BMSN AND SPIDERNET AS LARGE SCALE ATM SWITCH INTERCONNECTION ARCHITECTURES

By

Kin-yu Cheung

A Thesis

submitted in partial fulfillment of the requirements

for the Degree of Master of Philosophy

Division of Information Engineering

The Chinese University of Hong Kong

June 1997

Acknowledgement

I would like to express my warmest gratitude towards many people who have helped me towards the completion of my research. First of all, I would like to thank my supervisor, Professor P.C.Wong, for his invaluable guidance. I appreciate his committed attitude towards research very much. My colleagues, Gang Feng, Patrick Yee, and Jack Lee are very good to me. Mr Feng has always been treating me as a close friend. Patrick has given me many useful advices and encouragement all the time. Jack always never minds in sharing his research experience to me.

There are other colleagues whom I would like to acknowledge, especially those gentlemen in the Broadband Communications Laboratory. I will never forget those exciting and dangerous trips when we went hiking together. Special thanks go to those colleagues of the Technical Support Centre, especially Miss Maxi Hui who has kindly allocated enough computing resources to me for running the simulation programs.

Finally, and most importantly, I would like my father and mother to know that I love them very much. My father has been working very hard to give financial support to my education in the past twenty years. My mother is the best in the world. She has always been caring about my health during the hard

days in the course of my research.

You deserve the honour.

Kin-yu Cheung

June 1997

Abstract

The design of large scale ATM switch architectures has been an important research topic in recent years. The motivation is originated from the anticipation of the possible requirement of the future BISDN central office to install ATM switch fabrics that can provide thousands of high-speed ports [7]. Many researchers [10, 21, 23, 20, 12] have been searching for ways to achieve the goal. However, many designed switches are technologically too difficult to implement.

Constructing a large ATM switch by using a number of small switches seems a very promising strategy. Under this strategy, switch interconnection topologies should be investigated. In this thesis, we start out in this direction and propose the use of the Bidirectional Manhattan Street Network (BMSN) as the underlying topology. We hope to take the merit of the torus structure of BMSN to achieve good switching performance. Using BMSN for this purpose, each node now becomes an ATM switch module with its own local input and output ports, and four extension channel groups to connect horizontally and vertically with its neighbouring modules. In this way, the switch size and capacity can be increased by adding switch modules in a stepwise and planar manner, and there is no need of rewiring the I/O ports and most of the interconnections.

We extend BMSN into another architecture which is named SpiderNet. This

architecture connects each module with four more neighbours in the diagonal directions. The average path length is reduced by having more direct paths. SpiderNet, therefore, gives a better call blocking and cell loss performance, and has a higher switch capacity with the same number of interconnection fibres.

Another topic we investigate is on the design of multichannel switches. We propose a multichannel switch design strategy that carries two major advantages. (1) No assumption is made on the required switching technology. This means that popular switch types, such as space-division switch, shared-memory switch, shared-bus switch, etc., can be used; (2) Sequential delivery of cells are guaranteed at the destination switch port even after cells have passed through a series of channel groups. We will describe two channel allocation algorithms. One is distributed in nature called the VC-Based String Round Robin (VCB-SRR) Algorithm. Another is a centralized one called the Channel Group-Based Round Robin (CGB-RR) Algorithm. After that, we will design the corresponding multichannel switches that support these two algorithms. We will also propose cell resequencing mechanism to reorder out-of-sequence cells at the destination. The proposed algorithm is proven to be robust enough to deal with wrapped-around sequence identifier and cell loss in the switching environment. No false resequencing or deadlock will occur in the mechanism. Finally, the designed switches will be applied as the switch modules in SpiderNet. Switching cell loss, resequencing cell loss, and delay performance of the interconnection architecture will be studied at the end.

Contents

Chapter 1

Introduction

Broadband Integrated Services Digital Network (BISDN) is the emerging telecommunication network promised to carry a very broad range of network services. Network applications ranging from low bit-rate voice transfer to high-bandwidth video distribution, and from constant bit-rate circuit-emulation to bursty data services, can be accommodated in this single network. The flexibility of this network is gained by the Asynchronous Transfer Mode (ATM) in which information is carried by fixed size packets called cells through its slotted transfer mechanism. It defines the BISDN to be a Quality-of-Service- (QoS-) guaranteed packet-switched network.

While it is anticipated that future BISDN central offices might need to provide thousands of lines [7], only small scale ATM switches have been implemented or are commercially available. The size of an ATM switch is either limited by physical constraints like I/O circuitries and connector space, or technological constraints like memory or bus speed. In view of this problem, researchers have been proposing large scale ATM switch architectures through the

interconnection of small switch modules [10, 21, 23]. Below, we give a brief review of the major large scale ATM switch architectures proposed so far in the literature.

1.1 Multistage Interconnection Architectures

Many interconnection architectures proposed are multi-stage architectures. They may contain two, three or more stages. A well-known two-stage switch architecture was proposed by Lee [21]. It is a nonblocking switch for supporting thousands of ATM ports. Its inputs are grouped into equal partitions. Each partition is a nonblocking Batcher-Binary-Banyan switch module. They comprise the first stage of the switch. Outputs of these modules are multiplexed at the second stage to their targeted output ports. A problem with this structure is that the Batcher modules require very stringent synchronization between stages of routing elements.

The three-stage Clos network is another multi-stage architecture. In this architecture, each module in the first stage and the third stage has more than one interconnection links to the middle stage modules. Therefore, there are multiple paths between any pair of input and output modules. It was proved in [15] that the switch fabric can be internally nonblocking if the routing paths are determined on a cell-by-cell basis. Also, it gives a higher throughput by allowing cells to be routed to alternate paths. Yet it requires almost double the hardware than two-stage architectures, and needs mechanisms to ensure that cells are delivered in sequence [31, 6, 20].

Liew and Lu [23] proposed another three-stage switch architecture which

is illustrated in Figure 1.1. In order to provide high throughput as well as low cell loss probability, input expansion (i.e. $m > n$) and channel grouping are required to increase the internal switching capacity of the switch. By having input queueing in the first stage, packet dropping in the second stage, and output queueing in the third stage, cell sequence along the channel-grouped switching paths can be maintained.

Another multi-stage interconnection architecture was proposed by Eng *et al.* [10]. Two design principles that guarantee high throughput and low interconnect complexity are realized in the prototype. They are the Generalized Knockout Principle and the Output Queueing Principle. The first one exploits the statistical fact that, in a uniform traffic environment, it is very unlikely that a majority of the input packets will content for the same output port (or the same group of output ports) simultaneously. Thus, an $N \times N$ switch only needs to sweep at most m out of its N input packets to a group of n output ports ($N >> m > n$). The resulted cell loss probability can be reduced to a certain level by setting suitable values of m and n. For example, when $N \rightarrow \infty$, $m = 33$ and $n = 16$, cell loss probability can be kept below 10^{-6} for 90% load [10]. Small increment in m can significantly reduce the cell loss probability by one or two orders of magnitude. For the output queueing principle, [17] has proved that queueing of packets at output ports can result in higher throughput because of the absence of HOL blocking. Based on these two principles, a high-performance prototype 2.5Gb/s ATM switch was implemented [11].

There are several drawbacks in the above multi-stage architectures. First, these architectures place the input ports at one side and the output ports at the other side. They require the rewiring of I/O or internal links whenever

new modules are added. Because of this, one-sided (or folded) architectures are proposed which put the input and output ports together at the first stage. The back stages are operated for routing purpose. Arranging the switch modules in this way, modules can be added without the rewiring of the existing modules [12]. Second, the growth of multi-stage architectures is limited by stages of expansion which is nonlinear in general. For example, we cannot grow a switch from 1024x1024 to 1056x1056 by adding one or a few 32x32 modules. In fact, multi-stage architectures have to grow by adding in stages of intermediate modules. These intermediate modules do not have their own I/O ports; they serve to route the cells for other modules only. Finally, multi-stage architectures require a three-dimensional structure. Stages of switches grow from the inside of a switch. The structure takes up significant space and requires complicated mechanical and wiring designs to ensure that modules are properly connected. When one of the intermediate modules fails, it is difficult to extract the module for servicing without affecting other modules.

1.2 Interconnection Topologies

We learnt from the previous section that putting input and output ports together at the same module will make a switch grow without affecting existing wirings. Under this arrangement, switch modules also have extension ports to connect other modules through a specific interconnection topology. There are many possible topologies, but a good one should give a high switching capacity, and offer stepwise-scalability for the switch architecture.

Let us review some network topologies such as the linear bus and ring. They

are shown in Figure 1.2. We first note the importance of the average node-to-node distance in these topologies. This value indicates how much network resource is taken on average by the establishment of a new call over the network. The larger the value, the more likely that the network can hold fewer calls. We use the word 'more likely' because the trunking efficiency that the network offers is also another important factor affecting the capacity.

For a linear topology with N nodes as shown in Figure 1.2a, the table below shows the node-to-node distance between any two nodes in the network.

	Node 0	Node 1	Node 2	Node 3	...	Node (N-1)
Node 0	0	1	2	3	...	N-1
Node 1	1	0	1	2	...	N-2
Node 2	2	1	0	1	...	N-3
Node 3	3	2	1	0	...	N-4
⋮	⋮	⋮	⋮	⋮	⋱	⋮
Node (N-1)	N-1	N-2	N-3	N-4	...	0

Summing up all the entries and taking the average, the average node-to-node distance $D_L(N)$ is found equal to

$$\frac{N^2-1}{3N}$$

For a ring topology as shown in Figure 1.2b, assuming connections are made via the shorter path, the average node-to-node distance $D_R(N)$ is

$$\begin{cases} \frac{N}{4}, & N \text{ is even;} \\ \frac{N^2-1}{4N}, & N \text{ is odd.} \end{cases}$$

We see that the ring topology gives a shorter average node-to-node distance. Thus the ring topology can support more calls than the linear topology. However, both of their node-to-node distances still have the same order of $O(N)$.

Another topology has a higher degree of interconnection than the previous two. It is the Bidirectional Manhattan Street Network (BMSN) which was derived from Maxemchuk's Manhattan Street Network (MSN) in [24]. Figure 1.3 illustrates its network layout. The nodes are arranged in a 2D grid structure. Each connects with its neighbours horizontally and vertically. At the edges of the structure, the connections are looped around from one edge to the opposite edge. This forms a ring on each row or column of the network. The network becomes a torus structure.

With this toroidal topology of BMSN, the average node-to-node distance is further reduced. To find its value, we can refer again to Figure 1.3. With m rows and n columns of nodes, we observe that no matter based on which node we are viewing, we get the same toroidal arrangement of nodes. The average node-to-node distance can therefore be found by simply finding the average distance from any one node to any other nodes over the network. Assuming uniform distribution of traffic, the average node-to-node distance $D_B(m,n)$ of the BMSN is equal to

$$\begin{cases} \frac{m}{4}+\frac{n}{4}, & m \text{ and } n \text{ are both even;} \\ \frac{m}{4}+\frac{n^2-1}{4n}, & m \text{ is even}, n \text{ is odd;} \\ \frac{m^2-1}{4m}+\frac{n}{4}, & m \text{ is odd}, n \text{ is even;} \\ \frac{m^2-1}{4m}+\frac{n^2-1}{4n} & m \text{ and } n \text{ are both odd.} \end{cases}$$

In particular, if $m = n = \sqrt{N}$, (m and n are positive integers), the average

node-to-node distance will become

$$\begin{cases} \frac{\sqrt{N}}{2}, & \sqrt{N} \text{ is even;} \\ \frac{N-1}{2\sqrt{N}} & \sqrt{N} \text{ is odd.} \end{cases}$$

As different from the linear and the right topologies, the node-to-node distance of BMSN has an order of $O(\sqrt{N})$.

We can think of more sophisticated network topologies that can reduce the average node-to-node distance. These network topologies can then be employed for increasing the switch capacity in a large scale switch interconnection architecture.

1.3 Design of Switch Module–An Example of Multichannel Switch

A switch module that supports both input/output ports as well as extension channel groups is in fact an example of the multichannel switch [27]. In a multichannel switch, several ports are bundled together to form a logically high-bandwidth transmission channel for inter-switch traffic. This strategy is called *channel grouping* or *trunk grouping* [28], and the logical channel formed is the *channel group* or *trunk group*. With the high transfer capability provided by channel groups, multichannel switches are very suitable for use in a large scale switch interconnection architecture.

Typical ATM switches can be modified to become a multichannel switch. In Pattavina's switch, each switch port is given an additional address called the channel address on top of the port address. Ports belonging to the same

channel group will have contiguous channel addresses for better address organization, and easy recognition of their channel group. The Batcher-Banyan switch is modified to carry out switching of cells over channel groups using the channel addresses provided. Also, a round-robin channel allocation algorithm is incorporated to evenly distribute cells over the multiple links of the channel groups.

With parallelized transfer of cells over a channel group, cells can get out-of-sequenced. There are several papers dealing with this problem. In [5], the time order of cells from the top to the bottom channels of a channel group is maintained when cells are transferred. Since output buffering is used, the time order is maintained at the array of output buffers of the channel group. This is done by introducing dummy cells at the HOL of the output buffer to avoid a cell being retrieved too early. In [19], a single virtual FIFO memory queue is placed inside the switch. All cells that attempt to get switched to their output channels are first queued up at the FIFO queue. They are then switched to their particular channel groups according to the time order.

The price to pay for guaranteeing the cell sequence integrity in a multichannel switch is that traditional point-to-point ATM fabrics cannot be used *directly.* They have to be modified to incorporate algorithms for allocating channels and maintaining cell sequence.

1.4 Organization

The major work of the thesis is described in chapter 2 and 3. In Chapter 2, we start out by describing BMSN. Issues such as switch module design, call routing by relative addressing, cell routing over the interconnection network will

be addressed. We will then extend BMSN into SpiderNet in which each switch module connects eight instead of four neighbours. Call blocking, cell loss, delay performance of both switches will be studied based on simulations.

In Chapter 3, we will introduce the multichannel switch design strategy, and discuss the basic components of a multichannel switch. Their major functions are channel allocation and cell resequencing. Two channel allocation algorithms, one centralized and one distributed in nature, will be discussed. They will be employed together with their cell resequencing mechanisms to form two proposals of multichannel switches. These switches will be used as the switch modules for SpiderNet. Performance impact of the channel allocation algorithms as well as the cell resequencing mechanisms on the cell loss and delay inside the switch architecture will be studied.

Finally, we conclude our study in Chapter 4.

1.5 Publication

The results of Chapter 2 have been published in IEEE ICC '97.

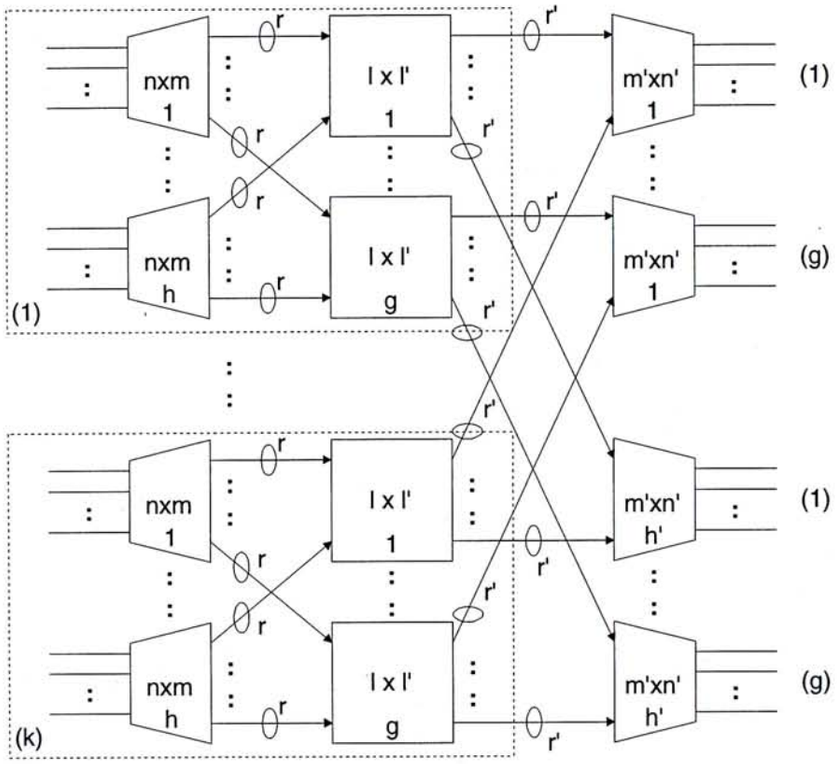

Figure 1.1: Three-stage Interconnection Architecture proposed by Liew and Lu

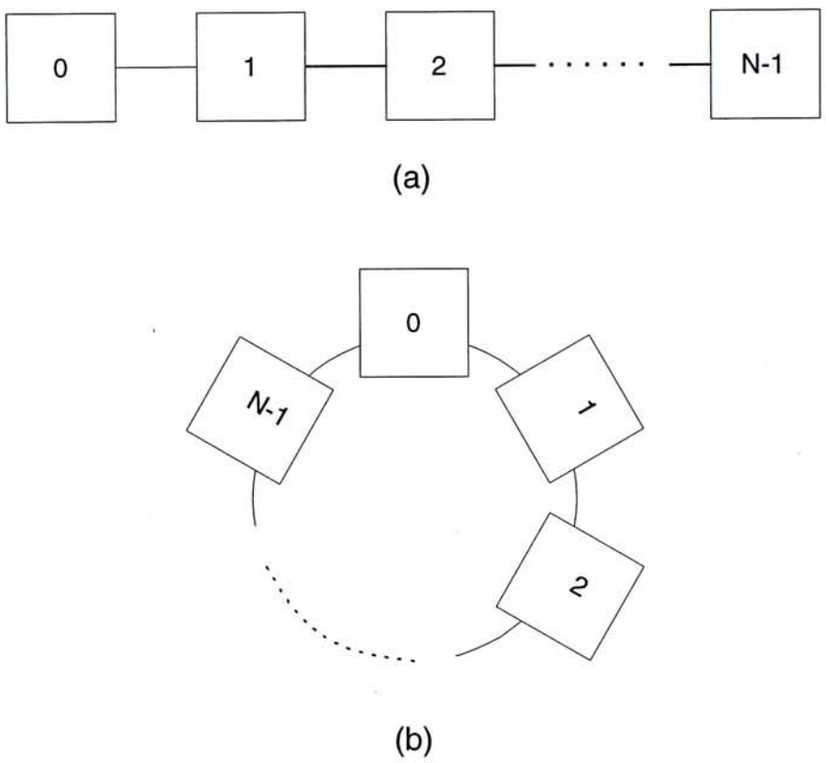

Figure 1.2: (a) Linear topology; (b) Ring topology

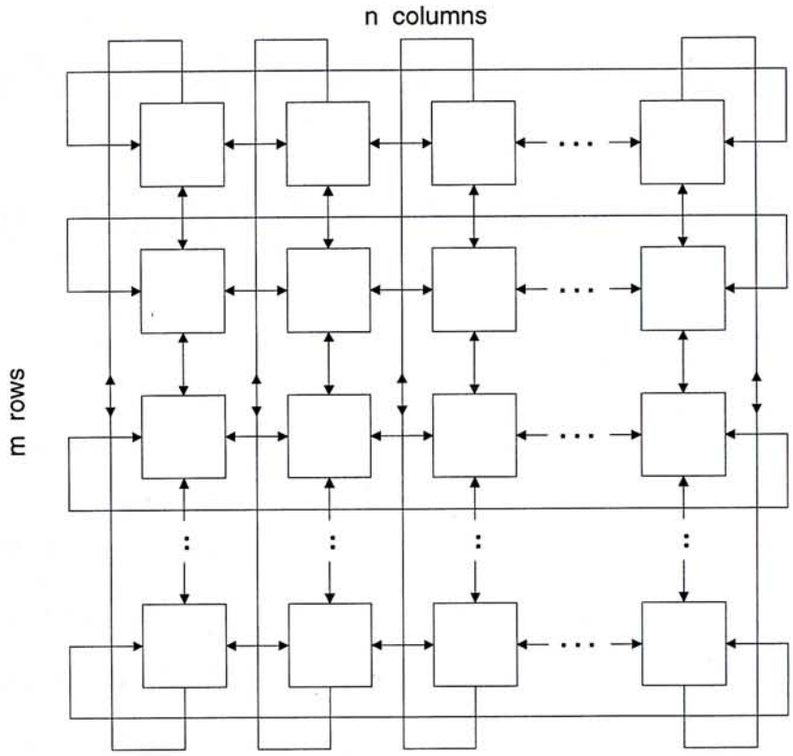

Figure 1.3: Topology of Bidirectional Manhattan Street Network

Chapter 2

BMSN and SpiderNet: Two Large Scale ATM Switches

2.1 Introduction

In this chapter, we consider the use of the Bidirectional Manhattan Street Network (BMSN) for interconnecting fixed size modules into a two-dimensional switch structure. BMSN was derived from the Manhattan Street Network (MSN) [24] originally proposed as a packet switched metropolitan area network. Here, each node is an ATM module having its own input and output ports, and has four extension trunks for connecting to its four neighbouring modules. The switch can grow into a two-dimensional structure which is easy to maintain. The overall switch size and capacity can scale up in a stepwise manner, and there is no need of rewiring the I/O ports and most of the internal links when new modules are added. Most existing service therefore will not be interrupted.

We consider also an extension of BMSN into another architecture called SpiderNet, which connects eight instead of four neighbours. In doing so, more direct paths are available, and the average path length of each connection is reduced. Consequently, SpiderNet gives a higher switch throughput and a lower cell loss probability.

2.2 Architecture

2.2.1 Topology

Figure 2.1 shows the topology of BMSN in which there are vertical and horizontal links that connect the modules to form a two-dimensional torus structure. The extreme nodes are connected with each other by looping around the links on one side to the opposite side. This loop-back has two advantages. First, the maximum hop count between any two modules is reduced. Secondly, the routing function is simplified as it only depends on the relative location between the source and the destination modules. Consider an interconnection of N_R rows and N_C columns. Suppose every module has an absolute address(e.g. as depicted in Figure 2.1). The absolute addresses of the source and destination modules are (X_s, Y_s) and (X_d, Y_d) respectively. We see that the interconnection pattern can be rotated to such a view that the source module is positioned at or near the centre of the graph. We say that this module has a relative address (RA) of $(0, 0)$. At the same time, the address of the destination can be transformed to an address (X_r, Y_r) relative to the source where X_r and Y_r can be computed by

the following formulae.

$$X_r = (X_d - X_s + \lfloor \frac{N_C}{2} \rfloor) \bmod N_C - \lfloor \frac{N_C}{2} \rfloor \quad (2.1)$$

$$Y_r = (Y_d - Y_s + \lfloor \frac{N_R}{2} \rfloor) \bmod N_R - \lfloor \frac{N_R}{2} \rfloor \quad (2.2)$$

Call routing from the source to the destination can then be based on (X_r, Y_r). Figure 2.1 also shows the relative address for every module with Module $(2, 2)$ as the source module.

2.2.2 Switch Modules

Figure 2.2 shows the internal structure of a module in BMSN. Each module has m pairs of input and output ports, and four pairs (incoming and outgoing) of trunks for connecting its four neighbours. Each output port or outgoing trunk has an output buffer. Each module is therefore assumed to be an output-buffered switch. Both incoming and outgoing trunks have r times the bandwidth of a link, where r is referred as the *trunk size*. A trunk can be implemented with r links, or with a high speed link which rate is r times of a link. We denote each module as an (m, r) module which consists of two submodules: the Input Port Controller Submodule (IPCSM) and the Routing Submodule (RSM). The IPCSM consists of m Input Port Controllers (IPCs) which process the incoming cells from local inputs and generate routing tags for guiding the cells through the switch. The RSM is a simple $(m + 4r) \times (m + 4r)$ shared memory fabric. For example, if a $(16, 12)$ switch module is to be implemented, a 64×64 RSM is needed.

We propose here a shared memory fabric for RSM. First, memory fabric implemented as single-stage switches are of lower complexity when compared

with space-division switches [12]. This is because both queueing and switching are carried out via the shared buffer. Second, the use of a shared memory fabric can guarantee that cells are delivered in sequence even if we use parallel links for switching the cells. The memory is partitioned into FIFO queues for each local output port and each extension trunk. At each time slot, cells are stored one by one into the shared memory into their output queues, according to the input link order (Figure 2.3). So the cell from link 1 is stored first, cell from link 2 is stored second, and so on. Note that link order is mandatory for links within the same group, but is not necessary for links belonging to different groups. So different link groups may store their cells in parallel. On the other hand, cells from the output queue are transmitted one by one to the output port or link group. In the case of a link group, the first r cells (or less if there are less than r cells in the queue) will be transmitted on the r links in the same group. Again, cells are transmitted according to a link order, i.e., cell 1 is transmitted on link 1, cell 2 is transmitted on link 2, and so on.

With the above arrangement, the time order of cells is maintained when cells are switched by each module. In fact, there is an implicit time order for two cells arriving on link a and b ($a < b$) of the same link group at the same time slot. The cell on link a is of a *higher time order*. If both cells belong to the same connection, the cell on link a is supposed to be earlier. It will be stored at an earlier position in the shared queue, and will be delivered to an output port at an earlier time slot or to an output link with a higher time order. As cells maintain their order at each stage of switching, the cell sequence can be guaranteed on an end-to-end basis if cells of the same connection are routed through the same path (i.e., the same sequence of links).

One may concern about the hardware complexity of implementing the switch module. The challenge is on the implementation of the RSM. As we will see in Section 2.5, in order to satisfy some performance requirements, an RSM of size 64×64 is needed for a 1024×1024 switch. If we assume the port speed of the RSM is 150Mbps (cell's duration is about 2.83 μs), the shared-memory switch fabric should be able to perform a *read* and a *write* operation pair for 64 times in 2.83 μs. This means a memory access time of 22.1ns is needed. Technically, this can be realized [11].

2.3 Routing

In most research on switching architectures, only cell routing is concerned as they are either single-stage architectures or they have only one path for each pair of input and output. In our architecture, there are multiple possible paths between each pair of modules. When a module receives a cell from a local input port, it routes the cell through a certain path inside the network until the cell reaches the destination module, where the cell is delivered to the corresponding local output. To ensure that cells are delivered in sequence, it is desirable that cells are all routed with a path pre-established during connection setup. We have therefore two levels of routing: Virtual Path/Virtual Channel Routing, in which we concern how to allocate VP/VC paths inside the network, and Cell Routing, in which we concern how cells are actually routed on the predefined VP/VC path.

2.3.1 VP/VC Routing

Figure 2.4 shows a recursive algorithm to determine a path to accommodate a new VP/VC call so that all subsequent cells can be routed. Without loss of generality, source module A is located relatively at the lower left part of destination module C, and A has a relative address of $(0, 0)$. The relative address (X_r, Y_r) of C can be computed as discussed previously. To find the optimal path from A to C, for example, we first branch out in several directions towards C. For example in the figure, a path is branched out from A to its neighbour B in the direction d. By treating B as the new source module and solving out the optimal path from B to C, the optimal path from A to C through B can be determined. Other possible paths that connect A to C through some other neighbours of A are computed in a similar way. Normally, there will be several paths having sufficient spared bandwidth for accommodating a new VP/VC. We therefore need some criteria for selecting an optimal one. Here, we consider two simple criteria. First, we choose the shortest paths (SP) which have the minimum number of links. This minimizes the number of links for supporting each VP/VC. Secondly, among the shortest paths, we choose the path which has the maximum spare bandwidth (MSB), so as to distribute the traffic load uniformly over the links. We define the spare bandwidth of a path as equal to the spare bandwidth of the mostly congested link of the path.

In the routing procedure, the first step in each recursive level is to determine the set $\mathbf{D}$ of branching directions d in which the path should be augmented/branched out from a module towards the destination. Obviously, the elements in $\mathbf{D}$ should depend on the relative location of the destination. To reduce search complexity, we confine our search directions towards the destination

module only. The algorithm partitions the relative location into eight regions, and defines the proper set **D** for each one of them. The regions are

First Quandrant $(X_r > 0, Y_r > 0)$, **D** = { E, N }

Positive Vertical Axis $(X_r = 0, Y_r > 0)$, **D** = { N }

Second Quandrant $(X_r < 0, Y_r > 0)$, **D** = { N, W }

Negative Horizontal Axis $(X_r < 0, Y_r = 0)$, **D** = { W }

Third Quandrant $(X_r < 0, Y_r < 0)$, **D** = { W, S }

Negative Vertical Axis $(X_r = 0, Y_r < 0)$, **D** = { S }

Fourth Quandrant $(X_r > 0, Y_r < 0)$, **D** = { S, E }

Positive Horizontal Axis $(X_r > 0, Y_r = 0)$, **D** = { E }

For example, if the destination module is located at the first quandrant, the possible directions from the current module are confined to be { E, N }. So the next level of recursion will be repeated at the east and north neighbours. In this way, the paths will be augmented module-by-module until the destination is reached. Below gives the recursive routing algorithm based on the SP-MSB criterion:

Definition

(X_s, Y_s): Absolute address of the source module.

(X_d, Y_d): Absolute address of the destination module.

(X_c, Y_c): Absolute address of the new source module(we call it the *current module*) in the current recursive level. At the beginning, it is assigned as (X_s, Y_s).

(X_r, X_r): Relative address of the destination module with respect to the current module.

$P^d_{(x,y)}$: Optimal path that connects the module with absolute address (x, y) to the destination through branching at direction d. It is a set of trunks that constitute the path.

$l^d_{(x,y)}$: Outgoing trunk of direction d branched out from the module with absolute address (x, y).

$B_L(l)$: Spare bandwidth of outgoing trunk l.

$B_P(p)$: Spare bandwidth of path p. Defined as $\min_{\forall l \in p} \{B_L(l)\}$.

B_{call} Bandwidth requirement of the call.

$F((X_1, Y_1), (X_2, Y_2))$: Recursive function for finding the optimal path from absolute address (X_1, Y_1) to (X_2, Y_2). If the path is successfully found, it returns the path set;otherwise, it returns the set {-1} indicating that no path is found.

Procedure

Find optimal path P_{opt} where $P_{opt} = F((X_s, Y_s), (X_d, Y_d))$

$F((X_c, Y_c), (X_d, Y_d))$ is defined below:

1. If (X_c, Y_c) equals (X_d, Y_d), return ϕ; otherwise, do the following steps.
2. Calculate (X_r, Y_r) from (X_c, Y_c) and (X_d, Y_d), then determine from it the set of proper branching directions **D** going out from (X_c, Y_c).
3. For every $d \in \mathbf{D}$,
   (a) Augment the path through $l^d_{(X_c,Y_c)}$ from (X_c, Y_c) to its neighbour. Then get its neighbour's absolute address and denote it as (X^d_n, Y^d_n).
   (b) Call $F((X^d_n, Y^d_n), (X_d, Y_d))$, and store the returned result in a temporary set T^d.
4. For every $d \in \mathbf{D}$ with $T^d \neq \{-1\}$, $P^d_{(X_c,Y_c)} = \{l^d_{(X_c,Y_c)}\} \cup T^d$.
5. From all $P^d_{(X_c,Y_c)}$'s obtained in 4, return $P^{d'}_{(X_c,Y_c)}$ of direction d' that has $B_P(P^{d'}_{(X_c,Y_c)}) \geq B_{call}$, and yet is the shortest path. If there are more than one such shortest path, return the one with the maximum spare bandwidth. If no path can accommodate the call, return {-1}.

When the number of modules on a row N_R or on a column N_C is even, there may be more than one region we need to search. Consider Figure 2.1, the shortest path from RA:(0,0) to RA:(-2,0) can be found not only by going *west* but also *east*. To force searching to the *east*, we bias the source to the left from the centre, making the destination appear at the *Positive Horizontal Axis*. If both N_R and N_C are even, in the worst case, we have to bias the source module upward, downward, to the left, and to the right, search out a total of four optimal paths, and select the *best* one among them. This happens, for example, when we need to connect modules (0,0) and (2,2) in Figure 2.1.

Biasing the source module is achieved by modifying formula 2.1 and 2.2 to the following ones:

$$X_r = (X_d - X_s + \lfloor \frac{N_C \pm \Delta_C}{2} \rfloor) \bmod N_C - \lfloor \frac{N_C \pm \Delta_C}{2} \rfloor \tag{2.3}$$

$$Y_r = (Y_d - Y_s + \lfloor \frac{N_R \pm \Delta_R}{2} \rfloor) \bmod N_R - \lfloor \frac{N_R \pm \Delta_R}{2} \rfloor \tag{2.4}$$

where Δ_C and Δ_R are any real numbers in $(0, 1)$. The term $\pm\Delta_C(\pm\Delta_R)$ will lead to the same relative address map when $N_C(N_R)$ is odd. However, when N_C(or N_R) is even, the $+$ and $-$ will give respectively the right- and left-biased maps (or upward- and downward-biased maps).

We use these two formulae when multiple 'optimal' paths exist, that is, when N_C(and/or N_R) is even, and the destination is $\frac{N_C}{2}$ columns(and/or $\frac{N_R}{2}$ rows) away from the source.

2.3.2 VP/VC Routing Control

We assume that there is a centralized call processor which maintains the status of all links inside the network. When it receives a VP/VC connection request, the processor performs the routing algorithm to determine the optimal routing path. An alternative is to use a decentralized approach. The source module will send out resource management (RM) cells to other modules to search for possible routing paths. Other modules will check their own link utilizations, reserve the bandwidth, and return a candidate path if available. The source module will then choose the optimal routing path and send a RM cell to confirm that path. It can send a release RM cell to those candidate paths not chosen, or simply let those paths timeout and release their resources.

The centralized approach can greatly simplify routing processing, as individual modules do not need to maintain link utilization and handle call processing. Nevertheless, we have now a single point of failure, and too many VP/VC requests may overload the call processor. The first problem can be solved by having a standby processor which mirrors all call information. When the active processor fails, the standby processor replaces its role. The second problem can be eased by having the source modules reserve VP bandwidths whenever possible, so the loading on the call processor will be greatly reduced. When a source module needs a new VC, it first searches through the existing VPs to see if they can accommodate the new VC. If so, the source module can directly establish a VC to the destination module without the intervention of the call processor. Otherwise, the source module can always request another new VP to accommodate the new VC. Note that the use of VP in this sense is somewhat different from the definition of VP in the ATM and BISDN standards. In the

standards, VPs are end-to-end paths carrying a number of VCs. Here, VP is just a reserved bandwidth inside the fabric connecting a source to a destination module.

2.3.3 Cell Routing

Once a VP/VC is established, cells are routed hop-by-hop from the source to the destination module. There are two approaches for cell routing. The first approach is to let each switch process the VP/VC labels to determine the next routing step. This means that each module is somewhat like an independent ATM switch for cell routing, and each module has full VP/VC processing and port translation capability even for the extension trunks. The second approach is to let the source module define the routing tags for guiding the cells through the switch. This is known as *source routing.* Intermediate modules will then be much simplified.

Figure 2.5 describes the format of a routing tag defined for each VP/VC, which consists of a hop-count field (HC), a sequence of routing units $RU\#n..RU\#1$, and an output port address field (OPA). The HC indicates the number of hops that a cell is to be routed, which is initialized to n by the source module. During each hop of routing, the module will check if HC=0, which indicates that the module itself is the destination. In this case the cell will be delivered to the targeted output port according to the OPA field. Otherwise the module will decrement HC by one, and use the next RU for routing. Based on the RU, a module can tell which trunk a cell should be forwarded. For BMSN, there are four outgoing trunks from each module, so two bits are sufficient for each RU. As the cell is routed to the outgoing trunk, the corresponding RU is striped

from the routing tag, leaving the remaining RUs for subsequent routing.

Figure 2.6 gives a routing example of routing a cell from module (1,1) to (3,3). The routing tag value is $<HC$=4, $RU\#4$=01, $RU\#3$=00, $RU\#2$=01, $RU\#1$=00, OPA=1011>. We see that the cell is first routed east (by the source module), then north, then east, then north. The module (3,3), on receiving the cell, will find that HC=0, indicating that it is the destination module. It delivers the cell to the port address 1011.

Instead of striping away the used RUs, we may choose to use a fixed size routing tag which has the maximum number of RUs for a given size of network. During each stage of routing, the module can reference to the corresponding RU based on the value of HC. For example when a module sees HC=3, it will reference the last third RU value in the routing tag. Having a fixed size tag simplifies cell synchronization and avoids the complexity for striping the RUs, but network expansion will be limited unless we set a large tag size.

2.3.4 Alternate Path Routing for Fault Tolerance

In BMSN, modules are connected by bi-directional trunk. Under normal operation, neighbouring modules can send status cells to one another. When a module or a link in a trunk fails, the neighbouring modules will detect immediately and send error cells to the centralized call processor. Identifying the cause and impact of failure, the call processor may reconfigure paths to reroute the affected calls, or terminate some lower priority connections if necessary.

2.4 SpiderNet

Figure 2.7 shows a module in SpiderNet which connects 8 neighbours instead of 4 as in BMSN. It shortens the routing paths by connecting more modules with direct paths. Since the number of extension trunks doubles, assuming the same size of RSM as in BMSN, the trunk size of SpiderNet has to be reduced by half. For the VP/VC routing algorithm, the sets of branching directions **D** have to be changed. We have

First Quandrant $(X_r > 0, Y_r > 0)$, **D** = { E, NE, N }

Positive Vertical Axis $(X_r = 0, Y_r > 0)$, **D** = { NE, N, NW }

Second Quandrant $(X_r < 0, Y_r > 0)$, **D** = { N, NW, W }

Negative Horizontal Axis $(X_r < 0, Y_r = 0)$, **D** = { NW, W, SW }

Third Quandrant $(X_r < 0, Y_r < 0)$, **D** = { W, SW, S }

Negative Vertical Axis $(X_r = 0, Y_r < 0)$, **D** = { SW, S, SE }

Fourth Quandrant $(X_r > 0, Y_r < 0)$, **D** = { S, SE, E }

Positive Horizontal Axis $(X_r > 0, Y_r = 0)$, **D** = { SE, E, NE }

Finally, the following checking mechanism has to be added in the recursive routing procedure, making sure that the augmentations will direct toward the destination module.

0. Check whether (X_c, Y_c) is moving closer to the destination than the previous module. Continue the following steps only if the condition is satisfied; otherwise, return {-1}.

Figure 2.8 gives a scenario to show what happens if the check is not done. In the figure, module A is located at the west of the destination module D. Assume that paths are to be augmented from A. Since D is on the *positive*

horizontal axis relative to A, A should augment the paths in {SE, E, NE} as given earlier. So assume that a path is first augmented to module B. (Note that B has the same distance from D as that of A. As D is on the *negative vertical axis* of B, B could augment the paths in {SW, S, SE}. This means that B could be augmented to A, forming an infinite loop. The checking step avoids this problem by ensuring that each augmentation will bring the end-point of the path closer to the destination. So B will not be augmented back to A, and no looping occurs.

2.5 Performance and Discussion

In most studies on switching architectures, performance is usually measured by cell loss, delay, or throughput. Here we propose one more measure - VP/VC call blocking probability. This is a useful measure for two reasons. First, ATM is connection-oriented. We concern about the likelihood of a call being blocked, especially due to the inavailability of bandwidth at the interconnection links. Secondly, the new measure parameter can reflect the call-carrying capacity of an ATM switch.

2.5.1 BMSN *vs* SpiderNet

We have built simulation models for both BMSN and SpiderNet. For fair comparison between them, in both models, we use 64×64 memory fabrics, each of which provides 16 pairs of local I/O ports and 48 pairs of extension ports. 64 such modules are interconnected to implement a 1024×1024 switch. The topology is 8×8 in dimension.

In ATM networks, VP/VC calls can be of different and perhaps variable bit rates. For simplicity, we assume here that the bandwidth of each link (local or extension) is 150Mbps, and the bandwidth of each call is of constant rate of 5Mbps (unless stated otherwise). The arrival process of calls at the local I/O ports to each module is Poisson and the duration of each call is exponentially distributed. Each call has equal probability of being addressed to any output port. When a call arrives, the switch will check whether the I/O ports of both source and destination modules have enough bandwidth. If so, the switch will try to route the call in the switch. A call is blocked if the route cannot be made, and the call blocking probability is defined as the ratio of blocked calls to the total number of calls that have performed routing.

Figure 2.9 shows the call blocking probability of BMSN and SpiderNet against input loading. We can see that BMSN can support 69% of input loading without severe call blocking, whereas SpiderNet can support up to 91%. Note that we cannot obtain the call blocking of SpiderNet at a higher input loading because many calls have been rejected by the input and output links at either the source or destination module. Given the same number of internal links, SpiderNet has a higher switch capacity than BMSN. This is because SpiderNet has diagonal trunks which can offer more direct paths.

Figure 2.10 gives the cell loss probability of BMSN and SpiderNet under different trunk(or output) buffer sizes B in unit of cells. By increasing the buffer size B, we can reduce the cell loss probabilities of both architectures. They become reasonably low when B=100 for SpiderNet and BMSN at 90% and 67% loading respectively. One interesting observation is that the cell loss probability appears to be less sensitive to the increase or decrease in loading in SpiderNet.

We find from the simulation that, in SpiderNet, cell loss mostly occurs at a congested output port instead of at intermdiate trunks. As connections are randomly established inside the network, some output links will have more traffic than the other links. The loss in these output links will dominate the overall loss probability, even if the loading is slightly increased or decreased. In BMSN, cell loss is reduced very rapidly when input loading is decreased. This is because in BMSN, the internal trunks are more congested than the output port. Cell loss usually occurs at the trunk buffers instead of the output buffers. A decrease in input loading results in general reduction in the trunk utilization. The cell loss will be reduced significantly. We anticipate that the same result will apply to SpiderNet if the internal trunks become the bottleneck of switch capacity.

Figure 2.11 shows the average end-to-end delay for cells that pass through different number of hops HC. The end-to-end delay of a cell is defined as the time elapsed between its arrival at an input port and departure at an output port. We see that the cell delay of both architectures are relatively small even when we have multiple hops of switching. While longer hops imply a larger delay, the delay difference between long paths and short paths is minimal. Roughly speaking, a cell will experience on average one more time slot delay if its path is one hop longer. This is because the queues at the internal trunks are relatively short.

2.5.2 Network Capacity

Effect of Trunk Size

Figure 2.12 shows the effect of increasing the trunk size r on the call blocking probability of a 1024×1024 SpiderNet. With an increase in r, we can achieve a smaller call blocking probability under the same offered load. Figure 2.13 shows the actual port throughput versus offered load with various values of r. We see that if the internal trunks are not overloaded, an increase in load can increase the switch utilization. When offered load reaches a certain level, the switch utilization actually drops with a further increase in loading. This is because when the internal trunks are overloaded, many calls will be routed on longer paths which in turn will affect the overall switch efficiency.

Effect of Localized Traffic Ratio

Figure 2.14 shows the effect of localized traffic ratio on the call blocking probability in SpiderNet. We see that when there is more localized traffic, a lower call blocking probability is obtained for the same offered load. In another words, it means that more calls can be accepted by the switch. This is because when we place the input and output ports at the same module, the localized traffic will not consume any bandwidth of the interconnection network. The saved bandwidth can be used to support more calls.

2.6 Concluding Remarks

In this chapter, we considered two switch interconnection architectures named BMSN and SpiderNet. Both have a grid structure based on torus topology. The nice features of these two architectures include: (1) scalability, modules can be added to increase the switch size and capacity; (2) stackability, no rewiring of existing I/O ports is needed when modules are added; (3) fault-tolerant, multiple alternate paths are available for re-routing; and (4) reasonably good call blocking performance. As there is no need for intermediate switches, all modules carry the same functions. They all have local I/O ports, and extension ports for switch expansion. While it appears that larger switching fabrics (64×64 instead of 16×16) are needed, the additional complexity could be small as only the RSM submodule is affected. The extension submodules do not need ATM VP/VC processing capabilities. Consider that the cost of ATM switches is dominated by interfacing electronics, optical components, processing circuitries and software for control and signaling, the additional hardware cost for using a larger RSM switching fabric can be relatively small.

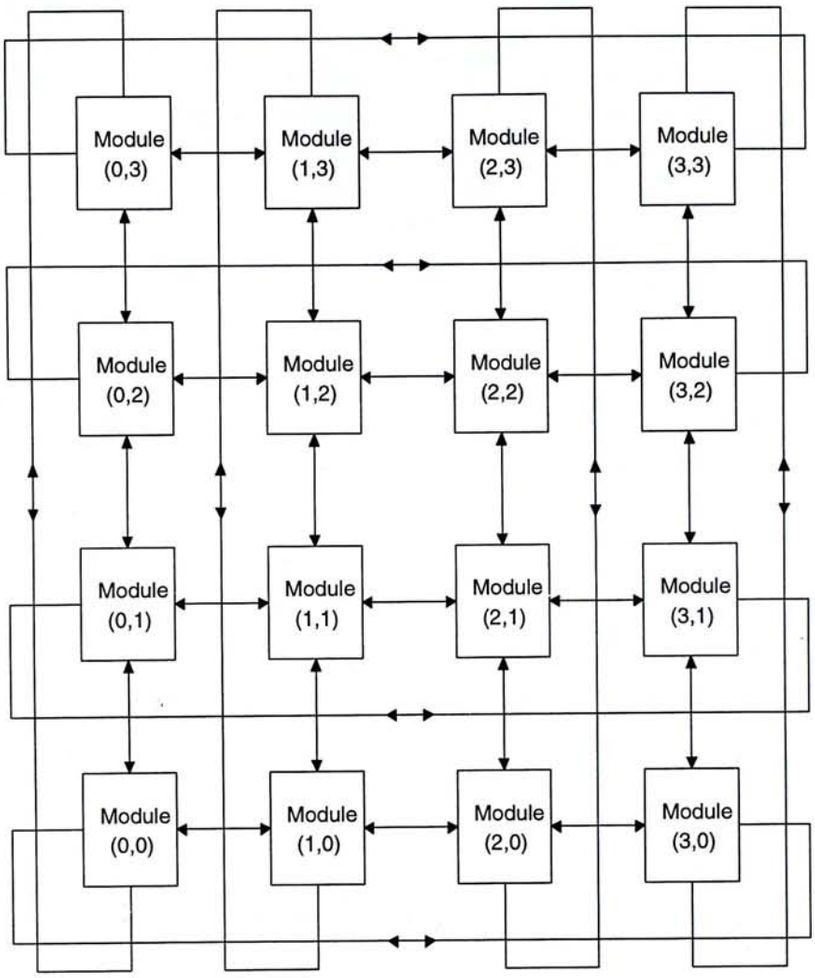

Figure 2.1: A 4×4 BMSN

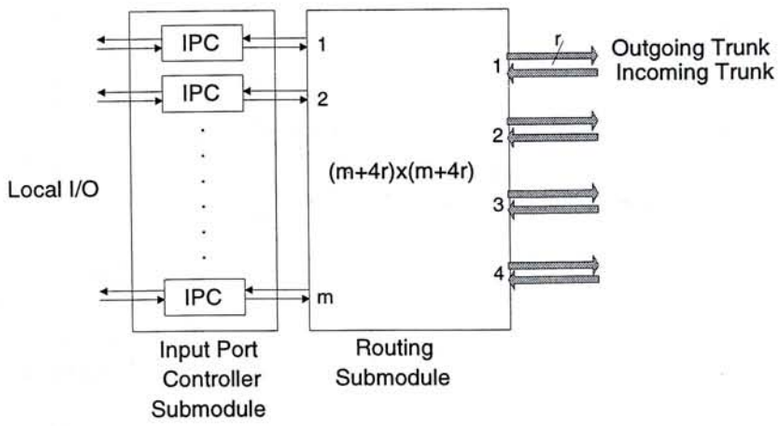

Figure 2.2: Internal Structure of a (m, r) Switch Module for BMSN

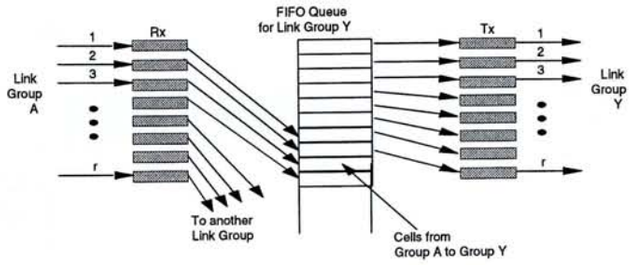

Figure 2.3: Maintaining Link Order When Cells are Stored and Transmitted

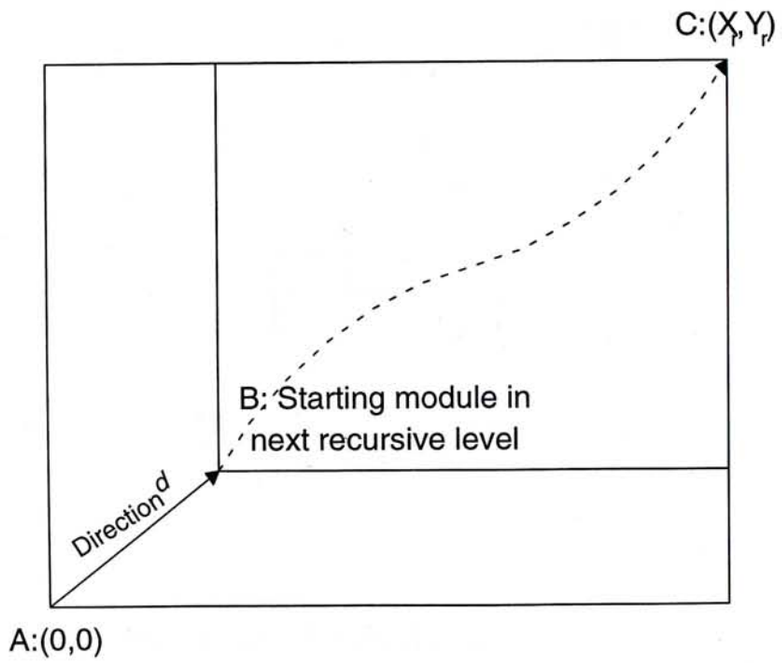

Figure 2.4: Recursive Routing Algorithm

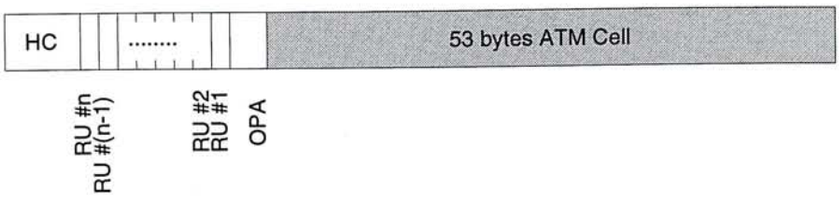

Figure 2.5: Routing Tag

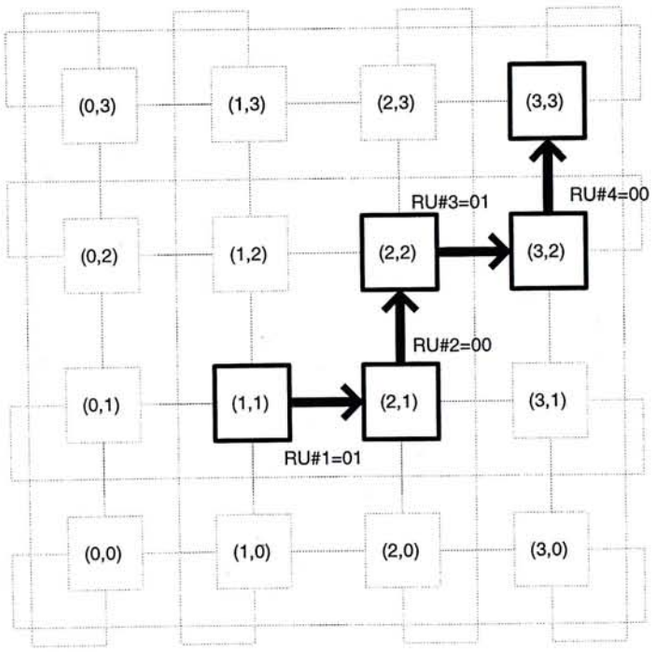

Figure 2.6: A Cell Routing Example

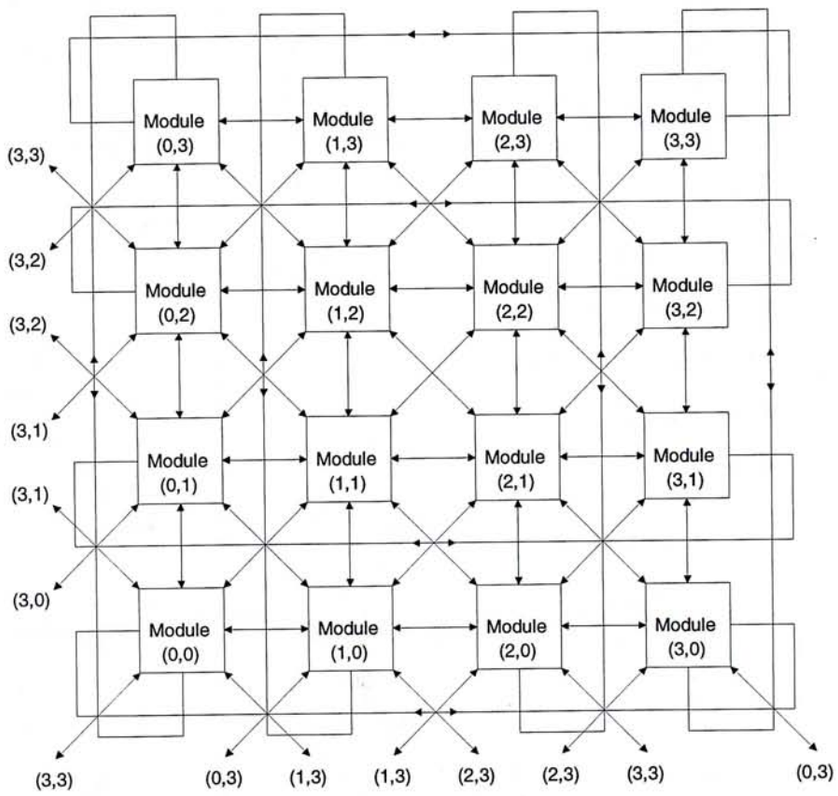

Figure 2.7: A Module in SpiderNet

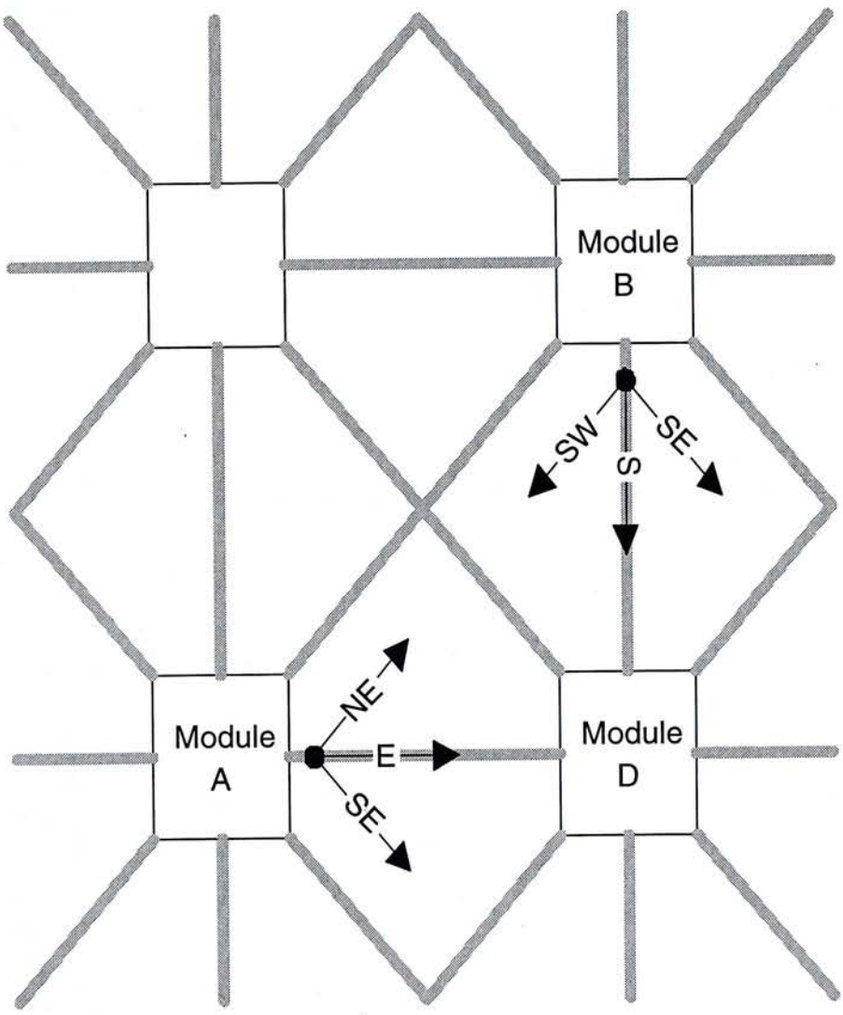

Figure 2.8: Looping Problem in the Routing Algorithm

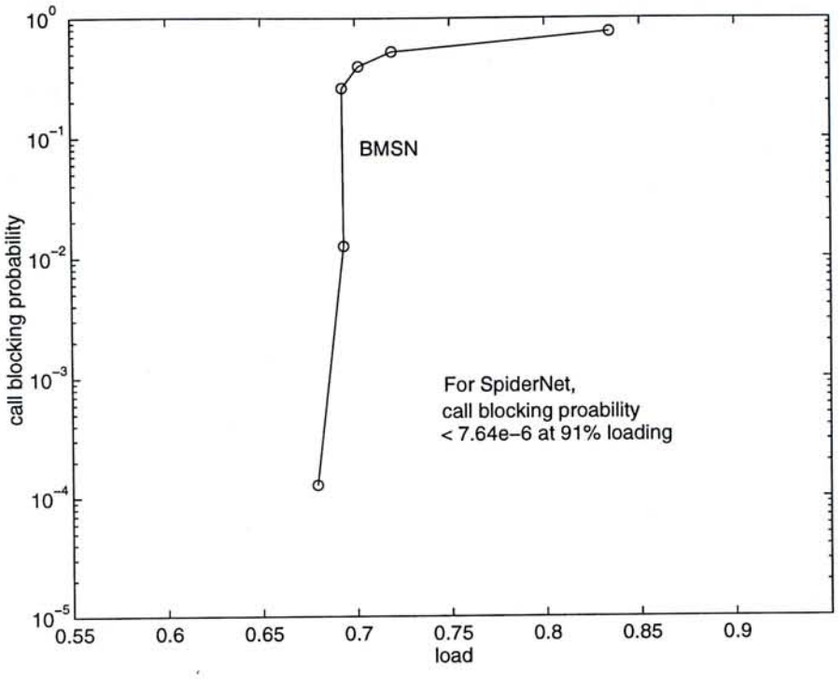

Figure 2.9: Call blocking probability *vs* input loading

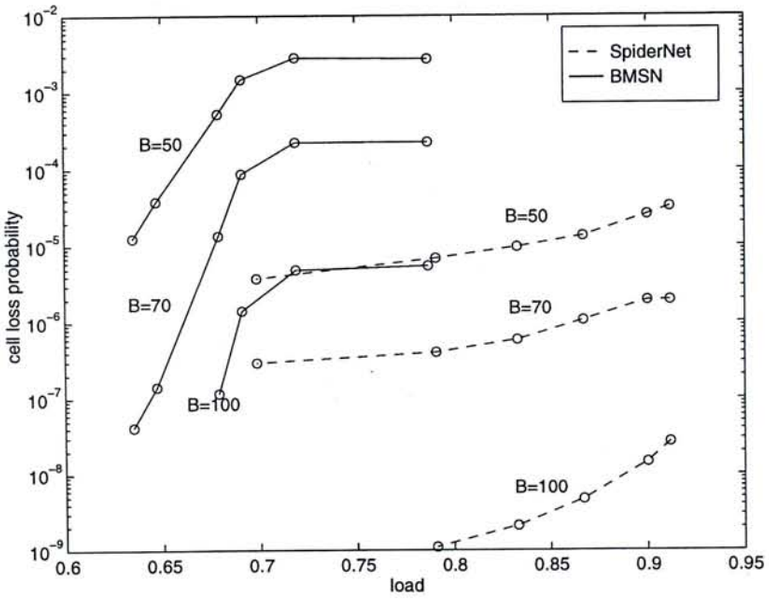

Figure 2.10: Cell loss probability of 1024x1024 BMSN and SpiderNet

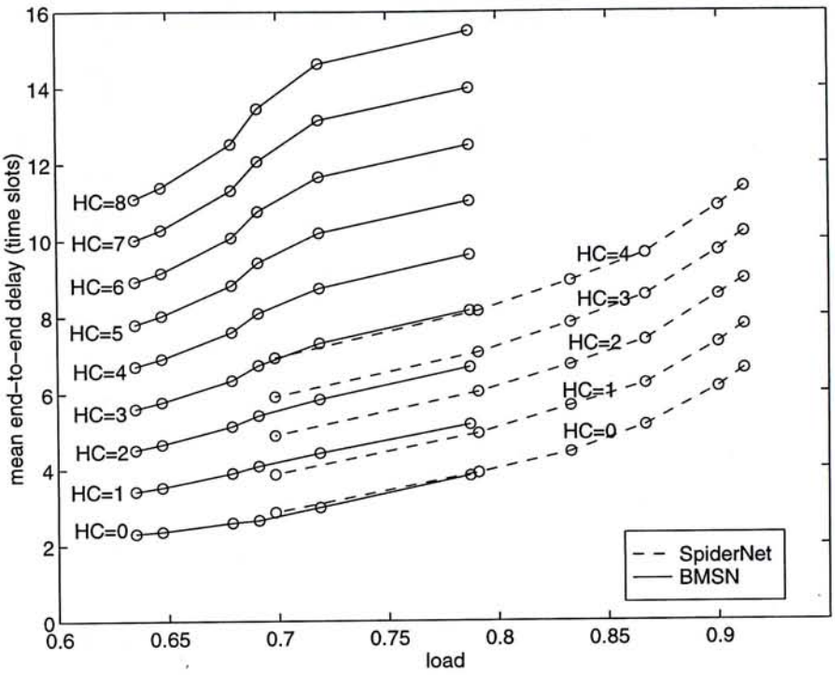

Figure 2.11: Mean end-to-end delay profile *vs* input loading

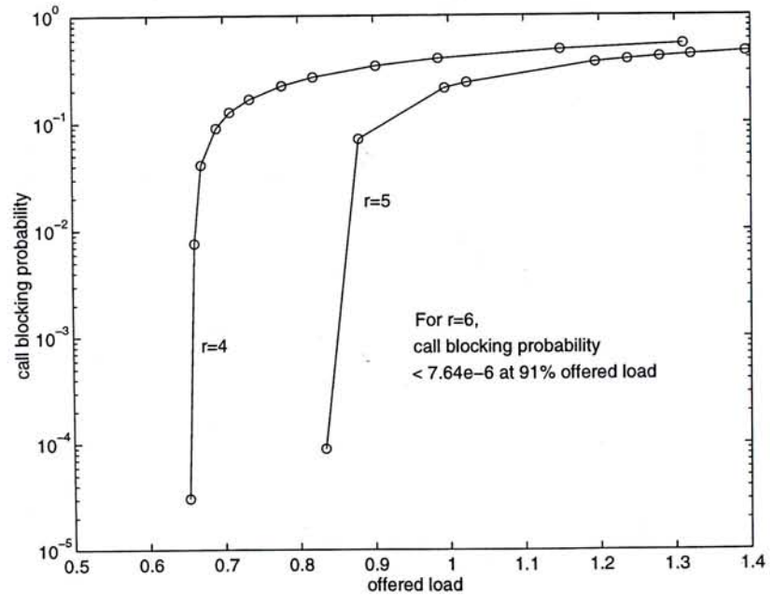

Figure 2.12: Call blocking probability in SpiderNet with different trunk size r

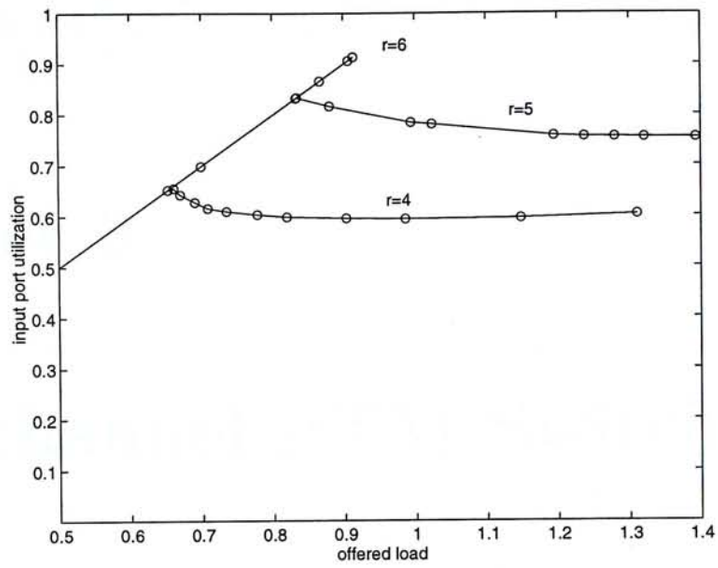

Figure 2.13: Input port utilization *vs* offered load

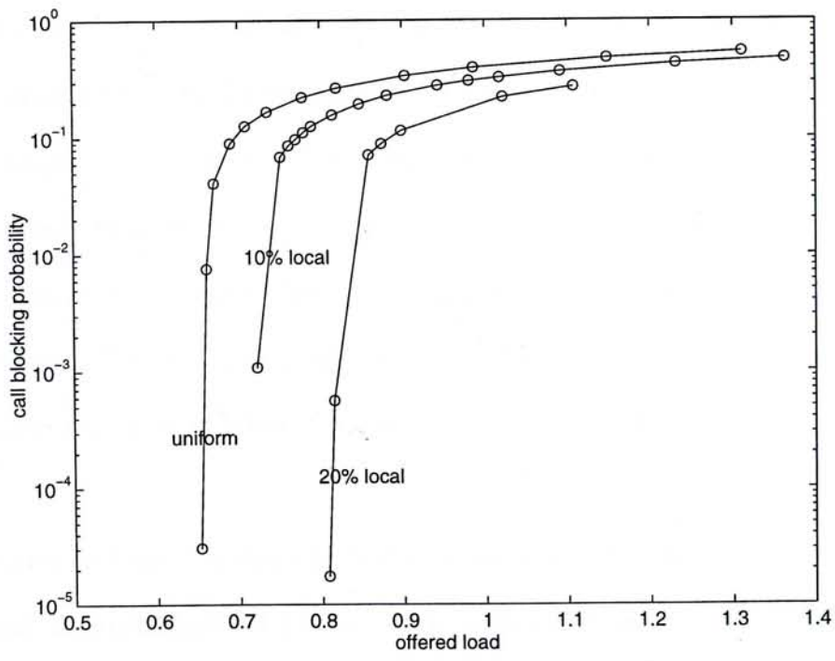

Figure 2.14: Effect of localized traffic ratio on call blocking probability

Chapter 3

Multichannel ATM Switching

3.1 Introduction

The previous chapter has described two scalable and stackable switch interconnection architectures, BMSN and SpiderNet. Both architectures require their modules to support channel grouping for inter-module communications. As mentioned in Chapter 1, these modules are examples of a family of switches called the *multichannel switch.*

Current designs of multichannel switches are usually based on a specific switching fabric. For example, the modules of our architectures assume the use of shared-memory switch fabrics whereas, [27, 5, 19] assume the space-division type.

We consider in this chapter the use of a generic point-to-point switch module to implement a multichannel switch. For a point-to-point switch, cells have to be routed to a specific output port. In a multichannel switch, cells can be routed to any one output port of the same channel group. To implement a multichannel

switch based on a point-to-point switch, we need to assign the channel group traffic to specific output link so that they can be routed through the point-to-point switch. Channel allocation schemes are designed for this purpose.

On the other hand, as cells are routed across a number of links in parallel, we also need to consider mechanisms to guarantee that cells are delivered in sequence at the output port of the switch.

3.2 Switch Design

The design of a multichannel switch based on a point-to-point switch is shown in Figure 3.1. Similar to the switch module described in the previous chapter, it has local I/O ports as well as input and output channel groups for switch extension. Besides, there are some additional components. They are the Generic Switch Fabric (GSF), Channel Allocator, Port Controllers, Channel Group Controllers as well as the Cell Resequencers. These additional components serve to provide the switching, channel allocation, and cell resequencing functions of the switch. Their operations are briefly described as follows.

In each time slot, cells enter the switch through its local input ports and input channel group. When they arrive, they are stored respectively in the corresponding Port Controllers and Channel Group Controllers to wait for the routing decisions. If they are targeted for the local output ports, they will be switched through the GSF to their destined output ports. The GSF can be any kind of point-to-point switch. It can be a shared-memory fabric, space-division fabric, or bus-type fabric, etc. There is no assumption on whether the cell buffers are placed at the input or at the output. If, in another case, they are targeted for

other remote switches instead (i.e., they have to pass through a specific channel group), the link within the group allocated to each of them is determined by the Channel Allocator. The channel allocation algorithm is either centralized or distributed in nature.

Finally, cells arrived at their destination ports can get out-of-sequenced because of the multiple paths provided by channel groups. Cell Resequencers are needed at the local output port to reorder the cells before they are delivered out of the switch.

3.3 Channel Allocation Algorithms

This section introduces two channel allocation algorithms. One is VC-based and another is channel group-based. They are respectively called the VC-Based String Round Robin (VCB-SRR) Algorithm and the Channel Group Based Round Robin (CGB-RR) Algorithm.

3.3.1 VC-Based String Round Robin (VCB-SRR) Algorithm

A channel allocation algorithm is described as VC-based if it allows each VC to have its own channel allocation process independent of other VCs. We consider the use of String Mode transmission [9] to distribute cells of a VC over the multiple links of the channel groups. In this mode, cell stream of a VC is chopped into groups of fixed number of cells, say N. Each group of cells is called a *string*, and N is its *string length*. The strings are then distributed in a round-robin fashion to the channels of a channel group.

The features of this transfer mode are twofolds:

1. Cells carried by the string mode are more likely to arrive at the destination in the correct order. It is because cells belonging to the same string are always allocated with the same link when they pass through a channel group. This will maintain their order within a string. Therefore, sequence problem only happens on the string level. This implies a lower out-of-sequence probability of cells at the ultimate destination. The computational burden of the resequencers can hence be reduced.

2. There is trade-off between the out-of-sequence probability and the load-sharing ability with the adjustment of the string length. In one extreme, if the string length goes to infinity, no cell will get out-of-sequenced because cells will always be allocated on the same channel. But, there will be no loading-sharing among channels either. In another extreme, if the string length is one, cells are more likely to get out-of-sequenced, but loading is better shared among the channels.

The VCB-SRR algorithm is distributed in nature that allows channel allocation mechanism to be implemented at every port controller of the switch. A centralized CA, which is usually more difficult to implement, is not needed. However, because of the same fact that there is not a centralized channel allocator to act as a coordinator, different VCs may allocate the same channel at the same time for their cells. As a result, the allocated channel can become congested whereas another may be underutilized for a certain period of time. The loading will not be shared uniformly among channels of the same group.

3.3.2 Implementation of the VCB-SRR Algorithm

At the local input port, the port controller performs two tasks: *stream chopping* and *channel allocation.* Stream chopping means cutting the cells of a VC into strings. This is done by attaching an additional header to the cells. The additional header contains two fields, the String Identifier (SID) and the End-Of-String (EOS) bit. The SID is an s-bit field which is used to identify which string the cell belongs. The EOS bit is set to indicate that the cell will terminate the string. We call this cell the *EOS cell.* To implement stream chopping, each port controller needs a state table. An example is shown below:

VCI	Current SID	Counter
a	5	3
b	0	0
c	10	8
⋮	⋮	⋮

Each record has two fields that keep track of the chopping status of a VC. The two fields are the Current SID and the Counter. The Current SID indicates the string that is currently being chopped, and the Counter stores the accumulated length of the string.

At every time slot, when a cell arrives at the port controller, its VCI is extracted from its ATM header and is used as an index to look from the table for its assigned SID. The assigned value is read off from the Current SID field of its VC's record, and written to the SID field of the cell. After that, the Counter which keeps the accumulated length of the string, will be incremented. If it reaches the targeted string length N, it will return to zero, and the EOS bit of

the cell is set. At the same time, the Current SID is also incremented to get ready for the chopping of a new string. If the Current SID exceeds it maximum value $2^s - 1$ after the increment, it will be wrapped around to zero automatically. Following this procedure, strings carrying their SID are transmitted one after another into the switch.

The second major task of the port controller is the carrying out of the VCB-SRR Algorithm to allocate channels for the incoming strings. To implement the algorithm, we add a Channel Pointer field to the above state table.

The table now becomes:

VCI	Current SID	Counter	Channel Pointer
a	5	3	$channel(0,1)$
b	0	0	$channel(0,3)$
c	10	8	$channel(2,1)$
⋮	⋮	⋮	⋮

Each VC record has its Channel Pointer to point to the channel to be allocated for a new cell of the VC. Its entry is in the form of $channel(y, x)$ which means channel number y of group x. If group x has R channels, then its channels are named $channel(0, x), channel(1, x), \cdots$, and $channel(R-1, x)$.

At the beginning, when a VC is established, its channel pointer is randomly set to any channel of the targeted channel group. In the duration of the VC, the channel number of the field will be updated according to the round-robin manner whenever a new string is chopped.

For the channel group controller at the input channel group, it has to carry out the same job for each of its incoming cells within a time slot.

Cell Resequencers

To guarantee sequential delivery of cells, a cell resequencer is needed at the local output port to reorder cells before they are delivered. This device provides resources such as resequencing buffers and resequencing mechanisms to store and reorder the cells.

With the use of strings to transfer cells, no resequencing is needed for the cells within a string because they are transferred along the same path. Out-of-sequence problem, therefore, will only happen between different strings. For example, we have two strings. String h (SID=h) originally precedes string i (SID=i). However, due to different cell delay variation (CDV) experienced by the cells, some cells of string i arrive earlier at the resequencer before the complete arrival of string h. These cells are needed to be held up in the resequencer until all cells of string h have arrived.

The arrival of the last cell (EOS = 1) of string h will signal the resequencer the completion of string h. If there is no cell loss, the EOS cell of string h will ultimately arrive. However, cell loss will occasionally occur. If the lost cell is not the EOS one, the resequencer will still be able to detect the termination of string h. However, if the EOS cell is lost, string i will be waiting forever for this lost cell. To solve this problem, we introduce a time-out period D(in units of cell-time) to every string that is waiting for delivery. If a string has already waited for D cell-times, we assume some cells of previous strings have been lost. The waiting string will immediately stop waiting, and be allowed to leave the resequencer.

The above strategy has only solved one part of the resequencing problem. Another part is the periodicity of the SID. With this identifier wrapping around

periodically, it is possible to have more than one strings with the same SID simultaneously transferring inside the multichannel switches. Consider the previous example. If string h is lost, string i will wait at the resequencer. The next string h generated due to the periodicity of the SID may arrive within the time-out period of the waiting string. If this happens, the next string will be wrongly appended to the previous string.

We introduce the concept of *valid range* of SID for the resequencer, and define that only SIDs that fall within the valid range can be accepted by the resequencer. In order to continuously accept new strings along the duration of the VC, this range will slide along the SID axis. So, when a string with SID=x is successfully resequenced and transmitted out of the resequencer, the valid range will begin from SID=$(x+1)$ mod P_{SID} to SID=$(x+1+W-1)$ mod P_{SID} where P_{SID} and W are the SID period and the valid range respectively.

Consider again the previous example. Now, the waiting string i only needs to wait for up to $W-1$ previous strings to arrive at the resequencer. That is, those strings beginning from SID=$i-(W-1)$ onwards will be accepted by the resequencer within the time-out period.

To find the minimum value of the SID period P_{SID} required, we consider Figure 3.2. The figure illustrates the worst case situation in which (1) cells are sent out back-to-back in the peak cell rate PCR which results in a shortest wrap-around period of $\frac{P_{SID} \times N}{PCR}$ where N is the string length, (2) the leading cell of string i experiences the longest transfer delay CTD_{max} before arriving at the resequencer, and (3) when it arrives, it finds that the resequencer is still waiting for string $(i-(W-1))$. Under this situation, the next string $(i-(W-1))$ will most likely arrive before the time-out of the waiting string i, and be wrongly

inserted at the front of i. This would happen when the switch is very congested or there is a link failure. In this case, string i have to wait for D cell-times. To force the leading cell of the next string $(i-(W-1))$ to arrive after the time-out, the following inequality should hold:

$$\begin{aligned} T_0 + \frac{[P_{SID} - (W-1)]N}{PCR} + CTD_{min} &> T_0 + CTD_{max} + D \\ \frac{[P_{SID} - (W-1)]N}{PCR} &> CTD_{max} - CTD_{min} + D \\ &> CDV + D \end{aligned}$$

$$P_{SID} > \frac{PCR \times (CDV + D)}{N} + W - 1 \tag{3.1}$$

For example, if $CDV = D = 100$ cell-times, $N = 5$, $W = 50$, and $PCR = 1$ cell/cell-time,

$$\begin{aligned} P_{SID} &> \frac{1(100+100)}{5} + 50 - 1 \\ 2^s &> 89 \\ s &> \log_2 89 \\ s_{min} &= 7 \end{aligned}$$

We need 7 bits to carry the SID information.

Figure 3.3 illustrates the block diagram of a cell resequencer at the local output port. It has three major components. They are the Resequencing Controller, the Resequencing Buffers, and the Scheduler. The first component defines the

resequencing mechanism. The second component stores out-of-sequenced cells and carries out buffer management, and the third one determines the transmission order of the resequenced cells from different VCs to the local output port.

At every time slot, the GSF sends at most one cell to the cell resequencer from its output port. When a cell arrives, the Resequencing Controller first reads off its VCI and SID from the cell header, and determines if the SID falls within the valid range of the VC. The table beneath the controller helps to find this range. It keeps track of two values, the Expected String Identifier (ESID) as well as the parameter W for each VC. The ESID field records the SID expected to be delivered from the VC. It is also the starting SID (SID_s) of the valid range. With W given from the table, the ending SID (SID_e) can be obtained. It is equal to $(SID_s + W - 1) \bmod P_{SID}$.

The following gives the necessary and sufficient condition for the SID to fall into the valid range:

Case 1: $SID_s < SID_e$

String's SID is in the valid range iff $SID_s \leq SID \leq SID_e$.

Case 2: $SID_s > SID_e$

String's SID is in the valid range iff $SID \geq SID_s$ and $SID \leq SID_e$.

Case 3: $SID_s = SID_e$

This case happens when $W = 0$ or $W = P_{SID}$. The former condition will not allow any cell to enter the resequencing buffer, whereas the latter will cause string misinsertion in case of cell loss. Therefore, it is reasonable to assume that this case does not exist.

Satisfying the above condition still does not mean that the cell of the string can be stored in the resequencing buffer. The cell might be the early arrival of the next string with the same valid SID. Previous paragraphs have discussed how to use the time-out period D for a string stored in the resequencing buffer to recognize this situation. After waiting for D cell-times, the string will time-out and inform the cell resequencer to reject these strings through the change in the Resequencing Status Registers.

If the cell passes the above two checks, the resequencer will send it to the ordered position of the resequencing buffer for storage. At the same time, its VCI will be appended to the Scheduling Queue for transmission out of the buffer. The resequencer scans the Scheduling Queue once in every time slot. It checks the entries one-by-one starting from the head until it finds a VC which has a resequenced cell ready for transmission. Then, it issues a command to the MUX to deliver that cell.

The resequencing buffer of each VC is organized as a cyclic list of W string buffers. It is associated with a String Pointer that points to its heading memory location. This location is the place that stores the string with SID=$ESID$. When that string is transmitted out, the String Pointer will advance to the next location. In most cases, the cell with SID=$ESID$ will gradually arrive and enter the heading location. Thus, the pointer can usually advance one location at a time in a smooth way. However, when cell loss happens, and the waiting string in the buffer times-out, the pointer will skip the empty locations at the head, and retrieve those waiting strings. It recognises this status change by looking at the Resequencing Status Registers. Under both situations, it will gradually advance to the physical end of the list. In this case, it returns to the

opposite end, and the process repeats again. Resequenced strings are therefore continuously delivered out of the resequencing buffer.

3.3.3 Channel Group Based Round Robin (CGB-RR) Algorithm

The Channel Group Based Round Robin (CGB-RR) Algorithm is a centralized channel allocation algorithm. Cells targeted for any particular channel group of the switch would consult a centralized channel allocator for their own channels. The allocator will carry out the round-robin mechanism at each output group to distribute the cells over its channels.

It can be proved from the following theorem that the CGB-RR Algorithm is capable of sharing the input load fairly among the channels of a channel group.

Theorem 3.1 *For a channel group y with R channels, the CGB-RR Algorithm guarantees that, at every time slot, the difference in the accumulated number of cells destined for each channel is at most 1.*

Proof: By the channel allocation algorithm, we have the first cell being assigned to $channel(0, y)$, the second cell to $channel(1, y)$, the third cell to $channel(2, y)$, and so on in a round-robin manner. Assume that the number of cells going to channel group y at time slot n is k_n. Then, at time slot n, the accumulated number of cells going to channel x $(0 \leq x \leq R-1)$ of group y is

$$\lfloor \frac{\sum_{i=0}^{n} k_i}{R} \rfloor + \Delta_x(n)$$

where $\Delta_x(n) = \begin{cases} 1, & x < \sum_{i=0}^{n} k_i - R\lfloor \frac{\sum_{i=0}^{n} k_i}{R} \rfloor; \\ 0, & \text{otherwise.} \end{cases}$

□

Thus, we see that the accumulated numbers of cells of different channels always differ by at most 1. Loading is evenly distributed to all the channels at all the time.

3.3.4 Implementation of the CGB-RR Algorithm

As mentioned earlier, the CGB-RR Algorithm is centralized in nature. The channel allocator dictates the distribution of cells to all the output channel groups.

At every time slot, the port controllers and the channel group controllers hold up incoming cells from their respective local input ports and channel groups for routing purpose just like the case in the VCB-SRR Algorithm. They determine whether a cell is targeted for a local output port or an output channel group. If it is the latter case, they will find out the ID of the particular channel group, pack up a *channel request message* embedding the ID and send it to the channel allocator for channel allocation.

How fast the channel allocator processes these channel request messages depends on how many messages will come up in a time slot. For an $N \times N$ multichannel switch, it will receive at most N messages in a time slot. With the 424 bits that an ATM cell contains, the device must be fast enough to process

a request within $\lfloor \frac{424}{N} \rfloor$ bit-times if we assume sequential processing. For example, when $N = 64$, the device should complete processing each request within 6 bit-times.

Here is the procedure in processing a request message: At the beginning, the channel allocator obtains the output channel group ID from the request message, and uses it as an index to find out the allocated channel from a look-up table which maintains a Current Channel Number entry for each channel group. The table is shown below:

ID of Output Channel Group	Current Channel Number
0	2
1	4
2	0
3	5
⋮	⋮

For example, the channel allocator will allocate channel 2 to a cell that goes to group 0. After reading off the allocated channel from the table, the channel allocator returns the result back to the corresponding port or channel group controller. Finally, it updates the Current Channel Number according to the round-robin mechanism. This is done by applying a $modulo - R$ increment operation to the field. The job cycle then finishes, and a new one starts up for the next request message.

Cell Resequencers

The cell resequencing mechanism considered here is a special case of the mechanism described Section 3.3.2. It is the case when the string length $N = 1$.

Strings are no longer used in the transfer of cells. The cell resequencer treats each cell independently in the resequencing process. Therefore, each cell carries its own sequence number SN. We can still use Inequality 3.1 to find out the minimum SN period, but here, N is set to 1.

3.4 Performance and Discussion

In this section, we put the constructed multichannel switches running the VCB-SRR and CGB-RR Algorithms into one of our switch interconnection architectures, SpiderNet. We observe how the two algorithms will affect the cell loss probability inside the GSF, and study the resequencing delay and cell loss probability of the resequencers.

Using a similar simulation model as that in Chapter 2, we construct a 1024×1024 large scale ATM switch by the interconnections of 64 multichannel switch modules arranged in a 8×8 torus structure. Each module has 64 pairs of ports. 16 pairs of them are the local input/output ports. The rest 48 pairs are organized to form 8 pairs of extension channel groups. It means that we have a group size of 6. The GSF is an output-buffered space-division fabric which has two types of buffers, the channel buffer and the output buffer. The former is the buffer placed at each output channel, whereas the latter is the buffer for the local output port. Their sizes are independent of each other so that they can be tuned separately. The output buffer size is set at 80. We use a Bernoulli cell generation process for incoming cell stream. The mean bit rate is 5Mbps for each stream. The port bandwidth is 150Mbps. They are the same as those used in the previous chapter.

Figure 3.4 shows the cell loss probability at the channel buffers for the VCB-SRR and the CGB-RR Algorithm. We also show the result of using the shared-memory switch modules as in Chapter 2. The cell loss performance of the space-division switch is compared with that of the shared-memory switch. We see that the CGB-RR Algorithm gives a smaller cell loss probability than VCB-SRR. Also, its cell loss performance is the same as that of the shared-memory switch modules. It is because, in the CGB-RR Algorithm, the centralized channel allocator evenly distributes cells to all channels of the same group. All the channel buffers are treated as a single buffer that is similar to the shared FIFO for each channel group in the shared-memory switch. For the VCB-SRR, as expected, the switch gives a higher cell loss probability. Also, increasing the string length will cause more cell loss. It is because the loading among channel buffers of the same group is not evenly distributed. Increasing the string length will deteriorate the load-sharing effect of channels of the same group. That is why a longer string length will make the cell loss performance worse.

Figure 3.5 and 3.6 reveals the performance difference between CGB-RR and VCB-SRR in another perspective. They show the cell transfer delay across the switch interconnection architecture under the algorithms. Cell transfer delay is defined as the time elapsed between the entering of a cell into the switch and the leaving of the cell from the GSF in the destination module. Therefore, resequencing delay is not considered here. From the figures, we see that CGB-RR results in a smaller cell delay variation (CDV) for cells tranferring through different hop counts when compared with VCB-SRR. This is due to better load-sharing effect under CGB-RR that makes the queue lengths of the channel buffers very close with each other.

The following results concentrate on the resequencing delay and cell loss performance at the resequencers. Two performance parameters are measured. They are the mean resequencing delay and the resequencing cell loss probability. The resequencing delay is defined as the time duration that a cell stays in its resequencing buffer. Multiplexing delay experienced by different VCs inside the resequencer is thus included. On the other hand, a cell is dropped when one or both of the following two events happen: (1) Cell's SN/SID does not fall into the valid range of the sequence identifier. This may happen when the cell arrives too early at the resequencer or the valid range is not able to slide smoothly due to events such as cell loss; (2) Even though it has a valid sequence identifier, it will still be dropped if the resequencer asserts that its SN/SID belongs to the next period. This will happen after the time-out of a waiting cell in the resequencing buffer.

Figure 3.7 and 3.8 illustrate the effects of string length (only for the VCB-SRR Algorithm) on the two parameters. Channel buffer size is set to a small value of 30 so that cell loss will happen inside the GSF, and in turn affect the resequencing mechanism. In Figure 3.7, we see that when the string length is very small, mean resequencing delay will rise up to a peak. Then it will start to fall after a string length of 5. The reason is that when the string length increases, cell delay variation across the interconnection architecture will increase also. With this higher jitter of cell stream in case of longer string, the multiplexing delay component in the resequencing delay will become larger, hence resulting in a rising resequencing delay. At the another extreme (i.e., string length is very long), although the cell delay variation is high, but in most of the time cells belong to the same string. So, most cells arrive at the resequencer in the

correct sequence. The reordering time component in the resequencing delay is very small. With the competition of the two effects, the mean resequencing delay will rise to a peak and then fall as the string length increases.

From Figure 3.8, we see that the resequencing loss probability also behaves non-monotonically when the string length increases. When string length is small, cell delay variation is relatively small. The major cell loss event, i.e. the out-of-valid-range event, thus happens rarely. This contributes to the low cell loss probability. On the other extreme, when the string is very long, say approaching infinity, cells are also in sequence and the valid range does not need to slide along the axis of sequence identifier. The two resequencing loss events decribed previously will not occur. With these two competitive factors, the resequencing loss probability also rises to a peak and then falls as the string length increases.

Here, we consider how to determine a suitable value for the time-out parameter D and the valid range W. These two design parameters are very important to the resequencing cell loss performance. If they are inproperly set, large amount of cells will be dropped at the resequencer. Figure 3.9 illustrates the effects of varying D and W on the resequencing loss probability under the CGB-RR Algorithm. Channel buffer size is set to 7 so that a buffer overflow probability of 1.18073×10^{-6} is resulted across the switch. Generally, larger W will give lower resequencing loss probability because more cells with different sequence identifiers can simultaneously be accepted by the resequencer. For the parameter D, it is not desirable to use a very small value or a very large one. In both cases, large resequencing loss probability will be obtained because it is either very easy for a waiting cell to time-out or wait too long that will interrupt the sliding mechanism of the valid range, and causes cell loss. From the figure, we

see that the optimal range of D in case $W = 4$ is from 10 to 15. Similarly, for different W, the optimal D can also be obtained in this way.

3.5 Concluding Remarks

This chapter investigates the design of a multichannel switch that allows the use of generic point-to-point switch fabrics for its switching component. Two channel allocation algorithms are studied, one centralized and one distributed. The centralized algorithm called the CGB-RR Algorithm outperforms the distributed VCB-SRR Algorithm in terms of load-sharing capability inside the channel group. The weakness of the latter algorithm is that it cannot avoid contention of cells for the same allocated channel because the VCs do not coordinate with each other in the channel allocation process. At the same time, larger string length in the String Mode will make the situation even worse. Although using a string length larger than 1 can reduce the out-of-sequence probability, the larger cell delay variation resulted will cause severe channel buffer overflow, as well as higher resequencing delay and resequencing loss probability. Therefore, string length of 1 is the best in terms of cell loss and delay performance.

For the selection of D and W for good resequencing cell loss performance, a larger W will give a lower resequencing loss probability. A suitable D can then be obtained by simulation. One point that requires special emphasis is that we cannot increase W to arbitrarily large for good cell loss performance. It is because the period of the sequence identifier should be increased accordingly as required by Inequality 3.1.

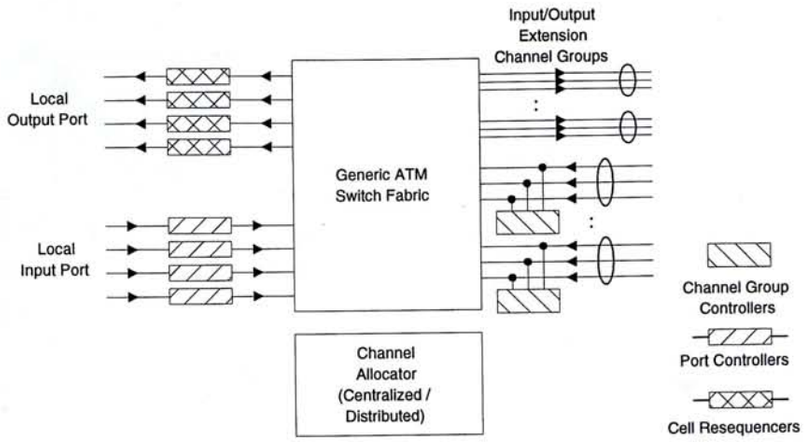

Figure 3.1: Design of a Multichannel ATM Switch

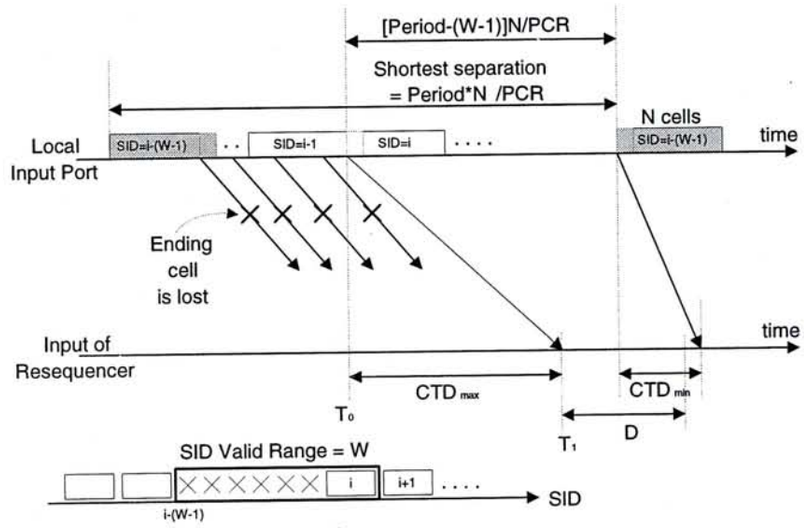

Figure 3.2: Worst Case Situation

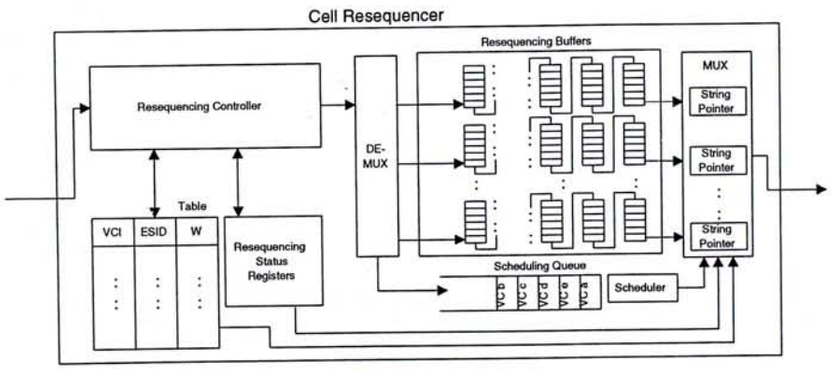

Figure 3.3: Block Diagram of Cell Resequencer

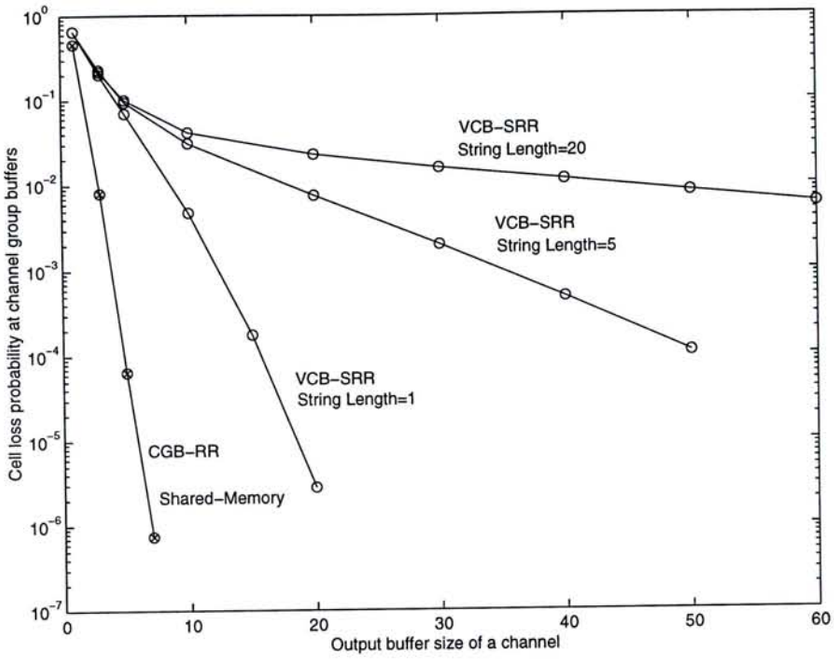

Figure 3.4: Cell loss probability at channel buffers for different channel allocation algorithms

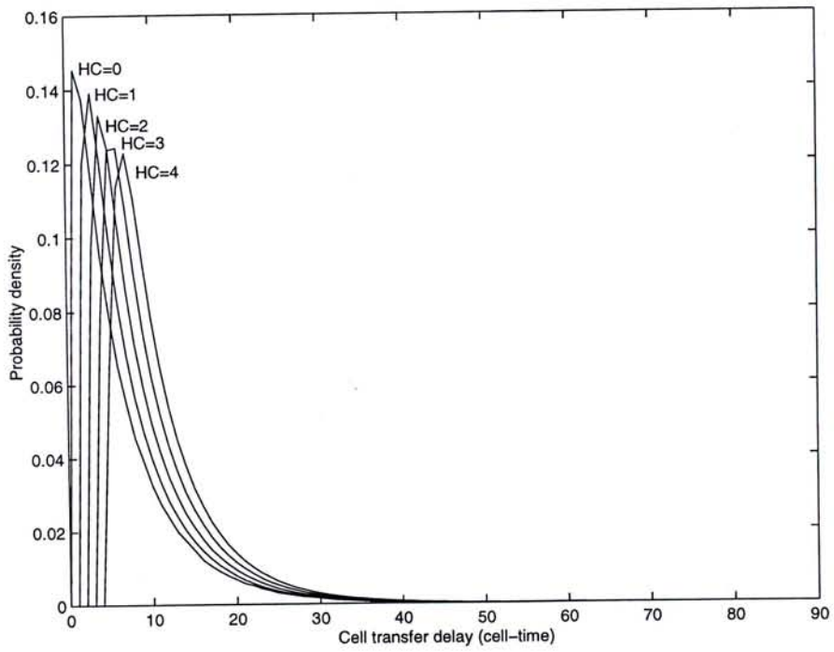

Figure 3.5: Cell transfer delay profile under the CGB-RR Algorithm

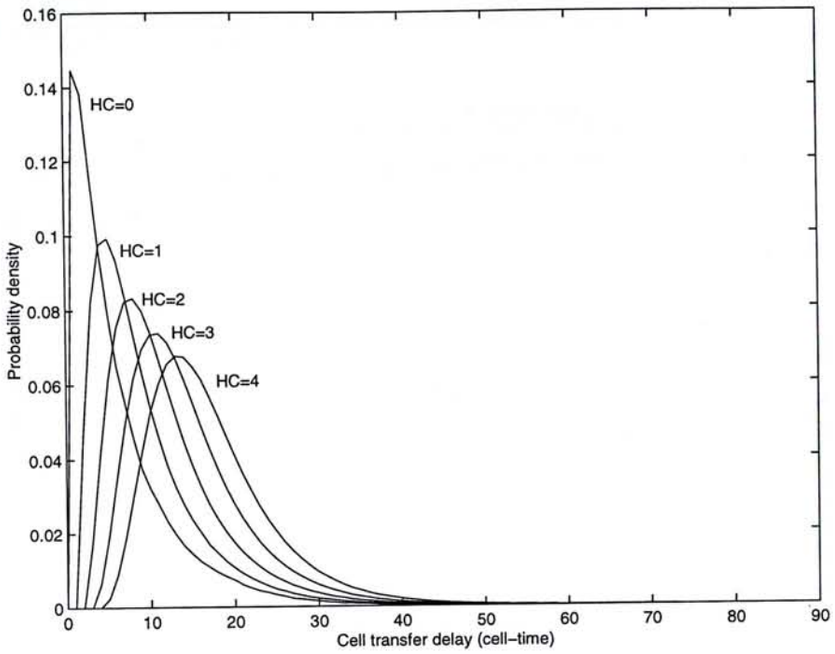

Figure 3.6: Cell transfer delay profile under the VCB-SRR Algorithm(String Length = 1)

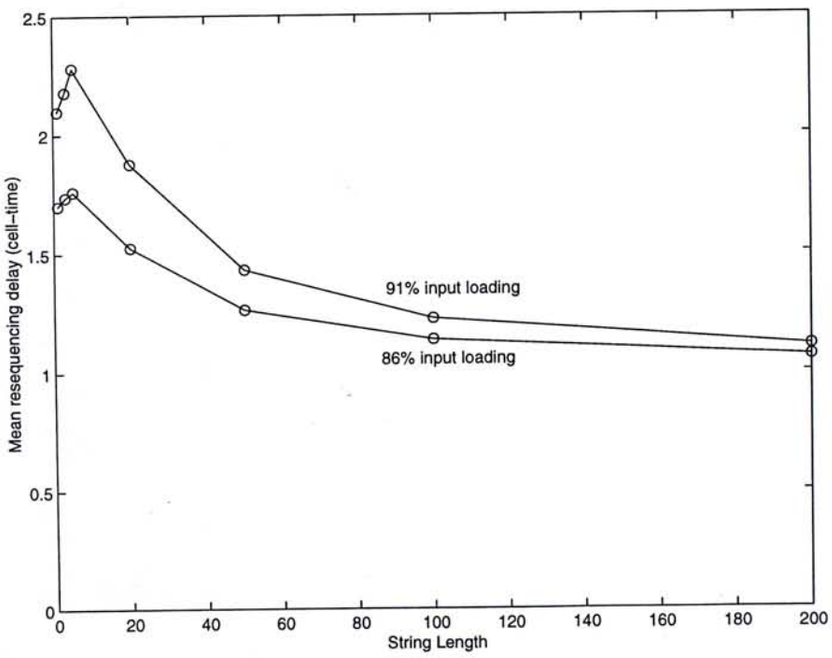

Figure 3.7: Mean resequencing delay vs different string lengths

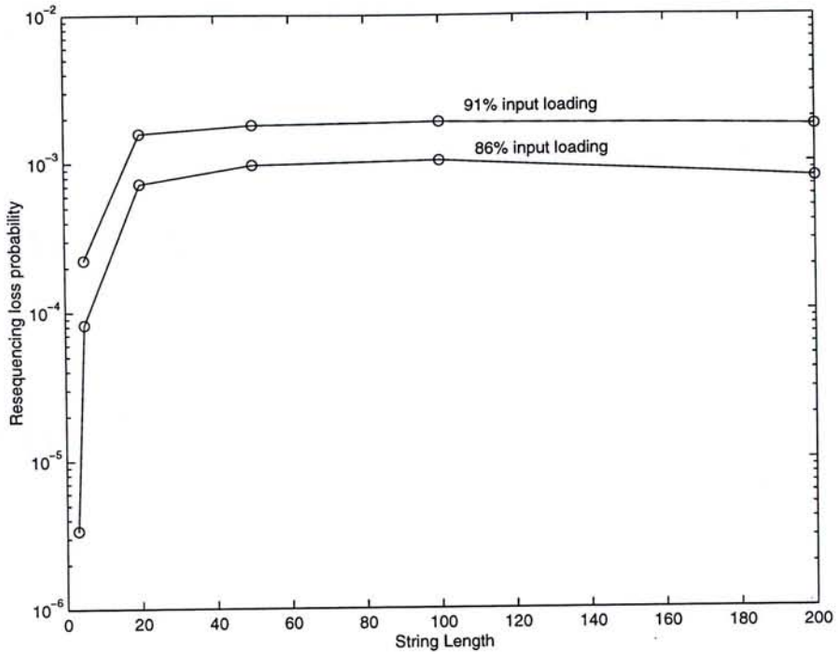

Figure 3.8: Resequencing loss probability vs different string lengths

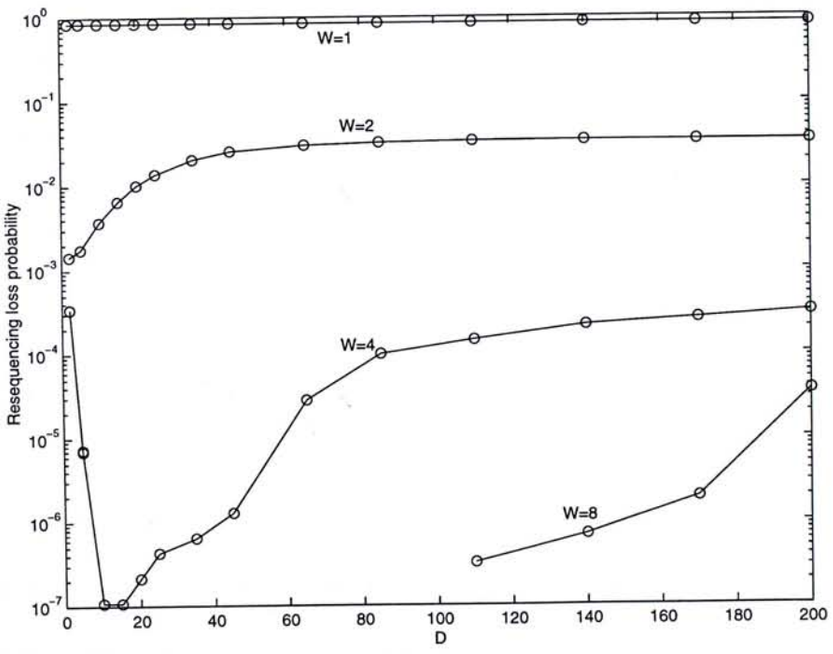

Figure 3.9: Effects of D and W on resequencing loss probability

Chapter 4

Conclusion

In this thesis, we have considered two large scale ATM switch interconnection architectures named BMSN and SpiderNet. Both have a grid structure based on torus topology. The nice features of these two architectures include: (1) scalability, modules can be added to increase the switch size and capacity; (2) stackability, no rewiring of existing ports is needed when modules are added; (3) fault-tolerance, multiple alternate paths are available for re-routing; and (4) reasonably good call blocking performance. With more direct interconnections between switch modules in SpiderNet, ATM connections require less interconnection bandwidth. Hence, the switch has a higher switching capacity, lower cell loss probability, and smaller end-to-end delay than its counterpart, BMSN, in general.

Another issue we have investigated is on the design of multichannel switch. We have proposed a multichannel switch design strategy that allows generic point-to-point fabric to be used, and guarantees sequential delivery of cells

through channel groups. Since switch modules in our two interconnection architectures are also multichannel switches, the design strategy will make implementation of the switch modules more flexible. That is, different modules inside the same switch architecture can choose their own point-to-point fabric for implementation.

Bibliography

[1] H. Ahmadi and W. E. Denzel. "A Survey of Modern High-Performance Switching Techniques". *IEEE JSAC*, 7(7):1091–1103, Sept. 1989.

[2] J. Anderson, B. T. Doshi, S. Dravida, and P. Harshavardhana. "Fast Restoration of ATM Networks". *IEEE JSAC*, 12(1):128–138, Jan. 1994.

[3] T. Aramaki, H. Suzuki, S. Hayano, and T. Takeuchi. "Parallel 'ATOM' Switch Architecture for High-Speed ATM Networks". In *IEEE ICC '92*, pages 250–254, 1992.

[4] F. Borgonovo and E. Cadorin. "Routing in the bidirectional Manhattan network". In *Third International Conference on Data Communication Systems and their Performance*, June 1987.

[5] T. H. Cheng. "A Multichannel ATM Switch with Output Buffering". In *Proceedings of IEEE Singapore International Conference on Networks/International Conference on Information Engineering '93*, pages 364–368, 1993.

[6] A. Choudhury and E. L. Hanhe. "Buffer Management in a Hierarchical Shared Memory Switch". In *Proceedings of IEEE Infocom '94*, pages 1410–1419, June 1994.

[7] J. P. Coudreuse, W. D. Sincoskie, and J. S. Turner. Guest Editorials. *IEEE JSAC*, 6:1452–1454, Dec. 1988.

[8] Jacques H. Déjean, Lars Dittmann, and Claus N. Lorenzen. "Multichannel Bandwidth Allocation in a Broadband Packet Switch". *IEEE JSAC*, 9(9):1452–1460, Dec. 1991.

[9] J.H. Déjean, L. Dittmann, and C. N. Lorenzen. "String Mode — A New Concept for Performance Improvement of ATM Networks". *IEEE JSAC*, 9(9):1452–1460, Dec. 1991.

[10] Kai Y. Eng, Mark J. Karol, and Y. S. Yeh. "A Growable Packet (ATM) Switch Architecture: Design Principles and Applications". In *Proceedings of IEEE GLOBECOM '89*, pages 1159–1165, Nov. 1989.

[11] Kai Y. Eng *et al.* "A High-Performance Prototype 2.5Gb/s ATM Switch For Broadband Applications". In *Proceedings of IEEE GLOBECOM '92*, pages 111–117, Dec. 1992.

[12] Joan Garcia-Haro and Andrzej Jajszczyk. "ATM Shared-Memory Switching Architectures". *IEEE Network Magazine*, pages 18–26, July/August 1994.

[13] R. Guérin, H. Ahmadi, and M. Naghshineh. "Equivalent Capacity and Its Application to Bandwidth Allocation in High-Speed Networks". *IEEE JSAC*, 9(7):968–981, Sept. 1991.

[14] Sanjay Gupta, Keith W. Ross, and Magda El Zarki. "Routing in Virtual Path Based ATM Networks". In *Proceedings of IEEE GLOBECOM '92*, pages 571–575, 1992.

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

[16] Ren-Hung Hwang. "LLR Routing in Homogeneous VP-based ATM Networks". In *Proceedings of IEEE Infocom '95*, pages 587–593, 1995.

[17] Mark J. Karol, Michael G. Hluchyj, and Samuel P. Morgan. "Input vs. Output Queueing on a Space-Division Packet Switch". In *Proceedings of IEEE GLOBECOM '86*, pages 659–665, Nov. 1986.

[18] R. Kawamura, K. Sato, and I. Tokizawa. "Self-Healing ATM Networks Based on Virtual Path Concept". *IEEE JSAC*, 12(1):120–127, Jan. 1994.

[19] Hyong S. Kim. "Multichannel ATM Switch with Preserved Packet Sequence". In *Proceedings of IEEE ICC '92*, pages 1634–1638, 1992.

[20] T. H. Lee and S. J. Liu. "Performance Analysis of a Large Scale ATM Switch with Input and Output Buffers". In *Proceedings of IEEE Infocom '94*, pages 1465–1471, June 1994.

[21] T. T. Lee. "A Modular Architecture for Very Large Packet Switches". *IEEE Transactions on Communications*, 38(7):1097–1106, July 1990.

[22] W. T. Lee and L. Y. Kung. "Binary Addressing and Routing Schemes in the Manhattan Street Network". *IEEE/ACM Transactions on Networking*, 1(3):26–30, Feb. 1995.

[23] S. C. Liew and K. Lu. "A 3-stage Interconnection Structure for Very Large Packet Switches". In *Proceedings of IEEE ICC '90*, pages 316.7.1–316.7.7, 1990.

[24] N. F. Maxemchuk. "The Manhattan Street Network". In *Proceedings of IEEE GLOBECOM '85*, pages 255–261, Dec. 1985.

[25] N. F. Maxemchuk. "Routing in the Manhattan Street Network". *IEEE Transaction on Communications*, COM-35(5):503–512, May 1987.

[26] K. Ohtsuki, K. Takemura, J. F. Kurose, H. Okada, and Y. Tezuka. "A High-Speed Packet Switch Architecture with a Multichannel Bandwidth Allocation". In *IEEE Infocom '91*, pages 155–162, 1991.

[27] A. Pattavina. "Multichannel Bandwidth Allocation in a Broadband Packet Switch". *IEEE JSAC*, 6(9):1489–1499, Dec. 1988.

[28] A. Pattavina. "A Multiservice High-Performance Packet Switch for Broad-Band Networks". *IEEE Transactions on Communications*, 38(9):1607–1615, Sept. 1990.

[29] N. T. Plotkin and P. P. Varaiya. "Performance Analysis of Parallel ATM Connections for Gigabit Speed Applications". In *Proceedings of IEEE Infocom '93*, pages 1186–1193, 1993.

[30] H. Saidi, P. S. Min, and M. V. Hegde. "Guaranteed Cell Sequence in Nonblocking Multi-Channel Switching". In *IEEE Infocom '94*, pages 1420–1426, 1994.

[31] K. Sezaki and Y. Yasuda. "A General Architecture of ATM Switching Networks which are Non-blocking at Call Level". In *Proceedings of IEEE TENCON '92*, pages 603–607, Nov. 1992.

[32] P. C. Wong and E. H. Tung. "A Large Scale Packet Switch Interconnection Architecture using Overflow Switches". In *Proceedings of IEEE ICC '93*, pages 708–714, 1993.

[33] P. C. Wong and M. S. Yeung. "Design and Analysis of a Novel Fast Packet Switch—Pipeline Banyan". *IEEE/ACM Transactions on Networking*, 3(1):63–69, Feb. 1995.

[34] Y. S. Yeh, M. G. Hluchyj, and A. S. Acampora. "The Knockout Switch: A Simple Architecture for High-Performance Packet Switching". *IEEE JSAC*, 5(8):1274–1283, Oct. 1987.

[35] Y. Yoshida. "Resequencing Delay Characteristics of Multilink System in Packet Switched Networks". *Electronics Letters*, 26(23):1971–1972, Nov. 1990.