Switching Techniques for Broadband ISDN

a thesis presented for the award of

Doctor of Philosophy

by

Martin Collier, BEng, MEng,
School of Electronic Engineering,
Dublin City University.

Supervisor: Dr. Tommy Curran.

July, 1993.

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of PhD in Electronic Engineering is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: [illegible] Date: 6/7/93
Martin Collier

Acknowledgements

I hereby acknowledge the invaluable advice and assistance of my supervisor, Dr. Tommy Curran, during the course of this research. I also thank Prof. Charles McCorkell for his continual support.

Mention is also due to the many people in the School of Electronic Engineering who provided help along the way, and in particular, to Karen, who typed much of Chapters Four and Five, Dave, Peter and John, who sorted out many difficulties with equipment, and to Aengus, who endured three years of sharing an office with a member of the telecommunications research group.

Lastly, I would like to thank my family, for their constant encouragement.

Table of Contents

1. SWITCHING IN BROADBAND ISDN. 1

1.1 Introduction. 1
1.2 Objectives. 2
1.3 Summary. 3

References 4

2. BROADBAND SWITCH ARCHITECTURES. 6

2.1 Introduction. 6
2.2 Delta networks. 6
2.3 Self-routing networks based on 2x2 switches. 7
2.3.1 Principles of self-routing networks. 7
2.3.2 A notation to describe self-routing networks. 7
2.3.3 Some link permutations. 11
2.3.3.1 The perfect shuffle. 11
2.3.3.2 The inverse perfect shuffle. 12
2.3.3.3 The swap permutation. 12
2.3.3.3 The bit-reversal permutation. 14
2.3.4 Network definitions. 14
2.3.4.1 Baseline network. 14
2.3.4.2 Omega network. 14
2.3.4.3 Indirect binary n-cube. 16
2.3.4.4 The GL-banyan. 16
2.3.4.5 The P-banyan. 18
2.3.4.6 The S-banyan. 18
2.3.5 Inverse networks. 19
2.3.6 Topological equivalence. 21
2.4 Sorting networks. 22
2.4.1 Principles of sorting networks. 22
2.4.2 The Batcher sorter. 22
2.5 Switch architectures based on the banyan network. 24
2.5.1 Banyan switches. 24
2.5.2 Switches with multiple paths. 26
2.5.3 Switches with multiple banyans. 31
2.5.4 Concluding remarks concerning banyan-based switches. 34
2.6 Switch architectures based on sort-banyan networks. 34
2.6.1 The Starlite switch. 34
2.6.2 Switching with sorters. 36
2.6.3 Input-buffered switches. 38
2.6.4 Multiple servers and channel grouping. 41
2.6.5 Scaling difficulties in sort-banyan switches. 45
2.6.6 Conclusions regarding sort-banyan switches. 46
2.7 Memory-based and shared-medium switch architectures. 47

2.8 Other output-buffered switch architectures....51
2.9 Remarks concerning output-buffered switches. 55
2.10 Multi-stage switches. 55

References....61

3. PERFORMANCE EVALUATION OF ATM SWITCHES. 65

3.1 Introduction....65
3.2 Performance of internally blocking switches. 65
3.3 Classification of internally non-blocking switches. 67
3.4 Output queuing switches....68
3.4.1 Homogeneous arrivals. 68
3.4.2 Correlated arrivals - Markov-modulated sources. 70
3.4.3 Non-Markovian models. 76
3.4.4 Mixing of traffic types. 76
3.4.5 Shared-buffer switches....77
3.5 Throughput analysis of input-queuing switches. 78
3.5.1 Homogeneous traffic....78
3.5.2 Geometric burst lengths. 80
3.5.3 Non-uniform loads....83
3.6 Queuing analysis of input-queuing switches....84
3.6.1 Homogeneous traffic....84
3.6.2 Non-uniform traffic. 87
3.7 Fairness in Batcher-banyan switches....88
3.7.1 Implementing fair access in Batcher-banyan switches. 88
3.7.2 Homogeneous traffic....89
3.7.3 Bursty Traffic. 93

References....104

4. EQUIVALENCE AND BLOCKING IN ATM SWITCHES. 107

4.1 Equivalence in binary self-routing networks. 107
4.1.1 Establishing topological equivalence....107
4.1.2 The equivalence between the S-banyan and the omega network. 113
4.1.3 The equivalence between the S-banyan and the baseline network. 115
4.1.4 The equivalence between the S-banyan and the inverse S-banyan. 117
4.1.5 Exact equivalence of the baseline and inverse baseline networks. 120
4.1.6 The general case. 121
4.2 Local addresses in binary self-routing networks. 123
4.2.1 Definition of local address. 123
4.2.2 Dependence on source address. 123
4.2.3 Dependence of local address on destination address. 126
4.2.4 Local address as a function of both source and destination addresses. 128
4.2.5 The routing strategy in self-routing networks. 129
4.2.6 Input equivalence and the $g(k)$ function. 130
4.3. Blocking properties of binary self-routing networks. 132

4.3.1 Bernabei's condition. 132
4.3.2 Collisions as a function of source address. 133
4.3.3 Collisions as a function of the destination address....133
4.3.4 Calculation of the distance function. 135
4.3.5 Blocking and equivalence. 139
4.4 Non-blocking conditions for the S-banyan and its inverse. 141
4.4.1 The S-banyan. 141
4.4.2 The inverse S-banyan. 144
4.5 Synthesis of binary self-routing networks....145
4.5.1 An example of network synthesis....145
4.5.2 Constructing a 2^{2n}-input S-banyan from 2^{n+1} 2^{n}-input S-banyans. 147
4.6 The re-routing banyan. 151
4.6.1 Definitions....151
4.6.2 Routing in the rerouting banyan. 153
4.6.3 Equivalence in the B_r network....156
4.7. Blocking properties of three-stage networks....157
4.7.1. Existing conditions. 157
4.7.2 A new non-blocking condition for circuit-switched networks. 157
4.7.3 Special cases of the non-blocking condition. 160
4.7.4 Asymmetric switches. 163
4.7.5 Non-blocking conditions for ATM networks with call-level path allocation. 163
4.7.6 The non-blocking condition for a three-stage ATM switch with cell-level path allocation....166

References....167

5. A NEW THREE-STAGE BROADBAND SWITCH. 168

5.1 Introduction....168
5.2 An algorithm for path allocation at cell level. 169
5.2.1 The objectives of a path allocation algorithm. 169
5.2.2 Existing algorithms for cell-level path allocation. 170
5.2.3 A new algorithm for path allocation. 171
5.2.4 Synchronisation of the algorithm. 173
5.2.5 Modular growth of the switch. 178
5.3 Implementation of the processing element. 178
5.3.1 Design requirements....178
5.3.2 A bit-serial implementation. 179
5.3.3 A faster bit-serial implementation....183
5.3.4 Bit-parallel implementations....189
5.3.5 Processor implementation in CMOS. 190
5.3.6 Processor speed requirements for an input-queued switch. 191
5.4 Additional hardware....191
5.4.1 Hardware functions....191
5.4.2 A sequential algorithm for counting requests. 192
5.4.3 A faster method for counting requests....192
5.5 Routing tag assignment. 196
5.5.1 Principles of the method. 196
5.5.2 Sequential implementation. 197
5.5.3 Routing tag assignment using a copy network....204
5.6 Elimination of control packets. 212

5.6.1 Advantages of eliminating control packets. 212
5.6.2 Request counting without control packets. 212
5.6.3 Elimination of control packets in routing tag assignment. 220
5.6.4 Hardware counts associated with synthetic control packets. 223
5.7 Choice of switch modules. 225
5.7.1 Input stage modules. 225
5.7.2 Intermediate stage modules. 226
5.7.3 Output stage modules. 226
5.8 Fault tolerance. 227
5.9 Implementation of loss priority. 228
5.10 Implementation of input queuing. 230
5.11 A design example. 231

References 235

6. PERFORMANCE OF THE THREE-STAGE ATM SWITCH. 237

6.1 Introduction. 237
6.2 Numerical methods of loss calculation. 237
6.3 Performance of the path allocation algorithm 239
6.3.1 Performance with uniformly loaded output modules. 239
6.3.2 Performance with non-uniformly loaded output modules. 241
6.4 The output modules. 247
6.4.1 An output-buffered design. 247
6.4.2 Dimensioning of output buffers. 250

References 253

7. CONCLUSIONS. 254

APPENDIX A: Discrete-time queuing models ....

PUBLICATIONS arising from this research ....

GLOSSARY of acronyms used in the text ....

Abstract

The properties of switching techniques suitable for use in broadband networks have been investigated. Methods for evaluating the performance of such switches have been reviewed. A notation has been introduced to describe a class of binary self-routing networks. Hence a technique has been developed for determining the nature of the equivalence between two networks drawn from this class. The necessary and sufficient condition for two packets not to collide in a binary self-routing network has been obtained. This has been used to prove the non-blocking property of the Batcher-banyan switch. A condition for a three-stage network with channel grouping and link speed-up to be nonblocking has been obtained, of which previous conditions are special cases.

A new three-stage switch architecture has been proposed, based upon a novel cell-level algorithm for path allocation in the intermediate stage of the switch. The algorithm is suited to hardware implementation using parallelism to achieve a very short execution time. An array of processors is required to implement the algorithm. The processor has been shown to be of simple design. It must be initialised with a count representing the number of cells requesting a given output module. A fast method has been described for performing the request counting using a non-blocking binary self-routing network. Hardware is also required to forward routing tags from the processors to the appropriate data cells, when they have been allocated a path through the intermediate stage. A method of distributing these routing tags by means of a non-blocking copy network has been presented.

The performance of the new path allocation algorithm has been determined by simulation. The rate of cell loss can increase substantially in a three-stage switch when the output modules are non-uniformly loaded. It has been shown that the appropriate use of channel grouping in the intermediate stage of the switch can reduce the effect of non-uniform loading on performance.

1. SWITCHING IN BROADBAND ISDN.

1.1 Introduction.

Recent advances in fibre optic communications make a broadband public network, which will replace or augment the existing public switched telephone network, a practical goal. The deployment of narrowband ISDN (Integrated Services Digital Network) is a step towards achieving this goal, allowing as it does the integration of voice and data signals [1].

The evolution to a broadband network will be accompanied by the introduction of new services, which may include digital HDTV (high definition television), video conferencing of high quality, multimedia communications, and distributed computing [2]. The precise mix of services which will be provided is still uncertain.

It follows that a key requirement of broadband ISDN is that it must support the flexible provision of services. This means, in particular, that access must be supported at multiple bit rates. An obvious method of providing a choice of bit rate is to employ packet switching, whereby the principle of demand multiplexing allows variable rate access. However, existing packet switching techniques are unsuitable for broadband networks.

The alternative approach is to use multi-rate circuit switching. This has the advantage that the quality of service can be more readily assured, since the bandwidth available to a connection is fixed, and is not subject to statistical fluctuations.

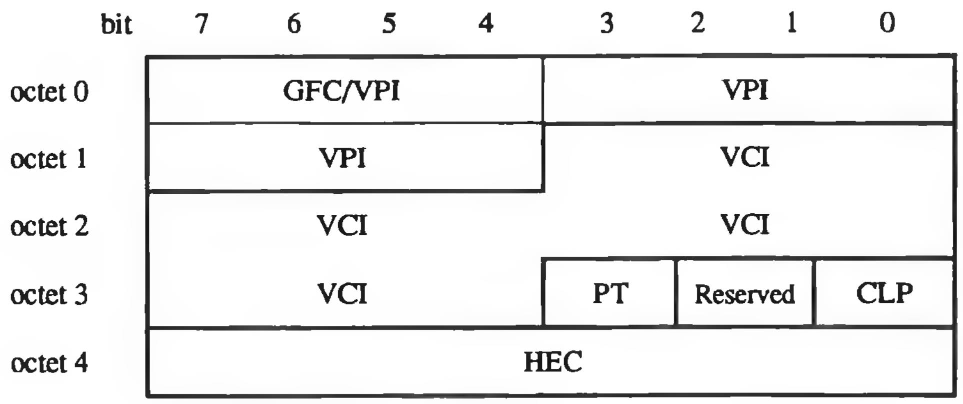
The respective merits of two approaches to broadband communications have been studied by the CCITT. These were STM (synchronous transfer mode, which corresponds to multi-rate circuit switching, and ATM (asynchronous transfer mode, which is equivalent to fast packet switching). ATM was chosen as the method for the transport of data in broadband ISDN [3,4]. The key features of ATM are:

• demand multiplexing. Data is transmitted in the form of fixed-length packets called cells. Each cell is 53 octets (i.e., 424 bits) long. Cells queue to gain access to network resources.

• high bit rate. Cells are transmitted at a bit rate of 155.52 Mb/s. Each cell occupies a time slot of approx. 2.5 μs duration.

• connection-oriented operation. Routing is based on a virtual channel identifier (VCI) or virtual path identifier (VPI). When the connection is established, a virtual channel must be set up through the network from source to destination, and a unique virtual channel identifier must be associated with that virtual channel on every link traversed by the data. Alternatively, the connection can be routed through a pre-allocated bundle of virtual channels known as a virtual path.

The 53 octet cell contains 48 octets of data and a 5 octet header. The structure of the cell header is shown in Fig. 1.1. It contains the virtual path identifier and virtual channel identifier, a cell loss priority (CLP) bit and error control bits. Also included in the header are bits for generic flow control (which are included only at the user-network interface) and a two-bit payload type field. A review of the CCITT recommendations as of August 1991 was given in [5].



GFC: Generic flow control
VPI: Virtual path identifier
VCI: Virtual channel identifier
PT: Payload type
HEC: Header error control
CLP: Cell loss priority

Fig. 1.1: The ATM cell header.

The combination of high bit rates and demand multiplexing means that special techniques are required in the design of ATM switches. Some connections may require a cell loss probability below 10^{-10} [6]. This will require stringent control of cell loss within the switch.

1.2 Objectives.

The objectives of this research are as follows:

- to review techniques for broadband switching;
- to investigate the topologies of binary self-routing networks;

- to determine conditions for the avoidance of blocking in three-stage and multi-stage networks;
- to evaluate numerical and analytical techniques for predicting the performance of ATM switches;
- to design an ATM switch of high throughput, and low loss, capable of expansion to several thousand inputs and outputs.

1.3 Summary.

Chapter Two contains a survey of switch designs which have been proposed for broadband switching. These designs are classified according to the basic principles of their operation, such as the use of multi-stage networks, or the use of broadcasts on a shared medium. These switch designs are critically compared, and recommendations are made concerning their suitability for use in ATM networks. A notation is introduced in Chapter Two for the compact description of binary self-routing networks. This facilitates the comparison of various designs of binary self-routing network, for which purpose the notation will be used in Chapter Four.

Chapter Three contains a description of techniques from discrete-time queuing theory and the theory of Markov chains which have been applied to model ATM switches. Methods for the analysis of multi-stage switches, input-buffered switches and output-buffered switches are discussed. Methods are also considered which allow variants on these switch types, such as shared-buffer switches, and switches with both input and output buffers, to be analysed. Models of source traffic used include homogeneous (Bernoulli) traffic, and bursty traffic. Bursty traffic is modelled by geometrically distributed burst and silence lengths at the input side of the switch, and by Markov-modulated sources at the switch outputs. The issue of fairness of access to the output ports in Batcher-banyan switches is considered in some detail. Recommendations are made concerning mechanisms to ensure fair access, and simulation results are presented to evaluate the effectiveness of these mechanisms.

Chapter Four considers the topological equivalence between binary self-routing networks. Three stronger types of equivalence are introduced, namely input equivalence, output equivalence, and exact equivalence. Attention is restricted to that class of binary self-routing networks where the interconnections between stages are drawn from a set termed the set of binary permutations. This restriction allows a powerful method to be developed for investigating the equivalence between two networks. The method is used to investigate routing strategies in binary self-routing networks, and is also applied to obtain the necessary and sufficient condition for two cells not to collide in a binary self-routing network. An elegant proof is given that the sorting operation in the Batcher-banyan switch is sufficient to ensure that the above condition is satisfied by the cells entering the banyan network. Previous authors have presented proofs that the Batcher-banyan switch is internally non-blocking, but they did not relate their proof to the necessary and sufficient condition for cells not to collide in the banyan network.

The condition for a three-stage switch to be non-blocking has been obtained in Chapter Four for the case where the inputs and outputs of the intermediate stage modules feature channel grouping and link speedup. Special cases of this condition are obtained which apply to multi-rate and single-rate circuit switching. Hence it is shown that a different non-blocking condition applies to a three-stage ATM switch where routing through the intermediate stage is performed at cell level, than to the corresponding switch with call-level routing.

A new design of three-stage ATM switch is described in Chapter Five. The design supports the use of channel grouping in the intermediate stage. Routing through this stage is performed at cell level. The new design requires a lower operating speed for the circuitry which implements the cell level path allocation than would be the case for previously published algorithms. This benefit is obtained at the expense of an increase in the amount of hardware required. The key component of the switch is an array of processors which try to reserve a path through the intermediate stage of the switch for every incoming cell. The logic required to implement the processor is described, so as to demonstrate that the processors are of low complexity, and so can be operated at the required clock rate. Additional hardware is required to initialise the processor array at the start of each time slot, and to forward routing tags to the cells indicating the available path through the intermediate stage. This hardware makes extensive use of non-blocking banyan networks, so as to ensure high-speed operation.

The performance of the new switch is evaluated in Chapter Six. Simulation techniques are used to determine the cell loss probability, due to loss of contention for a path through the intermediate stage, for an output-buffered switch. The advantages of channel grouping are demonstrated by the results obtained. It is also shown that the switch is relatively insensitive to non-uniform loading of the outputs. The performance of the output stage of the switch is evaluated analytically, using one of the methods described in Chapter Three.

The results of the above research are summarised in Chapter Seven.

References

[1] L. Kleinrock, "ISDN - the path to broadband networks", *Proc. IEEE*, vol. 79, no. 2, pp. 112-117, Feb. 1991.

[2] CCITT Recommendation I.211, "B-ISDN service aspects", 1990.

[3] CCITT Recommendation I.121, "Broadband aspects of ISDN", 1990.

[4] CCITT Recommendation I.361, "B-ISDN ATM layer specification", 1990.

[5] M. Kawarasaki and B. Jabbari, "B-ISDN architecture and protocol", *Journal Of Select. Areas Commun.*, vol. 9, no. 9, pp. 1405-1415, Dec. 1991

[6] CCITT Draft Recommendation I.35B, "B-ISDN ATM layer cell transfer performance".

2. BROADBAND SWITCH ARCHITECTURES.

2.1 Introduction.

A wide range of switch designs has been proposed for use in ATM networks [1-4]. Most broadband switches adopt one of a small number of basic techniques in order to achieve the high switching rates required whilst accommodating the unscheduled nature of cell arrivals. These basic techniques are investigated in detail below, and their strengths and weaknesses are assessed. Methods for evaluating the performance of ATM switches will be considered in Chapter Three.

The term ATM is of recent origin. Prior to its adoption, switches of the type appropriate for an ATM network were typically referred to as fast packet switches. Two of the earliest such switches proposed were the Starlite switch [5] and Turner's Fast Packet Switch [6].

Both of these switch designs achieved high speed operation by employing self-routing networks of the type first proposed for use as interconnection networks in a parallel processing environment [7]. These interconnection networks include the Omega network [8], the indirect binary n-cube [9], the baseline network [10] and the SW-banyan [11]. The above networks have been shown to be topologically equivalent [10]. The self-routing property is achieved by employing digit-controlled switch elements, and so these switches are all examples of delta networks [12]. The Starlite switch in addition employs a sorting network, of the type first proposed by Batcher [13].

2.2 Delta networks.

The definition of delta networks given by Patel [12] was as follows:

A delta network is an a^n x b^n switching network with n stages consisting of (for stage $1 \leq i \leq n$), $a^{n-i}\, b^{i-1}$ crossbar modules. Each crossbar module has a inputs and b outputs and is digit-controlled, i.e. an input is connected to an output d ($0 \leq d < b$) if a control digit (base b) supplied by the input is d. The module can arbitrate among conflicting requests for output d. The digit is chosen from the destination address of the input data. All inputs and outputs of the crossbar modules are connected. The link pattern between stages is such that a unique path of constant length exists between any source and any destination in the network.

From this definition, it can be seen that the class of banyan networks [11] has much in common with delta networks. In particular, an SW-banyan is an example of a delta network, if the 'cross-bars' of which it is composed are digit-controlled. Also, the interconnection networks mentioned above (such as the Omega network) are examples of delta networks where the crossbar modules are 2x2 switches.

2.3 Self-routing networks based on 2x2 switches.

2.3.1 Principles of self-routing networks.

Many self-routing switches have been proposed by different authors, and a variety of notations, terminologies, etc., are in use to describe such switches. A consistent method for the description of these switches, which is similar to that adopted by Wu and Feng [10], is described in the following. This method will be used in Chapter Four to analyse the conditions under which blocking occurs in an arbitrary binary self-routing network.

Most self-routing networks considered for use in ATM may be represented as shown in Fig. 2.1. It can be seen that the switch has 2^n inputs and 2^n outputs, and comprises n stages of 2^{n-1} switch elements. These switch elements are bit-controlled as follows. The input cells are transmitted serially through the network, with an n-bit destination tag $(d_{n-1}d_{n-2}......d_1d_0)$ as prefix; the switch elements present in stage k of the switch route their input cells to the upper or lower output according to the value of one of the routing tag bits, which is typically d_k; the cell is routed to the higher output if d_k = '1', and otherwise is routed to the lower output. A difficulty arises when both cells present at the switch element inputs have the same value for d_k, i.e., when they are contending for the same output. Methods of dealing with this contention shall be considered later.

If the link permutations (which are simply hard-wired re-arrangements of the order in which the outputs of one stage are connected to the inputs of the succeeding stage) are appropriately chosen, then a cell with a given destination tag will be routed to the same output port, irrespective of the input port at which it arrives, having passed through n switch elements, each of which routes it on the basis of a single (unique) bit of its destination tag. It is because of this property that such networks are described as self-routing.

2.3.2 A notation to describe self-routing networks.

The link permutations are what distinguish among these self-routing switch designs. Thus, to uniquely describe a particular network of the type in Fig. 2.1 it is sufficient to define the n link permutations $LP_0,...,LP_n$. A notation to describe the permutations of interest will be described below.

Two possible permutations of eight links are illustrated in Fig. 2.2. The simplest possible permutation is shown in Fig. 2.2(a), and is here called the straight-through (ST) permutation . In this permutation, input line 0 is linked to output line 0, input line 1 to output line 1, etc. In other words the ST permutation is a null permutation. Typically, at least one from the n-th link permutation (LP_n) and link permutation zero (LP_0) will be an ST permutation. Another common permutation is the perfect shuffle [14] of Fig. 2.2(b). The outputs labelled 0,1,.....,7 are connected to inputs labelled 0,4,1,5,2,6,3,7 respectively. Thus, the upper four inputs and the lower four inputs have been interleaved at the outputs. This is termed a perfect shuffle by analogy with the permutation produced by cutting a deck of cards in the middle and interleaving each half.

A link permutation can be regarded in mathematical terms as a transformation from the set of input line numbers to the set of output line numbers. Evidently, the transformation must be bijective (one-to-one and onto) if an input line is not to be linked to two output lines or *vice versa*. Let the set I_n denote the set of integers $\{0,1,2,......,2^n-1\}$. Then, in a permutation of 2^n lines, the output port labelled Y is connected to the input port labelled X if

$$Y = \tau(X)$$

where $\tau : I_n \rightarrow I_n$ is a bijective transformation.

One permutation (τ_a) can be immediately followed by another (τ_b), to produce a third permutation (τ_c), i.e., one where the output lines from the permutation defined by τ_a are connected as input lines to the link permutation defined by τ_b. The output port Y connected to input port X is then given by

$$Y = (\tau_b \circ \tau_a)(X) \;\underline{\Delta}\; \tau_b(\tau_a(X)).$$

It follows that $\tau_c = \tau_a \tau_b$ by analogy with matrix multiplication.

An alternative representation of a link permutation involves the use of a 2^n x 2^n matrix **M**, whose entries are defined by

$$M_{ij} = \begin{matrix} 1, & \text{if a link connects input } i \text{ to output } j \\ 0, & \text{otherwise.} \end{matrix}$$

Since only one link can be connected to each input or output, it follows that

$$\sum_{0 \le i < 2^n} M_{ij} = 1$$

and

$$\sum_{0 \le j < 2^n} M_{ij} = 1.$$

Evidently, if one permutation ($\mathbf{M}_a$) is followed by another ($\mathbf{M}_b$) to produce a third ($\mathbf{M}_c$), then

$$M_c = M_a M_b.$$

In order to identify the output link k to which input link i is connected via a link permutation represented by the matrix **M**, the following is calculated:

$$\mathbf{e}_{out} = \mathbf{e}_i \, \mathbf{M},$$

where $\mathbf{e}_i$ is a row vector containing 2^n entries defined by:

$$(\mathbf{e}_i)_j = \delta_{ij}$$

(where δ_{ij} is the Kronecker delta function)

and where $\mathbf{e}_{out}$ is a similar row vector defined by

$$(\mathbf{e}_{out})_j = \delta_{jk}.$$

The scalar value of j can be recovered using

$$j = \mathbf{e}_{out} \cdot \mathbf{v},$$

where v is a column vector with 2^n entries defined by

$$\mathrm{v}_k = k,\ 0 \leq k < 2^n.$$

The transformation notation shall be used henceforth because of its greater simplicity. However, it is apparent, from the analogy developed here, that the rules of matrix multiplication and inversion apply to operations involving consecutive or inverse link permutations. These rules shall be applied in Chapter Four to elucidate some of the properties of self-routing networks.

In general, a complete description of τ would require an ordered listing of the set of output link numbers, i.e. $\{\tau(0), \tau(1), \ldots\ldots, \tau(2^n-1)\}$. However, most of the permutations of interest in self-routing networks can be described more compactly, if the values of X and Y are represented in binary notation. Suppose that $X = b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \ldots + b_0$ and $Y = c_{n-1}2^{n-1} + c_{n-2}2^{n-2} + \ldots + c_0$, where $\tau(X) = Y$. The transformation $\tau(\,)$ will be referred to as a *binary permutation* if, for arbitrary X and Y,

$$c_{\tau'(i)} = b_i, 0 \leq i < n$$

where

$$0 \leq \tau'(i) < n,$$

and where

$$\tau'(i) = \tau'(j) \Leftrightarrow i = j.$$

In other words, $\tau'()$ is a permutation of the integers from 0 to n - 1. A binary permutation is thus simply a re-ordering of bits. This means, in particular, that the number of binary 1's in X and Y must be the same. An example of a binary link permutation is the perfect shuffle of eight lines depicted in Fig. 2.2(b). This can be defined by :

$$0 \xrightarrow{\tau'} 1$$
$$1 \xrightarrow{\tau'} 2$$
$$2 \xrightarrow{\tau'} 0$$

or

$$b_0 \xrightarrow{\tau} b_1$$
$$b_1 \xrightarrow{\tau} b_2$$
$$b_2 \xrightarrow{\tau} b_0.$$

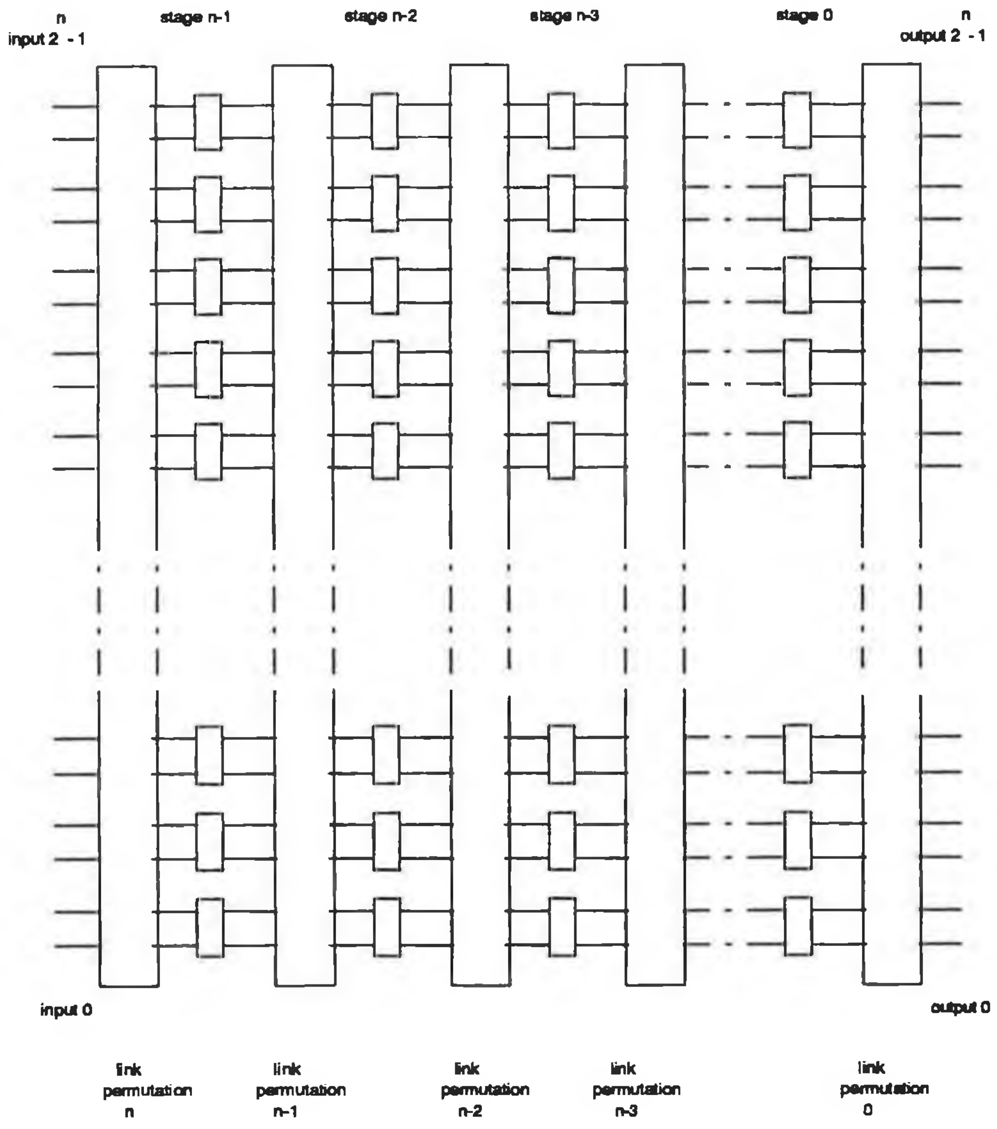


Fig. 2.1: A binary self-routing network.

More compactly, this may be written as

$$\tau\ (b_2 b_1 b_0) = b_1 b_0 b_2.$$

It shall be shown in section 2.3.4 that the link permutations used in many popular self-routing networks belong to the class of binary permutations. An example of a link permutation which falls outside this class is the crossover permutation [15].

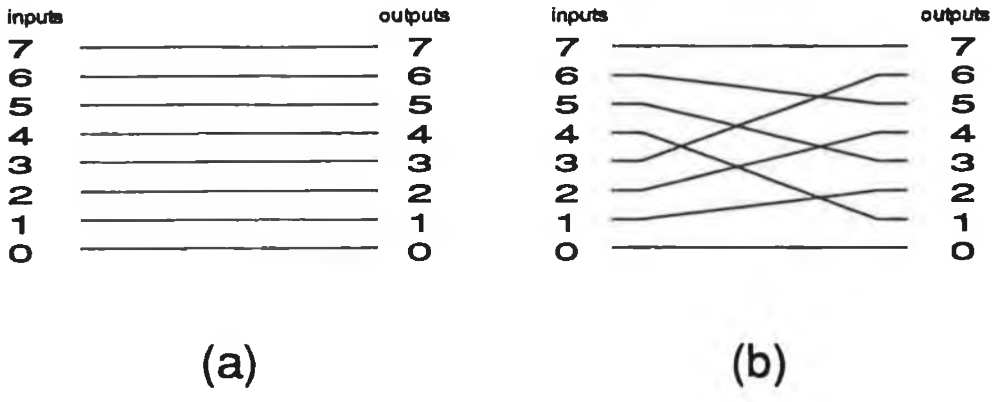


Fig. 2.2: Some link permutations.

The names chosen for the permutations of interest, and their definition using the notation introduced above, are as follows. It is assumed that the permutations are of 2^n lines, so that n bits are required to identify a line.

2.3.3 Some link permutations.

2.3.3.1 The perfect shuffle.

The perfect shuffle of all 2^n inputs is represented by the transformation $\mathbf{PS_n}$, which corresponds to a cyclic shift *left* of the input port number, i.e.

$$PS_n(b_{n-1}b_{n-2}.....b_0) = b_{n-2}........b_0 b_{n-1}.$$

Often, the perfect shuffle is performed on equal-sized subsets of inputs, e.g., separately on the upper and lower 2^{n-1} lines. Thus, the $\mathbf{PS_k}$ transformation is defined by :

$$PS_k(b_{n-1}b_{n-2}.....b_0) = b_{n-1}b_{n-2}.....b_k b_{k-2}...b_0 b_{k-1}.$$

This is a cyclic shift left performed on the least significant k bits of the input line number only. It is equivalent to grouping the inputs into 2^{n-k} contiguous groups, and performing a full perfect shuffle on each group individually. The four possible perfect shuffles for sixteen lines are shown in Fig. 2.3. Evidently, the **PS_1** transformation is equivalent to the ST transformation.

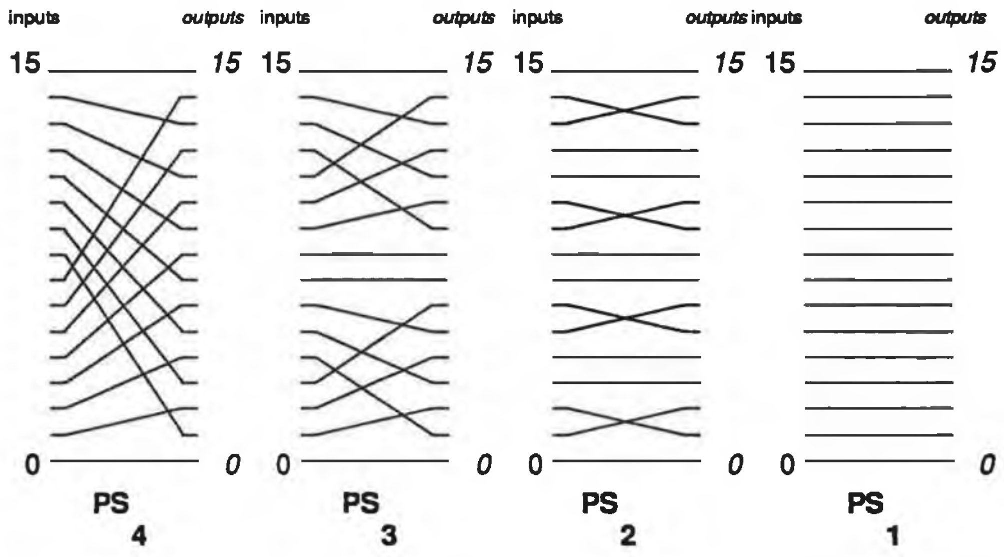


Fig. 2.3: The perfect shuffle.

2.3.3.2 The inverse perfect shuffle.

Another family of permutations is obtained from the inverse perfect shuffle, i.e. when the perfect shuffle is performed on the output lines to obtain the input lines, rather than *vice versa*. The IPS_k transformation is defined as a cyclic shift *right* of the least significant k bits of the input port number, i.e.,

$$IPS_k(b_{n-1}b_{n-2}.....b_0) = b_{n-1}b_{n-2}.....b_kb_0b_{k-1}...b_1.$$

A PS_k permutation followed by an IPS_k permutation reduces to a ST permutation, hence the name *inverse* perfect shuffle. Example inverse perfect shuffles are shown in Fig. 2.4. Note that the IPS_2 and PS_2 transformations are identical.

2.3.3.3 The swap permutation.

Another permutation, which is here called the swap permutation, is obtained by following an IPS_k permutation by a PS_{k-1} permutation. This gives the transformation S_k defined by

$$\begin{aligned} S_k(b_{n-1}b_{n-2}.....b_0) &= PS_{k-1}(IPS_k(b_{n-1}b_{n-2}.....b_0)) \\ &= PS_{k-1}(b_{n-1}b_{n-2}.....b_kb_0b_{k-1}...b_1) \\ &= b_{n-1}b_{n-2}....b_kb_0b_{k-2}...b_1b_{k-1}. \end{aligned}$$

Using the shorthand notation introduced earlier, this result may be written as

$$S_k = IPS_k PS_{k-1}$$

The transformation is so called because the least significant bit (b_0) and the k-th least significant bit (b_{k-1}) exchange places. Evidently, the swap permutation is its own inverse, i.e. $S_k(S_k(X)) = X$. The swap permutation is illustrated in Fig. 2.5.

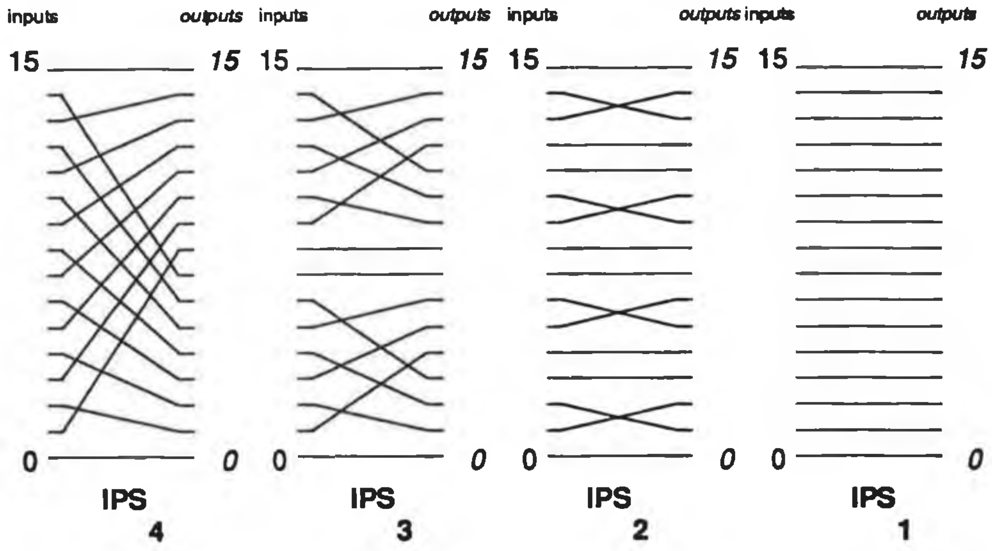


Fig. 2.4: The inverse perfect shuffle.

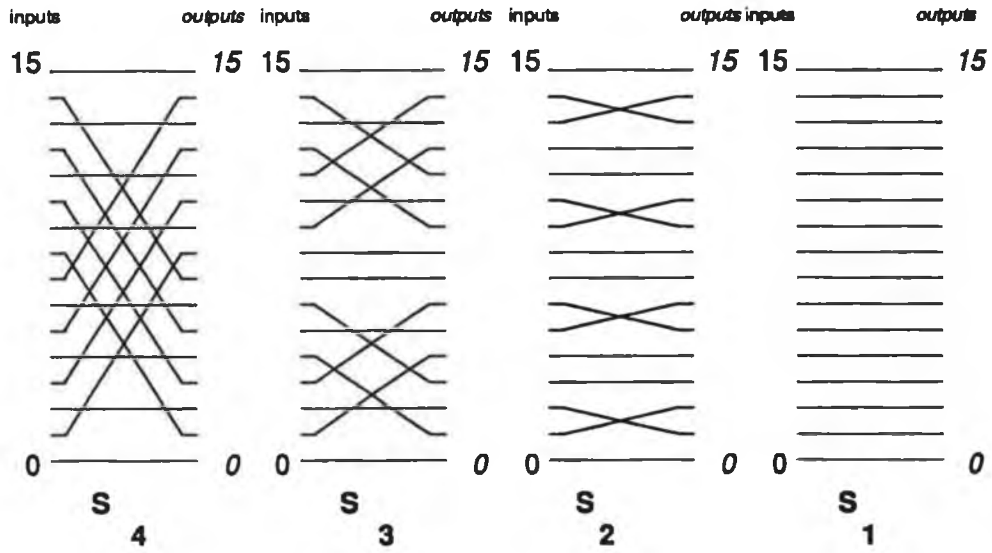


Fig. 2.5: The swap permutation.

2.3.3.3 The bit-reversal permutation.

The bit-reversal permutation R_k is defined by

$$R_k(b_{n-1}b_{n-2}.....b_0) = b_{n-1}b_{n-2}.....b_k b_0 b_1 b_2 ... b_{k-1}.$$

In other words, it consists of a bit reversal on the k least significant bits of the link address. The bit-reversal permutation is illustrated in Fig. 2.6.

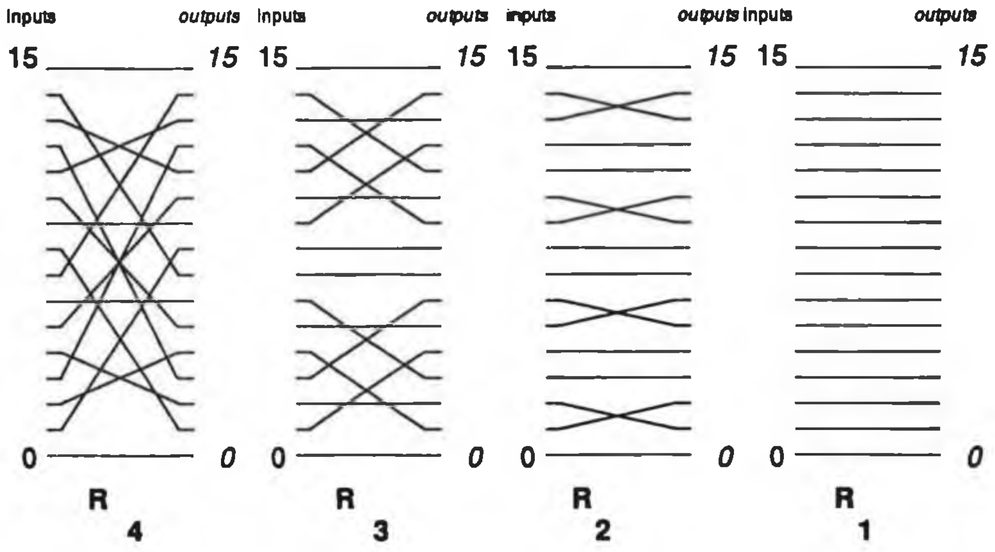


Fig. 2.6: The bit-reversal permutation.

2.3.4 Network definitions.

2.3.4.1 Baseline network.

The baseline network [10] is of the form shown in Fig. 2.1, with 2^n input and output lines, and with the link permutations defined by

$$LP_n = LP_0 = ST,$$

$$LP_k = IPS_{k+1}, \quad 0<k<n, \ k \text{ integer}.$$

A 16x16 baseline network is shown in Fig. 2.7.

2.3.4.2 Omega network.

The omega network [8] is of the form shown in Fig. 2.1, with 2^n input and output lines, and with the link permutations defined by

$$LP_0 = ST,$$

$$LP_k = PS_n, \qquad 0<k\leq n, \ k \text{ integer}.$$

A 16x16 omega network is shown in Fig. 2.8.

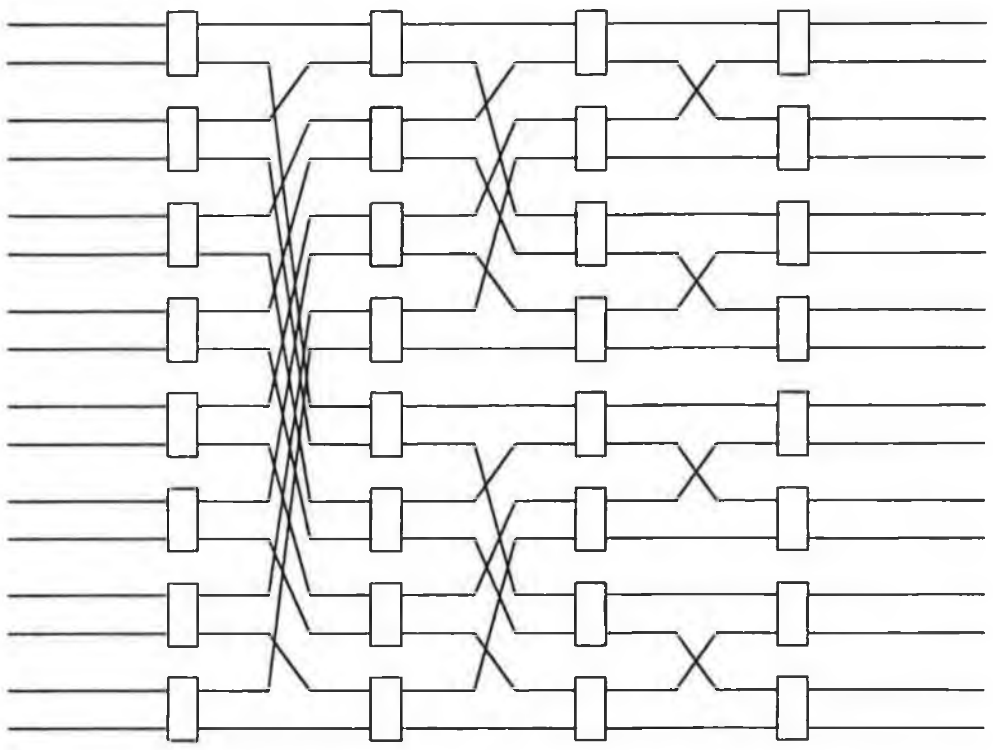


Fig. 2.7: The baseline network.

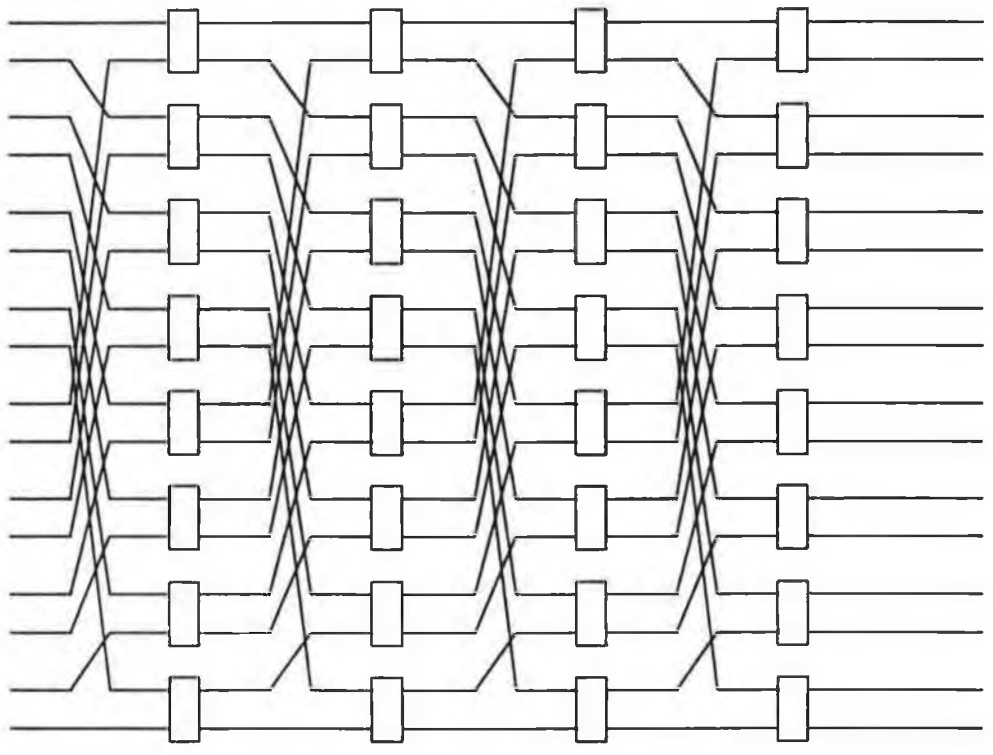


Fig. 2.8: An omega network.

2.3.4.3 Indirect binary n-cube.

The indirect binary n-cube [9] is a network of the form shown in Fig. 2.1, with 2^n input and output lines, and with the link permutations defined by

$$LP_n = ST,$$

$$LP_0 = IPS_n$$

$$LP_k = S_{n-k+1},\ 0<k<n,\ k \text{ integer.}$$

A 16x16 indirect binary n-cube network is shown in Fig. 2.9.

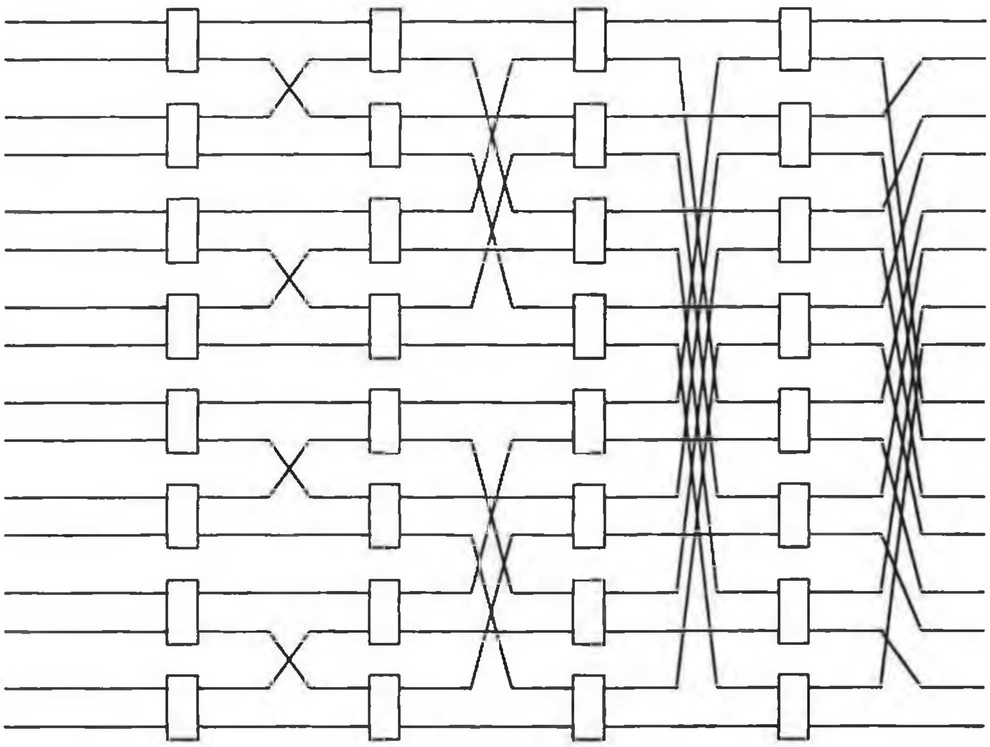


Fig. 2.9: The indirect binary n-cube.

2.3.4.4 The GL-banyan.

Goke and Lipovski presented definitions of general banyan networks, and of specific banyan structures [11]. The focus here shall be on those banyan structures that can be represented by Fig. 2.1. In the terminology of Goke and Lipovski, the number of stages in this switch is referred to as the number of *levels*, i.e., an L-level banyan has *L* stages of switching. The number of inputs and outputs of each switch element is referred to as the *fanout*(*F*) and the *spread*(*S*) respectively. If the values for fanout and spread are the same for each switch element, the resulting structure is referred to as a *regular* banyan. Thus, using this terminology, Fig. 2.1

depicts a regular (n-level) banyan with $S=F=2$. Goke and Lipovski described a recursive technique for synthesising a specific regular banyan called the SW banyan. Applying this technique in the special case where $S=F=2$, it is found that a 1-level SW banyan is equivalent to a 2x2 switch element, and that an L-level SW banyan may be synthesised by interconnecting two (L-1)-level banyans and 2^{L-1} 2x2 switch elements as shown in Fig. 2.10.

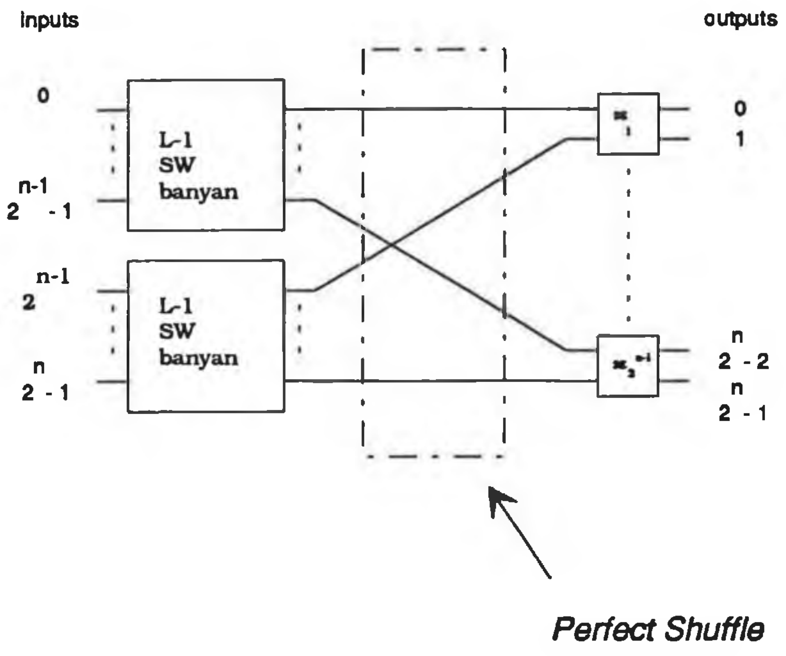


Fig. 2.10: Recursive rule for banyans.

Fig. 2.10 is an interpretation of Fig. 8 of [11] for the special case where 2x2 switches are used. The interconnection pattern used must be such that, if the upper input port of SE_k is connected to output port j of the upper (L-1)-level SW banyan, then the lower input port of SE_k must be connected to output port j of the lower (L-1)-level SW banyan. Thus, allowable link permutations include the PS_L and the S_L. The PS_L transformation is implied by Fig. 8 of [11], which illustrates the recursive rule, but two other examples of SW banyans presented in that paper used the swap permutation. Accordingly, other authors, in reporting on the SW banyan, have implicitly accepted the swap permutation as the one to be used. Accepting this convention, the resulting banyan shall be referred to as the GL-banyan. This is a network of the form shown in Fig. 2.1, with 2^n input and output lines, and with the link permutations defined by

$$LP_n = LP_0 = ST,$$

$$LP_k = S_{n-k+1},\ 0<k<n,\ k \text{ integer}.$$

A 16x16 GL-banyan network is shown in Fig. 2.11. It is evident that the GL-banyan is identical to the indirect binary n-cube, with the exception that the inverse perfect shuffle of the output port lines is not performed.

2.3.4.5 The P-banyan.

If, instead of using the swap permutation in recursively constructing the GL-banyan, the perfect shuffle had been used, the resulting banyan would be defined by

$$LP_n = LP_0 = ST$$

$$LP_k = PS_{n-k+1} \qquad 0<k<n,\ k \text{ integer.}$$

This network is referred to as a P-banyan. A 16x16 example of such a network is shown in Fig. 2.12.

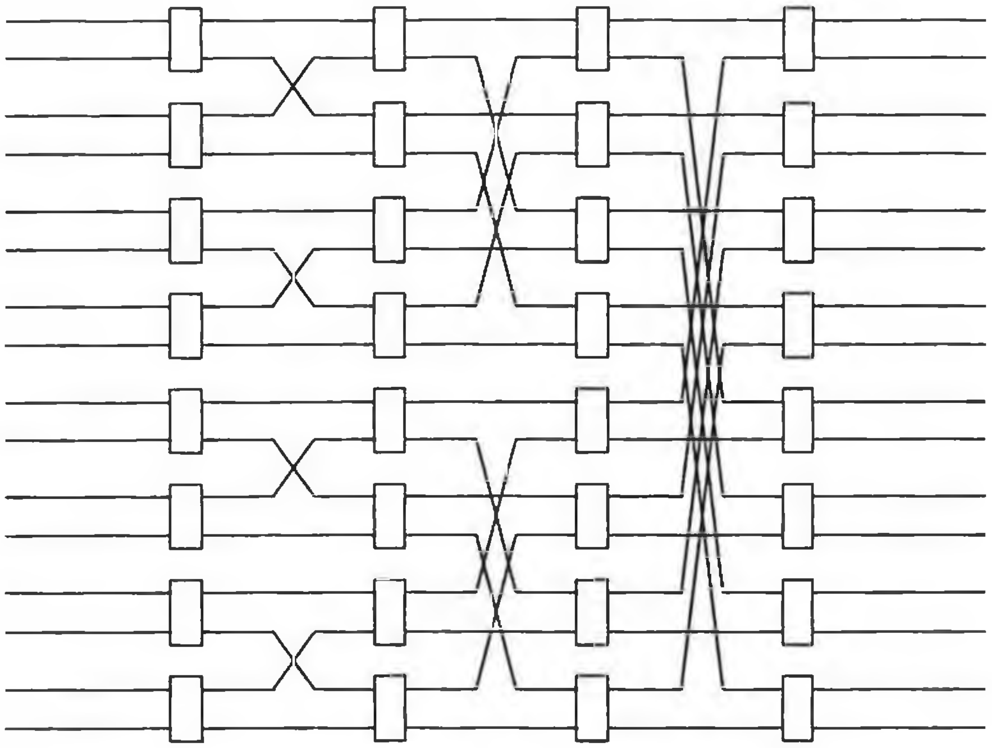


Fig. 2.11: The GL-banyan.

2.3.4.6 The S-banyan.

A certain amount of confusion exists in the literature regarding banyan networks, because of the generality of the definition of the banyan, as introduced by Goke and Lipovski [11]. Thus, all of the structures considered above can be regarded as banyan networks. An attempt has been made in this discussion to avoid confusion by always using the more specific title for a particular network (e.g. the baseline network) where it exists, and by inventing titles for networks without specific names (e.g., the GL-banyan, which would otherwise need to be described as 'a regular L-level banyan, with $S=F=2$, synthesized recursively using the swap permutation', in order to describe it completely). Unfortunately, the term 'banyan', without further qualification, is often used by the switching community to refer to the structure defined by :

$$LP_n = PS_n$$

$$LP_0 = ST$$

$$LP_k = S_{k+1}, \quad 0<k<n, \ k \text{ integer.}$$

This is referred to as the S-banyan. A 16x16 S-banyan is shown in Fig. 2.13.

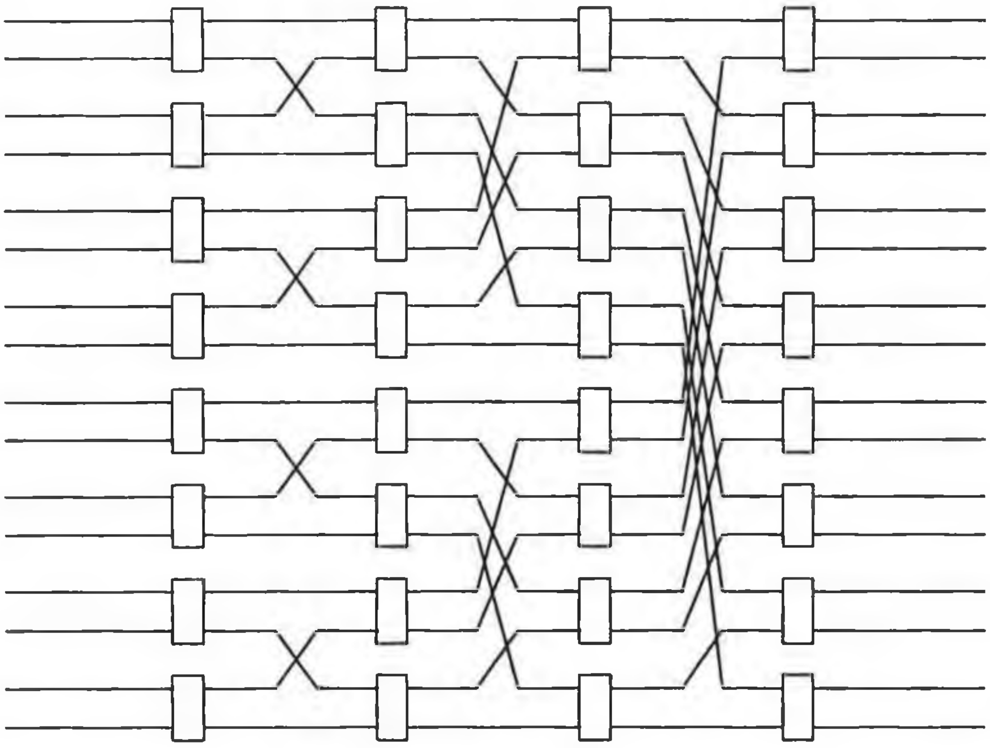


Fig. 2.12: The P-banyan.

2.3.5 Inverse networks.

The inverse of a permutation may be defined as that permutation which arises when the input and output lines are interchanged. It is apparent that a permutation, immediately followed by its inverse permutation, is equivalent to the straight-through permutation, i.e.,

$$ST = \tau\tau^{-1},$$

or

$$\tau^{-1}(\tau(X)) = X.$$

It has already been noted that the swap permutation is its own inverse, and that the **IPS**$_k$ permutation is the inverse of **PS**$_k$.

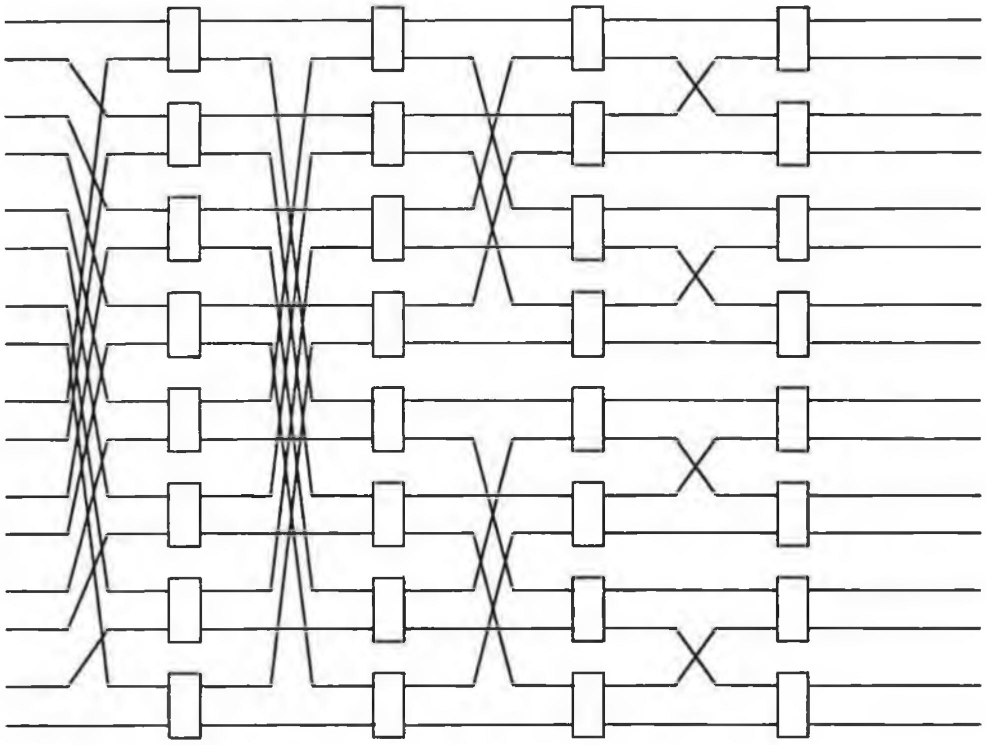


Fig. 2.13: The S-banyan.

The inverse network is obtained, assuming bi-directional switching elements, simply by applying the inputs at the right hand side of Fig. 2.1, and reading the outputs at the left hand side, i.e. by 'turning around' the network. More formally, the network defined by the transformations $\{LP'_0, LP'_1, ..., LP'_n\}$ is said to be the inverse network of that defined by $\{LP_0, LP_1, ..., LP_n\}$ if and only if

$$LP'_0 = [LP_n]^{-1}$$

$$LP'_n = [LP_0]^{-1}$$

$$LP'_k = [LP_{n-k}]^{-1}, \qquad 0<k<n.$$

Thus it may be seen that the inverse of the S-banyan is the indirect binary n-cube. New networks can be obtained simply by inverting other networks using the above formula.

2.3.6 Topological equivalence.

It would appear, at first sight, that a bewildering range of multi-stage interconnection networks is possible, each network with its own unique properties. In fact, it has been shown by Wu and Feng [10] that many of these networks are topologically equivalent. Their method was to renumber the switches in each stage (so that each switch was given a *logical* number, in addition to its *physical* number, which represents its position in the stage), and the input and output lines, of a multi-stage interconnection network. The network was then shown to be equivalent to a baseline network by appropriate choice of the logical numbers. The choice of logical numbers was such that the logical numbers of the switch elements linked to any given switch element (say the switch in stage k with logical number j) were the same as the physical numbers of the switches, in a baseline network, linked to the switch element in stage k whose physical number is j. Thus, to prove topological equivalence using this method, it sufficed to discover a mapping rule which mapped the physical numbers of the switch elements in a network into appropriate logical numbers. Using this method, Wu and Feng have shown the following networks to be equivalent to the baseline network:

i) the Omega network;

ii) the GL-Banyan;

iii) the flip network in STARAN [16] - this is equivalent to an inverse Omega network, although it is intended for use as a permutation network, rather than a self-routing network;

iv) the modified data manipulator - this is equivalent to an S-Banyan, but with the LP_0 permutation modified to be straight-through (the original data manipulator of [17] used 3x3 switch elements and thus cannot be represented by Fig. 2.1);

v) the inverse baseline network - Wu and Feng referred to this as the reverse baseline network.

It is readily apparent that the S-Banyan must also be equivalent to the above networks, since it differs from the modified data manipulator only in the ordering of the inputs to the first stage. Although all of these networks are topologically equivalent, the conditions which give rise to blocking in these networks may not be the same. This issue is considered further in Chapter Four, where the blocking properties of binary self-routing networks are discussed.

2.4 Sorting networks.

2.4.1 Principles of sorting networks.

The switch elements in Fig. 2.1 are assumed to be bit-controlled switches, and, with appropriate choices of link permutations, the network is a self-routing network. A total of n stages of switching is required to switch 2^n inputs to any one of 2^n outputs. If instead, the switch elements are replaced by sorting elements, then, with suitable selection of link permutations, a sorting network can be constructed. However, more than n sorting stages are required to sort 2^n items of data. For example, three sort stages are required for a four-input sorter. A sort element is a two-input, two-output device which compares the serial data arriving on the two inputs, and routes the data of higher value to the upper (lower) output. Thus the order in which the data appears at the outputs of the sorting network is a sorted list, i.e. the data of highest value appears at the highest (lowest) numbered output port, and that of lowest value appears at the lowest (highest) numbered port.

When a sorting network is used in switching applications, the data is prefixed by a routing tag. Consequently, the data at the outputs is sorted by destination. It is apparent that, where the set of destinations requested includes each possible destination (without repetitions) then the output ports of the sorter at which the data appears will correspond to the port identified by its routing tag. The sorting network thus performs the routing function, for this special set of input data. Two mechanisms prevent this pattern of output destination requests appearing at the inputs to an ATM switch. Firstly, not all input ports may have active cells present. If the number of output ports matches that of input ports, this precludes the possibility of all output ports being requested by the incoming data. Secondly, some destinations may be requested by more than one input, due to the unscheduled nature of arrivals in ATM. Thus additional hardware is required if a sorting network is to be used to implement the routing function.

2.4.2 The Batcher sorter.

A variety of algorithms is available for performing sorts [18]. The most popular of those which lend themselves to distributed decision-making by a network of sorting elements is that proposed by Batcher [13]. The chief reason for the popularity of Batcher's design is its regular structure, which lends itself to integration using VLSI, and to expansion.

A Batcher sorting network is constructed by the appropriate interconnection of bitonic sorters. A bitonic sorter is a network which sorts a bitonic sequence (the juxtaposition of two monotonic sequences, one ascending and the other descending) into a monotonic sequence. The procedure to sort 2^n items of data using bitonic sorters is described below.

The 2^n inputs are applied to 2^{n-1} 2-input bitonic sorters (i.e. 2-input sort elements). This generates 2^{n-1} 2-entry monotonic lists. Two such lists can then be juxtaposed to form a four-entry bitonic list. This list is merged by a four-input bitonic sorter into a monotonic list. This procedure is repeated to produce merged lists of 8,16,.....,2^n entries. A block diagram of such a sorter is shown in Fig. 2.14. The arrows show the outputs to which the highest-valued data items are sorted.

The problem of designing a sorting network thus reduces to the problem of designing a bitonic sorter. Batcher demonstrated that a $2n$-item bitonic sorter could be constructed recursively from n sort elements and two n-item bitonic sorters using the method of Fig. 2.15. Since a two-input bitonic sorter can be implemented using a single sort element, it is apparent that k stages of sort elements are required to sort a bitonic list of length 2^k. It can also be seen that the interconnection pattern (link permutation) between stages k and k-1 is an IPS_{k+1} followed by a PS_k permutation, i.e. an S_{k+1} permutation. Thus, a bitonic sorter with 2^n inputs can be represented by Fig. 2.1, if the switch elements are replaced by sort elements, where:

$$LP_n = PS_n\ ,$$

$$LP_0 = ST\ ,$$

$$LP_k = S_{k+1}, \quad 0<k<n.$$

This set of link permutations defines the S-banyan. It may be concluded that the structures of the bitonic sorter and the S-banyan are identical, apart from the function of the 2 x 2 elements used. This has important implications for the blocking properties of the S-banyan.

A bitonic sorter with 2^n inputs requires n stages of sort elements. Also, from Fig. 2.14, it can be seen that to construct a sorting network for 2^n numbers from bitonic sorters requires one 2^n input bitonic sorter, two 2^{n-1} input, four 2^{n-2} input, ... and 2^{n-1} two input bitonic sorters. Thus the total number of stages required is:

$$\sum_{1}^{n} i = \frac{1}{2}n(n+1).$$

Each stage contains 2^{n-1} sort elements. For example, the sixteen-input Batcher sorter of Fig. 2.16 requires 80 sort elements.

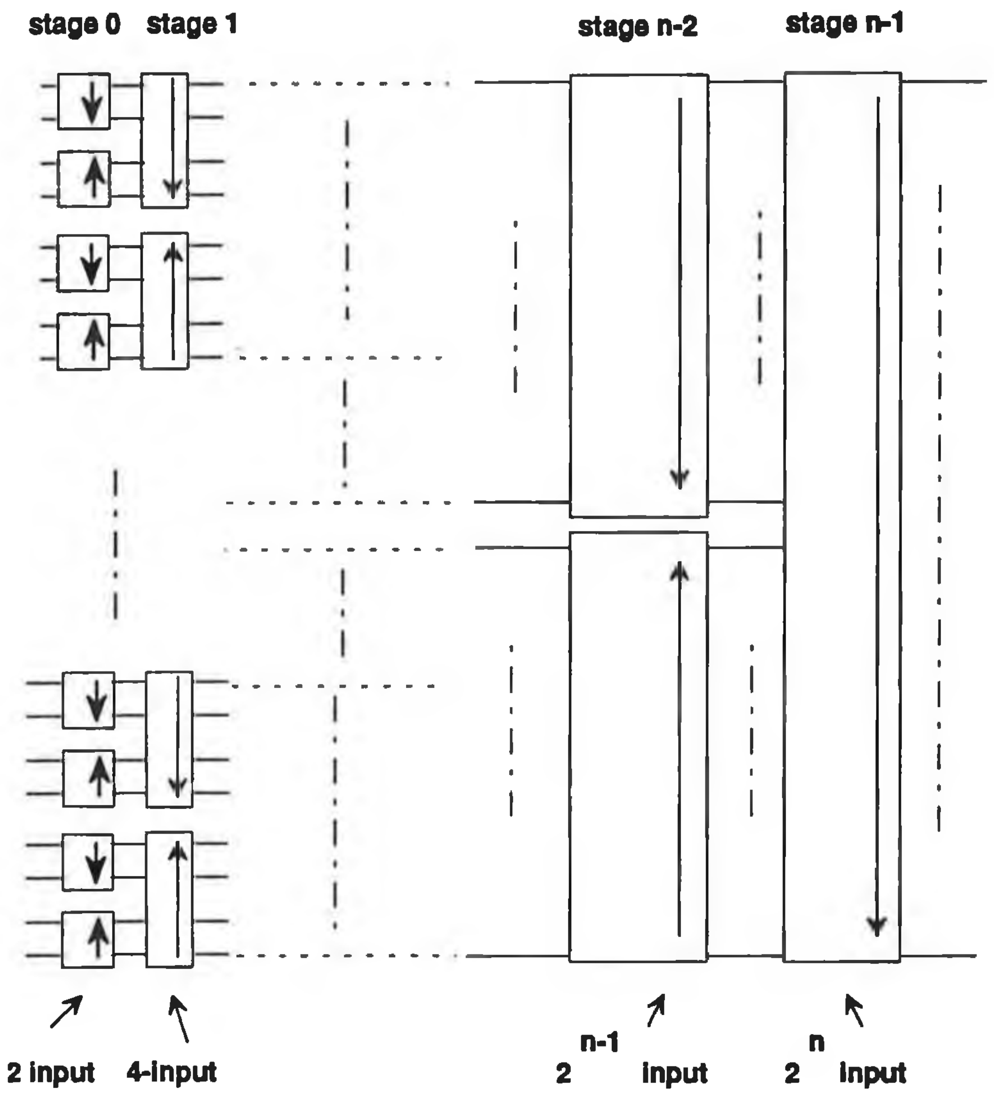


Fig. 2.14: Constructing a sorter from bitonic sorters.

2.5 Switch architectures based on the banyan network.

2.5.1 Banyan switches.

Fast packet switch architectures were first proposed in the early 'eighties [19-21]. Turner and Wyatt [21] proposed such an architecture for use in ISDN. A key feature of the architecture was the use of a self-routing switch. Self-routing switches (such as the buffered delta network of Dias and Jump [22]) had previously been developed for use as multistage interconnection networks in advanced computers, and it was suggested that a switch of this type could be employed. Turner extended these ideas into a proposal for an integrated services packet network [6] which used the banyan network as switch fabric. Turner, citing [23], pointed out that a number of designs for what he termed binary routing networks was possible. His illustrations of the switch fabric showed a baseline network. The switch elements in this switch fabric are single-buffered, and a back-pressure mechanism is used to prevent a packet arriving at a switch element when the buffer is full. This routing network is preceded, in order to minimise congestion, by a distribution network of identical structure, wherein each switch element routes packets alternately out of its two output ports, or from the first available port if both ports are not available. Turner suggested the use of 4x4 switch elements, in place of 2x2 elements, to reduce the number of stages in the switch fabric.

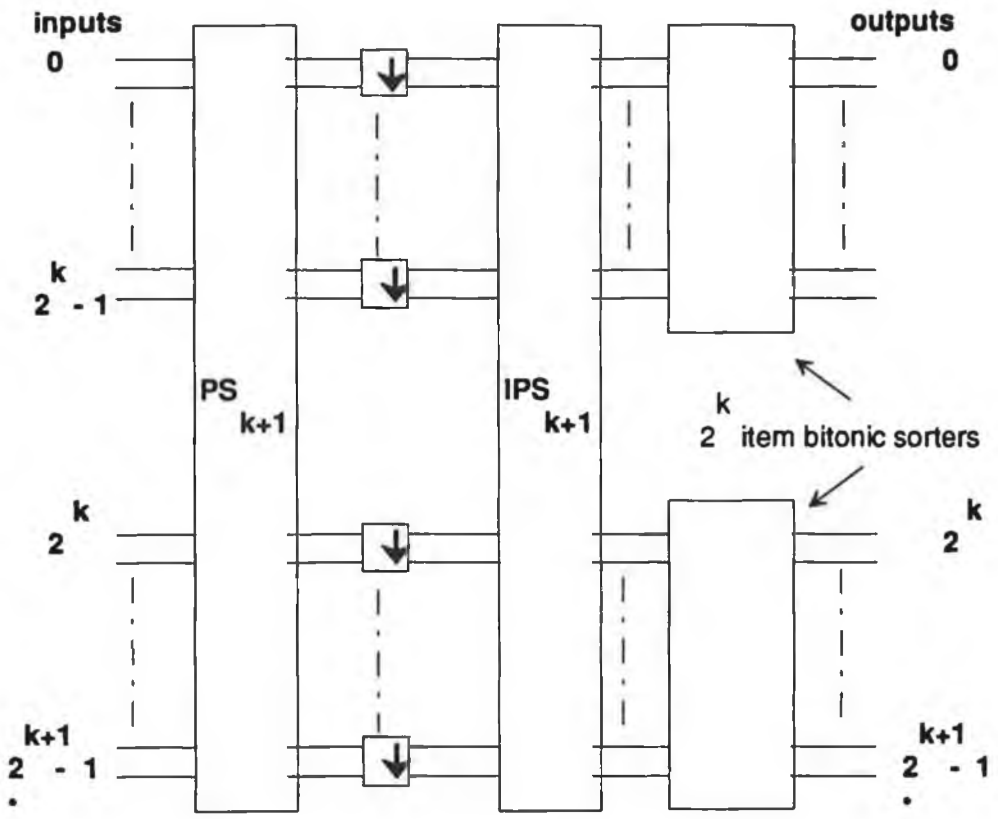


Fig. 2.15: Recursive rule for bitonic sorters.

Turner incorporated a broadcasting facility into his switch [24]. Other modifications included the use of double-buffered rather than single-buffered switch elements, and the use of eight bit wide data paths between switch elements. Another feature was the use of 'link groups', where any one of a set (group) of output lines could be used to route a packet to a given destination. This feature is also termed 'trunk grouping', 'channel grouping' and 'dilation' in the literature.

Both switch fabrics proposed by Turner feature an 8:1 increase in bandwidth on the links internal to the switch, compared to the input and output link bandwidths. This is achieved by speedup (i.e. a higher bit rate) in the first design, and by parallelism in the second. Thus the load placed upon the switch is low, never exceeding 12.5%. This restriction on the offered load is necessary because the banyan is a highly blocking network. Hence internal contention (contention for use of links interconnecting stages internal to the banyan) occurs with high probability, even for relatively light loads.

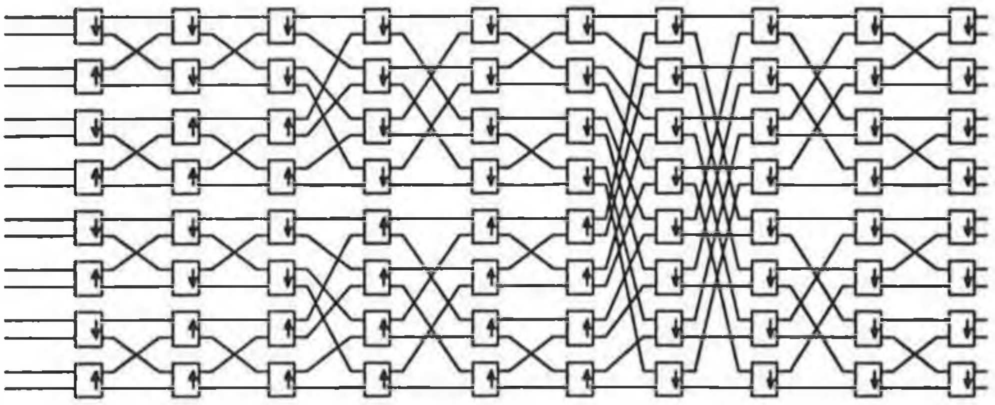


Fig. 2.16: A Batcher sorter.

The high bit rate of ATM means that link speedup cannot be used in a banyan network intended for use as an ATM switch. The alternative of using parallelism and/or a dilated network is not viable, because of the difficulties this creates with switch layout, given the limited pinouts of integrated circuits. This makes banyan networks suitable for the design of only small ATM switches. Most banyan-based ATM switches extend the basic banyan topology in such a way as to increase the number of routes available from source to destination through the switch. The two basic strategies employed are to provide multiple paths through the banyan network, by an appropriate modification of its topology, or to provide multiple banyans between the input and output sides of the switch, together with a means for distributing incoming cells among the banyans.

2.5.2 Switches with multiple paths.

A number of strategies has been proposed for increasing the throughput of banyan-based switch fabrics. Most of these are based on the principle of increasing the number of paths by which a packet may be routed from source to destination.

Perhaps the most obvious method of increasing the number of paths is to incorporate a distribution network, of the type suggested in [6], which routes packets from a given input port to an arbitrary input line of the banyan. This creates problems with the preservation of cell sequence on a virtual circuit. Anido and Seeto [25] considered a distribution network where the routing through the distribution network is not random - packets from the same virtual circuit are routed to the same output of the distribution network. This guarantees the preservation of cell sequence. They considered a number of possible algorithms for path selection at call set-up time.

A banyan using what were called feedback loops was described by Uematsu and Watanabe [26]. Essentially a banyan of double the size normally required is used, as shown in Fig. 2.17. Half of the available inputs are connected to half of the available outputs to form the feedback loops. Thus additional paths from an input to a given output are available, since if the direct path (called the banyan route) is blocked, a path via a feedback loop (a feedback route) may be taken. This necessitates the cell traversing the banyan twice, and so the delay on a feedback route is longer than on a banyan route. Hence, in order to preserve cell sequence, it is necessary to ensure that cells from the same virtual circuit take the same route through the banyan. This implies that a path selection algorithm must be employed, as with the switch of Anido and Seeto [25]. The algorithm proposed attempts to limit the flow of traffic on any link on a banyan route to a given level. If the use of a proposed banyan route would cause this limit to be exceeded, a feedback route must be selected instead. The algorithm attempts to select the feedback route which minimises the flow on each link, but is suboptimal. Uematsu and Watanabe pointed out the similarity between their network and the Benes network, which suggests that the network is re-arrangeable, and should provide high throughput, given a sufficiently sophisticated control algorithm. An obvious problem with implementation of the switch is the speed of execution of the algorithm. It is intended for use as a digital cross-connect system for virtual paths (pre-allocated bundles of virtual circuits [27]) and thus the pattern of interconnection requests will change more slowly than if individual virtual circuits were being switched.

The use of call-level routing, as described above, should be avoided where possible in an ATM switch, as opposed to a cross-connect system, since it increases the computational overhead of call admission control procedures. An algorithm of the type proposed above, which considers the traffic on links between every stage of the switch, would impose a particularly high overhead. The use of cell-level routing, whereby routing is performed independently for each cell, is to be preferred, when it is practicable. Switches based on the banyan network, but featuring multiple paths and cell-level routing, are described below.

The load-sharing banyan was proposed by Lea [28]. He pointed out that the nodes in a given stage of a banyan can be divided into *groups*, where all the nodes in a group have the same routing capability. Thus, if packets are transferred from one node in a group to another, they will still be routed correctly. Pairs of nodes in the same group share their input traffic in the load-sharing banyan. The pairing is chosen in such a way as to maximise the number of alternative routes available. Each node has four inputs and two outputs. If the output link requested of one node is busy, the data can be routed through the other node in the pair. A topology transformation is used to ensure that paired nodes are physically adjacent, so as to simplify the implementation. The load-sharing banyan thus provides an additional path for every cell between stages, reducing the probability of blocking. Lea showed that the throughput of an unbuffered load-sharing banyan was about 50% for a uniform load.

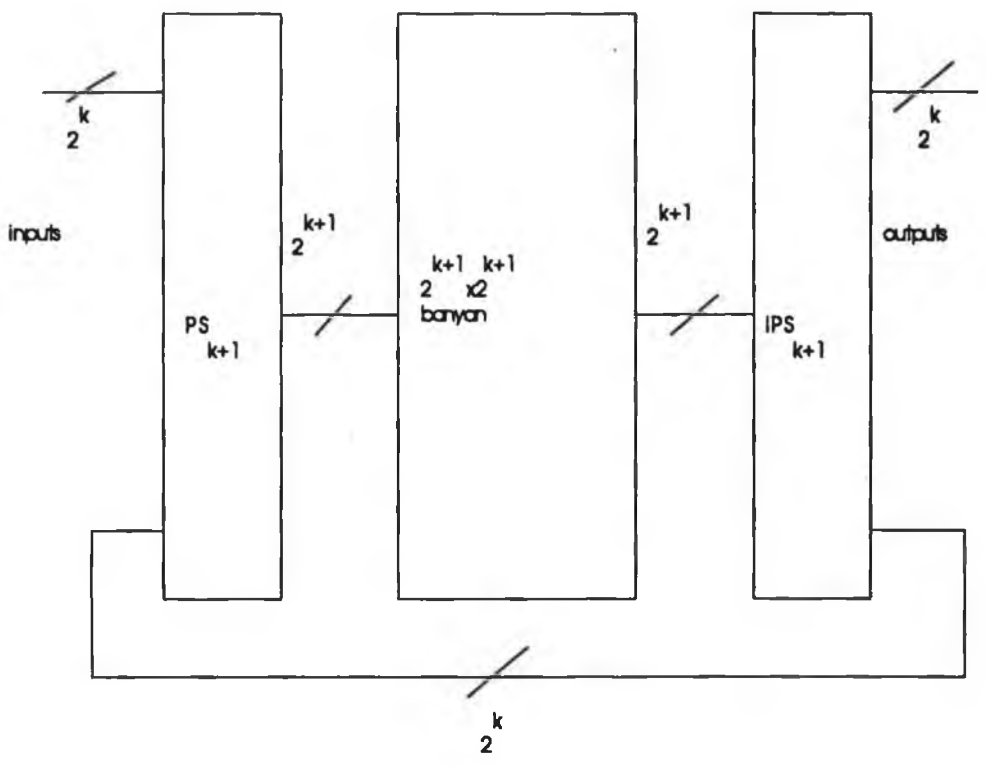


Fig. 2.17: A banyan with feedback loops.

A development of the load-sharing banyan is the load-sharing switch of Kim and Leon-Garcia [29,30]. This is a conventional S-banyan, but with distributor stages interspersed between switching stages. The function of the distributor stage is to distribute the cells arriving to a given group evenly among all the switch elements in the group. The distributor has the topology of the indirect binary n-cube [9] which was referred to as the reverse banyan by Kim and Leon-Garcia. A set of running sum adders at the inputs to the distributor generate the distributor output address to which each cell is to be routed. This ensures that the number of cells in each output buffer of the distributor differs at most by one from output to output (hence the name). Kim's switch thus provides a multiplicity of paths to its destination to each cell. The number of paths available to a cell at the output side of a stage is obviously dependent on the number of remaining stages. This switch offers a further increase in throughput, compared with the load-sharing banyan, at the expense of a considerable increase in switch complexity. The throughput was shown to approach 100% as the amount of buffer space increased. The maximum size of switch which can be constructed will be limited by the operating speed required of the running sum adder networks.

Another strategy for providing multiple paths is the rerouting banyan network of Urushidani *et al.* [31]. This switch, with 2^n inputs, contains m stages, with $m>n$. The switch elements in the extra stages (i.e., in stages zero through m-n-1) contain two additional input and output links called bypass links, which forward cells directly to the corresponding switch element in the next stage. The switch is defined by the following link permutations:

$$LP_0=LP_m=ST$$

$$LP_k=S_{n-(m-k-1)\mathrm{mod}(n-1)}, \qquad 0<k<m.$$

Thus the pattern of interconnections repeats itself every n-1 stages. The operation of the rerouting banyan is based on the assertion of Urushidani *et al.* that any consecutive n stages of their network are equivalent to a banyan. If a cell's routing is disturbed by contention with other cells, and hence the cell is deflected from its correct path, its routing can restart from the next stage. If a cell has been successfully routed through n consecutive stages (and thus has reached its destination position), it is routed to the appropriate output line via the bypass links. A parameter called the remaining judgment time (initially set to n) is used to determine the number of stages through which the cell must be routed before reaching its destination. Urushidani *et al.* also described a so-called short-cut routing algorithm which can be applied if the switch elements of all stages in the switch feature bypass links. In this algorithm, the initial value of remaining judgment time is set to n-h, not n, where h is the number of contiguous bits which match in the source and destination addresses, starting with the least significant bit. This value is chosen because, after n-h stages, the cell is at the correct destination position, and may then be routed via the bypass links. This further increases the usage of bypass links and reduces the probability of blocking on conventional links.

The routing strategy used in the rerouting banyan is as follows. Suppose the remaining judgment time for a given cell is k. If k exceeds zero, routing is based on bit n-1-(m-s-1) mod (n-1) of the destination address, where s is the stage number.

However, if *k* equals zero, routing is performed on the basis of the value of the least significant bit of the destination address. Thus the routing algorithm of the switch is comparatively complex.

The throughput of the switch can approach 100%, if the number of switch stages is sufficiently large. Urushidani *et al.* showed that, assuming infinite buffers in each switch element, a switch with 64 inputs and 34 stages had a cell loss probability below 10^{-12} for a uniform load of 90%. The number of stages required to achieve this performance fell to 25 when the short-cut routing algorithm was used. However, the rate of cell loss increased considerably when finite buffers were used. Nevertheless, this appears to be a promising technique for ATM switching. The size of switch which may be implemented will, however, be limited by the complexity of the interconnection pattern between stages. Also, preservation of cell sequence cannot be guaranteed.

The folded shuffle switch described by Campoli and Pattavina [32] has similar properties. This switch has only n-1 stages, but the outputs of the last stage are connected to the inputs of the first stage, so that, as with the rerouting banyan, the pattern of interconnections repeats itself every n-1 stages. The interconnection pattern between stages is the perfect shuffle, so that the folded shuffle switch can be regarded as an Omega switch, where the first and last stages are coincident. An advantage of the folded shuffle over the rerouting banyan is that all stages are identical, so that cells may be introduced to the switch at any switch element, and not just at the switch elements in the input stage. Each switch element has three inputs and three outputs. The additional input is used to accept cells from an input port, which can deliver cells to any switch element on a given row of the switch. The additional output goes to the output port controller, which can accept cells from any switch element in a row. Thus, once a cell has been routed to the correct row of the switch, it can be sent directly to the appropriate output port, without having to pass through bypass links, as is the case with the rerouting banyan. A handshaking mechanism is used to ensure that cells are only forwarded when the receiving switch element has buffer space available.

The routing strategy used resembles the short-cut algorithm of [31]. However, the calculation of the remaining judgment time, and the order in which destination address bits are examined differ, because different link permutations are used between stages. In the rerouting banyan, the routing tag bit to be inspected by a switch element in a given stage depends only on the number of the stage, and is independent of the remaining judgment time (provided that the latter is non-zero), whereas, in the folded shuffle switch, the bit inspected is independent of the stage number (since all stages are identical) and depends only on the remaining judgment time. The complexity of the routing algorithm of the rerouting banyan in the case when the remaining judgment time is zero is avoided, since the folded shuffle contains 2^n switch elements per stage, for a switch with 2^n inputs. Hence, when a cell has been routed to a switch element in the correct row of the switch, it has reached its destination, each switch element being associated with only one output port, whereas, in the rerouting banyan, the least significant bit of the destination address must then be inspected, to see which of two bypass links should receive the cell. A variant of this switch design was described in [33].

The folded shuffle switch has higher throughput than the rerouting banyan, for a given number of stages, because cells can be submitted to switch elements at any stage of the network. However, the switch elements are more complex than in the latter switch, and twice as many are required per stage. Hence, the performances of the two switches seem broadly comparable, for a given hardware complexity. The folded shuffle switch requires complex wiring, since each switch element must be connected to an input port and output port, in addition to four other switch elements. This complexity may make large switches impractical.

Tobagi [34] described a switch based upon multiple banyans in tandem. When a conflict occurs, the cell losing contention is marked as such, and is routed via the 'wrong' link of the switch element, i.e., the switch uses deflection routing. It subsequently will lose contention in later stages, since it has been marked as a contention loser. At the output, the contention winners are stored in output buffers, and losers are submitted to another banyan. This process is repeated for K banyans in tandem. Since the load on each successive banyan is reduced, a sufficiently large value of K can ensure low cell loss probability.

It is apparent that, although the tandem banyan switch resembles the rerouting banyan, a larger number of stages will be required to obtain a given level of performance, since deflected cells cannot be rerouted until they reach the next banyan in tandem. The results presented in [34] indicate that a 64-input switch requires 48 stages to achieve a cell loss probability below 10^{-6}, for a uniform 100% load. This is considerably in excess of the corresponding figure for the rerouting banyan.

2.5.3 Switches with multiple banyans.

Other authors have sought to improve upon the performance of the banyan by using multiple banyan networks in parallel. Bernabei *et al.* described a family of networks they called the parallel delta networks [35]. These networks consist of L delta networks (i.e. banyans) of size N x N preceded by an additional stage of switching. The overall switch size is N x N, because of the use of N 1xL expanders and N Lx1 concentrators. The authors showed that the resulting network is re-arrangeable (for 2 x 2 switch elements) if $L \geq 2^{\lceil S/2 \rceil - 1}$ (where S is the number of stages in the delta network, i.e. $S=\log_2 N$) and presented a suggested routing algorithm.

The same research team extended this concept by allowing the delta networks to be preceded by $r > 1$ additional stages of switching, which they called distribution stages [36,37], to produce the generalised parallel delta network (GPDN) of Fig. 2.18. The parallel delta network is thus the GPDN in the special case where $r = 1$. The condition for rearrangeability with this new network is that the number of delta networks be $L \geq 2^{\lceil (S-r-1)/2 \rceil}$. This result is consistent with that for parallel delta networks, if $r = 1$. The extension of this approach to multicasting was considered in [38].

Similar research has been carried out by Lea [39]. He considered the 'vertical stacking' of self-routing networks, and so his network is identical to a parallel delta network, but for the absence of a distribution stage. In fact, his network is equivalent to a GPDN with r =0. He shows that the condition for rearrangeability is $L \geq 2^{\lfloor S/2 \rfloor}$, i.e. $L \geq 2^{\lceil (S-1)/2 \rceil}$. This result matches that of [36] with r=0. Lea also presented a routing algorithm for his network, and additionally showed that the condition for a non-blocking network is :

$$L \geq \tfrac{3}{2}.2^{s/2}-1,\ S \text{ even.}$$
$$L \geq \sqrt{2}.2^{s/2}-1,\ S \text{ odd.}$$

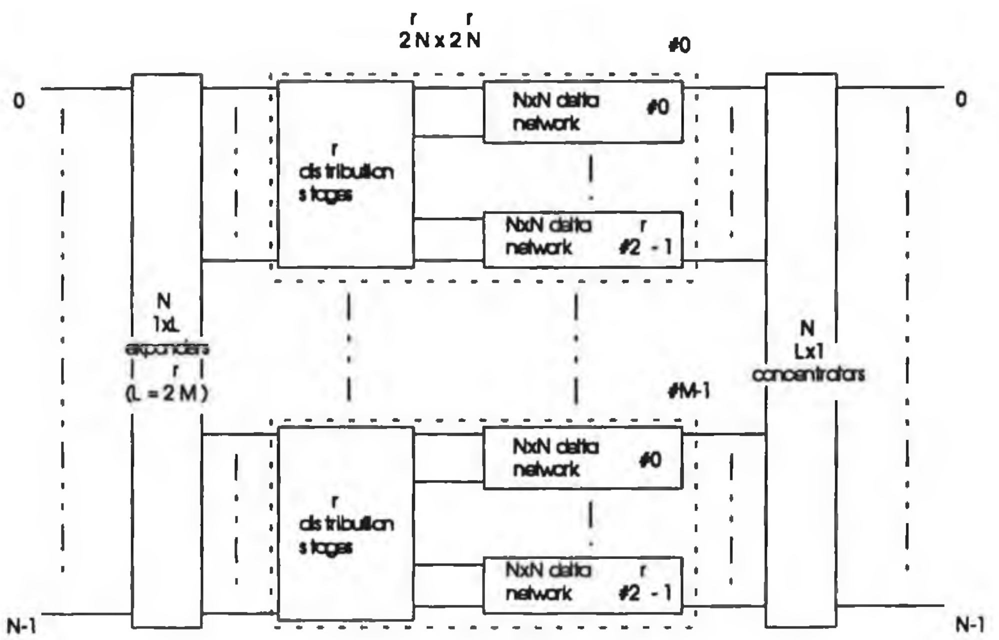


Fig. 2.18: The generalised parallel delta network.

Thus, for example, for a 256 x 256 switch, 23 delta networks are required to create a non-blocking switch, but only 16 delta networks are required for a rearrangeable switch. Lea compared the method of vertical stacking to the method of horizontal decomposition in [40]. Horizontal decomposition is a technique for recursively constructing a rearrangeably nonblocking switch. The resulting network has the same number of switch elements per stage as a banyan, but a larger number of stages. Such networks have similarities to networks consisting of the cascade of banyan networks, e.g. that of [25]. Lea claimed that a new network, combining vertical stacking and horizontal decomposition in one structure, and described in [40], allows a compromise to be made among conflicting design requirements, such as number of switch elements, fault tolerance and frequency of rearrangements.

Rearrangeable networks of the type described above cannot be used in conjunction with cell-level routing for obvious reasons. Lea's switch, even when dimensioned to be non-blocking, cannot be used with cell-level routing, because the delta networks are blocking networks. The non-blocking condition means that a free path can always be found from an idle input of the switch to an idle output. However, a control algorithm is still required to determine which of the possible paths available to a cell is a free path.

An alternative to routing multiple cells simultaneously to parallel banyans, which requires a control algorithm to determine which cells are routed to a given banyan, is to have a number of switch planes, and to attempt to route cells via each plane consecutively. This results in suboptimal routing, but requires no complex control hardware. Such a strategy was considered by Sarkies [41], who employed a bypass queue, as previously considered by Newman [42] and Bubenik and Turner [43]. This is a queue wherein, if the cell at the head of the queue is blocked from reaching its destination, other cells in the queue are offered to the switch. The use of multiple switch planes and bypass queuing can result in cells belonging to a given virtual circuit arriving out of sequence. Sarkies' solution was to ensure that each switch plane could attempt to route only one cell to a given output in a given time slot. Thus, if one cell belonging to a virtual circuit is rejected by the switch plane, then no further cells from that virtual circuit can be routed through the switch plane in that time slot (thereby arriving out of sequence).

The switch planes must be able to return a status signal indicating success or failure in delivering a cell to the output. This can be implemented in a similar way to the back-pressure mechanism used in Turner's switch [6]. Each switch plane accepts cells at its inputs during a certain period within a time slot (called a packet slot by Sarkie) referred to as an offer slot, sufficiently long for the status signal to return from the switch plane outputs. The packet slots for different planes are staggered in time, so that switch planes accept cells at different times. Each switch plane may have more than one offer slot, so that bypass queuing may be attempted. Cells which are rejected by a given switch plane are passed to the next in line. Cells rejected by the last switch plane are offered to the first switch plane in the next packet slot, ahead of new cells from the input buffer. The switch plane ordering is different from the viewpoint of each input port, so that traffic may be evenly distributed among the switch planes. The simulation results presented by Sarkies indicate that the throughput remains below 70%, even with a large number of switch planes, with a uniform 100% load.

2.5.4 Concluding remarks concerning banyan-based switches.

Banyan networks were among the first networks proposed for use as fast packet switches. The basic banyan switch is unsuitable for use in ATM because of its single-path routing feature, which results in a high probability of internal contention, and low throughput, even with a uniform load. Many of the switch designs described above introduce multiple paths from source to destination, but use call-level routing, which is likely to result in relatively low utilisation of link bandwidth. Two promising switch designs are the rerouting banyan [31], and the banyan with interstage distributors [29]. These switches offer throughputs close to 100%, with cell-level routing, and hence, should feature a low probability of cell loss.

2.6 Switch architectures based on sort-banyan networks.

2.6.1 The Starlite switch.

The banyan network is a blocking network. However, blocking need not occur if the destinations of the input packets conform to a certain pattern. This is the principle of the Starlite switch [5] of Huang and Knauer. They stated that blocking will not occur in an omega network [8] if the inputs to the network are sorted in ascending order of destination address, and if there are no repeated addresses. No proof of this assertion was provided, but proofs have been published subsequently by Lee [44] and Narasimha [45] for the S-banyan (described simply as the 'banyan' by both Lee and Narasimha). A new proof is provided in Chapter Four.

A difficulty arises when not all of the input ports have valid data present, i.e., if not all the inputs are active. This will typically be the case for packet switching networks. The switch elements must be capable of distinguishing between active and inactive inputs, so that the (spurious) destination addresses of inactive inputs may be ignored. This is achieved by inserting an 'activity bit' in the destination address of each packet. This modification ensures that active inputs cannot be blocked by inactive packets.

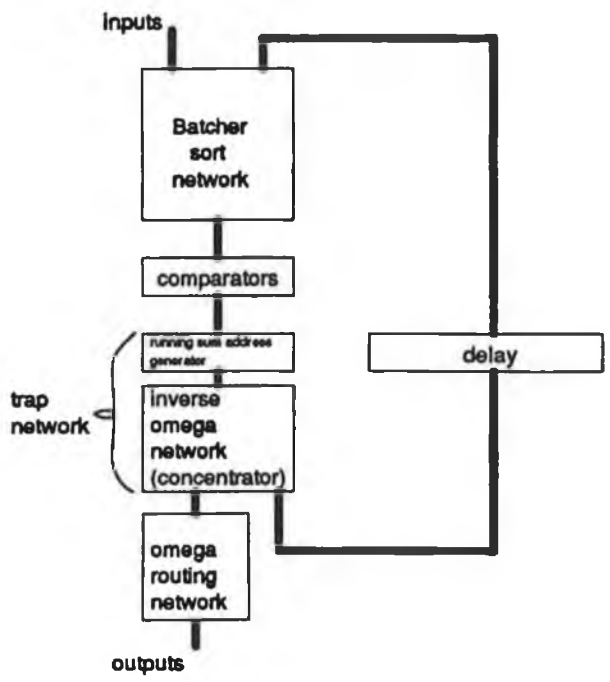


Fig. 2.19: The Starlite switch.

The asynchronous nature of arrivals to a packet switch means that two or more packets may arrive simultaneously intended for the same destination. The resulting contention for the same output port is inevitable in a packet switching system. Contention can also occur internally in the switch, if it is a blocking switch. The class of banyan switches described in section 2.5, for example, are blocking switches. The Starlite switch eliminates internal contention, despite employing an omega network for routing, by exploiting the property of the omega network, that a sorted list of input packets can be routed without contention. The packets which arrive at the omega network inputs must be sorted by destination address, and contain no repeated addresses, in order to eliminate both internal and output contention. The method used to achieve this is illustrated in Fig. 2.19.

The Batcher sort network sorts the inputs by destination address. The comparators flag packets with repeated addresses, but not the first packet with that address. The address generator (used to determine routing through the concentrator) calculates the running sum of the flags representing either unrepeated or inactive addresses. This results in packets winning contention being routed to the left, inactive packets to the middle and packets losing contention to the right of the concentrator. An inverse omega network is used as the concentrator. The inverse network is used, because the running sums are generated least significant bit first, whereas the conventional omega network requires the most significant bit first. Contention winners are routed without internal blocking through the omega routing network. Packets losing contention are discarded, or, optionally, may be re-admitted (via the recirculators) to the switch inputs during the next time slot.

The Starlite switch with recirculation may be modelled as a shared-buffer switch. Methods of modelling such a switch will be discussed in Chapter Three. The Starlite switch has the disadvantage, compared with other shared-buffer switches, that packets are buffered at the inputs of a sort network. Hence, the incremental cost of additional buffers is very high.

2.6.2 Switching with sorters.

A method of using Batcher networks to implement the routing function was described in [46]. The principle of the method is shown in Fig. 2.20. The control packets each contain a *token*, and a distinct address in the range 0:*N*-1. At the sorter output, the input and control packets are sorted by address. The control packet is always sorted to the top of the list of packets with a given address. The arbiter compares each control packet to the packet below it in the list. If it is a data packet (and thus has the same address) the token is transferred to the data packet and the control packet is destroyed; if the packet below it is also a control packet (and so no data packet with the same address is present) the control packet retains its token. The Batcher routing network routes the packets with tokens to its upper *N* outputs. The set of packets with tokens includes one (and only one) packet with each possible address in the range 0:*N*-1, and thus sorting these packets by address is equivalent to implementing the routing function.

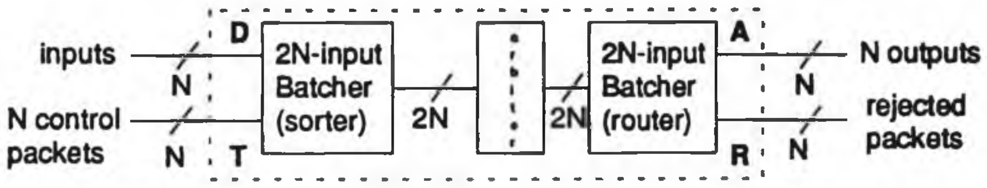


Fig. 2.20: Using tokens to perform routing.

A disadvantage of the above scheme is that the packet loss would be quite severe, since cells losing contention are discarded. The method of [46] is more complex, and uses the multiply re-entrant Batcher network of Fig. 2.21. Control packets are inserted at the lowest level. Their tokens are seized by data packets with the same address (if any such are present at the lowest level). Packets with tokens ascend to the next highest level, where their tokens can be seized by data packets of higher priority, with the same address. Packets without tokens, or which have lost their tokens, descend to the next lowest level, having their priority incremented. This ensures that the oldest packets have the highest priority, which is necessary to preserve packet sequence. Packets rejected at level zero are lost from the system. The number of packets lost depends on the number of levels, which affects the amount of storage available for packets without tokens.

Day *et al.* [46] also described how logical and physical addresses may be separated. The sorter operates on logical addresses, not physical addresses. Each control packet token contains the physical address corresponding to its logical address. The arbiter replaces the address of each packet possessing a token with the physical address. Thus the Batcher which implements routing sends packets winning contention to the appropriate physical address.

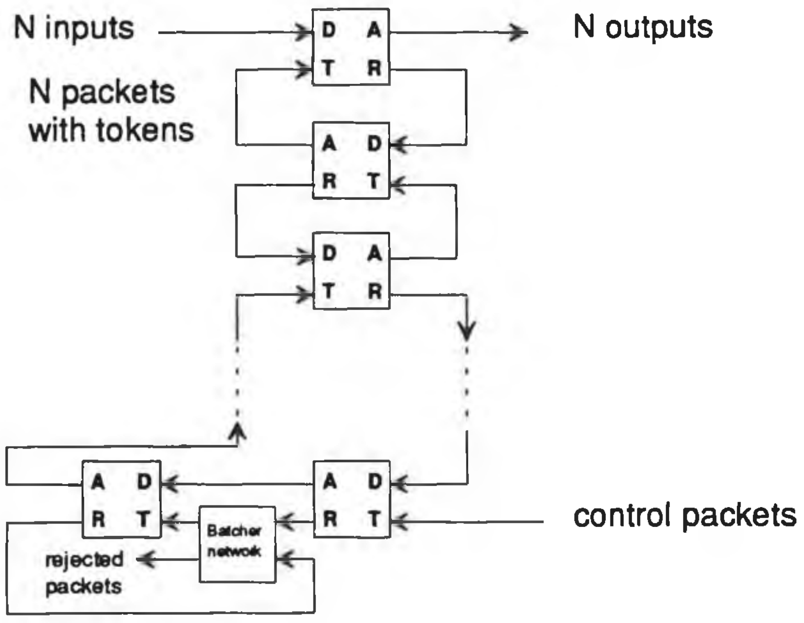


Fig. 2.21: The multiply re-entrant switch.

They also claimed that this technique permits the use of multiple ports to reach the same logical address (i.e., channel grouping), but gave no details of the modifications necessary to their architecture to implement this. These changes would be trivial if the group size were constant and fixed for all groups. Otherwise a difficulty arises because the sorter will produce a list containing all control packets with a given logical address, followed by all data packets with the same address. Allowing the number of control packets per destination to vary means that no simple way of associating control and data packets will be possible. No results were given for the performance of the switch in [46]. However, its complexity, evident in Fig. 2.21, makes it unsuitable for use in ATM.

2.6.3 Input-buffered switches.

A number of switch proposals use the non-blocking property of the Batcher-banyan network in conjunction with a method for the resolution of output contention to implement an input-buffered switch.

One such switch has been described by Hui [47,48]. This consists of a single Batcher and a single banyan, in conjunction with a three-phase algorithm which allows the same hardware to be used for both contention resolution and routing (Fig. 2.22). This algorithm allows the concentration stage of the Starlite switch to be eliminated, since this function is performed by the Batcher network. The three phases are as follows:

1) Contention Resolution: Each active input submits a request packet to the Batcher network, containing a source address and a destination address field. These packets are sorted by destination address at the Batcher network output. The arbiter grants permission for the corresponding data to be transmitted if the request packet destination address differs from that on the adjacent Batcher output line. Thus at most one packet per output line will win contention in this phase.

2) Acknowledgement: An acknowledgement packet, containing the original source address, is sent to each input port winning contention, as follows. The acknowledgement packets are submitted to the Batcher network by the controller. They are sorted by source address at the Batcher outputs, and are then routed through the banyan network to the original source ports. This occurs without contention since the addresses of the packets at the banyan inputs are distinct, and the packets are sorted by address.

3) Transmission: Input ports which received an acknowledgement in the second phase submit their data packets in this phase. These packets are routed without contention to the appropriate output line, since at most one data packet per output is transmitted.

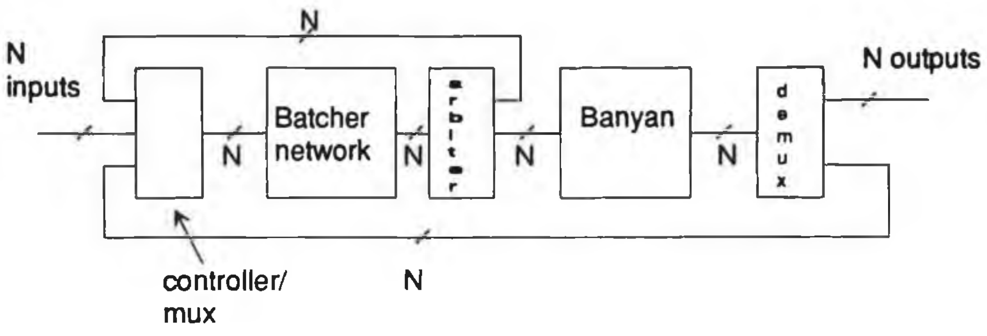


Fig. 2.22: The three-phase Batcher/banyan switch.

Packets losing contention are queued in an input buffer until the next time slot. The reduction in the amount of hardware required in Hui's switch, compared to the Starlite switch, is offset by the higher speed of operation required. The overhead imposed by the first two phases of the three phase algorithm means that switch must operate at a higher rate internally than the input and output line rate.

An alternative method of contention resolution was described by Bingham and Bussey [49]. This uses dedicated hardware in the form of a special network with a ring topology, on which each node is associated with a unique input port of the Batcher-banyan switch. A token is placed on the ring for each output port at the start of each time slot. These tokens are circulated around the ring, and are seized by the first input port (node on the ring) which has a packet at the head of its buffer with a destination address matching the address of the token. Since only one token per output port exists, at most one input port can seize a token for a given output port in any one time slot. Input ports seizing tokens submit the corresponding packets to the Batcher-banyan network in the next time slot, whereupon the packets are routed without contention to the appropriate output ports.

The Batcher-banyan network does not require a hardware speed-up, as with the three-phase switch. However, every packet is delayed by at least one time slot while the contention resolution algorithm operates. The ring must operate at a speed sufficiently high that the tokens visit each node on the ring within one time slot. Thus the speed of operation required of the ring increases linearly with the number of input ports. Fortunately, the hardware implementing the ring is quite simple. A single bit represents the presence or absence of a token. The token address is inferred by each node from an internal clock. This structure requires just five bits (including clocking information) to be sent from node to node on the ring. These bits are transmitted in parallel on a five wire ring. Bingham and Bussey pointed out that tokens may be seized by packets in the input buffers, other than those at the head of the buffer. This allows the first-come-first-served queuing discipline to be modified, which can improve throughput.

The Nemawashi switch described in [50] is similar, in that it uses a Batcher-banyan network for routing, and dedicated hardware for contention resolution, with packets winning contention being routed through the Batcher-banyan network in the succeeding time slot. Contention resolution is performed using the method of the first phase of the three-phase algorithm, i.e. using a Batcher network plus arbiter. The method of sending acknowledgements is conceptually simpler, but requires more complex hardware. The Batcher network sets up a bidirectional link between the input ports and the arbiter. Thus the acknowledgement may simply be sent down the return path to the input port which generated the request. This

Batcher network is distinct from that used for routing, allowing routing (of packets which won contention in the previous time slot) and arbitration to proceed simultaneously. The number of arbitrations performed per time slot may be greater than one, so that more than one packet per input line may attempt to win contention within a single time slot. This increases the probability that a packet will win contention for each output line, and so increases throughput. Packets which have won contention on previous iterations of the algorithm re-submit their requests during the current iteration, with a priority bit set which ensures that they will once again win contention (because the sorter places the higher priority packet at the top of the list of requests for a given output port).

A buffer subsystem was suggested for use in Batcher-banyan switches to reduce the amount of input buffering required [51]. This system is illustrated in Fig. 2.23. Cells from the input lines of the switch enter the input buffers via a sorting network. Cells from the input buffers are routed to the input ports of the Batcher-banyan switch via another sorting network. The first sorting network performs four consecutive sorts. The first sort generates a concentrated list of active input lines. The second sort generates a list of input buffer addresses sorted by occupancy. The third sort assigns an input buffer to each active input. The fourth sort routes input cells to their assigned buffers. The net effect is to route cells to the input buffers with the lowest occupancies.

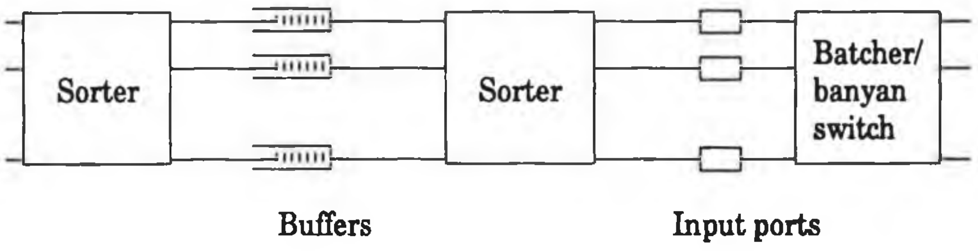


Fig. 2.23: A buffer subsystem for a Batcher-banyan switch.

The second sorting network also performs four sorts, whose net effect is to ensure that only idle input ports of the Batcher-banyan switch receive cells from the input buffers, and that those cells come from the buffers with the highest occupancies. Thus the buffers are effectively shared among all the input lines. This reduces the amount of buffer capacity required in the system, and ensures that a non-uniform loading of the switch inputs has no effect on its performance.

A feature of switches with input queuing and an FCFS queuing discipline is that switch throughput is limited to about 58%, as discussed in Chapter Three, by the phenomenon of head-of-line blocking. This problem can be reduced, and throughput thus increased, by scheduling cells in such a way as to avoid this type of blocking. Suitable schemes have been described in [52,53] which can increase the throughput of the switch to about 90% and 95% respectively. However, the complexity of the hardware required to implement these schemes would probably be excessive, and would certainly exceed the complexity of the switch itself. Simpler schemes, such as the multiple arbitrations used in [50], offer lower increases in throughput over an FCFS queuing discipline, but with much lower circuit complexity.

2.6.4 Multiple servers and channel grouping.

A limitation of the above networks is that at most one request per output port can be satisfied in a single time slot. The switch throughput can be increased if more cells can be forwarded to each output port in one time slot. The resulting switch contains queues at both the input and output side of the switch.

The Sunshine architecture [54,55] uses multiple banyans and output queuing. The structure of the Sunshine switch is shown in Fig. 2.24. It represents the extension of the Starlite architecture to allow more than one cell to be delivered to each output port in one time slot. The first Batcher sorts its input cells by destination address, and where cells have the same address, by priority. There are two priority fields, static (Quality Of Service) and dynamic (which increments for every time slot the cell is present in the system). The dynamic priority ensures that cells from a given virtual circuit (which will have identical quality of service or QOS fields) are delivered to the output in FCFS order. The trap network is identical to that of Starlite, with the exception that the cell on output line *i* of the Batcher wins contention if the cell on output line *i-k* has a different address, not if the cell on line *i*-1 has a different address. This ensures that as many as *k* cells per destination can win arbitration in one time slot.

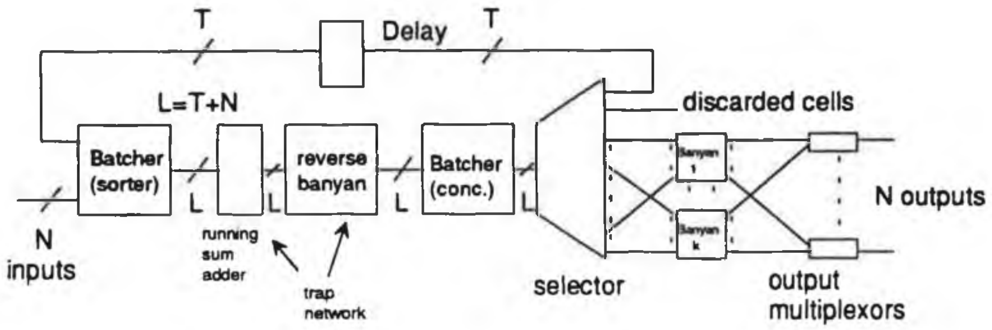


Fig. 2.24: The Sunshine switch.

Cells losing contention are sorted to the top of the concentrator in order of priority, not destination address, by interchanging address and priority fields. The *T* cells of highest priority which lost contention are re-circulated (with an incremented value of dynamic priority). If more cells lose contention they are discarded. The concentrator sorts cells winning contention, by address, to the lower outputs. These enter the *k* banyans, which route them without contention to the output ports. The output ports require buffers to store excess cells, since only one cell may depart per time slot. The interconnection pattern at the inputs and outputs of the banyans is the *k*-shuffle, as defined by Patel [12]. This ensures that no banyan receives more than one cell per output, as required for contention-free operation.

Hui described, in [56], a method for implementing channel grouping, where the size of each channel group is *k*. The method is simply to connect the output lines of the *k* banyans in Fig. 2.24 to individual output ports, rather than to concentrate the outputs at a single output port. This method seems simpler than that described in [46].

A more flexible, albeit more complex, approach to channel grouping was described by Pattavina [57]. His method allows channel groups of varying sizes to coexist in a single switch. The switch uses both logical and physical addresses for each output. The logical addresses of the output lines in a group are contiguous. This need not be the case for the physical addresses. Each input port is required to maintain two tables, one giving the physical addresses of the output lines (called channels by Pattavina) indexed by logical address, and the other giving the number of channels per group. The switch resembles that of Hui [47] and similarly incorporates a three-phase algorithm. It differs in including a channel allocation network, as shown in Fig. 2.25, in place of the arbiter of Fig. 2.22. The switch operation is as follows. During the arbitration phase, the request packets contain the logical address of the first channel in the requested group (the group leader) and are sorted by this address, before being submitted to the channel allocation network. This consists of a running sum adder network, which calculates, for each request packet, how many requests for the same group precede it in the sorted list. The input to the adder network is the output of a set of comparators similar to those used in the arbiter of [47]. A low comparator output resets the running sum, and a high output increments the sum, to calculate the required values.

The offset calculated by the channel allocation network is sent, as the acknowledgement packet, to the input port which generated the request. If the offset matches or exceeds the number of channels in the group, then the packet loses arbitration. Otherwise, the physical address to which the data is to be sent is looked up in the appropriate table, using the offset, added to the logical address of the group leader, as the index. The third phase of the algorithm is identical to that described in [47]. One problem with the switch is that FCFS delivery of cells belonging to a given virtual circuit is no longer guaranteed, since the delays experienced on different channels in a group may be unequal. This disadvantage is shared with all channel grouped architectures.

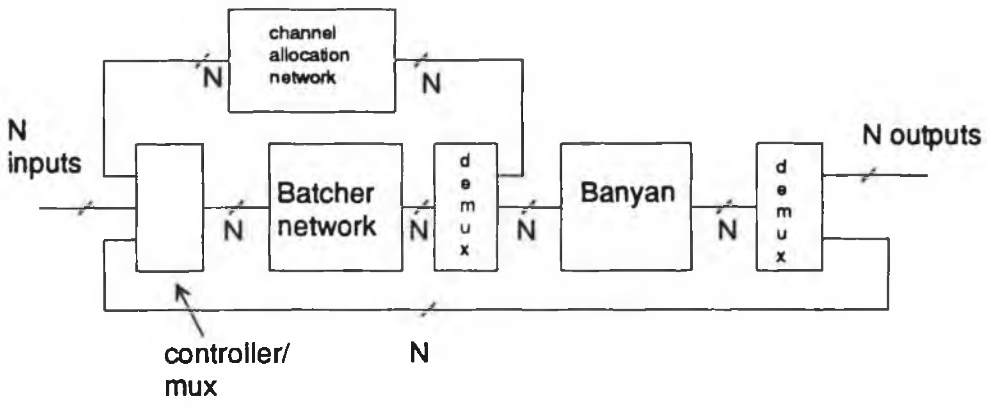


Fig. 2.25: A switch with channel allocation.

Pattavina described a number of improvements to this switch in [58]. The Batcher is preceded by a request rotation network which alters the order in which data is applied to the Batcher network. This increases the level of fairness of the contention resolution process. Another improvement is to re-order the channels within a group when calculating physical addresses, so as to distribute traffic evenly among the channels in the group. The first two phases of the three-phase algorithm are performed by dedicated hardware in this design, so that no internal speedup of the switch is required. Pattavina also describes an algorithm which, given two priority levels for traffic, and under certain assumptions concerning the sizing of channel groups, can ensure an upper bound on the delay experienced by high priority traffic, so as to guarantee packet sequence integrity. This algorithm essentially emulates circuit-switched routing for high priority data.

A similar switch, which does not incorporate channel grouping, was described in [59]. This is intended for use in central offices, i.e. where the switch ports are connected to end users, with one channel per user. The architecture of [57,58] is modified by multiplexing the channels in each channel group at the output ports to create a switch with output queuing. Hence the switch can route more than one cell to each output in a single time slot. The switch incorporates a mechanism to ensure that the output buffers never overflow. The routing algorithm has two phases, the probe phase and the data phase. Contention resolution is implemented during the probe phase. A cell loses contention if the number of cells, with the same destination, and which have already won contention, exceeds either the amount of buffer space available at that output port, or the number of routes available to that output. Cells winning contention are uniquely assigned one of the routes to that destination. Cells are transmitted to the output during the data phase.

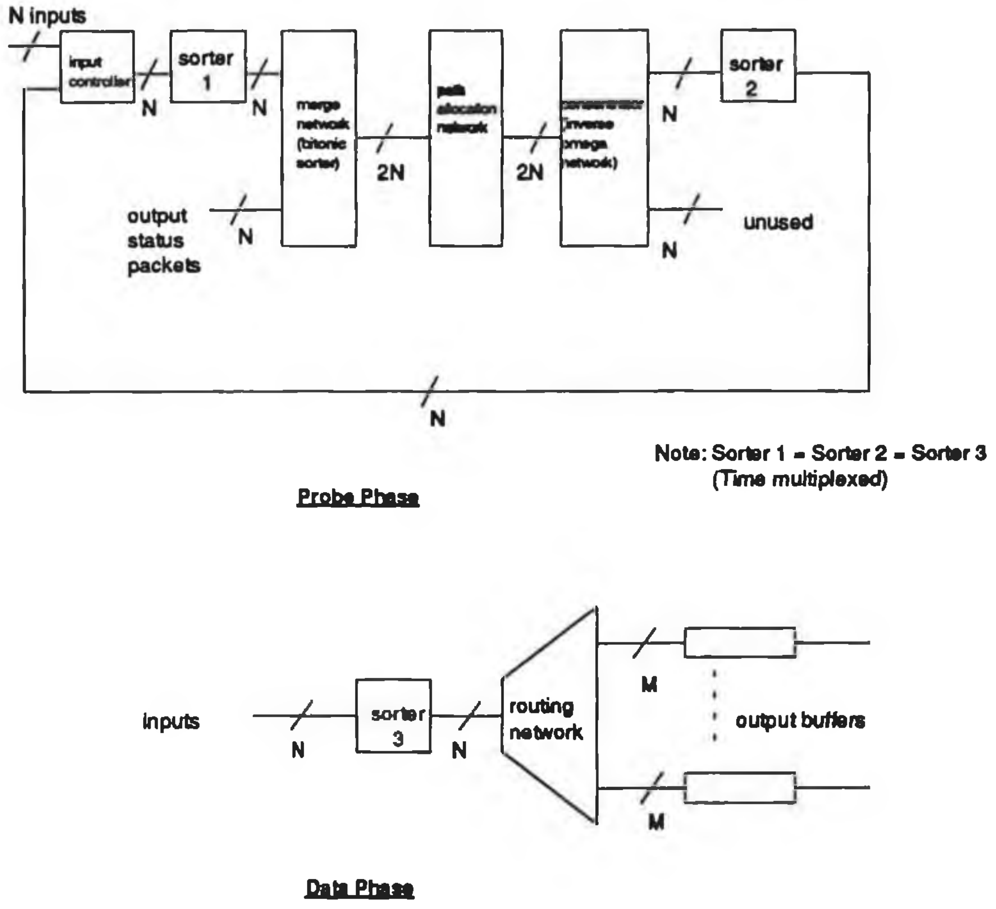


Fig. 2.26: A switch with zero loss in the output queues.

A diagram of this switch is shown in Fig. 2.26. Each output port submits a status packet detailing the buffer space available during the probe phase. These packets are merged with the request packets from the input ports and a path allocation network assigns a routing index to each request packet, which identifies which of the possible routes to the destination is assigned to that request. An invalid routing request is assigned to requests losing contention. The resulting acknowledgement information is then routed back to the requesting input port. The structure of the path allocation network very closely resembles that of the channel allocation network of [57], since the algorithm it implements is similar. The benefit of this switch design, that the output buffers can never overflow, is obtained at the cost of a considerable increase in the complexity of the switch.

2.6.5 Scaling difficulties in sort-banyan switches.

Sort-banyan based switch architectures require a large number of switch elements. Hui's switch [47] uses the least hardware, needing just a single Batcher and a single banyan. The banyan requires n stages of switching, for a 2^nx 2^n switch [12]. The Batcher requires $^1/_2n(n+1)$ stages [13]. Thus the total number of switch elements required is $2^{n-2}.(n-3).n$. A very large switch of this type is clearly impractical. For example, an 8192 x 8192 switch requires 91 stages of switching. In addition, the switch dimensions cannot readily be increased after the switch is constructed. Lee addressed this problem in describing how a large switch may be constructed from smaller Batcher and banyan modules [60].

The key component in this design is a non-square (M x KM) switching module comprising an M x M Batcher, K 1 x M expanders and K M x M banyan networks. This module, shown in Fig. 2.27(a), is non-blocking. The value of K is a power of two so that the 1:K networks are simple binary trees. The structure of the overall switch is shown in Fig. 2.27(b). There are S routes to each output port from each input module. The contention resolution algorithm is such that at most S cells from each input module, requesting the same output, win contention. Each contention winner is given a unique address, that of one of the S routes to the requested destination. The combination of the expander and the banyans allow contention winners to reach the required output port, using self-routing. The overall switch is clearly non-blocking if the input modules are non-blocking. The input modules have this property because the additional bits added to the routing tag to determine which banyan is used for routing are most significant bits. Thus the Batcher network will group cells destined for a given banyan on contiguous output lines. The k-shuffle interconnection pattern between the expander stage and the banyan inputs ensures that the sorted order of these cells is preserved at the banyan inputs, and that all inactive inputs are grouped at the bottom of the list. Thus the inputs to each banyan are sorted and concentrated, as required to ensure contention-free routing to the banyan outputs.

Lee suggested an extended version of the ring reservation algorithm [49], with modifications to cater for priority and multiple routes to each output, for use in contention resolution. The scaling difficulties associated with the ring reservation algorithm do not apply, because it is applied only to the set of input ports of one input module, not of the whole switch.

The number of outputs can be expanded N-fold by replacing the output multiplexors by $SLxN$ switches. The number of inputs can be expanded P-fold by mutiplexing the outputs from *P* such switches. This is the basis of the three-stage architecture of Liew and Lu [61].

The combination of asymmetric switch modules (with more outputs than inputs) and channel grouping in these switches results in a greater throughput than is the case with a conventional (square) input-buffered switch. This issue will be discussed further in Chapter Three.

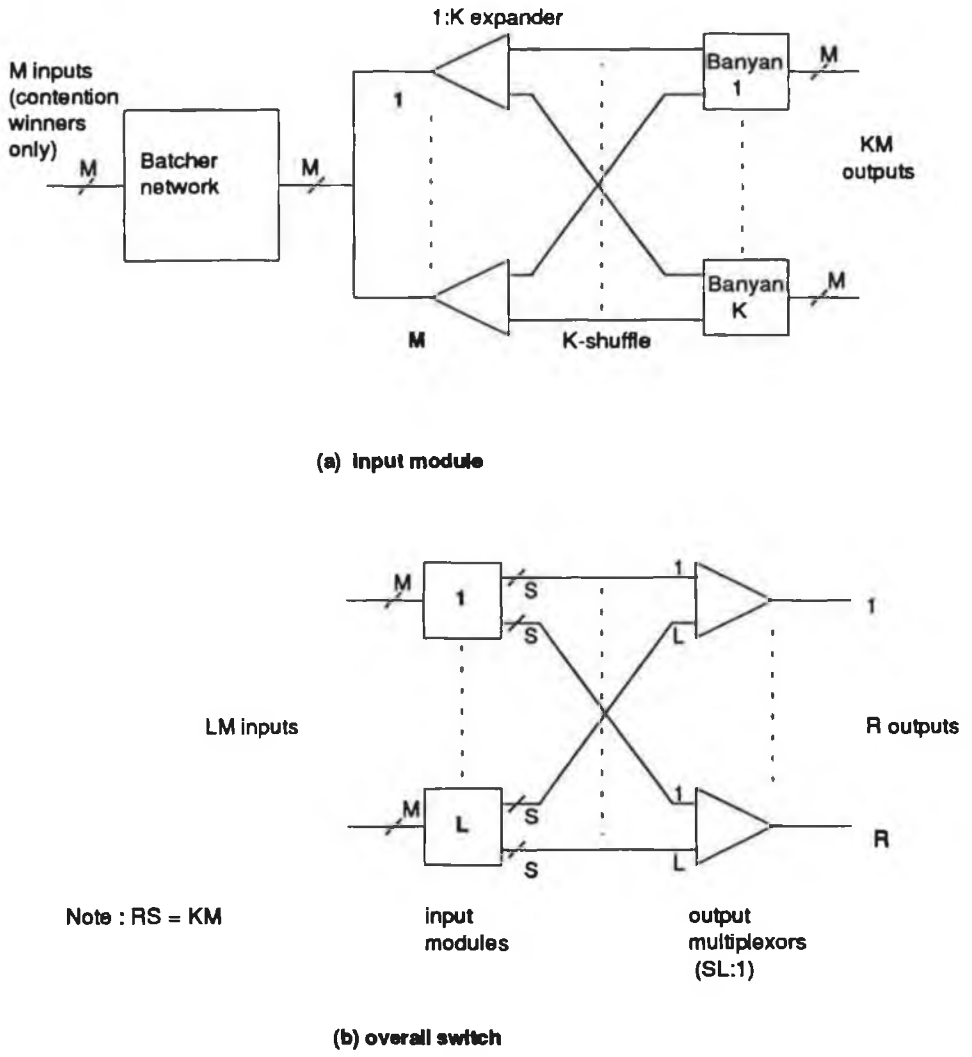


Fig. 2.27: A large switch constructed from sort-banyan modules.

2.6.6 Conclusions regarding sort-banyan switches.

Sort-banyan switches are ideally suited to applications where an internally non-blocking switch is required. However, output contention adversely affects their throughput when used in ATM switch architectures, unless sophisticated buffer management or cell-scheduling techniques are used. A switch with a large number of inputs and outputs, with acceptable throughput, will be very complex, if implemented using sort-banyan techniques. However, Batcher-banyan switches may be used as modules in larger two-stage switches. The use of sort-banyan modules in a three-stage switch will be considered in Chapter Five.

2.7 Memory-based and shared-medium switch architectures.

Memory-based switches seek to exploit the advances made in the very large scale integration of random-access memory. The technology used for switching is comparable to that used for time slot interchangers in contemporary circuit switching [62]. The earliest proposal for such a switch was the Prelude switch [63] for which a prototype was constructed [64]. The Prelude project was based on asynchronous time division techniques, and thus the switch, although not constructed to conform with ATM standards concerning cell size, etc., is based upon similar principles. The basis of its operation is illustrated in Fig. 2.28.

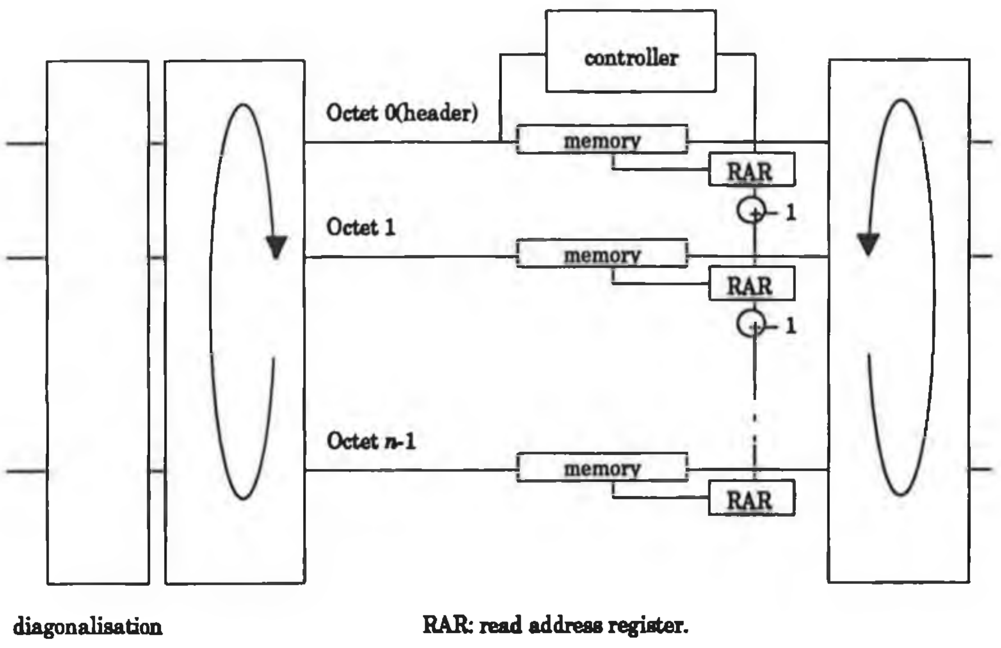


Fig. 2.28: The Prelude switch.

Data paths in the Prelude switch are an octet wide, to reduce speed requirements. Packets are fragmented into octets, and the fragments are stored in separate buffer memories. This is achieved by a process of diagonalisation which ensures that there is an octet stagger between the time slot boundaries on each input of the switch. A rotator then routes all the corresponding octets from each incoming packet to a single output. In particular, all the cell headers are stored in one memory. A control memory maintains queues, one for each output, giving the location in the memory of headers of cells to be routed to that output. These queues are interrogated sequentially to find the read address for the header of a cell destined for each output in turn. The cell fragments are then read out sequentially (header first) from the buffer memories, onto the appropriate output line. This is ensured by the read address registers. The uppermost register receives the address of the header of the appropriate packet from the control memory. Each read address register increments its contents before forwarding them to the next register. Hence, consecutive octets of the packet are read from consecutive addresses in adjacent memories. The second rotator routes these consecutive octets to the correct output port.

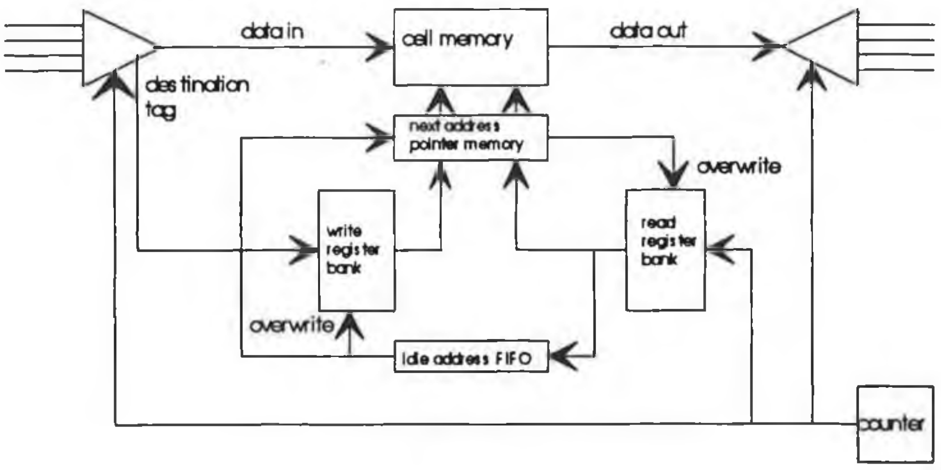


Fig. 2.29: A shared-buffer switch.

The complexity of the Prelude switch arises from the requirement to introduce parallelism into the hardware, so as to reduce circuit speed requirements. If the mechanisms whereby parallelism is introduced are ignored, then the operation of the switch can be described very simply. Cells are read into buffer memory from each input line in turn, and the storage address of each cell is entered into the appropriate control memory queue. The cells are read out non-sequentially, in an order determined by the contents of the control memory queues.

The method used to achieve parallelism in the Prelude switch imposes constraints on the size of switch which may be constructed. In particular, the number of switch inputs and outputs must equal the number of packet fragments, so that switches of different sizes will require substantially different hardware implementations.

Another memory switch is that described by Kuwahara *et al.* [65]. This stores the cells destined for a particular output in a linked list in dynamic memory. This is a shared buffer architecture, because the linked lists occupy the same memory space. The linked list operation is implemented using the hardware of Fig. 2.29.

Incoming calls are multiplexed cyclically onto the data input line. The destination tag is used to look up the next available address for cells to be routed to the requested output in the write register bank (which has one register per output port). Simultaneously, an idle address is extracted from the FIFO queue. The input data is written to the location given by the write register, whilst the idle address is written to the same location in the pointer memory, and also overwrites the write register contents.

The read registers for each output are interrogated in turn. The cell stored in the address given by the register is written to the output port via the demultiplexor and the corresponding pointer is stored in the register. The old register contents are entered in the idle address FIFO. Multiple priorities may be implemented using multiple banks of read and write registers. The number of cells stored in memory destined for a given output may be limited by associating an up/down counter with each pair of read and write registers, to count the number of cells being buffered. New arrivals are rejected if the counter value exceeds a threshold.

The switch design is evidently unsuited to the construction of large switches. The example switch described in [65] has 32 inputs and 32 outputs. The design of a three-stage switch using this shared buffer switch as a switch module was described in [66]. A condition for the number of intermediate stage switches required to achieve non-blocking with ATM traffic was presented, but no proof was given. A simple link selection algorithm was proposed to be used during call set-up, although its implementation was not discussed.

A switch architecture proposed as part of the RACE programme (project 1012: broadband local network technology) was described in [67]. It is based upon a multi-stage switch using a memory switch (the SIGMA switch) as a building block. This uses shared memory for cell storage, and output queues for routing tags, and closely resembles that of [65].

Fried [68] described a similar architecture which uses commercial RAM in conjunction with special memory controller ICs and a custom switching processor. The processor is programmable, allowing the switch to be adapted to changing ATM standards.

Koinuma *et al.* described a similar switch [69]. Their basic module is a 4 x 4 switch. It uses shared buffers to store cells and FIFO queues (one per output) to store routing information, as with [68], and maintains a queue of idle buffers for cell storage, as with [65]. One difference is that the switch features what is called dynamic link speed control. Each output line is capable of operating at four times the incoming line rate (i.e. at 620 Mb/s) using ECL technology. The total throughput of the switch is constrained to be 620 Mb/s by the bottleneck of the shared memory access. However, this can be allocated as 155 Mb/s per output line, or 620 Mb/s to a single line, or as some other combination, as appropriate. A sophisticated control algorithm would be necessary to utilise this feature in a multi-stage switch, but no details of such an algorithm are given.

Henrion *et al.* [70] described another switch architecture using a shared buffer switch as a switching element. The proposed switch element size is 32 x 32, and the structure resembles [65]. The cells are dived into smaller units for routing, to relax speed requirements by introducing parallelism. The multi-stage architecture appears to offer multiple path self-routing via a back-pressure mechanism. This crude mechanism would produce low throughput, so multiple planes of switching are used to raise throughput to an acceptable level.

The ATOM switch [71] is a memory switch using an output buffered architecture rather than shared memory. It is intended for implementation using a bit-sliced architecture. The principle of its operation is shown in Fig. 2.30. The time division bus is required to operate at n times the line rate, where n is the number of inputs. This requires the use of a parallel architecture to be achieved in practice. Each input places its cell on the bus in turn. The address filter at each output blocks all cells except those with that output as destination. Cells are converted back into serial form for transmission at the output port. The switch clearly cannot be used for large numbers of inputs. The authors suggest a 32 x 32 switch as being feasible. This basic element is used to construct a three-stage switch architecture. Route selection is performed at circuit set-up time.

A number of proposals exist for switches using a ring architecture. An early such proposal is the switch of Suzuki *et al.* [72] which uses multiple rings. Access to the rings is based on the multiple-token ring algorithm. An ATM ring was described in [73] which uses fibre optic links between nodes, and a slotted ring. Methods to ensure low delay and priority control were described. Rahnima [74,75] also described a slotted ring, focusing on the design of the hardware at each switch node. He considered [75] the modular construction of larger switches using a folded two-stage architecture. No algorithm for route selection in this larger switch was suggested.

Gard and Rooth [76] proposed a switch architecture based on a folded Clos structure. The 32 x 32 switch modules of which it is composed comprise a 32 bit wide ring, each line in the ring operating at 155 Mb/s. Thus the ring bandwidth equals the maximum load on the switch, so that no blocking can occur.

Ohtsaki *et al.* [77] proposed using multiple ring switches to implement channel groups. Cells not winning access to the requested output port on one ring overflow onto another. Corresponding outputs of each ring would be connected in a channel group.

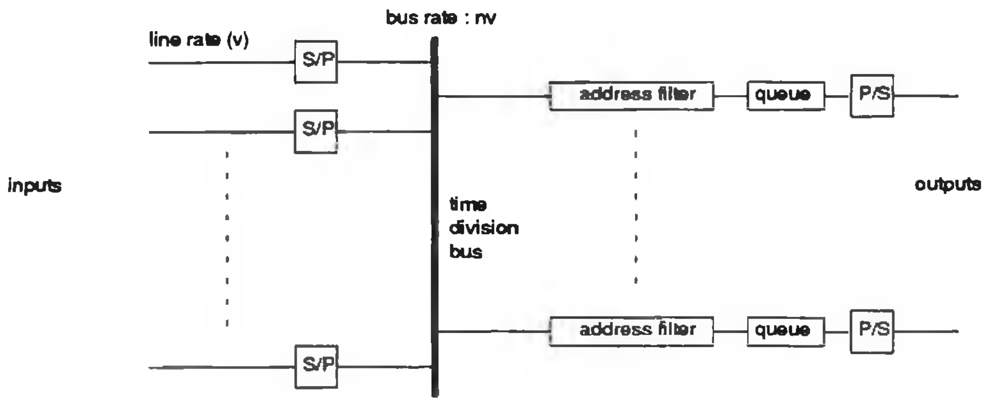


Fig. 2.30: The ATOM switch.

2.8 Other output-buffered switch architectures.

The knockout switch [78] of Fig. 2.31 can be regarded as the packet switch equivalent of the crossbar switch in circuit switching, because it is fully connected, i.e., there is a separate path from each input to each output. Each input port broadcasts its cell to all output modules. Each output module features an address filter at its input lines which passes only cells to be routed to that output. The concentrator has N inputs (where N is the number of input lines to the switch) and L outputs. If more than L cells arrive simultaneously for that output, the excess cells are discarded ('knocked out'). The probability of this occurring is low for independent arrivals at the switch inputs, if L is large. The authors claimed that L=12 is sufficient to ensure that the probability of cell loss is below 10^{-10} for arbitrarily large N. The concentrator is constructed by the appropriate interconnection of two-input, two-output modules, which determine the 'winning' and 'losing' cells in a 'tournament' to determine which cells reach the concentrator outputs. The barrel shifter ensures that the surviving cells are distributed evenly among the queues. The queues and multiplexor operate an algorithm which ensures that cells are delivered to the output line in FCFS order, and that the queue sizes never differ by more than one.

The knockout switch is not suited to large switch sizes, because of its fully connected architecture, so that its complexity grows rapidly with increasing switch size.

A number of variants on the original knockout switch design have been proposed. The SCOQ (shared concentration, output queuing) switch [79] simplifies the design of the concentrator by sorting incoming cells prior to broadcasting them to the concentrators. A non-blocking concentrator can then be implemented using a banyan network. This network has more than one output, so that the concentration function for a number of outputs is implemented by a shared network. This switch can be seen to be an elegant hybrid of the Batcher-banyan and knockout switch techniques. Another technique for buffer sharing in a knockout switch is described in [80].

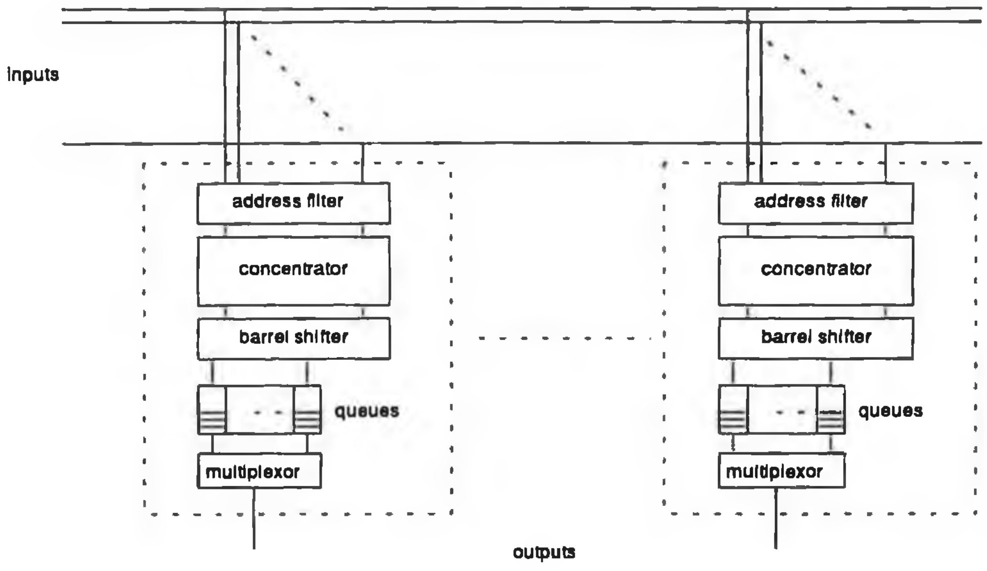


Fig. 2.31: The knockout switch.

Chao [81] described a distributed knockout switch, where the concentration function is distributed among switch elements positioned at the crosspoints between the N input lines of the switch and the L input lines of each output buffer. He also suggested that the output buffers be replaced by distribution networks with more than one output, reducing the value of L required to achieve a given value for cell loss probability. The distribution networks thus implement the shared concentration function described in [79].

Another output-buffered architecture was described by Ahmadi *et al.* [82]. It is based upon the use of an expansion stage (input module) and a multiplexor stage (output module), as shown in Fig. 2.32. The expansion stage is similar to a self-routing banyan, but each switch element has only a single input, giving rise to a tree-like structure. The expansion stage is non-blocking, since expansion is performed independently for each input.

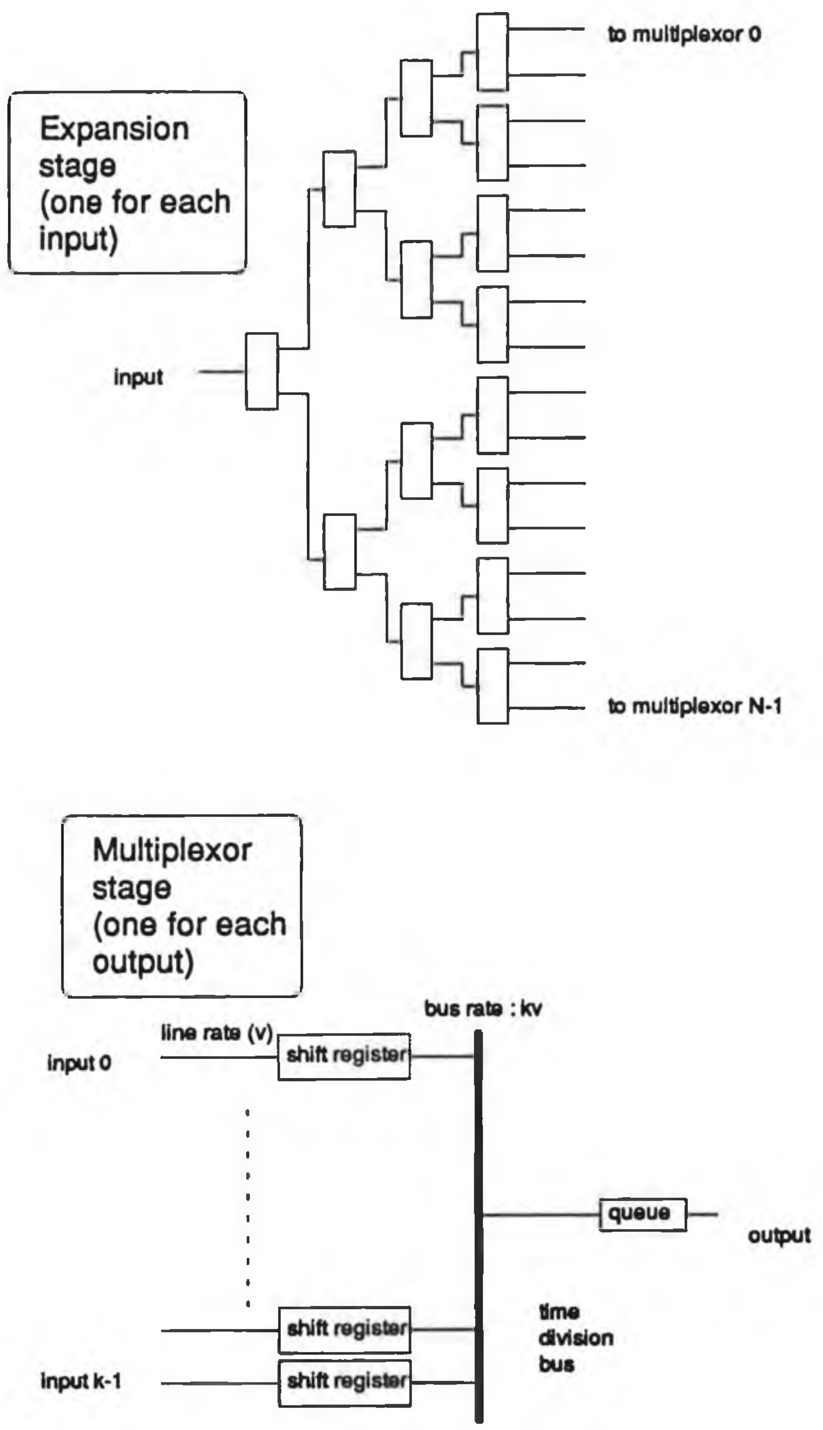


Fig. 2.32: A switch with expansion and multiplexing stages.

The multiplexor accepts cells from the appropriate expander output of each of k input modules. These cells are transmitted to the output queue within one time slot by using a bandwidth of k times the line rate (achieved via parallelism). The unit of information is a 'minipacket' (a fragment of a cell) rather than a full cell, to relax bandwidth requirements. Thus, this switch is similar to the ATOM switch

[71], but the function implemented by the address filters in the ATOM switch is here implemented by the expander stage.

The Christmas-tree switch [83] results if the multiplexor is implemented using 2x1 concentrators, using a topology which is the inverse of that of the expansion stage. The complexity of the resulting switch can be reduced, at the expense of introducing cell loss, by employing concentrators in the expansion stage, so that an *N*-input switch does not require *N*-input multiplexors.

Another similar switch is the M/1 switch element described in [84]. This switch, shown in Fig. 2.33, works as follows. The multiplexor output operates at a higher bit rate than the incoming line rate. The suggested speedup factor (v) is three. The m queues are necessary because at most v cells can be passed through the multiplexor in one time slot. Additional logic is necessary to control the multiplexor function so as to ensure FCFS delivery of cells.

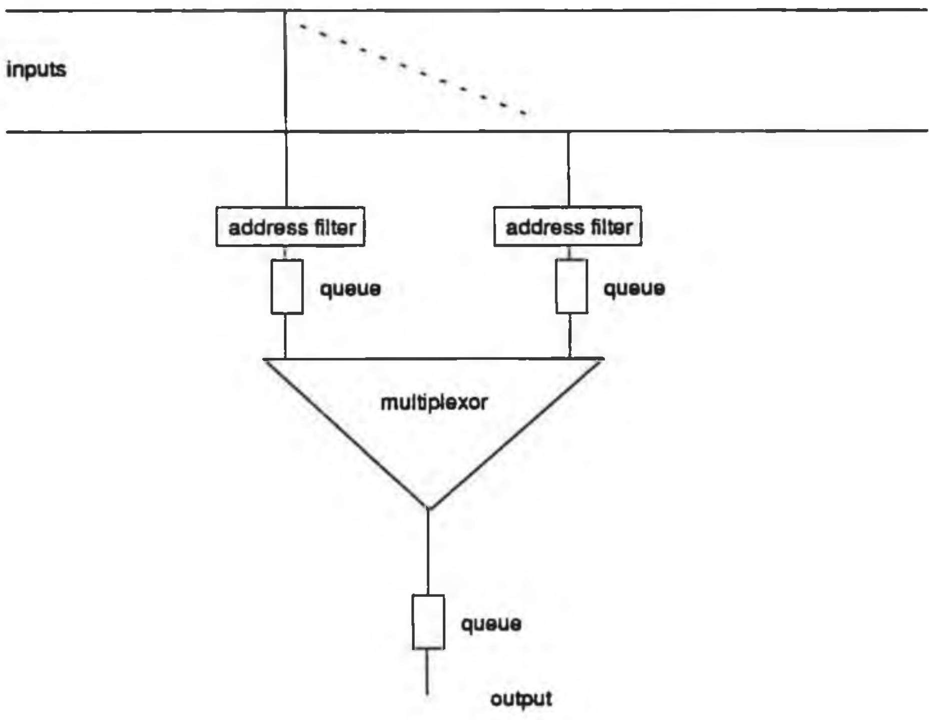


Fig. 2.33: The M/1 switch.

Another output-buffered switch is the Cylinder switch [85]. This resembles the knockout switch, but the multiplexing of cells onto a single output line is performed using a ring structure. Cells are stored on the ring until they win contention to the output line.

2.9 Remarks concerning output-buffered switches.

Output-buffered switches are the design of choice when a small ATM switch is required. They offer 100% throughput and, in the case of shared-buffer memory switches, relatively low buffer capacity requirements. Shared-medium switches such as memory switches and ring switches can readily be designed, for small switch sizes, for zero knockout loss, making them comparatively insensitive to nonuniform loading of the switch outputs.

However, as the switch size increases, the bandwidth required of the shared medium becomes excessive. The complexity of other output-buffered switches, such as the knockout switch, increases rapidly, at a rate proportional to N^r, where $2 < r \leq 3$, for an N x N switch, the value of r depending on the amount of concentration performed on the cells for a given destination, prior to arrival at the output buffer. Hence, these switches also become impractical as the switch size increases.

The maximum size of switch which may be constructed in practice will depend on the fabrication technology used, as well as on the switch design. Switches with hundreds or thousands of inputs will be required for broadband ISDN networks. A modular switch design will be required to allow output-buffered switching techniques to be used in a large switch.

2.10 Multi-stage switches.

Large switches may be constructed by the interconnection of multiple stages of switching, each stage being constructed from a number of smaller switch modules. The application of this technique to cross-point reduction in circuit switching was described in the classical papers by Benes [86] and Clos [87]. Consider the three-stage N x M switch shown in Fig. 2.34. The interconnection pattern between the input stage and the intermediate stage of switching is the r_1-shuffle, whilst the interconnection pattern between the intermediate and output stages is the m-shuffle. This network is often called *the* Clos network, although Clos' work did not refer to a specific switch structure. Often the network is assumed to be symmetric, in which case $N=M$, $n_1 = n_2$, and $r_1 = r_2$. The condition which results in a strictly non-blocking circuit switch is $m \geq n_1 + n_2 - 1$ [87], while the condition for a symmetric, rearrangeable circuit switch is $m \geq n$ [86]. The extension of the non-blocking criterion to packet-switched networks has been considered by a number of authors [66,88-90].

Many authors have proposed the use of three-stage switches for ATM. Some proposals (e.g., [66,69,76,82]) use call-level routing. This ensures that the sequence of cells on a virtual circuit is preserved as the cells pass through the switch, but requires the execution of a routing algorithm during call set-up which will typically be complex. The routing algorithm operates in conjunction with the call admission control process, which treats each switch module as an independent switch. Other designs use cell-level routing, so that the three-stage switch can be treated as a unit by the call admission control function. If routing is performed independently for each cell the hardware required for route selection is typically simple, but special mechanisms will be required to preserve cell sequence. More complex cell-level routing techniques can be used to avoid contention in the second stage.

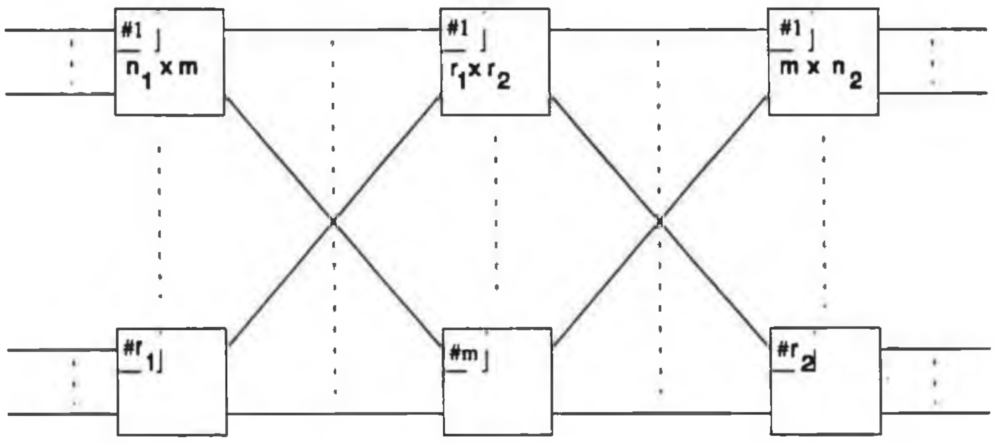


Fig. 2.34: A three-stage switch.

The multi-stage structure proposed in [70] time stamps each cell when it enters the switch, and buffers it at the output stage until its delay has reached a predetermined maximum value. Since all cells then experience the same delay, apart from an unavoidable variation due to multiple simultaneous arrivals (output contention), cell sequence on a virtual circuit is preserved. This method has the obvious disadvantage that it requires synchronisation of the time-stamping mechanism across all the input modules. Route selection in the intermediate stage is apparently achieved by a random selection of one of the available paths. A back-pressure mechanism is used to flag unavailable paths.

The synchronous ATM switch of Proctor and Maddern [91] uses a deterministic routing strategy which ensures that the delay through the intermediate stage is constant for any given pair of input and output ports, although the delay varies from pair to pair. This is sufficient to preserve cell sequence, even though cells may be routed through different switch modules in the intermediate stage. Proctor and Maddern considered the example of a 256 x 256 switch, the principle of whose operation is described below.

Each input port sends a number of requests (two requests are suggested) to each of the intermediate switch modules in turn, and awaits an acknowledgement. Each intermediate switch module contains a buffer for each output, with space for just one cell. An acknowledgement is returned if the requested buffer is available. The requests are piggy-backed on cells which have won arbitration in the previous time-slot, and which are being sent to the relevant intermediate switch module. Acknowledgements for input port k are piggybacked on the cell being sent from the intermediate switch module to output k, from where they are sent by a dedicated line to input k.

The switch operation is periodic. Cells which won arbitration during the previous time-slot are sent to the appropriate intermediate switch module during the corresponding 'window' in the current time-slot. Cells are requested from a given intermediate switch module by each output in turn during one time-slot. The cyclic way in which each intermediate switch module in turn requests cells from a given input port, and each intermediate switch module in turn delivers cells for a given output, ensures that the transit delay through the intermediate stage is constant for that pair of input and output ports.

The input and output modules are not capable of arbitrary connections, but instead are rotators, connecting input i to output $(i+k)\text{mod } N$, where N is the number of outputs of the switch module, and k is some integer. Only the intermediate switch modules are required to route arbitrary inputs to arbitrary outputs.

Proctor and Maddern suggested a practical implementation, wherein the bit rate is only 40 Mb/s, through the use of sixteen planes of switching. The operation of the planes, and of the rotators within a plane, is staggered in time so as to distribute flow control requests and acknowledgements more evenly. This, in conjunction with the use of sub-cells, rather than full-size cells, as the data unit in the switch, ensures that the fixed delays through the intermediate switch modules are kept low. The use of sixteen planes means that, even with a 100% load, less than half the bandwidth available through the intermediate stage is used. This would result in a strictly non-blocking circuit switch, and Proctor and Maddern claimed that only small input queues are needed when the switch is used for ATM. A similar switch was described by Beshai and Munter [92].

Eng. *et al.* [93] described a switch with similar properties, although with a very different implementation. They called their switch the Growable Packet Architecture. All the links operate at the full line rate. No buffering is provided in the input and intermediate switch modules. Cells which are not allocated a route to the requested output module at the first attempt are discarded. The route selection is performed by the following algorithm.

Each input module maintains a table of flags indicating the busy or free status of its output links. A similar table is maintained of input link flags for each output module. A cell can be routed via intermediate switch module k if the k-th output line of the input module is free, and if the k-th input line of the requested output module is also free. This condition can be checked by comparing the corresponding entries in the appropriate tables.

All flags are set to 'free' to initialise the algorithm. Each input module k generates its own table, and the table for output module k. At each iteration of the algorithm, each input module checks for available paths to the output module whose table of flags it possesses. If any path is available, and if the input module has a cell to route to that output module, then the flags for that path are set to 'busy'. If more than one path is available, a random choice is made among paths. The table of output module flags is then forwarded to the next input module, and the flag table of another output module is accepted from the previous input module, before commencing the next iteration. The number of iterations required is evidently equal to the number of output modules. This algorithm involves passing flags around a ring, and so resembles the ring reservation algorithm of [49] in that the speed of operation required grows linearly with the switch size.

The switch dimensions are determined using the so-called generalised knock-out principle. This principle is based on the observation that the probability of many cells simultaneously arriving at the switch inputs, requesting the same output, is low, assuming independent arrivals. The number of intermediate switch modules is chosen to keep the probability of more cells arriving, requesting the same output module, than the number of paths from the intermediate stage below a certain threshold (the knock-out loss). There is an additional loss due to the sub-optimal routing strategy, which does not allow re-arrangements of routes. This loss is called the scheduling loss. This loss is claimed to be negligibly low.

No implementation details were given for the cell scheduling algorithm. Interestingly, a more recent description of the growable packet architecture [94] abandoned the cell scheduling algorithm. Instead, the N inputs are sorted and then forwarded to m modified banyan networks, which route the cells to the appropriate (m-input) output module. This switch thus has more in common with the Sunshine switch [54] than with the switch of Fig. 2.34.

A different method for path allocation was considered by Cisneros [95]. His switch consists of eight planes of 1024 x 1024 switches, which receive cells from 1024 input ports, each receiving cells at eight times the normal cell rate, or receiving cells from eight lines at the normal rate. The switch also has 1024 output ports, each of which can also transmit cells at eight times the normal rate. Thus, the switch is equivalent to that shown in Fig. 2.34 where $r_1 = r_2 = 1024$, $m=8$ and

$n_1=n_2=8$. It can thus be regarded as an 8192 x 8192 switch. However, the switch dimensions are sufficient to make a circuit switch rearrangeable, but not strictly non-blocking, so that the throughput of the switch will be limited.

The key component in Cisneros' design is the contention resolution and path allocation mechanism. This works as follows. Within the duration of one ATM time-slot, the switch attempts to route the cell at the head of each input queue through the first switch plane. If this fails, an attempt to route through the second switch plane is made, and so on until an attempt has been made to route through all switch planes. The principle of this mechanism is similar to that of the multiple-plane switch of [41], but the implementation is different. In Cisneros' approach, each input port places a message on a ring with details of the requested destination, and, optionally, a priority. These messages are circulated around the ring and inspected by each input node. After each message has been circulated around the ring, each input port can determine whether or not any other head-of-line cell has precedence over its own, and thus whether its own cell should be routed through the switch plane concerned. This algorithm is repeated for each switch plane in turn.

A cell has precedence over contending cells if the following events occur:

1) no contending cell, on any input port, has a higher priority;

2) any other contending cells are of lower priority, or, if of equal priority, are on higher-numbered inputs of the switch.

The second condition gives precedence to inputs at one side of the switch. This is necessary so that the selection of one of the contending cells, among cells of equal priority, can be made unambiguously, but leads to unfairness in access to the outputs.

This problem is alleviated by the introduction of a crude form of dynamic priority, which has the advantage of being easily implemented, requiring only the maintenance of a single flag bit for each head-of-line cell. This dynamic priority bit is set if a cell loses priority to a contending cell of equal priority, but only if both cells had a clear dynamic priority bit. The effect of this is to group arbitration cycles for contending cells of equal priority into sessions. A session starts if none of the cells has its dynamic priority bit set. If there are n requests of equal priority at the start of the session, then each cell gets an opportunity to win arbitration once in the next n cycles of the algorithm (not counting cycles where cells of higher priority are contending for the same output). Cells of the same priority arriving during the session must await the end of the session before attempting to win contention. This mechanism is not completely fair since, during a session, cells are not served in order of arrival, but input port by input port.

The algorithm resembles the ring reservation algorithm [49], but differs in some respects. The data on the ring consist of tokens in the ring reservation algorithm, which are modified as they circulate around the ring, whereas the circulated data are never modified during a cycle of Cisneros' algorithm. The data circulated in the ring reservation algorithm describe availability of output ports, whereas the data circulated in Cisneros' algorithm describe the requests at the inputs. This difference could be important in the choice of an algorithm for non-square switches. The hardware required at each node of the ring in Cisneros' algorithm is more complex than that used in the ring reservation algorithm, but can process multiple priority levels in a single cycle. The ring reservation algorithm requires multiple cycles, one for each priority level [60], but with simpler hardware.

Another approach to the design of a multi-stage switch was taken by Newman [42]. He suggested a two-plane switch, where each plane is an unbuffered multi-stage switch. He considered the example of a 64 x 64 Benes switch constructed using 8 x 8 switch modules. A reverse path or back-pressure mechanism is used to flag collisions within the switch. Newman's approach differs significantly from that of other researchers in that arriving packets are allowed to enter the switch fabric asynchronously. In the event of a collision within the switch, the collision signal is asserted and returned over the reverse path to the input port controller. The input port controller then waits for a delay typically equal to one tenth of the packet duration before retrying.

Multiple paths are available to each destination within the switch fabric. Newman proposed two algorithms for route selection - searching and flooding. The searching mechanism is such that the input port controller retries over successive paths to the destination until it meets with success. In the flooding method, the incoming packet is broadcast over all possible paths to the destination, which selects one of the arriving copies. Colliding cells are discarded. Newman finally recommended a hybrid algorithm, which floods between planes but searches within a plane.

The switch proposed by Kim and Lee [96] similarly used switch elements with a large number of inputs and outputs. They suggested that knockout switches be used as switch elements, with a banyan interconnection pattern. Thus a 256x256 switch could be built using two stages of 16x16 knockout switches.

The design of a three-stage switch featuring cell-level routing will be considered further in Chapter Five.

References

[1] H. Ahmadi and W. Denzel, "A survey of modern high-performance switching techniques", *Journal Of Select. Areas Commun.*, vol. 7, no. 7, pp. 1091-1103, Sept. 1989.

[2] A. Pattavina, "Nonblocking architectures for ATM switching", *IEEE Commun. Mag.*, vol. 31, no. 2, pp. 38-49, Feb. 1993.

[3] G. Luderer and S. Knauer, "The evolution of space division packet switches", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. V, pp. 211-220.

[4] E. Zegura, "Architectures for ATM switching systems", *IEEE Commun. Mag.*, vol. 31, no. 2, pp. 28-37, Feb. 1993.

[5] A. Huang and S. Knauer, "Starlite: a wideband digital switch", *Globecom '84 Conference Record*, pp. 121-125, Nov. 1984

[6] J.S. Turner, "Design of an integrated services *packet* network", *Journal Of Select. Areas Commun.*, vol. 4, no. 8, pp. 1373-1379, Nov. 1986.

[7] C.-L. Wu and T.-Y. Feng, *Tutorial, interconnection networks for parallel and distributed processing*, IEEE Computer Society Press, 1984.

[8] D.H. Lawrie, "Access and alignment of data in an array processor", *IEEE Trans. Comput.*, vol. 24, no. 12, pp. 1145-1155, Dec. 1975.

[9] M.C. Pease, "The indirect binary *n*-cube microprocessor array", *IEEE Trans. Comput.*, vol. 26, no. 5, pp. 250-265, May 1977.

[10] C. Wu and T.-Y. Feng, "On a class of multistage interconnection networks", *IEEE Trans. Comput.*, vol. 29, no. 8, pp. 694-702, Aug. 1980.

[11] L.R. Goke and G.J. Lipovski, "Banyan networks for partitioning multiprocessor systems", *Proc. First Annual Symposium on Computer Architectures*, pp. 21-28, 1973.

[12] J.H. Patel, "Performance of processor-memory interconnections for multiprocessors", *IEEE Trans. Comput.*, vol. 30, no. 10, pp. 301-310, Apr. 1981.

[13] K.E. Batcher, "Sorting networks and their applications", *Proc. AFIPS 1968 Spring Joint Computer Conference*, pp. 307-314, 1968.

[14] H.S. Stone, "Parallel processing with the perfect shuffle", *IEEE Trans. Comput.*, vol. 20, no. 2, pp. 153-161, Feb. 1971.

[15] J. Jahns and M. Murdocca, "Crossover networks and their optical implementation", *Appl. Opt.*, vol. 25, no. 15, pp. 3155-3160, Aug. 1988

[16] K.E. Batcher, "The flip network in STARAN", *Proc. of the International Conference on Parallel Processing*, pp. 65-71, 1976.

[17] T. Feng, "Data manipulating functions in parallel processors and their implementations", *IEEE Trans. Comput.*, vol. 23, no. 3, pp. 309-318, Mar. 1974.

[18] D.E. Knuth, *The Art Of Computer Programming*, vol.2 : *Seminumerical Algorithms*, 2nd. ed., Addison-Wesley, 1973.

[19] J.J. Kulzer and W.A. Montgomery, "Statistical switching architectures for future services", *Proc. ISS '84*, pp. 43.1.1-43.1.6, May 1984.

[20] J.P. Coudreuse and M. Servel, "Asynchronous time-division techniques: an experimental packet network integrating videocommunications", *Proc. ISS '84*, pp. 32.2.1-32.2.7, May 1984.

[21] J.S. Turner and L.F. Wyatt, "A packet network architecture for integrated services", *Globecom '83 Conf. Rec.*, pp. 45-50.

[22] D.M. Dias and J.R. Jump, "packet switching interconnection networks for modular systems", *Computer*, vol. 14, no. 12, pp. 43-54, Dec. 1981.

[23] T.-Y. Feng, "A survey of interconnection networks", *Computer*, pp. 12-27, Dec. 1981.

[24] J.S. Turner, "Design of a broadcast packet switching network", *IEEE Trans. Commun.*, vol. 36, no. 6, pp. 734-743, June 1988.

[25] G.J. Anido and A.W. Seeto, "Mutipath interconnection: a technique for reducing congestion within fast packet switching fabrics", *Journal Of Select. Areas Commun.*, vol. 6, no. 9, pp. 1480-1488, Dec. 1988.

[26] H. Uematsu and R. Watanabe, "Architecture of a packet switch based on banyan switching network with feedback loops", *Journal Of Select. Areas Commun.*, vol. 6, no. 9, pp. 1521-1527, Dec. 1988.

[27] K.-I. Sato *et al.*, "broad-band ATM network architecture based on virtual paths", *IEEE Trans. Commun.*, vol. 38, no. 8, pp. 1212-1222, Aug. 1990.

[28] C.-T. Lea, "The load-sharing banyan network", *IEEE Trans. Comput.*, vol. 35, no. 12, pp. 1025-1034, Dec. 1986.

[29] H. Kim and A. Leon-Garcia, "A self-routing multistage switching network for Broadband ISDN", *Journal Of Select. Areas Commun.*, vol. 8, no. 3, pp. 459-466, Apr. 1990.

[30] H. Kim and A. Leon-Garcia, "A Multistage ATM switch with interstage buffers", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. V, pp. 15-20.

[31] S. Urushidani *et al.*, "The rerouting banyan network", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. V, pp. 27-31.

[32] A. Pattavina and P. Campoli, "An ATM switch with folded shuffle topology and distributed access", *Proc. ICC '91*, pp. 1021-1027.

[33] M. Decina, P. Giacomazzi and A. Pattavina, "Shuffle interconnection networks with deflection routing for ATM switching: the closed-loop shuffleout", *Proc. Infocom '91*, pp. 1254-1263.

[34] F. Tobagi and C. Kwok, "The tandem banyan switching fabric: a simple high-performance fast packet switch", *Proc. Infocom '91*, pp. 1245-1253.

[35] F. Bernabei *et al.*, "On non-blocking properties of parallel delta networks", *Proc. Infocom '88*, pp. 326-333, 1988.

[36] A. Autolitano *et al.*, "Application of generalized parallel delta networks to a hybrid broadband switch", *Proc. ICC '89*, pp. 124-127.

[37] F. Bernabei and M. Listani, "Generalized parallel delta networks: a new class of rearrangeable networks", *Proc. Infocom '89*, pp. 3a.2.1-3a.2.4, April 1989.

[38] A. Forcina, T. Di Stefano, and E. Taormina, "A multicast broadband switching module in a hybrid ATM environment", *Proc. ICC '89*, pp. 112-117.

[39] C.-T. Lea, "Multi-log$_2$N networks and their applications in high-speed electronic and photonic switching systems", *IEEE Trans. Commun.*, vol. 38, no. 10, pp. 1740-1749, Oct. 1990.

[40] C.T. Lea and D.J. Shyy, "Tradeoff of horizontal decomposition versus vertical stacking in rearrangeable nonblocking networks", *IEEE Trans. Commun.*, vol. 39, no. 6, pp. 899-904, June 1991.

[41] K.W. Sarkies, "The bypass queue in fast packet switching", *IEEE Trans. Commun.*, vol. 39, no. 5, pp. 766-774, May 1991.

[42] P. Newman, "A fast packet switch for the integrated services backbone network", *Journal Of Select. Areas Commun.*, vol. 6, no. 9, pp. 1468-1479, Dec. 1988.

[43] R.G. Bubenik and J.S. Turner, "Performance of a broadcast packet switch", *Proc. ISS '87*, pp. 1118-1122.

[44] T.T. Lee, "Nonblocking copy networks for multicast packet switching", *Journal Of Select. Areas Commun.*, vol. 6, no. 9, pp. 1455-1467, Dec. 1988.

[45] M.J. Narasimha, "The Batcher-banyan self-routing network: universality and simplification", *IEEE Trans. Commun.*, vol. 36, no. 10, pp. 1175-1178, Oct. 1988.

[46] C. Day, J. Giacopelli and J. Hickey, "Applications of self-routing switches to LATA fiber optic networks", *Proc. ISS '87*, pp. 519-523.

[47] J. Hui, "A broadband packet switch for multi-rate services",*Proc. ICC '87*, pp. 782-788.

[48] J.Y. Hui and E. Arthurs, "A broadband packet switch for integrated transport",*Journal Of Select. Areas Commun.*, vol. 5, No.8, pp. 1264-1273, Oct. 1987.

[49] B. Bingham and H. Bussey, "Reservation-based contention resolution mechanism for Batcher-banyan packet switches", *Electronics Letters*, vol. 24, no. 13, pp. 772-773, June 1988.

[50] N. Arakawa *et al.*, "ATM switch for multi-media switching system", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. V, pp. 9-14.

[51] P. Lau and A. Leon-Garcia, "Design and performance analysis of a buffer subsystem for the Batcher-banyan switch", *Proc. Globecom '90*, pp. 1926-1930.

[52] W. Chen, H.-J. Liu and Y.-T. Tsay, "High-throughput cell scheduling for broadband switching systems", *Journal Of Select. Areas Commun.*, vol. 9, no. 9, pp. 1510-1523, Dec. 1991.

[53] H. Matsunaga and H. Uematsu, "A 1.5 Gb/s 8 x 8 cross-connect switch using a time reservation algorithm", *Journal Of Select. Areas Commun.*, vol. 9, no. 8, pp. 1308-1317, Oct. 1991.

[54] J. Giacopelli *et al.*, "Sunshine: a high performance self-routing broadband packet switch architecture", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. III, pp. 123-129.

[55] J. Hickey and W. Marcus, "The implementation of a high speed ATM packet switch using CMOS VLSI", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. I, pp. 75-84.

[56] J. Hui, "Switching integrated broadband services by sort-banyan networks", *Proc. IEEE*, vol. 79, no. 2, pp. 145-154, Feb. 1991.

[57] A. Pattavina, "Multichannel bandwidth allocation in a broadband packet switch", *Journal Of Select. Areas Commun.*, vol. 6, no. 9, pp. 1489-1499, Dec. 1988.

[58] A. Pattavina, "A multiservice high-performance packet switch for broad-band networks", *IEEE Trans. Commun.*, vol. 38, no. 9, pp. 1607-1615, Sept. 1990.

[59] A. Pattavina, "A broadband packet switch with input and output queueing", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. VI, pp. 11-16.

[60] T.T. Lee, "A modular architecture for very large packet switches", *IEEE Trans. Commun.*, vol. 38, no. 7, pp. 1097-1106, July 1990.

[61] S. Liew and K. Lu, "A 3-stage interconnection structure for very large packet switches", *Proc. ICC '90*, pp. 771-776.

[62] H. Inose, *An Introduction to Digital Integrated Communications Systems*, Peter Peregrinus Ltd., 1979.

[63] J.-P. Coudreuse and M. Servel, "Prelude: an asynchronous time-division switched network", *Proc. ICC '87*, pp. 769-773.

[64] M. Devault, J.-Y. Cochennec and M. Servel, "The 'Prelude' ATD experiment: assessments and future prospects", *Journal Of Select. Areas Commun.*, vol. 6, no. 9, pp. 1528-1537, Dec. 1988.

[65] H. Kuwahara *et al.*, "A shared buffer memory switch for an ATM exchange", *Proc. ICC '89*, pp. 118-122.

[66] Y. Sakurai *et al.*, "Large scale ATM multi-stage switching network with shared buffer memory switches", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. IV, pp. 121-126.

[67] E. Garetti *et al.*, "An experimental ATM switching architecture for the evolving B-ISDN scenario", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. IV, pp. 15-22.

[68] J. Fried, "A VLSI chip set for burst and fast ATM switching", *Proc. ICC '89*, pp. 128-135.

[69] T. Koinuma *et al.*, "An ATM switching system based on a distributed control architecture", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. V, pp. 21-26.

[70] M. Henrion *et al.*, "Switching network architecture for ATM based broadband communications", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. V, pp. 1-8.

[71] H. Suzuki *et al.*, "Output-buffer switch architecture for Asynchronous Transfer Mode", *Proc. ICC '89*, pp. 99-103.

[72] H. Suzuki *et al.*, "Very high-speed and high-capacity packet switching for broadband ISDN", *Journal Of Select. Areas Commun.*, vol. 6, no. 9, pp. 1556-1564, Dec. 1988.

[73] H. Ohnishi *et al.*, "ATM ring protocol and performance", *Proc. ICC '89*, pp. 394-398.

[74] M. Rahnema, "The fast packet ring switch: a high-performance efficient architecture with multicast capability", *Proc. ICC '89*, pp. 884-891.

[75] M. Rahnema, "The fast packet ring switch: a high-performance efficient architecture with multicast capability", *IEEE Trans. Commun.*, vol. 38, no. 4, pp. 539-545, Apr. 1990.

[76] I. Gard and J. Rooth, "An ATM switch implementation - technique and technology", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. IV, pp. 23-27.

[77] K. Ohtsuki, K. Takemura, J. Kurose, H. Okada and Y. Tezuka, "A high-speed packet switch architecture with a multichannel bandwidth allocation", *Proc. Infcom '91*, pp. 155-162.

[78] Y. Yeh, M.G. Hluchyj and A.S. Acampora, "The knockout switch: a simple, modular architecture for high-performance packet switching", *Journal Of Select. Areas Commun.*, vol. 5, no.8, pp. 1274-1283, Oct. 1987.

[79] D. Chen and J. Mark, "SCOQ: a fast packet switch with shared concentration and output queueing", *Proc. Infocom '91*, pp. 145-154.

[80] C. Woodwarth, M. Karol and R. Gitlin, "A flexible broadband packet switch for a multimedia integrated network", *Proc. ICC '91*, pp. 78-85.

[81] H. Chao, "A recursive modular terabit/second ATM switch", *Journal Of Select. Areas Commun.*, vol. 9, no. 8, pp. 1161-1172.

[82] H. Ahmadi *et al.*, "A high-performance switch fabric for integrated circuit and packet switching", *Proc. Infocom '88*, pp. 9-18, 1988.

[83] W. Wang and F. Tobagi, "The Christmas-tree switch: an output queuing space-division fast packet switch based on interleaving distribution and concentration functions", *Proc. Infocom '91*, pp. 163-170.

[84] U. Killat *et al.*, "A versatile ATM switch concept", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. IV, pp. 127-134.

[85] B. Monderer, G. Pacifici and C. Zukowski, "The cylinder switch: an architecture for a manageable VLSI giga-cell switch", *Proc. ICC '90*, pp. 567-571.

[86] V. Benes, " On rearrangeable three-stage connecting networks", *Bell Syst. Tech. Journal*, vol. 41, no. 5, pp. 1481-1492, Sept. 1962.

[87] C. Clos, "A study of non-blocking switching networks", *Bell Syst. Tech. Journal*, vol. 32, no. 2, pp. 406-424, Mar. 1953.

[88] G. Niestegge, "Nonblocking multirate switching networks", *Proc. 5th ITC Seminar*, 1987.

[89] R. Melen and J. Turner, "Nonblocking multirate networks", *SIAM J. Comput.*, vol. 18, no. 2, pp. 301-313, Apr. 1989.

[90] K. Sezaki *et al.*, "The cascade Clos broadcast switching network - a new ATM switching network which is multiconnection non-blocking", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. IV, pp. 143-147.

[91] R. Proctor and T. Maddem, "Synchronous ATM switching fabrics", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. IV, pp. 109-114.

[92] M. Beshai and E. Munter, "A rotating-access ATM switch", in *Proc. ITC-13*, J. Cohen and C. Pack (eds.), pp.53-58, 1991.

[93] K. Eng *et al.*, "A modular broadband (ATM) switch architecture with optimum performance", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. IV, pp. 1-6.

[94] K. Eng and M. Karol, "The growable switch architecture: a self-routing implementation for large ATM applications", *Proc. ICC '91*, pp. 1014-1020.

[95] A. Cisneros, "Large packet switch and contention resolution device", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. III, pp. 77-83.

[96] Y. Kim and K. Lee, "KSMINs: knockout switch based multistage interconnection networks for high-speed packet switching", *Globecom '90 Conference Record*, pp. 218-223.

3. PERFORMANCE EVALUATION OF ATM SWITCHES.

3.1 Introduction.

ATM switches may be divided into a number of categories for the purpose of performance evaluation.

Two broad categories of switch exist.

i) Internally blocking networks (e.g. the banyan networks of [1]);

ii) Internally non-blocking switches (e.g. sort-banyans [2], memory switches [3]).

The two basic approaches available for modelling switch performance are the use of analytical methods and the use of simulation. We begin by discussing analytical methods.

Since the cell length is fixed at 53 octets, it follows that the time taken to transmit a cell from the output port of a switch is constant, assuming a constant line rate of 155.52 Mb/s. Since incoming cells on an input line have start and end times which are in synchronism (due to being contained within an ATD or SDH frame), and also since most ATM switches operate synchronously (i.e. cells from different inputs are launched into the switch fabric simultaneously) discrete-time queuing models seem appropriate for the analysis of ATM switches. Some elementary results in discrete-time queuing theory are presented in Appendix A.

3.2 Performance of internally blocking switches.

The performance of internally blocking switches of the banyan or delta class has been investigated by a number of authors. Three types of banyan network have been considered;

i) unbuffered networks;

ii) single-buffered networks;

iii) multi-buffered networks.

Patel [4] considered a delta network of size a^n x b^n constructed from a x b modules, with the following assumptions.

i) network operation is synchronous, with inputs generating data cyclically

ii) the probability that data is available at an input during each cycle is constant, with a value of λ. The data destination is uniformly distributed over all outputs.

iii) Data which is blocked during a cycle is discarded; there is no queuing.

Under these assumptions, Patel showed that the probability that the data will be routed to the requested destination is

$$p = \frac{b^n \lambda_n}{a^n \lambda}$$

where λ_n is calculated recursively from the equation

$$\lambda_k = 1 - \left(1 - \frac{\lambda_{k-1}}{b}\right)^a ; \lambda_0 = \lambda .$$

Hence the throughput decreases with increasing network size.

Dias and Jump [5] extended these results to the single-buffered case, and also considered the effect of a back pressure mechanism which ensures that data packets are not forwarded to full buffers. Their results were obtained using a Markov chain model of the probability that the nodes in the various stages of the network are in given states. The analysis is made tractable by the assumption that the transition probabilities are identical for all nodes at a given stage. They assumed 'maximum loading', i.e., $\lambda = 1$, and hence obtained an expression for the switch throughput. They showed that a single buffered 2 x 2 switch node can be in one of 14 states, and that, given a knowledge of the state of the network in one cycle, the probability of it being in another state in the next cycle can be determined. Having found these transition probabilities, enough iterations of the equations are performed to converge on the steady-state probabilities that each node of the switch is in a given state. The switch throughput is then given by the probability that the output stage has occupied input links. The performance in the multi-buffered case was found by simulation.

They concluded that the switch throughput increases with an increase in buffer space, but that the increase is marginal for more than two buffers.

Jenq [6] performed a similar analysis for a single-buffered network with back-pressure, and further showed that the results obtained are essentially unchanged if the buffers in a single switch element are assumed to be independent. This greatly simplifies the analysis, by reducing the size of the state space since all buffers in a stage are assumed to be statistically equivalent.

He further extended the model to consider the behaviour of the system if packets denied access to the switch inputs by the back-pressure mechanism queue until the next cycle i.e. if there is an input buffer. The model of the input queue used was the Geo/Geo/1 model, and the probability of service for this queue was found by an iterative process. Jenq found that a 10-stage input buffered banyan saturates with an offered load of 0.453.

Kruskal and Snir [7] showed that the value of λ_n obtained recursively by Patel is very well approximated by the value

$$\lambda_n = \frac{2k}{(k-1)n + \frac{2k}{\lambda}}$$

in the case where a square network is composed of *k* x *k* switch elements.

They gave a formula for the average number of cycles $\overline{W}$ a packet must queue at each stage of a banyan switch composed of *k* x *k* switch elements, i.e.,

$$\overline{W} = \frac{\left(1-\frac{1}{k}\right)\lambda}{2(1-\lambda)}$$

This result was obtained using the formula for the mean waiting time of a discrete-time Geo $^{(A)}$ /D/1 queue, with the *k* input ports each generating an arrival with probability λ in each time slot.

This formula ignores the issue of dependence introduced in the times of arrivals to the second and subsequent stages, but the authors claimed, by comparing the results to those obtained by simulation, that the error was not large.

The approach of Jenq to the single-buffered banyan in [6] was extended to the multi-buffered case by Yoon *et al.* [8]. They showed that the throughput increases from 0.453 to more than 0.7 when the number of buffers is increased from 1 to 6, with only a small further increase for a buffer size of 32. A similar extension of Jenq's approach has been undertaken by Theimer *et al.* [9].

The above results indicate that banyan networks have low throughput in the presence of a uniform load. It has been shown by Kim and Leon-Garcia [10] and by Bubenik and Turner [11] that the performance is degraded further when a non-uniform load is present.

3.3 Classification of internally non-blocking switches.

Internally non-blocking switches for ATM may be divided into three classes for the purpose of performance evaluation.

i) input queuing switches (e.g., the three-phase Batcher-banyan of [2]);

ii) output queuing switches (e.g., the Knock Out Switch [12]);

iii) Shared buffering (e.g., the shared memory switch of [3]).

Hybrids of these classes are also possible, for example switches using some input and some output queuing.

The above classes are ordered in terms of hardware complexity with the input queuing switch being easiest to implement. However, if ordered in terms of the analytical tractability of the models of the performance of these switches, then output queuing is the simplest to analyse, and input queuing the most difficult. Accordingly the output queued switch shall be considered first.

3.4 Output queuing switches.

3.4.1 Homogeneous arrivals.

This case was analysed by Karol *et al.* [13]. They used a Geo $^{(A)}$/D/1 queuing model, to model an N x N output-buffered switch with the size of the batch arrivals in the output buffer given by the moment generating function

$$A(z) = \begin{cases} \left(1-\frac{\lambda}{N}+z\frac{\lambda}{N}\right)^N & ;N<\infty \\ e^{-\lambda(1-z)} & ;N\to\infty \end{cases}$$

Hence the queue size after departures has moment generating function

$$N_D(z) = \frac{(1-\lambda)(1-z)}{A(z)-z}$$

and mean

$$\overline{N}_D = \frac{E[A^2-A]}{2(1-\overline{A})} = \begin{cases} \frac{(N-1)\lambda^2}{2N(1-\lambda)} & ;N<\infty \\ \frac{\lambda^2}{2(1-\lambda)} & ;N\to\infty \end{cases}.$$

Karol *et al.* then showed that the corresponding waiting time has the moment-generating function

$$W_D(z) = \frac{(1-\lambda)(1-A(z))}{\lambda(A(z)-z)}$$

and thus that the mean waiting time is

$$\overline{W}_D = \begin{cases} \frac{(N-1)\lambda}{2N(1-\lambda)} & ;N<\infty \\ \frac{\lambda}{2(1-\lambda)} & ;N\to\infty \end{cases}.$$

This is a special case of the formula for the waiting time in a Geo $^{(A)}$/G/1 queue obtained by Kruskal, Snir and Weiss [14].

Hluchyj and Karol extended the analysis to the finite buffer case in [15], i.e., they analysed the Geo$^{(A)}$/D/1/L queue. They used a Markov chain approach with transition probabilities P_{ij} representing the probability of the queue size (after departures) at the end of the time slot being j, given that it was i at the end of the previous time slot. They showed that

$$P_{ij} = \begin{cases} a_0 + a_1 & ; i = 0, j = 0 \\ a_0 & ; 1 \le i \le L, j = i - 1 \\ a_{j-i+1} & ; 1 \le j \le L-1, 0 \le i \le j \\ \sum_{m=j-i+1}^{N} a_m & ; j = b, 0 \le i \le j \\ 0 & ; otherwise \end{cases}$$

where a_k is the probability of k arrivals. Hence they obtained q_k, the steady state probability that the queue size is k. This leads to an expression for the cell loss probability P_{loss}, i.e., that

$$P_{loss} = 1 - \frac{1 - q_0 a_0}{\lambda}.$$

Thus, for example, the value of λ must be below about 0.80 for the cell loss probability to be below 10^{-10} with a buffer size of 50.

Note that this probability is the probability that an arriving cell will find the system to be full. Cell loss probability may alternatively be defined as the expected number of cells lost, as a proportion of the offered load.

Louvain *et al.* [16] presented similar results for a $Geo^{(A)}/D/1/L$ queue. They derived an identical formula for the cell loss probability. They presented closed form expressions for the distribution of queue size, although they chose the time immediately after arrivals as their measuring point.

They used the observation that, if $P_i = kP_0$, then the value of k is constant even with changes in system size, for queues of this type. Thus P_i for a finite queue may be obtained from the results for an infinite queue.

Yeh *et al.* [12] considered a practical limit of output-queued switches, which is that, for a switch with N inputs, it may be uneconomic to design the switch so that it can route all N input cells to one output, if required. They presented a formula for the number of cells lost per time slot in the situation where at most $L < N$ cells can be routed to each output port. They assumed geometric inter-arrival times at each input, and a uniform distribution of destination addresses across all output ports. In this situation, the cell loss probability for an N x N switch is (assuming infinite output buffer capacity)

$$P_{loss} = \begin{cases} \frac{1}{\lambda}\sum_{k=L+1}^{N}(k-L)\binom{N}{k}\left(\frac{\lambda}{N}\right)^{k}\left(1-\frac{\lambda}{N}\right)^{N-k} & ;N < \infty \\ \left(1-\frac{L}{\lambda}\right)\left(1-\sum_{k=0}^{L-1}\frac{\lambda^{k}e^{-\lambda}}{k!}\right)+\frac{\lambda^{L-1}e^{-\lambda}}{(L-1)!} & ;N \to \infty \end{cases}.$$

Bondi [17] considered the duration of the loss period in a $Geo^{[A]}/D/1/K$ system. The loss period is the duration during which cells are lost in consecutive time slots. He used a Markov chain model to describe the system, with K+2 states. The system is in state $k<K$ if k cells are in the buffer, in state K if the system is full, but no arriving cells have been lost, and in state K+1 if the system is full, and cell loss has occurred. The loss period is thus the duration of the stay in state K+1. The steady-state probability of state K+1 was found by the conventional method.

Desmet *et al.* [18] investigated the $Geo^{[A]}/D/L$ queue. This model is applicable where channel grouping is employed at the output side of the switch, with L channels per group. They developed an approximation for the probability that the queue size is n, valid for large n, of the form

$$P[N_D] \cong \left(\frac{-b}{z_0}\right) z_0^{-n}$$

where z_0 is the real pole of $N_D(z) = N(z)/D(z)$ with the smallest modulus, and $b = N(z_0)/D'(z_0)$. Hence they showed that the tail probability could be approximated by

$$P[N_D > k] \cong \frac{b_0}{1 - z_0}\frac{1}{z_0^{k+1}}.$$

3.4.2 Correlated arrivals - Markov-modulated sources.

Other authors have sought to extend the analysis to more complicated models of input traffic, which result in an arrival pattern with correlations from time slot to time slot. Such correlated traffic is frequently referred to as 'bursty' in the literature.

Descloux [19] used the following model for input traffic.

i) Cells arrive in bursts

ii) The probability of i (new) bursts arriving in one time slot is given by

$$P[i] = e^{-\lambda}\frac{\lambda^{i}}{i!}, i = 0,1,2,...$$

iii) The probability of a burst persisting in the next time slot is α. Thus the burst length is given by $L = 1 + X$ where X is a geometric random variable with parameter α.

iv) A burst generates a cell in each time slot with probability $1-\beta$.

With these assumptions, the system state is defined in each time slot by two variables (i and j) where i is the number of bursts active and j is the number of cells in the queue (after arrivals).

Descloux then obtained equations describing the transition probabilities from one state to another. He presented an algorithm for recursively calculating the equilibrium probability for each state in the special case where $\beta = 0$ (i.e., where each burst generates a cell in every time slot). He found the equilibrium probabilities in the general case where $\beta>0$ by iterating on the transition probability matrix, i.e., by finding

$$\lim_{k\to\infty} \mathbf{P}^k$$

where $\mathbf{P}$ is the matrix of transition probabilities. A difficulty with this approach is the large number of entries in the transition matrix $\mathbf{P}$.

An alternative solution to this problem, in the special case where $\beta = 0$, was considered by Bruneel and Brijland [20]. They used two-dimensional moment generating functions and showed that the mean buffer occupancy (the number of cells in the buffer after an arrival phase) is

$$\overline{N}_A = \frac{2\overline{B}(1-\overline{B}) + E\left[B^2\right] - \overline{B}}{2(1-\alpha)(1-\alpha-\overline{B})} - \frac{\alpha\overline{B}}{(1-\alpha)^2}$$

where B is the number of new bursts which arrive in one time slot. This result is valid regardless of the distribution of B. Making Descloux's assumption, that B has a Poisson distribution with rate λ, it follows that

$$\overline{B} = \lambda$$

and

$$E[B^2 - B] = \lambda^2.$$

The utilisation of the outgoing link is

$$\rho = \frac{\overline{B}}{(1-\alpha)}.$$

The mean burst length is given by

$$L = \frac{1}{(1-\alpha)}.$$

It may be shown for a fixed load ρ, with B drawn from a Poisson distribution, that the buffer occupancy increases linearly with the mean burst length L, i.e.,

$$\overline{N}_A = L\left(\frac{\rho^2}{1-\rho}\right) + \frac{\rho(2-3\rho)}{2(1-\rho)}.$$

Le Boudec [21] described a somewhat more general model of the input traffic, of which the above is a special case. He also used two variables to represent the system state, one of which is the queue size. The other state variable represents the modulator state. The input traffic is assumed to be generated by a modulator, with I states, which evolves as a Markov chain. The probability of k arrivals, given that the modulator is in state i, must also be specified.

Thus, for example, in analysing the system considered by Descloux, the number of active bursts would be chosen as the state of the modulator, with the probability of k cells arriving, given i active bursts, being

$$\binom{i}{k}(1-\beta)^k \beta^{i-k}.$$

Le Boudec presented the transition matrix of the Markov chain describing the evolution of the queue size just after departure instants. He observed that this matrix has the *upper block Hessenberg* property. This means that, if the matrix is partitioned into a number of square blocks, those blocks below the main diagonal, and separated from it by at least one block, contain only zeros. This property of the transition matrix allows a recursive relationship between the equilibrium probabilities to be obtained, according to the equation $\mathbf{X}_{n-1} = \mathbf{X}_n\mathbf{R}_n$, where $\mathbf{X}_n$ is a vector such that $\mathbf{X}_n(i)$ (the entry in the i-th row of $\mathbf{X}_n$) is the equilibrium probability that the system is in the state (Queue size = n, modulator state = i) and where $\mathbf{R}_n$ is a matrix function of the $\mathbf{R}_j$ vectors ($j < n$) and the transition probabilities, and thus can be calculated recursively, commencing with $\mathbf{R}_0$=0. The solution method is broadly comparable to the method of inversion of a triangular matrix by back-substitution.

This procedure allows functions of the equilibrium probabilities to be calculated without explicit calculation of the probability themselves. Thus, for example, the expected number of cells lost in one time slot for a buffer size of K+1 may be determined as

$$\mathbf{L} = \frac{\mathbf{X}_K \mathbf{L}_K}{\mathbf{X}_K S_K}$$

where

$$\begin{aligned}
\mathbf{L}_0 &= \mathbf{f}_0 \\
\mathbf{S}_0 &= \mathbf{e} \\
\mathbf{L}_n &= \mathbf{R}_n \mathbf{L}_{n-1} + \mathbf{f}_n \\
\mathbf{S}_n &= \mathbf{R}_n \mathbf{S}_{n-1} + \mathbf{e}
\end{aligned}$$

with **e** a column vector containing all 1's and $\mathbf{f}_n$ a vector such that $\mathbf{f}_n(i)$ is the expected number of cells lost in a time slot given that the system state at the start of the time slot was (queue size = n, modulator state = i).

This algorithm would appear to be superior to that of Descloux, since (assuming infinite precision arithmetic) it produces exact results, whereas Descloux's approach (for $\beta > 0$) is only asymptotically exact. One difficulty with the method is the computational effort associated with the calculation of $\mathbf{R}_n$. This involves numerous matrix multiplications and a matrix inversion. Hence K matrix inversions are required in order to determine **L**.

An alternative model is that of Li [22]. He used z-transform methods to calculate the moments of the queue size distribution. The arrival process in this approach is a one-dimensional Markov chain wherein the state of the arrival process is considered to be equivalent to the number of arrivals. This is a restrictive assumption compared with the model of Le Boudec, and, for example, could model the system of Descloux only with $\beta = 0$. It was assumed that M cells could be served in every time unit. Hence, the model could be applied to a switch with output channel grouping. However, the intended application was one where the basic unit of time was M time slots. Hence, M cells could be served by a single link in one time unit. The intention was to allow traffic types whose dynamics vary over different time-scales to be modelled, by appropriate choice of M.

The procedure used was to represent the system by a matrix of steady state probabilities **P** where P_{ji} is the joint probability of a buffer occupancy of j and of i arriving cells. Hence, a vector Q(z) of z-transforms was constructed, where

$$Q_i(z) = \sum_{j=0}^{K} z^j p_{ji} .$$

Hence, Q(z) could be written as a product of matrices:

$$\mathbf{Q}(z) = \left[\mathbf{I} - z^{-M}\mathbf{P}^T diag[z^i]\right]^{-1} \mathbf{P}^T diag[z^i]\mathbf{B}(z)$$

where **B**(z) is some vector function of z. The unknowns in this equation were then found by exploiting the analyticity of **Q**(z) inside the unit circle. Li showed that

$$Q_i(z) = \sum_{j=0}^{N} \sum_{k=0}^{M-j-1} P_{kj}(z^{-j} - z^{k-M}) \sum_{r=0}^{N} \frac{z^M \lambda_r(z)}{z^M - \lambda_r(z)} g_{ri}(z) h_{rj}(z)$$

where the P_{kj}'s are found from the boundary conditions, the $\lambda_r(z)$'s are the eigenvalues of

$$\mathbf{P}^T diag[z^i]$$

and where $h_r(z)$ ($g_r(z)$) is the right row (left column) eigenvector corresponding to $\lambda_r(z)$.

Li showed how to extend the model to handle an overall arrival process A consisting of multiple independent discrete-time processes A_k of the above type, i.e., $A = \sum_k A_k$ This allows Li's method to be used to model multimedia traffic. Li used two-state models to represent each traffic source A_k in his examples, so that each source had a geometrically distributed burst length and inter-burst time.

Wang and Frost [23] used the following technique to model a queue with deterministic service times, M servers, a buffer size of K, and where the arriving traffic is modulated by an L-state modulator X_n. The probability that i cells arrive, given that the modulator is in state j is denoted a_i^j. The maximum number of cells which can arrive in one time slot is N. The basis of their model is the system description in Fig. 3.1. This views the queue as a non-linear (but deterministic) system, with a stochastic input process.

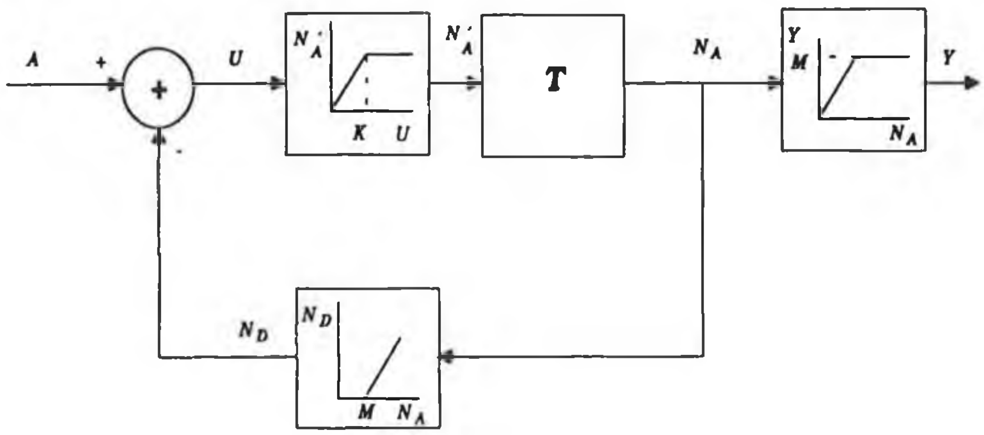


Fig. 3.1: A model of a discrete-time multi-server queue.

The quantities A, U, N_A, N_D and Y are bounded as follows.

$$0 \le A \le N$$
$$0 \le U \le N + K - M$$

$$0 \le N_A \le K$$
$$0 \le N_D \le K - M$$
$$0 \le Y \le M$$

Hence they can be described by vectors of state probabilities, as follows.

$$\mathbf{N}_A^{'} = \mathbf{H}_1\mathbf{U} \quad \text{where } \mathbf{H}_1 = \begin{pmatrix} \mathbf{I}_K & 0 \\ 0 & \mathbf{e}_{N-M+1} \end{pmatrix}$$

$$\mathbf{N}_D = \mathbf{H}_2\mathbf{N}_A \quad \text{where } \mathbf{H}_2 = \begin{pmatrix} e_{M+1} & 0 \\ 0 & I_{K+M} \end{pmatrix}$$

$$\mathbf{Y} = \mathbf{H}_3\mathbf{N}_A \quad \text{where } \mathbf{H}_3 = \begin{pmatrix} \mathbf{I}_M & 0 \\ 0 & \mathbf{e}_{K-M+1} \end{pmatrix}$$

The key to the method is the observation that, for the given model of input traffic, the relationship between U, A and N_D can be represented as

$$\mathbf{U}_S = \mathbf{A}_S\mathbf{N}_D \quad \text{where} \quad \mathbf{A}_S = \begin{pmatrix} a_0^S & 0 & \cdots & 0 \\ a_1^S & a_0^S & \ddots & \vdots \\ \vdots & a_1^S & \ddots & 0 \\ a_N^S & \vdots & \ddots & a_0^S \\ 0 & a_N^S & & a_1^S \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_N^S \end{pmatrix}$$

The entry in row i and column j of $\mathbf{A}_S$ gives the probability that the queue size after arrivals would be equal to i in a queue with an infinite buffer size, given that the modulator was in state S and that the number of cells in the queue before the arrival phase was j. The equilibrium distribution for N_A may be found by solving the linear system of equations

$$\mathbf{Q}_S = \sum_{j=1}^{L} p_{jm}\mathbf{H}_1\mathbf{A}_j\mathbf{H}_2\mathbf{Q}_j$$

where p_{jm} is the probability of a modulator transition from state j to state m and where $\mathbf{Q}_S(i) = P[X_n = S,\ N_A = i]$. This system of equations contains $L(K+1)$ unknowns. Thus the probability that the queue size is i may be found using

$$P[N_A = i] = \sum_S Q_S(i).$$

3.4.3 Non-Markovian models.

Other authors have considered non-Markovian models for cell arrivals. Tran-Gia and Ahmadi [24] considered the queuing model $G^{[A]}/D/1/L$, i.e., a finite-buffered queue with batch arrivals and a general inter-arrival time distribution. They presented an equation relating the probability distribution for the queue size after the arrival of the $(n+1)$-th batch to the queue size after the arrival of the n-th batch. Hence the equilibrium distribution could be obtained recursively by repeated iterations of this equation.

This approach does not allow correlations among successive arrivals to be modelled. Davie [25] suggested a similar approach but one in which the queuing model is the discrete-time GI/G/1 queue. The service time distribution was taken to be the burst length distribution. This is not an exact model of the output buffer of an ATM switch, since it assumes that bursts, rather that cells, are the fundamental units of traffic. Davie claimed that simulation evidence showed that this simplification does not introduce serious errors, if there are no gaps between the cells in a burst.

3.4.4 Mixing of traffic types.

Murata *et al.* [26] considered the aggregation of calls of two types. Specifically, they considered a $GI+Geo^{[A]}/D/1/L$ queue, where the batch arrivals, representing the aggregation of existing calls, have the same probability distribution as the arrivals considered by Karol *et al.* [13], and where the stream with a general distribution of inter-arrival times represents a new call. They used numerical methods similar to those of [24] to find the steady-state distributions for the queue size seen by cells from the GI-stream and the batch stream. Hence, the cell loss probabilities for the two streams could be determined. Murata *et al.* concluded that the burstiness of traffic has a major bearing on the cell loss probability which is obtained. This conclusion is supported by the results of Davie [25].

Hou and Wong [27] considered a queue with two types of traffic source. There are n_1 bursty sources, all with the same geometric distributions of burst and silence lengths, and n_2 Bernoulli sources, with the same arrival rate. The system balance equations were solved recursively for the joint probability distribution of the queue size and the number of active bursty sources. Implementation of their algorithm is straightforward, but it is only applicable in situations where the number of active sources changes only slowly, in comparison to the duration of a time slot.

3.4.5 Shared-buffer switches.

Sakurai *et al.* [28] considered the problem of estimating the cell loss probability in an N x N switch with a shared buffer. First they obtained the distribution of the queue size for each output individually, using the Geo $^{[A]}$/D/1 queuing model. They next sought to obtain an expression for the total number of cells in the shared buffer, i.e.,

$$S = \sum_{i=1}^{N} q_i$$

where N is the number of outputs and q_i is the number of cells buffered for output port i.

They estimated the cell loss probability by convolution and truncation of the queue size distribution, thereby assuming that the arrivals to each output are independent. The assumption of independence is asymptotically correct as the switch size approaches infinity.

Eckberg and Hou [29] pointed out that the q_i's are not independent but are negatively correlated. A large value for q_i implies that a low value is more probable for q_j, where $j \neq i$ because at most N cells can arrive in one time slot. They showed that

$$E[S] = NE[q_i]$$

and

$$Var(S) = N.Var(q_i) + N(n-1).Cov(q_1, q_2).$$

They used z-transforms to obtain a numerical method for determining $Cov(q_1, q_2)$ and approximated the distribution of S by a Gamma distribution with matching mean and variance. The buffer size required for a given cell loss probability was obtained by truncating the resulting distribution.

Rothermel [30] assumed that the number of cells arriving per time slot to each output had a Poisson distribution. The assumption of uniformly loaded outputs was also made. His solution method was based on the calculation of the transition probabilities for the buffer occupancy in a number of steps.

The distribution of the q_i's was first obtained. Hence the probability, P_k, that the buffer contained k cells was obtained by an N-fold convolution of the q_i distribution. Next, the probability was found of N-m cells being served by the output ports, given the presence of k cells in the buffer. This is simply the probability that m outputs are idle, which is given by

$$P'_{k,k+m-N} = \frac{\binom{N}{m} P[q_i = 0]^m P\left[\sum_{N-m} q_i = k \middle| q_i > 0\right]}{P_k}.$$

The conditional probability can be found by the (N-m)-fold convolution of the appropriate distribution. $P'_{k,k+m-N}$ is the probability of a transition from a queue size of k to a queue size of $k+m-N$, in the absence of arrivals. The probability of a transition from k to j can be found as

$$P_{kj} = \begin{cases} \sum_{u=0}^{j} P'_{ku} P[A = j-u], & j < N \\ \sum_{u=0}^{N} P'_{ku} P[A \geq N-u], & j = N \end{cases}.$$

The steady-state distribution of the shared buffer occupancy can then be found from the transition probabilities. This algorithm takes explicit account of the finite buffer size, and so models the problem more precisely than the methods of [28] and [29] in that respect. However, in calculating the transition probabilities, the assumption of independent arrivals to each output port was once again made. This will limit the accuracy of the model when applied to the case of a small switch size.

3.5 Throughput analysis of input-queuing switches.

3.5.1 Homogeneous traffic.

The performance of input queuing switches with a first come first served queuing discipline is degraded compared with that of an output-queued switched by a phenomenon known as head of line (HOL) blocking [2]. This occurs when only the head of line cell in each input buffer can contend for access to the requested output port. If this cell loses contention, because a cell from another input with precedence over it has requested the same output port, the cell behind it in the queue is denied access to the output ports in the next time slot, even though it might have won contention in that time slot, had the first call not been blocked.

Head of the line blocking causes the throughput of the switch to be less that 100%. Thus one goal in the analysis of input-queuing switches is to establish the achievable throughput, given a set of assumptions concerning the arrival process, queuing discipline and switch operation.

The achievable throughput was shown to be 0.586 (or, more precisely 2-√2) for an N x N switch (with N large), assuming a uniform distribution of destination address, by Karol *et al.* [13] and Hui and Arthurs [2].

Karol *et al.* [13] considered the situation when the input queues are saturated, i.e., where, every time a cell wins contention, it is immediately replaced by another at the head of the queue. They analysed a virtual queue consisting of the head of line cells with a given destination. This virtual queue is described by the equation

$$B_n^i = (B_{n-1}^i + A_n^i - 1)^+ .$$

where B_n^i is the number of HOL cells bound for output i in the n-th time slot and A_n^i is the number of cells bound for output i newly arrived at the head of input queues in the n-th time slot.

They proved that, in the limit as $N \rightarrow \infty$, A^i becomes a Poisson process. Thus the above equation describes an M/D/1 queue which has a mean given by

$$\overline{B}^i = \frac{\rho^2}{2(1-\rho)}$$

where

$$\rho = \overline{A}^i$$

is the switch throughput.

They also showed that

$$\overline{B}^i + \overline{A}^i = 1.$$

It follows immediately that $\rho = 2\text{-}\sqrt{2}$.

Hui and Arthurs [2] gave a similar derivation, but one which was somewhat more general. They assumed that cells arrive at each input port with probability λ in each time slot (i.e. Bernoulli arrivals). They further assumed that, with probability p a new cells arrives in the HOL position at an input port in each time slot, given that the queue was empty, or the HOL cell was served, in the previous time slot. The derivation of Karol et *al.* implicitly assumed that $p = 1$. They then derived an equation relating λ and p. Setting $p = 1$ in this equation required λ=2 - $\sqrt{2}$. Hence the throughput at saturation equals 2-$\sqrt{2}$.

Oie *et al.* [31] extended the analysis of throughput to the situation where more than one cell can be delivered through the switch fabric to each output in one time slot. They used the same approach as in [13], but modelled the virtual queue as an M/D/L system. They evaluated the mean queue size numerically and used the relationship

$$\overline{B} + \overline{A} = 1.$$

to obtain the data in Table 3.1, of maximum throughput ρ as a function of L.

These results show that the achievable throughput quickly approaches unity as L increases. Since L calls can be delivered to each output port, but only one cell can be transmitted across the output link, if follows that buffering at the output is also necessary with a switch of this type. These buffers were modelled by a Geo$^{[A]}$/D/1 queue, where the distribution for arriving batches was truncated at a batch size of L.

L	ρ
1	0.5858
2	0.8845
3	0.9755
4	0.9956
∞	1.000

Table 3.1: Maximum throughput vs. number of servers.

Chen and Guerin [32] have extended the approach of [2] to a situation where incoming cells have one of two priorities. High priority cells pre-empt low priority calls in the input queues, and have priority over low priority cells in winning contention for an output port. They obtained an expression relating the probability p of a low priority cell arriving at the head of queue position in an input queue not blocked in the previous time-slot, to the arrival rates λ_H and λ_L of high priority and low priority cells. This equation is a generalisation of the corresponding equation derived in [2]. Saturation occurs when $p = 1$ and is thus a function of λ_H and λ_L. When λ_L=0, the maximum value of λ_H is 0.586, and *vice versa*. For $0 < \lambda_H < 0.586$, the maximum value of λ_L is given by

$$\lambda_{L\max} = \frac{\lambda_H^2 - 6\lambda_H + 4 - \sqrt{-3\lambda_H^4 + 12\lambda_H^3 - 16\lambda_H + 8}}{2(1-\lambda_H)}$$

for the case where high priority cells losing contention return to the input queue, and are not discarded. Chen and Guerin concluded that the maximum throughput can exceed 0.586 (peaking at approximately 0.606) when cells belong to two priority classes. This result is consistent with simulation results obtained by various authors (e.g., [33]) who found that throughput could be increased by modifying the FCFS queuing discipline.

3.5.2 Geometric burst lengths.

Liew and Lu [34] extended the analysis of saturation in two ways. Firstly, they considered non-square N x M switches, where L cells can be routed to each output within one time-slot. Secondly, they considered a bursty model of cell arrivals, where, for the duration of the burst, cells arrive continuously at an input port in consecutive time slots, with the same destination.

The effect of HOL blocking will be exacerbated in the presence of bursty traffic since, even after a HOL cell causing blocking at other input ports wins contention, the new HOL cells will block the same inputs, if it belongs to the same burst.

The burst lengths were assumed to be geometrically distributed.

The probability of a burst continuing into the next time slot was $1\text{-}p$. The authors considered the virtual queue consisting of the number of bursts with cells present at the head of input queues bound for destination i, which we label B_i. They used the same argument as in [13] to show that the number of bursts newly arriving at the HOL position in input queues had a Poisson distribution, assuming saturated input queues. The mathematical treatment they used to determine the expected value of B_i is similar to that used in a Geo/Geo/L queue. The analogy with geometric service time arises because although a cell has a deterministic (unit) service time, it can be replaced in the next time slot, by a cell from the same burst, with probability $1\text{-}p$. Thus the probability that the burst terminated (was served) was p.

The resulting formula for E[B_i] contains an unknown parameter ρ_b, the rate at which new bursts arrive, and L constants, P_j $(0 < j < L)$, which represent the probability that $B_i = j$, and which depend on ρ_b. However, in saturation, the total number of bursts in the system must equal the number of input ports, i.e., $N = M \times$ E[B_i].

The value of ρ_b was obtained numerically by starting with an initial guess, calculating the P_j's and hence setting E[B_i] = N/M to calculate an improved guess for ρ_b. This process was repeated until the calculated value of ρ_b converged to a constant.

The maximum throughput of cells per output, ρ_{out}, could then be found using the relationship

$$\rho_{out} = \rho_b E[burst\ length] = \rho_b \frac{1}{p}$$

and the maximum throughput of cells per input, ρ_{in}, is then given by

$$N\rho_{in} = M\rho_{out}.$$

The analysis of an input queuing switch with multiple servers can be extended to the case where a back-pressure mechanism prevents cells from being delivered to full output buffers. Suppose that the output buffer size is b_0 and the number of cells which can be forwarded through the switch to the requested output in one time slot is L. Iliadis and Denzel [35] modelled such a switch with homogeneous traffic, in the case where $b_0 < L$. They found that the maximum achievable throughput depended on the size of the output buffer, in accordance with Table 3.2.

b_0	ρ
0	0.5858
1	0.6713
2	0.7228
3	0.7576
4	0.7831
∞	1.000

Table 3.2: Throughput vs. output buffer size.

These results were obtained by observing that the saturation throughput is related to the occupancy $\overline{B}$ of the virtual queue of head-of-the-line cells with the tagged destination, by the equation

$$\overline{B} + \rho = 1.$$

The sum of the number of packets in the output buffer and the packets in the virtual queue of packets at the heads of input queues, addressed to that output, can be modelled as an M/D/1 queue, assuming Bernoulli arrivals at the input ports. This observation is valid only in the case where the output buffer is always full, given that cells are present in the virtual queue. This condition is only fulfilled if the number of cells which can be routed to an output port (L) exceeds the buffer capacity (b_0). In this case $\overline{B} = \mathrm{E}[(Q\text{-}b_0)^+]$ where Q is the number of cells in the M/D/1 queue.

Gupta and Georganas [36] considered the more difficult case where $L \leq b_0$. They found the joint distribution of the number of cells in the virtual queue (of cells at the head-of-line position, requesting a given output) and the number of cells in the output buffer. This joint distribution was obtained by solution of the appropriate balance equations. The maximum throughput could then be determined from a knowledge of the mean occupancy of the virtual queue. They found that, for $b_0 > 3$, no appreciable increase in throughput was obtained by increasing L beyond 4. Their results agree with those obtained by Bruzzi and Pattavina [37].

3.5.3 Non-uniform loads.

Li [38] restricted his attention to square (N x N) switches with Bernoulli arrivals which route at most one cell to each output in each time slot, but generalised the model of this situation to cater for a non-uniform distribution of requested destinations, and differing arrival rates at the various input ports.

The rate of arrivals at input port i is given by

$$\lambda_i^{in} = f_i^{in}\bar{\lambda}$$

where

$$\Sigma_0^{N-1} f_i^{in} = N$$

and $\bar{\lambda}$ is the system throughput.

The probability that a cell arriving at input port i is destined for output port k is given by t_{ik} where

$$\sum_{k=0}^{N-1} t_{ik} = 1.$$

The output traffic intensity at port j is denoted λ_j^{out}. In steady state, the vector of output traffic intensities is related to the vector of input traffic intensities via the matrix of t_{ik}'s as

$$\lambda^{out} = \lambda^{in}\mathbf{T}.$$

Li modelled the head-of-queue contention process for output port k as a virtual queue described by

$$C_k{}' = (C_k - 1)^+ + A_k.$$

He used a complex argument to show that A_k, the arrival process at head-of-queue, is a Poisson process with parameter λ_k^{out}. Thus the virtual queue can be modelled as an M/D/1 queue.

The waiting time in this virtual queue is equivalent to the contention time at the head of an input queue, conditioned on the choice of destination k. Li derived an expression for this quantity, assuming that service in the virtual queue is in random order. The random variable describing the contention time at the head of input queue i was thus given by

$$S_i^{in} = \sum_{k=0}^{N-1} t_{ik} S_k^{out}.$$

Hence Li obtained the input utilisation factor $\rho_i^{in} = \lambda_i^{in}\bar{S}_i^{in}$ to be

$$\rho_i^{in} = f_i^{in}\overline{\lambda}\left(\Sigma_{k=0}^{N-1} \frac{t_{ik} f_k^{out}\overline{\lambda}}{2\left(1 - f_k^{out}\overline{\lambda}\right)} + 1 \right)$$

where $\lambda_k^{out} = f_k^{out}\overline{\lambda}$ and thus $\mathbf{f}^{out} = \mathbf{f}^{in}T$.

The system was considered to be in saturation if one or more input queues were saturated. Hence saturation occurs as $\max(\rho_i^{in}) \to 1$. Li found the input port which saturates first by finding the value of i which maximises ρ_i^{in} at an arbitrary value of $\overline{\lambda}$. Li equated the value of $\rho_{i_{max}}^{in}$ to unity, and solved for $\overline{\lambda}$. This gave the maximum throughput of the system with the pattern of non-uniform traffic dictated by the f_k^{in} and t_{ik} values.

He showed that the assumptions on which this derivation is based are invalid if N is small, or if the traffic imbalance is such that the arrivals at an output port come predominantly from a few input ports, so that A_k cannot be assumed to be Poisson. He presented an iterative procedure for refining the estimate of the maximum throughput to take account of these imperfections of his analytical model.

It was found that the achievable throughput varies considerably with the pattern of traffic imbalance. One pattern evaluated by Li resulted in a maximum throughput of just 9%. His model could be extended to the case where multiple cells can be served in each time slot.

3.6 Queuing analysis of input-queuing switches.

3.6.1 Homogeneous traffic.

Karol *et al.* [13] assumed that a Geo/G/1 queue could be used to model the input buffer. Their model of arrivals assumed a geometric interarrival time, with a uniform distribution of destination addresses. The input queues were assumed to be independent, which is a valid assumption if cells arrive independently on each input line, and if the number of inputs and outputs is large, so that waiting times at the head of queue position are uncorrelated from input queue to input queue. They also implicitly assumed that access to output ports is fair, so that no input queue is more likely than any other to have its head of queue cell win contention. This means that the distribution of waiting times at the head of queue position is the same for all queues.

The service time distribution in the Geo/G/1 queue is the distribution of waiting times at the head of queue position, assuming random selection among cells contending for the same output. Karol *et al.* argued that the rate of arrival of cells with a given destination at the heads of input queues becomes a Poisson process as the number of inputs approaches infinity. Thus, the waiting time at the heads of input queues can be represented as the delay distribution of a (virtual) discrete-time M/D/1 queue, with service in random order.

Karol *et al.* presented a numerical method for calculating the first and second moments E[G] and E[G^2] of this latter queue. Hence the mean sojourn time in the input queue can be calculated, using the appropriate formula for the Geo/G/1 queue, as

$$E[W_S] = \frac{\lambda(E[G^2] - E[G])}{2(1 - \lambda E[G])} + E[G]$$

where λ is the arrival rate of cells at the input ports.

Hui and Arthurs [2] assumed that the probability that a cell in the head-of-queue position wins contention in any one time slot is fixed, for a given cell arrival rate, at the heads of queues. Accordingly, they modelled the input buffer as a Geo/Geo/1 queue. The service rate of this queue remained to be determined. The service rate was calculated using the observation that, in the steady state, the mean time between arrivals to the buffer equals the mean service time, plus the mean time between arrivals at the head-of-queue position. Thus

$$\frac{1}{\lambda} = \frac{1}{\mu} + \frac{1}{p} + 1$$

where λ is the cell arrival rate, μ is the service rate, and p is the probability that a new cell arrives at the head-of-queue position in each time slot, given that the queue was empty, or that the head-of-queue cell was served, in the previous time slot.

Hui and Arthurs had already derived an expression relating p and λ , in performing their saturation analysis. It followed that

$$\mu = \frac{2(1-\lambda)}{2-\lambda}.$$

Hence, the mean sojourn time was (using the appropriate formula for a Geo/Geo/1 queue)

$$E[W_S] = \frac{(2-\lambda)(1-\lambda)}{\lambda^2 - 4\lambda + 2}.$$

The assumption of a geometric service time has been checked by simulation. The value of

$$E[W_S] - 1 = \frac{\lambda}{\lambda^2 - 4\lambda + 2}$$

has been compared in Fig. 3.2 to the corresponding result of a simulation of a 100 x 100 switch. A good match can be seen between the two curves, indicating that the first and second moments of the service time for a switch with a finite, but large, number of inputs must correspond closely to those of the geometric service distribution associated with a switch with an infinite number of inputs.

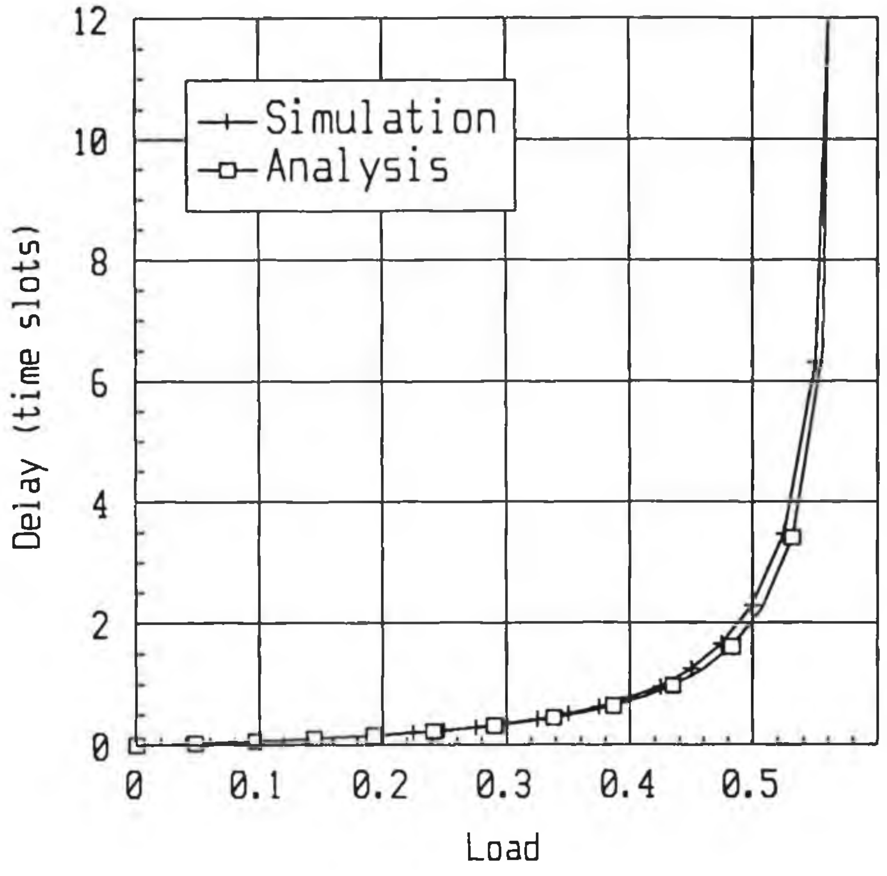


Fig. 3.2: Verification of geometrically distributed HOL time.

Bruzzi and Pattavina [37] modelled a switch with both input and output buffers. The output buffer size was b_O and L cells could be forwarded through the switch fabric to the output buffer in every time slot. The incoming traffic was homogeneous. They found the joint distribution $P(i,j)$ of the number of cells in the virtual queue of cells in the head of line position (i) and the number of cells in the output buffer (j), in a manner similar to that of Gupta and Georganas [36]. They then found the distribution of the input buffer occupancy as follows.

The input queue could be modelled as an M/G/1 queue, where the service time is the waiting time in the head-of-line position. The first and second moments of this waiting time must be found in order to determine the mean delay in the input buffer. It was implicitly assumed that a cell could not be forwarded to an output buffer until all head-of-line cells requesting the same output, and which arrived in previous time slots, had been forwarded to the output side of the switch. It follows that the waiting time in the virtual queue of head-of-line cells is a deterministic function of the number of cells preceding a cell in the virtual queue when it reaches the head-of-line position and of the number of cells in the output buffer.

The probability of waiting k time slots in the virtual queue could then be found by an appropriate summation over the number of arrivals in the head-of-line position in the same time slot (given by a Poisson distribution in accordance with the argument presented in [13]) and the joint distribution $P(i,j)$. The mean waiting time in the input buffer could then be found, using the appropriate formula for an M/G/1 queue.

3.6.2 Non-uniform traffic.

Li [38] has shown that the waiting time (contention time) in the virtual queue of cells at the heads of input queues contending for a given output has a phase-type distribution, under the following assumptions:

i. The arrival process at each input port is a Bernoulli process.

ii. The arrival rates can vary between input ports.

iii. The distribution of destinations for cells arriving at an input port is non-uniform.

iv. The number of inputs equals the number of outputs.

v. The switch size is large.

vi. The non-uniformity of switch traffic is not such that it is predominantly from a few input ports.

vii. Contending cells are served in random order.

viii. At most M ($M > 1$) cells can be routed to each output in one time-slot.

ix. Queue sizes can be finite or infinite.

Li obtained this result by considering the evolution of the Markov chain describing the contention process for a single output. It followed that the input queues could be modelled as Geo/PH/1 queues. Li used matrix methods to calculate values for the first and second moments of the contention time distribution, and hence obtained an expression for the mean waiting time in each input queue. He used a matrix method, adapted from that used by Neuts [39] in the analysis of the M/PH/1 queue, to calculate the distribution of the queue size.

3.7 Fairness in Batcher-banyan switches.

3.7.1 Implementing fair access in Batcher-banyan switches.

The models of input-queuing switches considered in sections 3.5 and 3.6 assume fair access to the output ports from every input port of the switch. In fact, a Batcher-banyan switch, as described in [2], provides unfair access, in that, given a number of head-of-line cells contending for a given output port, the cell from the lowest numbered input port will always win contention. This is referred to as low-end precedence.

The cause of this effect is the operation of the sort elements in the Batcher sorter, which, if they receive two cells with the same routing tag, will always route the upper input to the upper output, and the lower input to the lower output. Hence, the sorted output sorts cells with the same routing tag based on low-end precedence.

One solution to this difficulty is to design the sort elements to choose one cell at random to be routed to the upper output when cells with equal routing tags are received. This adds considerably to the complexity of the sort element. An alternative is to allow the sort elements to continue sorting on the basis of the data following the routing tag. If this has a random distribution, no low-end precedence will be observed. This solution is not appropriate in ATM networks, since the data immediately following the routing tag will be the VPI/VCI field of the header. Hence, a spurious priority level will be attached to cells based on their virtual path and virtual channel identifiers.

Pattavina [40] has suggested a number of hardware mechanisms to counter this difficulty. The sort elements can be modified to change the direction of sorting in alternate time slots. Hence the sorter switches between low-end and high-end precedence. This is referred to as alternating precedence. Another approach he suggested was the use of a rotator network preceding the sorter, which cyclically shifts the data so that, in one time slot, the cell at input port zero has the highest precedence, in the next time slot the cell at input port one has the highest precedence, and so forth. This could be achieved with a network with the topology of an S-banyan, so that $n.2^{n-1}$ switch elements would be required, for a switch with 2^n inputs. This is referred to as cyclic precedence.

The rotator network is controlled by a binary counter. If this is replaced by a shift register generating a random sequence, random precedence will result. In other words, a rigid precedence will be observed in every time slot, but the port with the highest precedence will be chosen at random in each time slot. The shift register sequence will, in practice, generate a pseudorandom sequence, but this should suffice in this application.

Random access will be desirable in situations where a significant proportion of incoming traffic is periodic. Such traffic will experience a possible correlation between the state of the rotator network and the arrival times of cells. True random access would require replacement of the rotator with a network capable of arbitrary permutations of its inputs.

An alternative to using a rotator network to implement cyclic or random precedence is to speed up the switch fabric, and to append a priority field after the routing tag, before submitting cells to the sorter. In this case, a counter (or pseudorandom sequence generator) will have to be provided at each input port of the switch. This could allow a very close approximation to true random access to be obtained, if each pseudorandom sequence generator was initialised appropriately.

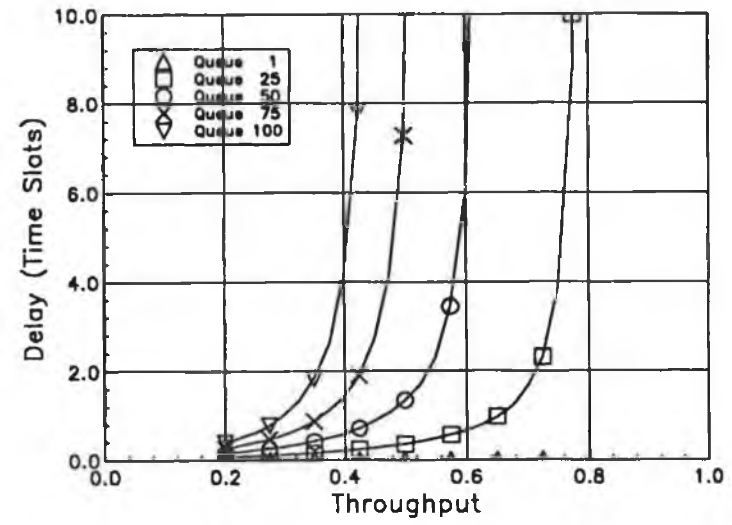
The performance of these various precedence schemes shall now be assessed by simulation.

3.7.2 Homogeneous traffic.

The simulation model is of a 100 x 100 switch, with Bernoulli traffic arriving at the inputs, uniformly distributed across the outputs. The input buffers were assumed to have infinite capacity. The mean number of time slots by which a cell is delayed in the input buffer is shown in Fig. 3.3, in the case of low-end precedence. The three-dimensional plot has been truncated at a delay of 15 time slots, in order to obtain a satisfactory plot of the region below saturation. It can be seen that the throughput varies from 100% at one side of the switch, to about 42% at the other side. Obviously such a switch would be unsatisfactory in ATM applications.

The performance in the cases of alternating precedence and cyclic precedence are shown in Figs. 3.4 and 3.5 respectively. The performance with alternating precedence displays considerable unfairness, but the cyclic precedence appears to offer completely fair access to the output ports. The slight asymmetry in the three-dimensional plot in Fig. 3.5 is an artifact of the plotting algorithm, associated with the resolution of the plot. The two-dimensional plot confirms that the difference in performance between the various input buffers is negligible.

These results confirm and extend those obtained by Pattavina [40] in the case of a switch with 256 inputs.



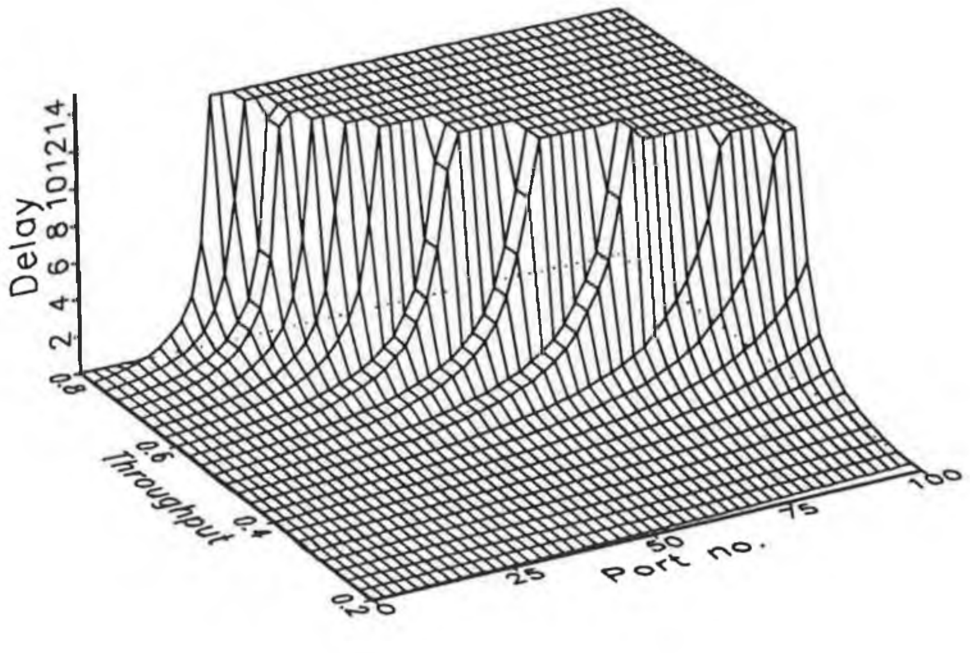
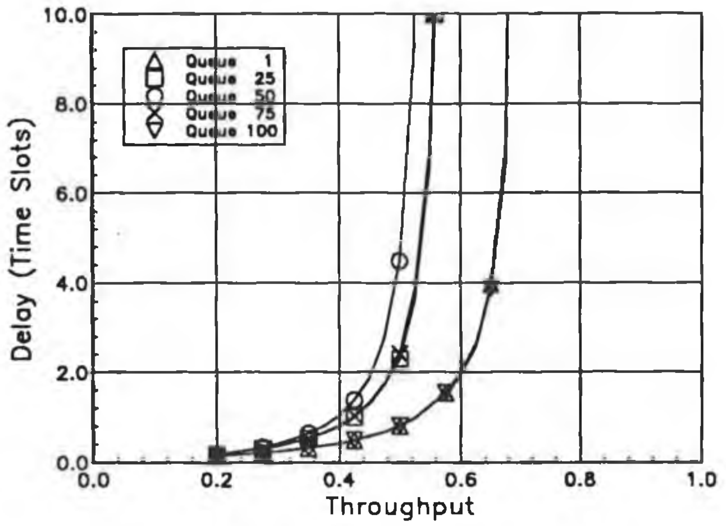


Fig. 3.3: Delay performance with low-end precedence.



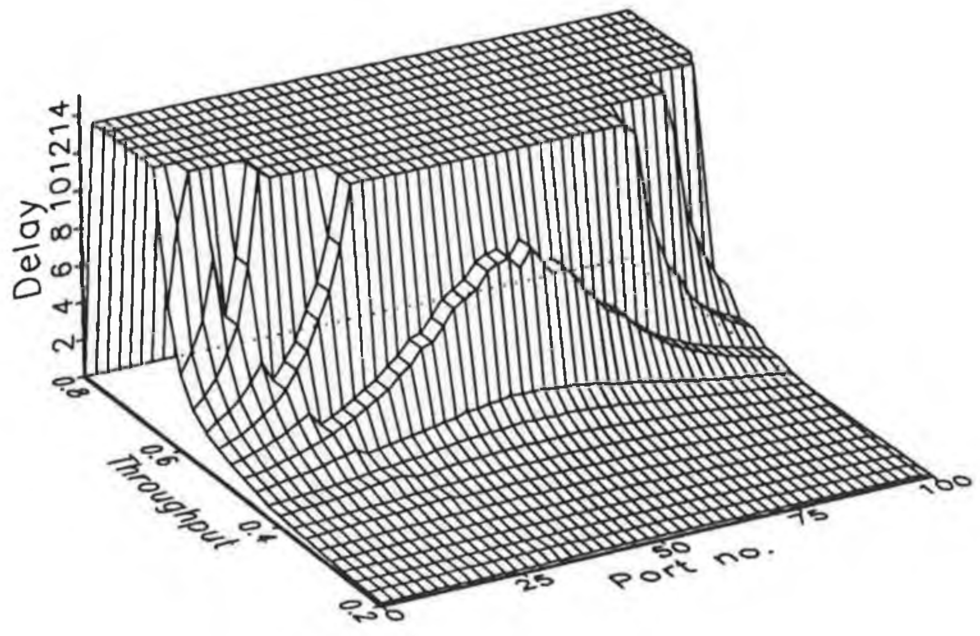
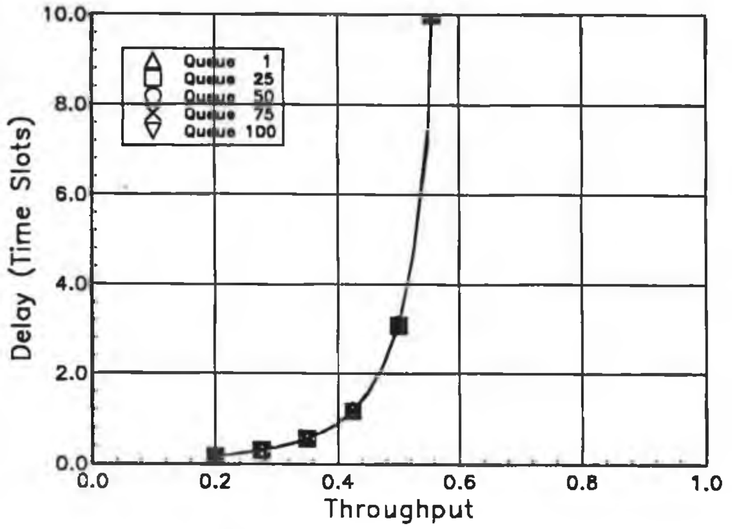


Fig. 3.4: Delay performance with alternating precedence.



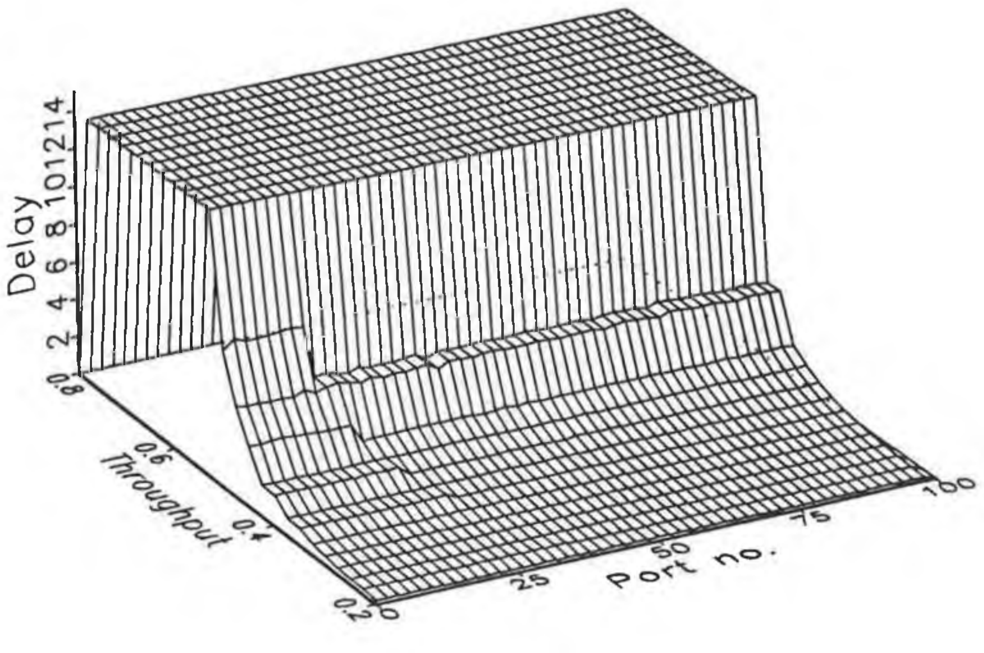


Fig. 3.5 : Delay for cyclic precedence.

The performance with random precedence and random access is shown in Figs. 3.6 and 3.7 respectively. The delay curves obtained for these two cases are identical. A comparison of these results to those presented in Fig. 3.5 indicate that the mean delay is slightly greater in the latter case. This may be attributed to the fact that a cell reaching the head-of-line position at an input port with a low value of precedence continues to receive a low level of precedence in the next time slot, for the cyclic algorithm, whereas, for the other two algorithms, the precedence level in the next time slot will be independent of that in the current time slot.

Random precedence is the preferred choice, because its implementation is only slightly more complex than cyclic precedence, and because its performance is expected to be superior in the presence of periodic traffic. True random access is much more complex to implement.

The performance of the switch may be improved through the use of dynamic priority. A priority field is appended to the routing tag, and the priority level is incremented after every time slot in which a cell loses contention. This ensures that cells newly arrived at the head-of-line position cannot win contention over cells which arrived in previous time slots. The effect of dynamic priority is illustrated in Figs. 3.8 through 3.11.

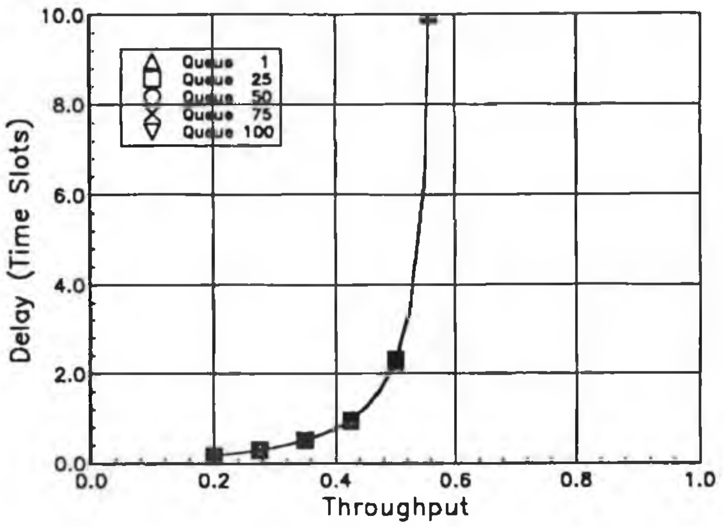
The worst-case throughput in the case of low-end precedence is increased to about 48%, but access is still unfair. However, the performance in the case of alternating priority shows very little variation among input ports. Hence, the use of alternating priority in conjunction with a dynamic priority scheme may be a viable strategy for ensuring fairness of access. However, this strategy may not be successful with very large switches.

The differences in the delay performance observed when using cyclic or random precedence are negligible when dynamic priorities are used. There is also a marginal reduction in the mean delay, as compared with the results in Figs. 3.6 and 3.7.

3.7.3 Bursty Traffic.

The performance degrades significantly if there is a correlation between arrivals in successive time slots. Simulations were performed for the case where cells arrived in bursts with geometrically distributed burst lengths, and geometrically distributed silences. All the cells in a burst request the same destination, and there are no empty time slots during a burst.

The delay in the input buffer in the absence of dynamic priority is shown in Figs. 3.12 through 3.15, for two values of mean burst length, and for fair and unfair access. The corresponding results when dynamic priority is employed are shown in Figs. 3.16 through 3.19.



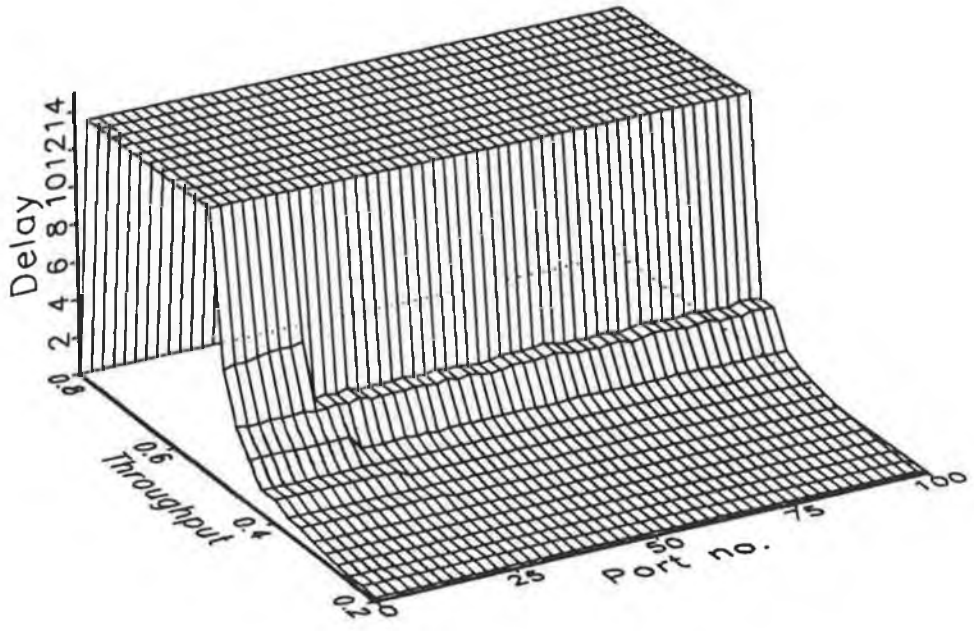
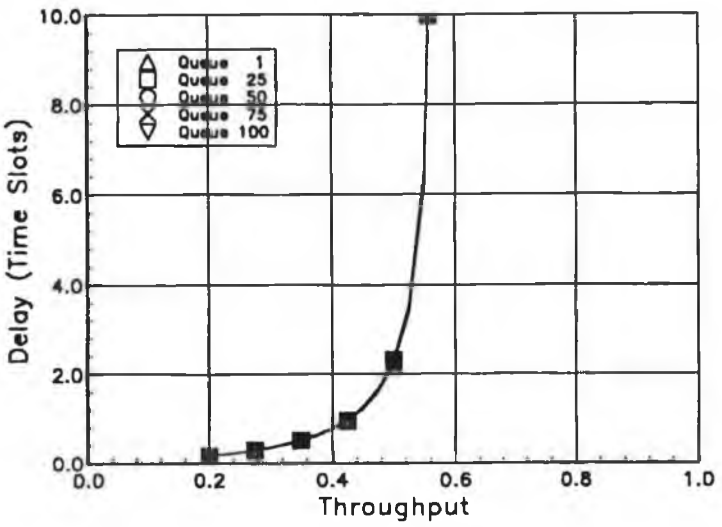


Fig. 3.6: Performance with random precedence.



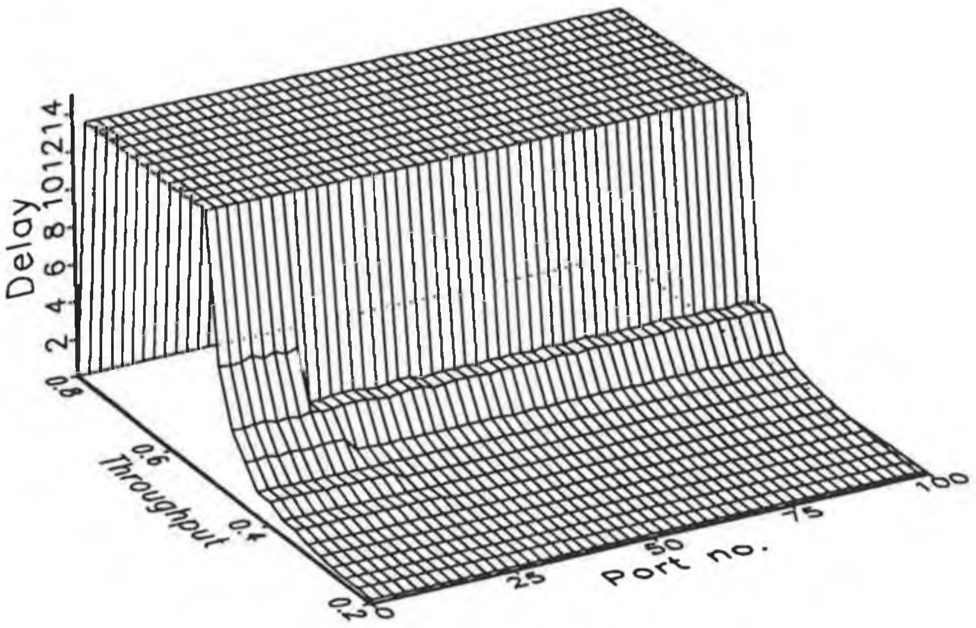
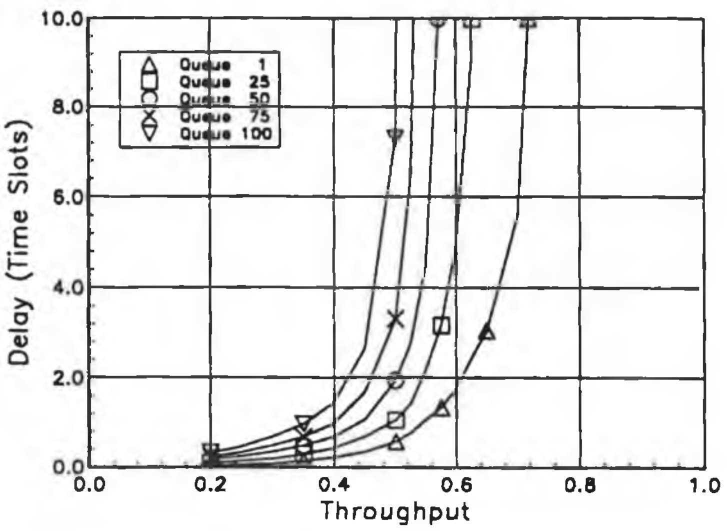


Fig. 3.7: Performance with random access.



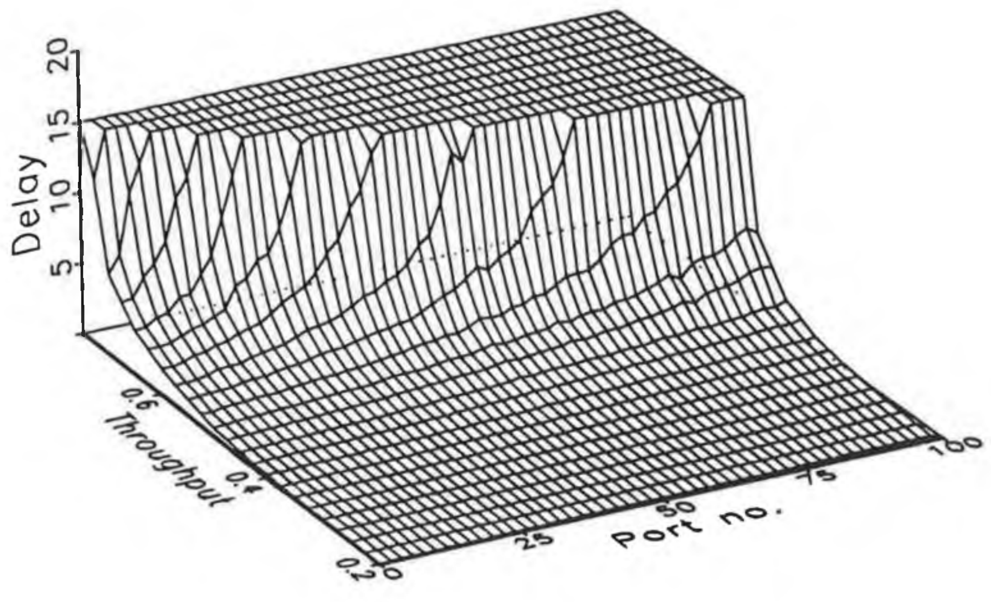
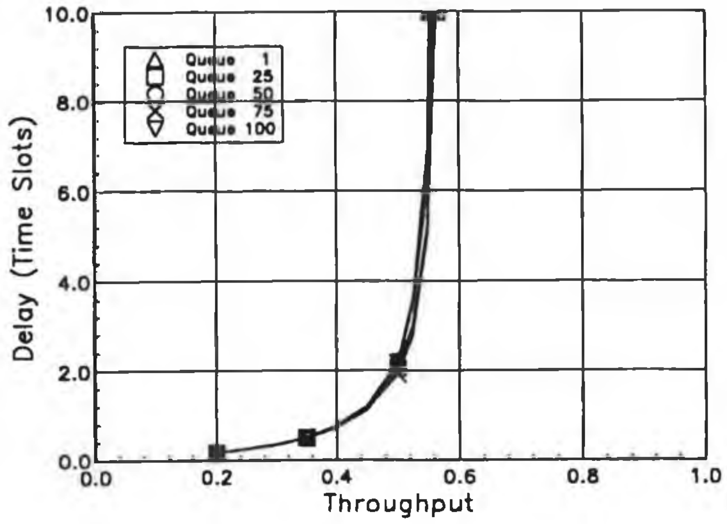


Fig. 3.8 : Delay performance with low-end precedence and dynamic priority.



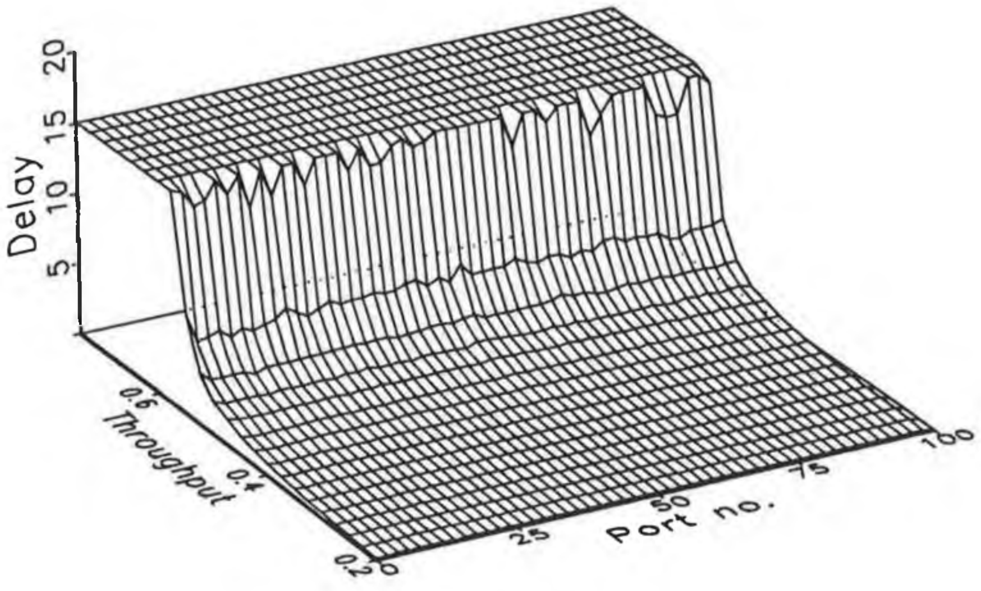


Fig. 3.9: Delay performance with alternating precedence and dynamic priority.

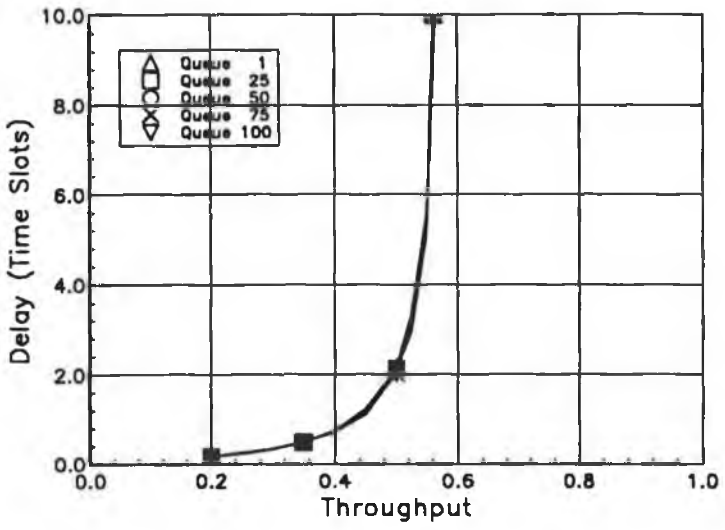


Fig. 3.10: Delay performance with cyclic precedence and dynamic priority.

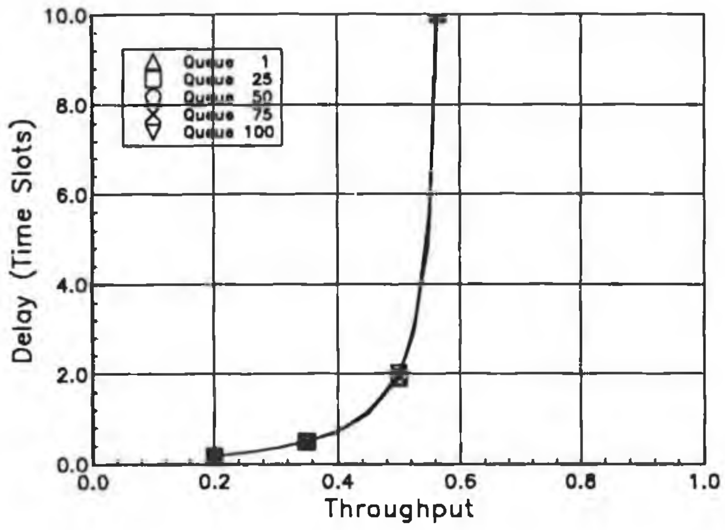
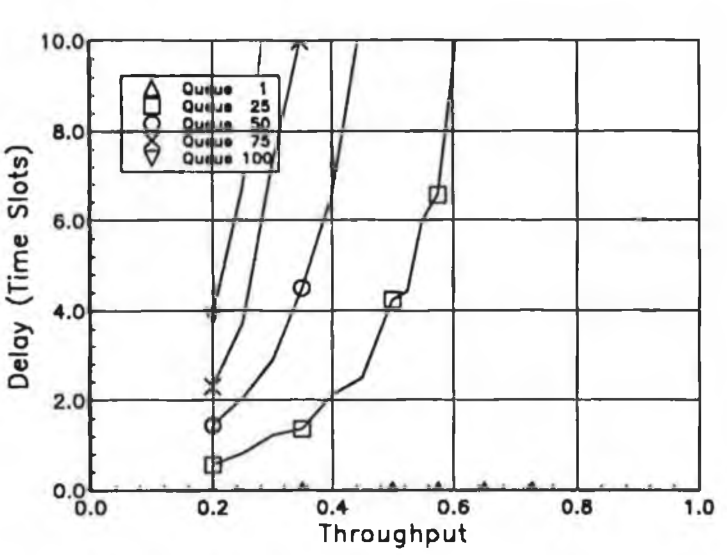


Fig. 3.11: Delay performance with random precedence and dynamic priority.

It is apparent that satisfactory performance cannot be obtained in the presence of such bursty sources, even for moderate switch loads. Hence, the call admission control process should reject such sources. It is interesting to note that access is almost fair, with low-end precedence, when dynamic priority is used. Evidently, the duration of each cell at the head-of-line position is typically so long that the delay is predominantly attributable to cells which arrived in previous time slots, and not to cells which arrived in the same time slot, whose relative position in the virtual queue of head-of-line cells is a function of their input port number.



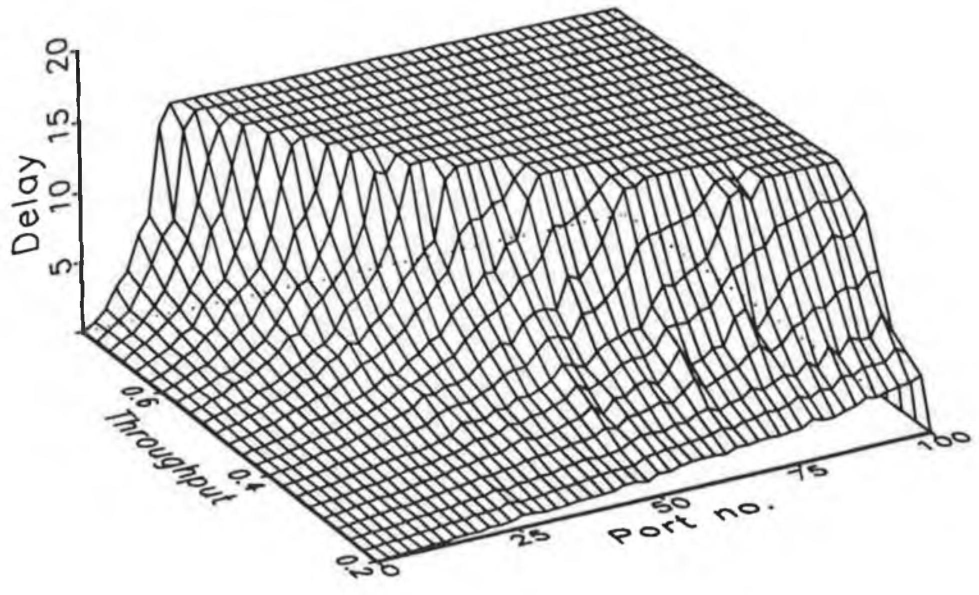
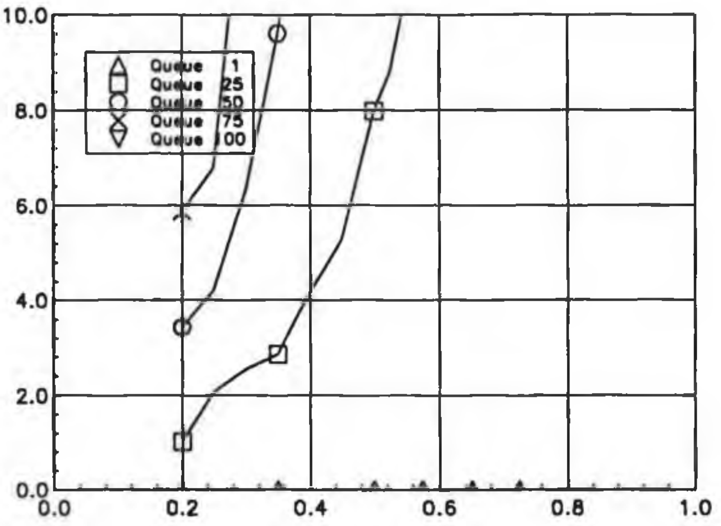


Fig. 3.12: Delay performance with low-end precedence and mean burst lengths of five.



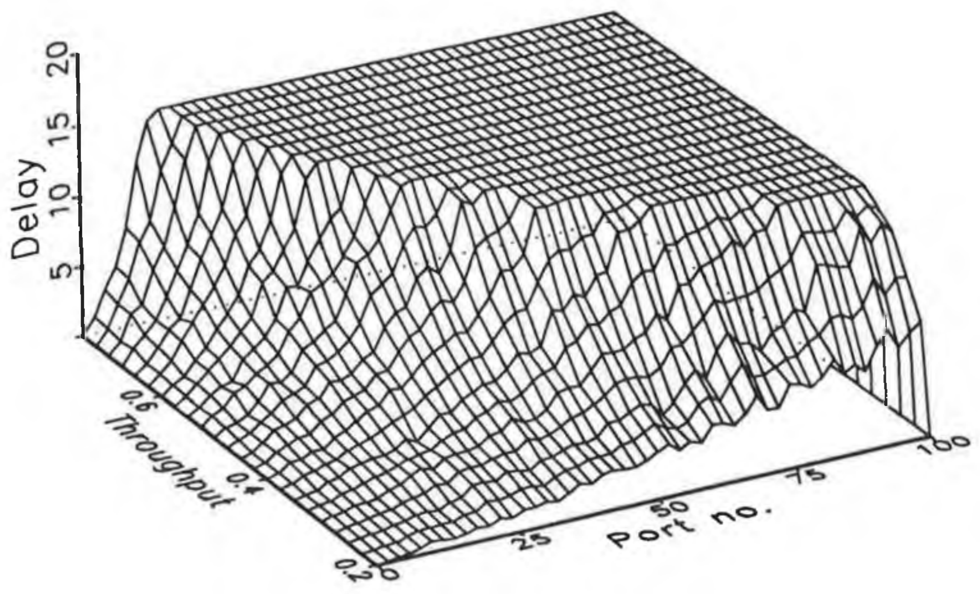


Fig. 3.13: Delay performance with low-end precedence and mean burst lengths of ten.

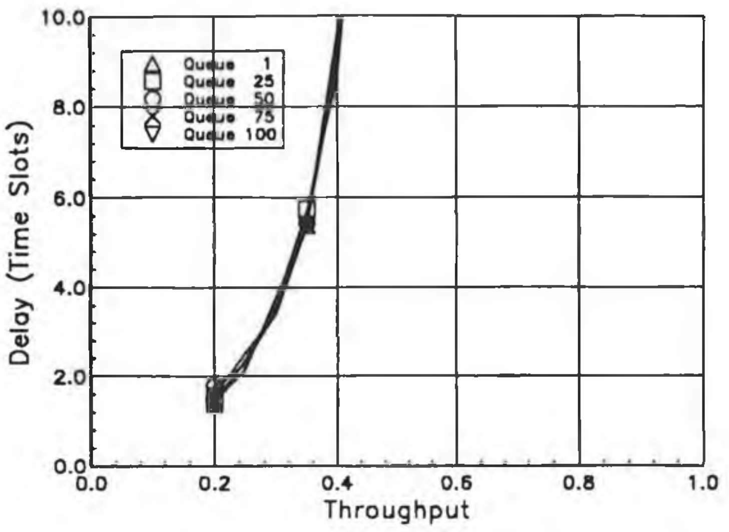


Fig. 3.14: Delay performance with random precedence and mean burst lengths of five.

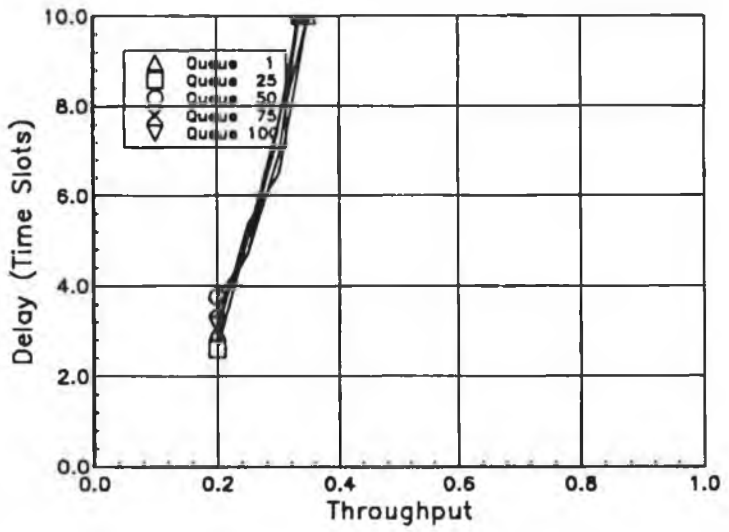
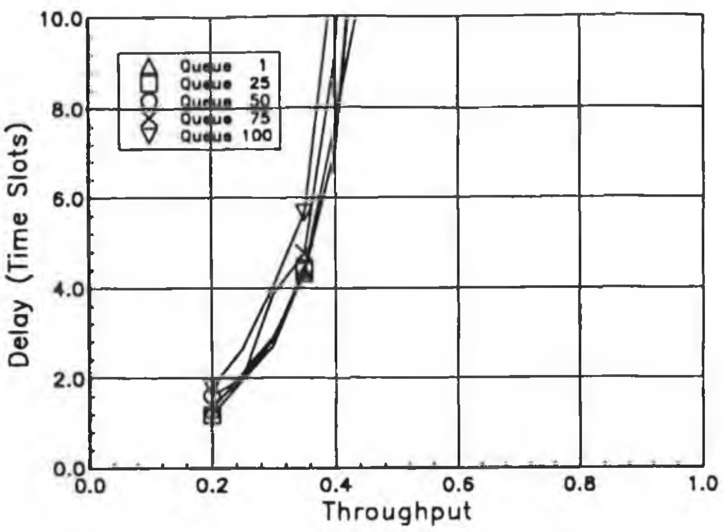


Fig. 3.15: Delay performance with random precedence and mean burst lengths of ten.



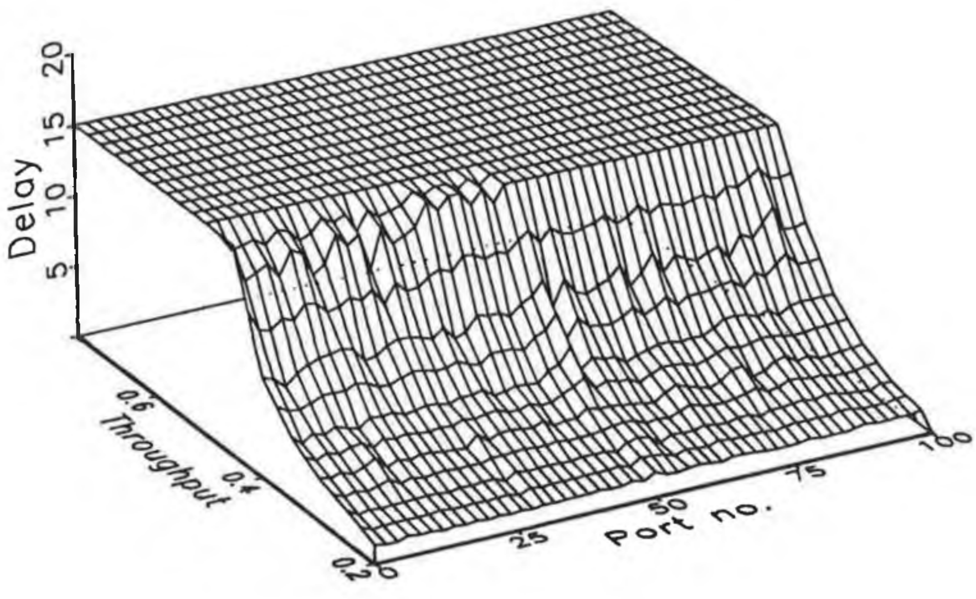
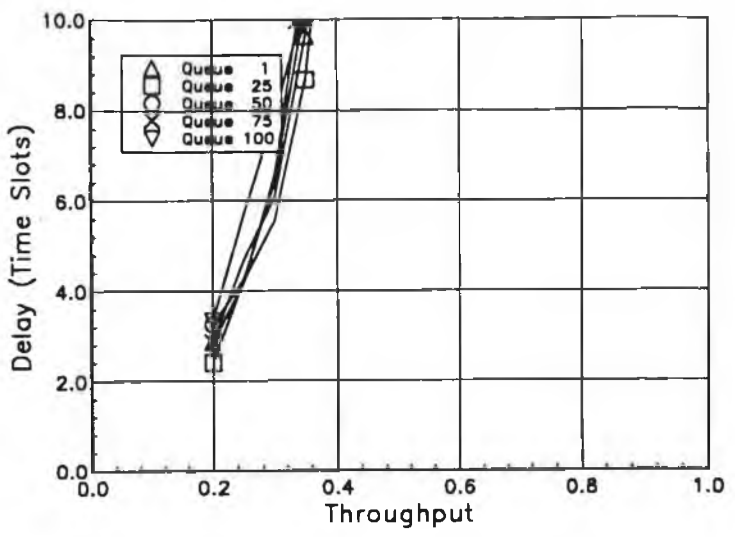


Fig. 3.16: Delay performance with low-end precedence, dynamic priority and mean burst lengths of five.



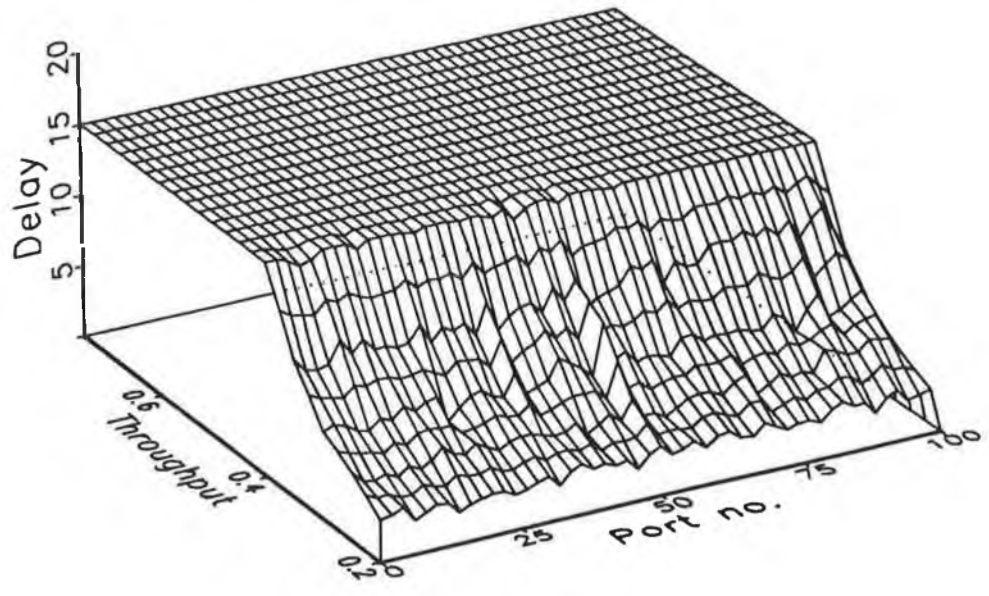


Fig. 3.17: Delay performance with low-end precedence, dynamic priority and mean burst lengths of ten.

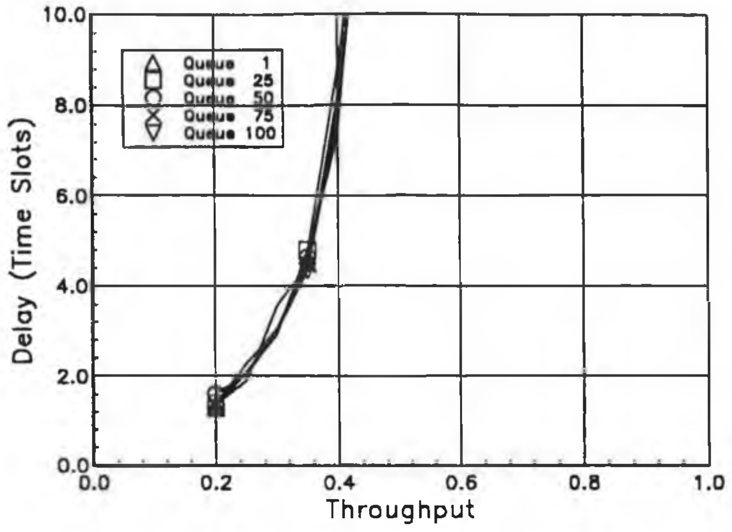


Fig. 3.18: Delay performance with random precedence, dynamic priority and mean burst lengths of five.

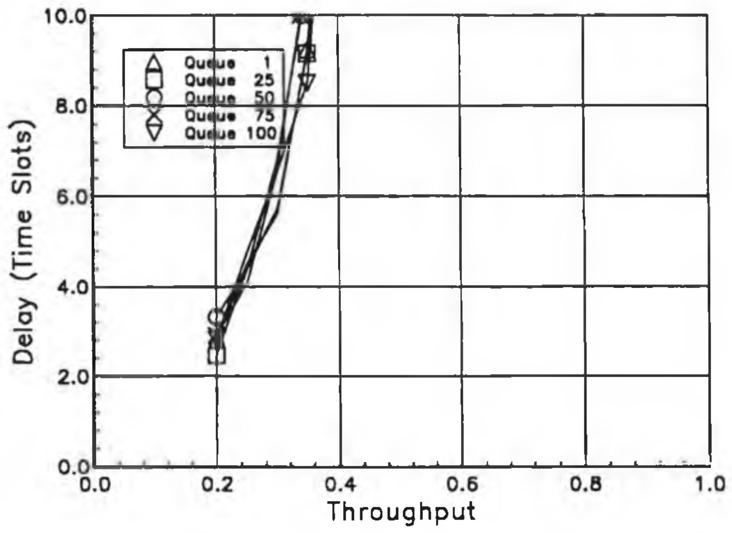


Fig. 3.19: Delay performance with random precedence, dynamic priority and mean burst lengths of ten.

References

[1] L.R. Goke and G.J. Lipovski, "Banyan networks for partitioning multiprocessor systems", *Proc. First Annual Symposium on Computer Architectures*, pp. 21-28, 1973.

[2] J.Y. Hui and E. Arthurs, "A broadband packet switch for integrated transport" *IEEE J. Select. Areas Commun.*, vol. SAC-5, No.8, pp. 1264-1273, Oct. 1987.

[3] H. Kuwahara *et al.*, "A shared buffer memory switch for an ATM exchange", *Proc. ICC '89*, pp. 118-122.

[4] J.H. Patel, "Performance of processor-memory interconnections for multiprocessors", *IEEE Trans. Comput.*, vol. C-30, no. 10, pp. 301-310, Apr. 1981.

[5] D.M. Dias and J.R. Jump, "Analysis and simulation of buffered delta networks", *IEEE Trans. Comput.*, vol. C-30, no.4, pp.331-340, Apr. 1981.

[6] Yih-chyun Jenq, "Performance analysis of a packet switch based on a single-buffered banyan network", *IEEE J. Select. Areas Commun.*, vol. SAC-1, No. 6, pp.1014-1021, Dec.1983.

[7] C. Kruskal and M. Snir, "The performance of multistage interconnection networks for multiprocessors", *IEEE Trans. Comput.*, vol. C-32, no. 12, pp. 1091-1098, Dec. 1983.

[8] H. Yoon, K. Lee and M. Liu, "Performance analysis of multibuffered packet-switching networks in multiprocessor systems", *IEEE Trans, Comput.*, vol. C-39, no. 3, pp. 319-327, Mar. 1990.

[9] T. Theimer, E. Rathgeb and M. Huber, "Performance analysis of buffered banyan networks", *IEEE Trans. Commun.*, vol. *COM*-39, no. 2, pp. 269-277, Feb. 1991.

[10] H. Kim and A. Leon-Garcia, "Performance of buffered banyan networks under nonuniform traffic patterns", *Proc. Infocom '88*, pp. 344-353.

[11] R.G. Bubenik and J.S. Turner, "Performance of a broadcast packet switch", *Proc. ISS '87*, pp. 1118-1122.

[12] Y. Yeh, M.G. Hluchyj and A.S. Acampora, "The knockout switch: a simple, modular architecture for high-performance packet switching", *IEEE J. Select. Areas Commun.*, vol. SAC-5, no.8, pp. 1274-1283, Oct. 1987.

[13] M.J. Karol, M.G. Hluchyj and S.P. Morgan, "Input versus output queueing on a space-division packet switch"*,IEEE Trans. Commun.*, vol. *COM*-35, No.12, pp. 1347-1356, Dec. 1987.

[14] C.P. Kruskal *et al.*, "The distribution of waiting times in clocked multistage interconnection networks", *IEEE Trans. Comput.*, vol. C-37, no.11, pp.1337-1352, Nov. 1988.

[15] M. Hluchyj and M. Karol, "Queueing in space-division packet switching", *Proc. Infocom '88*, pp. 334-343.

[16] J.-R. Louvion, P. Boyer and A. Gravey, "A discrete-time single server queue with Bernoulli arrivals and constant service time", *Proc. ITC 12*, pp. 1304-1311, 1989.

[17] A. Bondi, "On the bunching of cell losses in ATM-based networks", *Globecom '91 Conference Record*, pp. 444-447.

[18] E. Desmet, B. Steyaert, H. Bruneel and G. Petit, "Tail distributions of queue length and delay in discrete-time multiserver queueing models, applicable in ATM networks", *Proc. ITC-13*, J. Cohen and C. Pack (eds.), Elsevier, pp. 1-6, 1991.

[19] A. Descloux, "Contention probabilities in packet switching networks with strung input processes", *Proc. ITC 12*, pp. 815-821, 1989.

[20] H. Bruneel and I. Bruyland, "Performance study of statistical multiplexing in case of slow message generation", *Proc. ICC '89*, pp. 951-955.

[21] J.-Y. Le Boudec, "An efficient solution method for Markov models of ATM links with loss priorities", *IEEE J. Select. Areas Commun.*, vol. SAC-9, no. 3, pp. 408-417, April 1991.

[22] S.-Q. Li, "A general solution technique for discrete queueing analysis of multimedia traffic on ATM", *IEEE Trans. Commun.*, vol. *COM*-39, no. 7, pp. 1115-1132, July 1991.

[23] Q. Wang and V. Frost, "A new solution technique for discrete queueing analysis of ATM system", *Globecom '91 Conference Record*, pp. 358-364.

[24] P. Tran-Gia and H. Ahmadi, "Analysis of a discrete-time $G^{[X]}/D/1$-S queueing system with applications in packet-switching systems", *Proc. Infocom '88*, pp. 861-870.

[25] B. Davie, "ATM network modeling and design for bursty traffic", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. IV, pp. 35-39.

[26] M. Murata *et al.*, "Analysis of a discrete-time single-server queue with bursty inputs for traffic control in ATM networks", *IEEE J. Select. Areas Commun.*, vol. SAC-8, no. 3, pp. 447-458, Apr. 1990.

[27] T.Hou and A. Wong, "Queueing analysis for ATM switching of mixed continuous-bit-rate and bursty traffic", *Proc. Infocom '90*, pp. 660-667.

[28] Y. Sakurai *et al.*, "Large scale ATM multi-stage switching network with shared buffer memory switches", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. IV, pp. 121-126.

[29] A.Eckberg and T.-C. Hou, "Effects of output buffer sharing on buffer requirements in an ATDM packet switch", *Proc. Infocom '88*, pp. 459-465.

[30] K. Rothermel, "Priority mechanisms in ATM networks", *Globecom '90 Conference Record*, pp. 505.1-505.5.

[31] Y.Oie *et al.*,"Effect of speedup in nonblocking packet switch"*,Proc. ICC '89*, pp. 410-414.

[32] J. Chen and R. Guerin, "Performance study of an input queueing packet switch with two priority classes", *IEEE Trans. Commun.*, vol. *COM*-39, no. 1, pp. 117-126, Jan. 1991.

[33] M.G. Hluchyj and M.J. Karol, "Queueing in high-performance packet switching", *IEEE J. Select. Areas Commun.*, vol. SAC-6, no. 9, pp. 1587-1597, Dec. 1988.

[34] S. Liew and K. Lu, "Comparison of buffering strategies for asymmetric packet switch modules", *IEEE J. Select. Areas Commun.*, vol. SAC-9, no. 3, pp. 428-438, April 1991.

[35] I. Iliadis and W. Denzel, "Performance of packet switches with input and output queueing", *Proc. ICC '90*, pp. 747-753.

[36] A. Gupta and N. Georganas, "Analysis of a packet switch with input and output buffers and speed constraints", *Proc. Infocom '91*, pp. 694-700.

[37] G. Bruzzi and A. Pattavina, "Performance analysis of an input-queued ATM switch with internal speedup and finite output queues", *Globecom '90 Conference Record*, pp. 1455-1459.

[38] S.-Q. Li, "Nonuniform traffic analysis on a nonblocking space-division packet switch", *IEEE Trans. Commun.*, vol. *COM*-38, no. 7, pp. 1084-1096, July 1990.

[39] M. Neuts, Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach, John Hopkins University Press, 1981.

[40] A. Pattavina, "Fairness in a broadband packet switch", *Proc. ICC '89*, pp. 404-409.

4. EQUIVALENCE AND BLOCKING IN ATM SWITCHES.

4.1 Equivalence in binary self-routing networks.

4.1.1 Establishing topological equivalence.

Two binary self-routing networks (A and B) are said to be topologically equivalent if a re-ordering of the switch elements in each stage of network A, and a re-ordering of its inputs and outputs, can transform it into network B [1]. Three stronger forms of equivalence are introduced here. If no re-ordering of the input lines is required, the networks are said to be input-equivalent. If no re-ordering of the output lines is required, the networks are said to be output-equivalent. If the networks are both input- and output-equivalent, they are said to be exactly equivalent. It will be established in Section 4.3.5 that only if two networks are exactly equivalent are their blocking properties identical.

Wu and Feng [1] established the topological equivalence of a variety of binary self-routing networks. They presented the appropriate re-ordering scheme for each network to show that it was equivalent to the baseline network. No method was described for arriving at the appropriate scheme. A systematic technique shall be developed here for establishing the topological equivalence of two arbitrary binary self-routing networks.

Suppose it is necessary to renumber the switch elements of stage k of network A, in order to transform it into network B. The renumbering is such that the j-th switch element in stage k is moved to the position of the r-th switch element, where $r = T_k(j)$. If the link permutations in both networks are binary link permutations, as defined in Chapter Two, then the transformation T_k will also be such a permutation. The re-ordering of the switch elements is equivalent to preceding the stage by a link permutation T_k, and following it by a link permutation T_k^{-1}. The permutation must be such that the least significant bit of the link number is unaffected by the permutation. This ensures that two links which terminated on the same switch element in network A continue to do so in network B.

Network A is shown in Fig. 4.1. This is identical, apart from the re-numbering of switch elements, to the network in Fig. 4.2. In this new network, stages k and k-1 are connected by a link permutation defined by

$$(T_k^{AB})^{-1} \quad LP_k^A \quad T_{k-1}^{AB} .$$

If this network is identical to network B (apart from a possible re-ordering of the inputs or outputs), it follows that

$$LP_k^B = \left(T_k^{AB}\right)^{-1} \quad LP_k^A \quad T_{k-1}^{AB}\ , \qquad 0<k<n\ .$$

Thus, networks A and B may be shown to be topologically equivalent if suitable transformations T_k^{AB} $(1 \leq \ k < n)$ can be found to ensure the above. None of these transformations can affect the least significant bit of the link number.

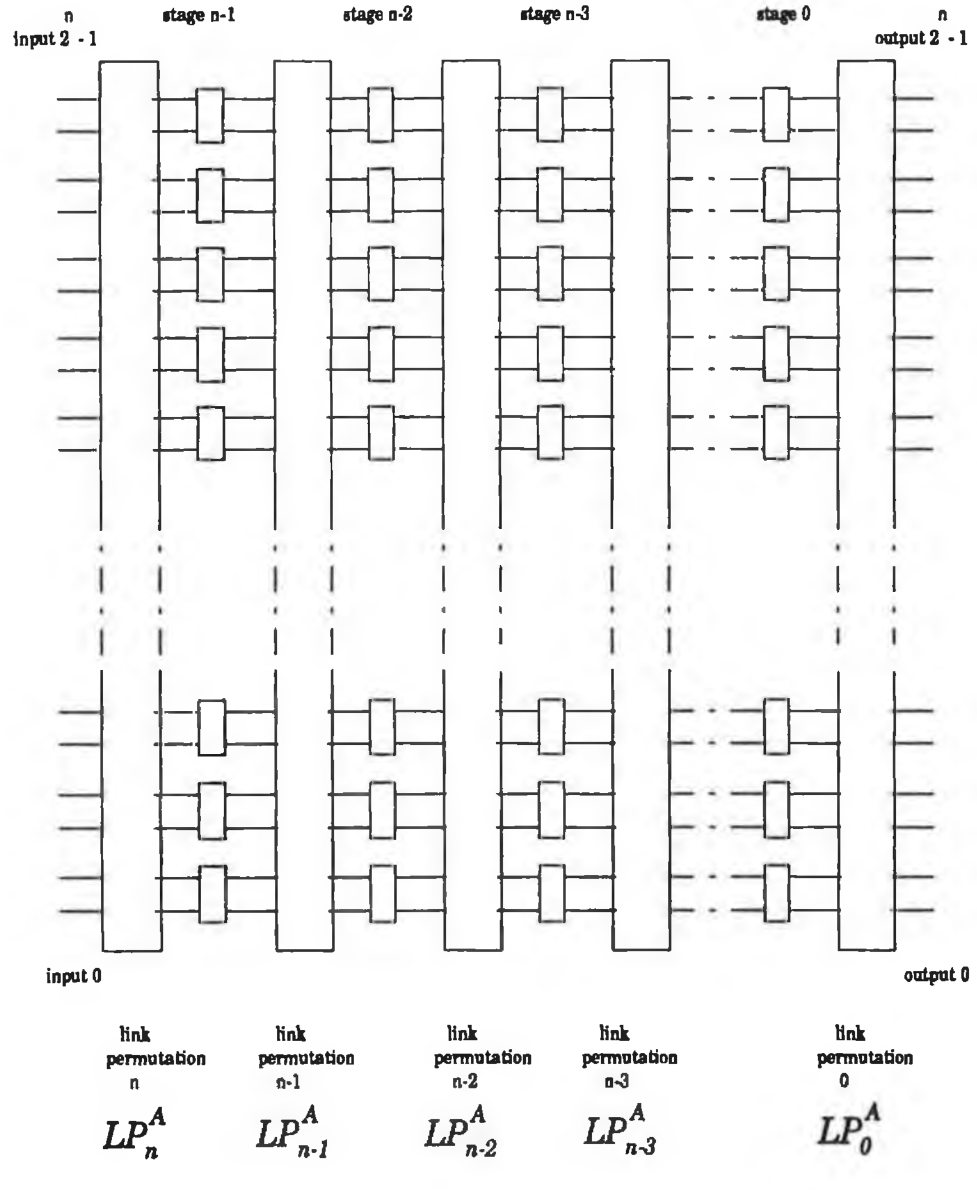


Fig. 4.1: Self-routing network A.

Suppose network A is preceded by a link permutation, T_{in}^{AB}, and followed by a link permutation T_{out}^{AB}. Then the network in Fig. 4.2 may be redrawn as shown in Fig. 4.3. The network in Fig. 4.3 is identical to network B if

$$LP_n^B = T_{in}^{AB} \quad LP_n^A \quad T_{n-1}^{AB},$$

and

$$LP_0^B = (T_0^{AB})^{-1} \quad LP_0^A \quad T_{out}^{AB} \ .$$

If $T_{in}^{AB} = ST$ networks A and B are input-equivalent. If $T_{out}^{AB} = ST$ networks A and B are output-equivalent.

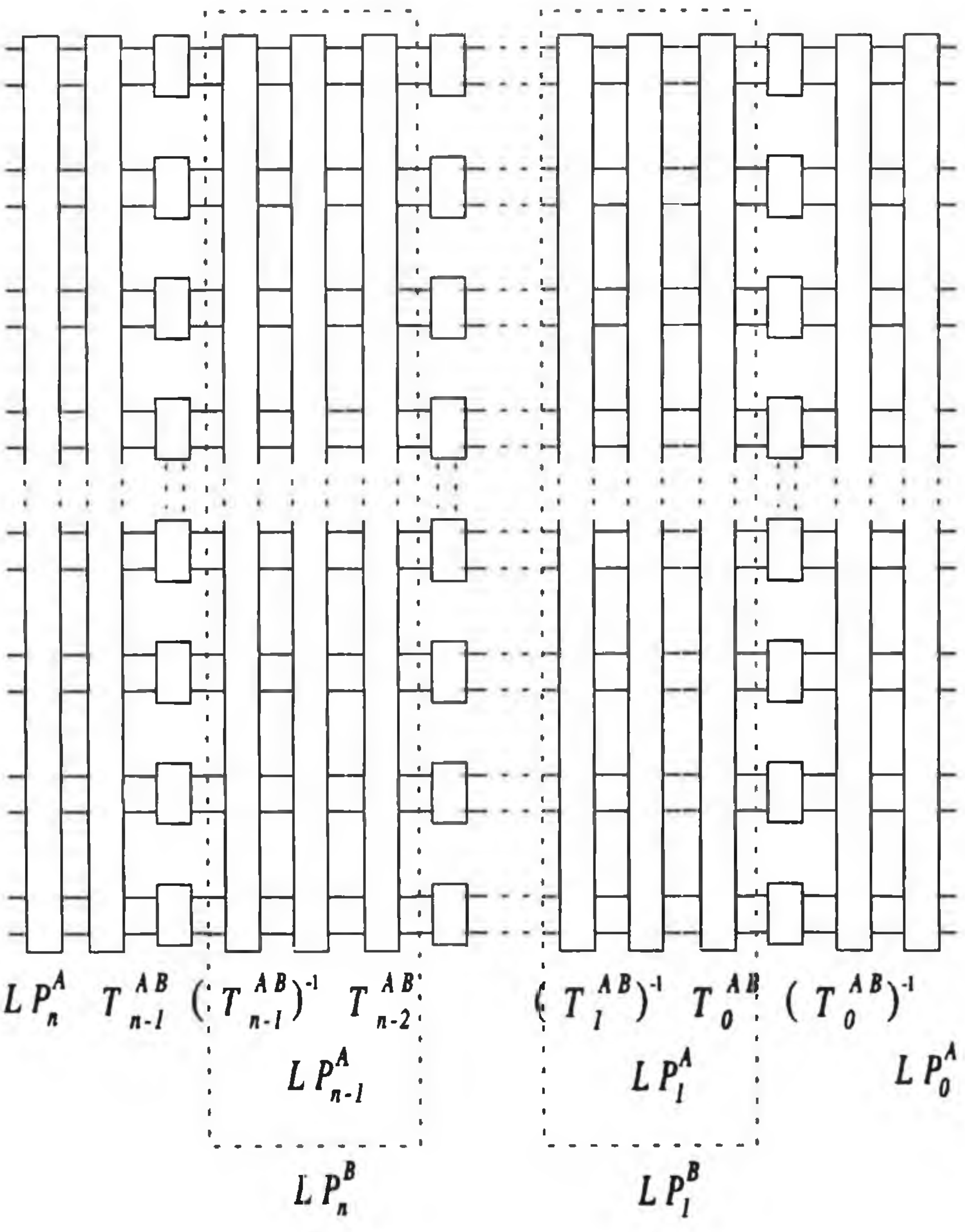


Fig. 4.2: Network A with re-numbered switch elements.

The transformations obey the recurrence relationship

$$T_k^{AB} = LP_k^A \quad T_{k-1}^{AB} \quad (LP_k^B)^{-1} \ .$$

It may be shown by induction that

$$T_k^{AB} = \left(LP_k^A \quad LP_{k-1}^A \quad \dots \quad LP_1^A\right) \; T_0^{AB} \; \left(LP_k^B \quad LP_{k-1}^B \quad \dots \quad LP_1^B\right)^{-1},$$

$$1 \le \; k < n,$$

i.e., defining the transformation

$$U_k = LP_k \quad LP_{k-1} \quad \dots \quad LP_1,$$

then

$$T_k^{AB} = U_k^A \quad T_0^{AB} \quad \left(U_k^B\right)^{-1},$$

$$1 \le \; k < n.$$

Thus T_k^{AB} depends on U_k^A and U_k^B (which are determined by the nature of networks A and B) and on T_0^{AB}.

Attention is restricted in the following to that class of link permutations referred to in Chapter Two as binary link permutations, i.e., those link permutations which are equivalent to a permutation of the binary address of the links.

The transformation T_0^{AB} may not be freely chosen. It must be chosen such that links which terminated on a common switch element in network A also do so in the transformed network of Fig. 4.3. Expressing this requirement mathematically, if

$$T_0^{AB}(2a) = 2b+\varepsilon$$

then

$$T_0^{AB}(2a+1) = 2b+\overline{\varepsilon},$$

where ε is 0 or 1 and $\overline{\varepsilon}=1-\varepsilon$. Note that $2a$ and $2a+1$ differ only in a single bit (the least significant bit). Since T_0^{AB} is a binary link permutation, it follows that $T_0^{AB}(2a)$ and $T_0^{AB}(2a+1)$ can also differ only in a single bit. It may be concluded that this must be the least significant bit, and that $\varepsilon = 0$. If T_0^{AB} is defined by

$$b_0 \xrightarrow{T_0^{AB}} b_{\tau(0)}$$
$$b_1 \xrightarrow{T_0^{AB}} b_{\tau(1)}$$
$$\vdots$$
$$b_{n-1} \xrightarrow{T_0^{AB}} b_{\tau(n-1)}$$

where $\tau(\ )$ permutes the integers 0 through n-1, the requirement may be stated as

$$\tau(0) = 0.$$

The function $zero(T_0^{AB})$ of the link permutation T_0^{AB} is defined as

$$zero\ (T_0^{AB}) \ = \ k \Leftrightarrow b_o \xrightarrow{T_0^{AB}} b_k .$$

Hence $zero(T_0^{AB}) = k$ only if bit zero of x becomes bit k of $T_0^{AB}(x)$. The $zero(\ )$ function may equivalently be defined as

$$\text{zero}(T_0^{AB}) = \tau(0).$$

It follows that the requirement on T_0^{AB} may be compactly stated as

$$zero\ (T_k^{AB}) = 0\ .$$

Otherwise the transformation T_k^{AB} would not represent a re-ordering of the switch elements in stage k of the switch, since links which terminated on the same switch element in network A would not do so in network B.

Note, for example, that

$$zero\ (S_n) \ = \ n\text{-}1$$

and that

$$zero\ (PS_n) = 1\ .$$

For equivalence to be established, it is necessary that

$$zero\ (T_k^{AB}) = 0,$$

i.e., that

$$zero(U_k^A \ \ T_0^{AB} \ \ (U_k^B)^{-1}) = 0.$$

Suppose

$$zero\ (U_k^A) = C_k{}^A$$

and

$$zero\ \ (U_k^B) = C_k{}^B\ .$$

Hence it is required that T_0^{AB} should move the bit in position $C_k{}^A$ to position $C_k{}^B$.

Thus T_0^{AB} is specified by

$$b_o \rightarrow \ b_o$$

$$bc_{k^A} \rightarrow bc_{k^B}, \qquad 1 \leq k < n.$$

If this describes a transformation which is one-to-one and onto, the two networks are topologically equivalent.

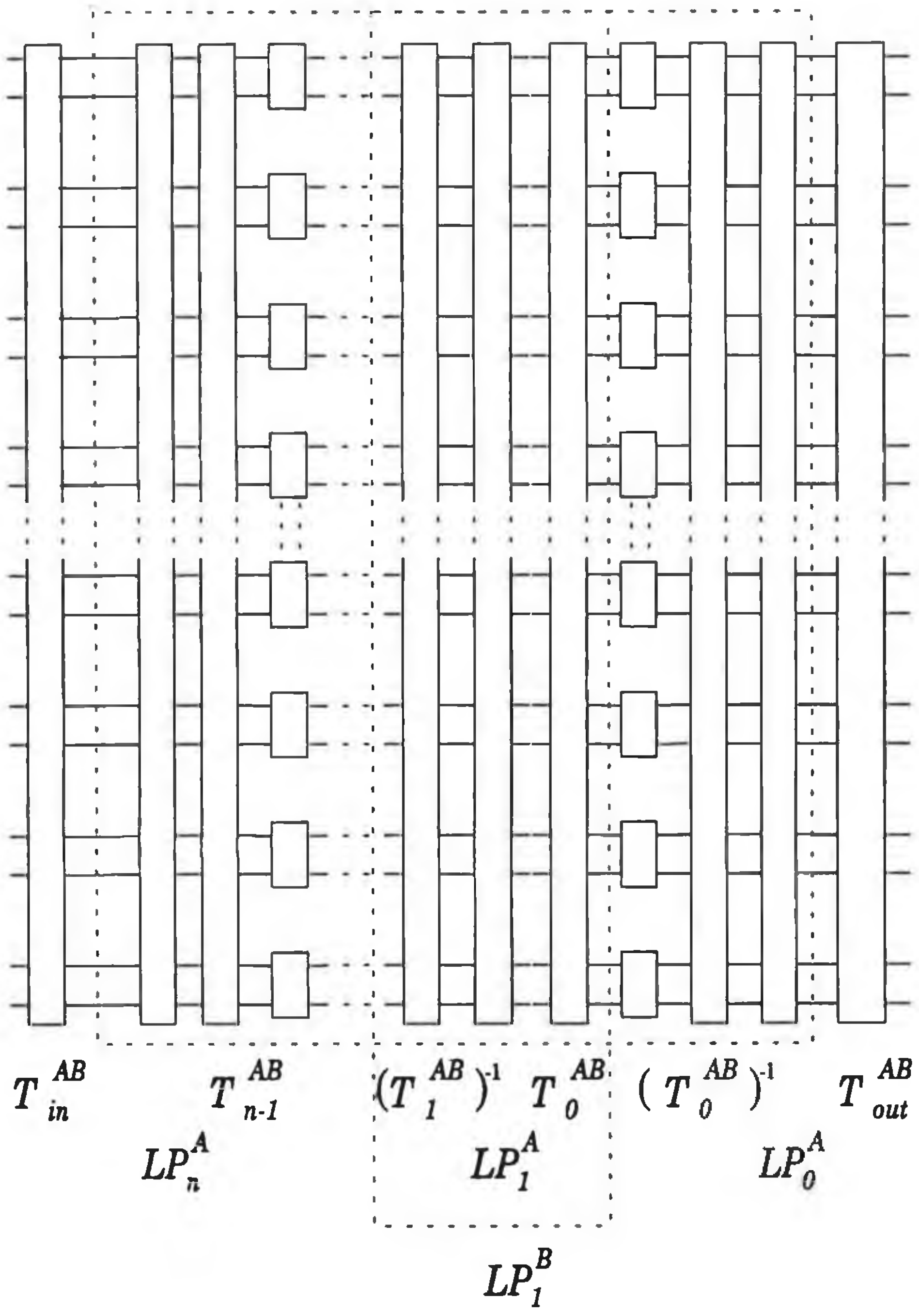


Fig. 4.3: A network corresponding to network B.

T_{in}^{AB} and T_{out}^{AB} may be found (from Fig. 4.2) using the expressions

$$(T_{in}^{AB})^{-1} = LP_n^A \quad T_{n-1}^{AB} \quad (LP_n^B)^{-1}$$

and

$$T^{AB}_{out} = \left(LP^{A}_{0}\right)^{-1} \quad T^{AB}_{0} \quad LP^{B}_{0} \,.$$

If $T^{AB}_{in} \neq ST$ then network A must be preceded by a link permutation T^{AB}_{in} in order to make it input-equivalent to network B. Alternatively network B could be made input-equivalent to network A by preceding it by a link permutation $\left(T^{AB}_{in}\right)^{-1}$. If $T^{AB}_{out} \neq ST$ network A must be followed by a link permutation T^{AB}_{out} to make it output-equivalent to network B or network B must be followed by a link permutation $\left(T^{AB}_{out}\right)^{-1}$ to make it output-equivalent to network A.

The nature of the equivalence between a number of well-known networks will now be established.

4.1.2 The equivalence between the S-banyan and the omega network.

Suppose network A is the S-banyan. Thus

$$LP^{A}_{n} = PS_{n},$$

$$LP^{A}_{0} = ST,$$

and

$$LP^{A}_{k} = S_{k+1}, \quad 0 < k \leq n-1 \,.$$

Network B is the omega network. Thus

$$LP^{B}_{0} = ST$$

and

$$LP^{B}_{k} = PS_{n}, \quad 0 < k \leq n \,.$$

Hence

$$U^{A}_{1} = S_{2},$$

$$U^{A}_{2} = S_{3}S_{2} = IPS_{3},$$

and, since $S_{k+1}IPS_{k} = IPS_{k+1}$, and by induction,

$$U^{A}_{k} = IPS_{k+1} \,.$$

Thus

$$zero\,(U^{A}_{k}) = k \,.$$

Also

$$U_1^B = PS_n,$$

$$U_2^B = (PS_n)^2,$$

and clearly

$$U_k^B = (PS_n)^k,$$

so that

$$zero\ (U_k^B) = k\,.$$

Thus T_0^{AB} is defined by

$$b_o \to b_o$$

$$b_k \to b_k, \qquad 1 \leq k < n\,.$$

Hence

$$T_0^{AB} = ST\,.$$

It follows that the re-ordering of switch elements in stage k is dictated by

$$T_k^{AB} = U_k^A \quad T_0^{AB} \quad (U_k^B)^{-1}$$

$$= IPS_{k+1}(IPS_n)^k\,.$$

Thus

$$T_{n-1}^{AB} = U_{n-1}^A \quad T_0^{AB} \quad (U_{n-1}^B)^{-1}$$

$$= IPS_n \,.\, ST \,.\, [(PS_n)^{n-1}]^{-1},$$

but

$$(PS_n)^{n-1} = IPS_n$$

$$\Rightarrow \quad T_{n-1}^{AB} = IPS_n \,.\, ST \,.\, PS_n$$

$$= ST.$$

Hence

$$(T_{in}^{AB})^{-1} = LP_n^A \quad T_{n-1}^{AB} \quad (LP_n^B)^{-1}$$

$$= PS_n \,.\, ST \,.(PS_n)^{-1}$$

$$= ST.$$

Also

$$T^{AB}_{out} = \left(LP^{A}_{0}\right)^{-1} \quad T^{AB}_{0} \quad LP^{B}_{0}$$

$$= (ST)^{-1} \,.\, ST \,.\, ST$$

$$= ST.$$

Thus the S-banyan is exactly equivalent to the omega network, as shown in Fig. 4.4.

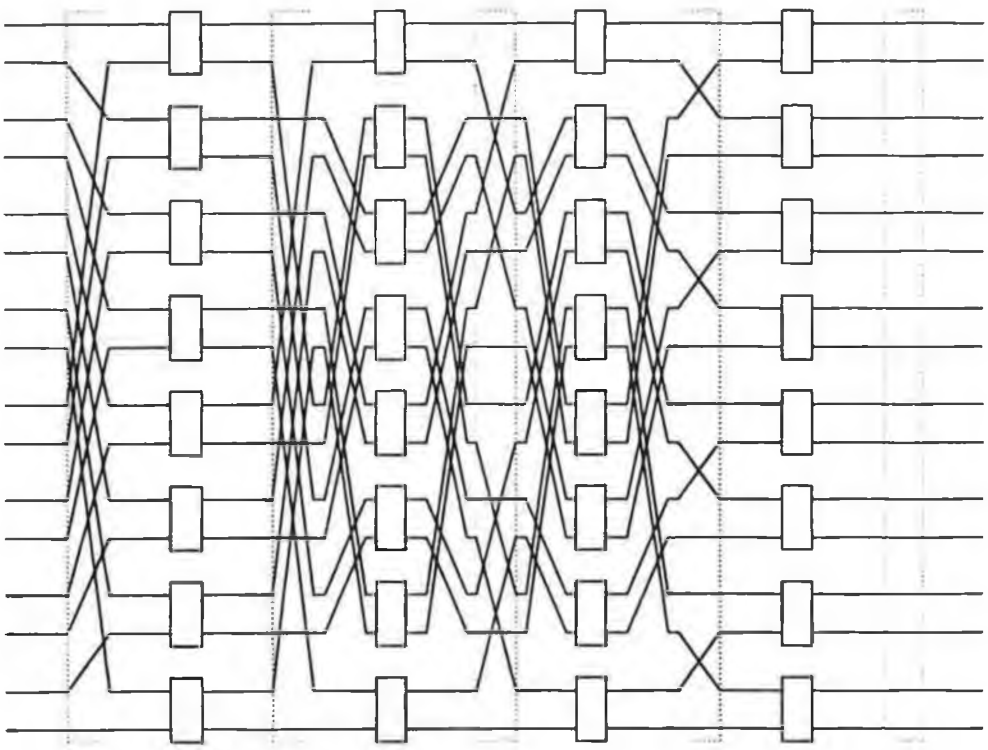


Fig. 4.4: Equivalence of the S-banyan and omega networks.

4.1.3 The equivalence between the S-banyan and the baseline network.

In this case the baseline network (network B) is defined by:

$$LP^{B}_{n} = ST,$$

$$LP^{B}_{0} = ST,$$

and

$$LP^{B}_{k} = IPS_{k+1},\ 0 < k < n\,.$$

Thus

$$U_1^B = IPS_2$$

and

$$U_2^B = IPS_3\, IPS_2 = R_3,$$

where R_n is a bit reversal of the n least significant bits of the link number.

It is found by induction that

$$U_k^B = R_{k+1}\,,$$

since

$$R_{k+1} = IPS_{k+1}\, R_k.$$

Thus

$$zero\,(U_k^B) = k.$$

Hence

$$T_0^{AB} = ST$$

and

$$T_k^{AB} = U_k^A \quad T_0^{AB} \quad (U_k^B)^{-1}$$

$$= IPS_{k+1}.ST.R_{k+1}.$$

In particular,

$$T_{n-1}^{AB} = U_{n-1}^A \quad T_0^{AB} \quad (U_{n-1}^B)^{-1}$$

$$= IPS_n.ST.R_n$$

$$= IPS_n R_n.$$

Hence

$$(T_{in}^{AB})^{-1} = LP_n^A \quad T_{n-1}^{AB} \quad (LP_n^B)^{-1}$$

$$= PS_n \;\; IPS_n\, R_n.ST$$

$$= R_n,$$

and

$$T_{out}^{AB} = (LP_0^A)^{-1} \quad T_0^{AB} \quad LP_0^B$$

$$= ST.ST.ST$$

$$= ST.$$

Thus the S-banyan and the baseline network are output-equivalent, but are not input-equivalent. This can be seen clearly in Fig. 4.5.

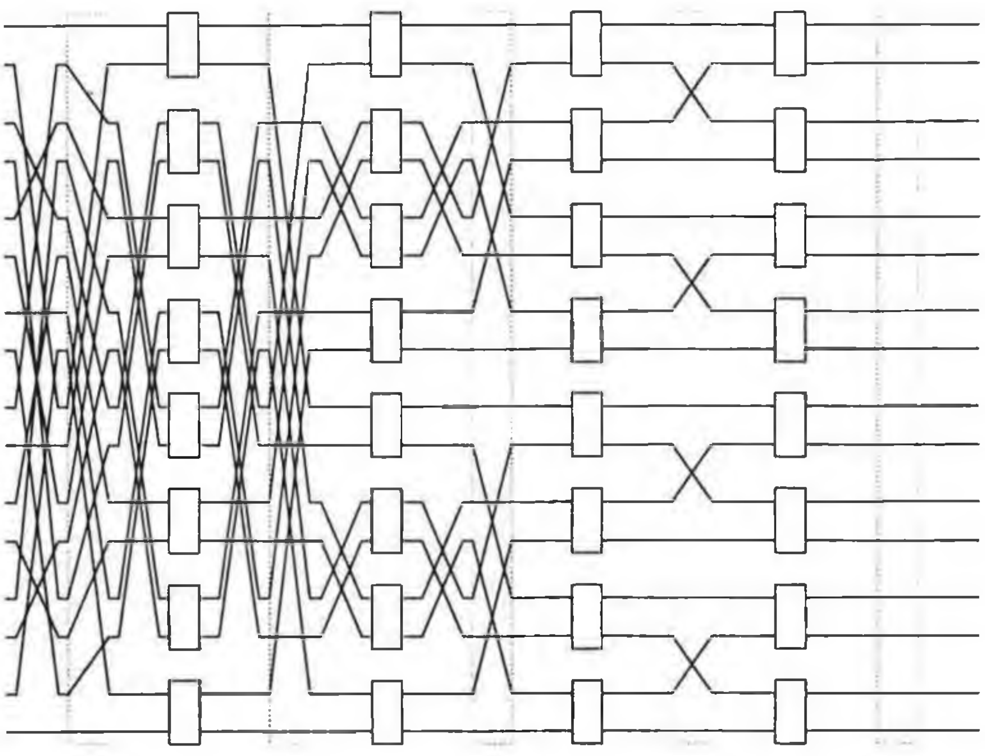


Fig. 4.5: Equivalence of S-banyan and baseline networks.

4.1.4 The equivalence between the S-banyan and the inverse S-banyan.

Suppose network A is the S-banyan.

Thus

$$LP_n^A = PS_n,$$

$$LP_0^A = ST,$$

and

$$LP_k^A = S_{k+1}, \quad 0 < k < n .$$

Network B, the inverse S-banyan, is thus defined by

$$LP_0^B = IPS_n,$$

$$LP_n^B = ST,$$

and

$$LP_k^B = S_{n-k+1}, \qquad 0 < k < n .$$

It has already been established that

$$U_k^A = IPS_{k+1}.$$

Thus

$$zero\,(U_k^A) = k .$$

The expression for U_k^B can be easily found. By definition

$$U_k^B = S_{n-k+1}S_{n-k+2}....S_n$$

$$= (S_2......S_{n-k})^{-1}(S_2.....S_{n-k})\,(S_{n-k+1}S_{n-k+2}....S_n)$$

$$= (S_2......S_{n-k})^{-1}(S_2........S_n)$$

$$= (S_{n-k}S_{n-k-1}...S_2)\,(S_n.....S_2)^{-1},$$

but it has already been established that

$$S_k......S_2 = IPS_k$$

$$\Rightarrow \quad U_k^B = IPS_{n-k}\,PS_n .$$

Hence

$$zero(U_k^B) = n-k .$$

Thus T_0^{AB} is defined by

$$b_o \rightarrow \quad b_o,$$

$$b_k \rightarrow \quad b_{n-k}, \quad 0<k<n .$$

Hence a bit reversal is performed on all but the least significant bit by T_0^{AB}, i.e.,

$$T_0^{AB} = R_n PS_n.$$

It follows that

$$T_k^{AB} = U_k^A T_0^{AB} (U_k^B)^{-1}$$

$$= IPS_{k+1}\,R_n\,PS_n\,(IPS_{n-k}\,PS_n)^{-1}$$

$$= IPS_{k+1}\,R_n\,PS_n\,IPS_n\,PS_{n-k}$$

$$= IPS_{k+1}\,R_n\,PS_{n-k} .$$

In particular

$$T_{n-1}^{AB} = U_{n-1}^{A} T_0^{AB} \left(U_{n-1}^{B}\right)^{-1}$$

$$= IPS_n\, R_n\, PS_n\;\; IPS_n \;=\; IPS_n R_n.$$

So

$$\left(T_{in}^{AB}\right)^{-1} = LP_n^A T_{n-1}^{AB} \left(LP_n^B\right)^{-1}$$

$$= PS_n\, IPS_n\, R_n\, ST$$

$$= R_n$$

$$\Rightarrow \quad T_{in}^{AB} = R_n.$$

In addition,

$$T_{out}^{AB} = \left(LP_0^A\right)^{-1} T_0^{AB}\, LP_0^B$$

$$= ST\;\; R_n\, PS_n\, IPS_n$$

$$= R_n.$$

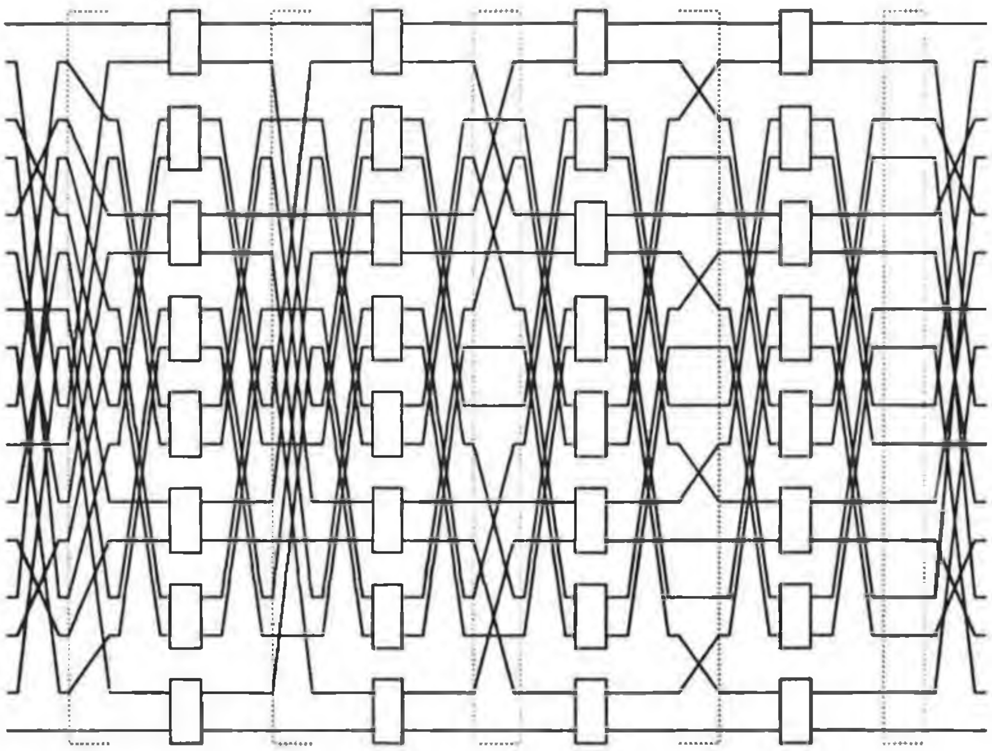


Fig. 4.6: Equivalence of the S-banyan network and its inverse.

This equivalence is shown in Fig. 4.6. Packets are routed to output addresses in the inverse S-banyan, which are the bit-reversal of the corresponding outputs of the S-banyan. This may be corrected for by the simple expedient of presenting the addresses to the switch in bit-reversed order, least significant bit first.

4.1.5 Exact equivalence of the baseline and inverse baseline networks.

The baseline and inverse baseline networks are exactly equivalent. To see this, suppose that the inverse baseline is network A, and the baseline is network B.

By definition,

$$LP_0^A = LP_n^A = ST,$$

$$LP_k^A = PS_{n-k+1}\ ,\ 0<k<n\ ,$$

$$LP_0^B = LP_n^B = ST,$$

and

$$LP_k^B = IPS_{k+1}\ , \qquad 0<k<n\ .$$

It was established in Section 4.1.3 that

$$U_k^B = R_{k+1}\ ,$$

so that

$$zero\ (U_k^B) = k\ .$$

Evidently

$$U_1^A = PS_n\ ,$$

and

$$U_2^A = PS_{n-1}\ PS_n\ .$$

It shall now be shown that

$$U_k^A = R_{n-k}R_n\ .$$

The transformation U_k^A is defined by

$$U_k^A = PS_{n-k+1}\ \ldots\ldots\ldots\ldots\ PS_n\ .$$

It can easily be established that

$$PS_2\, PS_3 \,.....\, PS_k = R_k\,,$$

since

$$R_{k+1}\;\; PS_k = R_k\,.$$

Thus

$$U_k^A = (R_{n-k})^{-1} \,.\, R_n$$

$$= R_{n-k} \,.\, R_n\,.$$

Therefore

$$zero\,(U_k^A) = k$$

$$\Rightarrow T_0^{AB} \;=\; ST.$$

Furthermore

$$T_k^{AB} = U_k^A \;\; T_0^{AB} \;(U_k^B)^{-1}$$

$$= R_{n-k}\, R_n \,.\, ST \,.\, R_{k+1}$$

$$\Rightarrow \quad T_{n-1}^{AB} = R_1 \,.\, R_n \,.\, ST \,.\, R_n \;=\; ST.$$

Hence

$$(T_{in}^{AB})^{-1} \;=\; LP_n^A \; T_{n-1}^{AB} \;(LP_n^B)^{-1}$$

$$= ST \,.\, ST \,.\, ST = ST.$$

Also

$$T_{out}^{AB} = (LP_0^A)^{-1} \; T_0^{AB} \;\; LP_0^B$$

$$=ST \,.\, ST \,.\, ST \;=\; ST.$$

Thus the exact equivalence of the two networks is established, and is demonstrated in Fig. 4.7.

4.1.6 The general case.

It can be seen, by generalising from the above examples, that the necessary and sufficient condition for a network B to be topologically equivalent to the S-banyan, given that all the link permutations in network B are binary link permutations, is that

$$zero\,(U_k^B) \neq \;\; zero\,(U_j^B)$$

where

$$k \neq j,$$

and

$$zero\,(U_k^B) \neq 0,$$

for $1 \leq j < n$ and $1 \leq k < n$.

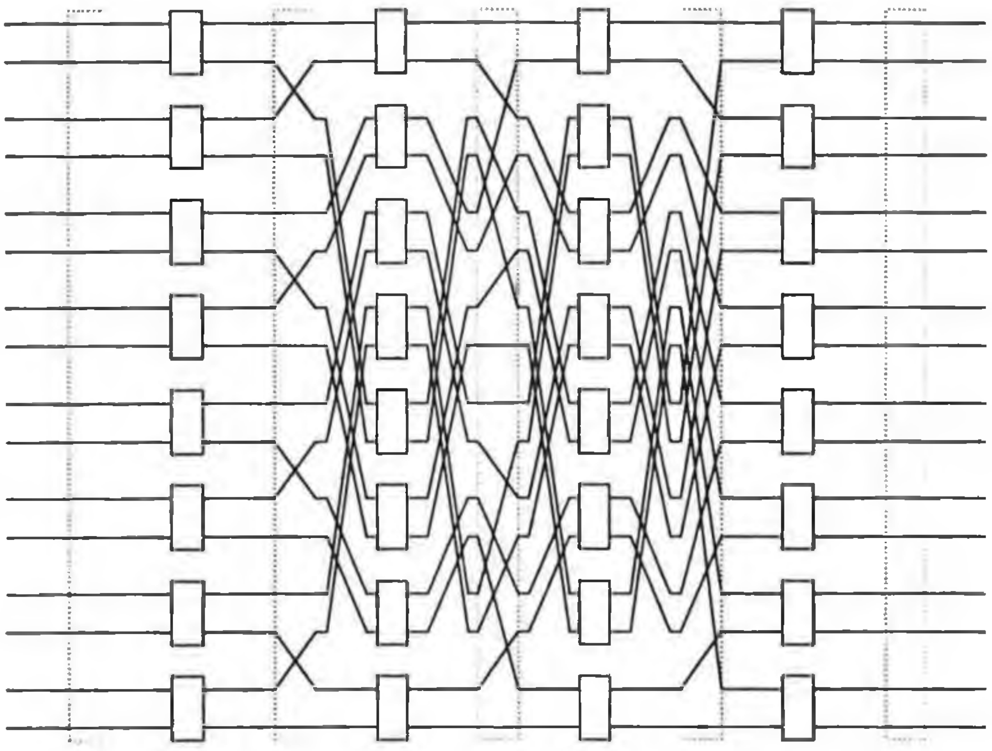


Fig. 4.7: Equivalence of the inverse baseline and baseline networks.

Thus the set of values $\{A_k \mid A_k = zero\,(U_k^B)\,,\, 1 \leq k < n\}$ is a permutation of the set $\{1, 2, 3, ..., n\text{-}1\}$. If a network is to be a binary self-routing network, the graph of possible connections downstream of a switching element must form a tree. This will not be the case for the switch elements in stage k, if zero $(U_k) = 0$, since the two subtrees emanating from any switch element in stage k will then intersect at stage 0. Similarly, if zero (U_k) = zero (U_j) for $k > j$, the subtrees from any switch element in stage k will intersect at stage j. Thus, it is required that

$$zero\,(U_k) \neq zero\,(U_j), k \neq j,\ 1 \leq k < n,\ 1 \leq j < n,$$

and

$$zero\,(U_k) \neq 0\,, \qquad 1 \leq k < n.$$

It follows that any self-routing network whose link permutations are exclusively binary link permutations is equivalent to an S-banyan.

It may be shown that self-routing networks from a more general class, i.e., those which satisfy the 'buddy property', are topologically equivalent to the S-banyan [2]. However, by restricting attention to networks containing only binary link permutations, a method has been found above which establishes the nature, as well as the existence, of the equivalence, allowing an input and/or an output link permutation to be added as appropriate, so as to obtain a network which is exactly equivalent to (for example) an S-banyan.

4.2 Local addresses in binary self-routing networks.

4.2.1 Definition of local address.

As a packet passes through a binary self-routing switch, the address of the switch element port at which it appears at each stage changes. The address of the switch element port at which a packet appears in stage k of a binary self-routing network will be referred to as the packet's *local address* at stage k. At the output side of the switch, the packet's local address equals the destination address, and at the input side of the switch, equals its source address. At intermediate stages, some bits of the local address depend on the source and others on the destination address.

4.2.2 Dependence on source address.

The operation of a binary self-routing network may be regarded as a sequence of transformations on the packet's local address. These transformations may be represented by the link permutations LP_n , LP_{n-1} , ... LP_0 and by the operation $b_0 \rightarrow x$, representing the operation of each switch element, which can interchange adjacent packets. The operation of a switch element cannot be represented by a bijective transformation unless the state of the switch is known, since it transforms the local address of the packet into two possible addresses. The letter x is used to represent any bivalent quantity, which may equal 0 or 1. Table 4.1 shows how the operation of a baseline network may be represented by such transformations.

In this table, $(k)^{in}$ represents the input side of stage k, and $(k)^{out}$ represents the output side of stage k. The x's in the address are governed by the destination address. The bit b_k is replaced by an x at the output of the stage where b_k is shifted into the least significant bit position. The actual values which will be assigned to the x's will depend on the switching rule used for each stage. Defining the function

$$tozero(T) = k \Leftrightarrow b_k \xrightarrow{T} b_0$$

it is apparent that bit b_j is replaced by an x at the output of stage k, where

$$j = tozero\,(F_k)$$

and

$$F_k = LP_n\, LP_{n-1} \,......\, LP_{k+1}, \quad o \leq \; k \leq \; n\text{-}1\,.$$

The value of $tozero(T_k)$ may be calculated in terms of the $zero()$ function and the U_k transformation defined earlier, as follows.

stage	*local address*
input	$b_{n-1}b_{n-2}......\,b_o$
$LP_n \downarrow$	
$(n\text{-}1)^{in}$	$b_{n-1}b_{n-2}......\,b_o$
stage n-1 $\downarrow$	
$(n\text{-}1)^{out}$	$b_{n-1}\;......\;b_1x$
$LP_{n-1} \downarrow$	
$(n\text{-}2)^{in}$	$x\,b_{n-1}\;......b_1$
stage n-2 $\downarrow$	
$(n\text{-}2)^{out}$	$xb_{n-1}......b_2x$
$LP_{n-2} \downarrow$	
$(n\text{-}3)^{in}$	$xxb_{n-1}\;......b_2$
etc. $\downarrow$	

Table 4.1. Dependence of baseline network local address on source address.

Evidently

$$tozero\,(T) = zero\,(T^{-1})\,.$$

Also, from the definitions of the transformations U_k and F_k, it is apparent that

$$F_k\, U_k \;=\; U_n$$

where U_n is as defined earlier, and where U_0 is defined to be a ST transformation.

It follows that

$$j = zero\,(U_k\ U_n^{-1})\,.$$

Thus the bit in the source address which is replaced by an x in the local address at the output of stage k may readily be found if U_k and U_n are known. Defining the function

$$g(k) = tozero(F_k) = zero\,(U_k\ U_n^{-1})\,,$$

it follows that bit $g(k)$ of the source address is replaced by an x in the local address at the output of stage k.

The transformation X_k is defined as

$$X_k\,(Y) = \{Y, Z\}$$

where Z differs from Y only in bit k,

or equivalently, expressing Y in binary notation,

$$X_k\,(b_{n-1}\ \ldots\ldots\ b_0) = b_{n-1}\ \ldots\ldots\ b_{k+1}\ x\ b_{k-1}\ \ldots\ldots\ b_0$$

where $x = 0$ or 1.

Thus the operation of a switching element is equivalent to an $X_0(\ )$ transformation.

The $X_k\,(\ )$ transformation may be considered to be one-to-one if each bit in the address is regarded as having three possible values - 0, 1 or x.

Using the above concepts, it may be shown that where the switching decision made at each stage of the switch is unknown, the local address $A_k^{out}(Y)$ of a packet with input address Y, at the output side of stage k of the switch can be represented as

$$A_k^{out}(Y) = W_k\,F_k\,(Y)$$

where

$$W_k = \prod_{j=k}^{n-1} X_{g(j)}$$

and, at the input side of stage k-1, the local address can be represented as

$$A_{k-1}^{in}(Y) = W_k\,F_{k-1}\,(Y)\,.$$

Applying this formula to the baseline network, where, as has already been shown,

$$U_k = R_{k+1}, \quad 0 \leq k < n,$$

and

$$U_n = R_n,$$

it follows that

$$F_k = U_n U_k^{-1}$$
$$= R_n R_{k+1} .$$

Hence

$$g(k) = tozero\,(R_n R_{k+1}) = zero\,(R_{k+1} R_n) = n\text{-}k\text{-}1 .$$

Thus

$$W_k\,(b_{n-1} \ldots\ldots b_o) = b_{n-1} \ldots\ldots b_{n-k}\, x\, x \ldots\ldots x.$$

Also

$$F_k\,(b_{n-1} \ldots\ldots b_o) = b_o \ldots\ldots b_{n-k-2}\;\; b_{n-1} \ldots\ldots b_{n-k-1} .$$

Hence

$$W_k\; F_k\;\,(b_{n-1} \ldots\ldots b_o) = x\, x \ldots\ldots x\;\, b_{n-1} \ldots\ldots b_{n-k}\, x ,$$

and

$$W_k F_{k-1}\,(b_{n-1} \ldots\ldots b_o) = x\, x \ldots\ldots x\;\, b_{n-1} \ldots\ldots b_{n-k} .$$

Consequently

$$A_k^{out}\,(b_{n-1} \ldots\ldots b_o) = x\, x \ldots\ldots x\;\, b_{n-1} \ldots\ldots b_{n-k}\, x$$

and

$$A_k^{in}\,(b_{n-1} \ldots\ldots b_o) = x\, x \ldots\ldots x\;\, b_{n-1} \ldots\ldots b_{n-k} .$$

These results are consistent with Table 4.1.

4.2.3 Dependence of local address on destination address.

The operation of a binary self-routing network may equally well be regarded as a method for transforming an unknown source address into the required destination address. A baseline network may be described, from this viewpoint, by the transformations in Table 4.2.

It is apparent that the unknown least significant bit of the address at the input side of stage k is transformed to bit d_j of the output address by the switch element, where $j = zero\,(U_k\, LP_0)$.

Define

$$h(k) = zero\,(U_k\,LP_0)$$

where

$$U_0 = ST.$$

stage	*local address*
output	$d_{n-1}d_{n-2}\ldots\ldots d_o$
	$LP_0\uparrow$
$(0)^{out}$	$d_{n-1}d_{n-2}\ldots\ldots d_o$
	stage 0 $\uparrow$
$(0)^{in}$	$d_{n-1}\ldots\ldots d_1x$
	$LP_1\uparrow$
$(1)^{out}$	$d_{n-1}\ldots\ldots d_2xd_1$
	stage 1 $\uparrow$
$(1)^{in}$	$d_{n-1}\ldots\ldots d_2xx$
	etc. $\uparrow$

Table 4.2: Dependence of local address on output address for the baseline network.

It may be shown that in general where the source address of a packet is unknown, but the destination address is known ($d_{n-1}\ \ldots\ldots\ d_o$), the local address of the packet may be written as

$$A_k^{in}\,(d_{n-1}\ldots\ldots d_o)$$

where

$$A_k^{in} = V_k(U_kLP_0)^{-1}$$

and where

$$V_k = \prod_{j=0}^{k} X_{h(j)}\,.$$

and

$$A_{k+1}^{out} = V_k (U_{k+1}\ LP_o)^{-1}.$$

Applying this to the baseline network, where

$$U_k = R_{k+1},\ 0 < k < n,$$

and

$$LP_o = ST,$$

it follows that

$$h(k) = zero(R_{k+1}\,ST) = k,$$

and thus

$$V_k (d_{n-1} \ldots\ldots d_o) = d_{n-1} \ldots\ldots d_{k+1}\ x \ldots\ldots x.$$

Since

$$(U_k\, LP_o)^{-1} = R_{k+1}$$

and

$$R_{k+1} (d_{n-1} \ldots\ldots d_o) = d_{n-1} \ldots\ldots d_{k+1}\ \ d_o \ldots\ldots d_k,$$

it follows that

$$A_k^{in}(d_{n-1} \ldots\ldots d_o) = d_{n-1} \ldots\ldots d_{k+1}\, x \ldots\ldots x.$$

Similarly

$$A_{k+1}^{out}(d_{n-1} \ldots\ldots d_o) = d_{n-1} \ldots\ldots d_{k+2}\, x \ldots\ \ldots x\, d_{k+1}.$$

These results are consistent with Table 4.2

4.2.4 Local address as a function of both source and destination addresses.

Combining these results, for the case where both the source address (b_{n-1} ...... b_o) and destination address (d_{n-1}......d_o) are known, yields the following result for the local address of a packet at stage *k*.

$$A_k^{out} (\, b_{n-1} \ldots\ldots b_o\, ,\, d_{n-1} \ldots\ldots d_o) = W_k\, F_k (\, b_{n-1} \ldots\ldots b_o)$$

$$+\, V_{k-1}\, (U_k\, LP_o)^{-1}\ \ (d_{n-1} \ldots d_o)$$

where additions involving the quantity x are defined by

$$0 + x = 0$$

and

$$1 + x = 1$$

and where

$$V_{-1} = ST.$$

For example, in the case of the baseline network,

$$\begin{aligned} A_k^{out}(b_{n-1} \ldots\ldots b_o, d_{n-1} \ldots\ldots d_o) &= x \ldots x\; b_{n-1} \ldots\ldots b_{n-k}\, x \\ &\quad + d_{n-1} \ldots\ldots d_{k+1}\, x \ldots\ldots x\, d_k \\ &= d_{n-1} \ldots\ldots d_{k+1}\;\; b_{n-1} \ldots\ldots b_{n-k}\, d_k \end{aligned}$$

and

$$\begin{aligned} A_k^{in}(b_{n-1} \ldots\ldots b_o, d_{n-1} \ldots\ldots d_o) &= x\, x \ldots\ldots x\; b_{n-1} \ldots\ldots b_{n-k-1} \\ &\quad + d_{n-1} \ldots\ldots d_{k+1}\, x \ldots\ldots x \\ &= d_{n-1} \ldots\ldots d_{k+1}\;\; b_{n-1} \ldots\ldots b_{n-k-1}\,. \end{aligned}$$

Note that A_k^{in} and A_k^{out} differ only in the least significant bit, as expected.

4.2.5 The routing strategy in self-routing networks.

Note that the switching decision at stage k of the network determines bit $h(k)$ of the destination address. Hence, if the routing decision at stage k of the network is made on the basis of the value of bit $h(k)$ of the routing tag, then, as may easily be shown, the address of the output port to which the packet is routed will equal the routing tag address. Hence, since $h(k) = k$ for the baseline network, this proves that packets are routed to the correct destination if the k-th-bit is used to make the routing decision at stage k.

It will now be formally verified that, if two networks are output-equivalent, packets will be routed, by the same routing strategy, to the same output port in each network.

It is sufficient to prove, for the two output-equivalent networks A and B, that if

$$h_k^A = zero\,(U_k^A\, LP_0^A)\,,$$

and

$$h_k^B = zero\,(U_k^B\, LP_0^B)\,,$$

then

$$h_k^A = h_k^B\,, \qquad 0 \le k < n\,.$$

The proof is straight-forward. Networks A and B are output-equivalent

$$\Leftrightarrow \quad T^{AB}_{out} = ST$$

$$\Leftrightarrow \quad LP^{A}_{0} = T^{AB}_{0} LP^{B}_{0} ,$$

but

$$h^{A}_{k} = zero\,(U^{A}_{k} LP^{A}_{0}) ,$$

i.e.,

$$h^{A}_{k} = zero\,(U^{A}_{k} T^{AB}_{0} LP^{B}_{0}) .$$

Since

$$T^{AB}_{k} = U^{A}_{k} T^{AB}_{0} (U^{B}_{k})^{-1}$$

it follows that

$$U^{A}_{k} T^{AB}_{0} = T^{AB}_{k} U^{B}_{k} .$$

Therefore

$$h^{A}_{k} = zero\,(T^{AB}_{k} U^{B}_{k} LP^{B}_{0}) .$$

Since the networks are topologically equivalent, it follows that

$$zero\,(T^{AB}_{k}) = 0$$

$$\Rightarrow \quad zero\,(T^{AB}_{k} U^{B}_{k} LP^{B}_{0})$$

$$= zero\,(U^{B}_{k} LP^{B}_{0})$$

$$= h^{B}_{k}$$

Thus

$$h^{A}_{k} = h^{B}_{k} .$$

This completes the required proof.

4.2.6 Input equivalence and the *g*(*k*) function.

It may also be shown that if

$$g^{A}_{k} = zero\,(U^{A}_{k} (U^{A}_{n})^{-1}) ,$$

and

$$g^{B}_{k} = zero\,(U^{B}_{k} (U^{B}_{n})^{-1}) ,$$

where the two networks A and B are input-equivalent, i.e., where

$$T_{in}^{AB} = ST,$$

then

$$g_k^A = g_k^B, \qquad 0 \le k < n.$$

Consider the permutations

$$G_k^A = U_k^A (U_n^A)^{-1}$$

and

$$G_k^B = U_k^B (U_n^B)^{-1}.$$

Input equivalence implies that

$$LP_n^B = LP_n^A T_{n-1}^{AB}.$$

Also

$$G_k^A = U_k^A (LP_n^A U_{n-1}^A)^{-1}$$

$$= U_k^A \left(LP_n^B (T_{n-1}^{AB})^{-1} U_{n-1}^A \right)^{-1}$$

$$= U_k^A (U_{n-1}^A)^{-1} T_{n-1}^{AB} (LP_n^B)^{-1},$$

but

$$T_{n-1}^{AB} = U_{n-1}^A T_0^{AB} (U_{n-1}^B)^{-1}$$

$$\Rightarrow \quad G_k^A = U_k^A T_0^{AB} (U_{n-1}^B)^{-1} (LP_n^B)^{-1}$$

$$= U_k^A T_0^{AB} (U_n^B)^{-1}.$$

Since

$$T_k^{AB} = U_k^A T_0^{AB} (U_k^B)^{-1},$$

it follows that G_k^A may be rewritten as

$$G_k^A = T_k^{AB} U_k^B (U_n^B)^{-1}$$

$$= T_k^{AB} G_k^B.$$

Since

$$zero(T_k^{AB}) = 0 ,$$

it follows that

$$zero(G_k^A) = zero(T_k^{AB} G_k^B) = zero(G_k^B) ,$$

i.e.,

$$g_k^A = g_k^B ,$$

4.3. Blocking properties of binary self-routing networks.

4.3.1 Bernabei's condition.

The necessary and sufficient condition for two packets not to collide in a baseline network has been obtained by Bernabei *et al.* [3]. They considered a general case of the baseline network, where the switch elements are of size K by K. Their results for the case $K = 2$ will be considered here. The extension to larger switch element sizes is straightforward.

They introduced the concept of distance between input ports, which can be defined as follows.

Consider the tree formed from the set of paths available to a packet which enters the network from input port I_1. Suppose this tree and the corresponding tree for input port I_2 intersect at stage n-k-1 of the network (and at no higher numbered stage). It follows that, regardless of the destination of packets arriving at input ports I_1 and I_2, collisions between these packets cannot occur prior to stage n-k-1. Thus, no collisions between these packets can occur in stages n-1, n-2, ... n-k , i.e., in the first k switch elements through which each packet passes. If a collision does occur, it occurs at the *output* side of stage n-k-1. The distance between input ports I_1 and I_2 is defined to be k or

$$d_{in}(I_1 , I_2) = k .$$

The distance between output ports is defined in an equivalent way. Suppose the two trees formed from the set of paths via which packets can arrive at output ports O_1 and O_2 intersect at stage k of the network (and at no lower numbered stage). It follows that no collisions can occur between packets which are to be routed to outputs O_1 and O_2 in stages j-1 , j-2 , ... (a total of j stages) regardless of the input ports on which these packets arrived. If a collision does occur, it is at the *input* side of stage j. Thus the distance between outputs O_1 and O_2 is k, or

$$d_{out}(O_1 , O_2) = j .$$

These two measures (of distance between input ports and between output ports) can be used to obtain a necessary and sufficient condition for the path from input port I_1 to output port O_1, not to intersect with the path from input port I_2 to output port O_2.

Packets arriving on input ports I_1 and I_2 can only collide (if at all) at the *output* side of stage n-k-1 and subsequent stages. In other words, if the routes taken by the two packets share links, the shared link nearest to the input side of the switch will be in LP_{n-k-1}. Packets being routed to output ports O_1 and O_2 can only collide (if at all) at the *input* side of stage j or earlier stages, i.e., if the routes to $O_1$and O_2 share links, the shared link nearest to the output side of the switch will be in LP_{j+1}. Thus no collision will occur if

$$j+1 > n - k - 1 ,$$

i.e., if

$$d_{in}\ (I_1 , I_2) + d_{out}\,(O_1 , O_2) \geq n\text{-}1 .$$

This condition has been derived without reference to the topology of the baseline network, and is thus applicable to any binary self-routing network.

4.3.2 Collisions as a function of source address.

Two packets will collide at the output side of stage k if their local addresses match at stage k. This will occur, for source addresses I_1 and I_2 respectively, if

$$W_k\ F_k\ (I_1) = W_k\ F_k\ (I_2) .$$

In particular, those bits in the local address which depend on the source address must match. Thus, writing $I_1 = b_{n-1} \ldots\ldots b_o$ and $I_2 = b'_{n-1} \ldots\ldots b'_o$, a necessary condition for a collision to occur at the output side of stage k is that

$$b_{g(j)} = b'_{g(j)} , \qquad 0 \leq j < k .$$

If, in addition, $b_{g(k)} \neq b'_{g(k)}$, then the packets cannot collide at the output of stage k+1, or any higher numbered stage. These results are summarised in Table 4.3.

4.3.3 Collisions as a function of the destination address.

Two packets will collide at the input side of a stage k, for destination addresses O_1 and O_2 respectively, if

$$V_k\,(U_k\ LP_o)^{-1}\ (O_1) = V_k\,(U_k\ LP_o)^{-1}\ (O_2) .$$

Writing $O_1 = d_{n-1} \ldots\ldots d_o$ and $O_2 = d'_{n-1} \ldots\ldots d'_o$, a necessary condition for such a collision is that

$$d_{h(j)} = d'_{h(j)} , \ k<j<n .$$

collision occurs at output side of stage	distance between inputs	bits in source addresses which need not match	bit in source addresses which must not match	bits in source addresses which must match
n-1	0	-	$b_{g(n-1)}$	$b_{g(n-2)},\ldots,b_{g(0)}$
n-2	1	$b_{g(n-1)}$	$b_{g(n-2)}$	$b_{g(n-3)},\ldots,b_{g(0)}$
● ●	● ●	● ●	● ●	● ●
n-k	k-1	$b_{g(n-1)},\ldots,b_{g(n-k+1)}$	$b_{g(n-k)}$	$b_{g(n-k-1)},\ldots,b_{g(0)}$
● ●	● ●	● ●	● ●	● ●
0	n-1	$b_{g(n-1)},\ldots,b_{g(1)}$	$b_{g(0)}$	-

Table 4.3: Conditions on source address for collisions to occur at each stage.

collision occurs at input side of stage	distance between outputs	bits in destination addresses which need not match	bit in destination addresses which must not match	bits in destination addresses which must match
0	0	-	$d_{h(0)}$	$d_{h(1)},\ldots,d_{h(n-1)}$
1	1	$d_{h(0)}$	$d_{h(1)}$	$d_{h(2)},\ldots,d_{h(n-1)}$
● ●	● ●	● ●	● ●	● ●
k	k-1	$d_{h(0)},\ldots,d_{h(k-1)}$	$d_{h(k)}$	$d_{h(k+1)},\ldots,d_{h(n-1)}$
● ●	● ●	● ●	● ●	● ●
n-1	n-1	$d_{h(0)},\ldots,d_{h(n-2)}$	$d_{h(n-1)}$	-

Table 4.4: Conditions on destination address for a collision to occur.

If, in addition, $d_{h(k)} \neq d'_{h(k)}$, then the packets cannot collide at the input side of stage k-1, or any lower-numbered stage. Table 4.4 is a summary of these results.

4.3.4 Calculation of the distance function.

Barnabei *et al.* [3] described a method for readily computing $d_{in}(I_1 , I_2)$ and $d_{out}(O_1 , O_2)$ for the baseline network. A function is now introduced which implements their method.

The function *match* (x , y) is defined as follows. Suppose $x = b_{n-1}b_{n-2} \ldots\ldots b_o$ and $y = a_{n-1}a_{n-2} \ldots\ldots a_o$. Suppose further that

$$b_j = a_j , \qquad j \geq k + 1 ,$$

and

$$b_k \neq a_k .$$

It follows that

$$match\ (x , y) = k .$$

The definition of the distance function given by Bernabei *et al.* [3] for the baseline network may be written using this function definition as

$$d_{in}\ (I_1 , I_2) = match\ (I_1 , I_2)$$

and

$$d_{out}\ (O_1 , O_2) = match\ (O_1 , O_2) .$$

Bernabei *et al.* [3] did not provide a proof of this result, which may be easily established in the case of the baseline network. A general technique for the calculation of d_{in} and d_{out} for any binary self-routing network may be readily determined from Tables 4.3 and 4.4, by evaluating $g(k)$ and $h(k)$ for the given network.

Consider the case of the baseline network. It has already been shown that

$$U_k = R_{k+1} , \qquad 0 < k < n .$$

Also

$$U_n = LP_n\, U_{n-1} = ST.R_n = R_n .$$

Hence

$$g(k) = zero\ (R_{k+1}\, R_n) = n\text{-}k\text{-}1 ,$$

and so

$$b_{g(n-k)} = b_{k-1} .$$

Substituting this finding into Table 4.3 yields the results in Table 4.5.

collision occurs at output side of stage	distance between inputs	bits in source addresses which need not match	bit in source addresses which must not match	bits in source addresses which must match
n-1	0	-	b_0	$b_1,...,b_{n-1}$
n-2	1	b_0	b_1	$b_2,...,b_{n-1}$
• •	• •	• •	• •	• •
n-k	k-1	$b_0,...,b_{k-2}$	b_{k-1}	$b_k,...,b_{n-1}$
• •	• •	• •	• •	• •
0	n-1	$b_0,...,b_{n-2}$	b_{n-1}	-

Table 4.5: Conditions on the source address for collisions to occur in the baseline network.

This table is consistent with the definition of $d_{in}\,(I_1\ , I_2)$ given by Bernabei *et al.* [3] for the baseline network.

The value of $h(k)$ for the baseline network is also easily obtained. It has already been found that $zero\ (U_k) = k$ and that $LP_0 = ST$. Thus $h(k) = k$.

Consequently, Table 4.4 may be replaced by Table 4.6, in the case of the baseline network.

Once again the result is consistent with the definition presented by Bernabei *et al.* [3].

The formula for calculating distance is not the same for all binary self-routing networks. Consider the tables for the S-banyan network. The values of $h(k)$ and $g(k)$ for the S-banyan network are found below.

By definition

$$h(k) = zero\ (U_k\, LP_0)\ ,$$

collision occurs at output side of stage	distance between outputs	bits in destination addresses which need not match	bit in destination addresses which must not match	bits in destination addresses which must match
0	0	-	d_0	$d_1,\dots,d_{n-1}$
1	1	d_0	d_1	$d_2,\dots,d_{n-1}$
• •	• •	• •	• •	• •
k	k-1	$d_0,\dots,d_{k-2}$	d_{k-1}	$d_k,\dots,d_{n-1}$
• •	• •	• •	• •	• •
n-1	n-1	$d_0,\dots,d_{n-2}$	d_{n-1}	-

Table 4.6: Influence of destination address on the occurrence of collisions in the baseline network.

but

$$LP_0 = ST.$$

Therefore

$$h(k) = zero\ (U_k)\ ,$$

and since

$$U_k = IPS_{k+1}\ ,\quad 1 \leq\ k < n\ ,$$

it follows that

$$h(k)\ =\ k.$$

Also

$$g(k) = zero\ (U_k\ U_n^{-1})$$

but

$$U_n = LP_n\ U_{n-1} = PS_n\ IPS_n = ST.$$

Hence

$$g(k) = zero\ (U_k)\ =\ k.$$

Hence the data in Tables 4.7 and 4.8 are obtained.

collision occurs at output side of stage	distance between inputs	bits in source addresses which need not match	bit in source addresses which must not match	bits in source addresses which must match
n-1	0	-	b_{n-1}	$b_{n-2},\dots,b_0$
n-2	1	b_{n-1}	b_{n-2}	$b_{n-3},\dots,b_0$
• •	• •	• •	• •	• •
n-k	k-1	$b_{n-1},\dots,b_{n-k+1}$	b_{n-k}	$b_{n-k-1},\dots,b_0$
• •	• •	• •	• •	• •
0	n-1	$b_{n-1},\dots,b_1$	b_0	-

Table 4.7: Conditions on the input address for collisions to occur in the S-banyan.

collision occurs at output side of stage	distance between outputs	bits in destination addresses which need not match	bit in destination addresses which must not match	bits in destination addresses which must match
0	0	-	d_0	$d_1,\dots,d_{n-1}$
1	1	d_0	d_1	$d_2,\dots,d_{n-1}$
• •	• •	• •	• •	• •
k	k-1	$d_0,\dots,d_{k-2}$	d_{k-1}	$d_k,\dots,d_{n-1}$
• •	• •	• •	• •	• •
n-1	n-1	$d_0,\dots,d_{n-2}$	d_{n-1}	-

Table 4.8: Conditions on the destination address for collisions to occur in the S-banyan.

Comparison of these results to those for the baseline network indicates that the distance between two outputs is the same in the S-banyan as in the baseline network. However, a permutation must be performed on the input addresses before the input distance can be calculated using the *match*(x, y) function. Specifically, a bit reversal must be performed on the input port addresses.

Calculation of the distance function by this method is tedious. A more elegant method of obtaining the distance functions is described in Section 4.3.5.

4.3.5 Blocking and equivalence.

A systematic procedure for determining the bit permutation which must be performed on the input or output port addresses for a given binary self-routing network, prior to application of the formula developed by Bernabei *et al.* [3], for the calculation of input and output distances, will now be described.

The procedure is to test for exact equivalence between the baseline network (network A) and the network of interest (network B), by determining T_{in}^{AB} and T_{out}^{AB}. The two networks shown in Fig. 4.8 are then exactly equivalent.

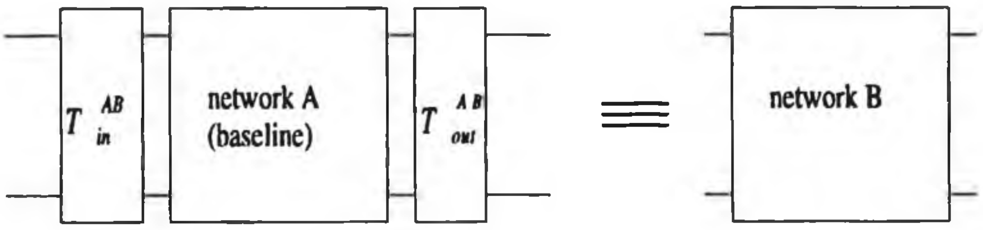


Fig. 4.8: Blocking and equivalence.

Thus, applying the transformation T_{in}^{AB} to input addresses of network B provides the corresponding input addresses for the baseline network. Applying the transformation $(T_{out}^{AB})^{-1}$ to the output addresses of network B similarly results in the corresponding output addresses for the baseline network. Hence the two distance functions may be calculated for network B as

$$d_{in}(I_1, I_2) = match\ (T_{in}^{AB}(I_1), T_{in}^{AB}(I_2))$$

and

$$d_{out}(O_1, O_2) = match\ ((T_{out}^{AB})^{-1}(O_1), (T_{out}^{AB})^{-1}(O_2)) .$$

For example, if network B is an S-banyan, then $T_{in}^{AB} = R_n$ and $T_{out}^{AB} = ST$.

Hence,

$$d_{in}(I_1, I_2) = match\,(R_n(I_1), R_n(I_2)$$

and

$$d_{out}\,(O_1, O_2) = match\,(O_1, O_2)\,.$$

These formulae produce results identical to those obtained by calculating $h(k)$ and $g(k)$ for the S-banyan network. They also show that the blocking properties of two networks are the same if and only if they are exactly equivalent. This result is not surprising, since it was already established that the $h(k)$ and $g(k)$ functions correspond in exactly equivalent networks, in Sections 4.2.5 and 4.2.6.

The d_{in} and d_{out} functions may be found without any calculation for an inverse network, if the corresponding functions for the forward network are known.

Suppose the baseline network A, is exactly equivalent to the network of interest (network B) if preceded by a link transformation T_{in}^{AB} and followed by a link transformation T_{out}^{AB}, as in Fig. 4.8.

It follows that the inverse of network B is exactly equivalent to an inverse baseline network preceded by a link transformation $(T_{out}^{AB})^{-1}$ and followed by a link permutation $(T_{in}^{AB})^{-1}$. This is illustrated in Fig. 4.9 (a). Since the inverse baseline network is exactly equivalent to the baseline network, the equivalence in Fig. 4.9(b) is obtained.

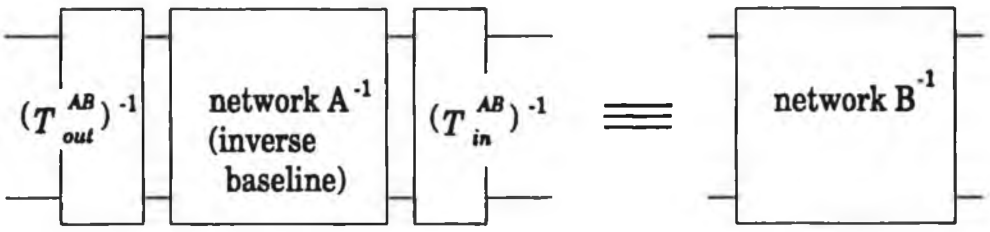
It follows that for the inverse network of B,

$$d_{in}\,(I_1, I_2) = match\,((T_{out}^{AB})^{-1}\,(I_1), (T_{out}^{AB})^{-1}\,(I_2))$$

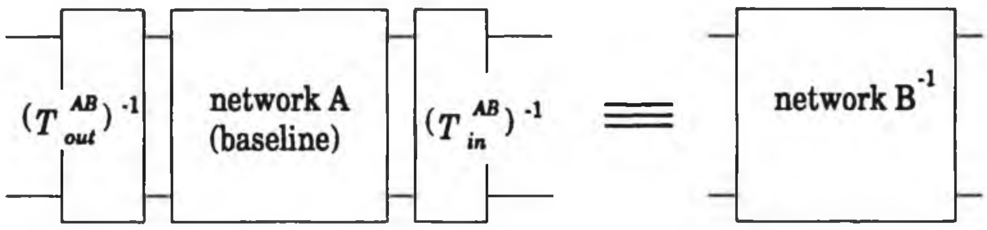
and

$$d_{out}(O_1, O_2) = match\,(T_{in}^{AB}\,(O_1), T_{in}^{AB}\,(O_2))\,.$$

Notice that the input distance function for the inverse network equals the output distance function for the original network, and *vice versa*. This follows since the inverse baseline and baseline networks are exactly equivalent. This property of the baseline network explains why the d_{in} and d_{out} functions are identical for the baseline.



(a) Equivalence with inverse baseline;



(b) Equivalence with baseline.

Fig. 4.9: Equivalence for inverse networks.

4.4 Non-blocking conditions for the S-banyan and its inverse.

4.4.1 The S-banyan.

The results obtained in Section 4.3.5 can be used to prove that a concentrated list of packets, sorted by destination and without repeated addresses, may be routed through an S-banyan network without blocking. This result also holds for the omega network, since the two networks are exactly equivalent. Two proofs of this result have been previously published, by Lee [4] and Narasimha [5]. Narasimha's proof was based on the equivalence between the S-banyan and the Batcher bitonic sorter. He used this equivalence to show that a bitonic list, without omissions or repetitions, could be routed through an S-banyan without blocking. The extension of this result to apply to a concentrated monotonic list, with omissions but without repetitions, is straightforward.

Lee's approach (attributed to Bechmann) was based upon the use of a special numbering scheme, labelling each switch element, and a proof by contradiction. It commenced with the observation that, where the packets at input ports I_1 and I_2 request destinations O_1 and O_2 respectively, and where the requests form a concentrated monotonic list without repetitions, then

$$I_2 - I_1 \leq O_2 - O_1 .$$

It shall now be shown that this condition implies that

$$d_{in}(I_1, I_2) + d_{out}(O_1, O_2) \geq n\text{-}1 .$$

This result strictly only applies when the list is sorted in ascending order, and when $I_2 > I_1$. More generally, it may be stated, for an arbitrary pair of requests, that if

$$|\ I_2 - I_1| \ \leq \ |\ O_2 - O_1|$$

then

$$d_{in}(I_1, I_2) + d_{out}(O_1, O_2) \geq n\text{-}1 .$$

The proof of this statement is as follows.

Without loss of generality, it is assumed that $I_2 > I_1$ and $O_2 > O_1$.

Suppose that

$$d_{out}(O_1, O_2) = r ,$$

i.e., that

$$match\,(O_1, O_2) = r .$$

This implies that O_1 and O_2 differ in the r-th bit, but correspond in more significant bits. Since O_2 is greater than O_1, it follows that they may be written as

$$O_2 = a.\,2^{r+1} + 1.2^r + b$$

where

$$0 \leq b < 2^r$$

and

$$O_1 = a.\,2^{r+1} + 0.2^r + c$$

where

$$0 \leq c < 2^r .$$

Hence

$$O_2 - O_1 = 2^r + b - c.$$

Clearly

$$-2^r < b - c < 2^r,$$

so

$$0 < O_2 - O_1 < 2^{r+1}.$$

If it is assumed that

$$d_{in}\ (I_1\ ,I_2)\ =\ s$$

then

$$match\ (R_n\ (I_1)\ ,R_n\ (I_2))\ =\ s.$$

Because of the bit reversal (R_n) it follows that I_1 and I_2 differ in the $(n\text{-}s\text{-}1)$-th bit, but match in the less significant bits. Since $I_2 > I_1$, it follows that

$$I_2\ =\ e.\,2^{n-s} + \varepsilon\ .2^{n-s-1} + f$$

with

$$f < 2^{n-s-1}\ ,$$

and

$$\varepsilon = 0\ \text{ or } 1.$$

Also

$$I_1\ =\ g.\ 2^{n-s} + (1\text{-}\varepsilon).\ 2^{n-s-1} + f.$$

Hence

$$I_2 - I_1 = (e\text{-}g)\,x\,2^{n-s} + (\varepsilon - 1 + \varepsilon).\ 2^{n-s-1} + f - f$$

$$=\ [2(e - g + \varepsilon) - 1].\ 2^{n-s-1},$$

but

$$I_2 > I_1$$

$$\Leftrightarrow\ 2\,(e\text{-}g) + \varepsilon - 1 > 0$$

$$\Leftrightarrow\ 2\,(e\text{-}g) + \varepsilon - 1 \geq\ 1$$

$$\Rightarrow\ I_2 - I_1 \geq\ 2^{n-s-1}$$

$$\Rightarrow\ \frac{1}{I_2 - I_1} \leq 2^{s+1-n}\ .$$

Combining the two inequalities yields the following result.

$$\frac{O_2 - O_1}{I_2 - I_1} < 2^{r+1+s+1-n}\ ,$$

i.e.,

$$\frac{O_2 - O_1}{I_2 - I_1} < 2^{r+s+2-n}\ ,$$

but, by assumption,

$$1 \leq \frac{O_2 - O_1}{I_2 - I_1} .$$

Hence

$$1 < 2^{r+s+2-n}$$

$$\Leftrightarrow \quad 0 < r + s + 2 - n$$

$$\Leftrightarrow \quad r + s > n - 2$$

$$\Leftrightarrow \quad r + s \geq n - 1,$$

Thus, it has been shown that $| O_2 - O_1 | \geq | I_2 - I_1 |$ is a sufficient condition to ensure that

$$d_{in} (I_1 , I_2) + d_{out} (O_1 , O_2) \geq n\text{-}1 ,$$

which is the necessary and sufficient condition for the two packets not to collide.

4.4.2 The Inverse S-banyan.

It will now be shown that a sufficient condition for this network to be non-blocking is:

$$| O_2 - O_1 | \leq | I_2 - I_1 |$$

where O_1 , O_2 are the destinations of two packets entering on input ports I_1 and I_2 respectively. This in turn proves the assertion of Kim & Leon-Garcia [6] that the inverse S-banyan is non-blocking when consecutive inputs (disregarding idle inputs) request consecutive outputs. Thus, the inverse S-banyan can be used as a non-blocking concentrator. This application was first suggested in the Starlite switch [7] (in the case of the inverse omega network), although no proof of the non-blocking criterion was given.

Defining the inverse S-banyan (indirect binary n-cube) network to be network B, and the baseline network to be network A, it can be readily shown that $T_{in}^{AB} = ST$ and $T_{out}^{AB} = R_n$. Hence

$$d_{in} (I_1 , I_2) = match \ (I_1 , I_2)$$

and

$$d_{out} (O_1 , O_2) = match \ (R_n (O_1) , R_n (O_2) .$$

Thus the method of calculation of d_{in} is identical to the method of calculation of d_{out} for the S-banyan, and *vice versa*. It follows that blocking will not occur if

$$| O_2 - O_1 | \le | I_2 - I_1 | .$$

This condition is satisfied for all packets if the network is used as a concentrator, where the destination address of each packet equals the number of active packets on lower-numbered input ports.

4.5 Synthesis of binary self-routing networks.

4.5.1 An example of network synthesis.

It may often prove necessary to synthesise a network exactly equivalent to a reference network, but using building blocks of another type. It has already been shown in Section 4.1.1 how one network may be made exactly equivalent to another of the same dimensions, by re-ordering the inputs and/or the outputs. A more general technique for ensuring exact equivalence is demonstrated below, by means of an example.

The example considered is shown in Fig. 4.10. It is a 16-input network which has been designed to be exactly equivalent to an S-banyan network, and is defined by the link permutations

$$LP_0 = ST,$$

$$LP_1 = PS_4S_3 ,$$

$$LP_2 = R_4PS_4S_4 ,$$

$$LP_3 = (PS_4)^2S_2 ,$$

and

$$LP_4 = PS_4 .$$

Note that

$$LP_0 (b_3 \, b_2 \, b_1 \, b_0) = b_3 \, b_2 \, b_1 \, b_0 ,$$

$$LP_1 (b_3 \, b_2 \, b_1 \, b_0) = b_2 \, b_3 \, b_0 \, b_1 ,$$

$$LP_2 (b_3 \, b_2 \, b_1 \, b_0) = b_0 \, b_2 \, b_3 \, b_1 ,$$

$$LP_3 (b_3 \, b_2 \, b_1 \, b_0) = b_1 \, b_0 \, b_2 \, b_3 ,$$

and

$$LP_4 (b_3 \, b_2 \, b_1 \, b_0) = b_2 \, b_1 \, b_0 \, b_3 .$$

Hence

$$U_0 (b_3 \, b_2 \, b_1 \, b_0) = b_3 \, b_2 \, b_1 \, b_0 ,$$

$$U_1 (b_3 \, b_2 \, b_1 \, b_0) = b_2 \, b_3 \, b_0 \, b_1 ,$$

$$U_2\,(b_3\,b_2\,b_1\,b_0) \;=\; b_2\,b_0\,b_1\,b_3\,,$$

and

$$U_3\,(b_3\,b_2\,b_1\,b_0) \;=\; b_0\,b_3\,b_2\,b_1\,.$$

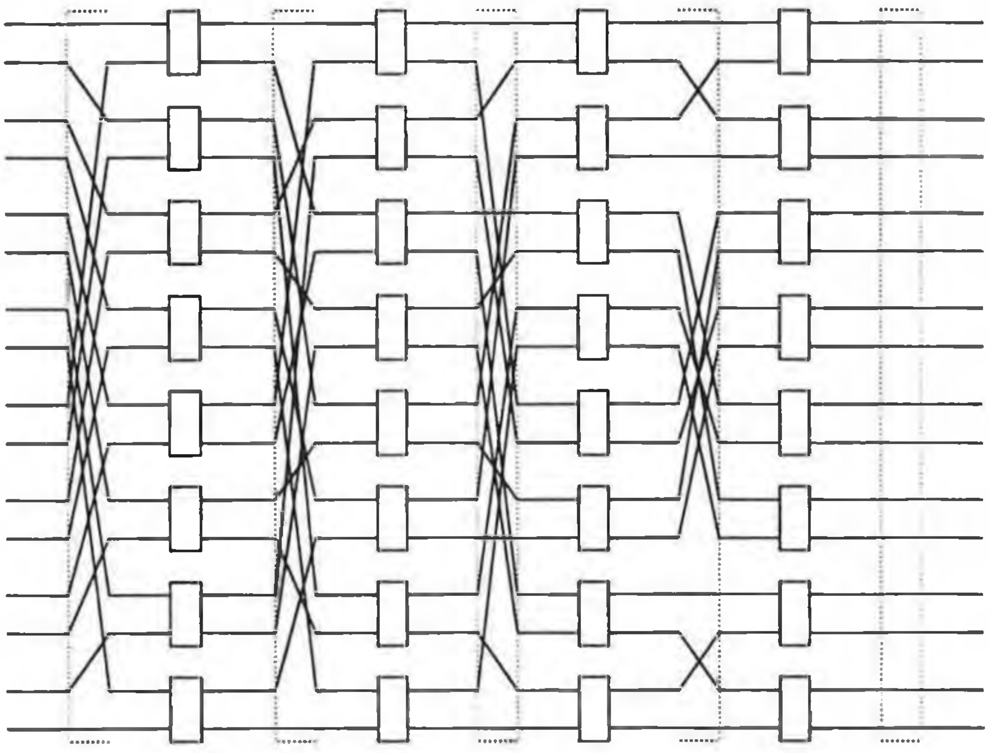


Fig. 4.10: A network exactly equivalent to an S-banyan.

Link permutations LP_0 through LP_3 have been chosen to ensure output equivalence with the S-banyan. Specifically, they have been chosen so that $h(k) = k$ for $0 \le k < 4$. Since $LP_0 = ST$, it follows that they have been chosen so that

$$zero\,(U_3) \;=\; 3\,,$$

$$zero\,(U_2) \;=\; 2\,,$$

and

$$zero\,(U_1) \;=\; 1.$$

Note that a number of binary link permutations satisfy these requirements. Hence there is considerable freedom in the choice of LP_0 through LP_3. Once these have been chosen, the form of LP_4 is dictated by the requirement of input equivalence. Suppose the network in Fig. 4.10 is network A, and network B is the S-banyan. Note that

$$U_3^A = U_3^B = IPS_4\,,$$

and that

$$LP_4^B = PS_4 .$$

By construction,

$$T_0^{AB} = ST.$$

Hence

$$T_3^{AB} = U_3^A . ST . (U_3^B)^{-1}$$

$$\Rightarrow \quad T_3^{AB} = ST.$$

So

$$(T_{in}^{AB})^{-1} = LP_4^A . T_3^{AB} . (LP_4^B)^{-1} ,$$

i.e.,

$$ST = LP_4^A . ST . IPS_4 .$$

Hence

$$LP_4^A = PS_4 .$$

It may easily be verified that

$$T_{out}^{AB} = ST . ST . ST = ST.$$

This confirms that the above network is exactly equivalent to a 16-input S-banyan.

4.5.2 Constructing a 2^{2n}-input S-banyan from 2^{n+1} 2^n-input S-banyans.

A more practical application where it is necessary to ensure exact equivalence of a new network to a reference network occurs when a large S-banyan is to be constructed from smaller modules, themselves S-banyans. Since cascading two n-stage switches produces a $2n$-stage switch, and since an n-stage banyan has 2^n inputs, it follows that a switch with 2^{2n} inputs can be formed using 2^{n+1} n-stage S-banyans, as shown in Fig. 4.11.

The link permutations T_{in}, T_{mid}, and T_{out} in Fig. 4.11 must be chosen to ensure exact equivalence to a $2n$-stage S-banyan. The allowable permutations for T_{mid} will be considered below, in the special case where $T_{out} = ST$.

Note that output equivalence can be obtained with $T_{out} = ST$

$$\Leftrightarrow \quad zero\,(U_k) = k , \quad 1 \leq k < 2n .$$

The link permutations describing the network in Fig. 4.11 are

$$LP_o = ST \,.\, T_{out} = ST ,$$

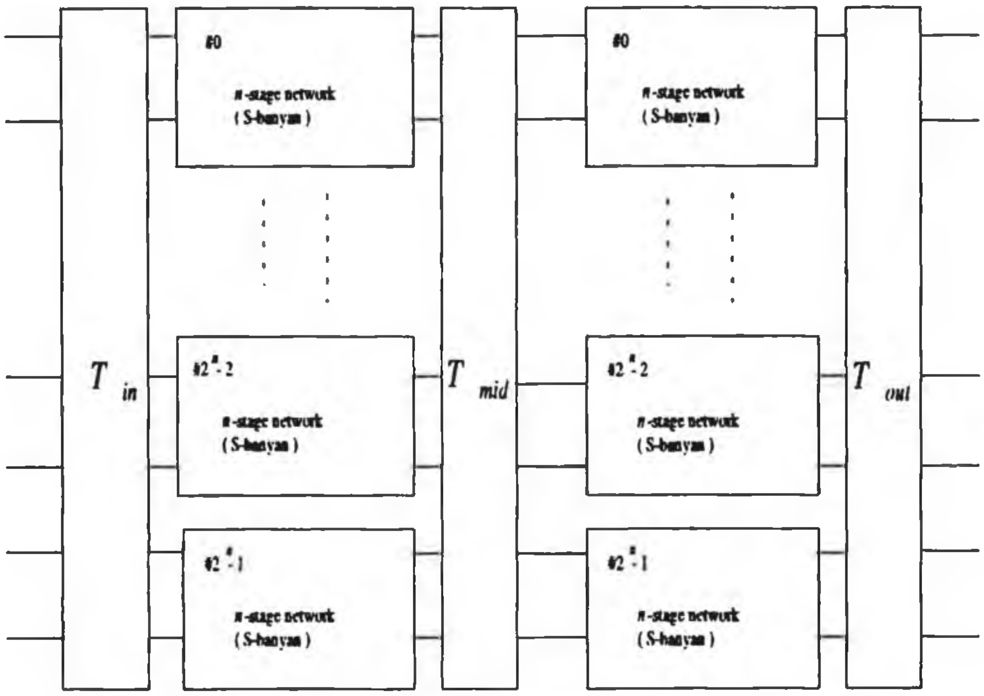
$$LP_k = S_{k+1} , \quad 0 \leq k < n ,$$

$$LP_n = T_{mid} \cdot PS_n ,$$

$$LP_k = S_{k-n+1} , \qquad n \leq k < 2n ,$$

and

$$LP_{2n} = T_{in} \cdot PS_n .$$



#0
n-stage network (S-banyan)
#0
n-stage network (S-banyan)
T_{in}
#2^n-2
n-stage network (S-banyan)
T_{mid}
#2^n-2
n-stage network (S-banyan)
T_{out}
#2^n-1
n-stage network (S-banyan)
#2^n-1
n-stage network (S-banyan)

Fig. 4.11: Synthesising a large network from smaller modules.

It is already known that

$$U_k = IPS_{k+1} , 1 \leq k < n .$$

Hence

$$\begin{aligned} U_n &= LP_n \, U_{n-1} \\ &= LP_n IPS_n \\ &= T_{mid} \cdot PS_n \,.\, IPS_n \\ &= T_{mid} \cdot \end{aligned}$$

Thus it is required to choose T_{mid} such that

$$zero\,(U_k) \;=\; k\,, \qquad n \leq k < 2n$$

Note that

$$U_{n+1} \;=\; S_2\,U_n \;=\; S_2\,T_{mid}\,,$$

and

$$U_{n+k} \;=\; IPS_{k+1}\,U_n \;=\; IPS_{k+1}\,T_{mid}\,, 1 \leq k < n\,.$$

Thus, for example,

$$U_{2n\text{-}1} \;=\; IPS_n U_n \;=\; IPS_n\,T_{mid}\,.$$

Hence it is required that

$$zero(U_n) = zero\,(T_{mid}\,) \;=\; n\,,$$

and

$$zero(U_k) = zero\,(IPS_{k+1\text{-}n}\;\cdot\;T_{mid}) \;=\; k\,, \qquad n < k < 2n\,,$$

in order to obtain output equivalence.

However,

$$zero\,(T_{mid}) \;=\; n \Leftrightarrow \quad b_0 \xrightarrow{T_{mid}} b_n\,.$$

Also

$$b_o \xrightarrow{IPS_2} b_1$$

$$\Rightarrow \quad zero\,(IPS_2.T_{mid}) \;=\; n{+}1 \;\Leftrightarrow\; b_1 \xrightarrow{T_{mid}} b_{n+1}\,,$$

and

$$b_o \xrightarrow{IPS_{k+1}} b_k$$

$$\Rightarrow \quad zero\,(IPS_{k+1}.T_{mid}) \;=\; k{+}n \;\Leftrightarrow\; b_k \xrightarrow{T_{mid}} b_{k+n}\,, \qquad 1 \leq k < n\,,$$

and so, for example,

$$b_o \xrightarrow{IPS_n} b_{n\text{-}1}$$

$$\Rightarrow \quad zero\,(IPS_n.T_{mid}) \;=\; 2n\text{-}1 \quad b_{n\text{-}1} \xrightarrow{T_{mid}} b_{2n\text{-}1}\,.$$

The necessary condition on T_{mid} is thus

$$T_{mid} \;=\; (PS_{2n})^n \;\cdot\; X_n$$

where X_n is some transformation which affects only the n least significant bits.

A check for input equivalence is now performed. Network A is the network shown in Fig. 4.11. Network B is a $2n$-stage S-banyan. T_{in}^{AB} is now determined.

$$(T_{in}^{AB})^{-1} = LP_{2n}^{A} T_{2n-1}^{AB} \; (LP_{2n}^{B})^{-1}$$

where

$$LP_{2n}^{A} = PS_n ,$$

$$LP_{2n}^{B} = PS_{2n} ,$$

and

$$T_{2n-1}^{AB} = U_{2n-1}^{A} \; T_0^{AB} \; (U_{2n-1}^{B})^{-1}.$$

By construction T_0^{AB} = ST.

Also

$$U_{2n-1}^{A} = IPS_n \, U_n^{A} = IPS_n \, (PS_{2n})^n \; X_n$$

and

$$U_{2n-1}^{B} = IPS_{2n} .$$

It follows that

$$T_{2n-1}^{AB} = IPS_n \, (PS_{2n})^n \; X_n . ST . PS_{2n} .$$

If $T_{in}^{AB} = ST$, X_n must be chosen such that

$$ST = T_{in} \, PS_n \, IPS_n \; (PS_{2n})^n \; X_n \; . \; PS_{2n} \; . \; IPS_{2n}$$

$$= T_{in} . ST \; . \; (PS_{2n})^n \; X_n \; . \; ST$$

$$\Rightarrow \; X_n = (IPS_{2n})^n \, (T_{in})^{-1}.$$

The transformation X_n can only affect the n least significant bits. Hence the transformation $(T_{in})^{-1}$ must map

$$b_0 \rightarrow \; b_n ,$$

$$b_1 \rightarrow \; b_{n+1} ,$$

$$.$$

$$.$$

$$b_{n-1} \rightarrow \; b_{2n-1} ,$$

and

$$\{b_n , b_{n+1} , \dots , b_{2n-1}\} \rightarrow \; \{b_0 , b_1 , \dots , b_{n-1}\}.$$

Thus it may be rewritten

$$(T_{in})^{-1} = (PS_{2n})^n \, . \, Y_n ,$$

where Y_n affects only the n least significant bits. Hence

$$X_n = (IPS_{2n})^n \, (PS_{2n})^n \, . \, Y_n \, .$$

Thus X_n and Y_n are related by

$$X_n = Y_n$$

and so

$$(T_{in})^{-1} = (PS_{2n})^n \, X_n \, ,$$

or

$$T_{in} = X_n^{-1} \, (IPS_{2n})^n.$$

Thus $2N$ N-input S-banyans can be combined to form a network exactly equivalent to an N^2-input S-banyan (where N=2^n) by using link permutations

$$T_{out} = ST \, ,$$

$$T_{mid} = (PS_{2n})^n \, . \, X_n \, ,$$

and

$$T_{in} = X_n^{-1} \, (IPS_{2n})^n \, ,$$

where X_n is any binary permutation which affects only the n least significant bits. Notice that $T_{in}^{-1} = T_{mid}$. Suppose

$$X_n = IPS_n \, .$$

Then

$$T_{mid} = (PS_{2n})^n \, . \, IPS_n$$

and

$$T_{in} = PS_n \, (IPS_{2n})^n.$$

The resulting circuit, shown in Fig. 4.12 for the case n=3, is exactly equivalent to a $2n$-input S-banyan.

4.6 The re-routing banyan.

4.6.1 Definitions.

The utility of the techniques introduced in this chapter for analysing binary self-routing networks shall now be demonstrated by applying them to a relatively complex network, namely the re-routing banyan [8].

Recall that an m-stage re-routing banyan with 2^n inputs is defined by the following link permutations

$$LP_m = LP_0 = ST,$$

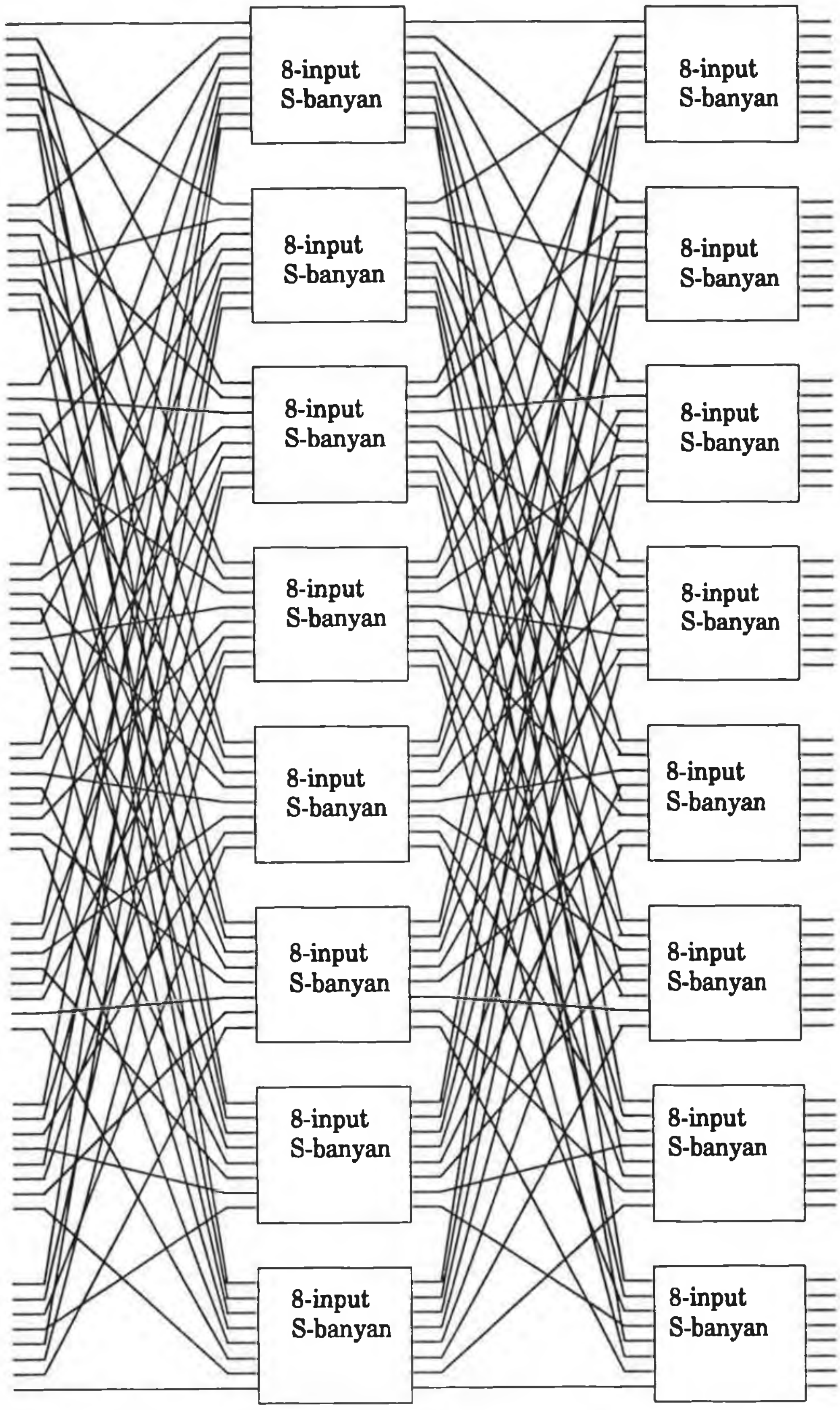


Fig. 4.12: A network exactly equivalent to a 64-input S-banyan.

and

$$LP_k = S_{n - (m - k - 1) \, mod \, (n - 1)}, \qquad 0 < k < m,$$

or, equivalently, by

$$LP_m = LP_0 = ST,$$

and

$$LP_k = S_{2 + (n - 1 + k - t) \, mod \, (n - 1)}, \; 0 < k < m,$$

where

$$t = m \, mod \, (n\text{-}1) \, .$$

The properties of the sub-network consisting of n consecutive stages of the re-routing banyan are of interest. It is useful, in this regard, to consider the B_r network, defined by

$$LP_0{}^{(r)} = LP_n{}^{(r)} = ST,$$

$$LP_k{}^{(r)} = \begin{cases} S_{n+k-r}, & 0 < k \leq r, \\ S_{k+1-r}, & r < k < n, \end{cases}$$

or, equivalently, by

$$LP_0{}^{(r)} = LP_n{}^{(r)} = ST,$$

and

$$LP_k{}^{(r)} = S_{2 + (k - r + n - 2) \, mod \, (n - 1)}, \qquad 0 < k < n,$$

where

$$0 \leq r \leq n - 1 \, .$$

Note, in particular, that $LP_r{}^{(r)} = S_n$ for $0 < r \leq n - 1$ and that the B_0 and B_{n-1} networks are identical.

It may be readily shown that the n-stage sub-network comprising stages j through $j + n$- 1 of the rerouting banyan is a B_r network, with $r = (m - j - 1) \, mod \, (n - 1)$. Hence, a cell which passes through n stages of a rerouting banyan without deflection has equivalently passed through a B_r network, with the appropriate value of r.

4.6.2 Routing in the rerouting banyan.

The routing properties of the B_r network are now investigated. The B_{n-1} network needs no further consideration, since it is identical to an S-banyan, apart from the absence of a perfect shuffle at the input side of the switch. The other B_r networks are considered below.

$U_k^{(r)}$ is defined as

$$U_k{}^{(r)} = LP_k{}^{(r)}\, LP_{k-1}{}^{(r)} \ldots\ldots LP_1{}^{(r)}.$$

It follows that

$$U_k{}^{(r)} = \begin{cases} S_{n+k-r}\, S_{n+k-r-1} \ldots\ldots S_{n+1-r}\,, & 0 < k \leq r\,, \\ S_{k+1-r}\, S_{k+1-r-1} \ldots S_2\, S_n\, S_{n-1} \ldots S_{n+1-r}, & r<k<n\,. \end{cases}$$

It has previously been determined that

$$IPS_k = S_k S_{k-1} \ldots\ldots S_2\,.$$

An expression for $S_nS_{n-1} \ldots\ldots S_{k-1}$, may be found by observing that

$$\begin{aligned} S_nS_{n-1} \ldots\ldots S_{k-1} &= (S_n\,, S_{n-1}\,, \ldots\ldots S_2)\ (S_k\,, S_{k-1}\,, \ldots\ldots S_2)^{-1} \\ &= IPS_n\,.\,(IPS_k)^{-1} \\ &= IPS_n\, PS_k\,. \end{aligned}$$

Hence it follows that

$$U_k{}^{(r)} = \begin{cases} IPS_{n+k-r}\ PS_{n-r}\,, & 0 < k \leq r\,, \\ IPS_{k+1-r}\, IPS_n\, PS_{n-r}\,, & r<k<n\,. \end{cases}$$

It may easily be shown that

$$zero(U_k{}^{(r)}) = \begin{cases} k-r+n-1\,, & 0 < k \leq r\,, \\ k-r\,, & r<k<n\,. \end{cases}$$

or, equivalently, that

$$zero\ (U_k{}^{(r)}) = 1 + (k - r + n - 2)\ mod\ (n - 1)\,, \quad 0 < k < n\,.$$

The routing strategy for the B_r network can be determined by calculating $h_k^{(r)}$, defined by

$$h_k{}^{(r)} = \begin{cases} zero(U_k^{(r)}LP_0)\,, & 0 < k < n\,, \\ zero(LP_0), & k=0. \end{cases}$$

Since $LP_o = ST$, it follows that

$$h_k{}^{(r)} = \begin{cases} 1 + (k - r + n - 2)\ mod\ (n - 1)\,, & 0 < k < n\,, \\ 0, & k=0. \end{cases}$$

Successful routing of a packet to the destination given by its routing tag requires that switching in the k-th stage of the B_r network be based on the value of bit $h_k^{(r)}$ of the routing tag.

Now consider the routing strategy in the rerouting banyan proper. Attention is focused on the cells present at stage s of the rerouting banyan. These cells will have passed through varying numbers of stages without deflection. Hence they may be considered to be in various B_r sub-networks.

Some cells are in stage 0 of a B_r sub-network comprising stages $s + n - 1$ through s of the rerouting banyan, and thus $r = (m - s - 1)\ mod\ (n - 1)$; other cells are in stage 1 of a B_r sub-network comprising stages $s + n - 2$ through $s - 2$ of the rerouting banyan, and thus $r = (m - s)\ mod\ (n - 1)$; and so on. In general, all cells in stage s of the rerouting banyan are equivalently in stage k of a B_r sub-network with

$$r = (m - s - 1 + k)\ mod\ (n - 1)\ ,$$

for some value of k in the range $0 \leq k < n$.

Substituting $r = (m - s - 1 + k)\ mod\ (n - 1)$ into the expression for $h_k^{(r)}$, it is found that

$$h_k^{(r)} = \begin{cases} 1 + (k - (m - s - 1 + k)\ mod.\ (n - 1) + n - 2)\ mod\ (n - 1)\ , & 0 < k < n\ , \\ 0, & k=0\ , \end{cases}$$

i.e.,

$$h_k^{(r)} = \begin{cases} 1 + (k - m + s + 1 - k + n - 2)\ mod\ (n - 1)\ , & 0 < k < n\ , \\ 0, & k=0. \end{cases}$$

Simplifying this expression results in

$$h_k^{(r)} = \begin{cases} 1 + (s - m + n-1)\ mod\ (n - 1)\ , & 0 < k < n\ , \\ 0, & k=0. \end{cases}$$

Note that $h_k^{(r)}$ is independent of k unless $k = 0$. Thus the correct routing strategy for the rerouting banyan is as follows.

Switching in stage s of the rerouting banyan should be based on the value of bit $(1 + (s - m + n - 1)\ mod\ (n - 1))$ of the routing tag, unless the cell has passed through n-1 stages without deflection. In this case, switching is based on the least significant bit of the routing tag.

At first sight the derived routing strategy appears to differ from that proposed by Urushidani [8]. His routing strategy is based on bit $(n - 1 - (m - s - 1)\ mod\ (n - 1))$ of the routing tag; both strategies are identical since $1+(s\text{-}m+n\text{-}1)\ mod\ (n\text{-}1) = n\text{-}1\text{-}(m\text{-}s\text{-}1)\ mod\ (n\text{-}1)$.

4.6.3 Equivalence in the B_r network.

The routing strategy for the B_r network could have been found more easily by testing for output equivalence with the S-banyan.

Suppose the S-banyan is network A, and the B_r network is network B. From previous calculations, it is known that:

$$zero(U_k^B) = \begin{cases} 1 + (k - r + n - 2) \bmod (n-1), & 0 < k < n, \\ 0, & k = 0. \end{cases}$$

and

$$zero\,(U_k^A) = k\,,\ 0 \leq k < n\,.$$

Hence T_0^{AB} is defined by

$$b_o \rightarrow b_o\,,$$

$$b_k \rightarrow b_{k-r}\,,\quad n > k > r\,,$$

$$b_k \rightarrow b_{n-1-r+k}\,,\quad 0 < k \leq r\,.$$

This is a cyclic shift right r times on all but the least significant bit. Hence

$$T_0^{AB} = S_n\,(IPS_{n-1})^r\,S_n\,.$$

Therefore

$$T_{out}^{AB} = (LP_0^A)^{-1}\,T_0^{AB}\,LP_0^B$$

$$= ST\,.\,T_0^{AB}\,.\,ST$$

$$= T_0^{AB}\,.$$

Hence T_{out}^{AB} is a cyclic shift right r times on all but the least significant bit of the output link address.

Network A (the S-banyan) is output-equivalent to network B (the B_r network) followed by a link permutation $(T_{out}^{AB})^{-1}$. Hence, to route a packet to output O_1 in the B_r network, the routing strategy must be that which, in the S-banyan, results in the destination $O_2 = (T_{out}^{AB})^{-1}\,(O_1)$.

Suppose

$$O_1 = b_{n-1} \cdots\cdots b_o\,.$$

then, since $(T_{out}^{AB})^{-1}$ is a cyclic shift left r times on all but the least significant bit of the link address, it follows that

$$O_2 = b_{n-r+1} \cdots\cdots b_1\,b_{n-1} \cdots\cdots b_{n-r-1}\,b_{n-r}\,b_o\,.$$

Since switching in the k-th stage of the S-banyan should be based on the value of the k-th bit of O_2, it follows that the bit which should determine the switch element output selected in stage k of the B_r network should be

$$\begin{array}{ll} b_o\,, & k = 0\,, \\ b_{n-r-1+k}\,, & 0 < k \leq r\,, \\ b_{k-r}\,, & r < k < n\,. \end{array}$$

In other words,

$$h_k^{(r)} = \begin{array}{ll} 1 + (k - r + n - 2)\, mod\, (n - 1)\,, & 0 < k < n\,, \\ 0, & k=0. \end{array}$$

This is the same result as was obtained earlier by direct calculation of $h_k^{(r)}$.

4.7. Blocking properties of three-stage networks.

4.7.1. Existing conditions.

A number of three-stage ATM switch designs were described in section 2.10. The blocking properties of such switches shall now be considered in greater detail.

The conditions for a single-rate, single-channel, three-stage circuit switch to be non-blocking are well known [9]. A number of authors have sought to extend these results to switches with multiple rates, or multiple channels, and to packet switching [10-16].

4.7.2 A new non-blocking condition for circuit-switched networks.

The non-blocking condition for the symmetric three-stage network in Fig. 4.13(a), where m is the number of modules in the intermediate stage, shall now be derived, under the following assumptions:

i. The input (output) modules of the switch have n inputs (outputs) and Sm outputs (inputs).

ii. The input and output of the switch links operate at a rate of v bits/sec.

iii. The intermediate links operate at a rate of rv bits/sec.

iv. The intermediate links are in trunk groups of S links each.

v. The call rate u may occupy a continuous or discrete set of values Z in the range $0 < u_{min} \leq u \leq u_{max} \leq v$.

vi. It is possible to offer 100% load to each input module, i.e., a load of nv bits/s.

vii. Traffic from one call can be assigned a route on only one link in a trunk group, i.e., traffic from one call may not be split across links in a group.

viii. Blocking occurs on a link when the offered load exceeds the link capacity.

ix. The switch modules are non-blocking.

The value of m required to ensure that this switch is strictly non-blocking for circuit-switched traffic shall now be obtained.

Assumption vii. above allows a distinction to be made between channel grouping and speedup as means of increasing the bandwidth of a path. The value of m required to ensure a non-blocking switch may be larger when bandwidth is increased by channel grouping than by speedup, since it may be easier to 'pack' calls onto a single high-speed link than across several slower links. This distinction is unnecessary if traffic from a single call can be distributed across a channel group. If this is the case, the value of r should be replaced by the value of rS in subsequent formulae, and the value of S should be replaced by unity.

Consider the case where a call of rate u is to be placed from input module 1 (IM_1) to output module 1 (OM_1), and where this will result in a 100% load on IM_1 and OM_1. The existing load on both switch modules is thus nv-u.

Each intermediate link of the switch has a capacity of rv. The minimum aggregate rate of existing traffic on such a link which results in that link being unavailable to a call of rate u shall be denoted $F(u)$. Let Ω be the set of non-zero values which the aggregate rate of traffic on an intermediate link can occupy. Then

$$F(u) = \min_{x \in \Omega} \{x \mid x + u > r\,v\}$$

In the worst case, the traffic from an input module to an intermediate module will be evenly shared among all the links in the corresponding trunk group. Hence the minimum level of traffic between the two modules which results in a call of rate u being blocked is

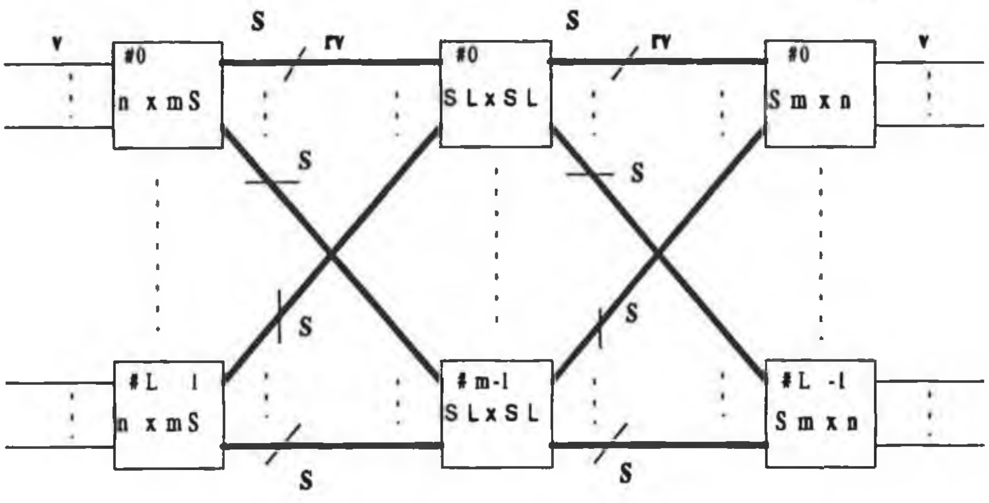
$$SF(u)\,.$$

Traffic from IM_1 to OM_1, will in the worst case, be distributed in such a way that the maximum possible number of trunk groups carry traffic of an aggregate rate of $SF(u)$. In this situation, the total number of intermediate modules which cannot accept a call of rate u from IM_1 is

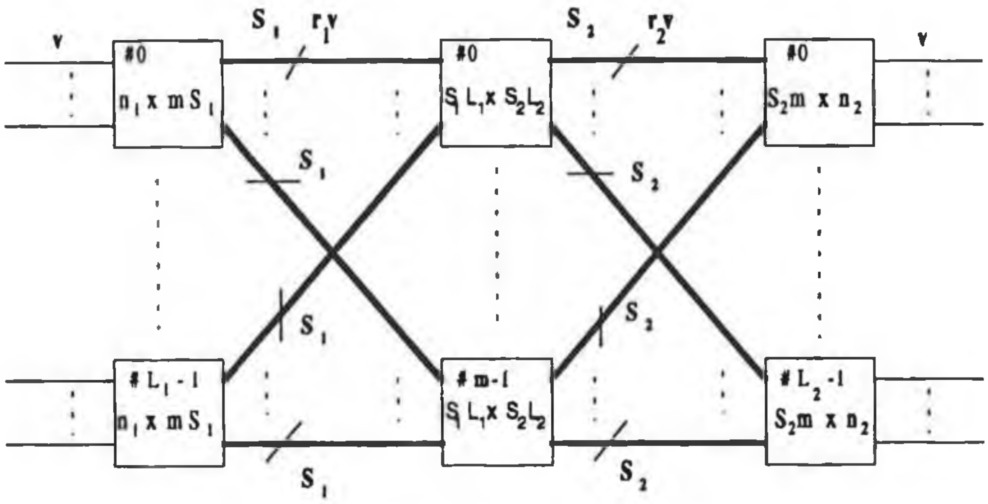
$$\left\lfloor \frac{nv-u}{S.F(u)} \right\rfloor.$$

A similar analysis shows that, in the worst case, the number of intermediate modules which cannot route a call of rate u to OM_1 is

$$\left\lfloor \frac{nv-u}{S.F(u)} \right\rfloor.$$



(a) A symmetric switch



(a) An asymmetric switch

Fig. 4.13: Three-stage switches with channel groups and link speed-up.

The maximum number of intermediate modules is unavailable to a call of rate u if the intermediate modules which block its call on the input side, and those which block on the output side of the module, are distinct. Hence, blocking is impossible for a call of rate u if

$$m > 2\left\lfloor \frac{nv-u}{S.F(u)} \right\rfloor .$$

Hence, no blocking can occur for any call if

$$m > 2 \max_{u \in Z}\left\lfloor \frac{nv-u}{S.F(u)} \right\rfloor .$$

4.7.3 Special cases of the non-blocking condition.

This condition for a three-stage circuit switch to be strictly non-blocking may be shown to reduce to previously obtained conditions in certain special cases [17].

• A single-rate switch with no channel grouping and no link speedup

Suppose $S = r = 1$ and $u_{min} = u_{max} = v$. Then

$$Z = \Omega = \{v\} .$$

Since

$$F(v) = v$$

it follows that

$$m > 2\left\lfloor \frac{nv-v}{v} \right\rfloor ,$$

i.e.,

$$m > 2(n-1) .$$

This is the result first obtained by Clos [9].

• A single-rate switch with channel grouping and speedup

Suppose $v = f_i\, u_b$ where f_i is an integer. Also $r = \frac{f_o}{f_i}$ where f_o is an integer and $f_o > f_i$. Then

$$Z = \{u_b\}$$

and

$$\Omega = \{ku_b, 0 < k \leq f_0\}.$$

Each intermediate link can support f_o calls. Hence

$$F(u_b) = \min_{0<j\leq f_o} \{j\, u_b \mid j\, u_b + u_b > f_o\, u_b\},$$

i.e.,

$$F(u_b) = f_o\, u_b.$$

Therefore

$$m > 2\left\lfloor \frac{nf_i u_b - u_b}{S.f_0.u_b} \right\rfloor$$

or

$$m > 2\left\lfloor \frac{nf_i - 1}{S.f_0} \right\rfloor.$$

This is the strictly non-blocking condition obtained by Jajszczyk [10].

• <u>A multi-rate switch with channel grouping and speedup</u>

This case is the extension of the above case to multi-rate traffic, where $Z = \{ku_b, 0 < k \leq k_m\}$, with $k_m \leq f_i$. Here

$$F(k\, u_b) = \min_{0<j\leq f_0} \{j\, u_b \mid j\, u_b + k\, u_b > f_o\, u_b\}$$

so

$$F(k\, u_b) = (f_o - k + 1)\, u_b$$

Therefore

$$m > 2 \max_{0<k\leq k_m} \left\lfloor \frac{nf_i u_b - ku_b}{S.(f_0 - k + 1).u_b} \right\rfloor.$$

i.e.,

$$m > 2 \max_{0<k\leq k_m} \lfloor \gamma(k) \rfloor,$$

where

$$\gamma(k) = \frac{nf_i u_b - k u_b}{S.(f_0 - k + 1).u_b} \; .$$

It may readily be shown that $\gamma(k)$ is a monotonic increasing function of k, provided that $nf_i > f_0 + 1$, which will be the case for any practical switch. Hence

$$\max_{0<k\le k_m} \lfloor \gamma(k) \rfloor = \left\lfloor \frac{nf_i - k_m}{S.(f_0 - k_m + 1)} \right\rfloor .$$

Therefore

$$m > 2\left\lfloor \frac{nf_i - k_m}{S.(f_0 - k_m + 1)} \right\rfloor .$$

• <u>A multi-rate switch with no channel grouping or link speedup</u>

In the further special case where $f_0 = f_i$ (i.e., where $r = 1$ and $S = 1$) this reduces to

$$m > 2\left\lfloor \frac{nf_i - k_m}{f_i - k_m + 1} \right\rfloor .$$

This is the result obtained by Niestegge [11] for a multi-rate network without internal speed-up.

• <u>A variable rate switch with link speedup and without channel grouping</u>

Suppose $S = 1$ and $Z = \{u \mid u_{min} \le u \le u_{min}\}$. Hence

$$\Omega = \{\, x \mid u_{min} \le x \le rv \,\}.$$

Therefore

$$F(u) = \min_{x \in \Omega} \{x \mid x + u > rv\} \, .$$

Consider first the case where $u_{min} = 0$. In this case

$$F(u) = \lim_{\varepsilon \to 0^+} \{rv - u + \varepsilon\} = rv - u \, .$$

However, if $u_{min} > 0$, it is necessary that $F(u) \ge u_{min}$.

Hence

$$F(u) = max\,(rv - u \, , u_{min}) \, .$$

Thus

$$m > 2 \max_{u_{\min} \le u \le u_{\max}} \left\lfloor \frac{nv - u}{\max(rv - u, u_{\min})} \right\rfloor .$$

This result can be shown to be the same as that obtained by Melen and Turner [12]. It follows that an unstated assumption of Melen and Turner is that the call rate u can occupy a continuum of values in the range from u_{min} to u_{max}. A circuit switch supporting such a continuum of call rates appears impractical. This result can, however, be applied to packet switching networks, as discussed in section 4.7.5.

4.7.4 Asymmetric switches.

The non-blocking condition for the asymmetric switch of Fig. 4.13(b) is

$$m > \max_{u \in Z} \left(\left\lfloor \frac{n_1 v - u}{S_1 . F_1(u)} \right\rfloor + \left\lfloor \frac{n_2 v - u}{S_2 . F_2(u)} \right\rfloor \right) ,$$

where

$$F_1(u) = \min_{x \in \Omega_1} \{ x \mid x + u > r_1 v \} ,$$

$$F_2(u) = \min_{x \in \Omega_2} \{ x \mid x + u > r_2 v \} ,$$

and Ω_1 (Ω_2) is the set of values which the aggregate rate of traffic on an intermediate link entering (leaving) the intermediate stage can occupy. The proof of this result follows by a straightforward extension of that for the symmetric case.

4.7.5 Non-blocking conditions for ATM networks with call-level path allocation.

The above condition needs to be modified before being applied to packet-switched networks or ATM networks. Two different sets of results must be obtained, because of the two distinct strategies possible for performing routing in three-stage ATM switches.

Multiple paths from source to destination are available in a three-stage switch. Hence a path allocation algorithm must be employed to select among the available paths. A distinction is made between two time scales over which path allocations may be performed:

- Call level;
- Cell level.

The condition already developed for circuit-switching can be applied to the call-level case. The bit-rate of a call can vary continuously from zero to the peak rate *u*. Hence, the appropriate formula is

$$m > 2\max_{0 \le u \le v}\left\lfloor \frac{nv-u}{S.(rv-u)} \right\rfloor ,$$

i.e.,

$$m > 2\left\lfloor \frac{n-1}{S.(r-1)} \right\rfloor .$$

This result has been obtained by Melen & Turner [12] in the case where $S = 1$, and is valid for $r > 1$. It indicates that a speed up of the internal links is essential if a three-stage packet switch with a call-level path allocation algorithm is to be non-blocking. Otherwise, even the smallest level of background traffic is sufficient to block a call at the peak rate *u*; in the worst case, all intermediate links will have some background traffic, however light. It is assumed that all cells on a virtual circuit are routed on the same *channel*, and not just on the same *channel group*. If this assumption does not hold, no distinction is necessary between channel speed-up (r) and group size (S). The condition on *m* then becomes

$$m > 2\left\lfloor \frac{n-1}{Sr-1} \right\rfloor .$$

The term 'blocking' requires some interpretation when used in the context of a call-level routing algorithm. Its meaning depends on what is defined by the call rate *u*. Suppose *u* is defined to be the peak rate of a virtual circuit. In that case, a non-blocking switch is one where the peak rate on any internal link in the switch never exceeds the link capacity, provided the peak rates on the inputs and outputs never exceed their capacities. This interpretation may be applied in an ATM network where bandwidth reservation is based on the peak call rate.

Another interpretation is to consider *u* to be the mean bit rate on a virtual circuit. Then a non-blocking switch may be interpreted to be one where the mean bit rate on any internal link never exceeds the link capacity, provided that the mean bit rates on the inputs and outputs never exceed their capacities. In other words, if the offered load is such that no input or output of the switch is saturated, then no internal link in the switch will be saturated. This interpretation of the criterion is not useful in ATM networks, since the requested quality of service may not be maintained for a call if it is carried on a link with a utilisation approaching 100%.

The main quality issue for a virtual circuit on an ATM network is the cell loss probability for the virtual circuit. The purpose of call admission control in ATM networks is to determine whether a call can be accepted by the network, while still maintaining the negotiated quality of service for existing calls, and providing the requested quality of service to the new call. A link in the switch should be regarded as blocked to a virtual circuit if the call admission control function would reject an attempt to set up the virtual circuit using that link.

The call admission control procedure makes its decisions based upon a set of traffic descriptors supplied during the call set-up attempt. No decision has been made by CCITT at the time of writing regarding the definition of traffic descriptors.

Unfortunately, the cell loss probability for virtual circuits sharing a link is likely to be a highly non-linear function of the traffic descriptors. Hence, the non-blocking condition above, which assumes that the cost of routing virtual circuits via a given link (i.e., the aggregate bit rate on the link) is a linear function of a single parameter describing each virtual circuit (i.e., the bit rate u of the virtual circuit), requires considerable modification.

One approach is to consider u to be the *equivalent bandwidth* (more strictly, the equivalent bit-rate) of the call. This is the amount of bandwidth which must be available to the call, if excessive cell loss is to be avoided. The equivalent bandwidth is calculated from the supplied traffic descriptor(s). Inherent in this approach is the assumption that the equivalent bandwidth of the aggregate of calls sharing a link is a linear function of the equivalent bandwidths of the individual calls. This allows the results for multi-rate circuit switching to be applied to the design of a nonblocking ATM switch.

The problem of ensuring that a three-stage ATM switch is nonblocking has been considered by Svinnset [13]. He suggested (implicitly) that the instantaneous bit rate of a virtual circuit be assumed to be drawn from a normal distribution. The bandwidth occupied by the virtual circuits sharing a link was then calculated as that value which, with a specified probability, was not exceeded by the instantaneous aggregate bit rate on the link. The normal assumption resulted in a linear function for the calculation of equivalent bandwidth. Svinnset also considered more complex models of traffic, where more than one traffic descriptor is required to characterise a call. However, the complexity of the resulting condition for the three-stage ATM switch to be non-blocking rapidly escalates, as the number of parameters describing a call (traffic descriptors) is increased.

The call admission control procedure used in practice will have to be chosen as a compromise between simplicity and efficiency (in the sense of achievable multiplexing gain). Since the choice of this function is unclear, how to formulate the non-blocking condition for a three-stage ATM switch featuring call-level routing remains an open question.

4.7.6 The non-blocking condition for a three-stage ATM switch with cell-level path allocation.

In this context, the conditions applicable when using call-level path allocation are not valid, since the cells belonging to a given virtual circuit are unlikely to be routed via the same intermediate switch module. A three-stage ATM switch more closely resembles a single-rate circuit switch when path allocation is performed at cell level since, for the duration of each time slot, each input is either inactive or transmitting data at the full ATM rate.

The relevant non-blocking condition for an asymmetric single-rate three stage circuit switch is obtained where

$$Z = \{v\} ,$$

$$\Omega_1 = \{kv , 0 < k \leq r_1\} ,$$

and

$$\Omega_2 = \{kv , 0 < k \leq r_2\} .$$

It is assumed that r_1 and r_2 are integers since otherwise some bandwidth on each intermediate link would be wasted in each time slot. Under these circumstances,

$$F_1 (v) = r_1 v ,$$

and

$$F_2 (v) = r_2 v .$$

Hence

$$m > \left\lfloor \frac{(n_1 - 1)v}{S_1 . r_1 v} \right\rfloor + \left\lfloor \frac{(n_2 - 1)v}{S_2 . r_2 v} \right\rfloor ,$$

i.e.,

$$m > \left\lfloor \frac{n_1 - 1}{S_1 . r_1} \right\rfloor + \left\lfloor \frac{n_2 - 1}{S_2 . r_2} \right\rfloor .$$

A switch satisfying this condition can route n_2 cells to each output module in each time slot. However, in packet switching, there is no longer a one-to-one correspondence between the number of output ports n_2 and the number of requests for an output. Thus, if all cells are to be guaranteed passage through the intermediate stage of the switch, it must be dimensioned so that *all* cells can be routed to the same output module, i.e., m must be chosen such that

$$m > \left\lfloor \frac{n_1 - 1}{S_1 . r_1} \right\rfloor + \left\lfloor \frac{n_1 L_1 - 1}{S_2 . r_2} \right\rfloor .$$

Such a switch would be prohibitively expensive. Hence another criterion must be used to dimension the switch. This topic is considered further in Chapter Six.

References

[1] C. Wu and T.-Y. Feng, "On a class of multistage interconnection networks", *Trans. Comput.*, vol. 29, no. 8, pp. 694-702, Aug. 1980.

[2] D. Agrawal, "Graph theoretical analysis and design of multistage interconnection networks", *Trans. Comput.*, vol. 32, no. 7, pp. 637-648, July 1983.

[3] F. Bernabei *et al.*, "On non-blocking properties of parallel delta networks", *Proc. Infocom '88*, pp. 326-333, 1988.

[4] T.T. Lee, "Nonblocking copy networks for multicast packet switching", *Journal Of Select. Areas Commun.*, vol. 6, no. 9, pp. 1455-1467, Dec. 1988.

[5] M.J. Narasimha, "The Batcher-banyan self-routing network: universality and simplification", *IEEE Trans. Commun.*, vol. 36, no. 10, pp. 1175-1178, Oct. 1988.

[6] H. Kim and A. Leon-Garcia, "A multistage ATM switch with interstage buffers", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. V, pp. 15-20.

[7] A. Huang and S. Knauer, "Starlite: a wideband digital switch", *Globecom '84 Conference Record*, pp. 121-125, Nov. 1984

[8] S. Urushidani, "Rerouting network: a high-performance self-routing switch for B-ISDN",. *Journal Of Select. Areas Commun.*, vol. 9, no. 8, pp. 1194-1204, Oct. 1991.

[9] C. Clos, "A study of non-blocking switching networks", *Bell Syst. Tech. Journal*, vol. 32, no. 2, pp. 406-424, Mar. 1953.

[10] A. Jajszczyk, "On nonblocking switching networks composed of digital switching matrices", *IEEE Trans. Commun.*, vol. 31, no. 1, pp. , Jan. 1983.

[11] G. Niestegge, "Nonblocking multirate switching networks", *Proc. 5th ITC Seminar*, pp. 449-458, 1987.

[12] R. Melen and J. Turner, "Nonblocking multirate networks", *SIAM J. Comput.*, vol. 18, no. 2, pp. 301-313, Apr. 1989.

[13] I. Svinnset, "Nonblocking ATM switching networks", *Proc. ITC-13*, A. Jensen and V. Iversen (eds.), pp. 1011-1016, North-Holland, 1991.

[14] Y. Sakurai *et al.*, "Large scale ATM multi-stage switching network with shared buffer memory switches", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. IV, pp. 121-126.

[15] K. Sezaki et al., "The cascade Clos broadcast switching network - a new ATM switching network which is multiconnection non-blocking", Proc. of the International Switching Symposium, Stockholm, 1990, vol. IV, pp. 143-147.

[16] A. Jajszcyk, "On combinatorial properties of broadband time-division switching networks", *Proc. ITC Specialists Seminar*, N. van Dijk *et al.* (eds.), pp. 377-382, North-Holland, 1990.

[17] M. Collier and T. Curran, "The strictly non-blocking condition in three-stage networks", submitted for publication.

5. A NEW THREE-STAGE BROADBAND SWITCH.

5.1 Introduction.

Many of the switch proposals described in Chapter Two are only practical in the design of small switches. For example, the number of switch elements in the Sunshine switch [1] becomes excessive as the switch size increases [2]. A different approach must be taken to the design of large switches.

An obvious method of implementing a large switch, given these constraints on switch size, is to design the switch with multiple stages, where each stage consists of smaller switch modules. A number of such switches has been described in Chapter Two. This approach to the design of a large switch typically introduces a new problem whereby multiple paths from source to destination become available. Thus, even if the individual switch modules possess the self-routing feature, this feature is not retained by the overall switch. Some method of path allocation is then necessary, to select among the available paths from source to destination.

A distinction was made in Chapter Four between *call*-level and *cell*-level routing in a three-stage ATM switch. The relative merits of these approaches to routing will now be discussed.

Performing path allocation at circuit set-up time has a number of advantages. The switch modules in each stage need to make routing decisions based only on information contained in the cell header (specifically the VCI/VPI) and a routing table updated at call set-up time. Hence they can be identical in design to a single stage switch. Routing of cells is simple once these tables have been updated at circuit set-up time. Also, preservation of cell sequence within a virtual circuit can be easily guaranteed, since all the cells follow the same route through the network.

However, the problem of path allocation is subsumed into the call admission control process, adding further complexity to the call set-up procedure. Global knowledge of data flows through each intermediate node is required, so that a centralised path allocation will be needed. The optimal choice of cost function to be used during call admission, to determine whether the call may be allocated a given path, is not clear, as discussed in section 4.7.5. If, in spite of the call admission process, saturation of an intermediate link does occur, traffic using that link cannot be diverted to other intermediate modules without tear-down of circuits, unless a complex alternate routing algorithm is used. A cost function which ensures that the cell loss probability on a link remains low, even in the presence of a complex traffic mix, is likely to allocate bandwidth conservatively, so that the link utilisation will be low.

With cell-level routing, the intermediate switch modules can be of comparatively simple design, since the path allocation algorithm should ensure that they never have to deal with output contention. Traffic is distributed more evenly across the intermediate switch modules than is the case with routing at call level, since routes are updated much more frequently. Thus, higher throughput should be attainable with this type of routing. No additional overhead is imposed by the switch during call set-up. The call admission procedure need only consider the traffic flowing on a requested output link, not throughout the switch, in considering a possible route for a virtual circuit. In addition, no routing tables or VCI/VPI translation tables need be maintained for the intermediate stage of the switch, as would be required with cell-level routing.

The chief disadvantage of cell-level routing is that high speed hardware is required to perform the path allocation, since paths must be reallocated once per time slot. This hardware must be highly reliable; a fault here can cause routing failures across the entire switch. Additionally, if cell sequence must be preserved within a virtual circuit, this will require special consideration, since cells are routed independently through the switch.

Path allocation can be performed at cell level if a path allocation algorithm can be implemented in hardware in such a way that the resulting circuitry is not required to operate at a prohibitively high speed. In practice, this means that the parallelism in the hardware must be maximised.

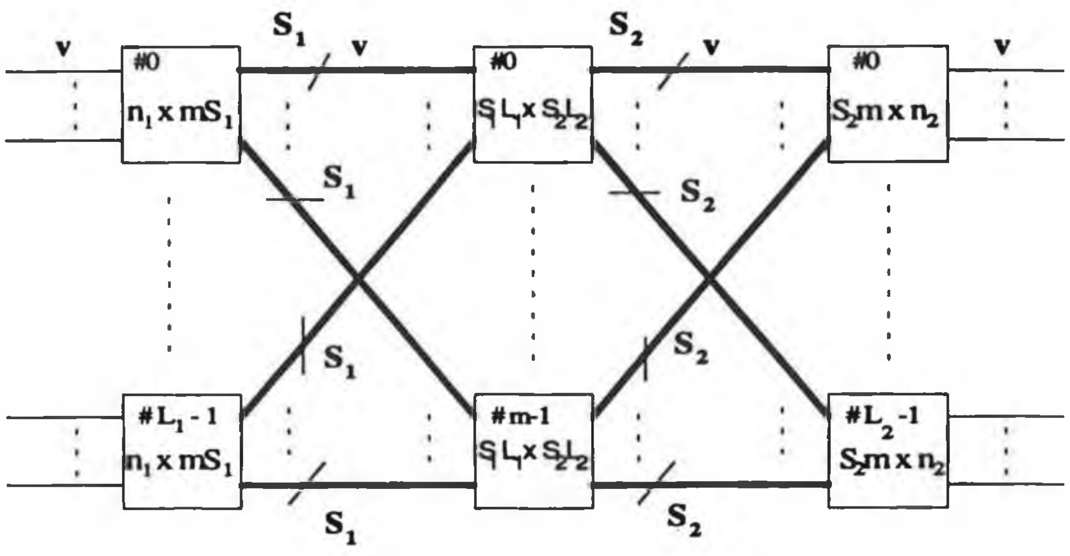
A method for cell-level path allocation is described in this chapter. It can be applied to the channel-grouped three-stage switch architecture of Fig. 5.1.

5.2 An algorithm for path allocation at cell level.

5.2.1 The objectives of a path allocation algorithm.

There are S_1 routes from each input module to each intermediate switch module. There are S_2 routes from each intermediate switch module to each output module. We must choose, for every input cell (if possible) an intermediate switch module through which to pass on the way to the selected destination, such that no input module attempts to route more than S_1 cells via any intermediate switch module, and no intermediate switch module attempts to route more than S_2 cells to any output module, in any one time slot.

Note that, in this problem, an attempt is made to reserve bandwidth for each input cell, such that it can pass through the intermediate stage without blocking. An alternative strategy would be to select intermediate stage modules based on some simpler criterion (e.g., random selection) without testing for the availability of a path from the intermediate stage to the output stage. The former strategy is preferred for a number of reasons:



V:	channel rate (155 Mb/s)
n_1 (n_2) :	the number of input (output) ports per input (output) module;
L_1 (L_2) :	the number of input (output) modules;
m:	the number of intermediate switch modules;
S_1 (S_2) :	the number of channels in the channel group connecting each input (output) module to each intermediate switch module.

Fig. 5.1: A three-stage switch with intermediate channel grouping.

(i) no queuing occurs in the intermediate stage; thus the delay through the intermediate stage is uniform, regardless of the path taken; this makes it possible to preserve cell sequence on a virtual circuit;

(ii) the intermediate stage can never be congested;

(iii) intermediate stage modules can be of simple design, since contention can never occur.

5.2.2 Existing algorithms for cell-level path allocation.

A number of solutions to this problem have been proposed, in the special case where $S_1 = S_2 = 1$ [3-7]. These algorithms typically use one bit to represent each channel in the switch. Thus the number of bits being processed by the path allocation algorithm is large for a large switch. The run-time of each algorithm may be presumed to be proportional to the number of bits processed by the algorithm. Accordingly, the amount of data processed by the algorithms described in section 2.10 shall now be considered.

In Cisneros' method [3], there are N units of data, one per input. These are circulated to all inputs, so that N iterations are required. One pass of the algorithm must be performed for each intermediate switch module, so that the run-time of the algorithm is proportional to Nm. The method has the advantage that multiple priorities can be processed simultaneously.

In Eng's method [4], a total of L_2 vectors, each m bits long, is circulated around the L_1 input modules. The switch is a square switch, so that $L_1 = L_2 = L$. The path assignment algorithm is only suitable for assigning a path to one cell at a time, and therefore n_1 iterations within each input module are required. Thus the execution time is proportional to $Ln_1 = N$. However, the algorithm would have to be repeated for each priority class, so that the runtime is proportional to kN_1 with k priority classes.

This algorithm was extended to the case of a channel-grouped architecture in [5]. However, the 'diagonal matching' technique used to select a free path from among the available channels adds further to the execution speed requirements. Unless relatively complex hardware is used to perform the diagonal matching on all cells in a channel-group simultaneously, the execution time will be increased by a factor of S, where S is the size of each channel group.

The implementation of Proctor and Madden [6] has an execution time proportional to mL_1. The first and third stage switches are rotators and so must be square. Thus $L_1 = m$. Hence, the execution time of this switch is also proportional to N. This algorithm would also have to be repeated for each priority class. Similar remarks apply to the rotating-access switch of [7].

All of the above algorithms have a runtime which increases linearly with increasing N. An alternative algorithm is described below, whose execution speed increases less rapidly with increasing N. The main feature of this algorithm is that it processes the accumulated requests from an input module in each time slot as a unit, rather than processing each input port individually. Also, the algorithm can allocate paths through the switch to several cells in a single operation. This is possible through the use of a channel-grouped architecture.

5.2.3 A new algorithm for path allocation.

A new and efficient algorithm will now be described [8,9]. It is suitable for use in a channel-grouped three-stage switch and requires only knowledge obtainable at the input side of the switch. The key to its high performance is the encoding of data concerning the availability of paths into binary words (thereby reducing the number of bits to be processed by the algorithm for a given switch size) and the extensive use of parallelism. It operates on the following quantities:

A_{ir} : the number of channels available from input module *i* to intermediate switch module *r*;

B_{rj} : the number of channels available from intermediate switch module *r* to output module *j*;

K_{ij} : the number of requests from input module *i* for output module *j*.

Note that A_{ir} and K_{ij} need only be local to the input module. The B_{ir}'s must be forwarded to each input module in turn. This is performed by a ring structure connecting each input module. Such an arrangement is shown in Fig. 5.2, where each row is located at an input module. Let R_{irj} be the number of cells to be routed from input module *i* to output module *j* via intermediate switch module *r*. The values of A_{ir}, B_{rj} and K_{ij} are updated using the procedure *atomic(i,r,j)* described below:

$$\left.\begin{aligned} R_{irj} &= \min(K_{ij}, B_{rj}, A_{ir}) \\ K_{ij} &\leftarrow K_{ij} - R_{irj} \\ B_{rj} &\leftarrow B_{rj} - R_{irj} \\ A_{ir} &\leftarrow A_{ir} - R_{irj} \end{aligned}\right\} \quad atomic(i,r,j)$$

This procedure is 'atomic' in the sense that it is the basic building block from which the path allocation algorithm is constructed. The procedure determines the capacity available from input module *i* to output module *j* via intermediate switch module *r* (i.e., the minimum of A_{ir} and B_{rj}). The number of requests which can be satisfied is equal to the minimum of the number of requests outstanding (K_{ij}) and the available capacity.

A sequential implementation of the path allocation algorithm requires the repeated execution of *atomic(i,r,j)* on a single processor for all possible values of *i,r* and *j*. Initially K_{ij} is set equal to the total number of requests from input module *i* for output module *j* (which number is obtained by examining the requests at the switch module inputs), A_{ir} is set equal to S_1 and B_{rj} is set equal to S_2.

A parallel implementation requires multiple processors, each executing the *atomic()* procedure for a different set of procedure parameters, subject to the following constraints:

- No two processors shall simultaneously require access to the same quantity. For example, *atomic*(2,0,0) uses A_{20}, B_{00} and K_{20}, so that *atomic*(2,0,X), *atomic*(2,X,0) and *atomic*(X,0,0) cannot be executed concurrently with *atomic*(2,0,0) for any X.

- The data required by a processor for the next iteration of the algorithm should be available locally, or from adjacent processors.

An implementation satisfying these two constraints will now be described.

5.2.4 Synchronisation of the algorithm.

The algorithm requires a total of L_1 x L_2 processors. The detailed operation of the algorithm depends on the number of switch modules in each stage. The simplest case, where $L_1 = L_2 = m$, is considered first.

Processor X_{ij} is initialised by loading the following three values:

(i) the initial value of K_{ij};

(ii) the initial value of $A_{i,(i+j) \ mod \ m}$ (S_1 by default);

(iii) the initial value of $B_{(i+j) \ mod \ m,j}$ (S_2 by default).

The algorithm then requires m iterations (iterations zero through m-1). Processor X_{ij} executes *atomic*(i, $(i+j-k) \ mod \ m$, j) during iteration k . After each iteration X_{ij} forwards the updated value of B_{rj} to $X_{(i+1) \ mod \ m, \ j}$ and of A_{ir} to $X_{i, \ (j+1) \ mod \ m}$, and retains K_{ij}.

A total of m iterations of the algorithm suffices to determine the path allocated (if any) for every input to the switch (a total of m x n_1 inputs). The mechanism for passing information to an input cell concerning the path allocated to it will be described in section 5.4. The hardware layout for the case where $m = 4$ is shown in Fig. 5.2, which illustrates the array of sixteen processors required, and the contents of their registers. Each row in Fig. 5.2 contains four processors, which are co-located with the corresponding input module. Each column in Fig. 5.2 processes requests for a single output module. Thus, for example, the processor in row one and column two of the array handles requests for cells to be routed from input module one to output module two. The processor contents at the start of each of the four iterations of the algorithm are shown in Fig. 5.2 (a) through Fig. 5.2 (d). A total of 64 paths is available through the switch (four for each input-output module pair). The sixteen processors attempt to allocate cells to sixteen of these paths during each iteration. After each iteration, the updated value of A_{ir} is passed to the adjacent processor in the same row, and the updated value of B_{rj} is passed to the adjacent processor in the same column. The directions of data flow are indicated by arrows in Fig. 5.2. No two processors can allocate a path sharing a channel in the same iteration. Nevertheless, after four iterations, all possible paths have been allocated.

The algorithm presented above [8] must be modified if $L_1 \neq m$ or $L_2 \neq m$.

Consider first the case where $L_1 < m$ and $L_2 = m$. In this situation, there are more intermediate switch modules through which paths can be sought during each iteration, than there are input modules to attempt such a search. This difficulty is overcome by continuing to use an m x m processor array, with an unchanged algorithm, apart from a modification to the initial values for K_{ij}, for $i \geq L_1$, which are set to zero.

The success of this algorithm is self-evident if we associate the additional processors (X_{ij}, $L_1 \leq i < m$) with $m - L_1$ null input modules. This switch can then be regarded as a special case of the switch with m modules per stage, where $m - L_1$ of the input modules contribute no traffic, and thus have $K_{ij} = 0$. Hence no paths are allocated to cells from the null input modules.

Next consider the case where $L_1 > m$ and $L_2 = m$. In this situation, there are insufficient intermediate switch modules for it to be possible for all input modules to attempt to allocate paths through the intermediate stage. The solution to this difficulty is as follows. A square array of L_1 x L_1 processors is used. Processor X_{ij} is initialised in the following manner.

$$K \leftarrow \begin{matrix} K_{ij}, & j < m \\ 0, & j \geq m.. \end{matrix}$$

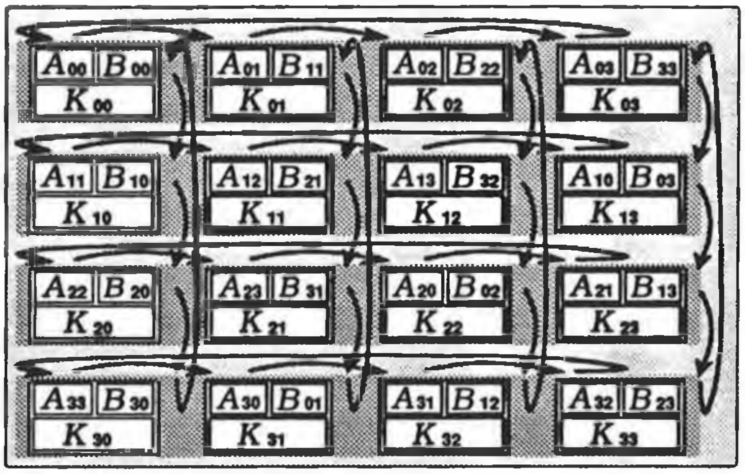
$$A \leftarrow \begin{matrix} A_{ir}, & r < m \\ 0, & r \geq m.. \end{matrix}$$

$$B \leftarrow \begin{matrix} B_{rj}, & r < m \\ 0, & r \geq m.. \end{matrix}$$

where $r = (i + j) \bmod L_1$.

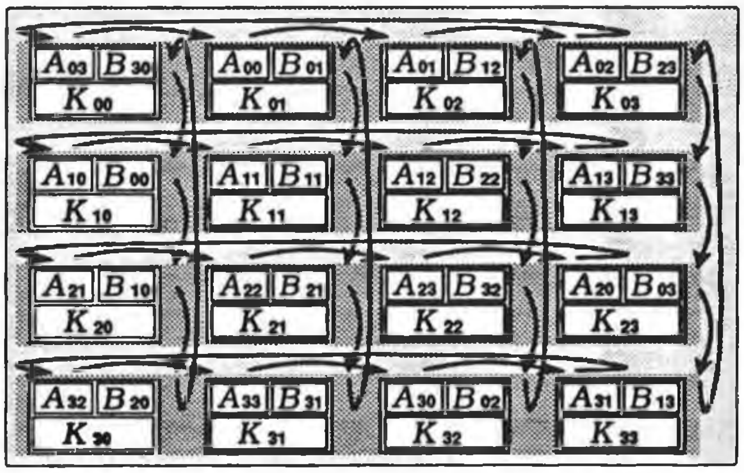
The algorithm now requires L_1 iterations. In iteration k, processor X_{ij} executes *atomic* (i, r, j) with $r = (i + j - k) \bmod L_1$. After each iteration, X_{ij} forwards the updated value of B_{rj} to $X_{(j+1) \bmod L_1, j}$ and of A_{ir} to $X_{i,(j+1) \bmod L_1}$, and retains K_{ij}.

This algorithm may be regarded as applying to a switch with L_1 switch modules per stage, but where L_1-m of the switch modules in the intermediate and output stages are null modules, so that $K_{ij} = B_{rj} = 0$ for $m \leq j < L_1$, and $A_{ir} = 0$ for $m \leq r < L_1$. Since no cells are allocated paths via the null intermediate modules, or to the null output modules, path allocation is performed correctly from the L_1 input modules to the m output modules.



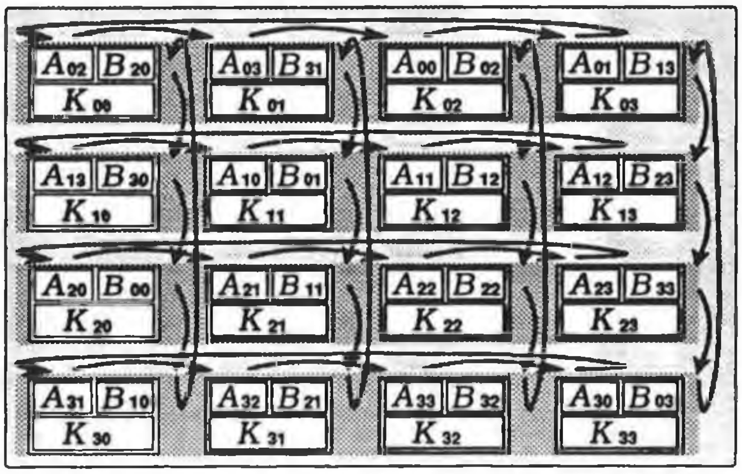
A_{00} B_{00} K_{00}
A_{01} B_{11} K_{01}
A_{02} B_{22} K_{02}
A_{03} B_{33} K_{03}
A_{11} B_{10} K_{10}
A_{12} B_{21} K_{11}
A_{13} B_{32} K_{12}
A_{10} B_{03} K_{13}
A_{22} B_{20} K_{20}
A_{23} B_{31} K_{21}
A_{20} B_{02} K_{22}
A_{21} B_{13} K_{23}
A_{33} B_{30} K_{30}
A_{30} B_{01} K_{31}
A_{31} B_{12} K_{32}
A_{32} B_{23} K_{33}

Fig. 5.2(a): **Contents of processors during Iteration 0.**



A_{03} B_{30} K_{00}
A_{00} B_{01} K_{01}
A_{01} B_{12} K_{02}
A_{02} B_{23} K_{03}
A_{10} B_{00} K_{10}
A_{11} B_{11} K_{11}
A_{12} B_{22} K_{12}
A_{13} B_{33} K_{13}
A_{21} B_{10} K_{20}
A_{22} B_{21} K_{21}
A_{23} B_{32} K_{22}
A_{20} B_{03} K_{23}
A_{32} B_{20} K_{30}
A_{33} B_{31} K_{31}
A_{30} B_{02} K_{32}
A_{31} B_{13} K_{33}

Fig. 5.2(b): **Contents of processors during Iteration 1.**



A_{02} B_{20} K_{00}
A_{03} B_{31} K_{01}
A_{00} B_{02} K_{02}
A_{01} B_{13} K_{03}
A_{13} B_{30} K_{10}
A_{10} B_{01} K_{11}
A_{11} B_{12} K_{12}
A_{12} B_{23} K_{13}
A_{20} B_{00} K_{20}
A_{21} B_{11} K_{21}
A_{22} B_{22} K_{22}
A_{23} B_{33} K_{23}
A_{31} B_{10} K_{30}
A_{32} B_{21} K_{31}
A_{33} B_{32} K_{32}
A_{30} B_{03} K_{33}

Fig. 5.2(c): **Contents of processors during Iteration 2.**

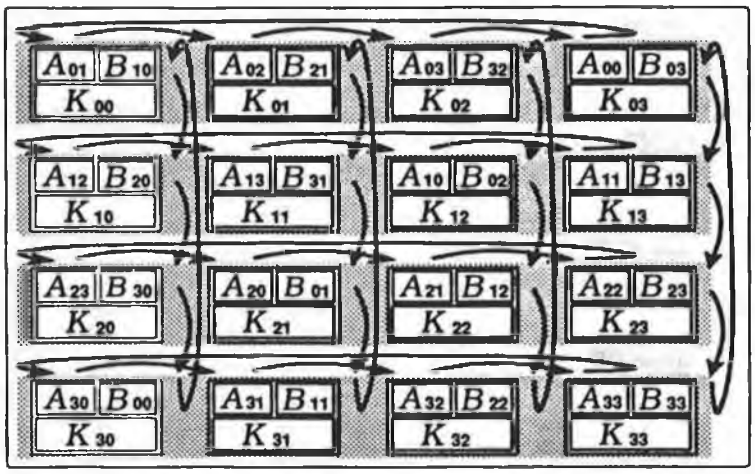


Fig. 5.2(d): Contents of processors during Iteration 3.

The algorithm modifications necessary when $L_2 \neq m$ are of a similar nature. Accordingly, the general algorithm, for a switch with an arbitrary number of modules in each stage, is as follows, where $m' = max\,(L_1, L_2, m)$.

A square array of m' x m' processors is used.

Processor X_{ij} is initialised as follows, where $r = (i + j)\ mod\ m'$.

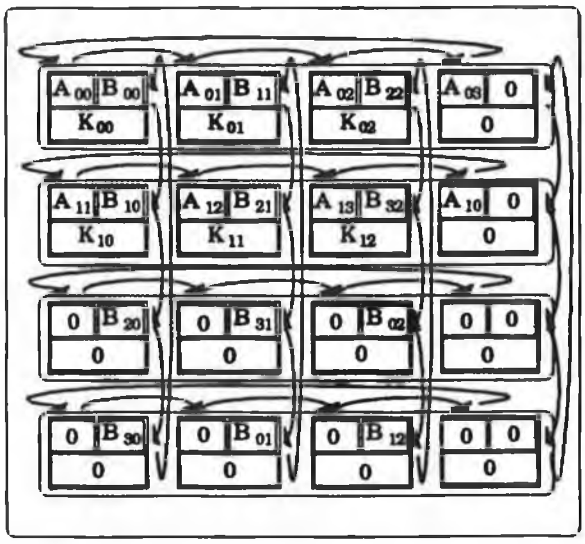
$$K \leftarrow \begin{cases} K_{ij}, & i < L_1, j < L_2 \\ 0, & otherwise. \end{cases}$$

$$A \leftarrow \begin{cases} A_{ir}, & i < L_1, r < m \\ 0, & otherwise. \end{cases}$$

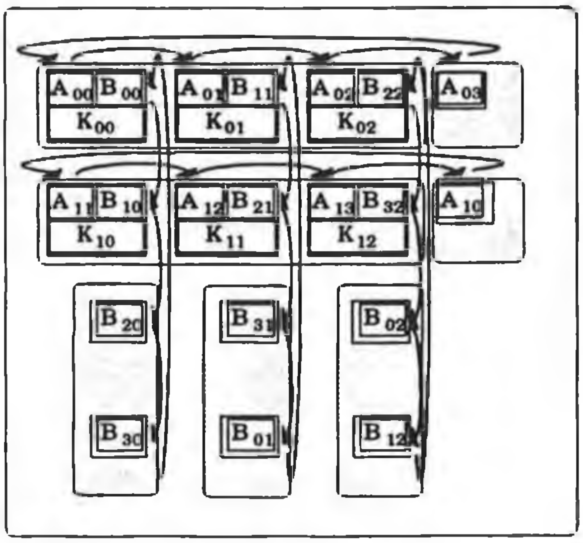
$$B \leftarrow \begin{cases} B_{rj}, & r < m, j < L_2 \\ 0, & otherwise. \end{cases}$$

The algorithm requires m' iterations (iterations zero through m'-1). Processor X_{ij} executes atomic $(i, (i + j - k)\ mod\ m', j)$ during iteration k. After each iteration processor X_{ij} forwards the updated value of B_{rj} to processor $X_{(i+1)\ mod\ m', j}$ and of A_{ir} to processor $X_{i, (j+1)\ mod\ m'}$ and retains K_{ij}.

Consider the application of this algorithm in the case where $L_1 = 2$, $L_2 = 3$ and $m = 4$. In this situation, $m' = 4$, and the implementation is identical to that considered in Fig. 5.2, where $L_1 = L_2 = m = 4$, except for the initial conditions, which are as shown in Fig. 5.3 (a).



(a) Full set of processors.



(b) Reduced set of processors.

Fig. 5.3: Initial conditions for L_1 =2, m = 4, and L_2 = 3.

Note that 22 of the 48 registers in this example are initialised to zero. If register K of a processor is initialised to zero, the contents of its A and B registers are never altered, but are simply stored for one iteration, before being forwarded to an adjacent processor. If all the K registers in a column contain zero, the B registers in that column are redundant. If all the K registers in a row contain zero, the A registers in that row are redundant. After the elimination of redundant registers, the processor layout is as shown in Fig. 5.3 (b).

Now only six (i.e., L_1 x L_2) processors are required. The necessary synchronisation between the arrival time of A_{ir} and B_{rj} into processor X_{ij} is ensured by the buffers at the end of each row or column, which store the number of channels available in those groups over which path allocation is not being attempted during the current iteration.

In general, a switch with L_1 input modules and L_2 output modules requires a processor array with L_1 rows and L_2 columns. If $L_1 < m$, each column requires $m - L_1$ additional B registers. If $L_2 < m$, each row requires $m - L_2$ additional B registers. Thus there is a modest penalty in terms of circuit complexity in implementing a switch where $m > min\ (L_1, L_2)$. This penalty can be avoided if the bandwidth of the intermediate stage is increased by increasing S_1 and S_2, rather than by increasing m.

5.2.5 Modular growth of the switch.

Switch growth will be achieved by fixing the dimensions of the switch modules, and then expanding the switch size by adding input and output modules when needed. The number of intermediate stage modules must also be increased as appropriate to maintain the quality of service.

At least two strategies may be considered in incorporating the facility for modular growth in the path allocation circuitry. Firstly, the circuitry must be designed such that it operates with a fully expanded switch, i.e., where $L_1 = L_1^{max}$, $L_2 = L_2^{max}$ and $m = m^{max}$.

Modular growth can then occur in the following ways:

(i) a full -sized L_1^{max} x L_2^{max} processor array is installed, with K_{ij}, A_{ir} and B_{rj} initialised to zero for the null modules. After the addition of an extra module, the initialisation circuitry would be reprogrammed appropriately. This corresponds to the example in Fig. 5.3 (a), where the full-size switch is assumed to contain four modules in each stage.

(ii) a minimally-sized L_1 x L_2 processor array is installed. Shift registers are used to store the excess A_{ir} and B_{rj} values in each row and column. This corresponds to Fig. 5.3 (b). These shift registers are replaced by processors as appropriate when expansion takes place.

The first approach requires the larger initial capital outlay, but has the advantage that switch expansion may take place while the switch is in commission without difficulty. Hybrids of these two approaches could also be considered.

5.3 Implementation of the processing element.

5.3.1 Design requirements.

The processor must execute the *atomic*() procedure, and thus must perform two types of operation:

1. Find the minimum of three numbers.

2. Perform three subtractions.

Since the number of processors required (L_1 x L_2) can be large, it is imperative that each processor be implemented using comparatively few gates. A second, and equally compelling, reason to have a simple processor implementation, is that the circuitry will be required to operate at a high speed.

Given that the path allocation algorithm, as described in section 5.2, will be feasible only if a simple, high-speed implementation can be found for the processing element, the design of this element will now be considered in some detail.

5.3.2 A bit-serial implementation.

A number of approaches to the implementation of the processing element are possible. One approach is to use a bit serial arithmetic. The width of the ring interconnecting rows of processors will be large, given the large number of entries in the vector of B_{rj} values which must be passed from input module to input module after each iteration. Transmitting the B_{rj} values in bit-serial form reduces the ring width. Fig 5.4 shows a possible implementation.

Unfortunately, determination of the minimum requires values to be presented most significant bit first, while bit serial subtraction requires values which are presented least significant bit first. Hence the processor must be able to perform bit reversal on the quantities processed.

It will now be demonstrated that the execution time of the path allocation algorithm is quite short, and that the hardware required is of modest complexity. This requires a description of the minimisation circuitry at the gate level. Implementation of the bit-serial subtractors poses no difficulties, and the bit reversal may be performed by the appropriate use of shift registers. Hence no details will be provided of the relevant circuits.

A method for determining the minimum using bit serial arithmetic will now be described. It may be shown that the minimum of three numbers may be found as follows:

Suppose the three numbers are represented by n binary digits, i.e.,

$$A = a_{n-1}\ a_{n-2} \ldots\ldots a_o ,$$

$$B = b_{n-1}\ b_{n-2} \ldots\ldots b_o ,$$

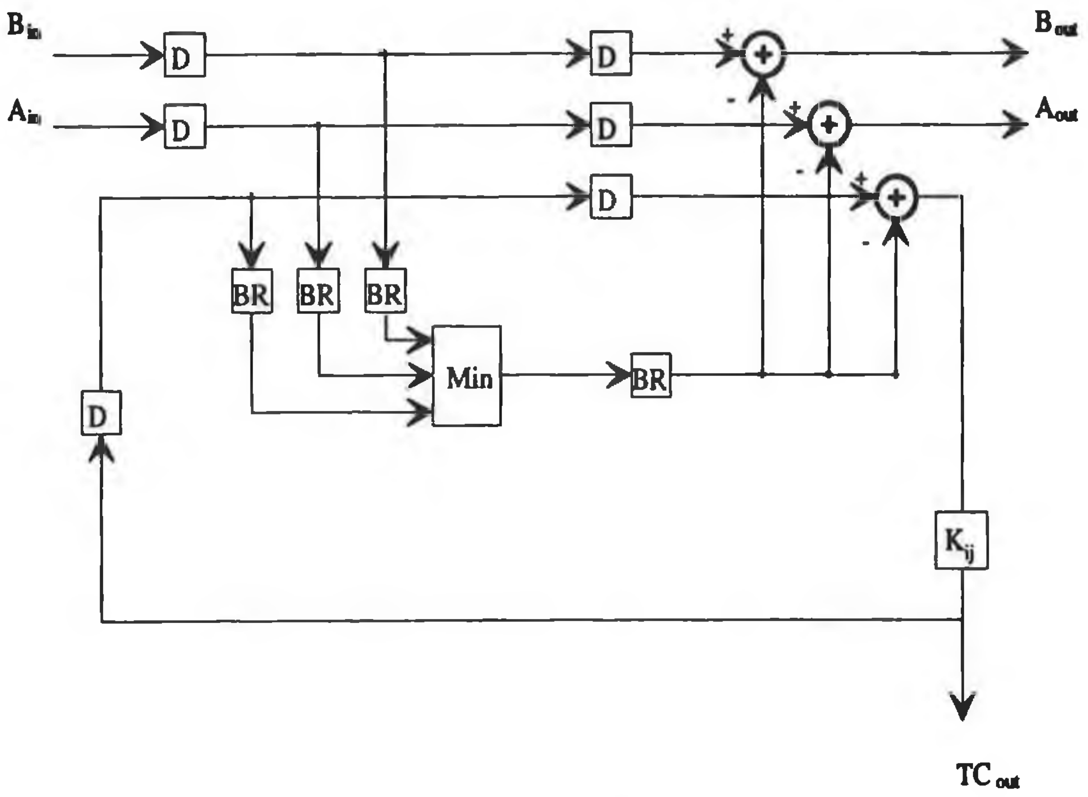
and

$$K = k_{n-1}\ k_{n-2} \ldots\ldots k_o.$$

The result is

$$R = min\,(A, B, K)$$

$$= r_{n-1}\ r_{n-2} \ldots\ldots r_o.$$



Min : Minimisation circuit

BR: Bit Reversal

D: Delay

TC: Token Count

Fig. 5.4: A possible Implementation of the *atomic*() processor.

A finite state machine with seven states is used. The state number is

$$S = S_A\ S_B\ S_K.$$

Note that $S_A = 1 \Leftrightarrow A$ is known to be greater than B or greater than K, that $S_B = 1 \Leftrightarrow B$ is known to be greater than A or greater than K, and that $S_K = 1 \Leftrightarrow K$ is known to be greater than A or greater than B. The case where $S = 111$ cannot arise.

The purpose of the finite state machine is to determine which of the three numbers A, B, and K is known to be greater than the minimum. If this situation is detected, the corresponding bit of the state number (S_A, S_B, or S_K) is set. Hence, the minimum corresponds to the number for which the corresponding state bit is zero. If two or more state bits are clear, the corresponding numbers must be equal in magnitude.

Initially, the state number is cleared to zero, since at first any one of A, B, or K could be the minimum. The state is then updated based on reading the bits of A, B and K, most significant bit first. It may be shown that the correct value for the state number will be obtained if the finite state machine evolves according to the following equations, where j is the number of the bit being inspected, which decrements from n-1 to zero.

$$S_A(j) = S_A(j+1) + \overline{X}.a_j ,$$

$$S_B(j) = S_B(j+1) + \overline{X}.b_j , 0 \leq j < n ,$$

and

$$S_K(j) = S_K(j+1) + \overline{X}.k_j ,$$

where

$$X = (S_A(j+1) + a_j)\ (S_B(j+1) + b_j)\ (S_K(j+1) + k_j)$$

and

$$S_A(n) = S_B(n) = S_K(n) = 0 .$$

The result R can then be found using

$$r_j = \overline{S_A(j)}.a_j + \overline{S_B(j)}.b_j + \overline{S_K(j)}.k_j , \qquad 0 \leq j < n.$$

This algorithm can be implemented in bit-serial form.

The realisation requires twenty-six gates, and uses n clock cycles to find the minimum. The three inputs are clocked serially, most significant bit first, into the circuitry, while the result R appears, most significant bit first, at the circuit output.

If A, B and K have different word lengths, the number of iterations required by the above algorithm is the number of bits in the longest word. The execution time can be reduced somewhat with a simple modification to the algorithm. Suppose B has the smallest word length with n bits, and that A and K are both m bits long. Then only n+1 iterations suffice to find the minimum. To achieve this, bit a_n is replaced by $a_n + a_{n+1} + ...+ a_{m-1}$ and bit k_n is replaced by $k_n + k_{n+1} + ... + k_{m-1}$, where '+' denotes a logical OR.

The algorithm still gives the correct result since A cannot be the minimum unless $A < 2^n$. However $A < 2^n \Leftrightarrow a_n + a_{n+1} + ... + a_{m-1} = 0$. If $A \geq 2^n$, it can be replaced by any value which exceeds 2^n-1, without introducing an error.

The execution time of the algorithm depends on the clock rate, the number of iterations required (which is a function of the switch size) and on the number of clock cycles required per iteration. The number of cycles required for the processor to complete one iteration of the algorithm will now be determined for a specific example. It is supposed that *A*, *B* and *K* have word lengths of 3, 4 and 7 bits respectively. The propagation delays between adjacent processors on the *A* ring and *B* ring are assumed to be 1 clock cycle and 2 clock cycles respectively.

Clock Cycle	Events
0	min. receives $k_6 + k_5 + k_4 + k_3$, b_3, 0.
1	min. receives k_2, b_2, a_2, outputs m_3.
2	min. receives k_1, b_1, a_1, outputs m_2.
3	min. receives k_0, b_0, a_0, outputs m_1.
4	min. outputs m_0 and forwards it to subtractors.
5	subtractors output new k_0, b_0, a_0.
6	subtractors output new k_1, b_1, a_1.
7	subtractors output new k_2, b_2, a_2, new a_0 received
9	subtractors outputs new k_4, $k_3 + k_4$; new a_2, b_1 received
10	subtractor outputs new k_5, $k_3+k_4+k_5$; new b_2 received
11	subtractor outputs new k_6, $k_3+k_4+k_5+k_6$; new b_3 received

Table 5.1: Execution schedule for the circuit of Fig. 5.4.

Cycle zero is assumed to be that cycle immediately before the minimum-calculating circuit (abbreviated below to min.) produces its first output bit. Hence execution proceeds according to Table 5.1.

It is apparent that cycle 11 corresponds to cycle zero of a new iteration. Thus one iteration of the algorithm requires eleven clock cycles. This would be increased to 9+Z if Z > 2, where Z is the propagation delay between processors on the *B* ring (in clock periods) since the new b_3 would not then be received until cycle 9+Z.

The value of Z will be implementation-dependent. It will depend on the distance between processors on the *B* ring, the group velocity on the ring, and on the drive capabilities of the B_{out} circuitry of each processor. This must be designed such that less than half a clock cycle is required for a transition between voltage levels, as otherwise the system could not be clocked at the required rate. Assuming that this can be achieved, and assuming reasonable values for processor separation and group velocity, a value for *Z* of one cycle or less appears achievable in practice, for clock rates up to several hundred MHz.

5.3.3 A faster bit-serial implementation.

The execution time per iteration can be further reduced by clever design of the *atomic*() processor. A faster bit-serial implementation of the processor is shown in Fig. 5.5. This calculates all possible results for the new values of *A*, *B* and *K*, and discards those which are incorrect. The logic equations governing its operation are described in detail below. The execution schedule for this circuit, for the example considered in section 5.3.2, is shown in Table 5.6. The execution time of this implementation is 82% of the earlier design, for a fixed clock rate. Such an incremental improvement in circuit performance may be crucial to the successful implementation of the path allocation algorithm, in the case where the required clock rate approaches the limit achievable with a given fabrication process.

The number of clock cycles per iteration is larger than it would otherwise be in the bit-serial implementation above because of the necessity of bit reversal. This requirement can be avoided by adopting an alternative strategy, whereby all possible results of the subtraction are calculated, and the correct results are multiplexed onto the output lines.

The processor must calculate:

$$\begin{aligned} A' &\leftarrow A - min\,(A, B, K), \\ B' &\leftarrow B - min\,(A, B, K), \\ K' &\leftarrow K - min\,(A, B, K). \end{aligned}$$

The required operations may be rewritten as

$$\begin{aligned} A' &\leftarrow max\,(A\text{-}A, A\text{-}B, A\text{-}K), \\ B' &\leftarrow min\,(B\text{-}A, B\text{-}B, B\text{-}K), \\ K' &\leftarrow min\,(K\text{-}A, K\text{-}B, K\text{-}K). \end{aligned}$$

i.e.,

$$\begin{aligned} A' &\leftarrow max\,(0, A\text{-}B, A\text{-}K), \\ B' &\leftarrow min\,(B\text{-}A, 0, B\text{-}K), \\ K' &\leftarrow min\,(K\text{-}A, K\text{-}B, 0). \end{aligned}$$

There are three possible sets of outcomes.

(i) if A is the minimum

$$\begin{aligned} A' &\leftarrow 0 \\ B' &\leftarrow B - A \\ K' &\leftarrow K - A\,; \end{aligned}$$

(ii) if B is the minimum

$$\begin{aligned} A' &\leftarrow A - B \\ B' &\leftarrow 0 \\ K' &\leftarrow K - B\,; \end{aligned}$$

(iii) if K is the minimum

$$\begin{aligned} A' &\leftarrow A - K \\ B' &\leftarrow B - K \\ K' &\leftarrow 0. \end{aligned}$$

Thus only six subtractions need to be performed to obtain the results A', B' and K'. The key to this approach is to select which register (A, B or K) is assigned the value zero, and which pair of subtraction results is assigned to the other two registers.

The algorithm proceeds as follows.

Calculate

$$\begin{aligned} R_1 &= B - A, \\ R_2 &= A - K, \\ R_3 &= B - K, \\ R_4 &= A - B, \\ R_5 &= K - A, \end{aligned}$$

and

$$R_6 = K - B.$$

Let

$$E_1 = \begin{matrix} 1, R_1 \geq 0 \\ 0, R_1 < 0 \end{matrix}.$$

Note that $E_1 = 1 \Leftrightarrow B \geq A$. Also

$$E_2 = \begin{matrix} 1, R_2 \geq 0 \\ 0, R_2 < 0 \end{matrix}.$$

Note that $E_2 = 1 \Leftrightarrow A \geq K$. Similarly

$$E_3 = \begin{matrix} 1, R_3 \geq 0 \\ 0, R_3 < 0 \end{matrix},$$

so that $E_3 = 1 \Leftrightarrow B \geq K$.

Given the values of E_1, E_2, and E_3, the relationship between the minimum and the values of E_1, E_2 and E_3 is as shown in Table 5.2.

E register values			Corresponding Inequalities			Minimum
E_1	E_2	E_3				
0	0	0	$B<A$	$A<K$	$B<K$	B
0	0	1	$B<A$	$A<K$	$B\geq K$	x
0	1	0	$B<A$	$A\geq K$	$B<K$	B
0	1	1	$B<A$	$A\geq K$	$B\geq K$	K
1	0	0	$B\geq A$	$A<K$	$B<K$	A
1	0	1	$B\geq A$	$A<K$	$B\geq K$	A
1	1	0	$B\geq A$	$A\geq K$	$B<K$	x
1	1	1	$B\geq A$	$A\geq K$	$B\geq K$	K

Table 5.2: Truth table for determination of the minimum.

An x in the minimum column indicates a combination of E_1, E_2 and E_3 which leads to a contradiction and which thus cannot arise. Without exploiting these 'don't care' states, the minimum can be determined as in Table 5.3.

Hence, A', B' and K' may be determined in accordance with Table 5.4.

The 'don't care' states can now be exploited to yield the minimum set of conditions shown in Tables 5.5 (a) through 5.5 (c). This algorithm may be implemented in bit-serial arithmetic as shown in Fig. 5.5. The number of clock cycles required per iteration of the algorithm shall now be determined.

Suppose A, B and K are each n bits long. A_{in} , B_{in} , and K_{in} enter least significant bit first. After n clock cycles, the n-bit registers R_1-R_6 have been loaded with the correct values. On the following clock cycle, the subtractors produce a '1' if underflow occurs. Since, for example, $E_1 = 0$ if and only if underflow occurs in calculating B-A , the 1-bit registers E_1-E_3 are loaded with the correct values during this cycle. A further n iterations suffices to transmit all bits of the results to A_{out} , B_{out} , and K_{out}.

Condition	Minimum
$E_1 = 0,\ E_3 = 0$	B
$E_2 = 1,\ E_3 = 1$	K
$E_1 = 1,\ E_2 = 0$	A

Table 5.3: Minimised truth table.

Condition	A'	B'	K'
$E_1 = 1,\ E_2 = 0$	0	R_1	R_5
$E_1 = 0,\ E_3 = 0$	R_4	0	R_6
$E_2 = 1,\ E_3 = 1$	R_2	R_3	0

Table 5.4: Generation of A', B', and K'.

The absence of bit reversal in this implementation means that the pipelining in the architecture may be increased, thereby reducing the number of clock cycles per iteration. Considering a situation where A, B and K have word lengths of 3, 4 and 7 bits respectively, as was assumed in the schedule of Table 5.1, the number of clock cycles required is given by Table 5.6.

Condition	A'
$E_1 \overline{E_2}$	0
$\overline{(E_1\overline{E_2})}.\overline{E_3}$	R_4
$\overline{(E_1\overline{E_2})}.\mathrm{E}_3$	R_2

(a) generation of *A'*.

Condition	B'
$\overline{E_1}\, \overline{E_3}$	0
$\overline{(\overline{E_1}\overline{E_3})}.\overline{E_2}$	R_1
$\overline{(\overline{E_1}\overline{E_3})}.\mathrm{E}_2$	R_3

(b) generation of *B'*.

Condition	K'
$E_2 E_3$	0
$\overline{(E_2 E_3)}.\overline{E_1}$	R_6
$\overline{(E_2 E_3)}.\mathrm{E}_1$	R_5

(c) generation of *K'*.

Table 5.5: Generation of *A'*, *B'* and *K'*.

It is apparent that nine clock cycles suffice for one iteration to be completed. Here it has been assumed that data transmitted on the A ring or the B ring is received by the adjacent processor after a delay of one cycle. If the propagation delay Z (in cycles) is larger, the number of cycles per iteration increases to $8 + Z$.

Each subtractor requires 14 gates to implement. Thus the total number of gates required is approximately 200. This assumes the use of static storage techniques to implement the registers (with four gates per register) and could be reduced substantially if dynamic techniques were employed.

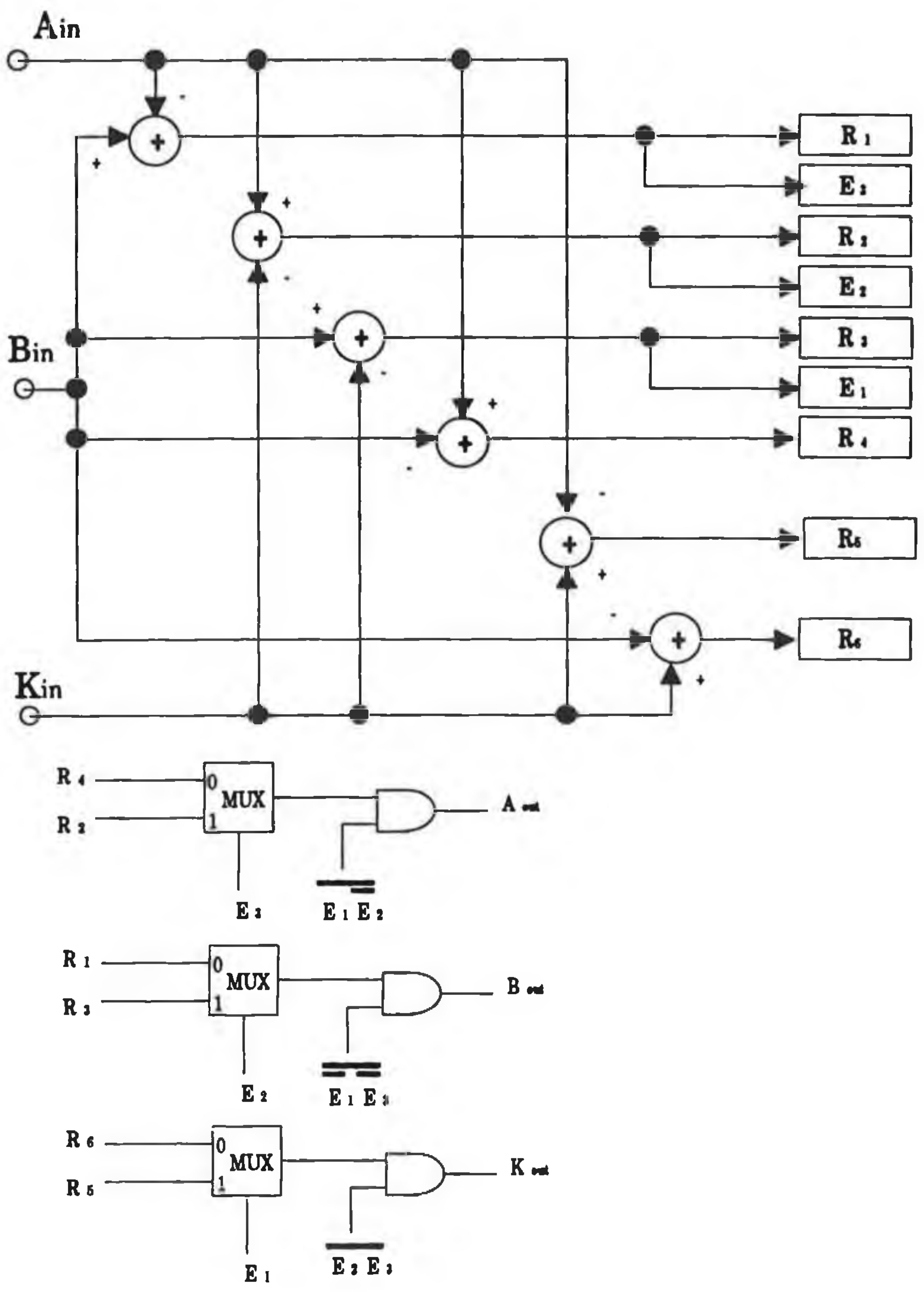


Fig. 5.5: A faster bit-serial processor.

The execution time could be reduced further if a bit-parallel implementation was used for calculations involving *K*. The values of E_1, E_2 and E_3 would then be known at the end of cycle 4, a saving of three cycles in the execution time. This saving would be unlikely to be worthwhile, given the considerable increase in the number of gates required to implement the processor. However, if the word length of *K* was greater, this might be the more economic realisation.

Clock Cycle	Actions
0	Read a_0, b_0, k_0
1	Read a_1, b_1, k_1, output r_0 to register r_2 ($1 \leq i \leq 6$)
2	Read a_2, b_2, k_2, output r_1
3	Read b_3, k_3, output r_2
4	Read k_4, output r_3
5	Read k_5, output r_4
6	Read k_6, output r_5
7	Read k_7, output r_6; E_1 E_2, E_3 available at end of this cycle
8	Calculate a_0, b_0, k_0
9	Calculate a_1, b_1, k_1; read a_0, b_0, k_0 at input side

Table 5.6: Execution schedule for the circuit of Fig. 5.5.

5.3.4 Bit-parallel implementations.

If bit-parallel arithmetic is used, the number of cycles required can be reduced. The amount of reduction depends on the complexity of hardware used. For example, the minimisation circuit for the example above has eleven inputs and three outputs. If combinatorial logic is used to implement it, the output will be available within one cycle. The use of sequential logic decreases the complexity, but now more than one clock cycle is required to obtain the result. Thus, if one iteration of the algorithm is required to execute within a fixed amount of time, a trade-off may be performed between circuit complexity and operating speed. The number of iterations required of the algorithm is $2+Z+t_m+t_s$, where t_m and t_s are the number of clock cycles in excess of one required by the minimisation and subtraction operations. Thus, assuming a hardware of reasonable complexity, calculation of the minimum would be performed in two cycles (operating on two numbers per cycle) as would subtraction. With the previously considered value of Z, this gives an iteration duration of six cycles, as compared with eleven cycles for the bit-serial implementation. Thus the clock rate required is approx. 50% of that required for bit-serial operation.

The bit-serial implementation requires approximately 90 gates. The bit-parallel version could have various numbers of gates, depending on implementation, but 220 might be a typical figure. Thus the product of clock rate and number of gates is approximately constant.

5.3.5 Processor implementation in CMOS.

The feasibility of implementing an *atomic*( ) processor has been demonstrated above. Three possible design strategies have been discussed. It has been shown that the processor may be implemented with 100-200 gates. Thus many copies of the processor could be fabricated on a single chip. It follows that the chip size may be limited, not by the processor complexity, but by pinout limitations.

The feasibility of the design may be demonstrated by reference to the state of the art in MOS IC design for broadband communications, as typified by the following sample of results from the literature.

Hickey and Marcus [10] described a CMOS implementation of the Sunshine switch, described in Chapter Two. Their basic building blocks were 32 x 32 Batcher and banyan chips, and they achieved a data rate of 170 Mb/s, using a 1.2 μm process. However, they reported difficulties in constructing larger switches from these modules, due to problems in maintaining bit synchronisation.

Other authors have reported switch designs operating at data rates greater than or equal to the ATM rate. Hoffman and Muller [11] described a single-chip 16 x 8 shared-buffer switch which operates at a data rate of 180 Mb/s. The chip contains 770k transistors in a 1 μm process, and consumes 3W. Weston *et al.* [12] designed two integrated circuits comprising a multiplexor-demultiplexor pair. The process used was 0.75 μm NMOS. Each chip contained approx. 1k transistors, and consumed 0.5 W with a 3.5 V logic swing. The multiplexor output was at a rate of 622.08 Mb/s.

High-speed logic systems of considerable complexity can be fabricated on one chip. Yano *et al.* [13] described the fabrication of a static 32-bit ALU for a 250 MHz RISC system. A 0.3 μm 3.3 V BiCMOS process was used, the number of transistors on the chip was 14k, and power consumption was quoted as 39 mW. This suggests that the row of *atomic*( ) processors required for each input module could be implemented using only a few chips. However, the requirement to transmit the B_{out} signal off-board will slow the *atomic*( ) processors.

The achievable off-chip data rate of CMOS integrated circuits is much less than that on-chip. This problem was considered by Ishibe *et al.* [14] who described the implementation of high-speed CMOS I/O buffer circuits. They fabricated current-mode buffers in a 0.5 μm process, which could be designed to interface with ECL or voltage-mode CMOS circuits. A 700 Mb/s inter-chip data rate could be achieved.

The results reported by the above researchers suggest that it will be possible to construct a switching system of the type proposed here using CMOS or BiCMOS technology in the short term, provided that a robust technique is used to maintain bit synchronisation across the system.

5.3.6 Processor speed requirements for an input-queued switch.

The time within which the algorithm is required to execute depends on whether cells losing contention are discarded or are queued until the next time slot. Consider the case where cells losing contention join an input queue. The queue controller, when it submits a cell to the path allocation process, retains a copy in the input buffer. It then awaits an acknowledgement signal from the path allocation hardware, indicating whether the cell has been allocated a path through the switch, or has been discarded. It then submits the copy cell (if the original cell was discarded) or it purges the copy cell and submits the next cell in the input buffer (if the first cell was successfully routed). The acknowledgement must be returned within the duration of one time slot so that cells can be submitted to the switch in successive time slots. Hence the time taken for the path allocation process to execute should be less than the duration of one time slot. This stringent requirement could be relaxed if preservation of cell sequence was not mandatory, since a cell losing contention could then rejoin the queue in a later time slot.

No acknowledgements are required if there is no input queuing. Hence the path allocation process need not execute within one time slot. However, it must still be possible to submit cells to the switch in successive time slots. Additional copies of the path allocation hardware are required to ensure this, equal in number to the execution time of the path allocation process in time-slots. For example, if the path allocation process has an execution time of two time slots, cells arriving during even-numbered time slots will be processed by one copy of the hardware, and cells arriving during odd-numbered time slots will be processed by another. Hence a trade-off may be performed during switch design between processor speed and the number of processors required. Omitting the input queues has the additional advantage that no hardware is required to generate the acknowledgements.

5.4 Additional hardware.

5.4.1 Hardware functions.

Hardware is also needed in each input module to perform the following tasks before and during the path allocation process :

- to count the number of requests for each output module so as to obtain the initial values of the K_{ij}'s;

• to forward a routing tag based on the results of path allocation to each input cell.

Two methods for implementing the former task will now be described.

5.4.2 A sequential algorithm for counting requests.

The counting of requests can be performed by the hardware of Fig. 5.6 [8]. This merges the input cells with a set of control packets, one for each input module, in a Batcher sorter (with n_1+L_2 inputs and outputs), in the manner described in [15]. Idle inputs submit an inactive packet to the sorter. The sorter output contains (starting at the *lowest*-numbered output in Fig. 5.6) the control packet for output module 0, followed by all the data cells intended for output module 0, followed by the control packet for output module 1, etc. The inactive packets are sorted to the highest-numbered outputs. This is achieved using a routing tag of the type shown in Fig. 5.7.

Here A = 1 (0) for an inactive (data or control) packet and $D/\overline{C}$ = 1 (0) for a data cell (control packet).

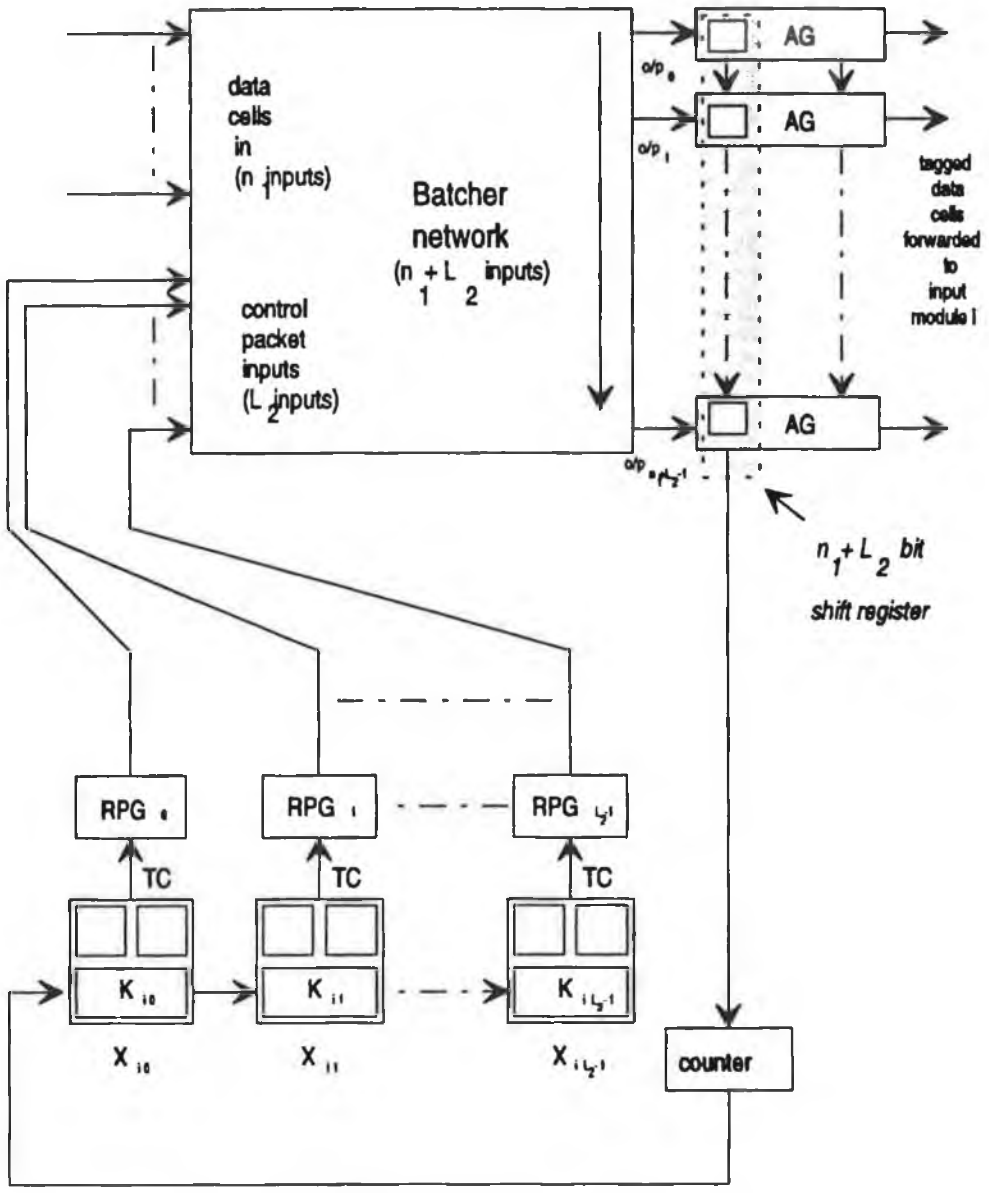
The address generators in Fig. 5.6 serve different purposes during the counting of requests and in routing tag assignment. When counting requests, they determine the type of packet which is present at the corresponding output of the Batcher network, and generate a bit (the *identity* bit) which is 1 for a control packet or an inactive packet and 0 for a data packet (cell). A copy of the identity bit is stored in a one-bit register which is connected to its neighbours in adjacent address generators in such a way as to form a shift register.

These bits are shifted n_1+L_2 times (in the direction shown in Fig. 5.6) from address generator to address generator, and hence into a counter of wordlength $\lceil \log_2(m) \rceil$ which is reset upon receiving a 1 (a control packet or inactive packet) and incremented on receipt of a 0 (data cell). Each time a 1 is received the counter contents are rotated into $K_{i,0}$, $K_{i,0}$ is rotated into $K_{i,1}$, etc. After n_1+L_2 shifts the appropriate values are stored in the K_{ij} registers.

5.4.3 A faster method for counting requests.

The method of request counting described in the previous section has the disadvantage that it *sequentially* determines the number of cells requesting output modules L_2-1, L_2-2, etc. The total number of clock cycles required is $n_1 + L_2$, i.e., the sum of the number of input ports per input module, and the number of output modules. Hence, the required clock rate may be excessive in a large switch, given that request counting, path allocation, and routing tag assignment, must (ideally) all take place within the duration of one time slot.

A faster, parallel, implementation would simultaneously calculate K_{ij} (the number of requests from input module i for output module j) for all values of j ($0 \leq j < L_2$). Suitable hardware will now be described. The execution time for this hardware is $2\lceil \log_2(n_1 + L_2) \rceil + 1$ clock cycles, which is substantially less than that for the sequential hardware described above, at the price of an increase in hardware complexity.



data cells in (n1 inputs)
Batcher network (n1 + L2 inputs)
control packet inputs (L2 inputs)
o/p0
o/p1
o/p n1+L2-1
AG
AG
AG
tagged data cells forwarded to input module I
n1+L2 bit shift register
RPG 0
RPG 1
RPG L2-1
TC
K i0
K i1
K iL2-1
X i0
X i1
X iL2-1
counter

X_{ij} : Processor (input module i, output module j)

RPG: Routing Packet Generator

AG: Address Generator

Fig. 5.6: Hardware for counting requests and generating routing tags.

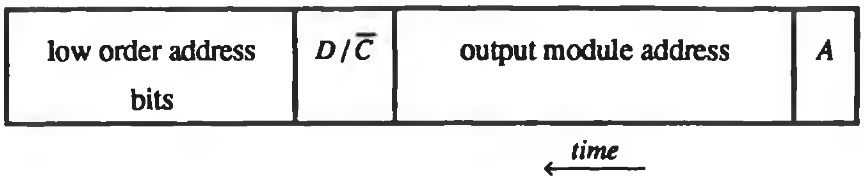


Fig. 5.7: Routing tag for Batcher network.

The hardware required is shown in Fig. 5.8 [9]. Data cells from the n_1 input ports associated with input module i are merged with L_2 control packets (one per output module) by a Batcher sorting network. The merge operation is performed in such a way that idle cells (i.e., empty cells from inactive input ports) are sorted to the highest-numbered output ports of the Batcher network. This is achieved by inverting the $D/\overline{C}$ bit in the routing tag shown in Fig. 5.7.

If the control packet for output module j appears at output D_j of the Batcher network, then the data cells (if any) requesting that output module appear at lower-numbered output ports of the sorter (ports D_j-1,D_j-2, etc.), as shown in Fig. 5.8.

Under these circumstances, it may readily be shown that

$$D_j = \sum_{u=0}^{j} K_{iu} + j$$

where K_{iu} is the number of data cells requesting output module u, and i is fixed, since the Batcher network processes only requests from input module i.

The key to the new method of request counting is the observation that

$$K_{ij} = \begin{matrix} D_0, & j=0 \\ D_j - D_{j-1} - 1, & j>0 \end{matrix}.$$

The necessary subtraction can be performed very efficiently, since

$$D_j - D_{j-1} - 1 = D_j + \overline{D}_{j+1}$$

where $\overline{D}$ is the 1's complement of D, obtained by bitwise inversion of D. It follows that the value of K_{ij} can be generated using a serial adder, and can then be stored in the K register of the appropriate *atomic*() processor, labelled X_{ij}.

It is necessary to generate a concentrated list of the values $D_0, D_1, D_2, \ldots, D_{L_2-1}$ as input data for the serial adders. These values are available at the address generators which have received control packets from the sorter outputs, but are not concentrated onto contiguous outputs. Hence a concentrator is required. This is the purpose of the inverse S-banyan shown in Fig. 5.8.

The address generators forward only control packets to this network. Address generators which have received a data cell or an idle cell through the Batcher network submit an inactive packet to the concentrator. The address generator which receives control packet j from output D_j of the Batcher network appends a data field to the packet containing the value of D_j. This packet is then routed to output j of the concentrator. A total of L_2 control packets is thus simultaneously launched into the concentrator, and these are routed to the serial adders at outputs zero through L_2-1, without blocking. The absence of blocking within the concentrator will now be verified.

It was shown in Chapter Four that the inverse S-banyan is non-blocking if, for any pair of input ports I_1 and I_2, requesting output ports O_1 and O_2 respectively, with $I_2 > I_1$ and $O_2 > O_1$, the following is true:

$$O_2 - O_1 \leq I_2 - I_1.$$

In the present application, the input ports are

$$I_1 = D_j$$

and

$$I_2 = D_k$$

and the output ports are

$$O_1 = j$$

and

$$O_2 = k,$$

with $k > j$.

Hence the non-blocking condition becomes

$$k - j \leq D_k - D_j.$$

The number of requests for each output module must be non-negative, i.e.,

$$K_{ij} \geq 0.$$

Since

$$D_k - D_j = \sum_{u=j+1}^{k} K_{iu} + k - j$$

it follows that the condition for the network to be non-blocking is always satisfied.

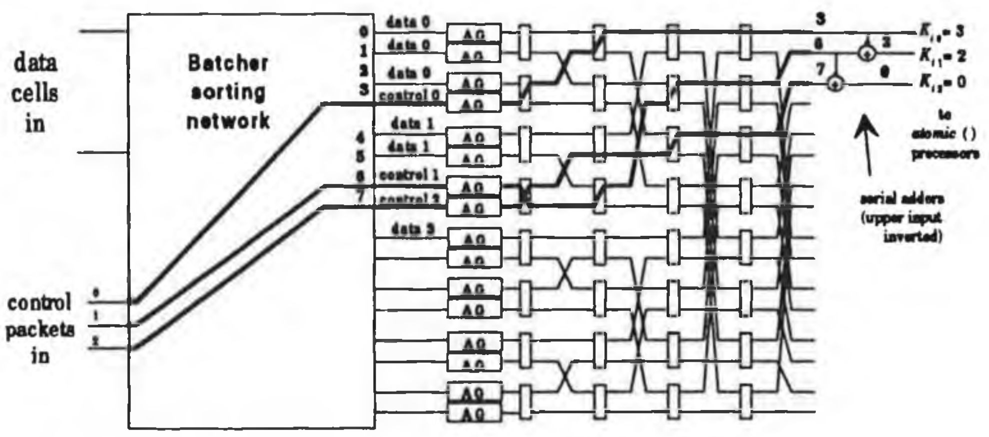
The concentrated list of D_j values is read by the serial adders shown in Fig. 5.8, the upper input being inverted. Hence the K_{ij} values are generated, and passed to the *atomic*() processors. The example considered shows three requests for output module zero, two for output module one, and none for output module two. It can be seen that the correct values (i.e., 3, 2 and 0) are returned to processors

X_{i0}, X_{i1} and X_{i2} respectively.

The submitted packets take two cycles to propagate through each stage of the concentrator (one cycle to identify if the packet is active, and another to determine where to route it) and an additional clock cycle is required before the serial adder generates the least significant bit of the appropriate K_{ij} value. Hence the number of clock cycles required by the request count hardware before path allocation can commence is

$$2\lceil \log_2(n_1 + L_2) \rceil + 1.$$

This compares favourably with the $n_1 + L_2$ clock cycles required by the previous algorithm.



AG: Address Generator

Fig. 5.8: A faster method of counting requests.

5.5 Routing tag assignment.

5.5.1 Principles of the method.

The algorithm used for routing tag assignment is best described by means of an example. We assume that there are four intermediate switch modules, labelled ISM_0 to ISM_3, and consider how to assign routing tags to the cells from input module zero (IM_0) which have requested output module one (OM_1).

Let us assume that four cells from IM_0 have requested output module zero (OM_0), and that seven cells from IM_0 have requested OM_1. Hence, the first thirteen outputs of the Batcher sorter are as shown in Fig. 5.9, after the data cells have been merged with control packets, as discussed in section 5.4.2.

The results of the path allocation process for cells from IM_0 requesting OM_1

are produced by processor X_{01}. Some means must be found to forward these results to the relevant cells, which appear at sorter outputs six through twelve. We assume that paths are allocated in the order shown in Table 5.7. Thus one cell loses contention.

iteration	0	1	2	3
routing via	ISM_1	ISM_0	ISM_3	ISM_2
no. of paths	3	1	0	2

Table 5.7: An example of path allocation.

Note that a connection has already been established from processor X_{01} (via the routing packet generator for OM_1) to sorter output five, as shown in Fig. 5.9. Thus the relevant routing information may be easily forwarded to sorter output five. The problem remains of how to relay this information to the data cells at sorter outputs six through twelve.

The solution to this problem would be trivial if all seven cells were to be allocated a route via the same intermediate switch module. The address generator at sorter output five could generate seven tokens, each granting access to that intermediate switch module. These seven tokens would be forwarded to sorter output six, where one would be seized. The remaining six tokens would be forwarded to sorter output seven. After seven iterations of this procedure, all seven data cells would possess the necessary token, stored in the associated address generator.

5.5.2 Sequential Implementation.

This token passing algorithm is easily implemented in hardware. Each address generator stores data concerning tokens as a routing packet containing two fields, which are the token address (indicating the address of the intermediate switch module to which access is being granted) and the token count (indicating the number of such tokens). Routing packets are received by the address generators associated with control packets (such as that at output five in our example), from the appropriate routing packet generator. Other address generators receive a routing packet from their neighbours at the start of each iteration of the algorithm. If the token count is zero, this routing packet is discarded. Otherwise the token count is decremented, and the routing packet is stored in the address generator, and is forwarded to the adjacent generator at the start of the next iteration.

The hardware required is thus very simple, with a unidirectional flow of data from address generator to address generator. Once all the tokens have been distributed (after seven iterations), subsequent iterations of the algorithm cause no changes in the route assignments, so that the number of iterations performed is not critical, provided it is not too low.

The same hardware may be used to solve the problem of assigning routing tags to the data cells when paths through more than one intermediate switch module have been allocated. The technique is to execute multiple passes of the above algorithm, each for a different token address, and each with an appropriately chosen value for the token count. An address generator may then receive a succession of routing packets, each with a different token address. However, only the last such packet received will contain the correct token.

Five passes of the algorithm suffice to supply all the data cells with the correct token in the example of Fig. 5.9, as shown below.

Pass One: Seven tokens are received for ISM_1. At the end of this pass, all seven cells have been assigned a route through ISM_1.

Pass Two: Four (i.e., 7-3) tokens are received for ISM_0. At the end of this pass, four cells have been assigned a route through ISM_0. Three cells (i.e., those at sorter outputs ten through twelve) have retained their tokens for ISM_1.

Thus, at the end of Pass Two, the correct number of cells has been assigned a route through ISM_1.

Pass Three: Three (i.e., 7-3-1) tokens are received for ISM_3.

Pass Four: Three (i.e., 7-3-1-0) tokens are received for ISM_2.

Pass Five: One (i.e., 7-3-1-0-2) null token is received, indicating that one cell has lost contention.

At the end of Pass Five, a route through each intermediate switch module has been assigned to the correct number of cells, as shown in Fig. 5.9. In particular, no route is assigned via ISM_3. The key to achieving the required result was the correct choice of token count for each pass of the algorithm. The sequence chosen was {7,4,3,3,1}. This is identical to the sequence of K_{01} values generated by processor X_{01} during the path allocation process, i.e., it is equal to the number of cells not yet allocated a path after each cycle of path allocation. Thus, the necessary sequence of token counts may be obtained from the TC_{out} output of the processor, shown in Fig. 5.4.

Each pass of the algorithm can commence after the first iteration of the preceding pass. Hence the execution time increases only slowly with the number of passes. Routing tag assignment can thus be carried out as follows [8].

The routing packet generator associated with processor X_{ij} generates a routing packet, concurrently with each iteration of the path allocation algorithm. The token address is the value of *r*, the intermediate switch module through which the processor is attempting to route cells. This value can be easily generated by a counter decremented after every iteration of the path allocation algorithm. The token count is set to the value of K_{ij}. This routing packet is forwarded to the address generator associated with control packet *j* through the Batcher network in Fig. 5.6.

Each address generator performs the actions illustrated in Fig. 5.10 after every iteration of the path allocation algorithm. Upon completion of the path allocation algorithm, the value of K_{ij} is equal to the number of cells which have lost contention. This is forwarded to the relevant cells in a special routing packet, whose token address field indicates that these are null tokens, which flag the corresponding cells as having lost contention.

Upon completion of this process, the token address and token count values stored by each address generator comprise a unique routing tag, which is then prefixed to the associated data cell. The cell is submitted to the first stage of the switch, and is thereby routed to the appropriate intermediate switch module.

The operation of this algorithm for the example considered earlier is shown in Table 5.8. The data stored in each address generator at the end of each iteration of the algorithm are shown. After nine iterations, the routing tags have been successfully assigned.

It shall now be shown that the address generator is of modest complexity. A possible implementation is shown in Fig. 5.11 in the case where the token count occupies 3 bits and the tag is also three bits long. A description of the functions performed by the various registers shown in Fig. 5.11 follows.

A bit-parallel arithmetic is used in the interest of high-speed operation. This requires special care because there is then a mismatch between the rate at which an address generator may load data (in parallel) from the address generator above it, and the rate at which it may receive data (serially) through the Batcher network. The solution to this problem of mismatched data rates is to generate dummy outputs with zero token counts while data is being read from the Batcher network. These dummy routing packets are then ignored by the next address generator in line. The inputs and outputs are as follows.

S_{in} is the serial output from the Batcher network. S_{out} is the serial output to the first stage of the switch. TA_{in} (TA_{out}) is the token address from (to) the address generator above (below). TC_{in} (TC_{out}) is the token count from (to) the address generator above (below).

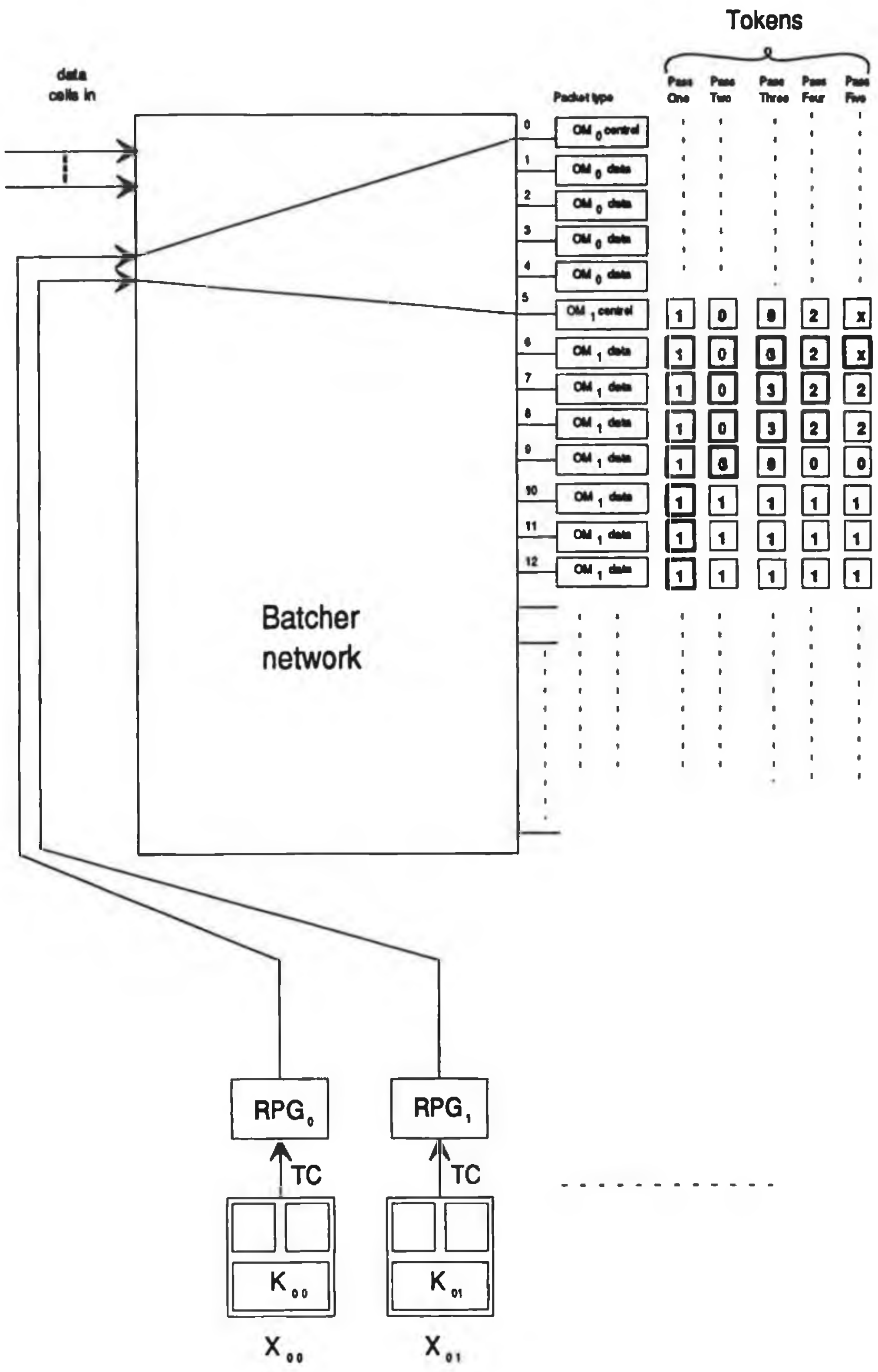


Fig. 5.9: An example of route assignment.

The address generator contains three registers, which are:

TC: this 3-bit register stores the token count. It may be loaded serially or in parallel.

TA: this 3-bit register stores the token address, and may also be loaded serially or in parallel.

I: this one-bit register stores the identity bit, which is set to one for a control packet or an inactive cell. The system need not distinguish between these two cases, since all cells below an inactive cell are themselves inactive, and so any spurious routing packets generated by the address generator will not be forwarded to active cells. This register will also be set if a null token is received.

Three simple combinational logic circuits are also shown, i.e.:

DEC: this produces an output which is equal to its input less one. Thus it decrements the input. Its output for a zero input is undefined.

NTD: this is the null token detector, whose output goes high if the token address corresponds to a null token. This is used to set the *I* register if a null token is received, thereby ensuring that only cells which have received a path allocation will be processed by the input module.

ZERO: This is simply a NOR gate, whose output is high if TC_{in} is zero.

The address generator has four modes of operation, which are described below.

First the *I* register is loaded with the correct value. It must be set if the packet received is a control packet or an inactive packet. Accordingly, the register is set to the logical OR of the activity (*A*) and identity ($D/\overline{C}$) bits, which are read in from S_{in} by clocking in the routing tag shown in Fig. 5.7.

Next the address generator enters its request counting mode. The contents of the *I* register are first copied to the most significant bit of the *TA* register. Hence the most significant bits of the *TA* registers constitute the $(n_1 + L_2)$-bit shift register indicated in Fig. 5.6. These bits can be clocked from *TA* register to *TA* register, by enabling parallel loading of these register. Hence the counter in Fig. 5.6 receives the appropriate sequence of bits.

The address generator next enters one of two distinct modes of operation, depending on the setting of the *I* register.

- If $I = 0$, the *TA* and *TC* registers load data in parallel. However, the *TC* register can only be overwritten if TC_{in} is non-zero, as determined by the ZERO circuit. Thus, the address generator will not accept routing packets with a token count of zero from the address generator above. If the token count exceeds zero, it is decremented before being stored, and the new value is forwarded to the next address generator.

- If $I = 1$, the TA and TC registers load data serially. The generator must periodically wait while a new routing packet is shifted into TA and TC from the Batcher network. While this happens, the value of TC_{out} is cleared to zero, so that the next address generator in line will reject the spurious values appearing on the TA outputs. This is performed by the TC-reset circuit in Fig. 5.11, which simply performs a logical AND of the TC register contents and binary zero, to ensure that TC_{out} is zero. When loading is complete, the TC and TA register contents can be forwarded to the next generator, since the TC_{out} value is no longer being forced to zero.

Note that if a null token is received, the I register is set to one.

Hence, it follows that the hardware implements the algorithm in Fig. 5.11, in addition to the request counting algorithm of section 5.4.2.

After routing tag allocation is completed, the data cell is clocked through the address generator from S_{in} to S_{out}, in such a way that

(i) the cell is preceded by a new activity bit, which equals the value of the I register;

(ii) this is followed by the contents of the TA register;

(iii) this is followed by the original routing tag of the cell.

Note that the A and $D/\overline{C}$ bits in the original routing tag (Fig. 5.7) are deleted and the resulting one-bit gaps in the routing tag are closed up. Hence the input module receives a cell from the address generator with a routing tag of the type shown in Fig. 5.12.

It follows that the input module only recognizes cells with $A' = 0$ (i.e., active data cells which have been allocated a path through the intermediate stage) as active data, and that it will forward these cells to the appropriate intermediate switch modules.

The routing assignment algorithm described above has the benefit of simplicity but its execution time is quite long. However, it operates in parallel with the path allocation process, although it takes longer to execute, because of the delay in propagating routing packets through the address generators. A faster algorithm will now be described.

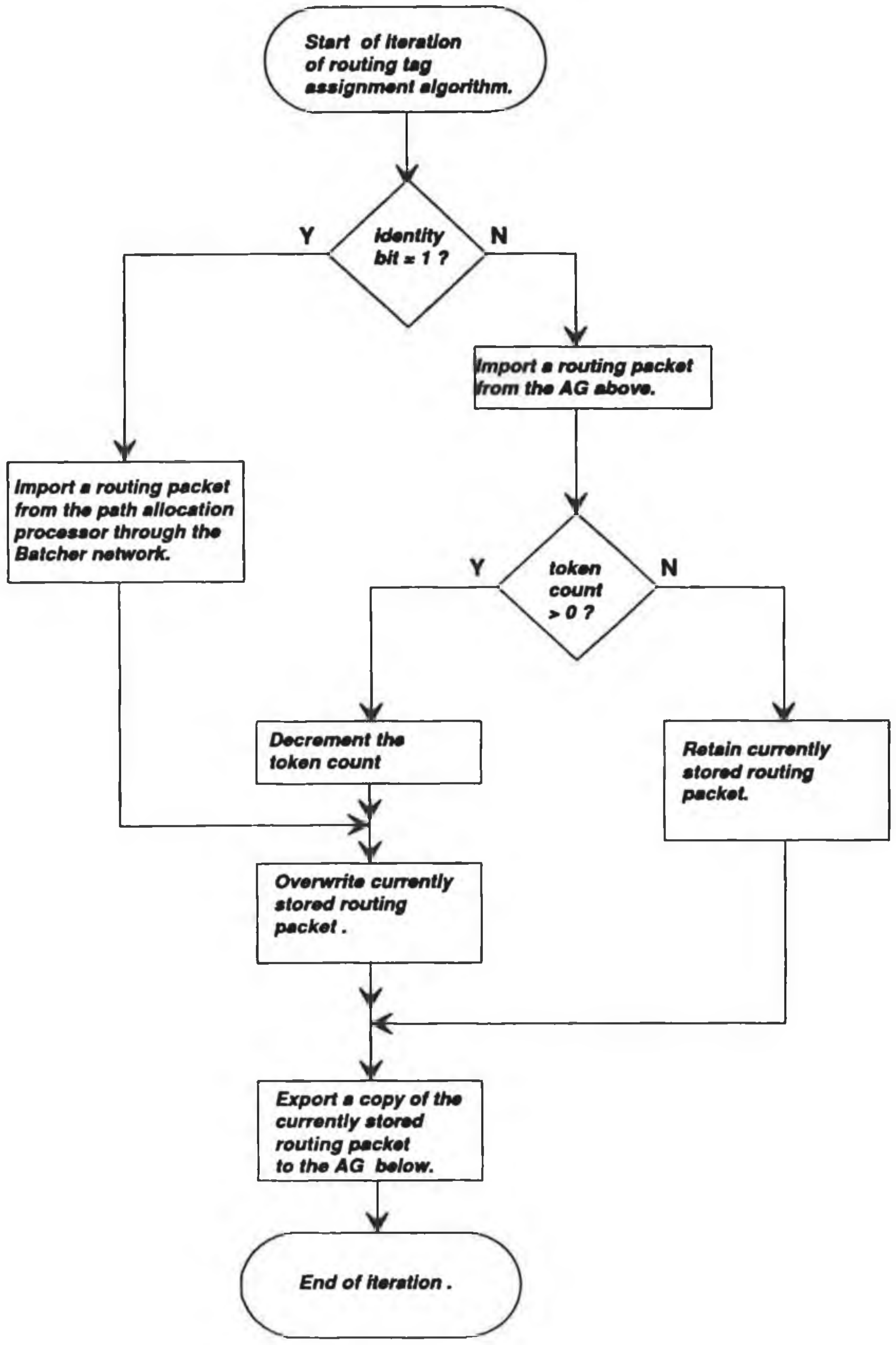

Start of iteration of routing tag assignment algorithm.
identity bit = 1 ?
Y
N
Import a routing packet from the AG above.
Import a routing packet from the path allocation processor through the Batcher network.
token count > 0 ?
Y
N
Decrement the token count
Retain currently stored routing packet.
Overwrite currently stored routing packet .
Export a copy of the currently stored routing packet to the AG below.
End of iteration .

Fig. 5.10: Address generator (AG) operations during routing tag assignment.

sorter o/p no.	packet type	iteration 0	1	2	3	4	5	6	7
		routing packet contents (*token address, token count*)							
5	*OM_1 control*	1,7	0,4	3,3	2,3	x,1	x,0	x,0	x,0
6	*OM_1 data*	-	1,6	0,3	3,2	2,2	x,0	x,0	x,0
7	*OM_1 data*	-	-	1,5	0,2	3,1	2,1	2,1	2,1
8	*OM_1 data*	-	-	-	1,4	0,1	3,0	2,0	2,0
9	*OM_1 data*	-	-	-	-	1,3	0,0	0,0	0,0
10	*OM_1 data*	-	-	-	-	-	1,2	1,2	1,2
11	*OM_1 data*	-	-	-	-	-	-	1,1	1,1
12	*OM_1 data*	-	-	-	-	-	-	-	1,0

Each column represents an iteration of the algorithm.

Each row represents the contents of an address generator after it has seized a token.

'x' indicates a null token.

Table 5.8 : An example of routing tag assignment.

5.5.3 Routing tag assignment using a copy network.

Routing tag assignment may be performed more rapidly using a modified version of Lee's copy network [16]. This network is used to broadcast a routing tag simultaneously to all the address generators which should receive it. The operation of Lee's copy network will now be briefly reviewed.

Lee's copy network is an S-banyan whose routing algorithm has been modified to allow broadcasting to a continuous set of output addresses. The routing tag for a 2^n-input network consists of $2n$ bits, viz., the address of the lowest requested destination $l_{n-1}l_{n-2}$ ....l_0 and the address of the highest requested destination $h_{n-1}h_{n-2}$ ......h_0. The values of l_k and h_k are inspected in order to determine the routing at stage k.

There are two possible outcomes of the switching process at stage k.

(i) If $l_k = h_k$, the packet is routed to the appropriate switch element output, and the upper and lower addresses are unchanged. This case corresponds to the usual unicast routing algorithm for the S-banyan.

(ii) If $l_k = 0$ and $h_k = 1$ packet replication occurs. One packet is routed to output zero, with an unchanged lower address, and an upper address of $h_{n-1} \ldots\ldots h_{k+1}\ 0\ 1 \ldots\ldots 1$. The other packet is routed to output one, with an unchanged upper address, and a lower address of $l_{n-1} \ldots\ldots l_{k+1}\ 1\ 0 \ldots\ldots 0$.

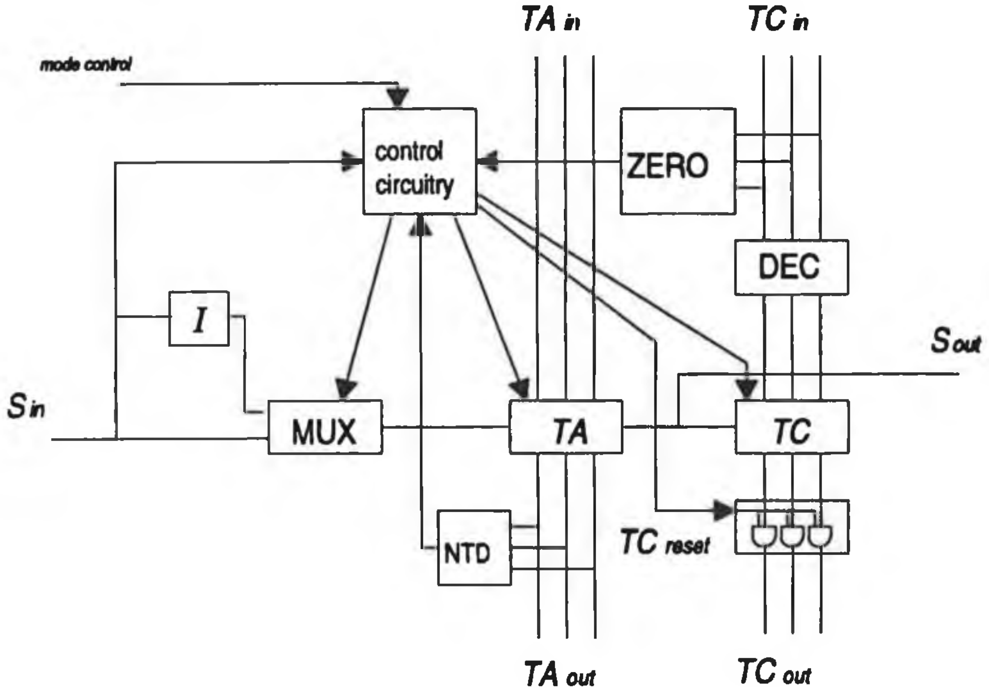


Fig. 5.11: A possible address generator implementation.

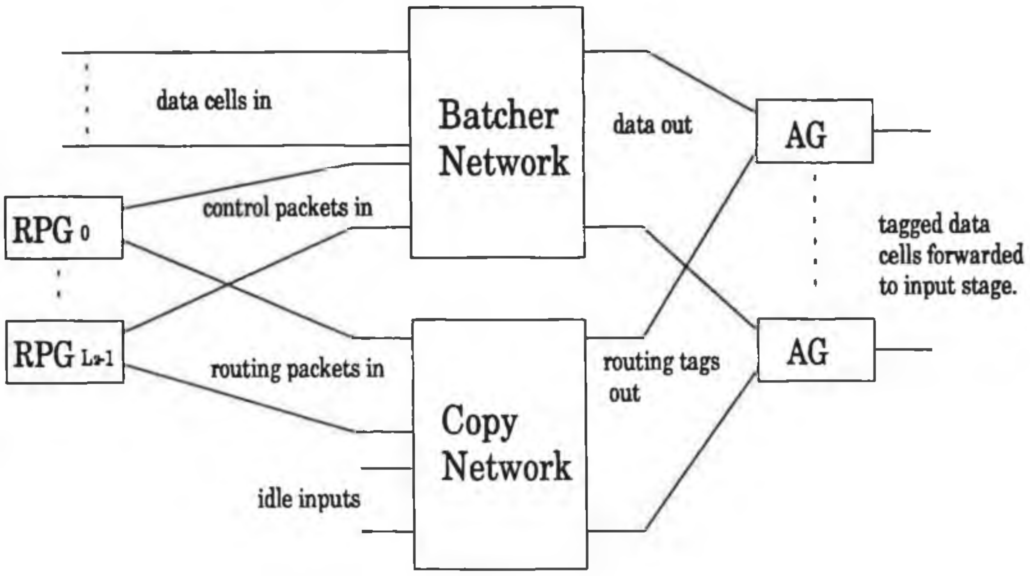
low order address bits	output module address	intermediate module address	*A'*

$\xleftarrow{\textit{time}}$

Fig. 5.12: Routing tag after path allocation.

Note that the case where $l_k = 1$ and $h_k = 0$ cannot arise, since, by definition, the upper address is greater than or equal to the lower address, and $l_j = h_j$ for $k < j < n$.

Lee refers to the above algorithm as the Boolean interval splitting algorithm. In practice, since the routing in stages k-1 through zero depends only on $l_{k-1}\ l_{k-2}, \ldots\ldots l_0$ and $h_{k-1}\ h_{k-2}, \ldots\ldots h_0$, the upper address for output zero in case (ii) need only be $x_{n-k}1_{k-1}$ where x is a 'don't care' and x_{n-k} (1_{k-1}) means a sequence of n-k (k-1) consecutive x's (1's) respectively. Similarly, the lower address for output one in case (ii) may be $x_{n-k}\,0_{k-1}$, without giving rise to routing errors.



RPG : routing packet generator

AG : address generator

Fig. 5.13: A faster method of routing tag allocation.

Routing tag allocation proceeds as follows.

A copy network with $n_1 + L_2$ inputs and outputs is used. The routing packet generators are connected to L_2 of the inputs, and the remaining inputs are idle. The outputs are connected to the address generators at the output of the Batcher network in Fig. 5.6. Hence the implementation is as shown in Fig. 5.13.

The cells requesting output module j appear at outputs $D_{j-1} + 1$ through $D_{j-1} + K_{ij}^0$ of the Batcher network, where K_{ij}^0 is the initial value of K_{ij}, the number of unallocated requests for output module j.

The value of D_j is the address of the sorter output port at which control packet j appears. Evidently

$$D_j = \sum_{u=0}^{j} K_{iu}^0 + j\,.$$

It is assumed that the value of D_{j-1} has been forwarded to RPG_j. This could be done be appropriately modifying the request count hardware.

Each broadcast by the copy network implements a single pass of the routing tag assignment algorithm. All routing tags are delivered simultaneously to the intended address generator, so that the delays associated with the previously described hardware are avoided.

During each pass of the algorithm, RPG_j submits a routing packet to the copy network, to be broadcast to address generators $D_{j-1} + 1$ through $D_{j-1} + K_{ij}$, containing in the data field the token address, i.e., the address of the intermediate switch module through which a route has been allocated. If $K_{ij} = 0$, an inactive packet is submitted.

All address generators receive routing tags simultaneously, so that the assignment of routing tags is complete once the last set of routing tags has been broadcast through the copy network. Hence there is no delay, after path allocation has concluded, for the tags to propagate sequentially through the address generators, as was the case with the hardware of Fig. 5.9.

The routing packets submitted to the copy network do not collide, since they satisfy the condition for avoiding internal contention in Lee's copy network.

It may be shown, by a straightforward extension of the non-blocking property of the S-banyan discussed in Chapter Four, that two broadcasts, from input I_1, to outputs O_1^L through O_1^H, and from input I_2 to outputs O_2^L through O_2^H, where $O_2^L > O_1^H$, do not block if

$$O_2^L - O_1^H \geq |I_2 - I_1|.$$

The proof of this result follows from the observation that

$$|O_2 - O_1| \geq O_2^L - O_1^H$$

if

$$O_1^L \leq O_1 \leq O_1^H < O_2^L \leq O_2 \leq O_2^H .$$

A special case of this non-blocking condition will now be considered, which is relevant to the problem of routing tag assignment.

Consider a copy network with L_2 inputs. Input j broadcasts to a total of K_{ij} outputs, with $K_{ij} \geq 0$. Specifically, these destinations are outputs $D_{j-1} + 1$ through $D_{j-1} + K_{ij}$, where

$$D_j = \begin{matrix} D_{j-1} + K_{j-1}^0 + 1, & j \geq 0 \\ -1, & j = -1 \end{matrix} ,$$

and where

$$K_j^0 \geq \ 0.$$

The necessary condition on K_{ij} such that the network is non-blocking will now be determined.

The broadcasts from inputs j and l (with $j > l$) do not collide if, in accordance with the non-blocking property above,

$$D_{j\text{-}1} + 1 - (D_{l\text{-}1} + K_{il}) \geq j - l.$$

Since

$$D_l = D_{l\text{-}1} + K^0_{ij} - 1 ,$$

this condition may be rewritten as

$$D_{j\text{-}1} + 1 - (D_l - K^0_{il} - 1 + K_{il}) \geq j - l,$$

i.e.,

$$D_{j\text{-}1} - D_l + 2 + K^0_{il} - K_{il} \geq j - l,$$

but

$$D_{j\text{-}1} - D_l = \sum_{u=l+1}^{j-1} K^0_{iu} + j - 1 - l \geq j - l - 1,$$

so that

$$D_{j\text{-}1} - D_l + 2 + K^0_{il} - K_{il} \geq j - l + 1 + K^0_{il} - K_{il} .$$

Hence, the condition is satisfied provided that

$$K_{il} \leq K^0_{il} + 1.$$

However, this constraint is always satisfied in the case of routing tag assignment, in which case

$$K_{il} \leq K^0_{il} .$$

Hence contention cannot occur during the process of routing tag assignment.

An apparent difficulty with this method of routing tag assignment is the amount of data which must be processed during each pass of the algorithm. In general, the copy network must process two bits (one each from the upper and lower address) in addition to the activity bits, at each node. Hence, the interval between successive passes of the algorithm, in bit times, will be quite large. The speed of the algorithm can be increased by observing that, in this application of the copy network, after the first pass of the algorithm, the lower address bit processed at each node never changes. Hence, on subsequent passes of the algorithm, there is no need to distribute the lower address, so that the header on the routing packet may be shortened, reducing the delay through the copy network.

The proof of the assertion that the lower address bit to be processed at each node of the copy network never changes after the first pass, is as follows.

The Boolean interval splitting algorithm proposed by Lee may be described in the following terms.

Let the lower and upper outputs of a switch element of the copy network be referred to as OUT_0 and OUT_1 respectively. Consider the switching which occurs at stage *k* of the network. The incoming packet is described by three quantities, *A* (the activity; A=1 if a packet is present; A=0 for an idle input), *L* (the lower address) and *U* (the upper address). The corresponding quantities for OUT_0 are $A(\mathrm{OUT}_0)$, $L(\mathrm{OUT}_0)$ and $U(\mathrm{OUT}_0)$, and for OUT_1 are $A(\mathrm{OUT}_1)$, $L(\mathrm{OUT}_1)$ and $U(\mathrm{OUT}_1)$.

Using this notation, it follows that

$$A(\mathrm{OUT}_0) = 1 - sgn(L - 2^k);$$

$$A(\mathrm{OUT}_1) = sgn(U - 2^k);$$

$$L(\mathrm{OUT}_0) = L;$$

$$L(\mathrm{OUT}_1) = max(L\ , 2^k);$$

$$U(\mathrm{OUT}_0) = min(U\ , 2^k - 1);$$

$$U(\mathrm{OUT}_1) = U,$$

where

$$sgn(x) = \begin{matrix} 1, & x \geq 0 \\ 0, & x < 0 \end{matrix}.$$

Note that when A=0, the values of *L* and *U* are 'don't cares'. There are three possible outcomes of the Boolean interval splitting algorithm (routing to OUT_0, routing to OUT_1, routing to both outputs). It may be shown that the above description of the data on OUT_0 and OUT_1 gives results consistent with those described by Lee, in all three cases. This proves the validity of this description of Lee's algorithm.

It is apparent that the values of $L(\mathrm{OUT}_0)$ and $L(\mathrm{OUT}_1)$ are dependent only on *L* and *k*. Hence it follows that the value of l_k for each node in the tree consisting of copy network links which are carrying a packet which originated at a single input is dependent only on *L*.

It can be concluded that two broadcasts, both from the same input port, and with (A, L, U) equal to $(1,\ L, U_1)$ and $(1, L, U_2)$ respectively, present the same bit as l_k to switching elements which lie on the tree common to (1, L, U_1) and (1, L, U_2). However, if $U_1 < U_2$, the tree associated with (1, L, U_1) contains only links which are shared with the (1, L, U_2) tree. Therefore, if the (1, L, U_1) broadcast is preceded by that for (1, L, U_2), the value of L need not be transmitted, provided the values of l_k are stored in the relevant switch elements.

In the routing tag assignment application, the broadcasts on successive passes of the algorithm involve a non-decreasing fanout, with an unchanging lower address. Hence, after the first pass, the lower address need not be transmitted. Since the network is contention-free, the lower address bit stored in a switch element will never be read by a packet from an input port other than that from which the bit was received.

The success of this approach to routing tag assignment shall be demonstrated by an example. This example is an extension of that considered in Table 5.7, and is described in Tables 5.9 through 5.11.

j	0	1	2	3
K_{0j}	4	7	0	1

Table 5.9: A pattern of requests.

The broadcasts for each pass of the algorithm are illustrated in Figs. 5.14 (a) through 5.14 (e). Also shown are the l_k bits for each switch element output. It can be seen that these never alter after the first pass of the algorithm. After the fifth pass, the correct number of cells has been allocated a path via each intermediate switch module.

The length of each routing packet (after the first) is

$$L_r = 1 + \lceil \log_2 (m+1) \rceil + \lceil \log_2 (n_1 + L_2) \rceil,$$

i.e., one activity bit, enough bits to represent the token address, and sufficient bits to represent the requested upper copy network output.

Hence, routing packets may be submitted to the network at the rate of one every L_r clock cycles. If this exceeds the number of clock cycles required for one iteration of the path allocation algorithm, an undesirable delay is introduced, whereby the path allocation can only proceed at the rate of one iteration per L_r clock cycles.

A solution to this difficulty involves a reduction in the value of L_r. The token address is not broadcast, except during the first pass. The address generator stores the token address received during the first pass, and on subsequent passes calculates the token address by decrementing the previous value. Therefore, the value of L_r can be reduced by $\lceil \log_2(m+1) \rceil$ bits.

	Iteration	0	1	2	3
OM_0	routing via	ISM_0	ISM_3	ISM_2	ISM_1
	no. of paths	0	2	1	1
OM_1	routing via	ISM_1	ISM_0	ISM_3	ISM_2
	no. of paths	3	1	0	2
OM_2	routing via	ISM_2	ISM_1	ISM_0	ISM_3
	no. of paths	0	0	0	0
OM_3	routing via	ISM_3	ISM_2	ISM_1	ISM_0
	no. of paths	1	0	0	0

Table 5.10: An example of path allocation.

Pass	1	2	3	4	5
OM_0	1,0,3	1,0,3	1,0,1	1,0,0	0,x,x
OM_1	1,5,11	1,5,8	1,5,7	1,5,7	1,5,5
OM_2	0,x,x	0,x,x	0,x,x	0,x,x	0,x,x
OM_3	1,14,14	0,x,x	0,x,x	0,x,x	0,x,x

Table 5.11: Routing packet header contents (*A*,*L*,*U*).

The advantage of the approach to routing tag allocation described in this section is that, once path allocation is complete, the tag allocation process requires only a further

$$L_r + 2\lceil \log_2(n_1 + L_2) \rceil$$

clock cycles to terminate (this is the time required to generate the final null routing packet, and propagate it through the copy network). This compares favourably with the corresponding number of clock cycles for the previously described method, which, in the worst case, is

$$[n_1 - min(S_1, S_2)].\Delta$$

clock cycles, where Δ (typically equal to one) is the number of clock cycles required to propagate a routing packet through an address generator. Which of the two strategies is to be preferred depends on the required operating speed of the circuitry.

5.6 Elimination of control packets.

5.6.1 Advantages of eliminating control packets.

An architecture using control packets is essential if the serial method of routing tag allocation is used. However, if a copy network is used, it is possible to eliminate the control packets. This reduces the dimensions of the Batcher network. Since a Batcher network has far more stages, for a given number of inputs, than a banyan network, reducing the Batcher network dimensions can reduce the overall complexity of the switch, even when account is taken of the additional hardware required, described below.

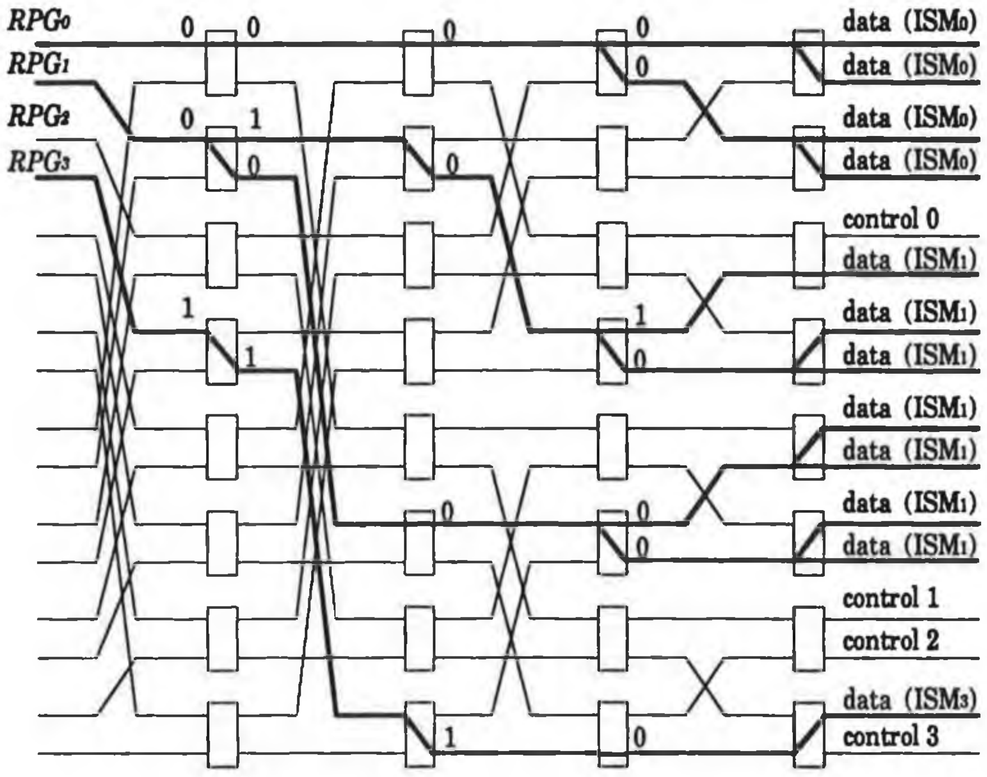
5.6.2 Request counting without control packets.

If request counting is performed using the shift register method, the following modifications must be made.

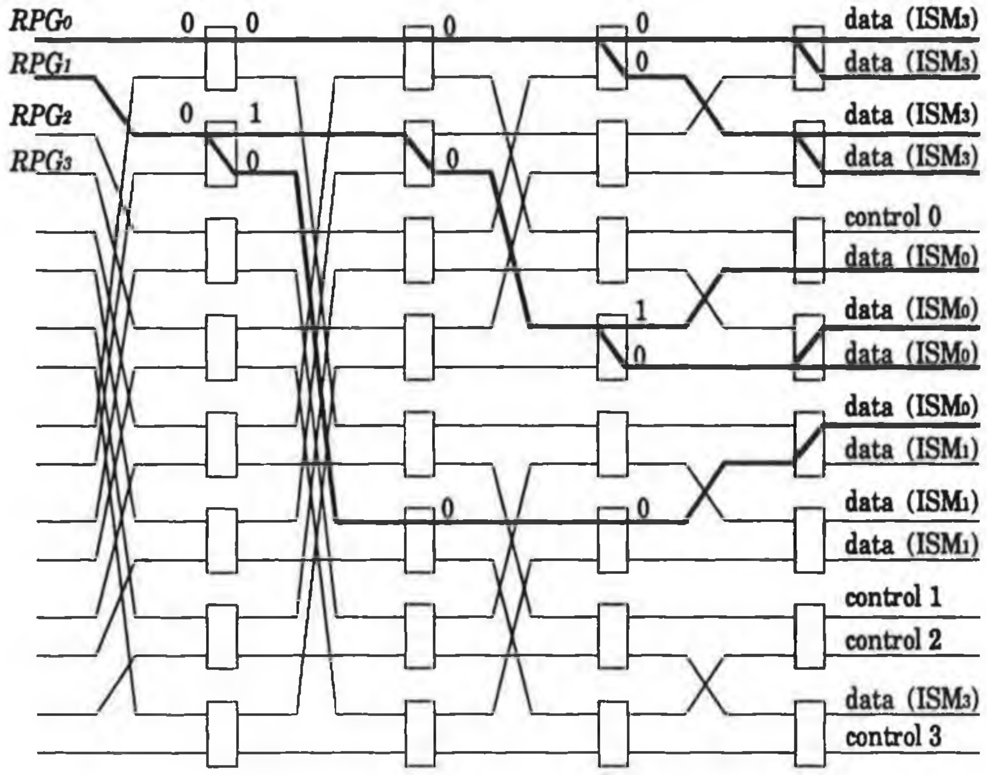
An identity bit is not generated. Instead, the cell address (including the activity bit) is shifted from address generator to address generator. The counter circuit which eventually receives these addresses is more complex than heretofore. Its operation is described below.

The counter ignores inactive cell addresses. When the first active cell address is received, an address register is loaded with the value of L_2 - 1 and the counter is reset.

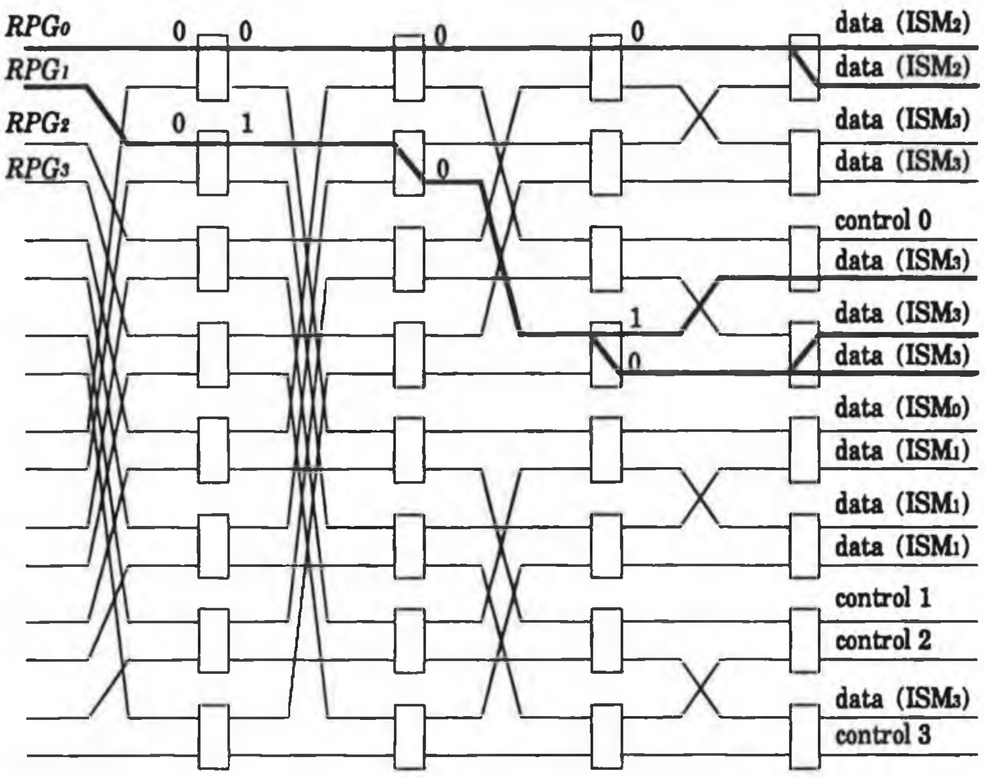
On each iteration of the algorithm (after the idle cell addresses have been clocked through), the address register contents are compared to the cell address. If they match, the counter is incremented. If there is a mismatch, the current counter contents are clocked out to the K_{ij} registers, as in Fig. 5.6, and the address register is decremented. This process is repeated until the address register matches the cell address. The next iteration of the algorithm may now be performed.



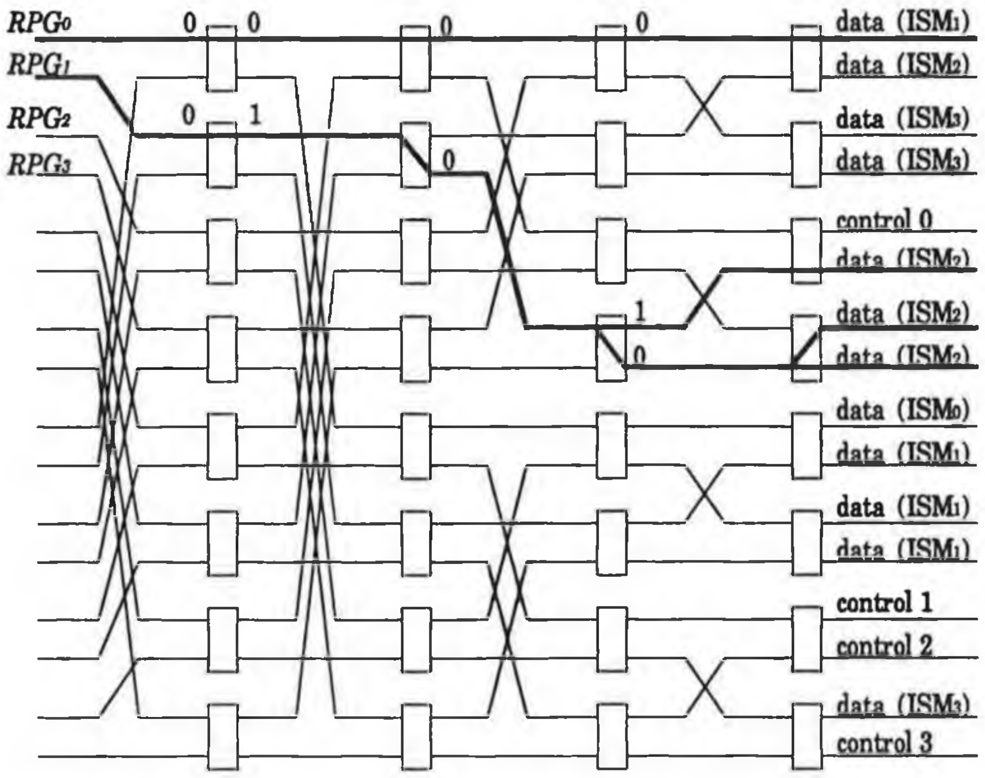
(a) Pass One.



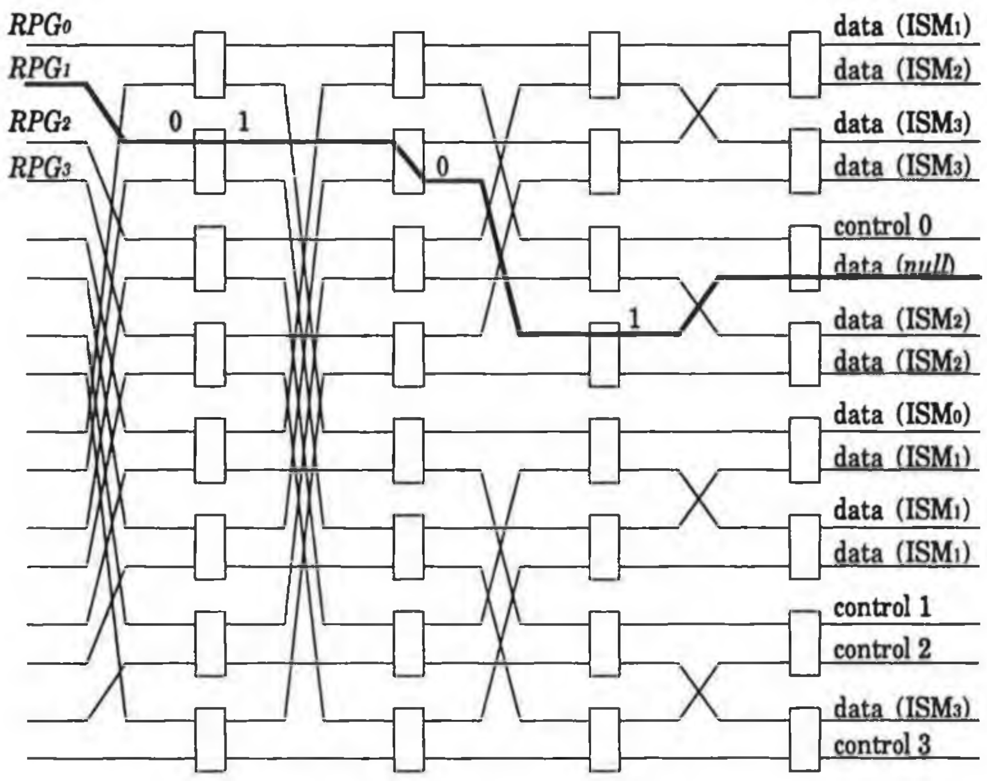
(b) Pass Two.



(c) Pass Three.



(d) Pass Four.



(e) Pass Five.

Fig. 5.14: An example of routing tag assignment.

The number of clock cycles required per iteration is at least one. The cycles in excess of one represent the processing of 'virtual' control packets. Hence, the number of clock cycles required is unchanged, at $n_1 + L_2$. The counter operation is more complex than was the case with the method of section 5.4.2, because of the processing of these 'virtual' control packets. A possible flowchart of counter operation is shown in Fig. 5.15. The counter can determine when execution should terminate either by counting $n_1 + L_2$ clock cycles, or, as shown in Fig. 5.15, by detecting a dummy idle cell address forwarded (after $n_1 + L_2$ clock cycles) from the input side of the most remote address generator. This is the purpose of the 'idle flag' shown.

The second method of request counting, involving an inverse S-banyan, requires the use of control packets, and thus cannot be employed at the output of the Batcher network in Fig. 5.6, if the Batcher network does not merge control packets with the incoming data cells. It is possible to generate synthetic control packets, which appear at the correct input ports of the inverse S-banyan, without merging them with the data cells in a sort network, using the method indicated in Fig. 5.16.

The synthetic control packets are generated using the copy network in Fig.

5.16. These packets are submitted to an inverse S-banyan identical to that used in Fig. 5.8, and the numbers of requests can then be determined by serial addition, as discussed in section 5.4.3. The sorter has only n_1 inputs and outputs, since it receives no control packets.

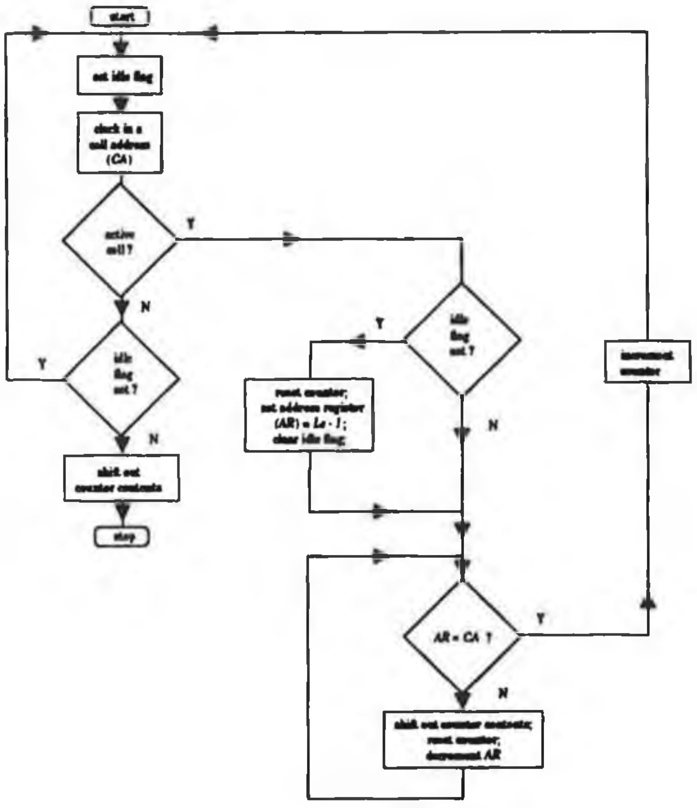

start
set idle flag
clock in a cell address (CA)
active cell ?
Y
N
idle flag set ?
Y
N
shift out counter contents
stop
idle flag set ?
Y
N
reset counter; set address register (AR) = Ls - 1; clear idle flag;
increment counter
AR = CA ?
Y
N
shift out counter contents; reset counter; decrement AR

Fig. 5.15: Counter operation when no control packets are present.

Request counting will be performed successfully if the inputs to the inverse S-banyan are identical to those in Fig. 5.18. The purpose of the copy network is to generate the appropriate inputs. Thus, for $0 \leq j < L_2$, a packet should appear at the inverse S-banyan inputs, with the following characteristics:

input port:	D_j ;
requested output port:	j ;
data field contents:	D_j ,

where

$$D_j = \sum_{u=0}^{j} K_{iu} + j .$$

and K_{iu} is the number of data cells requesting output module u.

This packet is generated at the output of the copy network. Hence the copy network must supply, for $0 \leq j < L_2$, a packet to input D_j of the inverse S-banyan, indicating the corresponding value of j. This packet is functionally equivalent to a control packet, and is therefore referred to as a synthetic control packet.

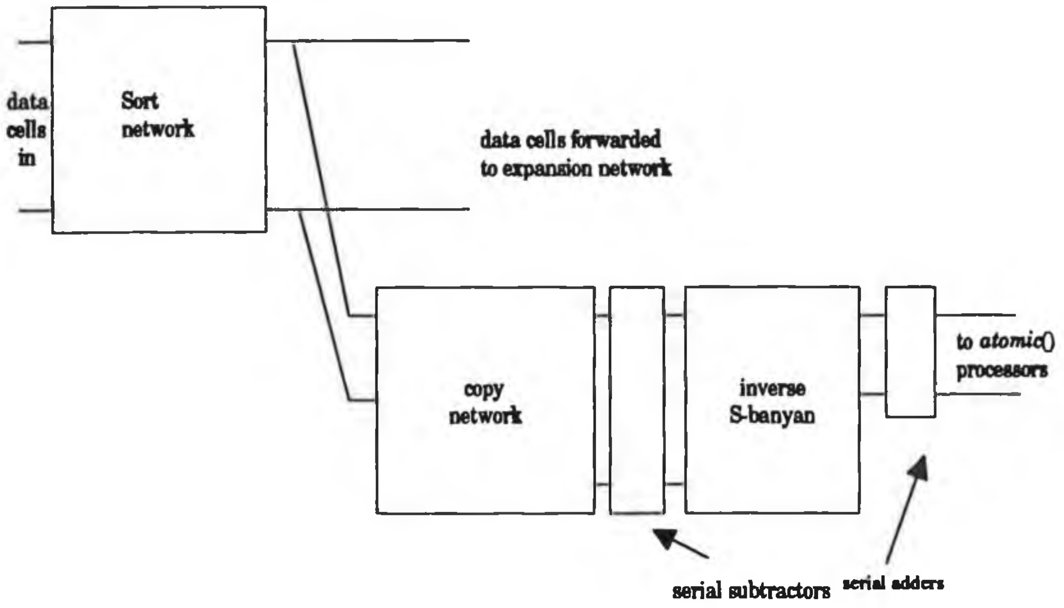


Fig. 5.16: Request counting using synthetic control packets.

Define F_j to be the lowest-numbered sorter output at which a cell appears with an address greater than j. Evidently, F_j is equal in value to the number of cells (from input module i) with an address (requested output module) of j or less, i.e.,

$$F_j = \sum_{u=0}^{j} K_{iu}.$$

Notice that

$$D_j = F_j + j.$$

Now suppose that a packet is submitted to input F_j of an S-banyan, requesting output D_j, and another is submitted to input F_l, requesting output D_l, with $F_j > F_l$. Both will be routed to the requested output of the S-banyan without contention, since

$$D_j - D_l = F_j - F_l + j - l \geq F_j - F_l,$$

and so the non-blocking condition,

$$O_2 - O_1 \geq I_2 - I_1,$$

is satisfied.

If the packet submitted to input port F_j of the S-banyan contains the value of j in the data field, then output D_j of the S-banyan receives the value of j, which can then be submitted to input D_j of the inverse S-banyan, as required.

To determine F_j, each data cell at the sorter output compares its address j to that of the next lower-numbered cell. If the addresses are unequal, that data cell must be at output F_{j-1} of the Batcher network. Note that this operation is also performed for idle cells.

A difficulty arises when $K_{ij} = 0$, since then $F_{j-1} = F_j$. Therefore, the packet launched into input $F_{j-1} = F_j$ of the S-banyan should be routed to outputs D_{j-1} *and* D_j. In other words, a copy network is required in place of the S-banyan, since a single input may be required to deliver data to multiple outputs.

Suppose one cell at the output of the sorter has address l, and the next higher cell is addressed to output module j, with $j > l+1$. Then $F_l = F_{l+1} = F_{l+2} = \ldots\ldots = F_{j-1}$. Hence, the packet submitted at input F_l of the copy network should request outputs D_l, D_{l+1}, ... and D_{j-1}. Since $F_l = F_{l+1}$ it follows that $D_{l+1} = D_l + 1$. Similar remarks hold for D_{l+2}, etc. Therefore the packet should request delivery to outputs D_l through $D_l + j - l - 1$ of the copy network, or, equivalently, outputs $F_l + l$ through $F_l + j - 1$.

Hence each input port of the copy network for which the corresponding cell has an address (j) other than that of the next lower-numbered input (l) submits a packet with the following characteristics.

Input port:	F_l	(where $K_{il} > 0$) ;
Output ports:	$F_l + l$ through $F_l + j - 1$	(where j > l, $\sum_{u=l+1}^{j-1} K_{iu} = 0$ and $K_{il} > 0$) ;
Data:	F_l .	

Notice that the data transmitted is not the data cell address, since a total of j-l output ports are to receive data. Output port

$$D_g = F_l + g, \quad l \leq g < j$$

should receive the value g. Thus each output port requires different data, which cannot be achieved with a copy network, which delivers the same data to each of the requested outputs.

Instead, the value of F_l is broadcast. This is easily obtained, since it is the address of the input port from which the broadcast is made. The value of g may easily be calculated at the output of the copy network, since

$$g = D_g - F_l ,$$

and the value of D_g is known locally.

It may easily be shown that no contention occurs during the above broadcast. Consider the broadcasts from two inputs, one from input $I_1 = F_l$, broadcasting to outputs $F_l + l$ to $F_l + j$-1 as indicated above, and the other from input $I_2 = F_s$ (where $s \geq j$), broadcasting to outputs F_s+s through $F_s + t$, with t>s.

Evidently

$$O_2^L = F_s + s$$

and

$$O_1^H = F_l + j\text{-}1.$$

Hence

$$\begin{aligned} O_2^L - O_1^H &= F_s - F_l + s - j + 1 \\ &= I_2 - I_1 + s - j + 1 \\ &> I_2 - I_1 \end{aligned}$$

since

$$s \geq j.$$

It follows that the non-blocking condition is satisfied.

The inverse S-banyan is also non-blocking. Consider the case where input I_1 (D_l) requests output O_1 (l) and input I_2 (D_j) requests output O_2 (j), where $j > l$. Then

$$I_2 - I_1 = D_j - D_l = \sum_{u=k+1}^{j} X_u + j - l \geq O_2 - O_1 ,$$

so that the non-blocking condition is satisfied. At the output side of the inverse S-banyan, K_{ij} can be calculated from D_j, since

$$D_j = D_{j\text{-}1} + K_{ij} + 1,$$

and so

$$K_{ij} = D_j - D_{j\text{-}1} \text{-}1,$$

as in Fig. 5.8.

The operation of this algorithm shall now be demonstrated by way of an example, details of which are shown in Tables 5.12 and 5.13.

The resulting paths taken through the copy network are shown in Fig. 5.17 (a), and the inverse S-banyan data are shown in Fig. 5.17 (b).

Note that, in Table 5.13, the first idle cell is considered to have an address of L_2. Also, the (non-existent) cell preceding the cell at output zero of the sorter is assumed to have an address of zero. These two assumptions ensure that the algorithm, as described, correctly calculates the values of D_0 and D_{L_2-1}. The first idle cell may be easily identified as such, simply by having it inspect the activity bit of the cell above it.

Output module	No. of requests
0	0
1	0
2	1
3	0
4	2
5	1
6	2
7	0
idle	2

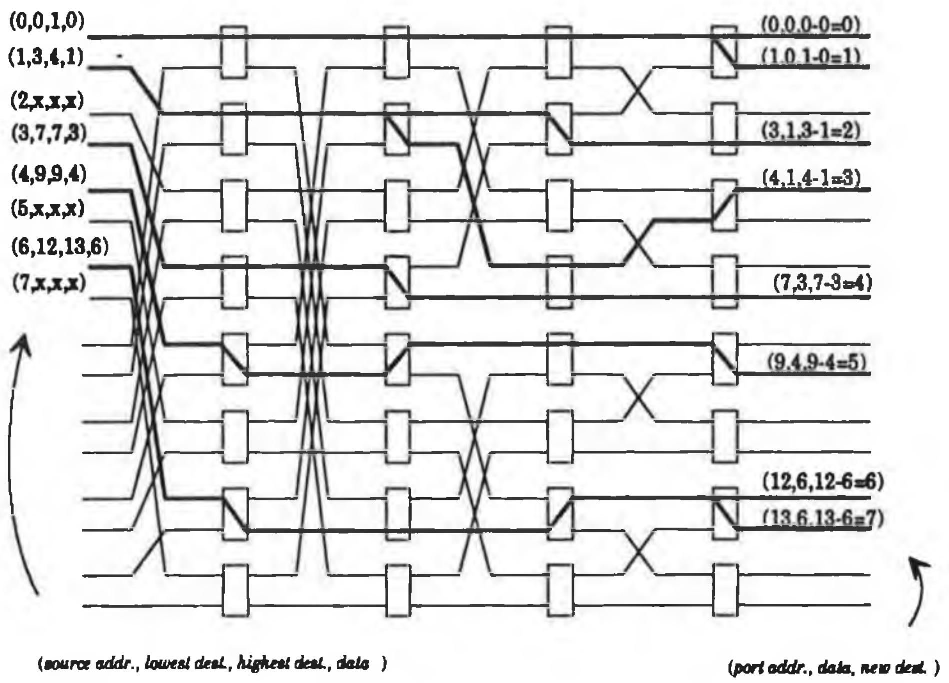
Table 5.12: A pattern of requests.

copy network input (F_l)	address of current cell (j)	cell address for previous copy network input (l)	Control packets to be synthesised (l to j - 1)	requested outputs	data
0	2	0	0 to 1	0 to 1	0
1	4	2	2 to 3	3 to 4	1
3	5	4	4 to 4	7 to 7	3
4	6	5	5 to 5	9 to 9	4
6	8 (i.e., L_2)	6	6 to 7	12 to 13	6

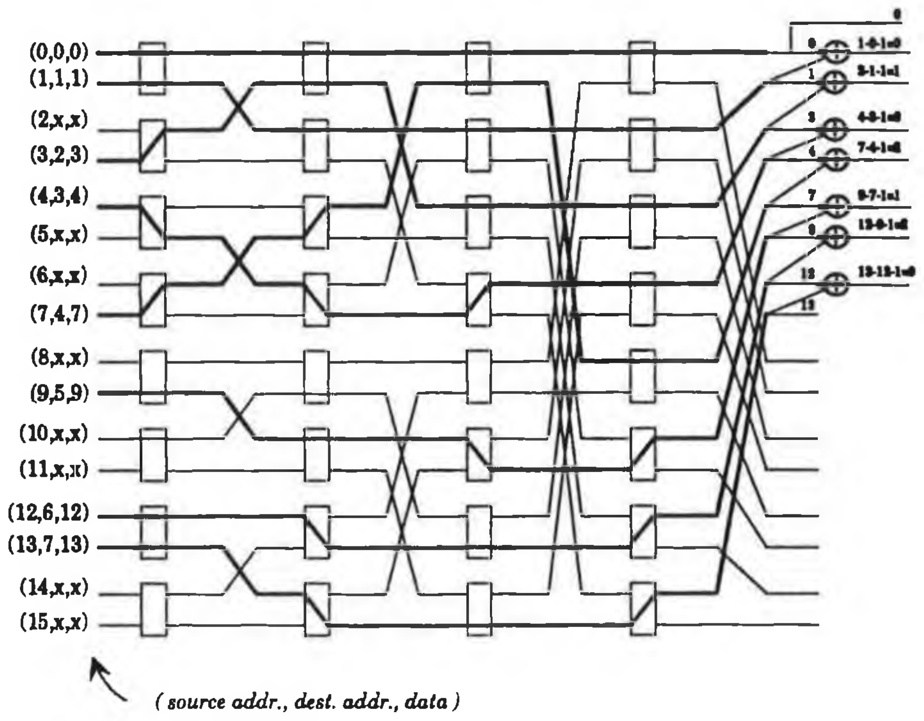
Table 5.13: Copy network data.

5.6.3 Elimination of control packets in routing tag assignment.

The copy network in Fig. 5.13 cannot be used to deliver routing tags to data cells at the output of a sorter, if cells for different addresses are not separated by control packets. The reason is that blocking can occur, if no requests are present for a number of destinations.



(a) Generation of synthetic control packets.



(b) Routing of synthetic control packets.

Fig. 5.17: Request counting with synthetic control packets.

This difficulty can be overcome by routing the cells through a network which introduces a gap between cells with consecutive addresses. This network routes cells to the outputs at which they would appear, if separated by control packets. Such a network will now be described.

Let K_j be the number of cells requesting output module j. If $K_j > 0$, the cells requesting this output appear at outputs F_{j-1} through $F_{j-1} + K_{ij} - 1$ of the sorter, where

$$F_j = \begin{cases} \sum_{u=0}^{j} X_{iu}, & j \geq 0 \\ 0, & j = -1 \end{cases}.$$

These cells should be routed to outputs $D_{j-1} + 1$ through $D_{j-1} + K_{ij}$, respectively, of the network, where

$$D_j = \begin{cases} \sum_{u=0}^{j} X_{iu} + j, & j \geq 0 \\ -1, & j = -1 \end{cases}.$$

Note that

$$D_j = F_j + j$$

and that

$$D_{j-1} + K_{ij} = F_j - 1 + j.$$

Hence the necessary routing will be achieved if a cell appearing at input I_1 of the network, and with an address of l, is routed to output $O_1 = I_1 + l$ of the network.

Consider a second cell, appearing at input I_2 of the network, with an address of j, where $I_2 > I_1$. Since the inputs to the network are sorted, it follows that $j \geq l$. This cell is routed to output O_2, where $O_2 = I_2 + j$.

Hence

$$O_2 - O_1 = I_2 - I_1 + j - l \geq I_2 - I_1.$$

It follows that the required routing may be accomplished without blocking by an S-banyan network. At most n_1 of its inputs will have active data, but $n_1 + L_2$ outputs are required.

The operation of this network is now considered for the example of Table 5.9. The appropriate pattern of inputs and output addresses is shown in Table 5.14.

This cell expansion is performed by the S-banyan network in Fig. 5.18. The routing tag assignment hardware is now as shown in Fig. 5.19. The operation of the copy network in Fig. 5.19 is identical to that in Fig. 5.13, and so copying occurs without contention.

Network input	Cell address	requested output
0	0	0
1	0	1
2	0	2
3	0	3
4	1	5
5	1	6
6	1	7
7	1	8
8	1	9
9	1	10
10	1	11
11	3	14
12	x	-

Table 5.14: Cell expansion pattern.

5.6.4 Hardware counts associated with synthetic control packets.

The savings in hardware obtained by the elimination of control packets (if any) shall now be assessed.

Suppose $p = \lceil \log_2(n_1 + L_2) \rceil$ and $q = \lceil \log_2(n_1) \rceil$. The network for merging control packets and data cells then requires $\frac{1}{2}p(p+1)$ stages, each containing 2^{p-1} sort elements, for a total of $\frac{1}{2}p(p-1)2^{p-1}$ sort elements. If synthetic control packets are used, this is replaced by a sort network for the data cells, containing $\frac{1}{2}q(q-1)2^{q-1}$ sort elements. However, a copy network, containing $p2^{p-1}$ copy elements, and an expansion network, containing $p2^{p-1}$ switch elements, are also required.

If it is assumed that each type of 2x2 element is of identical (unit) cost, the overall cost of both strategies is as shown in Table 5.15.

Cost of strategy	
(a) using a merge network	(b) using synthetic control packets
$\frac{1}{2}p(p-1)2^{p-1}$	$\frac{1}{2}q(q-1)2^{q-1}+p2^{p}$

Table 5.15: Cost comparison of a merge network and synthetic control packets.

Evidently, the latter strategy is to be preferred, provided $p > q$ (except for very low values of p and q, such as $p = 3$ and $q = 2$), i.e., there is a real hardware saving achieved through the elimination of control packets, if $n_1 + L_2 > 2^t > n_1$, for some integer t.

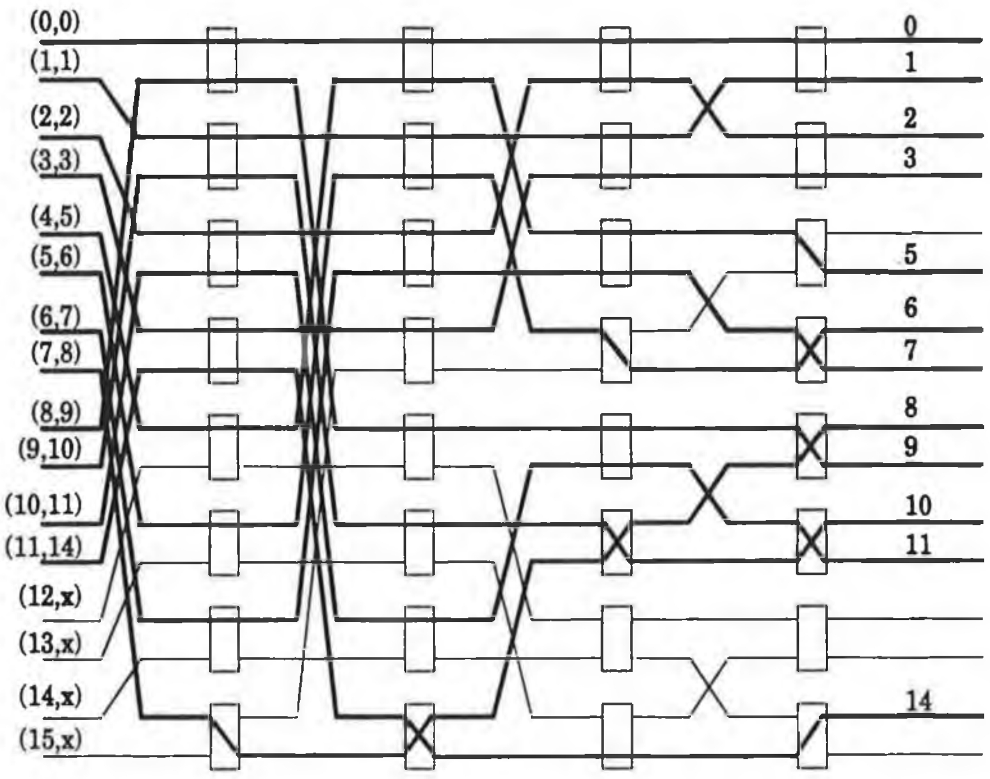


Fig. 5.18: An example of cell expansion.

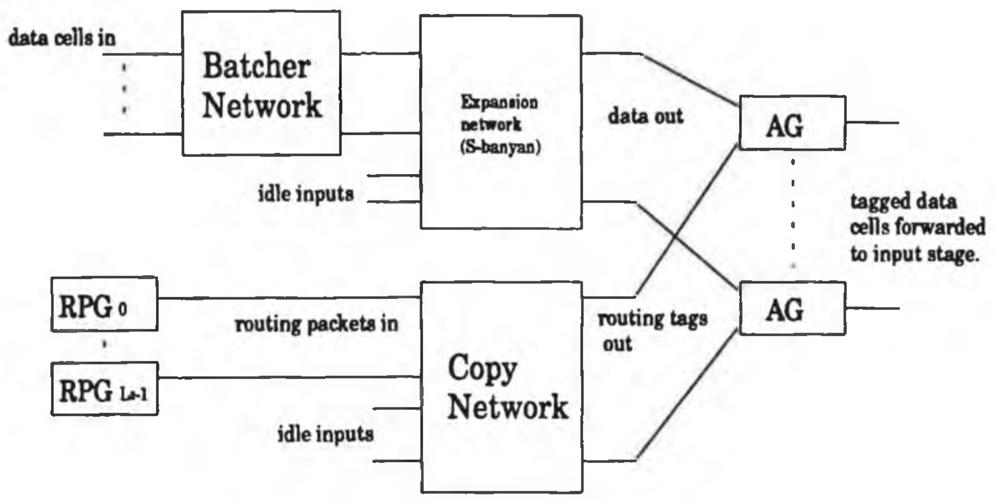


Fig. 5.19: Fast routing tag allocation without control packets.

5.7 Choice of switch modules.

5.7.1 Input stage modules.

The input modules are connected to the outputs of the path allocation circuitry as shown in Fig. 5.20. The cells are forwarded from the routing tag allocation outputs to the corresponding input module. The data cells are dispersed over $n_1 + L_2$ outputs of the routing tag allocation hardware. Hence the input module must have $n_1 + L_2$ inputs, at most n_1 of which bear active cells. The number of inputs required can be reduced to n_1 if the input module is preceded by a concentrator, which works as follows.

The concentrator is an inverse S-banyan with $n_1 + L_2$ inputs and n_1 (active) outputs.

A data cell entering the concentrator on input line I_1, with a destination of output module j, is routed to output $O_1 = I_1 - j$. Another cell, entering an input line $I_2 > I_1$, will request output $O_2 = I_2 - k$. Since the cells are sorted by requested output modules, it follows that $k \geq j$.

Hence

$$O_2 - O_1 = I_2 - I_1 - (k - j) \leq I_2 - I_1 .$$

The concentrator network is thus non-blocking. The routing it performs is the inverse of that performed by the expansion network in Fig. 5.18. Thus it produces a concentrated list of data cells at its lower n_1 outputs.

The routing tags of the cells must be adjusted at the concentrator output so that the address of the requested intermediate stage module appears first, since routing in the input module will be based on this address.

Contention is impossible in the input module, because of the nature of the path allocation. Hence contention resolution need not be performed, nor is any queuing required. A suitable switch module is a Batcher/banyan switch, where only the third phase of Hui's three-phase algorithm [17] is required. Any internally non-blocking switch could equally well be used.

5.7.2 Intermediate stage modules.

The intermediate stage is also free of contention. Once again any internally non-blocking switch module design may be used. There is no queuing, although a buffer sufficient to store the incoming cell will be required at the input side of the switch.

That part of the routing tag which indicates the selected intermediate stage module must be deleted, either at the output side of the input module, or at the input side of the intermediate module, so that routing in the intermediate stage is based on the requested output module.

5.7.3 Output stage modules.

Contention can occur in the output modules, and therefore they must be of the same complexity as a single-stage switch. An additional requirement must be imposed if preservation of cell sequence must be guaranteed.

The difficulty arises because cells belonging to the same virtual circuit may arrive from different intermediate stage modules, and therefore may arrive on different input ports of the output module. If the output module features input queuing, the earlier cell may reach the head of the queue after the later arrival, so that the cells are delivered out of sequence.

There are two solutions to this problem:

(i) Use of an output-buffered or shared-buffer switch. The variable delays in input queues can be avoided by using a switch which does not feature queuing at the input side of the switch. Suitable designs include the Sunshine switch [1] and Suzuki's memory switch [18].

(ii) If an input-queuing switch is used, a time-stamping mechanism can be used to ensure a uniform delay across the switch module. Such a mechanism was proposed by Henrion *et al.* [19] in the case of a three-stage switch. It is obviously easier to implement within a switch module than across a three-stage network. However, a large resequencing buffer may be required at each switch module output. The buffer management required may be complex. This approach also has the disadvantage that it increases the delay through the switch.

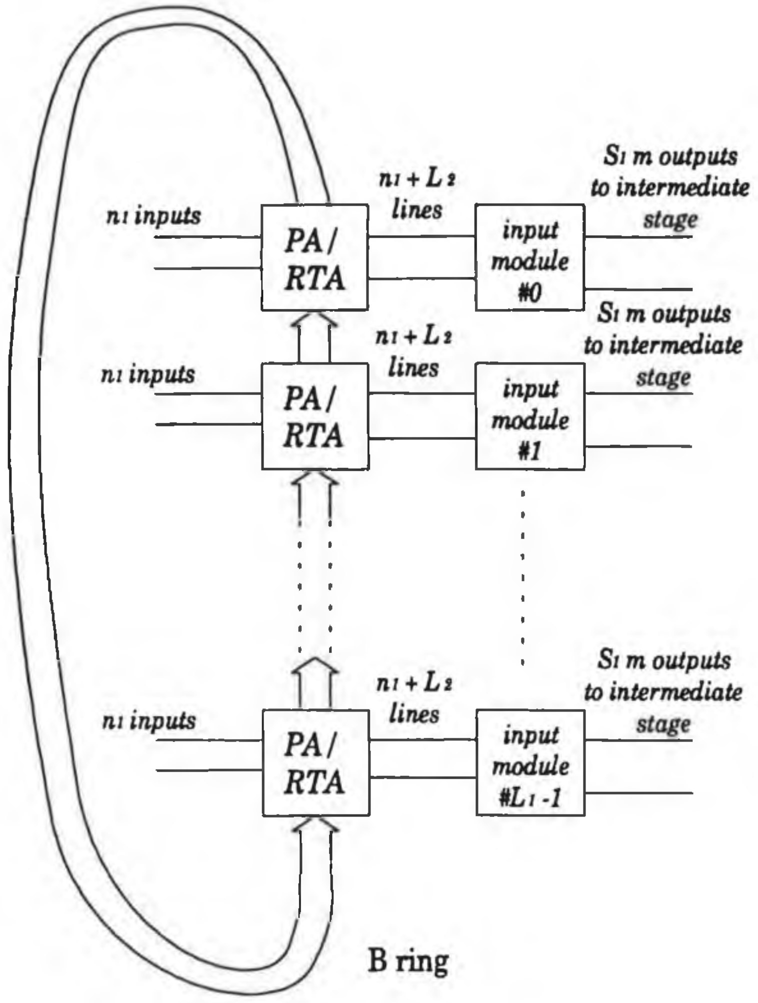


Fig. 5.20: Interconnection of input stage and path allocation circuitry.

Routing in the output module is based on the least significant bits of the destination address, the upper bits, indicating the requested output module, being discarded. Given technological constraints, there will be an upper limit on the size of single-stage switch which may be constructed. The three-stage network described here has the advantage that switch complexity is concentrated in the output modules. This allows a large switch to be constructed using many switch modules, the majority of which are of simple design.

5.8 Fault tolerance.

The path allocation algorithm may be easily modified to isolate a defective switch module. The procedure is to initialise the appropriate values of A_{ir} and B_{rj} to zero.

To isolate input module i, the value of A_{ir} is cleared to zero for $0 \leq r < m$. Hence no routes will be allocated out of that input module. The A registers which must be reset lie on a single row of the processor array.

To isolate intermediate module r, either the value of A_{ir} is reset for $0 \leq i < L_1$ or the value of B_{rj} is reset for $0 \leq j < L_2$. The value of A_{ir} is initially loaded into $X_{i,(r-i) \bmod m'}$ where $m' = max(L_1, L_2, m)$. The value of B_{rj} is initially loaded into $X_{(r-j) \bmod m', j}$. Hence isolating an intermediate module requires resetting the A or B registers along a diagonal of the processor array.

To isolate output module j, the value of B_{rj} is cleared to zero for $0 \leq r < m$. The relevant B registers lie on a column of the processor array.

This feature of the algorithm allows graceful degradation in the event of a switch module failure. In particular, if the faulty module is in the intermediate stage, the connectivity is unaffected, although switch performance will be impaired. A self-monitoring or self-test facility must be incorporated into the switch software, in order to detect module faults. This could feature the generation of test cells, the correct delivery of which would verify the switch operation. Once the fault is detected, module isolation can happen in the next time slot.

A failure of a single input module can be tolerated if adjacent input modules are designed so that, if one fails, the traffic which it should receive is diverted to its neighbour. Only simple hardware would be required to implement this. Cell loss probability would be high after the fault, but connectivity would be maintained.

A similar strategy might be considered at the output side of the switch. Corresponding outputs from adjacent modules could enter a 2x2 switch element. In the absence of a fault, cells entering on one input would leave on the corresponding output of the switch element. If one module is faulty, cells requesting that module are instead routed to its neighbour. An address bit is maintained at the head of the data cell, so that the 2x2 switch element can be used to route the cell to the requested destination.

A fault in the request count or routing tag allocation hardware can be handled in a similar manner to an input module fault. This is not the case in the event of a fault in the processor array.

Failure of the processor array can have a catastrophic effect on switch performance, since it may result, in the worst case, in an inability to allocate a path to any incoming cell. This is the major disadvantage of this method of path allocation. It is consequently advisable to replicate the processor array, to minimise the probability of catastrophic failure. This is unlikely to add significantly to the cost of the switch, given the low level of complexity of the processor implementation.

5.9 Implementation of loss priority.

The CCITT recommendations for ATM [20] specify that the cell header should contain a bit allowing two classes of traffic to be distinguished, which differ in the acceptable value of cell loss probability. Hence, the switch should be capable of favouring the retention of cells belonging to the low loss class when cells must be discarded. Cells belonging to a class requiring a low cell loss probability shall be referred to as high priority cells below.

High priority cells can be guaranteed precedence over low priority cells if two rounds of path allocation are performed. In the first round, the value of K_{ij} equals the number of high priority cells from input module i requesting output module j. Hence no more cells are allocated access to a path than the number of high priority cells available. Routing tag allocation is organised in such a way that it is these high priority cells which receive the relevant routing tags. The A and B register values are not re-initialised at the start of the second round. Hence paths allocated to high priority cells in the first round remain unavailable during the second round. The value of K_{ij} during the second round is equal to the number of outstanding requests (from both high and low priority cells) for output module j.

This approach guarantees that low priority cells are not allocated paths at the expense of high priority cells. However, request counting is more complicated than heretofore, since two types of request must be distinguished. More seriously, the number of iterations required of the path allocation algorithm is doubled, and so the operating speed required to ensure a given execution time is doubled. This limitation will prohibit the use of this approach in large switches.

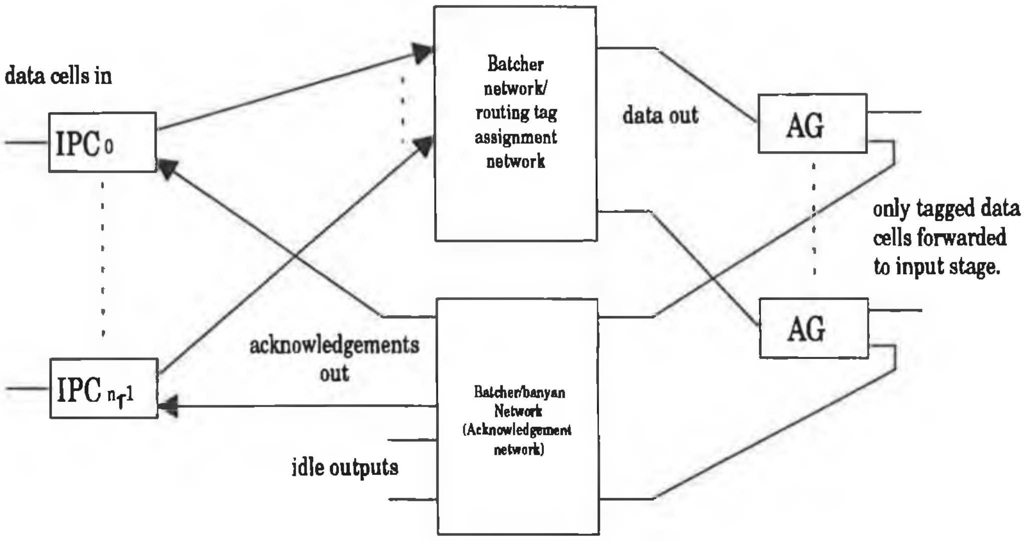
A simpler alternative distinguishes between high and low priority cells only within an input module. Null routing tokens, when issued, are given to the low priority cells first.

This can be achieved quite simply, by sorting the cells requesting a given output module on the basis of priority, such that the higher priority cells are farthest from the corresponding control packet. A single additional bit in the header used by the Batcher sorter in Fig. 5.6 ensures this. Since all tokens are distributed to the cells closest to the control packet, high priority cells are assured of precedence in access to paths.

It is possible, using this approach, that a low priority cell may be given precedence over a high priority cell from another input module. However, if the probability of this occurring is low, the cell loss probability for high priority cells may be expected to be much lower than for low priority cells. The differential in cell loss probabilities will depend on the switch dimensions, the offered load, and the proportion of cells in each class.

5.10 Implementation of input queuing.

If input queues are used, acknowledgements must be returned to the input port controllers indicating success or failure in winning contention. The hardware required resembles that used in the second phase of Hui's three-phase algorithm [17].


data cells in
IPC 0
IPC n-1
Batcher network/ routing tag assignment network
data out
AG
AG
only tagged data cells forwarded to input stage.
acknowledgements out
Batcher/banyan Network (Acknowledgement network)
idle outputs

5.21: Returning acknowledgements.

Essentially a Batcher-banyan switch is used to route a packet to the input port controller. Each data cell must have appended to its routing tag, a field indicating the source address. This is used as a routing tag by the acknowledgement network, or flag network. The data packet submitted to the flag network by the address generator in fig. 5.21 contains only a single bit (the flag bit). Upon receipt of the flag bit, the input port controller can either purge the cell, replacing it at the head-of-a-queue position with a new one, if the submitted copy of the cell has been allocated a routing tag, or else retain the cell until the time slot, when it will once more attempt to win contention.

The flag network has a complexity comparable to that of an input module, or intermediate module. Hence the cost of implementing input queuing is comparable to that of increasing *m*, the number of intermediate modules. Switch performance, as measured by the cell loss probability, may be improved for a switch which discards cells losing contention, by either

(i) using input queuing, so that fewer cells are discarded.

(ii) increasing *m*, so that fewer cells lose contention.

Which of these strategies provides the lower cost solution to improving cell loss probability requires further investigation.

5.11 A design example.

The trade-offs involved in the design of a three-stage switch using the path allocation algorithm described in this chapter shall be demonstrated by reference to a specific example. The case considered is of a 3072 x 8192 switch constructed by choosing $L_1 = L_2 = m = 32$, $n_1 = 96$, $n_2 = 256$, $S_1 = 4$ and $S_2 = 8$ in Fig. 5.1. The input module dimensions are 128 x 128. At most 96 of the 128 inputs carry active data. The dimensions of the intermediate and output stage modules are 128 x 256, and 256 x 256 respectively. The input and intermediate switch modules can be of simple design, since they are contention-free. Thus the only stage of the switch which represents a major design challenge is the output stage, where the 256 x 256 switch modules, where contention can occur, should introduce a low cell loss probability.

The number of system clock cycles required for the various hardware components to execute shall now be determined.

The number of clock cycles required to implement request counting using the method of Fig. 5.6 is $n_1 + L_2 = 128$ clock cycles. Employing the method of Fig. 5.8 reduces this to $2\lceil\log_2(n_1 + L_2)\rceil + 1 = 15$ clock cycles, as discussed in section 5.4.3.

Each iteration of the path allocation algorithm requires 11 clock cycles, according to Table 5.1, when using the circuit of section 5.3.2, or 9 cycles (as shown in Table 5.6) using the circuit of section 5.3.4. This figure could be reduced in a bit-parallel implementation. A total of $m' = \max(L_1, L_2, m) = 32$ iterations is required.

Consecutive iterations of the routing tag assignment algorithm depicted in Fig. 5.9 occur at intervals of at least $\log_2\lceil n_1 + 1\rceil + \log_2\lceil m + 1\rceil$ clock cycles. This is the number of bits required to represent the contents of a routing packet, which are the token count (a value in the range zero through n_1) and the token address (a value in the range zero through m-1, or an additional value representing a null token). Hence, for the example considered here, the interval between iterations must be 13 clock cycles.

Routing tag assignment and path allocation must proceed concurrently. Hence, the number of clock cycles between successive iterations of either algorithm must be the same. Thus, with the figures given above, the interval between iterations must be max (11,13) = 13 clock cycles, when the slower implementation of the *atomic*() processor is used. No improvement results from the use of the faster processor design, unless the length of routing packets can be reduced from 13 bits to a value of 9 bits or less. The total number of clock cycles required is 13 x m' = 13 x 32 = 416 cycles.

This value can be reduced, at the expense of an increase in the complexity of the address generator shown in Fig. 5.11, if the routing packets contain only the token count, and not the token address. This reduces the minimum number of clock cycles between successive iterations of the routing tag assignment algorithm to just $\log_2 \lceil n_1 + 1 \rceil = 7$ clock cycles. Hence, when using the faster *atomic*() processor implementation, the interval between iterations is reduced to max (9,7) = 9 clock cycles, and the total number of clock cycles required by the path allocation process is 9 x m' = 288 cycles.

Since the token address is then not transmitted by the routing packet generator, it must be generated locally by the address generator. Initially, in accordance with the algorithm described in section 5.2.4, the token address is

$$TA_0 = (i + j) \bmod m',$$

where i is the input module number and j is the requested output module. The former quantity is fixed for all address generators associated with a given input module, while the latter quantity is available to the address generator from the associated control packet, received during the process of request counting. Hence, the initial token address may be easily determined by the address generator.

After each subsequent iteration of the path allocation and routing tag assignment algorithms, the token address is modified in accordance with the equation

$$TA_k = (TA_{k-1} - 1) \bmod m'.$$

Thus the token address may be easily updated using a decrementing circuit. It follows that an appropriate and straightforward augmentation of the address generator circuit allows the transmission of token addresses through the Batcher network to be dispensed with, resulting in the indicated reduction in execution time for the path allocation process.

The excess delay due to propagation of null tokens down the address generator chain after path allocation has concluded shall now be found. It may be shown that the worst case occurs when all input ports of the switch request the same output module. No cells are then allocated routes after the first iteration of the path allocation algorithm, and so a total of $n_1 - \min(S_1, S_2)$ cells must receive null tokens. Propagation of these tokens occurs at the rate of one address generator per clock cycle when using hardware such as that in Fig. 5.11. Hence, for the example considered, an additional 92 clock cycles are required for token propagation, after path allocation has been completed.

When the hardware of Fig. 5.17 is used to perform routing tag assignment, the transmitted routing tags have a length of $\log_2\lceil n_1 + L_2 \rceil + \log_2\lceil m+1 \rceil + 1$ bits (where both the upper copy address and token address are transmitted, along with a leading activity bit) or of $\log_2\lceil n_1 + L_2 \rceil + 1$ bits (if the token address is generated locally). This corresponds to a tag length of 14 or 8 bits for the example considered here. It follows that, at best, there must be 8 clock cycles between successive iterations of the path allocation algorithm. Hence the number of clock cycles per iteration is unchanged, at max (8,9) =9, and the total number of clock cycles required to perform path allocation is unchanged, at 288 cycles.

The hardware of Fig. 5.17 thus appears to offer no advantage over the simpler circuitry of Fig. 5.9. In fact, the first iteration of the algorithm requires the lower copy address to be transmitted as well, adding a further $\log_2\lceil n_1 + L_2 \rceil = 7$ clock cycles to the execution time for the first iteration. However, no additional overhead is imposed by this requirement, since the first iteration can proceed concurrently with the request counting process, and will conclude simultaneously with the latter process (if an inverse S-banyan is used for request counting) or long before it (if the sequential algorithm is used). The advantage of this method lies in the reduced number of clock cycles required to forward the null tokens to the appropriate address generators after path allocation has been performed. This requires just $2\log_2\lceil n_1 + L_2 \rceil + 1 = 15$ clock cycles, which is the propagation delay through the copy network, plus one cycle to strip off the leading activity bit.

Note that if an input-queuing strategy is used, an additional 42 cycles is added to the execution time, this being the propagation delay through the 128-input Batcher-banyan network used for forwarding acknowledgements.

The clock rate required for a switch of the indicated dimensions shall now be determined, based on the above calculations, for the following two scenarios, with and without input queuing:

(i) request counting is performed as in Fig. 5.6;

path allocation is performed using the faster bit-serial processor;

routing tag allocation is performed using the method of Fig. 5.9, with token addresses generated locally;

(ii) request counting is performed as in Fig. 5.8;

path allocation is performed using the faster bit-serial processor;

routing tag allocation is performed using the method of Fig. 5.19, with token addresses generated locally.

clock cycle	events
0	input cells arrive;
28	The A bit (Fig. 5.7) appears at the Batcher output;
33	The $D/\overline{C}$ bit appears at the Batcher output; the I register is set;
161	request counting concludes;
449	path allocation concludes;
541	all tokens received;
583	acknowledgements received (optional)

(a) Execution schedule for first scenario.

clock cycle	events
0	input cells arrive;
28	The A bit appears at the Batcher output;
33	The $C/\overline{D}$ bit appears at the Batcher output; the I register is set;
48	request counting concludes; copy network initialised
336	path allocation concludes;
351	all tokens received;
393	acknowledgements received (optional)

(b) Execution schedule for second scenario.

Table 5.16: Execution schedules for two switch designs.

The appropriate execution schedules are shown in Tables 5.16 (a) and 5.16 (b) respectively. They indicate that the number of clock cycles required to process all the cells arriving in one time slot is 541 (583) for scenario (i) and 351 (393) for scenario (ii) without (with) input queuing. The execution time must be less than one time slot, which has a duration of 2.726 μs (based on a 53-octet cell, at 155.52 Mb/s). Hence the required clock rates (to the nearest MHz) are as shown in Table 5.17.

	minimum clock rate (MHz)	
	without queuing	with queuing
implementation (i)	198	214
implementation (ii)	129	144

Table 5.17: Minimum clock rates.

Note that the increased complexity of the second implementation allows a clock rate reduction of about 36 %, and that the implementation of input queuing adds approx. 15 MHz to the required clock rate. The clock rate can be further reduced, if input queuing is not used, to 99 MHz and 64 MHz respectively, if two copies of the hardware process requests in alternate time slots.

The required clock rates should prove achievable in CMOS technology in the short term, given the present state of the art, as discussed in section 5.3.5. Hence the feasibility of the approach described here to the design of a large ATM switch has been demonstrated. It remains to be discovered how to dimension the switch so as to achieve a low cell loss probability. This is the subject of the next chapter.

References

[1] J.N. Giacopelli, W.D. Sincoskie and M. Littlewood, "Sunshine: a high performance self-routing broadband packet switch architecture", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. III, pp. 123-129.

[2] T.T. Lee, "A modular architecture for very large packet switches", *IEEE Trans. Commun.*, vol. COM-38, no. 7, pp. 1097-1106, July 1990.

[3] A. Cisneros, "Large packet switch and contention resolution device", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. III, pp. 77-83.

[4] K.Y. Eng, M.J. Karol and Y.S. Yeh, "A growable packet (ATM) switch architecture: design principles and applications", *Globecom '89 Conference Record*, pp. 1159-1165.

[5] K.Y. Eng and C.-L. I, "Performance analysis of a growable architecture for broadband packet (ATM) switching", *Globecom '89 Conference Record*, pp. 1173-1180.

[6] R. Proctor and T. Maddern, "Synchronous ATM switching fabrics", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. IV, pp. 109-114.

[7] M. Beshai and E. Munter, "A rotating-access ATM switch", *Queueing Performance and Control in ATM, Proc. ITC-13*, pp. 53-58, 1991.

[8] M. Collier and T. Curran, "Path allocation in a three-stage broadband switch with intermediate channel grouping", *Proc. Infocom '93*, pp. 927-934.

[9] M. Collier and T. Curran, "Cell-level path allocation in a three-stage ATM switch", submitted for publication.

[10] J. Hickey and W. Marcus, "The implementation of a high speed ATM packet switch using CMOS VLSI", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. 1, pp. 75-84.

[11] R. Hofmann and R. Muller, "A multifunctional high-speed switch element for ATM applications", *IEEE J. Solid-State Circuits*, vol. SC-27, no. 7, July 1992, pp. 1036-1040.

[12] H. Weston *et al.*, "A submicrometer NMOS multiplexer-demultiplexer chip set for 622.08-Mb/s SONET applications", *IEEE J. Solid-State Circuits*, vol. SC-27, no. 7, July 1992, pp. 1041-1049.

[13] K. Yano *et al.*, "3.3-V BiCMOS circuit techniques for 250-MHz RISC arithmetic modules", *IEEE J. Solid-State Circuits*, vol. SC-27, no. 3, March 1992, pp. 373-381.

[14] M. Ishibe *et al.*, "High-speed CMOS I/O buffer circuits", *IEEE J. Solid-State Circuits*, vol. SC-27, no. 4, April 1992, pp. 671-673.

[15] C. Day, J. Giacopelli and J. Hickey, "Applications of self-routing switches to LATA fiber optic networks", *Proc. ISS '87*, pp. 519-523.

[16] T.T. Lee, "Nonblocking copy networks for multicast packet switching", *Journal Of Select. Areas Commun.*, vol. 6, no. 9, pp. 1455-1467, Dec. 1988.

[17] J. Hui, "A broadband packet switch for multi-rate services",*Proc. ICC '87*, pp. 782-788.

[18] H. Suzuki *et al.*, "Output-buffer switch architecture for Asynchronous Transfer Mode", *Proc. ICC '89*, pp. 99-103.

[19] M. Henrion *et al.*, "Switching network architecture for ATM based broadband communications", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. V, pp. 1-8.

[20] CCITT recommendation I.361, "BISDN ATM layer specification", 1990.

6. PERFORMANCE OF THE THREE-STAGE ATM SWITCH.

6.1 Introduction.

The three-stage switch proposed in Chapter Five must be designed in such a way as to ensure an adequate quality of service for all users. Attention is restricted here to the case where the input and intermediate stages operate as a loss system rather than as a delay system. In other words, it will be assumed that calls which are not allocated a path through the second stage of the network on the first attempt are discarded, and that there is no queuing at the input side of the switch. The complexity of the path allocation hardware, and the required speed of operation, is reduced, for a given switch, if there is no input queuing, since no acknowledgements need to be returned to the input ports.

Cell loss can occur due to one of two mechanisms:

- loss in the output module;
- loss during path allocation.

The former loss can be evaluated using one of the methods described in Chapter Three. The loss during path allocation must now be determined.

6.2 Numerical methods of loss calculation.

The calculation of blocking or loss probabilities in three-stage circuit switches is a topic which has received considerable attention from teletraffic theorists. Exact solutions are typically not available to problems of this type, and so assumptions must be made concerning the balancing of traffic flows, and the independence of links. Methods which can be applied to the problem of loss calculations in three-stage circuit-switched networks include those of Lee, Pippenger and Jacobaeus [1-3]. The PPL method [4] is unusual in that it is a heuristic method, based on the calculation of the so-called 'effective accessibility', using a formula chosen empirically to agree with simulation results.

All of the above methods implicitly assume that a random search strategy is used to find free paths through the switch. The performance of other search strategies must be evaluated using simulations. Hebuterne [5] discussed the relative performance of random search and sequential search strategies. His conclusion, based on a survey of published simulation results, was that sequential search strategies offered performance superior to a random search. A similar results was obtained by Jajszcyk *et al.* [6] in the case of a three-stage switch with channel grouping. They also extended the PPL method to cater for the multi-channel case. The Lee and Pippenger methods were extended to the case of multi-rate circuit switching by Voldimarsson [7].

These results, although developed to determine the behaviour of circuit-switched networks, have relevance to ATM networks. The concept of equivalent bandwidth, as described in Chapter Four, has been used to apply the results developed for multi-rate circuit switching to ATM networks. The values for loss probabilities obtained by this method apply at the call level; they indicate the probability that a call request will be blocked by the call admission control process. They do not predict values for cell loss probabilities, except in so far as the operation of the call admission control process ensures that an upper bound on the cell loss probability is not exceeded. Fitzpatrick *et al.* [8] used this approach to apply their method for calculating loss probabilities in multi-rate three-stage circuit switches to the determination of blocking probabilities in ATM networks.

The numerical methods listed above cannot be readily adapted to determine the probability of cell loss in an ATM switch employing cell level path allocation. Requests for bandwidth occur simultaneously at the cell level, so that the results of the path allocation process depend on the order in which these requests are processed.

Karol and Chih-Lin I [9] extended the method of Jacobaeus [4] to the determination of cell loss probability for a three-stage ATM switch. They assumed that the cell scheduling algorithm of the Growable Packet Switch [10] was used. They showed that the cell loss probability was upper bounded by

$$P_{loss} \leq \sum_{s=m+n-1}^{m-1} \binom{N-1}{s} \left(\frac{np}{N-1}\right)^s \left(1-\frac{np}{N-1}\right)^{N-1-s} \frac{(n-1)!s!}{(n-1+s-m)!m!} + \sum_{s=m}^{N-1} \binom{N-1}{s} \left(\frac{np}{N-1}\right)^s \left(1-\frac{np}{N-1}\right)^{N-1-s}$$

where each input port receives a cell with probability p in each time slot, for an N-input symmetric switch, with n inputs per input module. The extension of this method to a channel-grouped architecture was also considered.

This inequality cannot be applied to evaluate the performance of the path allocation algorithm described in Chapter Five, because of the differences in the sequence of paths searched, compared with the cell scheduling algorithm.

During each iteration of the cell scheduling algorithm, each input module attempts to allocate paths to one output module, via each intermediate stage module. The algorithm is thus intrinsically unfair, unless the sequence in which output modules are processed is varied after every time slot. In contrast, during an iteration of the path allocation algorithm, each input module attempts to find a path to every output module, via a unique intermediate stage module. This feature of the path allocation algorithm makes it fairer than the cell scheduling algorithm, but results in the method of [10] being inapplicable to this case. Accordingly, simulation techniques must be used to determine the cell loss probability for a switch using the latter algorithm [11].

6.3 Performance of the path allocation algorithm.

6.3.1 Performance with uniformly loaded output modules.

The simulation model used to evaluate performance is based on the following assumptions:

(i) There is no input queuing; cells which are not allocated a path on the first attempt are discarded.

(ii) The switch is offered a maximum load; each input port of the switch submits a cell in every time slot.

(iii) The destination of each cell is drawn from a uniform distribution; all output modules receive the same load.

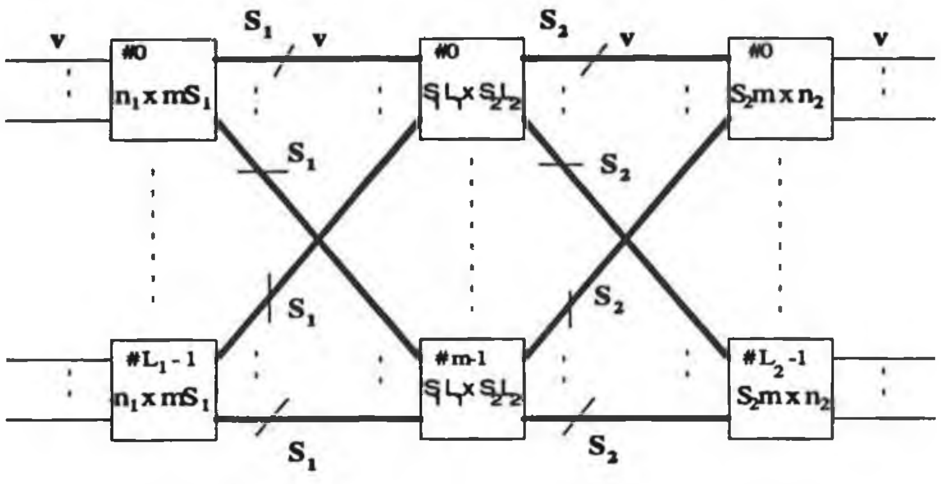
(iv) The switch modelled is that shown in Fig. 6.1, for various choices of the parameters L_1, L_2, m, S_1, S_2 and n_1.

(v) The maximum number of cells generated is 10^9. If zero cell loss is recorded during the simulation, the cell loss probability is assigned the value 5×10^{-11}. The probability of losing contention is assumed to be independent for each cell, at low levels of loss. With this assumption, the probability that the cell loss probability (*CLP*) is below 5×10^{-11}, given that no losses were recorded, is above 95%, i.e.,

$$\Pr\left[CLP < 5\times10^{-11}\right] = \left(1-5\times10^{-11}\right)^{10^9} > 0.95.$$

The QOS measure of most interest in ATM networks is the cell loss probability on a virtual circuit. This is difficult to obtain by simulation without recourse to extremely long simulation run lengths, and the results obtained will depend on the models of source traffic used, and on the traffic mix. In contrast, the cell loss probability for the entire switch may be calculated simply as the expected number of cells lost per time slot, divided by the offered load. The latter quantity, which is related to the grade of service, has been found by simulation below for switches of various sizes.

The influence of the choice of channel group size in Fig. 6.1 on the cell loss probability is considered in Figs. 6.2 through 6.5. The performance in the case where $S_1 = S_2 = 4$ is shown in Fig. 6.2. Note that $L_1 = L_2 = m$ for the three switches simulated. The curves obtained show that, with the number of switch inputs fixed, the cell loss probability falls as the number of switch modules is increased, as expected. A more interesting result is observed if the results are plotted against the number of inputs per input module, as shown in Fig. 6.3. These loss curves indicate that, if modular growth is achieved by increasing L_1, L_2 and m in the same proportion, the cell loss probability will decrease. Hence, the additional loss due to the higher switch load is offset by the increasing diversity of paths available.



V:	channel rate (155 Mb/s)
n_1 (n_2) :	the number of input (output) ports per input (output) module;
L_1 (L_2) :	the number of input (output) modules;
m:	the number of intermediate switch modules;
S_1 (S_2) :	the number of channels in the channel group connecting each input (output) module to each intermediate switch module.

Fig. 6.1: A three-stage switch with intermediate channel grouping.

The results for three switches similar to those considered above, but where S_1 has been increased to 8, are shown in Fig. 6.4. It can be seen from these results that increasing the channel group size at the input side of the intermediate stage gives rise to only a marginal decrease in the cell loss probability. This is not the case where the channel group size is increased at the output side of the intermediate stage, as can be seen from the simulation results for the corresponding switches with $S_1 = 4$ and $S_2 = 8$, shown in Fig. 6.5.

These simulations indicate that the intermediate modules should be designed as expansion modules, with more outputs than inputs (i.e., with $S_2 > S_1$), to obtain the best performance. This seems intuitively reasonable; a cell may be routed to any intermediate module, but can be routed to only one output module. The value of m required to ensure that a given value of cell loss probability can be achieved will be much higher if channel grouping is not used (i.e., if $S_1 = S_2 = 1$). This has the disadvantage that the number of iterations required by the path allocation algorithm will increase, so that higher speed hardware may be required. The effect of varying m in the range 30 to 34, for a switch with $L_1 = L_2 = 32$, and with channel groups given by $S_1 = 4$ and $S_2 = 8$, is shown in Fig. 6.6.

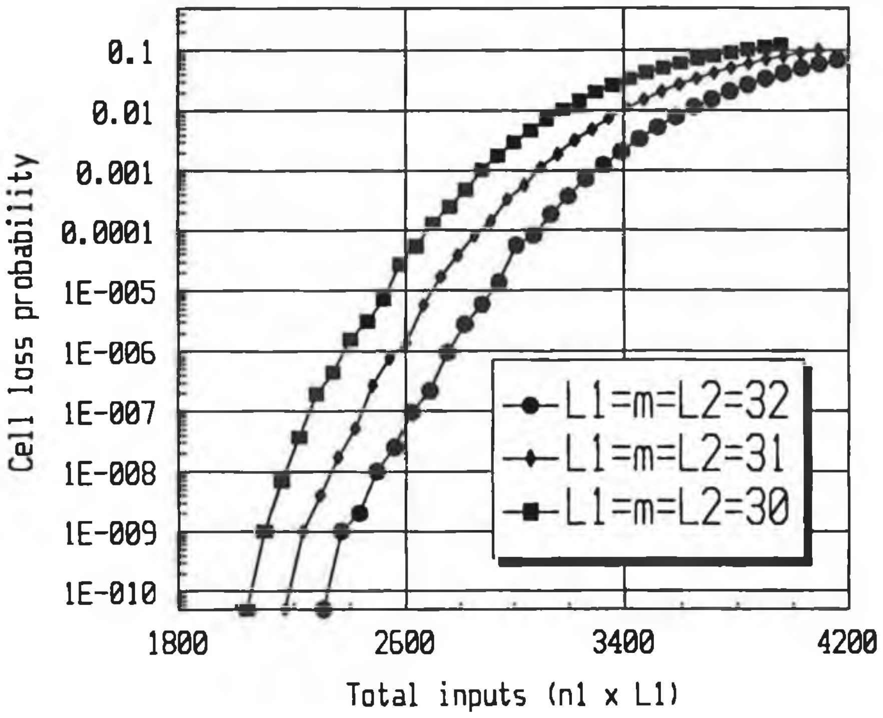


Fig. 6.2: Performance with uniform traffic ($L_1 = m = L_2$, $S_1 = S_2 = 4$).

The efficiency of the path allocation algorithm may be demonstrated by comparing its performance with the lower bound on achievable performance. This bound corresponds to the knockout loss in [10]; at most S_2m cells can be delivered to each output module in any one time slot. This bound, for the case where $L_1 = L_2 = m = 32$ and $S_1 = S_2 = 4$, is compared to the simulation results in Fig. 6.7. It can be seen that the performance of the algorithm is close to the lower bound. Also shown for comparison is the upper bound calculated from the relevant non-blocking condition in Chapter Four.

6.3.2 Performance with non-uniformly loaded output modules.

The above results all apply to a situation where there is a uniform load across all the output modules. This is unlikely to be the case in practice, and so the performance must be assessed in the presence of a non-uniform loading of outputs. Accordingly, assumption (iii) of the simulation model must be modified.

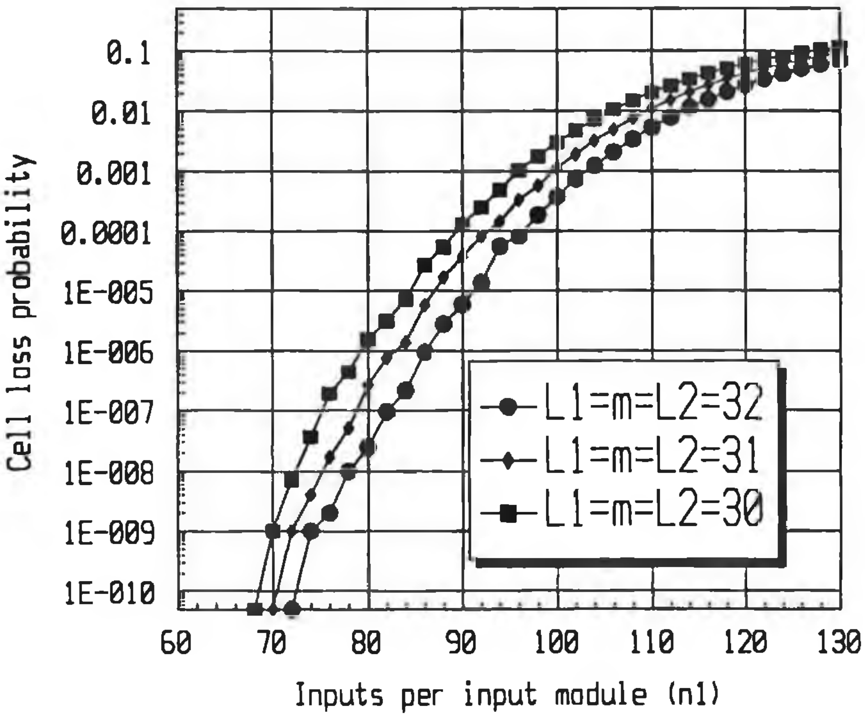


Fig. 6.3: Variation of cell loss with n_1 ($L_1 = m = L_2$, $S_1 = S_2 = 4$).

Fig. 6.8 shows the results obtained for a switch with $L_1 = L_2 = m = 32$, for various channel group sizes, in the case where 75% of arriving cells request one from sixteen (contiguous) output modules, each output module of the sixteen being selected with equal probability. The remaining 25% of cells select (with equal probability) one of the other sixteen output modules. Hence 75% of the load is offered to only 50% of the available outputs.

It can be seen that the cell loss probability increases dramatically, compared with the case of uniform loading (i.e., Fig. 6.2 and Fig. 6.4), when $S_1 = S_2 = 4$, or when $S_1 = 8$ and $S_2 = 4$. Furthermore, the additional bandwidth available from the input stage of the switch to the intermediate stage in the latter case results in a negligible decrease in the cell loss probability. In contrast, the cell loss probability increases by a relatively small amount (compared with Fig. 6.5) for the case where $S_1 = 4$ and $S_2 = 8$. This further underlines the benefit of having a large bandwidth at the output side of the intermediate stage.

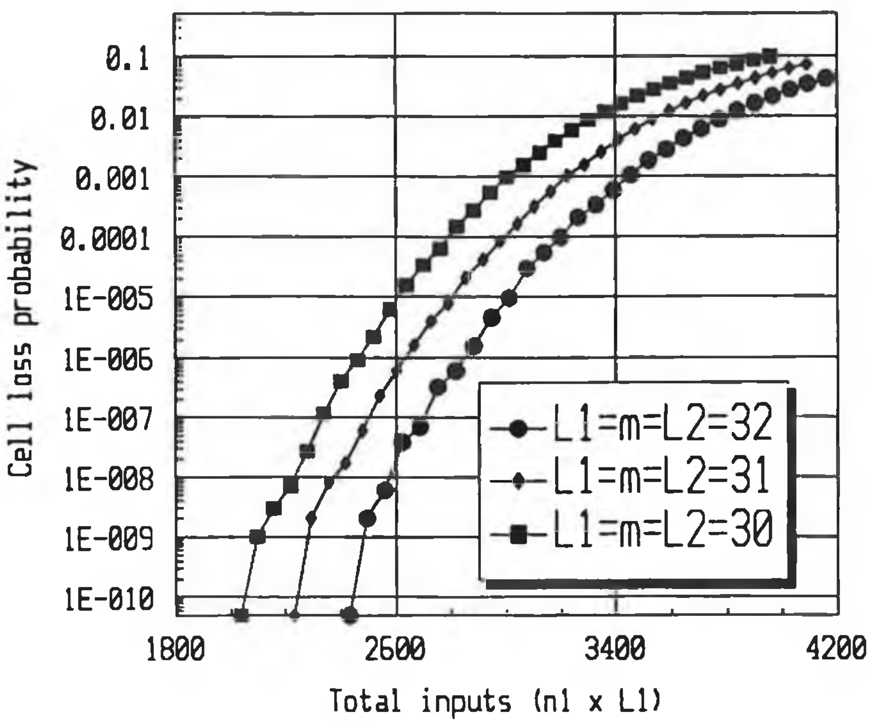


Fig. 6.4: Performance with uniform traffic ($L_1 = m = L_2$, $S_1 = 8$, $S_2 = 4$).

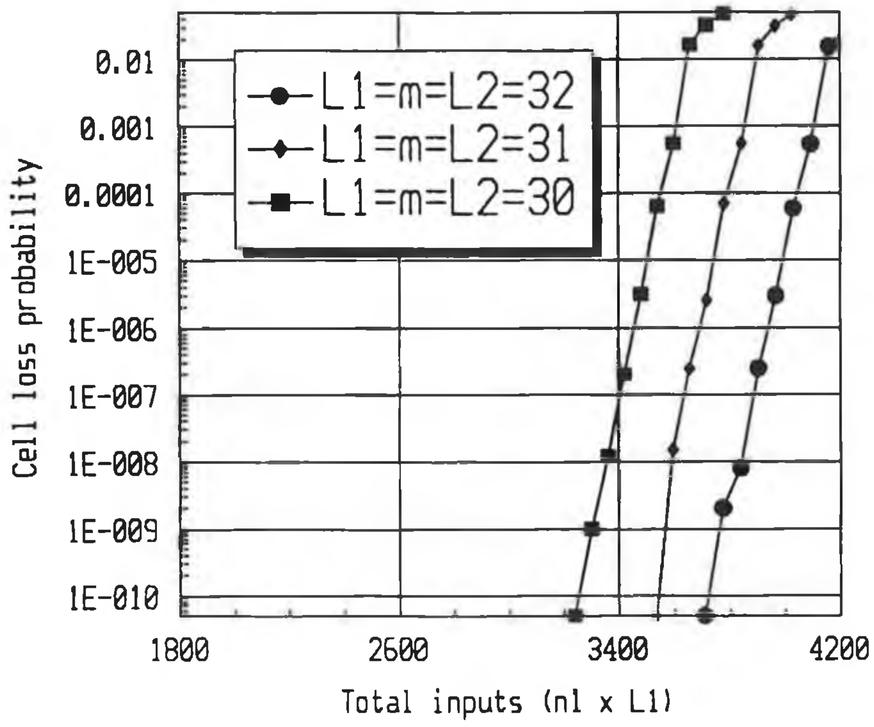


Fig. 6.5: Performance with uniform traffic ($L_1 = m = L_2$, $S_1 = 4$, $S_2 = 8$).

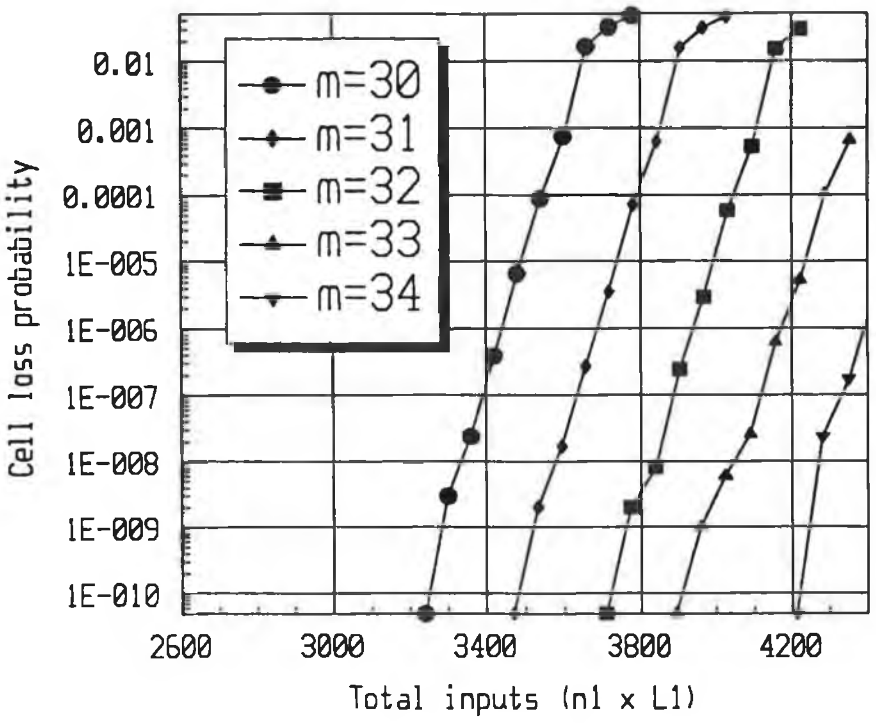


Fig. 6.6: Effect of changing *m* ($L_1 = L_2 = 32$, $S_1 = 4$, $S_2 = 8$).

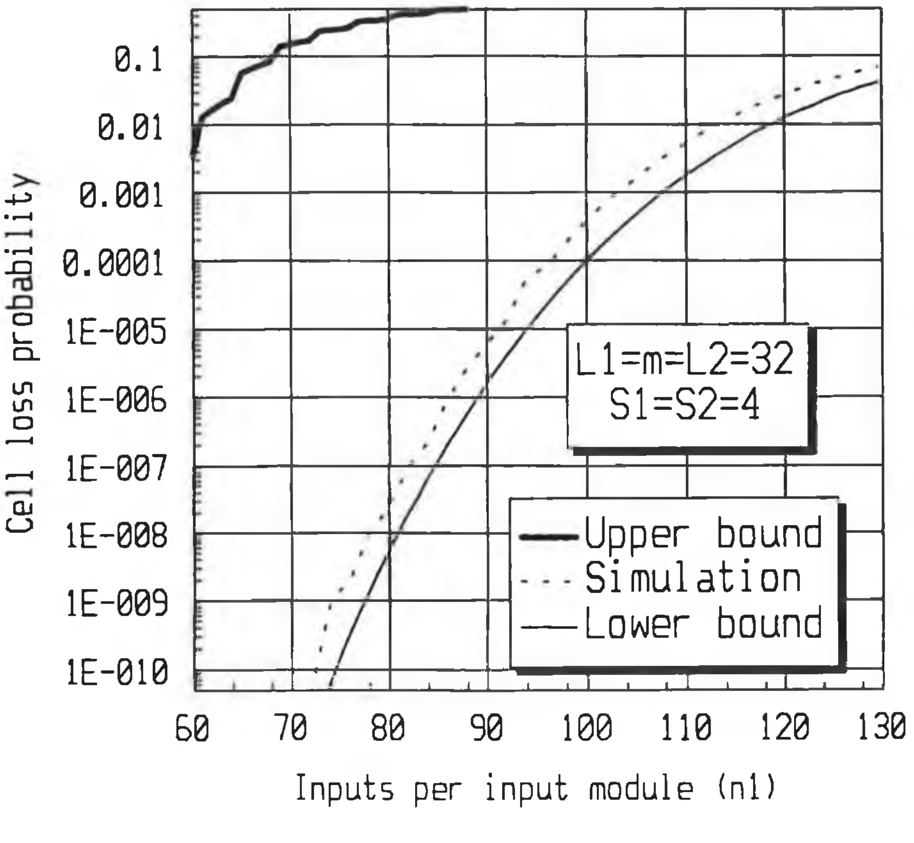


Fig. 6.7: Lower and upper bounds compared to simulation results

($L_1 = m = L_2 = 32$, $S_1 = S_2 = 4$).

The performance of the path allocation algorithm in the presence of non-uniform loading of the outputs, shall now be considered in more detail, in the case of a specific switch. The switch dimensions are chosen to be $L_1 = L_2 = m = 32$, $S_1 = 4$, $S_2 = 8$ and $n_1 = 96$. The effect of progressively increasing the asymmetry of the load on the switch outputs shall now be investigated.

The output modules are divided into two groups. One group (termed the demand group) contains k output modules, with $0 < k < L_2$. The probability of a cell requesting an output module in the demand group is chosen so that the expected number of cells requesting each module in the demand group is $r.S_2m$, where $0 < r < 1$. If a cell does not request an output module in the demand group, it requests one of the remaining $L_2 - k$ output modules, with equal probability.

The cell loss probability is shown in Fig. 6.9 for various values of k and r, in the case where the output modules in the demand group are contiguous. The probability of cell loss is shown only for cells requesting an output module in the demand group. No losses were recorded in the simulations for cells requesting the background group.

The probability of loss can be seen to increase significantly as the size of the demand group (k) is increased. However, this probability stays below 10^{-10} if the load offered to each output module is below 55% of its input capacity. Since each output module has 256 inputs, each can deliver data at the full ATM rate to as many as 140 output ports, without excessive cell loss, even in the presence of a severe traffic imbalance. Setting $n_2 = 140$ in Fig. 6.1, with the values of the other switch parameters as chosen above, results in a 3072 x 4480 switch, with a cell loss probability below 10^{-10} in the presence of a 100% offered load and an asymmetric loading of the outputs. A square (3072 x 3072) switch should have a much lower figure for cell loss. The exact figure cannot be obtained, because of the extreme computational expense associated with simulating events of such low probability.

The cell loss probability depends to some extent on the pattern of traffic imbalance present. Fig. 6.10 compares the loss for a switch with a contiguous demand group and a switch with a demand group whose output modules are interspersed with those carrying background traffic. The variation in cell loss probability is most pronounced in the case of the more unbalanced load ($k = 16$). Note that the cell loss probability is higher for the contiguous demand group.

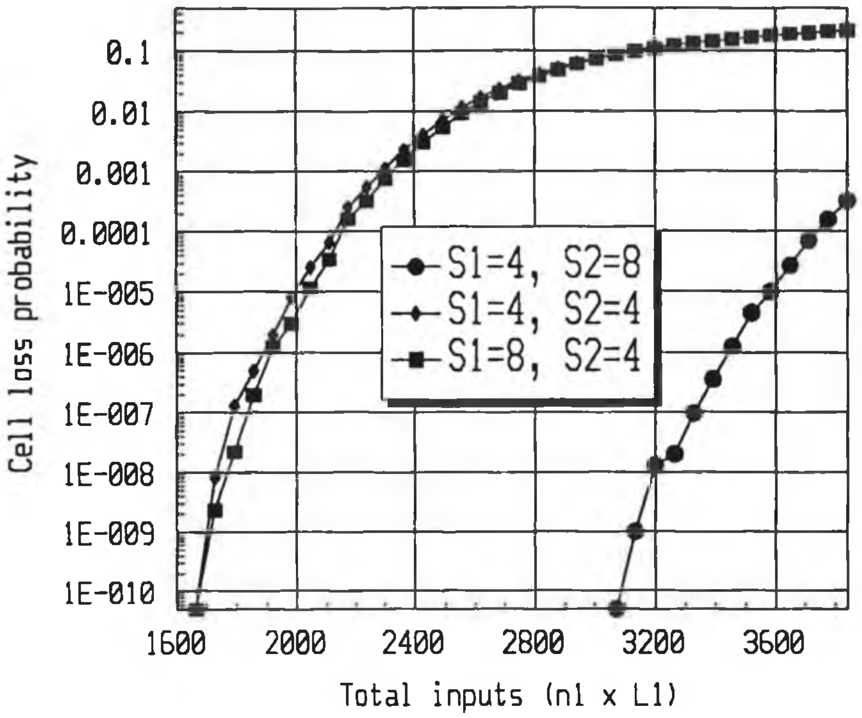


Fig. 6.8: Performance with nonuniform traffic for various values of intermediate stage bandwidth ($L_1 = m = L_2 = 32$).

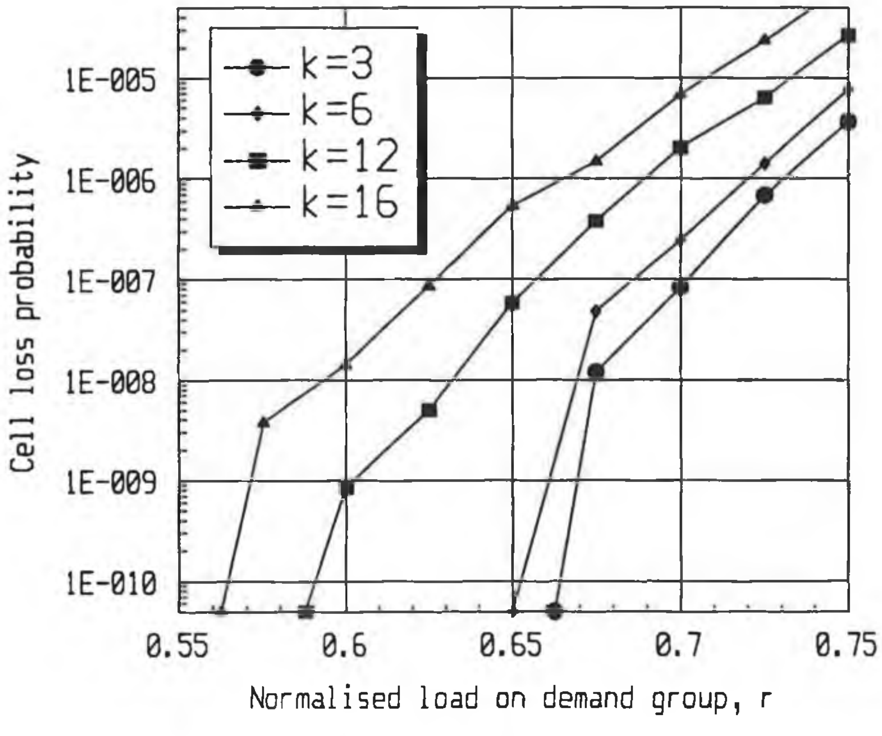


Fig. 6.9: Variation of cell loss with traffic imbalance, where contiguous output modules have a high load ($L_1 = m = L_2 = 32$, $S_1 = 4$, $S_2 = 8$, $n_1 = 96$).

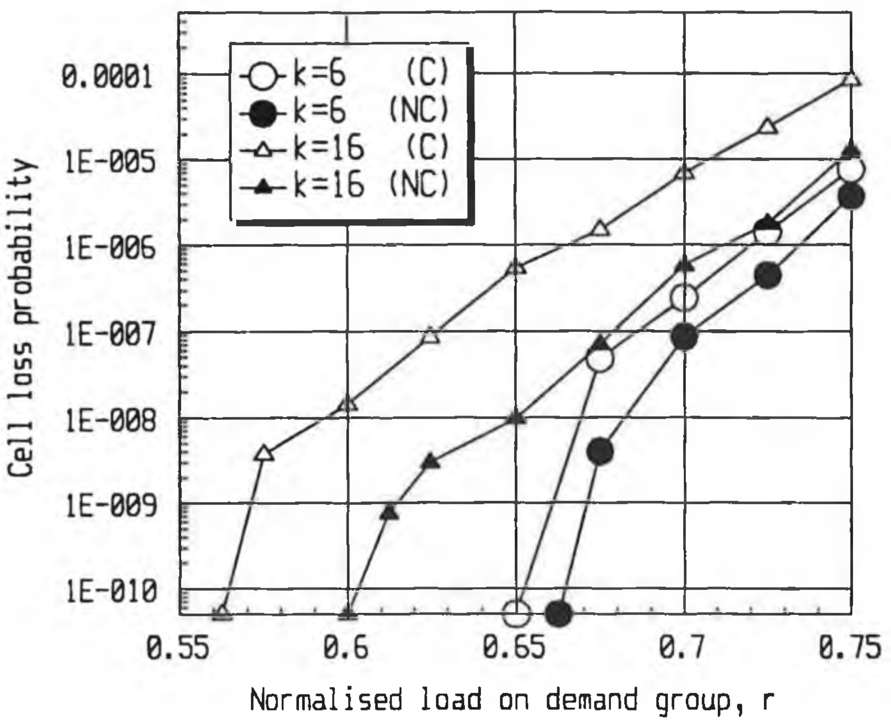


Fig. 6.10: Comparison of performance for contiguous and non-contiguous demand groups ($L_1 = m = L_2 = 32$, $S_1 = 4$, $S_2 = 8$, $n_1 = 96$).

6.4 The output modules.

6.4.1 An output-buffered design.

The output modules are more complex than the switch modules in the other two stages, because they are required to handle contention from multiple cells for the same output port, whereas the operation of the path allocation algorithm ensured that no contention occurred in the input and intermediate stages. A switch with output buffering offers higher throughput than an input-buffered switch. Also, the requirement of cell sequence preservation on a virtual circuit is most easily met by use of an output-buffered switch. The design of such a switch, of dimensions suitable for use in the output stage, and featuring negligible cell loss, is considered below.

The example switch described above requires output modules of dimensions 256 x 96. The direct implementation of such a switch module, as a memory switch, Sunshine switch or knockout switch does not appear feasible, given the current state of the art. An alternative is to implement the output module as a two-stage switch, where the first and second stages are called the expansion and terminal stages respectively. The terminal stage contains output-buffered switches of dimensions M_1 x M_2. The value of M_2 is chosen to be sufficiently small that the terminal stage modules may be implemented as memory switches. Choosing M_2 = 24 means that four such switches are required in the terminal stage. The value of M_1 must be chosen such that the knockout loss at the input side of the memory switch is acceptably low. The cell loss probability due to knockout loss in the terminal stage is shown in Fig. 6.11 for various values of M_1 under the following assumptions:

i. The overall switch has 3072 inputs and outputs

ii. There are 128 (i.e., 3072/24) terminal modules.

iii. Each input port generates a cell in every time slot.

iv. Each cell is equally likely to request any terminal module.

v. There is zero loss during path allocation.

In practice, there will be a finite loss during path allocation, so that the knockout loss will be less than that shown in Fig. 6.11. However, the results shown represent a tight upper bound for the case where the loss during path allocation is low. Choosing a value of 64 for M_1 results in a negligible probability of knockout loss, even though there is a 100% load being offered to the terminal module. The structure of the output module is now as shown in Fig 6.12 (a).

The expansion stage must route cells to the requested terminal module without blocking, if the output module as a whole is to be an output-buffered switch. A possible implementation satisfying this requirement is that shown in Fig. 6.12 (b). The 1 x 2 expanders route cells based on the most significant bit of the destination address. Sorter #0 sorts all cells requesting terminal module TM_0 to one end of the list of 256 outputs, and all cells requesting TM_1 to the opposite end of the list, separated by the inactive cells. Hence cells requesting a given terminal module are concentrated at the sorter outputs which are connected to the appropriate terminal module. No blocking occurs, since contention cannot occur in a sorting network.

The switch design is now complete, apart from the dimensioning of the terminal module output buffers. The switch has 3072 inputs and outputs, and contains 32 96 x 128 switches in the input stage, 32 128 x 256 switches in the intermediate stage, 128 64 x 24 switches in the output stage, as well as 64 256 x 256 sorters (in the expansion stage) and all the circuitry required by the path allocation algorithm. The amount of hardware required may seem excessive, but is justified by the switch performance.

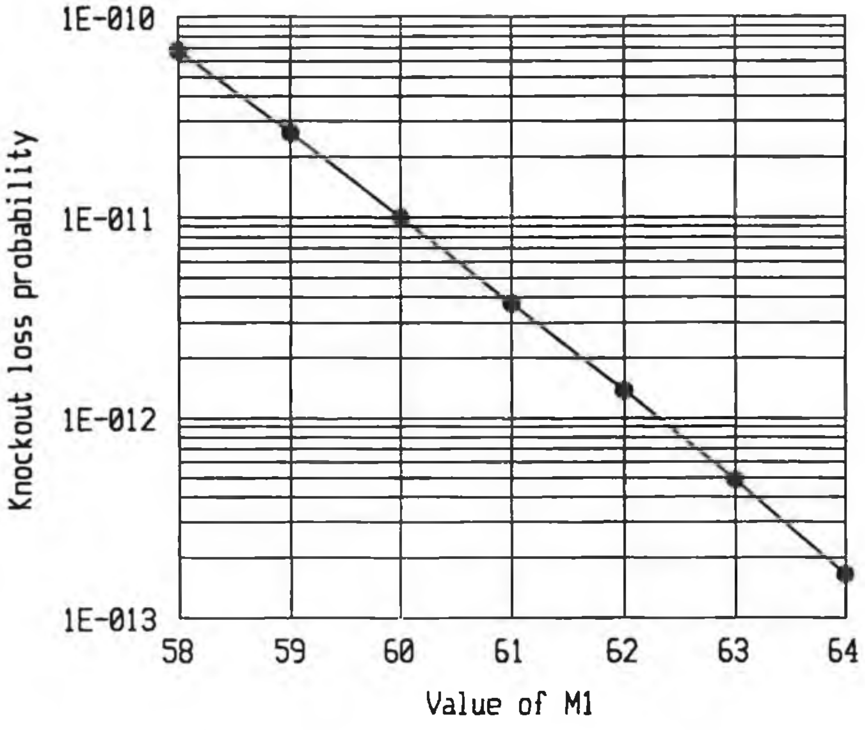
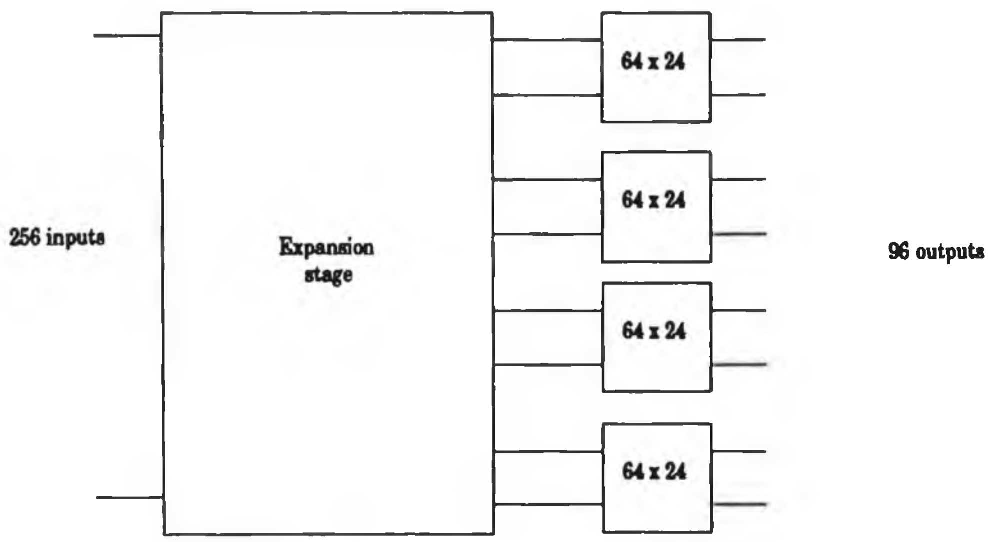


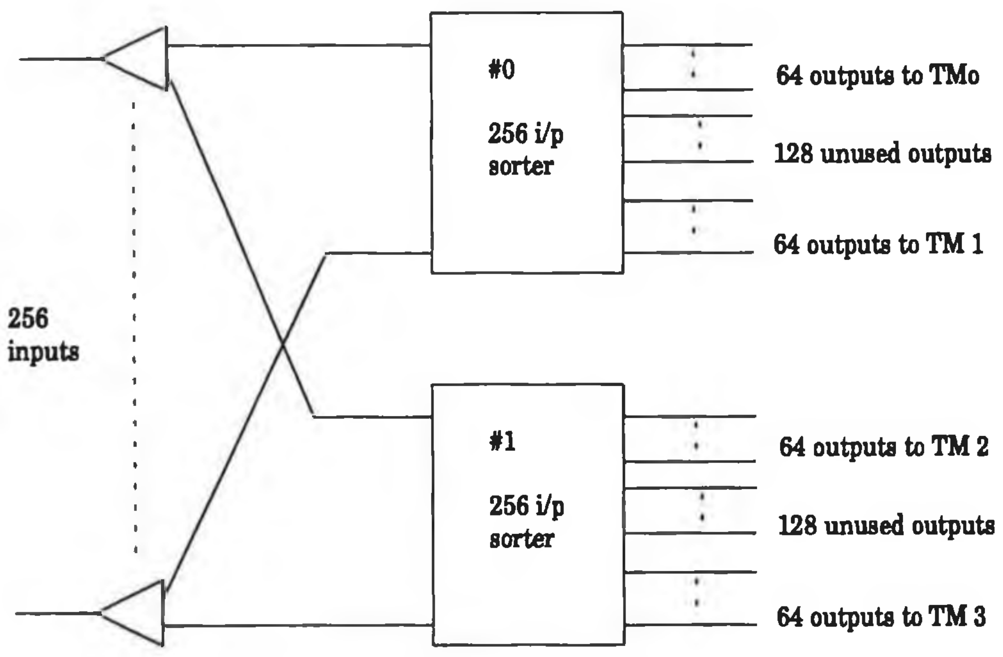
Fig. 6.11: Knockout loss probabilities

The knockout loss is below 2^{-13} (from Fig. 6.11) and the path allocation loss is below 5×10^{-11} (from Fig. 6.9). The actual value of the probability of cell loss during path allocation is probably far less, since the peak load which can be placed on a 256 x 96 output module without saturating its outputs is 96/256 = 37.5%, whereas the simulation results shown in Fig. 6.9 merely indicate that the loss probability is below 5×10^{-11} if the switch has a load of 56% or less. Combining these two figures, and assuming that losses due to non-allocation of paths and to knockout effects are independent, shows that the probability of a cell being lost before arrival in the output buffer is below about 5×10^{-11}. It may readily be shown that a knockout switch with the same performance would need to be able to route 12 cells simultaneously to each output (thereby achieving a knockout loss probability of 2.1×10^{-11}).

This upper bound on loss in the switch is valid even in the presence of 100% loads on input modules and output modules, or in the presence of a load imbalance. Hence, the call admission control process, when deciding whether a call may be placed through the switch, need only consider the loads on the relevant inputs and outputs of the switch, and need not consider its internal characteristics, greatly reducing the complexity of the algorithm required.



(a) Basic structure.

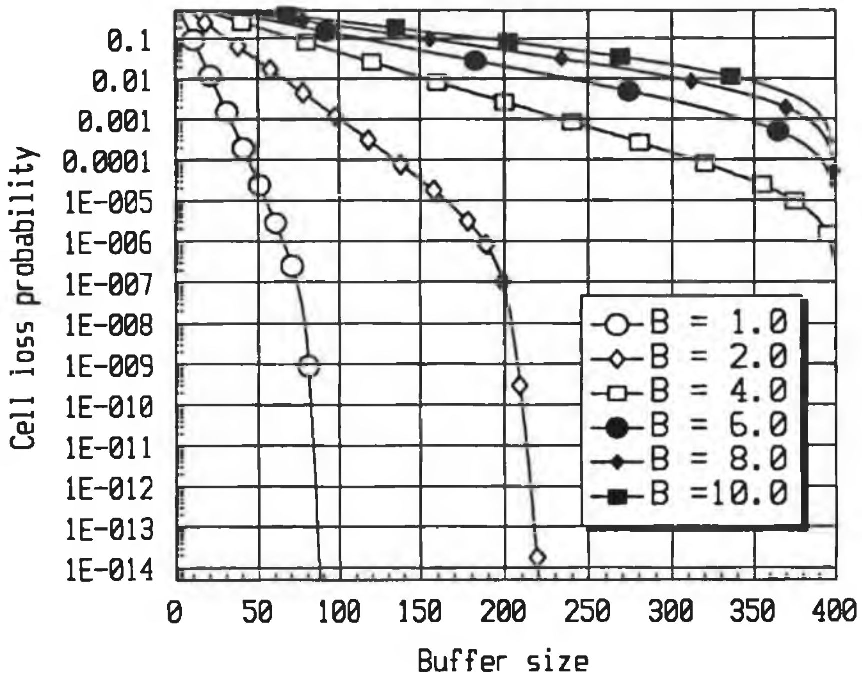


(b) Expansion stage implementation.

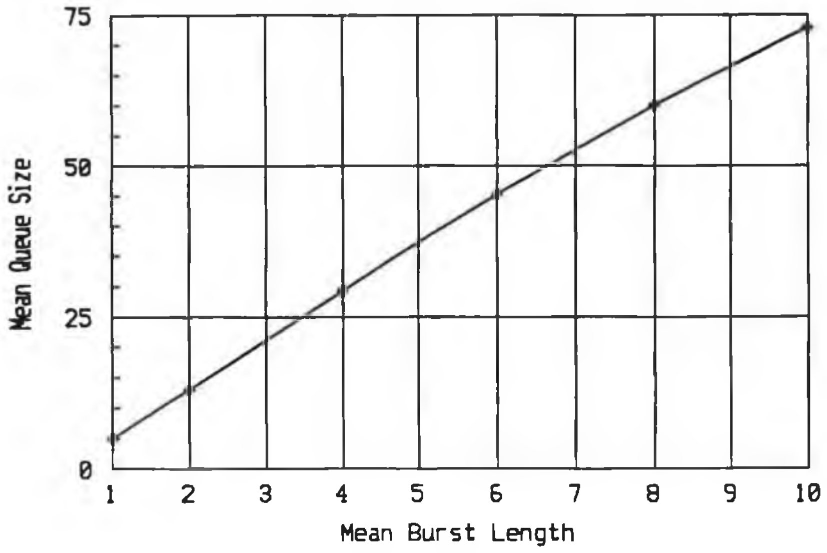
Fig. 6.12: Two-stage output module design.

6.4.2 Dimensioning of output buffers.

The only buffers in the switch are contained at the outputs of the terminal modules. Hence, correlations in the pattern of arriving traffic in consecutive time-slots, due to the presence of bursty sources, do not significantly affect the cell loss probability due to loss of contention during path allocation, or due to knockout effects. Accordingly, the arriving traffic at each input port of the switch has been assumed to be non-bursty (i.e., Bernoulli) heretofore. However, the burstiness of traffic must be explicitly accounted for in dimensioning the output buffers. The model of burstiness used is that described by Descloux [12], which was discussed in Chapter Three.



(a) Cell loss Probability.



(b) Mean Queue size for a buffer length of 300.

Fig. 6.13: Influence of source traffic burstiness on switch performance.

The cell loss probability was determined for various buffer lengths, using the numerical method of Descloux. A 90% load of the output was assumed. Descloux's method assumes that new bursts arrive in accordance with a Poisson process of appropriate rate. A more realistic model might feature a Markov-modulated arrival process, where the number of active bursts is constrained to lie in the range from zero up to the number of switch inputs *N*. The value of *N* in the switch considered here is large (3072), so that the Poisson assumption is quite realistic.

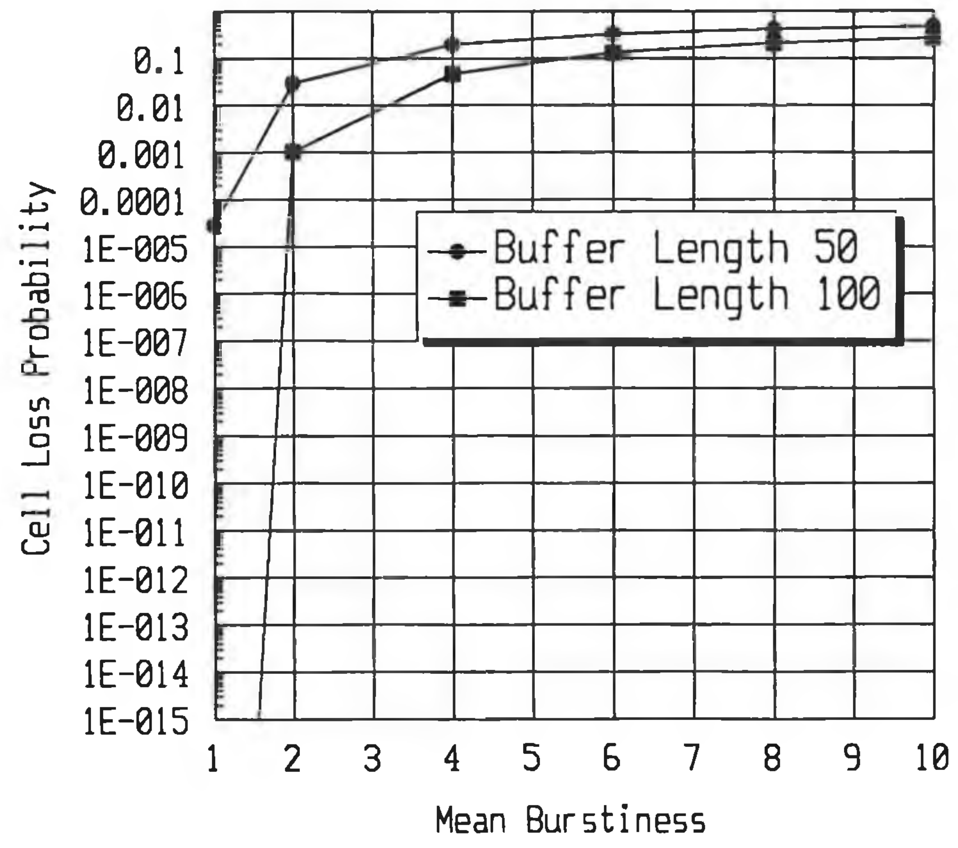


Fig. 6.14: Variation in cell loss probability with burstiness for two buffer sizes.

The results obtained are shown in Figs. 6.13 and 6.14, as a function of *B*, the mean burstiness of each source. The mean queue size increases at a rate close to linearity as *B* increases. This linearity was predicted in section 3.4.2, for the case of a switch with an infinite buffer capacity. It can be seen that the probability of cell loss is very high, even for traffic of moderate burstiness. Hence, the call admission control process should ensure that bursty sources are not admitted. A buffer size of 256 should suffice to keep the cell loss probability below 10^{-12}, provided the mean burstiness of each source is below 2.0. Hence, each terminal module requires 256 x 24 x 53 x 8 = 2.544 Mb of memory. This figure could be reduced substantially if a shared-buffer architecture was used.

References

[1] C.Y. Lee, "Analysis of switching systems", *Bell Syst. Tech. Journal*, vol. 34, no. 6, 1955, pp. 1287-1315.

[2] N. Pippenger, "On crossbar switching networks", *IEEE Trans. on Commun.*, vol. COM-23, no. 6, June 1975, pp. 616-659.

[3] C. Jacobaeus, "A study of congestion in link systems", *Ericsson Technics*, no. 48, 1950.

[4] A. Lotze, A. Roder and G. Thierer, "PPL: a reliable method for the calculation of point to point loss in link systems", Proc. 8th ITC, Melbourne, 1976, pp. 547/1-547/14.

[5] G. Hebuterne, *Traffic Flows in Switching Systems*, Artech House, 1987, pp. 118-120.

[6] A. Jajszczyk, J. Kleban and J. Kubasik, "On congestion in switching networks composed of digital switching matrices", *Proc. 12th ITC*, Elsevier, 1989, pp. 908-914.

[7] E. Valdimarsson, "Blocking in multirate networks", *Proc. Infocom '91*, pp. 579-588.

[8] G. Fitzpatrick, M. Beshai, and E. Munter, "Analysis of large-scale three-stage networks serving multirate traffic", *Proc. ITC-13*, Elsevier, 1991, pp. 905-910.

[9] M. Karol and C.-L. I, "Performance analysis of a growable architecture for broadband packet (ATM) switching", *Globecom '89 Conference Record*, pp. 1173-1180.

[10] K. Eng *et al.*, "A modular broadband (ATM) switch architecture with optimum performance", *Proc. of the International Switching Symposium*, Stockholm, 1990, vol. IV, pp. 1-6.

[11] M. Collier and T. Curran, "Cell-level path allocation in a three-stage ATM switch", submitted for publication.

[12] A. Descloux, "Contention probabilities in packet switching networks with strung input processes", *Proc. ITC 12*, pp. 815-821, 1989.

7. CONCLUSIONS.

The research undertaken may be classified under the following headings:

- a survey of ATM switching techniques and their performance;
- a study of the topology of binary self-routing networks;
- the design of a new ATM switch, suitable for the implementation of large switches.

A survey has been undertaken of existing proposals for ATM switch architectures. Three basic types of ATM switch have been compared:

- internally blocking switches of the banyan type;
- non-blocking switches with input buffering;
- output-buffered switches.

Banyan switches provide only a single path from source to destination, which is shared with other source and destination pairs. Hence, they are highly blocking networks. Banyan networks feature low throughput whether constructed using buffered or unbuffered switch elements, and whether or not flow control (in the form of a back pressure mechanism) is used.

Since the throughput is low, the interstage links of a banyan must have a bandwidth higher than that required for transmission of a single ATM cell. This can be achieved using parallelism, dilation (i.e., channel grouping) or link speedup. None of these options is attractive, since they require either very high speed integrated circuits, or complex interconnections of integrated circuits.

A number of switches based on the banyan network, but featuring multiple paths between source and destination, so as to increase throughput, have been proposed. The throughput increases only slowly with the complexity of the switch in many cases, and many of the switch designs require call-level routing to be used. Two switches which appear to be promising for ATM applications are the rerouting banyan and Kim's banyan-distributor switch. These offer throughputs close to unity, when uniformly loaded. However, neither switch is internally non-blocking, so that successful delivery of a cell through the switch can never be guaranteed.

Input-buffered internally non-blocking switches are typically implemented using Batcher-banyan techniques. The throughput of such switches, while greater than for a banyan network, is low, because of head-of-line blocking, being theoretically equal to 58.6% in the case of a large switch with a uniform load. However, the effect of head-of-line blocking can be reduced in a number of ways.

The FCFS queuing discipline at input buffers can be relaxed, to allow multiple arbitrations to be performed in each time slot ('windowing'). The number of cells which can be transferred to each output port from each input port can be increased, so that some of the buffering requirement is moved to the output side of the switch. Additionally, if the switch provides expansion (i.e., a greater number of outputs than inputs) head-of-line blocking is reduced, and therefore throughput increases.

Batcher-banyan switches provide unfair access to the output ports unless carefully designed. The problem arises because of the rigid top-down precedence of the sorted output, if the sort elements route identical data straight through from the inputs to the corresponding outputs. This unfairness results in preferential access for cells from one end of the list of inputs to the requested output ports, in the event of output contention, when sorting is based only on the requested destination. A number of methods are available to alleviate this problem of unfair access. All lead either to an increase in switch complexity, or require a speedup of the Batcher network.

The internally nonblocking property of the Batcher-banyan switch makes it ideally suited to applications where cells have been scheduled to avoid output contention. However, the modifications required to ensure fair access and high throughput when cell arrivals are unscheduled, together with the $O(N \log^2 N)$ complexity of an N-input Batcher network make the Batcher-banyan technique unattractive for large switches.

A wide range of output-buffered switches have been proposed. Most feature a complexity of $O(N^2)$ for an N-input switch. If a shared medium is used, the required bit-rate increases with $O(N)$. Hence, large output-buffered switches are impractical. Shared-medium switches such as memory switches are the solution of choice where a small ATM switch is required. They offer high performance (many having zero probability of cell loss prior to arrival at the output buffer), high throughput and low delay, and, importantly, their performance can be evaluated with comparative ease. This simplifies the task of network planning, and also simplifies the operation of the call admission control process, since the switch can be represented, to a close approximation, as being ideal.

The time scale over which routes are allocated has been considered. The two types of routing considered were cell-level and call-level routing. It seems reasonable to expect that the link utilisations obtainable when using call-level routing will be lower than those obtainable in the cell-level case. ATM is a connection-oriented protocol, and therefore routing between ATM switches must be performed at the call level. However, within a switch, cell-level routing is to be preferred, whenever it is feasible, because of the anticipated greater efficiency of bandwidth utilisation, and because the overhead associated with call-level routing within the switch is eliminated. This has been a key consideration in designing the switch described by this research.

Binary self-routing networks are widely used as components of ATM switches, Accordingly, the properties of binary self-routing networks have been investigated. Attention has been focused on networks whose interconnections can be represented by binary permutations, since the majority of binary self-routing networks of interest in ATM fall within this class. A notation has been proposed to describe these binary link permutations which leads to a very economical and unambiguous description of a given network. This method of description may be applied to specify any network within the class of interest, and is a useful tool for the automatic layout of interconnections when designing such networks at the integrated circuit and board levels.

The unified description of these networks facilitates a comparison of their respective properties. The issue of equivalence in binary self-routing networks has been considered in some detail. A formal mathematical method for determining the nature of the equivalence between two binary self-routing networks has been developed, which is quite general in its application. The only restriction on the two networks is that they must belong to the class described above. This method has obvious practical applications since it can be used to develop a formal procedure for constructing an arbitrary binary self-routing network (of the above class) from integrated circuits of a single type (such as 8 x 8 baseline network chips).

The above method can also be applied to quickly obtain the correct routing strategy for a given self-routing network, based on the nature of its equivalence to a baseline network. The correct routing strategy is that which forwards the cell to the output port requested in its routing tag. Switching must be based on the correct sequence of routing tag bits if this is to occur. This method has been used to formally verify the routing strategy for the rerouting banyan. It also allows the necessary and sufficient condition on the traffic submitted to the network for blocking to be avoided to be readily obtained.

This non-blocking condition has been used to prove that a concentrated monotonic list of data (sorted by destination address) can be routed without contention to the requested destinations by an omega network, or a network exactly equivalent to it. Proofs of the sufficiency of this condition have been published before, but the proof shown here is the first to relate this sufficient condition to the necessary and sufficient condition. It was similarly proven that an indirect binary n-cube may act as a non-blocking concentrator.

It is apparent from the properties of the ATM switching techniques described above that no single design of ATM switch is suitable when a large switch is required. Hence, a large switch must be designed in a modular fashion, with modules of a size suitable for implementation using the above techniques. A modular switch also has the advantage that switch growth may be more easily accomplished.

The modular switch may feature single-path or multiple-path routing. A two-stage switch provides only a single path between any pair of input and output modules, although the path capacity can be augmented through the use of channel grouping. Selection of paths is obviously trivial in this case, but the switch may offer poor performance in the presence of a non-uniform loading of the outputs, unless the bandwidth of paths is engineered conservatively.

A three-stage switch shares the bandwidth of the intermediate stage among all the switch inputs. Hence, it may result in superior performance in the presence of a non-uniform loading of the switch outputs, since there will be a negative correlation between the requests for access to intermediate stage modules from cells requesting different output modules. However, routing is more complex than in the case of a two-stage switch, since there is a choice of paths available through the intermediate stage.

The design of a three-stage ATM switch was undertaken. Attention was focused on cell-level path allocation. The frequency with which the routing is updated in this case should result in more efficient use of bandwidth than when a call-level algorithm is used. It was shown that speedup is required on the input and output links of the intermediate stage modules to satisfy the non-blocking condition for a three-stage ATM switch when a call-level path allocation algorithm is used, but that such a speedup is not required for cell-level path allocation.

The key to the successful design of a three-stage ATM switch featuring cell-level path allocation is obviously the choice of path allocation algorithm, and of hardware to implement it. A number of existing proposals for cell-level path allocation have been reviewed. The speed of operation required of the corresponding hardware increases linearly with the number of switch inputs. This limits the achievable switch size. Accordingly, an algorithm has been developed for path allocation whose required speed of operation increases less rapidly with switch size.

The new algorithm for cell-level path allocation was compared to the cell scheduling algorithm of the growable packet switch. Its advantages over the latter algorithm include the following.

- The execution time of the cell scheduling algorithm is $O(N)$ for an N-input switch with $m < N$ modules in each stage, compared to $O(m \lceil \log(N/m) \rceil)$ for the new algorithm.

- The cell scheduling algorithm cannot easily be applied to a switch with intermediate channel groups, whereas the new algorithm is intended for such applications.

- Similarly, the cell scheduling algorithm cannot readily be applied when a switch has non-square intermediate stage modules, unlike the new algorithm.

- The cell scheduling algorithm is unfair in that, when the inputs and outputs of the switch are uniformly loaded, the cell loss probability varies between pairs of input and output modules. The cell loss probability is uniform for the algorithm proposed here.

The new algorithm has the disadvantage over the cell scheduling algorithm that more hardware is required. However, since the required operating speed will be less than for the cell scheduling algorithm in most cases, costs may be reduced through the use of slower and cheaper integrated circuits.

The viability of the algorithm has been demonstrated by means of a detailed study of the hardware required, at the functional and logic levels. A large array of processors is required. Two bit-serial implementations of the processor have been presented, to demonstrate its simplicity, and to assess the execution time for one iteration of the algorithm.

Two additional functions must be carried out by the hardware, namely the counting of requests for each output module, which is necessary to initialise the algorithm, and routing tag assignment, which involves forwarding the results of the path allocation process to the relevant cells. Two methods have been presented for the implementation of each function, a slow method using simple hardware, and a faster method, using more complex hardware. Hence, circuit complexity can be traded off against circuit speed, larger switches requiring the more complex hardware, so as to reduce the clock speed requirements.

Extensive use has been made of banyan networks in implementing these functions, and the conditions under which such networks are non-blocking have been exploited to the full, to ensure that delays due to internal contention are avoided. A large sorter is required by the algorithm to merge the data cells arriving to each input module with a set of control packets. The number of inputs required of this sorter may limit the achievable switch size. Hence, modifications to the above hardware have been presented which allow the merge function to be eliminated.

The new path allocation algorithm may be applied either to a switch containing only output buffering, or to a switch with a combination of input and output buffering. The former choice requires less hardware. Accordingly, the performance of the switch has been evaluated, by means of computer simulation, for an output-buffered switch.

The following conclusions may be drawn from the simulation results.

- The best performance is achieved, in the case of uniformly loaded output modules, when the intermediate stage modules are expansion modules. There are then more links from a given intermediate module to each output module than there are from each input module to the intermediate module. This result is intuitively reasonable, since a cell may be routed via any intermediate stage module, but must be sent to the requested output module.

- The cell loss probability increases when the output modules are non-uniformly loaded. However, the increase is more pronounced when square intermediate stage modules are employed.

The scheduling of traffic to the input and intermediate modules of the switch means that these can be of simple design, provided that they are non-blocking. Batcher-banyan switches are suitable for this application. However, output-buffered switches must be used in the output stage, to ensure good throughput, and the preservation of cell sequence.

The use of two different types of switch module causes difficulties when selecting the size of switch module to be used. The path allocation algorithm is easier to implement, for a given switch size, if larger switch modules are used. However, the maximum size of output-buffered switch which may be economically constructed will be much smaller than in the case of a Batcher-banyan switch, because of its greater complexity.

This difficulty is circumvented by implementing the output modules as two-stage switches. The first stage is a non-blocking distributor, which routes cells to the appropriate output-buffered switch in the second stage. The distributor is implemented using Batcher sort networks.

The resulting switch can have many thousands of inputs and outputs, and contains buffers only at the output ports. The probability of cell loss prior to arrival in the output buffer can be reduced to a negligible level by appropriately dimensioning the intermediate stage of the switch, and the two-stage output modules.

The major source of loss in the switch is buffer overflow in the output buffers. This can be a significant source of loss, when bursty traffic is present, even with a large buffer capacity. This problem is intrinsic to the operation of

ATM, and will be shared by any output-buffered switch. Excessive cell loss will be prevented by appropriate call admission control and flow enforcement procedures.

APPENDIX A

Discrete-Time Queuing Models.

A.1 Introduction.

Some selected results from discrete-time queuing theory are presented below. Derivations of these results may be found in [1,2] except where otherwise stated.

A.2 Notation and assumptions.

Each unit of time (time-slot) is divided into two phases, an arrival phase (when arrivals can occur) and a departure phase (when departures can occur).

A cell, arriving at the start of a time-slot (i.e. during the arrival phase) to an empty system may obtain service in the same time-slot (i.e. during the departure phase which immediately follows). The service discipline is First-Come-First-Served. The service time is the duration for which a cell waits in the head-of-line (HOL) position, i.e., at the head of the queue. The service time is considered to be unity if a cell enters and departs the system within a single time-slot. Service times are independently but identically distributed.

The sojourn time is the total amount of time spent by a cell in the system. The queuing time is the time spent in the system prior to reaching the HOL position, i.e., it equals the sojourn time less the service time.

The following notation is used, and applies to a system operating in a steady state.

ρ: system utilisation.

N_D: the queue size following a departure.

W_S: the sojourn time for a cell.

W_Q: the queuing time for a cell.

G: the service time for a cell.

A_i: the number of cells arriving in one time slot.

$N_D(z)$: moment generating function for the queue size immediately following a departure .

$\overline{N}_D$: mean queue size after departures.

$W_S(z)$: moment generating function for the sojourn time.

$W_Q(z)$: moment generating function for the queuing time.

$\overline{W}_S$: mean sojourn time.

$\overline{W}_Q$: mean queuing time.

$G(z)$: moment generating function for the service time.

$\overline{G}$: mean service time.

A.3 The Geo/Geo/1 queue.

Suppose the arrival process features geometrically distributed inter-arrival times with parameter λ, and geometrically distributed service times with parameter μ. Then

$$N_D(z)=\frac{(\mu-\lambda)}{\mu(1-\lambda)-\lambda z(1-\mu)},$$

provided that

$$\rho=\frac{\lambda}{\mu}<1.$$

The probability P_j that the queue size (after a departure) is j is given by

$$p_j=\left(1-\frac{\lambda}{\mu}\frac{(1-\mu)}{(1-\lambda)}\right)\left(\frac{\lambda(1-\mu)}{(1-\lambda)\mu}\right)^j,j>0.$$

The mean queue size is

$$\overline{N}_D=\frac{\lambda(1-\mu)}{(\mu-\lambda)}.$$

The queuing time distribution may be found using

$$W_Q(z)=\frac{(1-\rho)(1-(1-\mu)z)}{(1-\lambda)-(1-\mu)z}$$

and the mean queuing time is

$$\overline{W}_Q=\frac{\lambda}{\mu}\frac{(1-\mu)}{(\mu-\lambda)}.$$

A.4 The Geo/G/1 queue.

If the service time has an arbitrary distribution with moment generating function *G(z)* and mean $\overline{G}$ then the resulting Geo/G/1 queue has a moment generating function for the stationary distribution of the number of cells after departures given by

$$N_D(z)=(1-\lambda\overline{G})\frac{1-z}{G(1-\lambda+\lambda z)-z}\frac{G(1-\lambda+\lambda z)}{1-\lambda+\lambda z}.$$

This process can be shown to have a mean of

$$\overline{N}_D=\frac{\lambda^2(E[G^2]-\overline{G})}{2(1-\lambda\overline{G})}+\lambda(\overline{G}-1).$$

It can be shown that the formulae for Geo/G/1 reduce to the formulae for Geo/Geo/1 in the special case of geometric service, when $G(z)=\frac{\mu z}{1-(1-\mu)z}$.

The queuing time is

$$W_Q(z)=\frac{(1-\lambda\overline{G})(1-z)}{1-\lambda+\lambda G(z)-z}.$$

Hence the mean queuing time may be obtained as

$$\overline{W_Q}=\frac{\lambda(E[G^2]-\overline{G})}{2(1-\lambda\overline{G})}.$$

The following relationships exist between the queuing time, sojourn time and queue size in the Geo/G/1 queue:

$$W_S = W_Q + G;$$

$$N_D = \sum_{i=1}^{W_S-1} A_i .$$

It follows that

$$W_S(z)=W_Q(z)G(z),$$

$$N_D(z)=\frac{W_S(1-\lambda+\lambda z)}{1-\lambda+\lambda z},$$

and that

$$\overline{N_D} = \overline{A}(\overline{W_S}-1).$$

A.5 Other queuing models.

The Geo/G/1 model may be extended to consider batch arrivals, i.e. the Geo$^{(A)}$/G/1 queue. It is assumed that cells arriving in the same batch are served in random order. Kruskal, Snir and Weiss [3] have shown that the queuing time in a Geo$^{(A)}$/G/1 queue has moment generating function

$$W_Q(z)=\frac{1-\overline{A}\overline{G}}{\overline{A}}\frac{1-z}{1-G(z)}\frac{1-A(G(z))}{A(G(z))-z}$$

and mean

$$\overline{W_Q}=\frac{\overline{G}(E[A^2-A])+\overline{A}^2(E[G^2-G])}{2\overline{A}(1-\rho)}.$$

Oie *et al.* [4] have obtained $N_D(z)$ for a Geo$^{(A)}$/D/L queue to be of the form

$$N_D(z) = \frac{\sum_{k=0}^{L-1} p_k \sum_{j=0}^{L-k-1} a_j (z^{k+j} - z^L)}{A(z) - z^L}$$

where p_k = Prob[N_D=k] and a_j = Prob[j arrivals] and hence have found, for a Poisson batch distribution, $A(z) = e^{-\lambda(1-z)}$, that

$$\overline{N}_D = \frac{\lambda^2 - L(L-1)}{2(L-\lambda)} + \sum_{j=1}^{L-1} \frac{1}{1-z_j}$$

where z_j is a root of the equation $A(z) = z^L$.

References

[1] J.J. Hunter, *Mathematical Techniques of Applied Probability*, vol. 2, Academic Press, 1983.

[2] T. Meisling, "Discrete time queueing theory", *Oper. Res.*, vol. 6, pp. 96-105.

[3] C.P. Kruskal, M. Snir and A. Weiss, "The distribution of waiting times in clocked multistage interconnection networks", *Computer*, vol. C-37, no.11, pp.1337-1352, Nov. 1988.

[4] Y.Oie, M. Murata, K. Kubota and H. Miyahara, "Effect of speedup in nonblocking packet switch",*Proc. ICC '89*, pp. 410-414.

Publications arising from this research.

The following work has been published as a result of this research:

Martin Collier and Tommy Curran, "Path allocation in a three-stage broadband switch with intermediate channel grouping", *Proc. Infocom '93*, pp. 927-934, San Francisco, Mar.-Apr. 1993.

The following papers have been submitted for publication:

Martin Collier and Tommy Curran, "Cell-level path allocation in a three-stage ATM switch".
Martin Collier and Tommy Curran, "The strictly nonblocking condition in three-stage networks".

Glossary of acronyms used in the text.

AG:	Address Generator.
ATD:	Asynchronous Time Division.
ATM:	Asynchronous Transfer Mode.
BISDN:	Broadband ISDN.
BR:	Bit Reversal.
CCITT:	International Telegraph and Telephone Consultative Committee.
CLP:	Cell Loss Priority.
CMOS:	Complementary Metal-Oxide-Silicon.
FIFO:	First-In-First-Out.
GFC:	Generic Flow Control.
GPDN:	Generalised Parallel Delta Network.
HEC:	Header Error Control.
HOL:	Head-Of-Line.
IM:	Input Module.
IPS:	Inverse Perfect Shuffle.
ISDN:	Integrated Services Digital Network.
ISM:	Intermediate Stage Module.
LP:	Link Permutation.
NTD:	Null Token Detector.
OM:	Output Module.
PS:	Perfect Shuffle.
PT:	Payload Type.
RACE:	Research in Advanced Communications in Europe.
RAR:	Read Address Register.
RPG:	Routing Packet Generator.
SDH:	Synchronous Digital Hierarchy.
ST:	Straight-Through.
STM:	Synchronous Transfer Mode.
TA:	Token Address.
TC:	Token Count.
VCI:	Virtual Channel Identifier.
VPI:	Virtual Path Identifier.