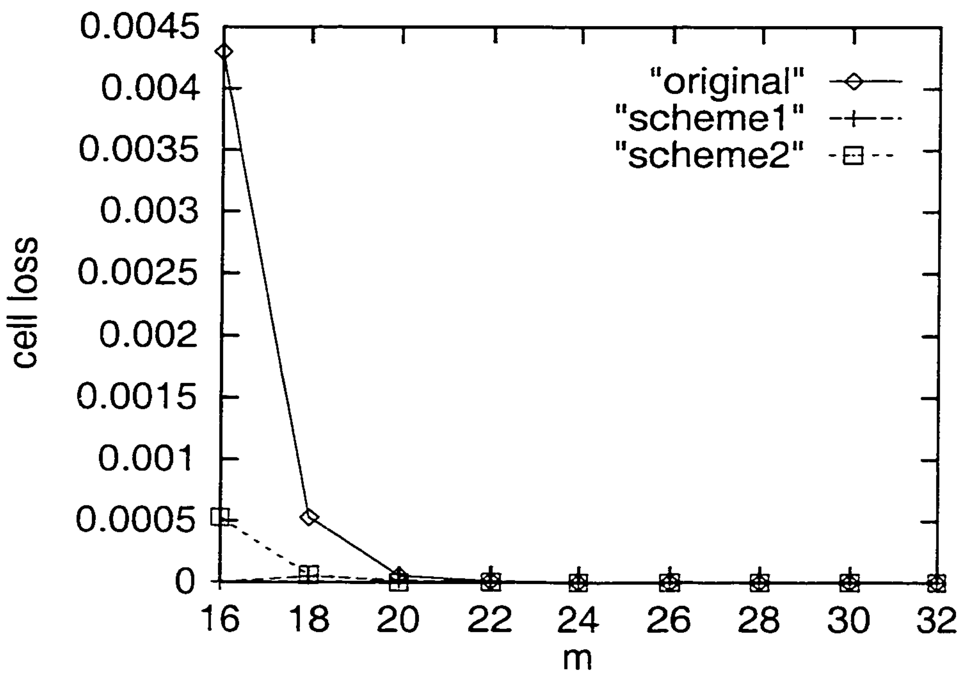


Figure 6.4: original versus scheme 1 and scheme 2, $\rho = 0.8$

From the above results, we can see that the network with sorting can improve performance in overall. The sorting method is most preferred for the network with less middle stage modules and the heavy traffic. However, the sorting network increases the architecture complexity of the whole network.

6.4.2 Randomization of Scheduling Order

In the original method, at the beginning of one time slot, the scheduling order is decided by *base*. For example, given a particular value of *base*, the scheduling order can let input module i schedule its cells to output module ($i + base + l \bmod k$) in minislot l. Instead of using this kind of scheduling order which is decided by *base*, we can use some random ones. Let's use the following example to explain this method. In this example, $n = m = k = 6$ and $\rho = 1$. The traffic matrix is

$$\begin{pmatrix} 0* & 0* & 3 & 1 & 1 & 1 \\ 0* & 0* & 0 & 2 & 2 & 2 \\ 0 & 3 & 3* & 0 & 0 & 0 \\ 1 & 2 & 0 & 3* & 0 & 0 \\ 2 & 1 & 0 & 0 & 3* & 0 \\ 3 & 0 & 0 & 0 & 0 & 3* \end{pmatrix}$$

If we use *base*-decided scheduling order and *base*= 0, then the scheduling order should be in Table 6.2.

If using this scheduling order, the above traffic matrix can lose 12 cells. However, if we use the scheduling order in Table 6.3 which can not be decided just by one parameter *base*, only two cells lost.

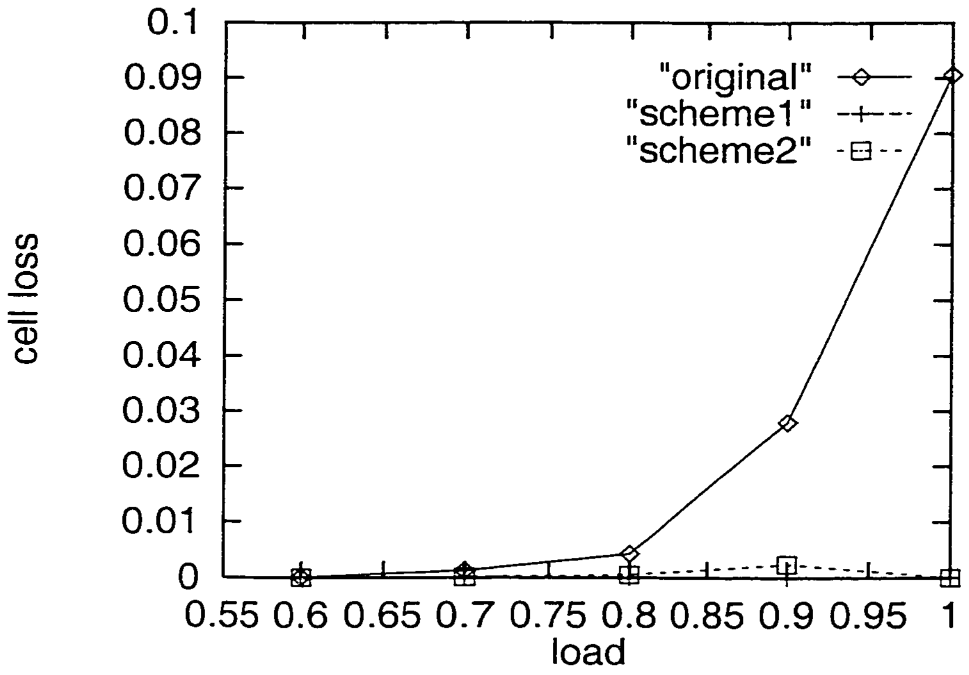


Figure 6.5: original versus scheme 1 and scheme 2, $m = 16$

minislot	IM 0	IM 1	IM 2	IM 3	IM 4	IM 5
0	OM 0	OM 1	OM 2	OM 3	OM 4	OM 5
1	OM 1	OM 2	OM 3	OM 4	OM 5	OM 0
2	OM 2	OM 3	OM 4	OM 5	OM 0	OM 1
3	OM 3	OM 4	OM 5	OM 0	OM 1	OM 2
4	OM 4	OM 5	OM 0	OM 1	OM 2	OM 3
5	OM 5	OM 0	OM 1	OM 2	OM 3	OM 4

Table 6.2: The Scheduling Order when *base*= 0

The method of changing *base* is a special case of this randomization method. Instead of using *base*=0, we can set *base* to be another value less than k. For the matrices (T_1, T_2, T_3, T_4) in Section 6.1, if we use *base*=1, 2, 3, the main diagonal can be avoided being scheduled in the first minislot. As a result, these four matrices lose two cells instead of 6 cells for *base*= 0. The methods provided in this section can help to alleviate the problem caused by using the basic algorithm directly to the *bad* matrices. However, they still have some drawbacks. The sorting method can reorganize the incoming traffic, but the sorting network increases the hardware complexity of the whole architecture. The randomization method can change the scheduling order, but how to decide which alternative random scheduling order should be used is still a problem.

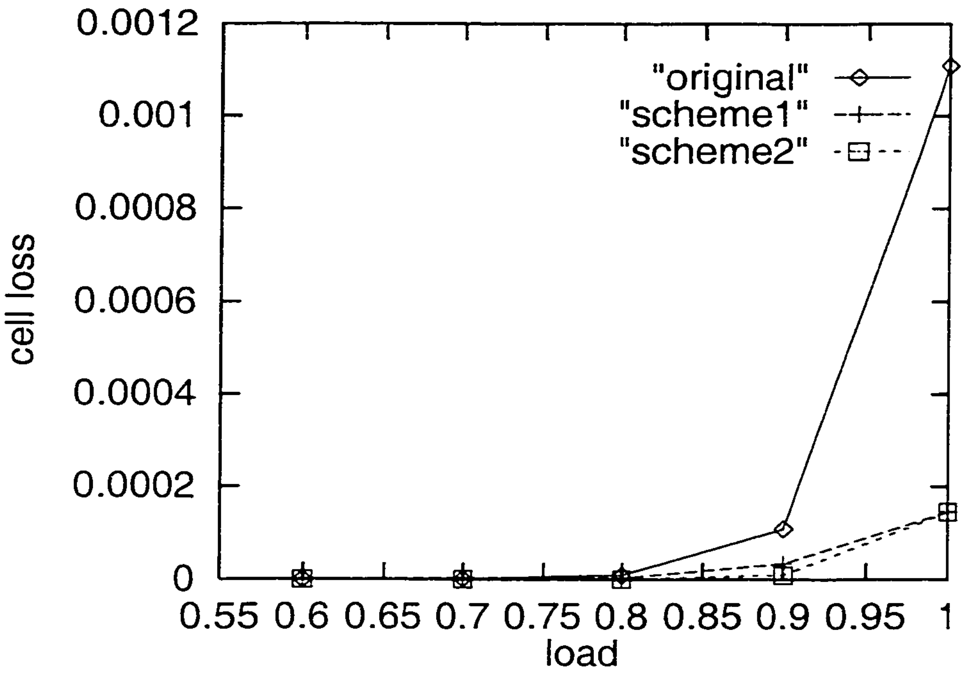


Figure 6.6: original versus scheme 1 and scheme 2, $m = 22$

minislot	IM 0	IM 1	IM 2	IM 3	IM 4	IM 5
0	OM 2	OM 1	OM 0	OM 5	OM 4	OM 3
1	OM 1	OM 0	OM 5	OM 4	OM 3	OM 2
2	OM 0	OM 5	OM 4	OM 3	OM 2	OM 1
3	OM 5	OM 4	OM 3	OM 2	OM 1	OM 0
4	OM 4	OM 3	OM 2	OM 1	OM 0	OM 5
5	OM 3	OM 2	OM 1	OM 0	OM 5	OM 4

Table 6.3: The Random Scheduling Order

6.5 Concluding Remarks

In this chapter, we take a further study on the worst case scenarios for the *straight matching* algorithm. Based on the simulation results, we found the a class of *bad* traffic matrices. The algorithms are proposed to generate such *bad* matrices for the fully-loaded case ($\rho = 1$) and non-fully-loaded cases ($\frac{1}{2} < \rho < 1$), under the assumption that *base*$= 0$. For the fully-loaded case, the traffic matrices which are constructed by using *Algorithm 1* (Section 6.2.2) can cause asymptotically $\frac{kn}{3}$ cell loss for any network size kn. Furthermore, the two algorithms are extended to the cases with non-zero *bases*. This chapter also shows that a kind of traffic matrix transformations can preserve the cell loss. At last, some preprocessing methods are introduced which may improve the original straight matching algorithm for the *bad* matrices. Some simulation results are provided to show that some preprocessing methods, such as *sorting*, can reduce the cell loss in general.

Chapter 7

Conclusions and Future Works

Along with the rapid growth of Internet and the emerging of new multimedia applications, there is an urgent need for telecommunication and data communication networks to be able to provide high bandwidth and integrated services. As a dominating transfer mode in optical networks which can provide higher capacity and reduce cost for new applications, *circuit-switching* emerges in association with a new technology. The blocking behavior of Clos networks in circuit-switching environment is a well-studied topic in the literatures. In the first part of this thesis, we extend existing results by investigating 2-call blocking behavior of Clos networks in circuit-switching environment. On the other hand, *ATM* is a technology which is designed to carry different kinds of traffic and services in one uniform backbone. In the second part of this thesis, we focus on studying the performance of some typical ATM switch designs. In this chapter, we summarize the contributions of this thesis, and suggest some directions for future works.

In the first part of this thesis (Chapter 3 and Chapter 4), we first reviewed some previous works on the point-to-point call blocking behavior of 3-stage Clos networks. Then we extended previous works to the analysis of 2-call blocking probability. We studied two in-switch multicasting schemes: second-stage multicasting and first-stage multicasting. Two analytical models are presented for these two schemes. Furthermore, based on the analytical results and simulation results, a comparison of the analytical models and the simulation results and a preliminary comparison of the two schemes are provided. It is shown that the first-stage multicast scheme has a good potential for producing lower blocking probability than the second-stage multicast scheme with the cost of consuming more network resources.

In the second part of this thesis (Chapter 5 and Chapter 6), we first reviewed some typical ATM switch designs. We selected the AT&T growable ATM switch as our studying object. In Chapter 6, we first studied the performance of this switch design under uniform unicast traffic. We characterized a class of traffic matrices that cause a significant cell loss. Then we proposed three algorithms used to generate the bad matrices under different network configurations. Furthermore, we studied some pre-processing methods which may improve the performance of the switch for the bad matrices. These approaches include sorting and randomization of scheduling orders. Based on simulation results, it was shown that sorting can improve the performance in general.

Although significant progress has been made over the past several years, much additional work in circuit-switching and ATM is still needed. Below, we outline some possible future directions:

- Today, many applications such as video-on-demand, distant learning, distant diagnosis, video conferencing and so on, require point-to-multipoint communication. Multicasting is an important issue we should give more effort in the future work. The future work can be done in the two aspects. First, a study of k-call blocking behavior is needed. This study can be done for random routing strategy and other routing strategies. Also such a study can be done for the network allowing the rearrangement of the existing calls. Although rearrangement of the existing calls can somehow affect the quality of the existing calls, it can reduce the architecture complexity of the networks, for instance, crosspoints.

- Multicasting in ATM switches has been widely studied in the literatures. In ([8],[7]), the performance of the AT&T growable ATM switch under multicast traffic is provided. Upper bounds on the cell loss probability for arbitrary patterns of independent cell arrivals under the second-and-third-stage multicast scheme and the first-and-third-stage multicast scheme are given. However, there is no previous work for the study of the worst-case multicast traffic for the AT&T growable ATM switch. So the work about the *bad* traffic matrices can be extended to find the *bad* multicast traffic types.

- In this thesis, we studied two pre-processing methods to improve the performance of the switch under a traffic load corresponding to the load of *bad* matrices. More work can be done in this direction to propose other possible pre-processing methods and also to improve the existing pre-processing methods. Beside the preprocessing ideas, we can also use post-processing methods to refine the path assignment obtained from the original routing algorithm. If we can get a better path assignment after post-processing, then the better one can be used and the cell loss can be reduced; otherwise, the network can still use the path assignment from the original routing algorithm.

- In Chapter 5, we reviewed some typical ATM switch designs. We note that the *path switching* algorithm used in the *Cross-Path* switch ([13]) is in contrast with the straight matching algorithm used in the AT&T growable ATM switch. In particular, the *path switching* algorithm uses predetermined connection patterns to satisfy the bandwidth requirements of the VCs in a round-robin fashion. Since after the VCs setup, the switching network knows the QoS requirement, the arrival rate, the incoming link and the output link of each VC going through it, it is able to predict many aspects of the incoming traffic. The *path switching* algorithm takes the advantage of such information in cell scheduling. But the straight matching algorithm doesn't take this into consideration. If it happens that the traffic matrices with high arrival frequency are bad matrices, the performance can be affected severely. So if the switch can predict the traffic matrices with high arrival frequency and decide if they are bad matrices, then it can adjust the basic routing algorithm to reduce the cell loss and improve the performance. The future work can be done to combine the advantages of these two routing methods and try to propose some new routing algorithms or switch designs in between.

With today's communication networks experiencing enormous growth, it is important that we develop and take advantage of technologies to keep pace with those increasing demands. We believe there will be a bright future in communication networks waiting for us.

Bibliography

[1] A.Huang and S.Knauer. Starlite: A wideband digital switch. *IEEE Globecom*, pages 121–125, November 1984.

[2] C.Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, pages 406–424, 1953.

[3] C.Y.Lee. Analysis of switching networks. *The Bell System Technical Journal*, pages 1287–1315, Nov. 1955.

[4] F.K.Hwang and C.H.Lin. Broadcasting in a three-stage point-to-point nonblocking network. *International Journal of Reliability, Quality and Safety Engineering*, pages 299–307. May 1995.

[5] J.Y. Hui. *Switching and Traffic Theory for Integrated Broadband Networks*. Kluwer Academic Publishers, 1990.

[6] C. Jacobaeus. A study on congestion in link systems. *Ericsson Technics*, vol. 51 1950.

[7] Daniel J.Marchok and C.E.Rohrs. First stage multicasting in a growable packet (ATM) switch. *IEEE International Conference on Communication*, pages 1007–1013, June 1991.

[8] Daniel J.Marchok, C.E.Rohrs, and R.M.Schafer. Multicasting in a growable packet (ATM) switch. *IEEE INFOCOM*, pages 850–858, April 1991.

[9] J.N.Giacopelli, J.J.Hickey, W.S.Marcus, W.D.Sincoskie, and M.Littlewood. Sunshine: A high-performance self-routing broadband packet switch architecture. *IEEE Journal on Selected Areas in Communications*, pages 1289–1298, October 1991.

[10] K.Y.Eng, M.J.Karol, and Y.S.Yeh. A growable packet switch architecture: Design principles and applications. *IEEE GLOBECOM*, pages 1159–1165, November 1989.

[11] M.J.Karol and Chih-Lin I. Performance analysis of a growable architecture for broadband packet (ATM) switching. *IEEE transactions on Communication*, pages 431–439, February 1992.

[12] N.Tzeng, K.Ponnuru, and K.Vibhatavanij. A cost-effective design for ATM switching fabrics. *IEEE International Conference on Communication*, 1999.

[13] T.T.Lee and C.H.Lam. Path switching — a quasi-static routing scheme for large-scale ATM packet switches. *IEEE Journal on Selected Areas in Communications*, pages 914–924, June 1997.

[14] V.E.Benes. On rearrangeable three-stage connecting networks. *Bell System Technical Journal*, pages 1481–1492, September 1962.

[15] Y.Yang and N.H.Kessler. Modeling the blocking of Clos networks. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1998.

[16] Y.Yeh, M.G.Hluchyi, and A.S.Acampora. The Knockout switch: A simple, modular architecture for high-performance packet switching. *IEEE/ACM Transactions on Networking*, pages 142–151, February 1993.