Symmetric Rearrangeable Networks and Algorithms

By

Amitabha Chakrabarty

Thesis directed by:

Dr. Martin Collier

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

School of Electronic Engineering

Dublin City University

ABSTRACT

A class of symmetric rearrangeable nonblocking networks has been considered in this thesis. A particular focus of this thesis is on Beneš networks built with 2×2 switching elements. Symmetric rearrangeable networks built with larger switching elements have also being considered. New applications of these networks are found in the areas of System on Chip (SoC) and Network on Chip (NoC). Deterministic routing algorithms used in NoC applications suffer low scalability and slow execution time. On the other hand, faster algorithms are blocking and thus limit throughput. This will be an acceptable trade-off for many applications where achieving "wire speed" on the on-chip network would require extensive optimisation of the attached devices. In this thesis I designed an algorithm that has much lower blocking probabilities than other suboptimal algorithms but a much faster execution time than deterministic routing algorithms. The suboptimal method uses the looping algorithm in its outermost stages and then in the two distinct subnetworks deeper in the switch uses a fast but suboptimal path search method to find available paths. The worst case time complexity of this new routing method is $O(N\log N)$ using a single processor, which matches the best known results reported in the literature.

Disruption of the ongoing communications in this class of networks during rearrangements is an open issue. In this thesis I explored a modification of the topology of these networks which gives rise to what is termed as repackable networks. A repackable topology allows rearrangements of paths without intermittently losing connectivity by breaking the existing communication paths momentarily. The repackable network structure proposed in this thesis is efficient in its use of hardware when compared to other proposals in the literature.

As most of the deterministic algorithms designed for Beneš networks implement a permutation of all inputs to find the routing tags for the requested input-output pairs, I proposed a new algorithm that can work for partial permutations. If the network load is defined as ρ, the mean number of active inputs in a partial permutation is, $\overline{m} = \rho N$, where N is the network size. This new method is based on mapping the network stages into a set of sub-matrices and then determines the routing tags for each pair of requests by populating the cells of the sub-matrices without creating a blocking state. Overall the serial time complexity of this method is $O(N\log N)$ and $O(\overline{m}\log N)$ where all N inputs are active and

with $\overline{m} < N$ active inputs respectively. With minor modification to the serial algorithm this method can be made to work in the parallel domain. The time complexity of this routing algorithm in a parallel machine with N completely connected processors is $O(\log^2 N)$. With $\overline{m}$ active requests the time complexity goes down to $(\log\overline{m}\log N)$, which is better than the $O(\log^2\overline{m} + \log N)$, reported in the literature for $2^{\frac{1}{2}[(\log^2 N - 4\log N)^{\frac{1}{2}} - \log N]} \leq \rho \leq 1$. I also designed multistage symmetric rearrangeable networks using larger switching elements and implement a new routing algorithm for these classes of networks.

The network topology and routing algorithms presented in this thesis should allow large scale networks of modest cost, with low setup times and moderate blocking rates, to be constructed. Such switching networks will be required to meet the bandwidth requirements of future communication networks.

ACKNOWLEDGEMENTS

I would like to express my thanks to everyone who helped me throughout my PhD journey. But above everyone, my sincere gratitude goes to my supervisor, Dr. Martin Collier, without whom this journey would not have been possible. Thanks go to him for providing me with the right mix of freedom and guidance throughout my period in DCU.

I wish to thank Prof. Barry McMullin, Prof. Noel O'Connor, Dr. Conor McArdle and Dr. Eugen Schenfeld for serving on my doctoral committee and for their comments and suggestions.

Thanks go to all the faculty members of School of Electronic Engineering, especially Dr. Noel Murphy for his kindness towards me. Thanks should also go to the members of the Switching and Systems Laboratory for making my stay at the lab such a pleasant experience.

I would like to thank all my friends in Dublin for being there with me always. Thanks go to my elder brother, Dr. Debasish Chakraborty and his family for their continuous support throughout. Lastly and most importantly thanks go to my parents for allowing me to study outside Bangladesh for my degree and to stay away from them for so long.

LIST OF PUBLICATIONS

- **A. Chakrabarty**, M. Collier, Symmetric Rearrangeable Networks: A New Routing Algorithm and Emerging Applications, International Journal of Electronics and Communications (Elsevier) (Submitted).
- **A. Chakrabarty**, M. Collier, Optimum Cost Multistage Symmetric Repackable Networks, International Journal of Grid and High Performance Computing (IJGHPC),IGI Global.(In press)
- **A. Chakrabarty**, M. Collier, And S. Mukhopadhyay, Adaptive Routing Strategy for Large Scale Rearrangeable Symmetric Networks , International Journal of Grid and High Performance Computing (IJGHPC),IGI Global, 2010.
- **A. Chakrabarty**, M. Collier, *Efficient Implementation of Symmetric Multistage Repackable Networks* . Proceeding of the International Conference on Computational Intelligence and Communication Networks (CICN-2010). IEEE Computer Society Press, 2010.
- **A. Chakrabarty**, M. Collier, And S. Mukhopadhyay, *Symmetric Rearrangeable Networks: Algorithms and Rearrangement Limits* . Proceeding of the 7th International Conference on Information Technology : New Generations. IEEE Computer Society Press. Las Vegas, Nevada, USA, April 12-14, 2010.
- **A. Chakrabarty**, M. Collier, And S. Mukhopadhyay, *Matrix-Based Routing Algorithm for Benes Networks* . Proceeding of the International Conference on Future Computational Technologies and Applications (FUTURE COMPUTING 2009). IEEE Computer Society Press. Athens, Greece, November 15-20, 2009.
- **A. Chakrabarty**, M. Collier, And S. Mukhopadhyay, *Dynamic Path Selection Algorithm for Benes Networks*. Proceeding of the IEEE International conference on Computational Intelligence, Communication Systems and Networks (CICSyN2009). IEEE Computer Society Press, 2009.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

The telecommunication standardization sector of ITU (International Telecommunications Union) defines switching as:

"The establishing, on demand, of an individual connection from a desired input to a desired output within a set of inputs and outputs for as long as is required for the transfer of information".

In its early days switching only involved connecting two communicating parties for telephone service. In recent days switching devices are required to handle much more than voice service. Switches must support high speed data and video communication, LAN to LAN communication, large file transfers and cable TV transmissions. A recent report[1] shows that file sharing consumes 25% of the global internet traffic and video (such as streaming video, Flash, and Internet TV) consumes 26% percent. Over one-third of the top 50 sites by volume are video sites. VoIP services such as Skype traffic grew by over 40%[2]. It is estimated that around 70% or more of broadband bandwidth is consumed by downloads

[1]http://gigaom.com/2010/10/26/why-broadband-changes-everything/

[2]http://technews.tmcnet.com/voip-software/topics/voip-software/articles/132217-skype-grows-global-traffic-40-percent-2010.htm

of music, games, video, and other content[3].Considering the increasing demand it is not possible to connect all the users with direct communication links as the cost grows exponentially. Hence various switch and switching mechanisms are in place to reduce the communication cost and increase the reliability of communication.

1.1 Motivation

Switching network architectures in use in recent communication networks may be catagorised as blocking or nonblocking. Nonblocking networks can be further classified into three different classes, *strict sense nonblocking* networks, *wide sense nonblocking* networks and *rearrangeable nonblocking* networks. Among these classes, rearrangeable nonblocking networks have the best scaling properties and are the subject of this thesis.

The properties of symmetric rearrangeable networks have been studied extensively in the literature, initially in the context of possible applications in telecommunication and subsequently for use in high-performance computing. The latter community has explored the use of parallel routing algorithms for such networks since they expected them to be deployed in support of parallel computing systems. New applications for such networks recently emerged for which these routing algorithms are unsuitable. The need for automatic deployment of digital subscriber line (DSL) technologies requires a capability for *analog* switching of perhaps thousands of subscriber loops. Secondly, increasing chip density is giving rise to Network on Chip (NoS) and System on Chip (SoC) solutions where the switching requirements are such as to make fully-connected or crossbar solution inefficient. Report[4] shows that big telecommunication companies are moving towards the use of NoC applications. Parallel routing solutions are unsuitable in

[3]http://www.cisco.com/en/US/prod/collateral/ps7045/ps6129/ps6133/ps6150/prod-white-paper0900aecd8023500d.html

[4]http://www.arteris.com/customers.php

such devices. This thesis explores the optimal design of multistage switching systems for analog and on-chip applications.

Possible blocking in a rearrangeable network is overcome by rearranging the state of the network. For time slatted operations this can be achieved in between the time slots, but for analog communication this will lead to disruption of the ongoing transmission. To overcome this issue use of extra bypass paths has been illustrated in the literature. The proposed extra paths in the network that act as bypass paths in an event of state change in the network. This type of rearrangeable network is called a repackable network. This thesis investigates the design of large scale repackable networks that require minimum hardware cost when compared to the existing proposals in literature.

Suboptimal algorithms for symmetric rearrangeable networks suffer from high blocking probabilities as the network size increases. At the same time optimal algorithms suffer from being less scalable as they require complex computation that increases with the size of the network. There is a need for an algorithm that can provide low blocking probability with low computational complexity. This thesis investigates the possibilities of constructing a feasible algorithm that meets these criteria and scales better than existing proposals.

Many optimal or deterministic routing algorithms reported in the literature are designed for full permutation, in other words, all inputs request an output. Also algorithms designed for 2×2 switching elements cannot be scaled to work for other symmetric rearrangeable networks built with larger switching elements. This thesis studies a new routing algorithm that can address these issues. Further, an investigation is focused to determine the possibilities of designing symmetric rearrangeable networks built with larger switching elements.

1.2 Thesis Contribution

1.2.1 Problem Statement

The focus of this thesis can be categorized in two broad classes:

Architecture: The contribution on the architectural domain of the rearrangeable networks can be categorized into two subclasses:

- To study the large scale repackable topology and the design of a minimum cost repackable network;
- To design new symmetric rearrangeable networks built with switching elements having more than two input output ports and without compromising the zero blocking probability.

Routing: Contribution on the routing algorithm domain of the rearrangeable networks can be categorized into two subclasses:

- To design a new suboptimal routing algorithm that can provide lower blocking probabilities and faster execution time than that of the other deterministic routing methods;
- To design a new deterministic parallel routing algorithm that works for partial permutations both in electrical and optical domain.

1.2.2 Solutions

A repackable network offers performance potentially indistinguishable from that of a strictly non blocking network, in that a free path through the repackable network is always available, provided that sufficient time has elapsed since the last connection request for the network to the repacked (i.e. to be reconfigured so that a free path is available between any pair of idle inlet and outlet). Therefore, as a solution extra bypass paths have been used to build these network using rearrangeable networks as base networks. The minimum hardware cost has been achieved by using the minimum number of bypass links in the innermost stage of these networks.

The purpose of designing a suboptimal routing algorithm is to reduce the required computational processing overhead. The processing required to implement existing routing methods limits their scalability. Thus, the new method proposed herein considers good scalability properties and can route a partial permutation.

A new deterministic algorithm is designed in this thesis that works for partial permutations. Most available algorithms only work when all the input-output pairs are active, or when dummy requests are set for inactive input-output pairs. The algorithm proposed in this thesis successfully determines the routing paths without the need for dummy requests to fill out partial permutations. A modification of the new method is carried out to make it work in the parallel domain. With ever increasing demand of optical domain communications, routing algorithms need to be feasible to map into the optical domain. The new method also works in generating semi permutation for planar optical rearrangeable networks. With only minor modifications this method generates semi permutations that ensures crosstalk free routing in optical domain.

The modification of rearrangeable network topology addresses the issue of reducing network depth by using larger switching elements and at the same time realising all possible permutations and also reduces the required hardware cost. The routing decision on networks using these larger switching elements is always very challenging, as binary decision making is no longer valid. The deterministic algorithm proposed in this thesis has been modified for routing on these networks.

1.2.3 Methodology

The solutions to the problems stated above have been validated by simulation for various sizes of network. In the simulator the traffic load is uniformly distributed across the network inputs and outputs, for various loads of occupancy of the network. The simulation results obtained have been compared with those of other routing algorithms. All the simulators have been tested in an Intel(R) Core(TM) 2 Quad 2.40 GHz CPU computer with a memory of 8GB.

Hardware complexity is an important issue in deploying any switching system. Crosspoint count has been used as a measurement of hardware complexity in this thesis. Hence the relative cost of using various switching element sizes has been investigated.

1.3 Summary of Contributions

The contributions presented in this research are listed below:

- I did a detailed study of symmetric rearrangeable networks in this thesis. Various design proposals have been studied and are shown to be derived from Beneš based networks. A literature review has been presented on routing algorithms for symmetric rearrangeable network using both serial and parallel processing domains. The reason for choosing rearrangeable networks as a topic for this research has also being discussed.

- A new repackable topology has been proposed in this thesis. The new proposal has a reduced hardware cost compared to existing proposals reported in the literature.

- A suboptimal routing algorithm, termed the hybrid routing algorithm has been proposed. This new method achieves quick routing by trading off blocking probability against execution time. Simulation studies have been carried out to compare its performance with that of other similar methods.

The proposed new method gives a faster execution time than the looping algorithm and a better blocking performance than random routing. Mathematical complexity analysis shows that the overall complexity of the routing algorithm is $O(N \log N)$ using a uni-processor system, where N is the number of inputs. The method does not fully support parallel implementation, but work in two inner subnetworks can be divided among four processors to reduce the execution time of the algorithm.

- A new deterministic routing algorithm has been proposed in this thesis. This method is designed to work for partial permutations without the need for dummy requests. It is always desirable to minimise the required mathematical processing associated with any algorithm. The network stages are abstracted as a set of sub-matrices to generate the conflict free routing tags. This method has better execution time than that of other existing methods. The time complexity for routing a partial permutation ($\overline{m} < N$), where $\overline{m}$ is the number of connections to route, has a better upper bound than other similar methods. Also when routing a full permutation its time complexity matches that of other proposals reported in the literature. This proposed algorithm also works in optical Beneš networks having planar topologies with necessary modifications.

- Rearrangeable networks built with large switching elements have been designed in this thesis. In the network depth is reduced without compromising the zero blocking probability by using switching elements of larger size than the 2×2. A new routing algorithm for this modified class of networks has been proposed in this thesis as binary decision making is no longer valid for these networks.

1.4 Outline of the Thesis

This thesis is organised as follows:

Chapter 2 An overview of available switching methods is presented, followed by a brief description of various transmission techniques. This chapter describes general network structures used in communication. Two broad classes of networks are highlighted. One is the class of self routing blocking networks and the other is the class of nonblocking networks. Three different types of nonblocking networks are also been discussed in this chapter .

Chapter 3 Details of rearrangeable networks are given in this chapter, with a focus on popular rearrangeable networks. Different constructions of Clos networks are presented. This chapter also describes 2×2 symmetric rearrangeable networks, also known as Beneš networks. A detailed study of networks derived from Beneš networks are presented with their permutation realisability characteristics. Overviews of Beneš network routing algorithms, both in serial and parallel domain, are discussed in this chapter. A discussion is given showing the time complexity of the algorithms proposed in the literature for this class of networks.

Chapter 4 Repackable topologies which allow rearrangement without disturbing existing communication links, are studied in this chapter. A new repackable structure is designed that requires fewer crosspoints than that of the other proposals reported in the literature.

Chapter 5 This chapter presents a new routing algorithm termed as, hybrid routing algorithm for rearrangeable networks. Its performance has been compared to that of existing algorithms using simulation results. A mathematical complexity analysis has been executed. The analysis shows that, even in the worst case situation, the hybrid routing algorithm is bounded by the upper limit reported in the literature. The simulation results show that this method has better performance metric than deterministic and other suboptimal algorithms, hence making it a better choice for SoC and NoC applications.

Chapter 6 A new routing algorithm with zero probability of blocking has been designed in this chapter. This method uses a matrix based abstraction for generating the routing tags, rather than using a complex mathematical model to determine the switching element settings. It has been shown in this chapter that the new algorithm is capable of routing requests for partial permutations. A complexity analysis shows that for serial implementation this algorithm matches the scalability in time of the state of the art. For parallel implementation and partial permutations this algorithm has complexities lower than that of other comparable methods. The designed algorithm in this chapter also works in the optical domain after minor modification. This chapter also addresses the issue of designing a symmetric rearrangeable network with larger switching elements. The algorithm proposed for networks built with 2×2 switching elements has been extended to work for modified such symmetric rearrangeable networks.

Chapter 7 This chapter summarises the overall thesis and explores suggestions for possible future work arising from this research.

CHAPTER 2

SWITCHING TECHNIQUES

Switching and multiplexing techniques play a vital role for the effective utilization of the communication channels. Multiplexing techniques are divided into two categories: Frequency Division Multiplexing (FDM) and Time Division Multiplexing (TDM). Similarly, switching techniques are divided into two broad classes: Packet Switching and Circuit Switching. This chapter will provide an overview of various multiplexing techniques followed by common switching techniques that are in current use in communications systems.

2.1 Multiplexing

Multiplexing is the method of sharing a common medium by two or more communication channels. This makes better use of the available bandwidth of the common medium, that to dedicate that medium for a single channel. In communication systems, transmission mediums such as fibre, coaxial or microwave have been shared by multiple channels for transmitting data simultaneously [2]. Fig. 2.1 illustrates the principle of multiplexing.

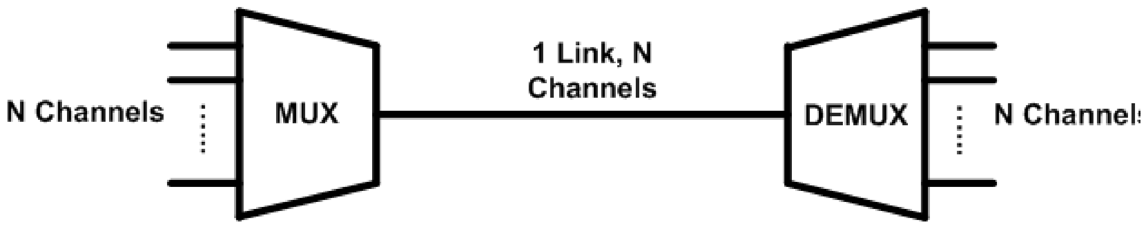


Figure 2.1: *General multiplexing technique.*

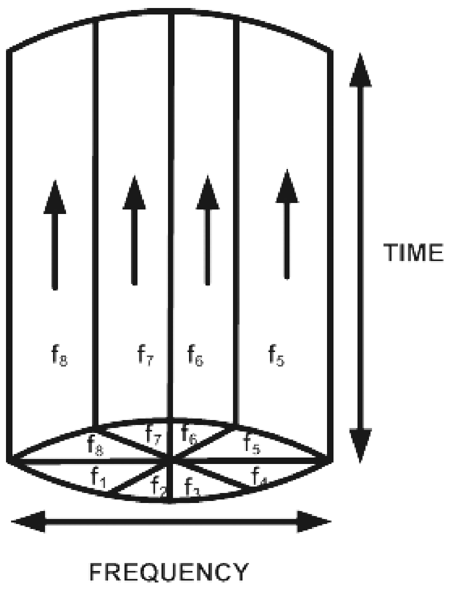


Figure 2.2: *Frequency division multiplexing.*

2.1.1 Frequency Division Multiplexing

Frequency division multiplexing (FDM) [3–5] is the oldest form of multiplexing technique for communication systems. Utilization of the entire bandwidth of communication link is carried out in this technique by modulating multiple signals into different career frequencies. In that way more than one signal(each with a bandwidth less than the link bandwidth) can be transmitted at the same time using a single communication link. This multiplexing works in the analog domain even if the individual signals are digital. An example of an FDM system is broadcast FM radio. Each FM station receives its own frequency band for transmission within the VHF FM band. There are multiple FM channels that broadcast their transmissions at the same time without interference. Fig. 2.2 illustrates the principle of dividing total bandwidth F among multiple signals $f_1 \dots f_n$.

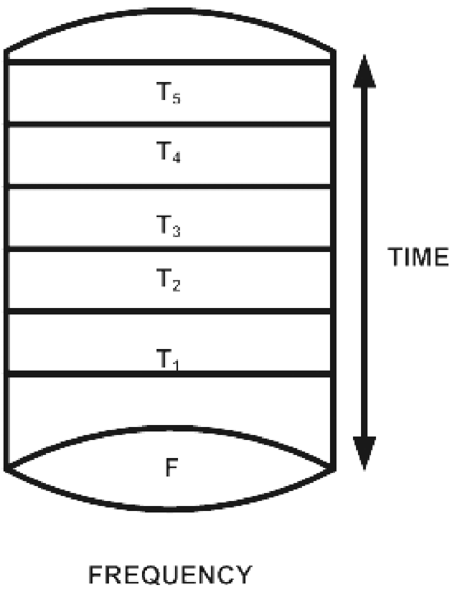


Figure 2.3: *Time division multiplexing.*

2.1.2 Time Division Multiplexing

Time division multiplexing (TDM) [6, 7] allows each signal to use the entire bandwidth of the transmission link for a short duration. In this way, TDM allows all the signals to use the entire channel capacity in turn. There are two kinds of TDM commonly used: Synchronous Time Division Multiplexing and Statistical Time Division Multiplexing. In a synchronous system, the system can be under used because the connected sources are active most of the time but they are not transmitting any data. Statistical TDM dynamically allocates the transmission link on demand. Statistical TDM can make better use of the transmission link than synchronous TDM, since bandwidth is not reserved for possible idle sources. Statistical TDM is associated with packet switching as discussed in 2.2.2. Fig. 2.3 illustrates a synchronous TDM link where time slots $T_1 \ldots T_n$ use the full bandwidth in turn.

2.1.3 Wavelength Division Multiplexing

Communication in the optical domain offers increased bandwidth. The equivalent of FDM in the optical domain is termed as wavelength division multiplexing

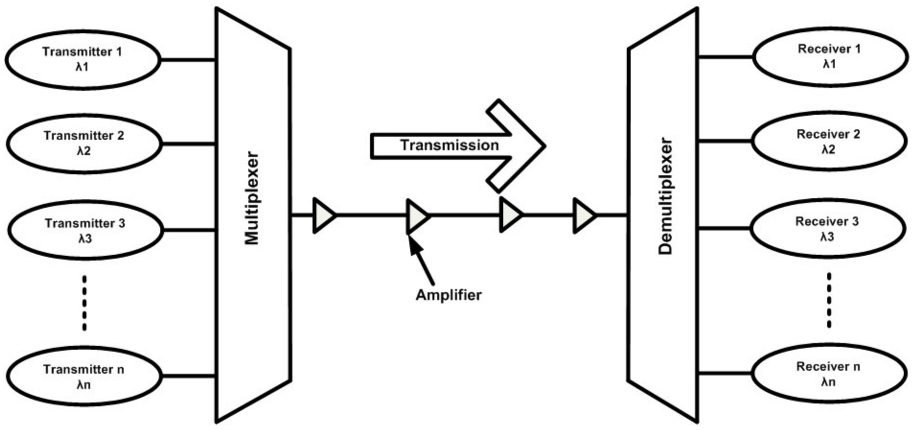


Figure 2.4: *(Dense)Wavelength division multiplexing.*

(WDM) [8, 9]. In WDM transmission , multiple optical rays of different wavelengths are transmitted through an optical link [10–12]. It was first demonstrated in Bell Labs in 1997 [13]. Their range of the transmitted wavelengths are in the nanometer range. This method creates a number of virtual fibres inside a single fibre, which can transmit light of different wavelengths.

With the growth of new high bandwidth consumer applications and the increasing demand in the use of Internet over the last decade, standard WDM dose not provide enough capacities. Current communication demands requires the use of Dense Wavelength division Multiplexing (DWDM) [14, 15]. DWDM allows more virtual optical fibres inside a single optical fiber than standard WDM, leaving small gaps between the virtual paths [16]. A common DWDM system uses two fibers, one as a transmission link and other one as a receiving link with amplifiers in between those cables. Fig. 2.4 shows the general block diagram of a (D)WDM system where $\lambda_1 \ldots \lambda_n$ are signals of different wavelengths.

2.2 Switching

To transmit data from source to destination, an effective and efficient switching mechanism is required [17]. There are two broad classes of switching: Circuit Switching [18, 19] and Packet Switching [20–22].

2.2.1 Circuit Switching

The basic concept of circuit switching [23, 24] is the establishment of dedicated communication paths from source to destination or in the reverse direction. The conventional telephone system is a good example of circuit switching. Once the calling party is connected to the called party, a circuit is established between the two parties. Until one of the parties hangs up , the circuit is occupied by that call. Fig. 2.5 shows the general concept of circuit switching. Circuit switching allows each channel to use fixed and dedicated bandwidth for the duration to the connection. A circuit connection has three phases:

Circuit Establishment: Before starting actual communication between parties, an end-to-end circuit needs to be established. This process checks the availability of a suitable path as well as the status of the calling party. If any one of these two is unavailable or busy, the circuit cannot be established.

Transmission of Data: After the establishment of the connecting path between the calling and called parties, it is possible to exchange information between the two parties.

Circuit Disconnect: Once the period of information exchange has ended, the connection is terminated, so that other parties can use the released resources.

Circuit switching can be of two different types: Space Division Switching [25–27] and Time Division Switching [17, 28]. Space division switching basically means establishing the actual physical paths between source and destination. This is achieved by setting collection of crosspoints in active state which are connected by connecting links. These crosspoints remain dedicated for the communication till the duration of the communication. Fig. 2.6 shows space division switching inside a switching Network.

Time division switching (TDS) brakes incoming low speed signals into pieces and creates a frame combining multiple pieces from the incoming singals and

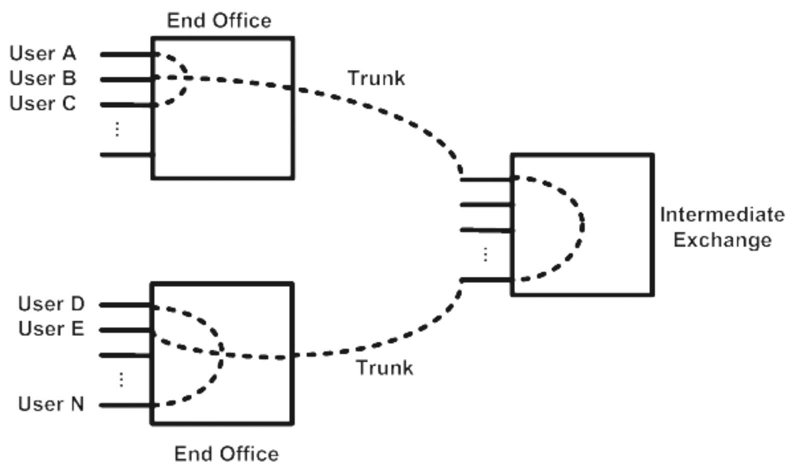


Figure 2.5: *Circuit switching.*

transmits through a high speed transmission link. Time slots and Time Slot Interchanger (TSI) [29, 30] are associated with TDS. Time slots makeup transmitted frames, where each frame can have N time slots. Each slot is dedicated to each transmitting links, which are transmitting at a bit rate for example b. So a shared link capable of supporting a bit rate of N can have a frame of N/b time slots. Information will occupy different time slots in the incoming and outgoing frames. This needs change in the time slots and the process is called Time Slot Interchange (TSI). TSI works like a buffer containing m time slots where input information are stored sequentially. Output frames are constructed by reading the buffer, where the values in the input buffer construct that output time slots. TDS is performed using TSI, where TDM creates the frame stored in the TSI buffer. TSI then interchanges the time slots according to the given permutation. Circuit switching in the optical domain establishes dedicated light paths between communicating optical devices [31, 32], which can carry multiple large bandwidth connections using DWDM.

2.2.2 Packet Switching

Circuit switching systems were mainly used for long distance telephone systems. This allows two connected parties to communicate information without any in-

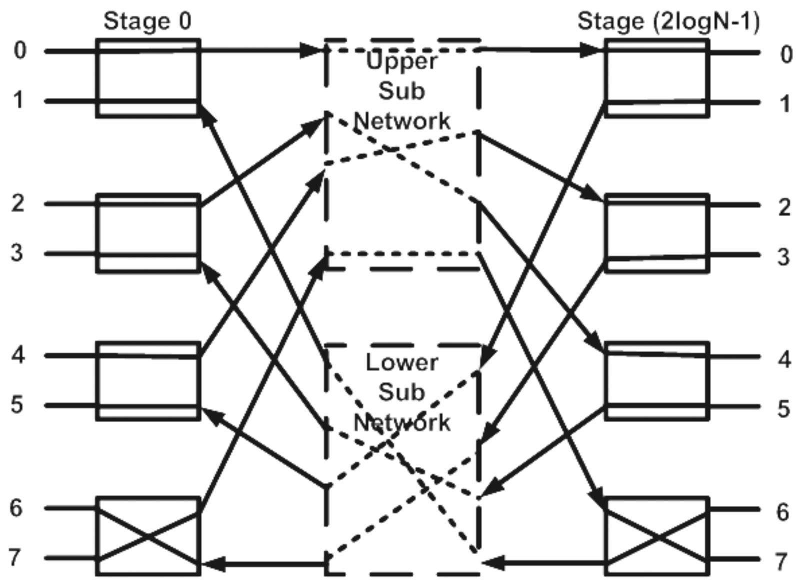


Figure 2.6: *Space division switching.*

terruption with a fixed bandwidth. But these systems are inefficient for transmitting data, because a dedicated path for transmitting data keeps the path unused some of the time.

Packet switching [33] allows long messages to be broken into small pieces called packets which are transmitted through the communication networks. Individual packets belonging to the same message might take different paths to reach the same destination. A packet contains a header with different fields inside it, such as source addresses and destination address, as well as the payload or actual information. Fig. 2.7 shows the principle of packet switching. In this figure, station A sends a message DATA to station B, but in the actual communication network this message is broken into four packets to be transmitted to station B. Station B rearranges the received packets to reconstruct the original message.

Packet switching has two different switching methods: Virtual Circuit routing [34–36] and Datagram routing [36–38]. A virtual circuit network is like packet-oriented circuit switching. Similarly to circuit switching, a virtual circuit once established, produces a dedicated logical path for transmitting packets from source to destination. Each virtual circuit has its own identifier and each packet has a virtual circuit identifier associated with it. Fig. 2.8 a illustrates virtual circuit.

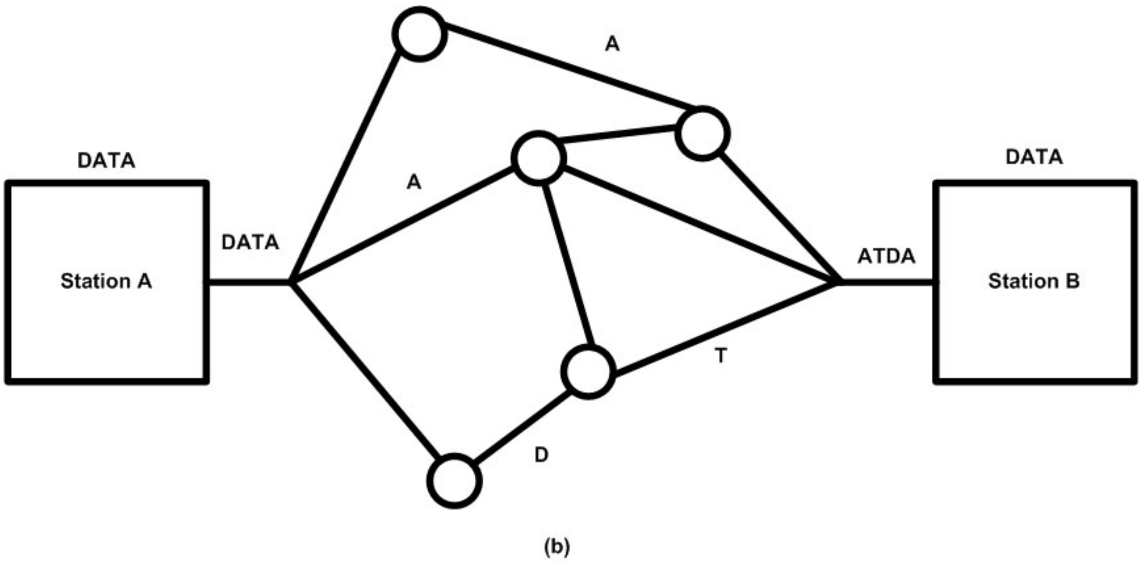


Figure 2.7: *Packet switching.*

In the datagram routing each packet belonging to the original message may get routed through a different path from the source to the destination since they are routed independently. Fig. 2.9 illustrate datagram routing.

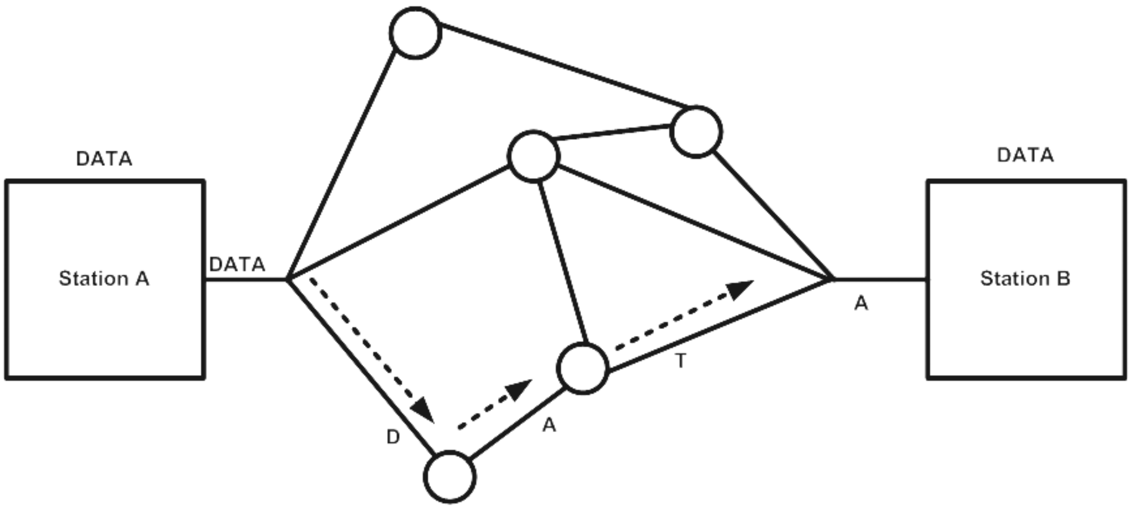


Figure 2.8: *Virtual circuit routing.*

2.2.3 Multi-Rate Circuit Switching

As explained Section 4.1, circuit switching can support a fixed data rate for all channels. Modified form of circuit switching communication is the multi-rate cir-

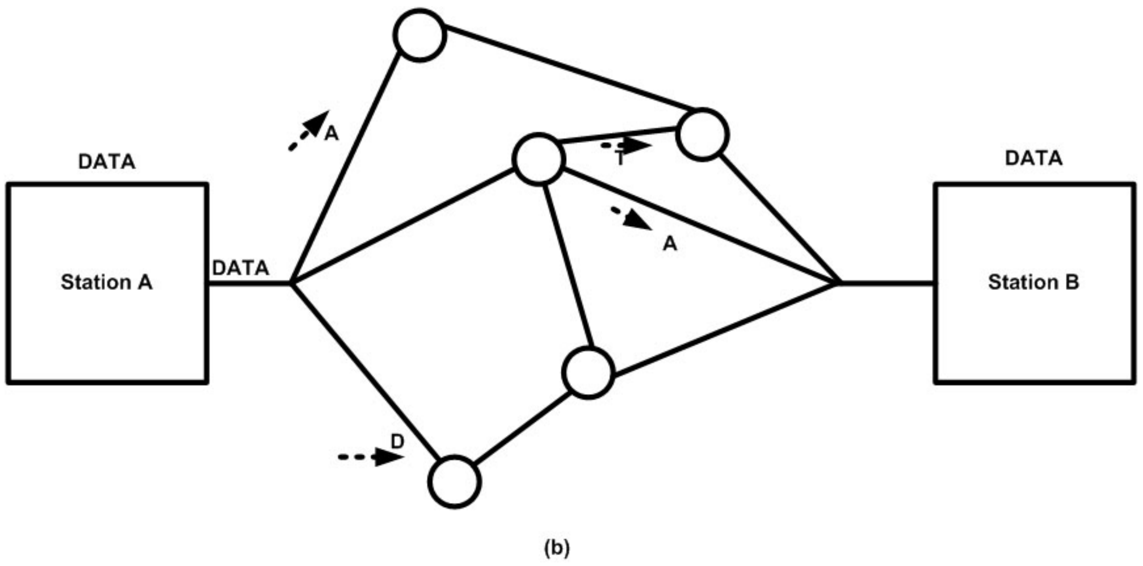


Figure 2.9: *Datagram routing.*

cuit switching [39], which allows transmission of signal as a multiple of a fixed data rate. The efficiency of this mechanism depends on the choice of base data rate. The downside of this method is that it might not provide similar performance for high or low data rates signaling. Intergraded Service Digital Networks(ISDN) [40] uses base bit rate of $64Kb/s$ for transmission of signals.

2.2.4 Virtual Cut-Through Switching

In packet switching method entire packet is stored in the node buffer before been transferred to the next node. For a long packet, some times it is not possible to transfer the entire packet in one cycle because of the bandwidth of the channel. As a result it requires multiple cycle to transfer the entire packet. Lets assume three nodes A, B and C involved in the transmission process. Where A is the source node and C is the destination node. When node A starts transmitting a packet to node C through intermediate node B, B must receive the entire packet from A before it can send it to C. This process requires the buffering time even if the transmitting channel for free. To overcome this delay, transmission should start as soon as header of the packet is received in node B and decisions have been made. This process followed by transmitting the data part of the packet.

This switching method is called virtual cut-through switching [41, 42]. Without any channel blocking, the delay experienced by the header in this method is the switching and transmission delay. Rest of the packet is transmitted in a pipelined fashion.

2.2.5 Wormhole Switching

Virtual cut-through switching required buffers to store the packets in an event of channel blocking. Requirements of large buffers are eliminated in wormhole switching [43–45]. In wormhole switching, messages packets are broken in to small pieces called flits (flow control digits)and the buffers at a the routers are large enough to store few flits. Similar to the virtual cut-through switching , packets are also pipelined in flits level in wormhole switching. In an event of channel blocking, message flits are stored in several router's buffer. In wormhole switching, header flits contains all the necessary switching information and all the data flits follows the header flit to reach the destination. Difference between virtual cut-through and wormhole switching is that for virtual cut-through unit of message flow is in packets, but for wormhole it is in flits.

2.2.6 Optical Burst Switching

Conventional routing struggles to cope with the increasing demands of Internet traffic. The introduction of optical circuit switching with the help of WDM allows multiple optical paths to be established between communication parties hence giving almost unlimited transmission bandwidth [46, 47]. However, alternative to the optical circuit switching is the optical packet switching, which allows packet based transmission of data in the optical domain. But optical packet switching is not commercially viable. A new switching method has been proposed which is optical burst switching [48–50]. In this method of switching, different incoming data streams are aggregated and then transmitted across the op-

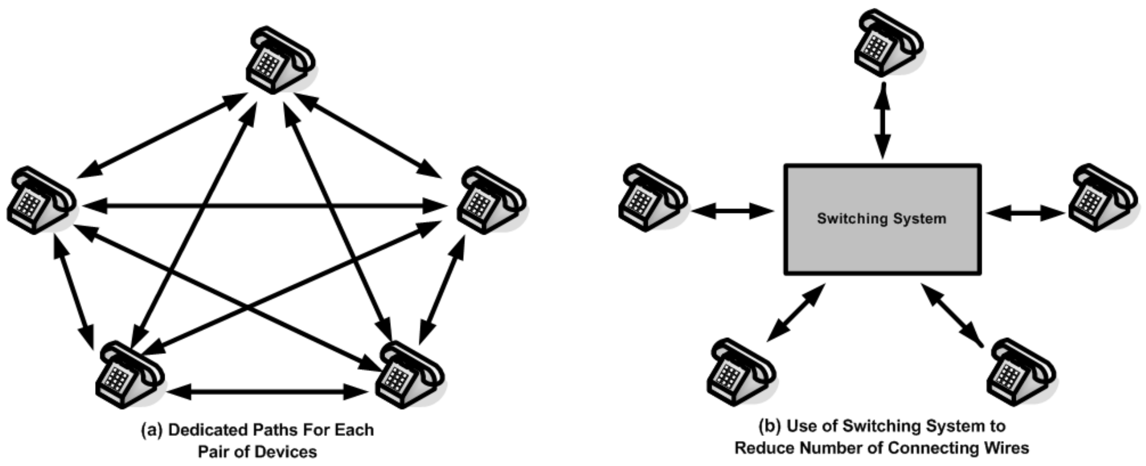

(a) Dedicated Paths For Each Pair of Devices
Switching System
(b) Use of Switching System to Reduce Number of Connecting Wires

Figure 2.10: *Communication with and without switching system.*

tical network as a burst of data. At the receiving end the burst gets separated in to the individual data streams for delivery to their respective destinations. To save the optical-electrical-optical switching cost, control signals are transmitted in a separate wavelength compare to the corresponding burst data. This allows the data burst to be processed entirely in optical domain, but without the technical challenges and cost of packet switching at optical routes.

2.3 Communication Networks

The easiest way to connect two communicating parties is to have a dedicated communication link between the two parties. This is a simple solution as long as the number of communicating parties is limited. With a total of N communicating devices, a total of $N(N-1)/2$ line is require using this dedicated method , which is not very efficient. Fig. 2.10(a) shows such a directly connected network. Avoiding this huge number of connecting links requires the use of switching systems to connect the devices.

A switching system provides following functionalities:

Signalling: The signaling function monitors the activities on the incoming lines and passes the appropriate control information to the control unit.

Control: The control section processes incoming information to set up paths appropriately.

Switching: The switching function provides the switching matrix for establishing a communicating path. The switching section is made of sequential arrangement of rows and columns with basic building blocks, called Switching Elements(SEs). The connection patterns between these switching elements define the interconnecting networks.

2.3.1 Interconnection Networks

The topologies, operation modes, control strategies and switching methodologies of interconnection networks have been surveyed extensively in [51–55]. Telecommunication motivated much of the initial research in interconnection networks. An increasing need for processing power has led to interconnection networks being also found in high performance computing systems [56]. Concurrent processing of data is a priority requirement for monitoring real-time events, such as weather system, where gigaflop (10^9 floating point operations/second) was the speed requirement, interconnection networks were the obvious choice. Fig 2.11 shows a basic model of concurrent processing using interconnecting networks, with $P_1 \dots P_N$ processors and $M_1 \dots M_M$ memory modules.

The following sections provide a general overview of interconnection networks used in communication systems. In this thesis the left side of the switch is consider as the input side and the right side as the output. A practical switch is usually symmetric and bidirectional. The latter property is often achieved by use of adjacent unidirectional networks. A switch will be consider to be unidirectional in this thesis unless otherwise stated.

2.3.1.1 Classification of Interconnection networks

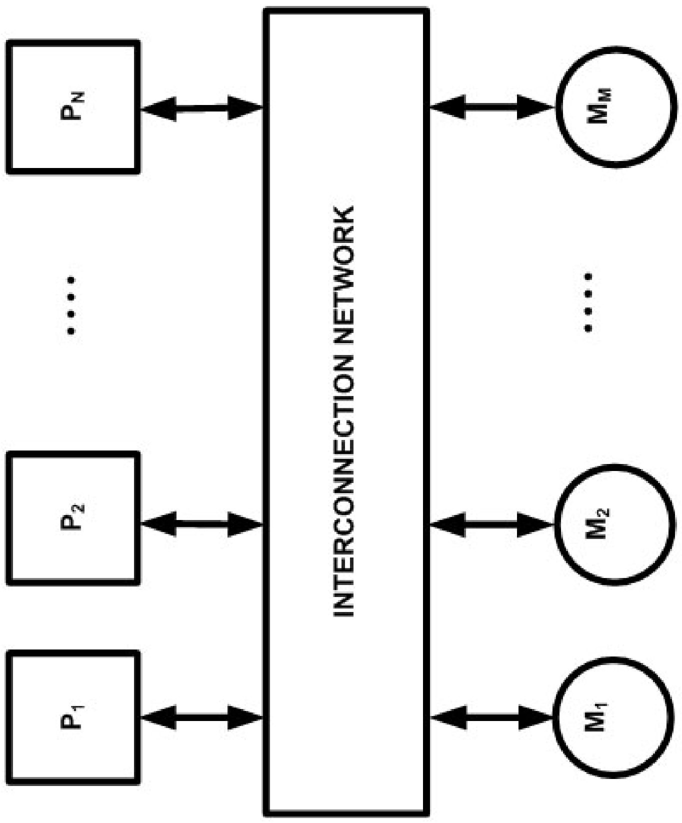


Figure 2.11: *Processor memory communication using an interconnection network.*

The topological description is an important specification of an interconnection network as defined below:

Dedicated-path networks regular static networks with dedicated paths for each processing elements which cannot be changed are called dedicated-path networks. Fig 2.12 shows examples of dedicated-path networks. Even though there are dedicated paths for each processing module, these networks are less scalable and they have different memory access time because of physical separation between processing elements.

Dynamic Networks networks where paths can be changed dynamically by configuring the network switching elements. Dynamic networks which have the capabilities of establishing communicating paths between any two processing elements, have three different classes according to their switching stages.

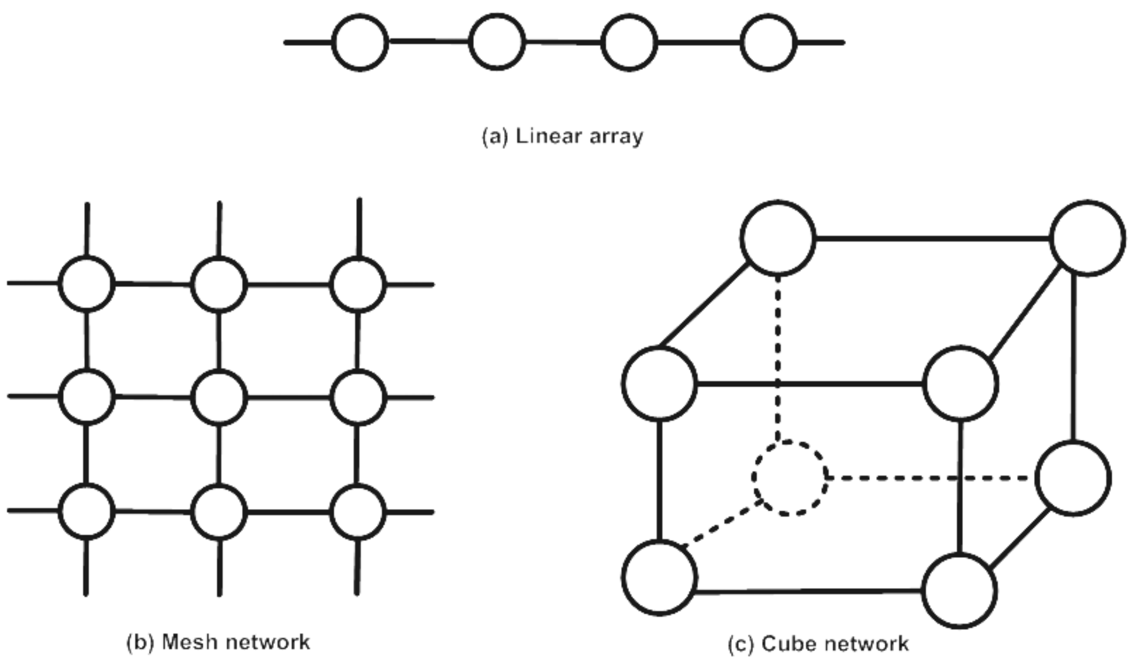


Figure 2.12: *dedicated-path networks.*

Single stage networks where data before reaching the destination recirculates inside the network to find the appropriate destination. For a general point of view, a single stage network contains N input selectors to go with N output selectors.

Crossbar these networks allow processing elements to communicate simultaneously without the need of recirculation [54]. In this network input-output paths are established by activating appropriate crosspoints in the network that connect requesting parties. Fig 2.13 shows single stage and crossbar network structure. On the other hand, networks having stages of switching elements connected by connecting links between stages are called

Multistage Interconnection Networks where the switching elements settings determine which paths a request will follow to reach a desired output. Details of multi stage interconnection networks are given in the next section.

2.3.1.2 Multistage Interconnection Networks

Multistage interconnection networks(MINs) have the following characteristics:

- Switching elements (SEs) are arranged in stages
- Generally numbered from $0, 1, 2 \ldots,$ from left to right
- Switching element outputs in $stage_i$ are connected to switching element inputs $stage_j$ with a fixed permutations, where $i > j$ (typically $j = i + 1$)

In this thesis all the log mentioned are $base\ 2$ unless otherwise stated and $n = \log N$ and N is the total number of inputs. The notations used in to describe interconnection networks are generalized form of the patterns used in [57]. Multistage interconnection networks can be either blocking or nonblocking [53, 58]. Blocking networks cannot establish paths for all input-output requests, whereas nonblocking networks will always find a path for any valid request. Banyan networks [59] have been proposed for distributing loads among processors. Omega networks [60] are an example of banyan networks. They have a shuffle-exchange [61] link pattern between adjacent stages. Omega network has self-routing capabilities. A general view of a self routing networks is given in Fig 2.14, where the routing is controlled by the routing tags which are the binary bits of the outputs requested by inputs. Data coming to the network is transmitted serially with a prifix of an n-bit representation of the requested output $(b_{n-1}\ b_{n-2}\ \ldots\ b_1\ b_0)$ and an activity bit.

2.3.2 Networks Overview

This section provides a brief description of some common networks built using 2×2 switching elements used as connecting network between communication parties. These networks have $(n - 1)$ stages of switching elements and n stages of link patterns. Each link is represented an n bit binary representation of their

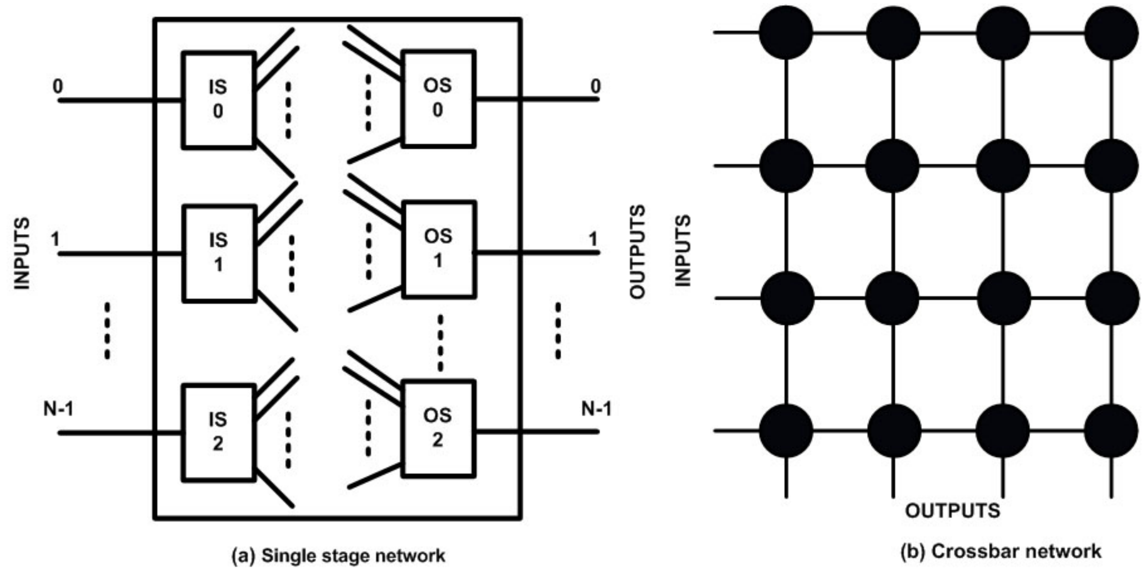


Figure 2.13: *Single stage and multi stage networks.*

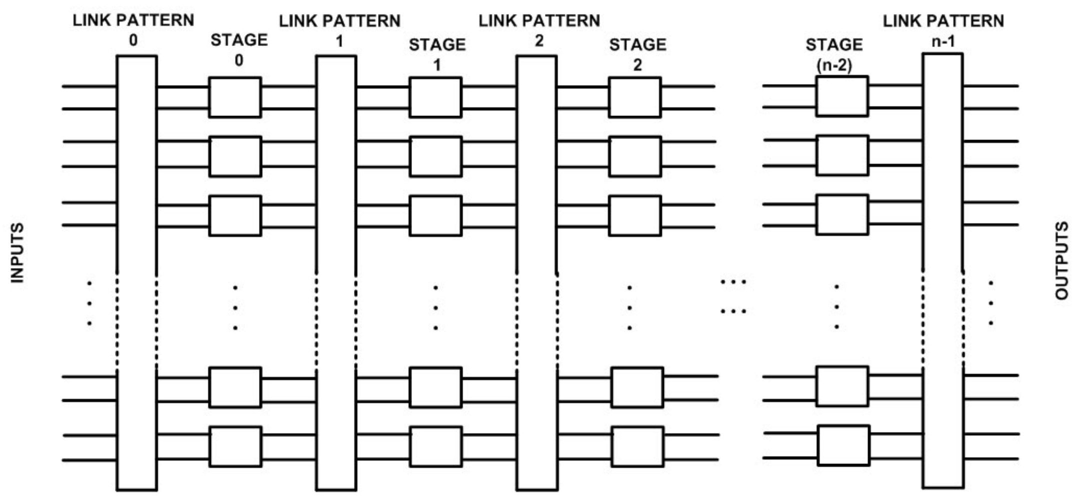


Figure 2.14: *General view of a self routing network.*

decimal value from top to bottom, where bit 0 is the least significant bit (LSB) and bit $(n-1)$ is the most significant bit (MSB). Similarly switching elements are numbered with $(n-1)$ bits with bit 0 being the LSB and bit $(n-2)$ being the MSB. The overall switch is a $2^n \times 2^n$ network.

Baseline Networks [62] The have a straight through pattern at stage 0 and stage n. The baseline networks uses inverse perfect shuffle link patterns [61]. The inverse perfect shuffle is cyclic shift right operation of least significant bit of the input port. Fig 2.15 (a) shows an 8×8 baseline network. The link patterns between switching stages can be given by the following equation where $0 \leq k \leq (n-2)$ and $l = (n-1)$:

$$\beta_k(b_{n-1}\, b_{n-2} \dots b_0) = (b_l\ \dots\ b_{l-k-1}\, b_0\, b_{l-k}\ \dots\ b_1\ ) \tag{2.1}$$

Reverse Baseline Networks [62] have the same number of inputs and outputs as the baseline networks. These networks have a perfect shuffle connection pattern between adjacent switching stages. The perfect shuffle is a cyclic left operation on the binary representation of the input port numbers. Fig 2.15 (b) shows an 8×8 reverse baseline network. The link patterns are given by the following equation where $0 \leq k \leq (n-2)$ and $l = (n-1)$:

$$\beta_k^{-1}(b_{n-1}\, b_{n-2} \dots b_0) = (b_l\ \dots\ b_{k+2}\, b_k\, b_{k-1}\ \dots\ b_0\, b_{k+1}) \tag{2.2}$$

Omega Networks [60] use perfect shuffle link patterns at each stage of the network except the last stage which follows a straight through pattern. Fig 2.15 (c) shows an 8×8 omega network. The link patterns at each switching stage is given by the following equation:

$$\Omega(b_{n-1}\, b_{n-2} \dots b_0) = (b_{n-2}\ \dots\ b_0\, b_{n-1}) \tag{2.3}$$

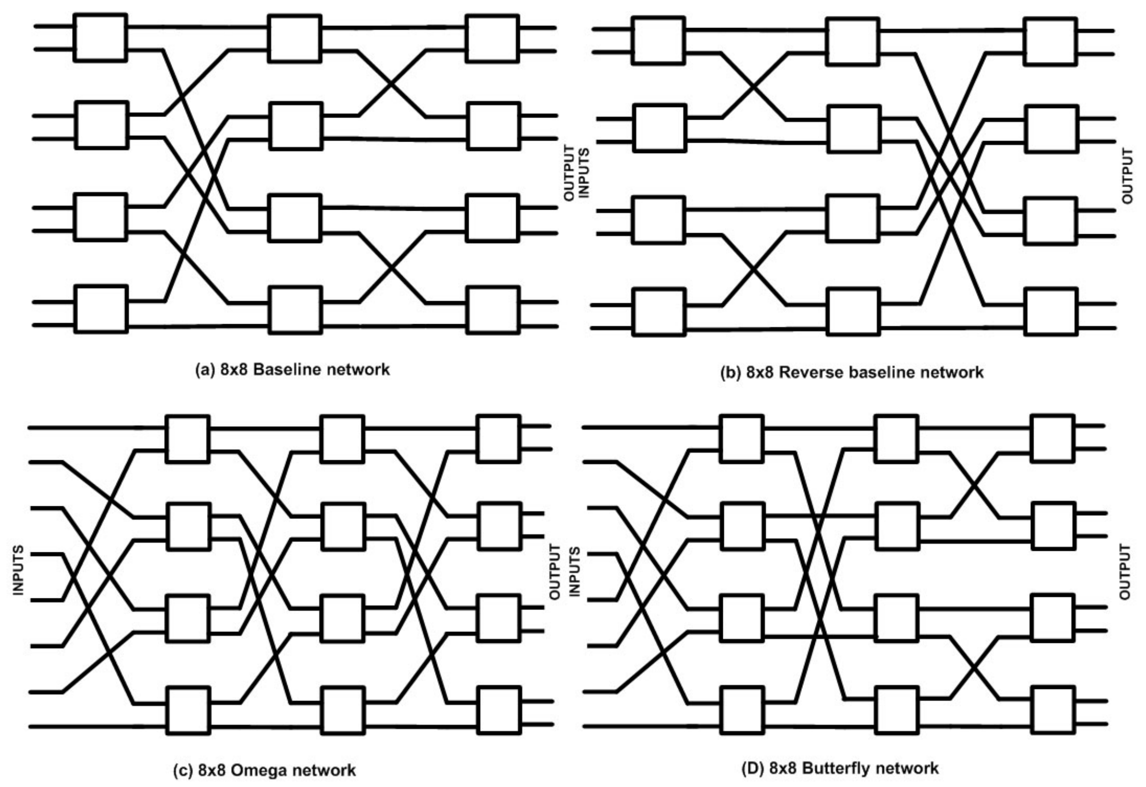


Figure 2.15: *Common networks*

Butterfly Networks [53] have three different link patterns in their construction. Link stage 0 follows inverse perfect shuffle (IPS) pattern, which is a cyclic right shift, from stage 1 to stage $(n-1)$ a swap pattern (SP) and is used the last stage follows a straight through pattern. Fig 2.15 (d) shows an 8×8 butterfly network. The mathematical presentation of the IPS and SP patterns is as follows:

$$IPS_0(b_{n-1}\, b_{n-2} \ldots b_0) = (b_{n-1} \ldots b_{n-2-k} b_0 b_{n-1-k} \ \ldots \ b_1) \tag{2.4}$$

$$SP_i(b_{n-1}\, b_{n-2} \ldots b_0) = (b_{n-1} b_{n-2} \ \ldots \ b_i b_0 b_{i-2}\, b_1 b_{k-1}) \tag{2.5}$$

where $1 \leq i \leq (n-1)$ and $0 \leq k \leq (n-2)$

The networks described above are all blocking networks. In other words these networks cannot establish paths for all input- output requests. These networks have only a single path each output from each input. As a result blocking occurs as multiple request try to use the same link to go to the desired output port.

2.3.2.1 Nonblocking Networks

The networks in the previous section have blocking characteristics. Modifications to make the network non-blocking can be by adding extra paths with switching elements of higher degree [63], adding extra stages to the existing network [64–66] or even adding multiple networks in parallel [67, 68]. Nonblocking interconnection networks [69–71] can be of three different types:

Strict sense nonblocking [72] networks can establish connecting paths for any input-output request. The large implementation cost and the required number of crosspoints increases rapidly with networks size. Let us assume that A be the set of inputs and B be the set of outputs and $a \subseteq A$ and $b \subseteq B$ be the inputs and outputs that are busy. Then for strict sense nonblocking networks, an input from the set $(A - a) \subseteq A$ can connect to an output $(B - b) \subseteq B$, without disturbing the already established connections. The required crosspoints count for this kind of network can be calculated using the following equation [73]:
For a single stage network:

$$C(1) = N^2; \tag{2.6}$$

For a 3 stage network:

$$C(3) = 3N^{\frac{3}{2}} - 3N; \tag{2.7}$$

For a 5 stage network:

$$C(5) = 16N^{\frac{4}{3}} - 14N + N^{\frac{2}{3}}; \tag{2.8}$$

With $k = 2t + 1$ where $t = 0, 1, 2 \ldots$ and k is the number of stages(odd)in the network and $N = n^{t+1}$ is the number of inputs, the general equation for any network can be given as:

$$C(2t + 1) = \frac{n^2(2n - 1)}{n - 1}[(5n - 3)(2n - 1)^{t-1} - 2n^t] \tag{2.9}$$

Wide sense nonblocking networks were proposed by Beneš [73]. A network can be called wide sense nonblocking if using a routing algorithm A, the network is always in a safe state. In other words, the algorithm can establish paths for new requests without rerouting existing connections. Later Smyth [74] proposed method to determine how the safe states can be achieved for a routing algorithm A. According to the proposed method, algorithms for these networks first identify the safe state of the network. If a network has a total of **S** state and there is a set of S' which will result in putting the network into blocking state, the remaining states $(S - S')$ are safe. Wide sense nonblocking networks require fewer of crosspoints than strict sense nonblocking networks , but they need complex algorithms to achieve this performance [71, 75–78].

Rearrangeably nonblocking networks [79–82] can connect every input-output pair. To set the network might need to re-route existing connections in order to establish the new request. One of the most common rearrangeable network is the Beneš [73] network. Because of the interesting path setup characteristic and low hardware complexities, more details on rearrangeable nonblocking networks are discussed in detail in Chapter 3.

2.4 Bidirectional Multistage Interconnection Networks

This section will briefly describe another class of networks called bidirectional networks [19, 83, 84]. These networks are used in areas such as wormhole switching [44]. In a bidirectional network each switching element port has two unidirectional channels for communication in both directions. This helps transmitting simultaneous information in both directions. These networks support three types of communications forward, reverse and turnaround. Lets assume that all the nodes are attached to the left hand side of the network. So in this case, forward communication is the process of going from left to the right of the network. Similarly reverse is coming from right towards the left of the network. And going from left to right and then reversing back from right to left side of the network

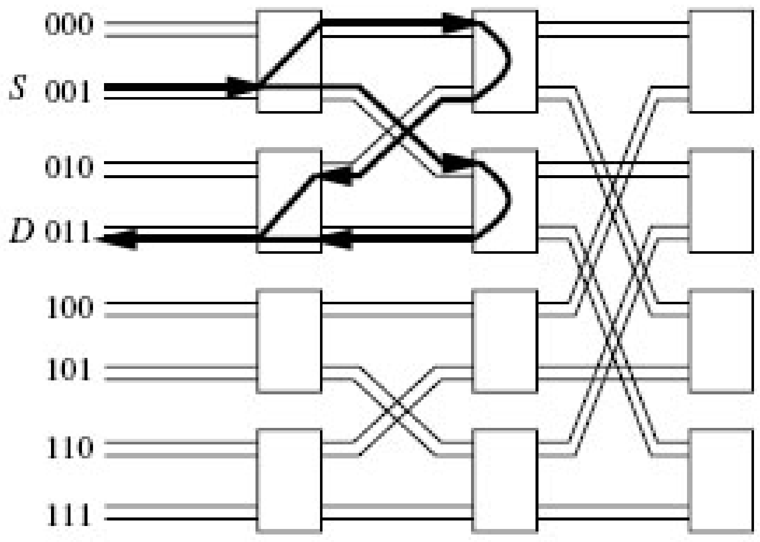


Figure 2.16: *Bidirectional butterfly network*
[84]

together called turnover process. Fig. 2.4 shows a bidirectional butterfly network and routing paths between source S and destination D. In forward direction routing there can be multiple paths similar to the unidirectional networks , but in reverse direction there can be only one path from source to destination. If the requested node is in the other half of the node list, directional networks faces worst case scenarios. In those cases the signal travels $(2\log N - 1)$ stages, which is similar to the Beneš [73] network routing. This is why a bidirectional baseline network can be termed as folded Beneš network.

A bidirectional butterfly network can be terms as a Fat tree [85, 86]. In a fat tree network nodes are located at the leaves, and intermediate node connecting these leaves are switches. As these networks goes towards the root, network bandwidth gets increased as the links associated with nodes at each stage increase. In a binary fat tree for each node at the leave has two unidirectional links in both direction. This trend continues for the switching nodes, as for a switching node with two processors attached to it has four links going out and coming into it. Similar to the bidirectional butterfly network, in worst case signal travels $(2\log N - 1)$ nodes before reaching the destination, which is again similar to the

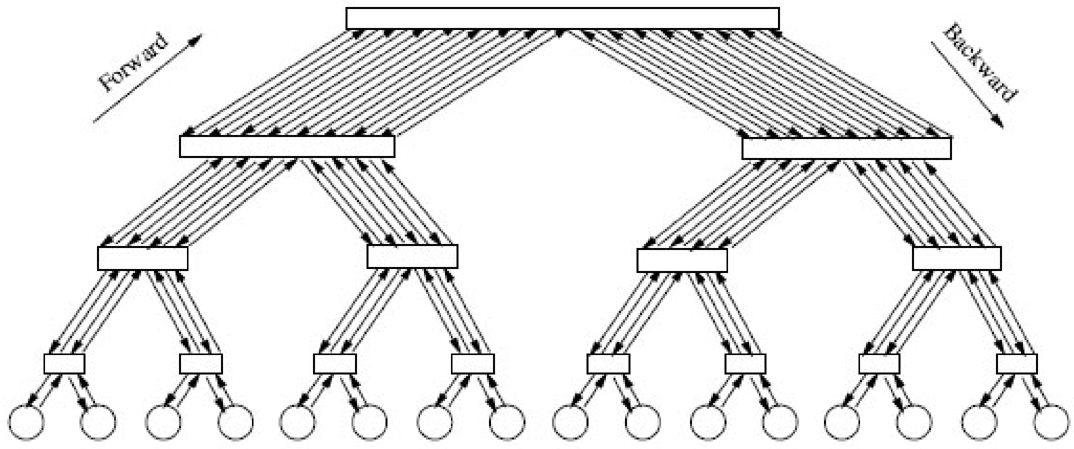


Figure 2.17: *Fat tree with* 16 *nodes*
[84]

Beneš networks. Fig 2.4 shows a binary fat tree.

2.5 Summary

This chapter overviews various multiplexing techniques and switching networks. TDM shows better use of the available bandwidth compare to the FDM. But implementation of TDM requires complex hardware than FDM, as it has to deal with time slots and frames. In optical domain, use of (D)WDM shows considerable improvement on the used bandwidth because of their ability to create multiple fibre environment inside a single fibre. From switch structure point, different multistage switching structure have been discussed. Multistage switching networks have less hardware complexities than compare to crossbar networks. Rearrangeably nonblocking networks have less hardware cost compared to strict nonblocking networks or crossbar. These networks structure show the promise for further investigations because of their low hardware cost and zero blocking performance.

CHAPTER 3

REARRANGEABLE NETWORKS

This chapter contains detailed descriptions of some common rearrangeable nonblocking networks, with an analysis of their strengths and weaknesses.

3.1 Clos Networks

Clos networks [87] are constructed with three switching stages, an input stage, a middle stage and an output stage. A common notation to describe a Clos network is the triple(m, n, r), where n is the number of inputs for each switching element at the input(respectively output) stage, m is the number of outputs for each switching element at input (respectively output) stage and r is the number of switching elements both in the input and output stages. Each input switching element has a connection to each middle stage switching element. Similarly each middle stage switching element has a connection to each output switching element. Fig 3.1 shows a diagram of a Clos network. A network with r switching elements in the input stage, has m, $r \times r$ switching elements in the middle stage.

Clos networks can be of three different types:

- Strict-sense Nonblocking
- Wide Sense Nonblocking
- Rearrangeably Nonblocking

Clos networks where $m \geq (2n-1)$, are strict-sense non blocking networks. Table 3.1 [73] shows the required number of crosspoints for strict-sense nonblocking Clos networks and networks built using crossbar switches. If $m \geq n$ the networks are rearrangeably nonblocking (Slepian-Duguid [88, 89]). They also proved that no more than $(2r-2)$ rearrangements are required to unblock a blocked request. Paull extended the result given by Slepian-Duguid to reduce the upper bound on the total number of rearrangements to $(r-1)$ [90]. Beneš [73]later proved that for $(2, 2, r)$ network, maximum number of required rearrangement is also $(r-1)$.

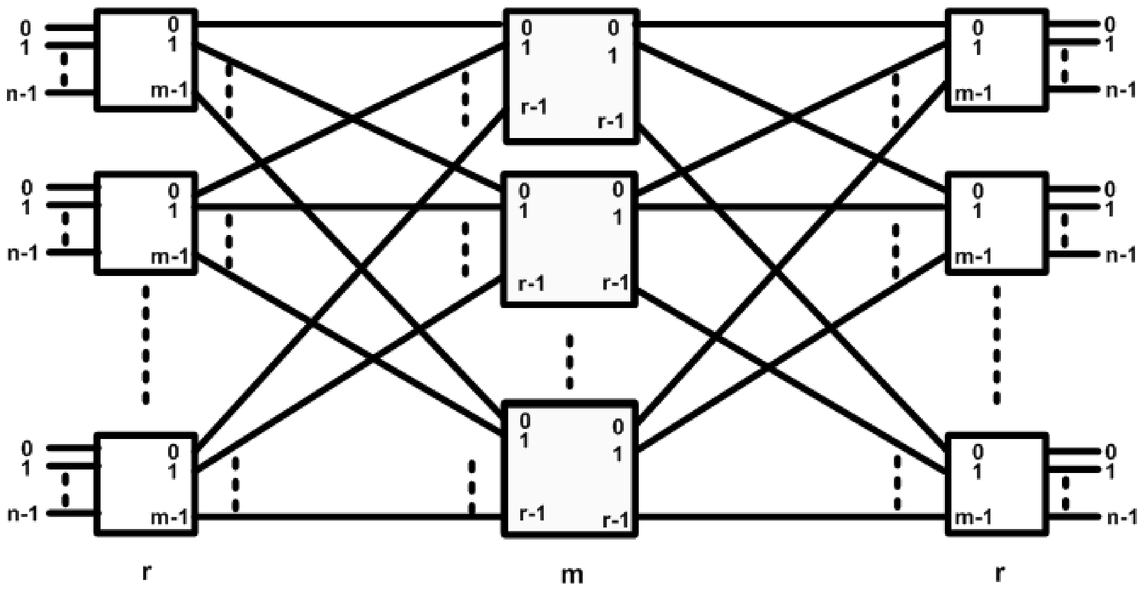


Figure 3.1: *Clos network.*

The proof that the upper bound is $(r-1)$ given in [91] is as follows. A $(2,\ r,\ r)$ switch will have $N = 2r$ inputs and outputs. When establishing a path for a new request, at most there can at most $(2r-1)$ active connections in the network. There is then only one free link from the input stage switching element to a single

Table 3.1: *Number of crosspoints for crossbar and 3-stage Clos networks.*

N	Crossbar Networks	3-stage Clos Networks
4	16	36
9	81	135
16	256	336
25	625	675
36	1,296	1,188
49	2,401	1,911
64	4,096	2,880
81	6,561	4,131
100	10,000	5,700
....	....	....
1,000	1,000,000	186,737
10,000	100,000,000	5,970,000

middle stage switching element as well as one free link from that middle stage switching element to some output switching element. Thus, in this particular state of the network, there is only one middle stage switching element having one free input and output link. This middle stage switching element is subsequently linked to a single input stage switching element and a single output stage switching element with free input or output port respectively. Since these are the only available links at this particular state of the network, so a new request between these two free ports is possible. The network experiences a blocking state when there are already $(2r - 2)$ active requests in the network and a free input is requesting for an available output but no paths are available to route the request from the input stage to the output one. Under such a circumstances, the path for the new request can only be established by rearranging the already established requests.

As mentioned in the blocked state there can be a maximum of $2r - 2$ active requests, and they form two loops. A loop can be defined as a set of paths where connecting paths starting from one input(output) switching element terminates in the same input(output) switching element respectively or in a switching element where terminating paths is the only active link. Let's assume the length of these paths can be P, which contains the number paths in a sequence or in other words length of the loop. Since there are two available inputs in two different input switching elements, so there can only be two loops of links available in this case. Suppose P_1 and P_2 are lengths of two loop and either of them can be rearranged to find path for new request. So:

$$P_1 + P_2 = 2r - 2 \tag{3.1}$$

If the values of P_1 and P_2 are the same:

$$P_1 = P_2 = r - 1 \tag{3.2}$$

If they have different values then it can be assumed that $P_2 > P_1$:

$$P_1 < r - 1 \text{ and } P_2 > r - 1 \tag{3.3}$$

Since P_1 contains fewer paths, it makes sense to rearrange P_1 which is upper bounded by $r-1$. Hence even in the worst case, it is always be possible to unblock a network with at most $(r-1)$ rearrangements.

The comparison of the implementation cost of rearrangeable networks to that of strict-sense nonblocking Clos network can be approximated using Eqn 3.4. Let's assume for a given strict-sense non-blocking Clos network with $m = 2n - 1$. Then the implementation cost $C(n, r)$ for the network is given :

$$\begin{aligned} C(n, r) &= n(2n-1)r + r^2(2n-1) + (2n-1)nr \\ &= 4n^2r - 2nr + 2nr^2 - r^2 \end{aligned} \tag{3.4}$$

Substituting $N = nr$, where N denotes the number of inputs, Eqn 3.4 becomes:

$$C(n, r) = 4Nn - 2N + 2\frac{N^2}{n} - \left(\frac{N}{n}\right)^2 \tag{3.5}$$

Differentiation of Eqn 3.5 with respect to n gives:

$$\frac{d}{dn}C(n,r) = 4N - \frac{2N^2}{n^2} + \frac{2N^2}{n^3} \tag{3.6}$$

To find the value of n, lets simplify Eqn 3.6 as:

$$\begin{aligned} 4N - \frac{2N^2}{n^2} + \frac{2N^2}{n^3} &= 0 \\ \Rightarrow \frac{2N}{n^2} - \frac{2N}{n^3} &= 4 \\ \Rightarrow \frac{1}{n^2}[1 - \frac{1}{n}] &= \tfrac{2}{N} \end{aligned}$$

Which in turn gives Eqn 3.7:

$$n \approx \sqrt{\frac{N}{2}} \tag{3.7}$$

Substituting Eqn 3.7 in Eqn 3.4 gives the approximate minimum number of crosspoints required for an $N \times N$ strict-sense nonblocking Clos network:

$$C(n,r)_{min} = 4\sqrt{2}N^{1.5} - 4N \tag{3.8}$$

For a rearrangeable Clos network the corresponding figure is:

$$C(n,r)_{rea} = 2n^2r + nr^2 \tag{3.9}$$

The overall crosspoint requirement can be reduced considerably by using a recursive topology in building the network. The large middle stage switching elements can be implemented as three stage subnetworks with smaller switching elements. This principle can be applied to the recursive construction of a Clos network as shown in Fig 3.2. This creates a network which is symmetric about the middle stage.

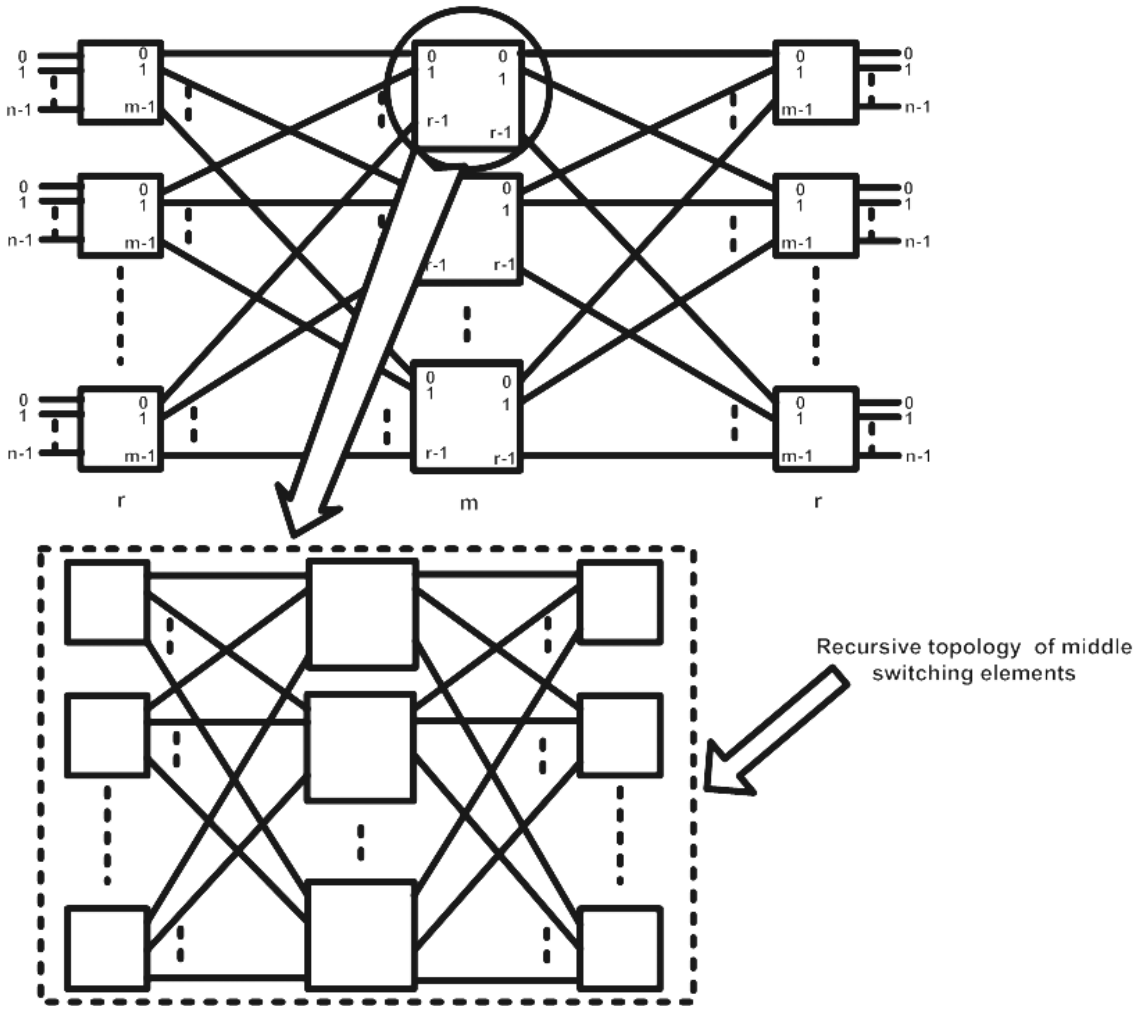


Figure 3.2: *Recursive Clos network.*

3.2 Beneš Network

A Beneš network is a recursively constructed Clos network built with 2×2 switching elements [35, 79, 92]. Waksman showed that it acts as a permutation network [93], which means it can realize all $N!$ possible patterns of input-output requests, with $N = 2^{\log N}$ inputs (outputs) and total stages $k = (2\log N - 1)$ [94–96]. The Beneš network may be regarded as the concatenation baseline network and its reverse network. The concatenation of two $\log N$-stage networks form a $2\log N$-stage network with a straight through link pattern between the two middle stages. Merging the two middle stages gives rise to a $(2\log N - 1)$ stage Beneš network.

Link pattern for the Beneš network described by the following equations [97]

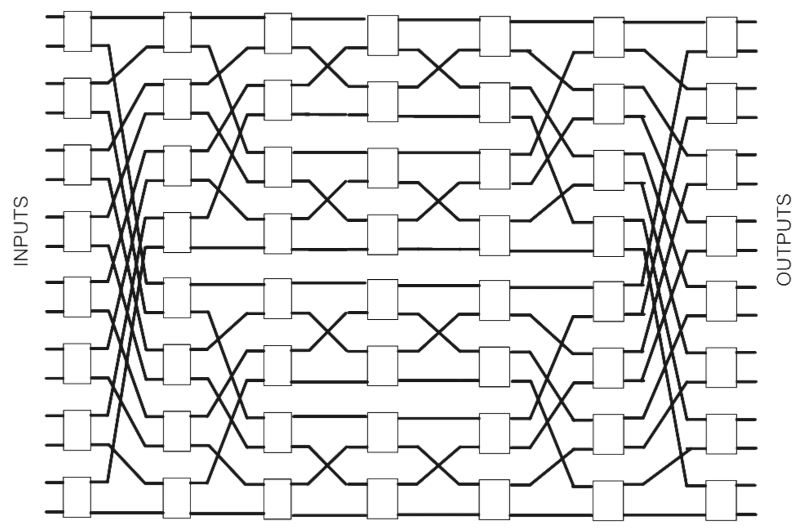


Figure 3.3: *A* 16×16 *Beneš network*

is generalized form of the link patterns presented in [62] .

$$L_{0:(2\log N-2)} = \begin{cases} (b_l \dots b_{l-k+1}\, b_0\, b_{l-k} \dots\, b_1, & 0 \leq k < \log N - 1) \\ (b_l b_{l-1} ... b_{k+2} b_k b_{k-1} ... b_0 b_{k+1}), & \log N - 1 \leq k < 2\log N - 2) \end{cases} \tag{3.10}$$

where the binary representation of O_k, which is the output O for switching element $\lfloor O/2 \rfloor$ at stage k, is $(b_l\ b_{l-1}\ \dots\ b_0)$ and $l = \log N - 1$.

For a 16×16 network with $l = 3$ the link patterns between stages 0 and 2 are as follows:

$$\text{stage } 0 : b_0 b_3 b_2 b_1$$

$$\text{stage } 1 : b_3 b_0 b_2 b_1$$

$$\text{stage } 2 : b_3 b_2 b_0 b_1$$

The link patterns for stages $(\log N - 1)$ to $(2\log N - 3)$ are given by:

$$L_{(\log N-1):(2\log N-3)} = (b_l b_{l-1} ... b_{i+2} b_i b_{i-1} ... b_0 b_{i+1}) \tag{3.11}$$

where $0 \leq i \leq (\log N - 1)$, and $l = \log N - 1$.

The link patterns for the remaining stages of the 16×16 network can thus be given by:

$$\text{stage } 3 : b_3 b_2 b_0 b_1$$

$$\text{stage } 4 : b_3 b_1 b_0 b_2$$

$$\text{stage } 5 : b_2 b_1 b_0 b_3$$

Fig 3.3 shows such a 16×16 Beneš network. The overall required number of crosspoints in this network is given by Eqn. 3.12. This specific class of rearrangeable network built with 2×2 switching elements will receive particular attention in the remainder of this thesis due to its low hardware complexities as well as for the capabilities of being able to realize all $N!$ permutation.

$$C_{Benes} = 4N\log N - 2N \tag{3.12}$$

3.3 Preliminaries

Input Permutation is the set of one to one requests between the switch inputs and outputs. In more mathematical term, a mapping of an input to an output is an element in the input permutation. Let us assume that $P_{0:(N-1)}$ is a given permutation such that, $P_{0:(N-1)} = \{x_i | x_i \in \{0 \ldots (N-1)\}\}$, where $x_i \neq x_j$, and $0 \leq (i,j) \leq (N-1)$. The mapping $P : i \rightarrow x_i$ indicates that input port i is requesting for the output port x_i.

$$\mathbf{P_{0:7}} = \begin{bmatrix} 0 \\ 7 \\ 3 \\ 2 \\ 4 \\ 1 \\ 5 \\ 6 \end{bmatrix}$$

For example, an 8×8 network with an input permutation $P_{0:7} : (0\ 7\ 3\ 2\ 4\ 1\ 5\ 6\ )$, maps $0 \rightarrow 0$, $1 \rightarrow 7$ and so on. A binary representation of these permutation for given by the above permutation matrix, where each row number corresponds to an input port and bits correspond to the requested output port, which can be used to expressed in binary the switching element settings.

Routing Tags they comprise an appropriate sequence of control bits per network stage, and optionally, an activity bit to indicate that a valid tag is present. In a Beneš network, some $(2\log N - 1)$ bits required to specify a route from source to destination. The first $(\log N - 1)$ bits may be freely chosen subject to the constraint that the resulting path is non-blocking. The remaining $\log N$ bits are for the bit controlled part of the network, i.e. the $\log N$ stages through which only a single path is available to the destination. Therefore, determining the last $\log N$ bits of the tag is straight forward, as they are the $\log N$ bits binary representation of the requested output port.

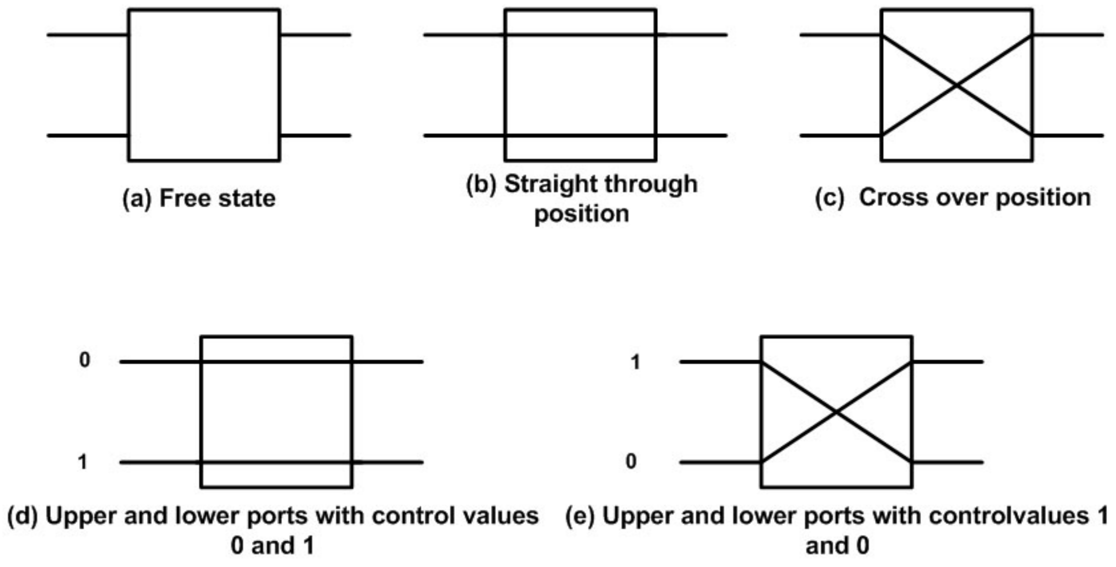

(a) Free state
(b) Straight through position
(c) Cross over position
0
1
(d) Upper and lower ports with control values 0 and 1
1
0
(e) Upper and lower ports with controlvalues 1 and 0

Figure 3.4: *Possible settings for switching elements.*

Switching Elements are the basic building blocks of Beneš networks, just as with any other switching network. In a Beneš network the total number of $2 \times$

2 switching elements is $(2\log N - 1) \times \frac{N}{2}$. Using a matrix notation, a switching element can be identified by its position $[i, j]$ in the switch,where $0 \leq i \leq \frac{N}{2} - 1$ and $0 \leq j \leq (2\log N - 2)$. A switching element can be in any of three possible states, one being a free state, which means the switching element can be set to any one of the other two possible states in response to input routing tags. The second is the straight through state, which connects the input ports to the corresponding output ports. The third state is cross state, which means that signals entering the upper input port, exit from the lower output port and a signal entering the lower port exits from the upper output port. These switching elements use a single bit in the routing tag as a control bit. The control bits for the two inputs have to be different to avoid any conflict. A control bit of 0 requests transmission to the upper output port and in case of a 1 it request the lower output port. Fig. 3.4 shows the three possible switching element states and how they are based on the control bits.

As already mentioned the calculation of the first $(\log N - 1)$ bits is the major concern as it determines a unique middle stage switching element for the path, that must be chosen to ensure conflict free routing. The matrix R below shows a sample set of conflict-free routing tag for an 8×8 Beneš network. The corresponding routes are shown in Fig 3.5.

$$\mathbf{R} = \begin{bmatrix} 00000 \\ 10011 \\ 01111 \\ 11110 \\ 11001 \\ 01010 \\ 00100 \\ 10101 \end{bmatrix}$$

The routing tags are processed from left to right, i.e the leftmost bit determines the next link for a packet in the last stage. An example of a routing with conflicts

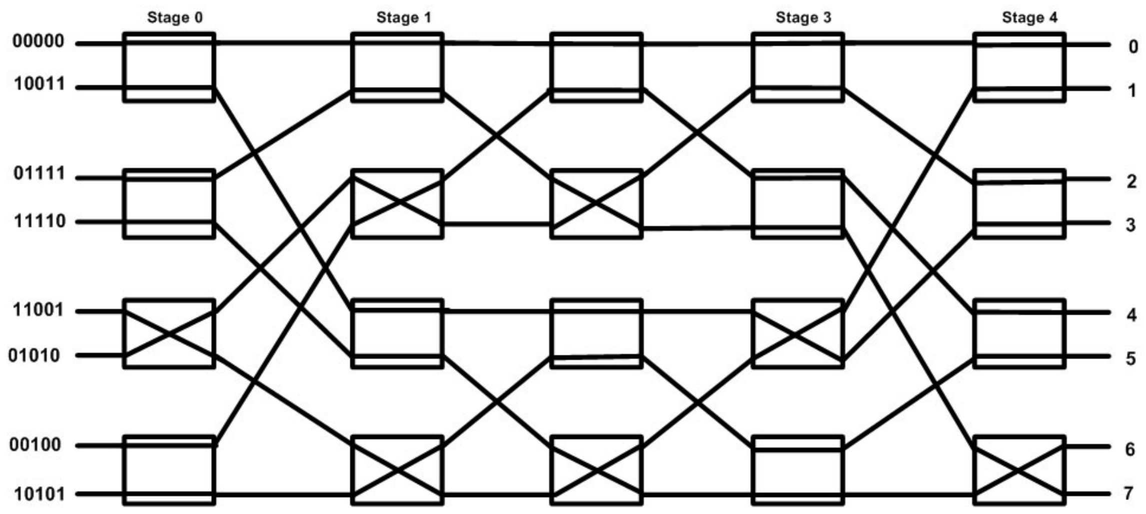


Figure 3.5: *Routing using the generated conflict free routing tags.*

is shown in the tag matrix R_c. Fig. 3.6 shows the conflicting state inside the network because of this conflicting routing tag matrix.

$$\mathbf{R_c} = \begin{bmatrix} 01000 \\ 10011 \\ 01111 \\ 11110 \\ 11001 \\ 01010 \\ 00100 \\ 10101 \end{bmatrix}$$

3.4 Blocking Probabilities: The Lee and Jacobaeus Method

This section overviews the blocking probabilities for alternative path networks. First the Lee method [98] will be described in brief followed by the Jacobaeus method [99].

A point-to-point communication process in a network is defined by two terminals involved in the communication. Links are the resource set that are shared

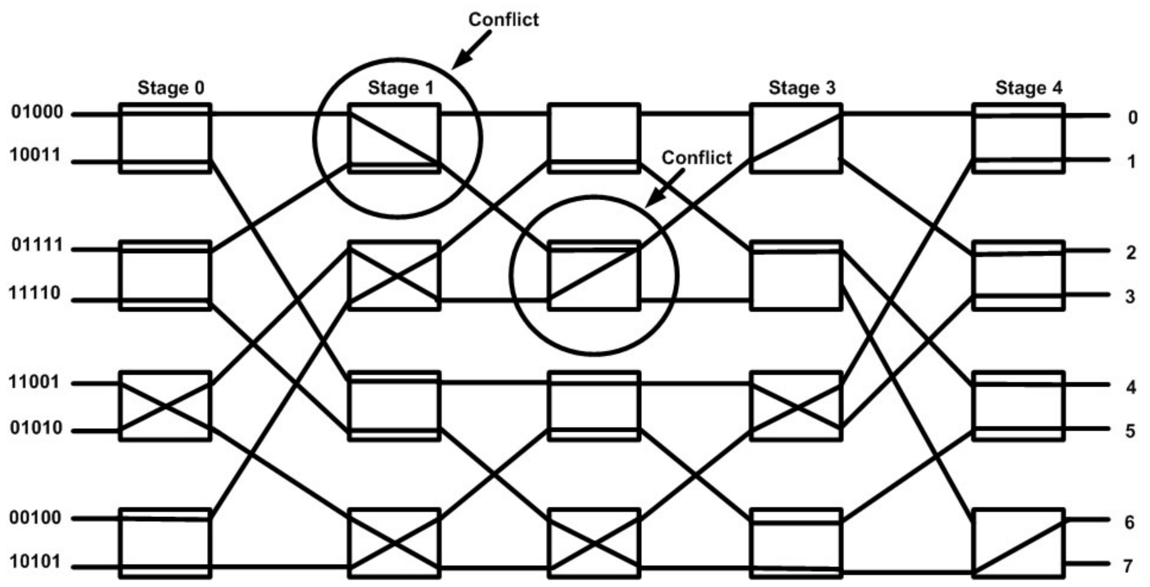


Figure 3.6: *Conflicting state inside the switch.*

by the communication in the network denoted as Υ. A set of links establish a path and that set is denoted as π_i, where for each request i. For single path networks there is at most one π_i for each i, for alternative path routing there will be multiple π_i. Lets assume that B be a random variable denoting blocking for any i and A being the availability of π_i, so $B = 1 - A$. The probability of blocking can be given as in Eqn 3.13:

$$P_B = E[B] = E[1 - A] = E[1 - \vee_{\pi \in M} \prod_{x \in \pi} X] \tag{3.13}$$

Where M is the set of available paths for a request i. For resource $x \in \Upsilon$, a Bernoulli availability random variable is X, where $X = 1$ is means resource x available for $X = 0$ it is not available. An alternative expression of Eqn 3.13 can be given as in Eqn 3.14, where it is assumed that all alternative paths for i are not available :

$$P_B = E[\prod_{\pi \in M} (1 - \prod_{x \in \pi} X)] \tag{3.14}$$

The Lee approximation assumes that for all $x \in \Upsilon$, the associated X are mutually independent. Given the independent assumption in the Lee model, the blocking probability given in Eqn 3.14 becomes:

$$E[B] = E[\prod_{\pi}(1 - \prod_{x\in\pi} X)] \tag{3.15}$$

$$= \prod_{\pi}(1 - E[\prod_{x\in\pi} X)] \tag{3.16}$$

$$= \prod_{\pi}(1 - \prod_{x\in\pi} E[X]) \tag{3.17}$$

With regular structure multistage networks, $E[X]$ can easily be determined. Networks such as three stages rearrangeable Clos networks are of these class. Rearrangeable Clos networks can show blocking characteristics if the network is not allowed to perform necessary rearrangements. Before going into the detail of derivation lets assumes two assumptions, *terminal symmetry* and *network symmetry*. The terminal symmetry assumes that input terminals are independent and identical as the network input. Also any input can request any available output. The network symmetry assumes links in a stage are symmetrical. Considering these symmetries, an assumption can be made about an occupancy probability p_k for all links at the output stage k, $1 \leq k \leq n$. Let there are l_k outgoing links at each stage. Since the number of established circuit is equal to the number of occupied links in each stage, Eqn 3.24 is true:

$$p_0 l_0 = p_1 l_1 = \ldots = p_n l_n \tag{3.18}$$

Given p_0, p_k be obtained easily. The probability of blocking is then given as:

$$E[B] = \prod_{\pi}(1 - \prod_{k=1}^{n-1}(1 - p_k)) \tag{3.19}$$

$$= [1 - \prod_{k=1}^{n-1}(1 - p_k)]^R \tag{3.20}$$

$$= [1 - \prod_{k=1}^{n-1}(1 - p_0\frac{l_k}{l_0})]^R \tag{3.21}$$

where R is the number of routes.

The Lee model assumes that all links within the network are independent. Which means that using one link in the network will not affect the the usability of other links. This gives a higher probability estimation and The Jacobaeus method improves the Lee model. The Jacobaeus method assumes more accurate description of the distribution for the number of occupied links at the input or the output of each switching elements. Probability of blocking using the Jacobaeus can be given as:

$$E[B] = \frac{(m!)^2(2-p)^{2m-r}p^r}{r!(2m-r)!} \tag{3.22}$$

With recursive application of the Lee method it can be shown that network with complexity $O(N\log N)$ have a blocking probability converging rapidly to a constant $0 < A < 1$. Lets assume that for a 3-stage Clos network, r_k links are coming out from first stage node or going out into each third stage node. For each of these links, let's assume link availability is$q_k = 1 - p_k$ and A_k be the probability that an input-output connection will be made. Applying the Lee assumption, A_k can be expressed in terms of q_k and A_{k-1}, the connection can be made in the middle stage.

$$A_k = 1 - \left(1 - q_k^2 A_{k-1}\right)^{r_k} \tag{3.23}$$

Furthermore, following conservation equation also holds:

$$p_k r_k m_k = T \qquad for\ all\ k \tag{3.24}$$

Where m_k is the number of switching elements in the first or third stage, and T in the total traffic through the network. This recursive relation can be use to calculate blocking probabilities for network with $5,\ 7,\ 9\ldots$ stages. Consequently assuming $r_k = r$ and $q_k = q$ for all k will show that for Beneš network it is possible to establish path for a request with a constant blocking probability A. All the derivations presented in this section are the generalized form of the derivation presented in [100]. More general derivation of these methods can be found in [101].

The assumption of these methods for calculating blocking are that the networks are not working as a rearrangeably nonblocking networks, hence there will the blocking matric associated in setting up path for any new request. Studies of the networks in this thesis consider that the networks are rearrangeably nonblocking. As a results non of the above mentioned blocking probability model will not be referenced in future discussion of this thesis.

3.5 Beneš based Networks

This section draws an overviews on the network topologies derived from Beneš network structures.

One of the early network designed using the Beneš network as their base is the Cantor networks [102]. Cantor network was designed to built a strict sense non-blocking network [103] but with less hardware cost than the Clos networks. It was designed as a parallel cascading of m Beneš networks, where $m = \log N$ and N is the input to the network. In the network there are N, $1 \times \log N$ splitter in input side of the network and N, $\log N \times 1$ in the output side of the network.

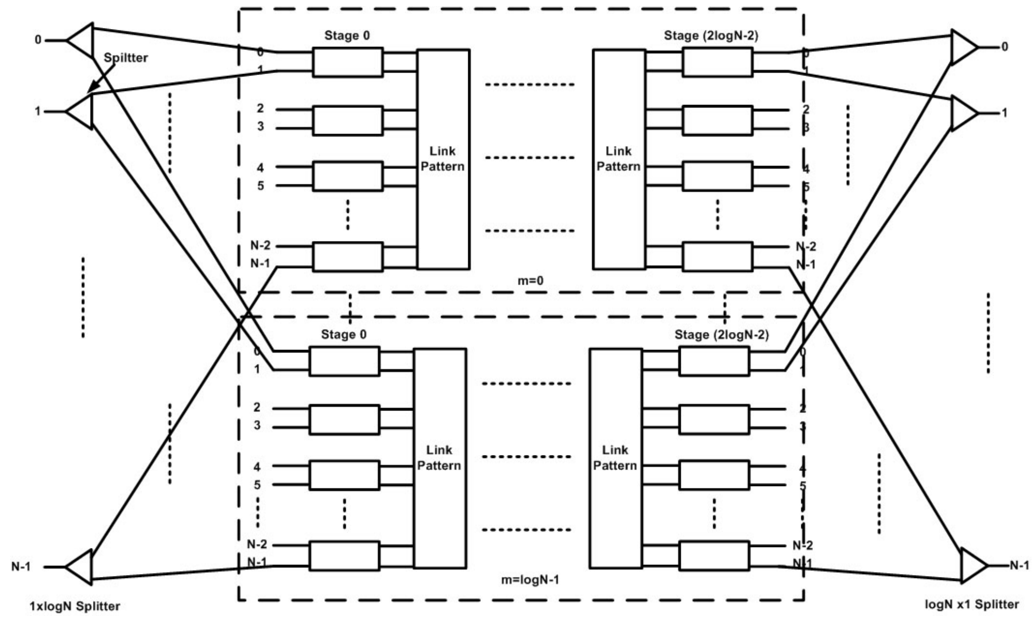


Figure 3.7: *$N \times N$ Cantor network.*

An $N \times N$ Cantor network can be constructed with roughly $4N\log^2 N$ crosspoints ignoring the cost of the splitter and there will be $2\log N + 1$ switching stages in the network. Input i of the network is connected to inputs i of the parallel $\log N$ networks, similar output j is connected to outputs numbered j in the parallel networks. Recently use of Cantor network in optical domain as a permutation network have been demonstrated in [104, 105]. Fig. 3.7 shows an $N \times N$ cantor network.

One of the recent permutation network derived from Beneš network is the KR-Beneš network [106]. In [106] it is suggested that with little modification to the regular Beneš network it is possible to design a permutation specific rearrangeable network. These networks can route those permutation with smaller latency and control over head than regular Beneš networks. First a K-Beneš was designed to satisfy K-bounded permutation, which satisfy the condition $|P(i) - i| \leq k$, for all inputs i, $0 \leq i \leq N-1$, k is any integer in $[0, 1, \ldots, N-1]$ and $K \leq N$ is the smallest power of 2 integer≥ 2. In general there are $K!(K+1)^{N-K}$ such permutations for $K \leq \frac{N}{2}$. K-Beneš has three logical parts in the network structure, matching networks, band-exchange network and routing network. There are $\frac{N}{K}$,

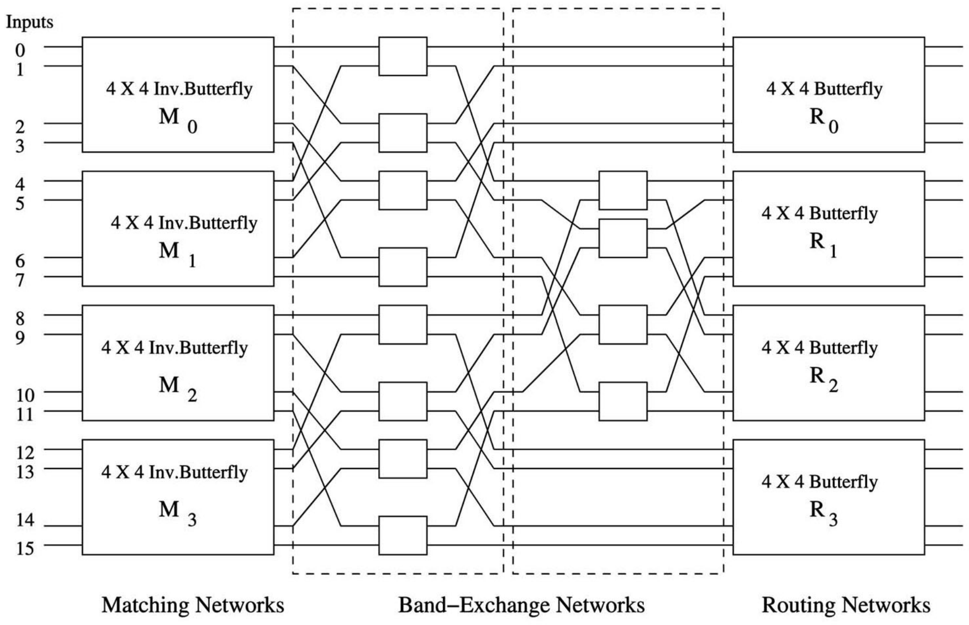


Figure 3.8: 16×16 *K-Beneš Network.*

$K \leq \frac{N}{4}$ parallel matching networks built with inverse butterfly networks also same number of inputs and orientations of routing network built with butterfly networks. The Band-exchange network constructed with column of switching elements connected with shuffle exchange link patterns. In the outer half of the network, switching elements are set using the looping algorithm, and in the Band-exchange network a matching algorithm is used. Overall depth of these networks is $2\log K + 2$. These networks can be used to reduce the hardware complexity when the effective permutation size dose not increase with time. But for realistic cases the permutation size will increase over time and hence use of Beneš is the solution in those scenarios. Fig 3.8 shows a 16×16 K-Beneš networks with its logical parts.

KR-Beneš was designed by using K-Beneš as a building block for the network. It has been demonstrated in [106] that using K-Beneš as building block a $3\log N - 3$ stages rearrangeable network can be built that is control optimal for specific permutations compared to regular Beneš networks.

A self routing network with larger depth than Beneš networks using complementary Beneš has been proposed in [107]. These networks have depth in $O(\log^3 N)$, and total number of switching elements in $O(N\log^3 N)$. This results have been improved to a $O(\log^2 N)$ depth and $O(N\log^2 N)$ switching elements in [108, 109]. In [110], a structure of rearrangeable networks have been presented. This study used network design similar to the cantor networks, where parallel plane are arranged connected to the input(output) through multiplexer(demultiplexes). This work is effective in designing switches for optical domain, as the signals has to travel less number of stage. Also this can show limited blocking probability if rearrangement functionality is not used. Each plane contains cascading to two networks one is the Beneš network for first l to $l + k - 1$ stages and the second part uses baseline network. With $l = 0$ and $k = \log N - 1$ this network transforms into a Beneš network. With $k = 1$ and $L = 2^l$ this network show similar properties as parallel delta networks [111, 112] though they are topologically different.

Lee [113] showed that network built with cascading omega and omega^{-1} are topologically equivalent to Beneš networks. She also proposed routing algorithm that can be used on these equivalent networks where last $\log N$ stages can be controlled using bit-controlled routing rather than the recursive looping algorithm. In [114], a r-truncated Beneš network is proposed. This was designed with a combination of r truncated stages of Beneš network and cascading of omega network. A randomized routing with $r = 1$ (one stage of Beneš network) is applied in this network and overall performance shows improvements compared to a single omega network of size N. For a network size of $N = 1024$, an optimal performance can be achieved when 4×4 crossbars are used as switching elements and r is set to 8, which means only one stage of randomized decision making. Another network uses $r \times r$ switching elements to built $3-$stage Beneš network [115], where $N = r^2$ and $r \geq 2$. It uses a self routing technique in the first stage to setup the switching element hence finds a free middle stage switching elements. One issue with this method is that when the value of r is big, this will require bigger size of switching elements hence making the self routing more complex.

Beneš based networks have been extensively studied for use in optical domain [116–119]. A network called N-stage planar optical permutation network is proposed in [120]. This was designed for optical domain and to reduce optical cross-talk. In electrical domain the cross-talk between two crossover signals is very negligible, but in optical communication this has big impact in overall communication. To overcome the cross-talk problem, a planar network is proposed. The network is designed in such a way that each switching element will have only one active input at a time. This design requires a total of $\frac{N(N-1)}{2}$ switching elements, where N is the total input size for the network.

Network proposed in [118] a cross-talk free optical permutation network is presented using Beneš. This work uses the concept of semi-permutation proposed by Hall [121]. The idea is to feed each switching element in the network only one input signal at a time. This process will eliminate cross-talk problem since only one input will be active at a time in a switching element.In these networks generating the partial permutation for each copy is carried out using the Euler-split technique for coloring bipartite graphs [122]. A partial permutation, $P' = \begin{pmatrix} x_0 & x_1 & \dots & x_{\frac{N}{(C-1)}} \\ y_0 & y_1 & \dots & y_{\frac{N}{(C-1)}} \end{pmatrix}$ for an $N \times N$ vertically stacked Beneš network, where x_i maps to y_i and $x_i, y_j \in \{0, 1, \dots N-1\}$ is a cross talk free feasible permutation if:

$$\{\lfloor \frac{x_0}{C} \rfloor,\ \lfloor \frac{x_1}{C} \rfloor, \dots, \lfloor \frac{x_{\frac{N}{C-1}}}{C} \rfloor\} = \{1,\ 2, \dots, \frac{N}{C} - 1\}$$

$$\{\lfloor \frac{y_0}{C} \rfloor,\ \lfloor \frac{y_1}{C} \rfloor, \dots, \lfloor \frac{y_{\frac{N}{C-1}}}{C} \rfloor\} = \{1,\ 2, \dots, \frac{N}{C} - 1\}$$

Detail proof of this method can be found in [123]. It has been show in the work that any permutation can be realized in two passes by avoiding cross-talk in the network. Another work presented in [1, 124] shows the modification to the general Beneš networks, by replacing middle stage with 3×3 structure. Modified networks have similar link patters as in general Beneš networks in the outer stages

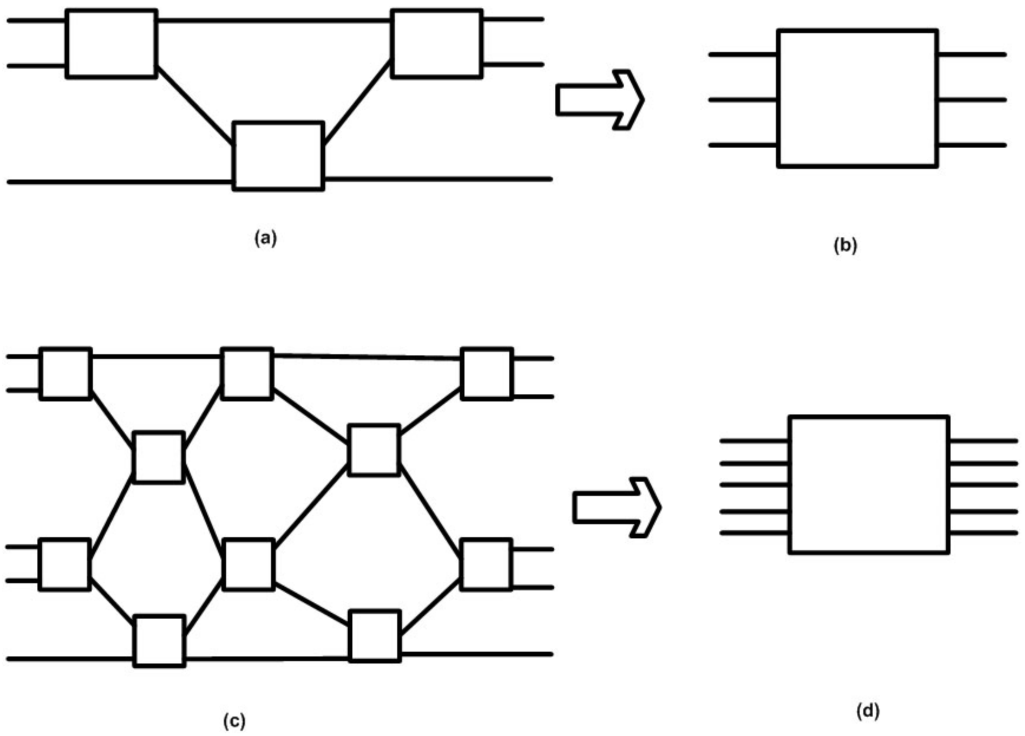


Figure 3.9: *Modification to the inner stage networks.*

but the inner stages are built with 3×3 and 5×5 switching networks. Algorithm proposed in [1, 124] works recursively and follows the classic form of the looping algorithm. Closer observation shows that the inner 3×3 network required number of total crosspoints are 12, but this could easily be replaced with a 3×3 crossbar which requires a total of 9 crosspoints per network. This saves 3 crosspoints per networks then the proposed one. Similarly for 5×5 network can be replaced by 5×5 crossbar than will save 15 crosspoints per network. Fig 3.3 shows the modifications. With the addition of the crossbar networks in inner stages will reduce routing complexity also is inner stage will be strict nonblocking.

3.6 Overviews on Beneš networks Routing Algorithms

In this section an overview of Beneš networks routing algorithm will be presented. This section will cover both the serial and parallel routing algorithms that have been proposed in the literature.

An elegant routing algorithm is proposed by Waksman [93] for Beneš networks. This method creates a permutation matrix for each permutation feed into the network. Output requests creating chains are indicated in the permutation matrix and then routing tags are determine depending on the status of the permutation matrix. Realizability of $N!$ permutation for Beneš networks is proved by Waskman. He also showed that Beneš networks are the smallest depth permutation networks built with 2×2 switching elements.

One of the most popular routing algorithm for rearrangeable networks is proposed by Opferman and Tsao-Wu [125], called the looping algorithm. This algorithm works for the family of Clos networks with $m = 2^t$ where t takes integer values. For Beneš networks the symmetric structure of the network is used to determine the switching element settings in a recursive manner using the looping algorithm. The looping algorithm works by setting switching elements in outermost stages of the networks, then it go on set the switching elements in inner most stages. It has a time complexity of $O(N\log N)$ in uni-processor system. This is the minimum time required for any single processor system reported for non-blocking routing in Beneš networks. Fig 3.10 shows an example of the strategy for setting the switching elements in the network. Later, Andersen [126] extended the looping algorithm for base 2^t rearrangeable networks.

A self routing algorithm was proposed by Nassimi and Sahni [127, 128] for Beneš network. Using parallel processing switch setting time of the algorithm is $O(\log N)$. They used a bit controlled routing technique for all the stages in the network. In their scheme they used binary equivalent of output request in both half of the network. Their switching elements setting scheme can be described by the following equations, where U_i and L_i are upper and lower input port of a switching element and D_i is the destination port for input i, where $0 \leq i \leq (N-1)$:

$$U_i = \begin{cases} D_{2i}, & \text{if } (D_{2i})_0 = 0 \\ D_{2i+1}, & \text{if } (D_{2i})_0 = 1 \end{cases} \tag{3.25}$$

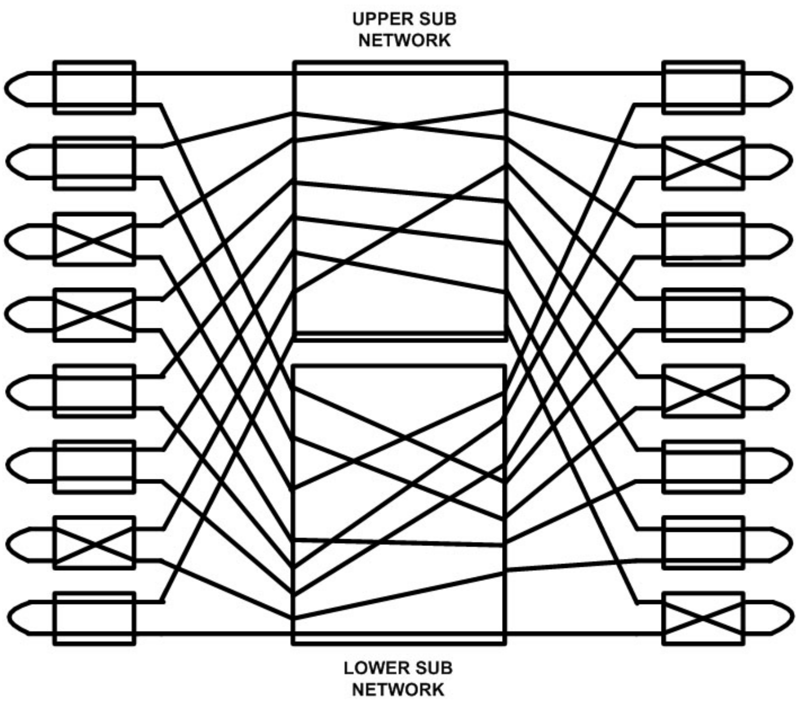


Figure 3.10: *Path setup using looping algorithm.*

$$L_i = \begin{cases} D_{2i}, & \text{if } (D_{2i})_0 = 1 \\ D_{2i+1}, & \text{if } (D_{2i})_0 = 0 \end{cases} \tag{3.26}$$

The above two switching element setting schemes suggest that when any switching elements at any stage receives routing tags it takes decision depending on whether the input is an upper or a lower input to the switching element. In case of an upper input and a routing tag of 0 the signal exits from the upper output port, otherwise it will exit from the lower output port. Similarly for lower input port, a routing tag of a 0 means signal exit from the upper output port of the switching element, otherwise the signal will exit from the lower output port. This proposed method has its limitations in performing a successful permutation. According to [127, 128], this scheme can only support the permutation class called bit permute complement described in [129]. For a network of size N this method can successfully route $N \times (\log N)!$ permutation compared to the full $N!$ permutations.

The looping algorithm is designed to work in parallel processing mode with completely connected, mesh connected [130],cube connected [131] and perfect shuffle computer [132] in [133]. The idea was to divide the outputs in two equivalency class, where two outputs in the same class will be routed through the same middle stage switch. For completely connected computers the algorithm has a time complexity of $O(\log^2 N)$. This is still the minimum time required to setup a Beneš network in the literature for full permutation request. For other connected computers the time complexities are $O(N^{1/2})$, $O(n^4)$ respectively.

Lee [134, 135] introduced the idea of routing partial permutation or incomplete assignments in the Beneš network for uni-cast routing. The available algorithm were designed for work for full permutation. This work address the routing algorithm for $O(m)$ inputs instead of $O(N)$, where $m \ll N$. The algorithm was implemented on two connecting topologies, one is completely connected and the other one is extended perfect shuffle. This method introduced the concept of closed chain and open chain. Inputs from same switching elements form two different chains where the status bit for each input have the same value. Lets assume that $(r_i, d^i_{\log N} \dots d^i_0)$ represents the header for input i, $0 \leq i \leq N-1$. Bit r_i is the activity bit that indicate if input i is paired with some other input. It also indicated the status of the input i.e, if $r_i = 1$ the input is busy and $r_i = 0$ the input is inactive. Depending of the input pairs, the routing algorithm assigns middle stage switch to the pairs. This method is applied for first $\lfloor \log m \rfloor$ stages of the network and then the routing is bit controlled. The overall complexity of this algorithm is $O(\log^2 m + \log N)$ in completely connected network and $O(\log^4 m + \log m \log N)$ in extended shuffle exchange network.

Routing algorithm proposed in [136–138], works for Beneš class of rearrangeable networks, called inside-out routing. Instead of going form inside towards the middle stage, this method works from the middle stage towards the outermost stage. Serial time complexity for the algorithm is in $O(N)$. But in [139, 140] showed that inside-out routing is not blocking free. Required modification to make the algorithm blocking free makes the time complexity approaches

$O(N\log N)$, which is similar to the looping algorithm. Two candidate algorithms for use as baseline blocking algorithm in the comparisons of Chapter 5 are the inside-out routing and the random routing [141]. The inside-out algorithm was not chosen for it's blocking performance because no simulation results are reported in literature. It has been shown that inside-out algorithm is blocking in [139, 140], and the required modifications to make it blocking free suggest that without these modifications the algorithm will give a high blocking rate. Random routing has been extensively discussed in the literature and demonstrates fast execution time with a moderate blocking rate. Hence Random routing was chosen as the blocking routing algorithm for any blocking rate comparison in this thesis.

In [142] an $O(\log N)$ parallel self routing algorithm has been presented for specific class of permutations. This work identifies four different class of permutations and showed that they can be route using self routing in Beneš. Successful permutations that can be realized using this algorithm is $(2^{\log N}(2^{\log N(\frac{N}{2})-\log N)} + (\log N)! - 1)$ from each group. Authors from same group reported a two passes routing algorithm for faulty Beneš network in [143]. In this method they divide the permutation in two equal parts and establish paths for request in each path in one pass.

The algorithm proposed by Lee [144] has similar time complexity as the looping algorithm, which is $O(N\log N)$. But this method divides the networks in to two parts. In doing so, it is suggested that in original hardware implementation it will reduce inter-chip information exchange at the time of routing. The networks are divided into two parts namely $\mathrm{NS1}$ and $\mathrm{NS2}$. $\mathrm{NS1}$ is the first $(n-1)$ stages of the network where a binary tree control algorithm is applied. $\mathrm{NS2}$ is the remaining n stages which uses bit-controlled routing. The algorithm considers the incoming permutation as Complete Residue System module m $(\mathrm{CSR}\bmod(m))$ where m is the number of inputs. Then it divides the CRS into 2^{N-1} Complete Residue Partitions (CRP), and this process continues for the first half of the network. co For an example $P_{0:7} = (1\ 7\ 6\ 3\ 2\ 4\ 5\ 0)$ is a $(\text{CRS}\bmod(8))$ and applying CPR mod(4) on the CRS gives two permutations $(1\ 7\ 6\ 4)$ and $(2\ 5\ 3\ 0)$ or $(7\ 5\ 0\ 2)\text{and}(1\ 3\ 4\ 6)$ and so on. Applying this process this method generates a full binary tree as shown in Fig 3.11. In this tree the root node contains the full permutation and child nodes contain partitions of the permutations. This method dose not require information exchange between switching stages as, this algorithm sets up switching elements stage by stage without requiring information of other stages. Also the bit controlled part requires less complex hardware as it is a straightforward operation.

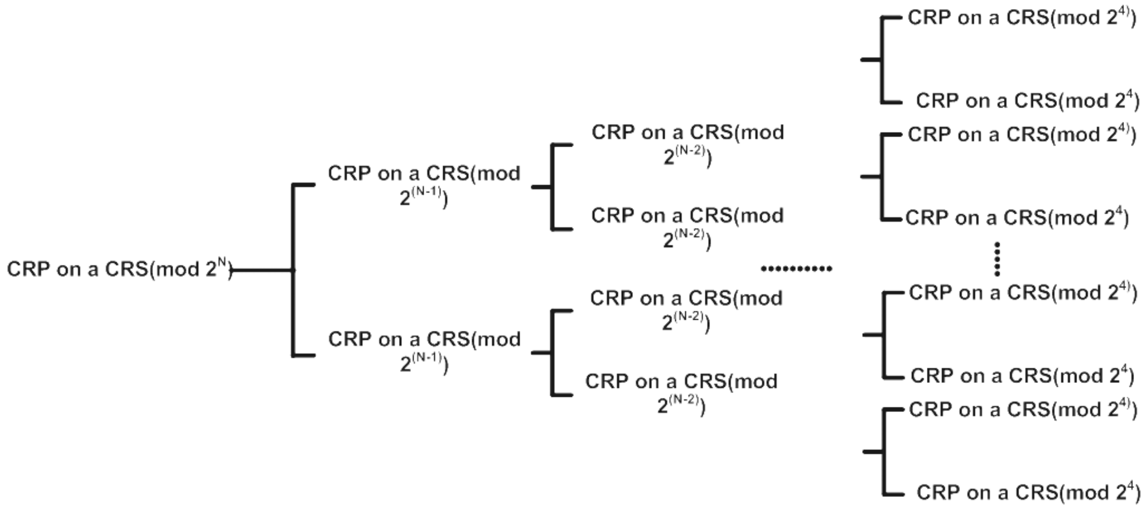


Figure 3.11: *A full binary tree of CRP's.*

The algorithm proposed by Lenfant [145] also works for a special class of permutations. This method was designed in such a way that if the presented permutation that does not fall in a special class then their routing is carried out using some existing algorithm that can realize $N!$ permutations. According to Lenfant using such a method saves memory space as well as computation time. The number of permutations that can be realised using this method is about $2^{2\log N}$ rather than $N!$.

Parallel routing method proposed in [146, 147] works for Beneš and Clos networks. For Beneš networks it determines the forward routing bits, which are the binary equivalent bits of the output request. Reverse routing bits, are the also binary equivalent bits going from the output towards the middle stage. And internal conflict free constant, which determines that for each input in the switching elements, have distinct tags to avoid internal conflict. This algorithm can route partial permutation and has a a parallel processing complexity of $(\log^2 N)$ using fully interconnected parallel computers.

Work efficient routing algorithm for symmetric rearrangeable network is proposed by Çam [148–150]. This work makes use of the concept of a balanced matrix [151] and graph 2-coloring [152] to determine the setting of switching elements and uses perfect matching graphs [153] to determine the routing tags for each input-output request. Graph coloring is used to assign different values to the connecting nodes so that two connected nodes do not share the same color or assignment. $2-coloring$ is the process of assigning vertices with the values 0 and 1, where two directly connected vertices have different values assigned to them. Fig 3.12 shows a $2-\text{colored}$ connected graph. A binary matrix B_M is a balanced matrix if and only if , for all the sequential m columns, $1 \leq m \leq n$, every m-bit binary number appears $2^{(n-m)}$ times in every row. In more general terms, a matrix with $N = 2^n$ rows and k columns, $k \leq n$ can have a single row repeated $2^{(n-k)}$. Matrix $B_{8\times3}$ shown below is a balanced matrix. A graph G is said to be perfectly matched if all the edges have two end vertices connected to them. From a balanced matrix point of view, if two rows i and j, $1 \leq (i,j) \leq N$, are identical then in a perfect matching graph they are connected by two vertices v_i and v_j where $(v_i, v_j) \in V$ and V is the set of vertices.

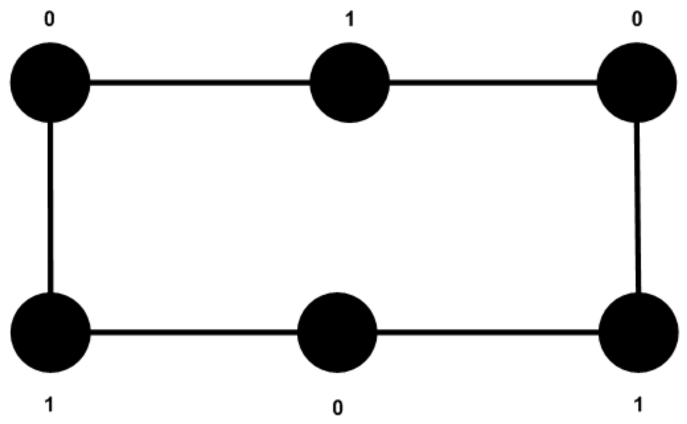


Figure 3.12: $2-$ Colored *connected graph.*

$$\mathbf{B_{8\times 3}} = \begin{bmatrix} 000 \\ 100 \\ 010 \\ 110 \\ 001 \\ 111 \\ 101 \\ 011 \end{bmatrix}$$

They have used the Parallel Random Access Machine [154] architecture for their algorithm. They considered p processors instead of N where $1 \leq p \leq N/\text{log}N$, which allows the algorithm to setup the switch in $O(N\text{log}N)$ time. Their main aim was to find a column vector V for permutation matrices $P_{1:n}$ and identity matrix $I_{1:\text{log}N}$ such that $[P_{2:\text{log}N}\ V]$ and $[I_{2:\text{log}N}\ V]$ are balanced. The proof of their method can be found in [150]. Tabel 3.2 gives a summary of the algorithms cited and their serial and parallel time complexities.

3.7 Application Domain of Beneš Networks

This section highlights the scope of the Beneš networks in real world applications.

3.7.1 On Chip Communication Applications

New applications have emerged for these class of networks in the field of system on chip (SoC) and network on chip (NoC). SoC is an arrangement of two or more complex micro electronic components in a single chip [122, 155–157]. Complex functionalities that required heterogeneous components attached to a PCB are replaced by SoCs. Advancements in the silicon technologies allow large functional unit to be built in a single chip. A typical SoC contains processors, on chip memories, accelerated functional units, signal processing units, logic circuits etc. The primary advantages of SoC is low cost, smaller in size and fast performance. Because of the SoC today's hand held devices are smaller in size compared to the bulky old versions.

To communicate between processing and other service provider module in an SoC it is required to have some form of connection between the units. Easiest solution to this is to use a bus base architecture that will allow communication between different parties [158]. But bus base architecture is not scalable as well as they can not support the required bandwidth for all attached units [159].

Direct point to point links are also easy to design and they can give better solutions in terms of bandwidth, power uses, latency when they are designed for a specific purpose. Issue with the point to point links is the increase in number of links with the increase of the cores, hence it puts impact on required space. The solution to these approaches is to use integrated switching network in SoC, which leads to the concept of network on chip (NoC).

NoC overcomes the scalability and performance issues related to the bus based or point to point communication structure in the SoC [157, 160–162]. The obvious choice for NoC in an SoC is the crossbar networks as they give superior performance than bus base models, but this network also suffers from scalability issue after a certain input number along with low network utilization [163].

Table 3.2: *Summary of Routing Algorithm Complexities*

Algorithm Time Complexity			
	Algorithms	**Serial Complexity**	**Parallel Complexity**
Blocking	Nassimi et al. [127, 128]	$O(N\log N)$	$O(\log N)$
	Das et al.[142]	–	$O(\log N)$
Nonblocking	Nassimi et al. [133]	–	$O(\log^2 N)$,$O(N^{1/2})$, $O(n^4)$
	Lee et al. [134, 135]	–	$O(\log^2 k + \log N)$,$O(\log^4 k + \log k \log N)$
	Lee et al. [146, 147]	$O(N\log N)$	$O(\log^2 N)$
	Cam et al. [150]	–	$O(N\log N)$
	Feng et al. [136–138]	$O(N)$	–

So the solution to these is multistage interconnection networks having switching elements arranged in rows and columns and each switching element is connected to the next stage via some fixed link patterns. These networks have better scalability property along with equal path distance between source and destination which make then viable for SoC.

Networks under observation in this thesis found their use in wireless communication standard. The efficient error correction capabilities of low-density parity-check (LDPC) [164, 165] codes have made their application in standard such as IEEE 802.11n, IEEE,802.16e and DVB-S2. Beneš networks have particular edge in designing SOC for LDPC in area they requires. Fig. 3.13 shows a general block diagram of an LDPC. Table 3.3 shows the comparisons among Beneš, Clos and Crossbar network for $N = 32$. This table highlights the number of gates or multiplexer required for implementing a network with $N = 32$. These comparison shows clear benefit of using Beneš network as the connecting networks for LDPC decoders. It has been reported that Beneš networks require less fabrication area in designing a turbo decoding architecture [166] compared to other nonblocking interconnection networks.

Algorithm proposed in [1, 124] works recursively and follows the classic form of the looping algorithm. But the network they use is an scale down version of the regular Beneš network that supports specific permutations.

This application domain requires an algorithm that can give faster execution time than the looping algorithm and lower blocking probabilities than suboptimal algorithms. In chapter 5 one such algorithm will be discussed, where a fast and simple routing solution will be achieved with lower blocking probabilities.

3.7.2 Digital Subscriber Loop Applications

Another new application domain for Beneš networks is in the field of digital subscriber line (DSL) technologies. These technologies work in the analog domain

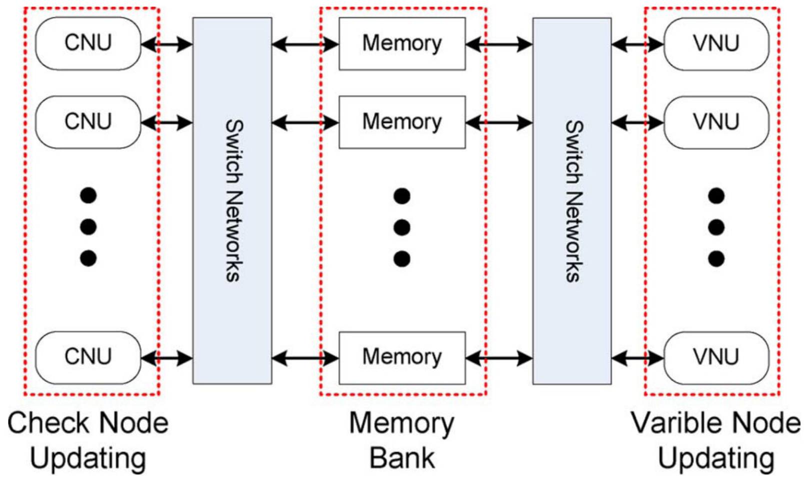


Figure 3.13: *Architecture of partial parallel LDPC decoder. [1]*

Table 3.3: *Complexity of different switching fabrics.*
[167]

	Gates	**2-to-1 MUX**	**Control bits**
Crossbar	1024	992	160
Clos	896	784	376
Beneš	576	288	144

and manage thousand of subscribers. DSL is a broadband service over ordinary telephone lines. Telephone operators provides broadband service (such as voice,video and data) along with normal telephone service. Both services operate in different frequency bands so that there is no chance of any interference. At the service provider end another device called the DSL access multiplexer (DSLAM) routes voice request to the exchange and data traffic to the Internet. Fig. 3.14 shows a typical DSL structure. The DSLAM is no needed for subscriber lines that are voice-only. To change the subscribers service (for example changing voice-only service to add data or vice versa) requires managing a huge number of cables. This manual effort can be automated by using a switching network that connects the cable to a DSLAM to provide the required service when appropriate [168]. Changing the subscriber line may require a rearrangement in the analog switching network and cause momentary interruption of the ongoing communications. This disruption of the communication can be avoided using the

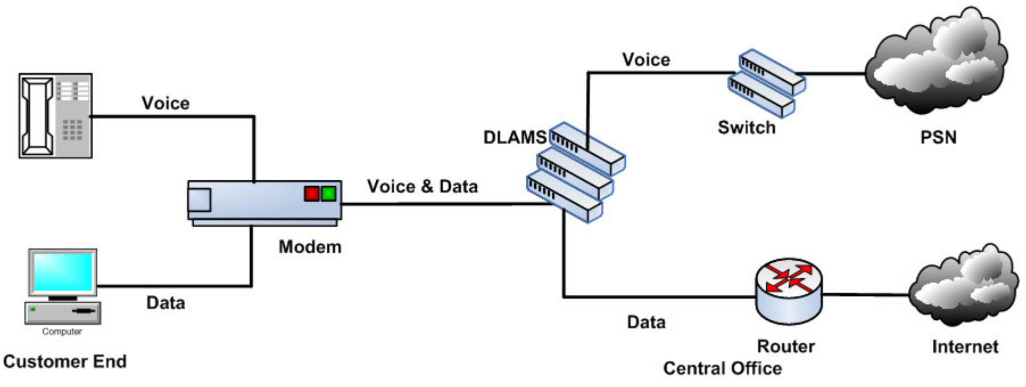


Figure 3.14: *Typical DSL configuration.*

technique described in Chapter 4.

Current DSLAM solutions are based on manual patching of connections, without any automation. Automation of this process requires handling thousands of input-output ports. Having a strict sense nonblocking network to handle such a large number ports would incur a huge hardware cost. Hence the Beneš network is the best choice for implementing the automation process, as the hardware does not grow exponentially in cost in these networks with increasing network size.

3.7.3 Optical Domain Applications

With the increasing demand from the Internet traffic, Dense Wavelength Division Multiplexing (DWDM) is being used in the core of large communication networks. Many techniques for optical switching are available in the literature such as interferometre, acoustooptic interaction, thermocapillary effect, electrooptic materials and microelectromechanical systems (MEMS) [169–174]. 2D MEMS can offer limited crosstalk, low polarization and wavelength independent, bit-rate and format transparent and an easy to manufacture solution [170].

The use of rearrangeable structure such as proposed in[73] has been suggested for high performance optical systems [175–177].These structure provide equal path lengths for all input-output requests also it has better port-to-port repeatability than a crossbar structure. A rearrangeable optical switch built with 2×2

switching elements, 2D MEMS has $N(\log N - \frac{1}{2})$ movable and double sided mirrors and has $\frac{N(\log N+1)}{2} - 1$ fixed and single sided mirrors for an $N \times N$ network. Path length is $\sqrt{2}(N-1)p$ in the rearrangeable structure, and as $(2N-2)p$ for upper bound in crossbar structure, where p is the pitch or distance between two nearest mirrors, [178]. Path length is important as for rearrangeable structure memory access distance is same for ever input-output request, but it varies in a crossbar structure.

The rearrangeable structure offers a mirror size which is almost 40% less than the crossbar structure and hence requires smaller fabrication area [179]. An $N \times N$ OXC proposed in [180] used the Waksman network [93] to build the switch, as this requires a minimum number of mirrors. The proposal suggested the use of a lookup table to implement incoming permutation to reduce the setup complexity. The MEMS-Beneš network (MBN) proposed in [181], is an optical Beneš network that can support any permutation. This network's control strategies are similar to those of a conventional Beneš networks. Straight through switching is performed by allowing the mirror to stay down that is parallel to the substrate and the cross operation is performed by configuring the mirror in tilted-up position.

3.8 Summary

This section gives a general discussion on the topics covered in this chapter. This chapter discussed mainly rearrangeable networks, their structures and routing algorithms. One of the most popular symmetric rearrangeable networks have been looked into detail in this chapter and that is Beneš network. It has been shown mathematically that Beneš networks save considerable amount of required crosspoints than compared to the $3-$stage Clos networks, by using recursive construction. Extensive literature have been reported in the literature on the Beneš networks both in the architecture and algorithm domain. Proposals have been suggested and shown in the literature that modified this network structure by reducing the network depth such as r-truncated or KR-Beneš networks. These modifications were applied to reduce the network latency by compromising the throughput. It has been described in this chapter, to realize $N!$ permutation, it is not possible to design a network built with 2×2 switching element that can work faster than Beneš networks. On the routing algorithms, Tabel 3.2 summarises all the cited algorithms. For single processor machine best reported time complexity is $O(N\mathrm{log}N)$ for N input-output Beneš networks. One algorithm has shown that it can be reduced to $O(N)$, but further studies showed that the algorithm is not blocking free and to overcome the blocking time complexity becomes $O(N\mathrm{log}N)$. Methods have been proposed in the literature to make the Beneš networks work as a self routing network, and achieve a parallel routing complexity of $O(\mathrm{log}N)$. But using those methods network can only realize $N.(\mathrm{log}N)!$ permutations in comparison of all $N!$. Parallel routing algorithm complexity have been shown to be $(\log^2 N)$ using completely connected parallel machine with N processors. It has been shown that the network can work with a complexity of $O(\mathrm{log}^2 m + \mathrm{log}N)$, for active requests $O(m) \ll O(N)$. But the complexity approaches $(\mathrm{log}^2 N)$ for large values of m. In recent years suggestions have been made for use of Beneš networks in optical domain as well as in SoC, NoC and DSL applications. Reported literature shows that using Beneš networks for these applications save hardware cost in actual implementation. To summarise this chapter it can be said that Beneš network is the optimum permutation network built with 2×2 switching element. Future chapters will focus on these networks unless otherwise stated.

CHAPTER 4

SYMMETRIC MULTISTAGE REPACKABLE NETWORKS

All though rearrangeable networks make efficient use of hardware they have the disadvantage that they momentarily disrupt the existing communications during reconfiguration. Path continuity is a major issue in some application of rearrangeable networks. Using a repackable network [182] is a solution to the path continuity problem and is discussed in this chapter.

4.1 Introduction

Repackable networks allow the path continuity even though it is built on top of rearrangeable networks. Basic structure of these networks contain bypass links, which route requests where paths need to be rearranged of those requests. This is done one at a time or all together through those bypass links for continuation of the ongoing communication. After the necessary rearrangements are made, requests through the bypass links are put back into their new rearranged routes and hence the term *repackable* is used for these networks. Addition of bypass links increases the hardware cost than a similar sized rearrangeable network but provides a performance close to a strict-sense nonblocking networks. The following sections detail the structural and operational principles of these networks.

4.2 Repackable Networks Overview

Most proposal for repackable networks are based on 3-stage Clos networks. Ackroyed [183] first introduced the concept of repacking for 3-stage networks. He proposed using one of the middle stage subnetworks for routing all the requests. This is generally termed as packing. If this packing technique results in blocking, the other subnetwork is used to route the blocked request. At a later time, that request is rerouted via the packed subnetwork. Jajszczyk [184] provided the basic condition for a 3-stage Clos network to be repackable. If n and m being the input and output for the each input(output) stage switching elements and r is the number of switching elements in the input (output) stage respectively, then repackable condition is, $m \geq 2n - \lceil n/(r-1) \rceil$. Jajszczyk [185] explained that it is always possible to establish connecting paths between an input and output using a repackable network. His proposed method routes requests from the less used subnetwork to the most used one before the arrival of a new call.

Schehrer [182, 186] proposed two method of reswitching in a repackable network (i) sequential (ii) simultaneous. The term *reswitching* is different from rearrangement. It means putting requests through the bypass network or paths, and put back to rearrangeable network once paths have been rearranged and making the bypass network or paths free for future use. In *sequential* reswitching, paths need to be rearranged are selected sequentially and put through the bypass paths. Once the paths are rearranged in the rearrangeable network they are put back to their new routes and then new set of paths are selected for the reswitching. On the other hand, *simultaneous* reswitching selects the smallest number of paths need to be rearranged and then put them to the bypass links or network.

Once the paths are rearranged in the rearrangeable network, they are put back to their new routes together.

Schehrer proposed to use a bypass network or bypass links to extend rearrangeable networks and make it a repackable network. The sequential method requires fewer crosspoints than simultaneous switching. Instead of holding the blocked request, a connecting path is setup for that request using the bypass paths, connections that need to be rearranged are also transferred to bypass paths. After finding new paths in the rearrangeable network for all the requests required rearrangement, they are switched to the rearrangeable network from the bypass paths or network along with the new request. This makes the bypass part of the network free for for future reswitching. In this way, path continuity is assured at the cost of providing the additional bypass capacity.

4.3 Preliminaries

This section presents the notation used throughout this chapter:

Definition 4.3.1 Chain: *Chains are used to identify the loop that needs to be broken, i.e the set of input-output pairs for which the switching element settings need to be changed. The length of the chain is the total number of paths forming the loop.*

If a candidate chain has r paths in it then the length of the chain is r. The length of the chain determines the number of rearrangement required for each blocked request. Fig. 4.1 shows an example of such a chain. Input 5 requests a connecting path to output 6. With the existing switching element state the free links at the input-output switching elements connect to two different middle stage switching elements, and so these links cannot be used to establish connecting paths between input port 5 and output port 6. The solution to this problem is to rearrange the chain having a loop $0 \rightarrow 3\ ;\ 2 \rightarrow 2\ ;\ 3 \rightarrow 0\ ;\ 4 \rightarrow 1$. Rearranging this chain will give both blocked switching elements access to a common middle switch.

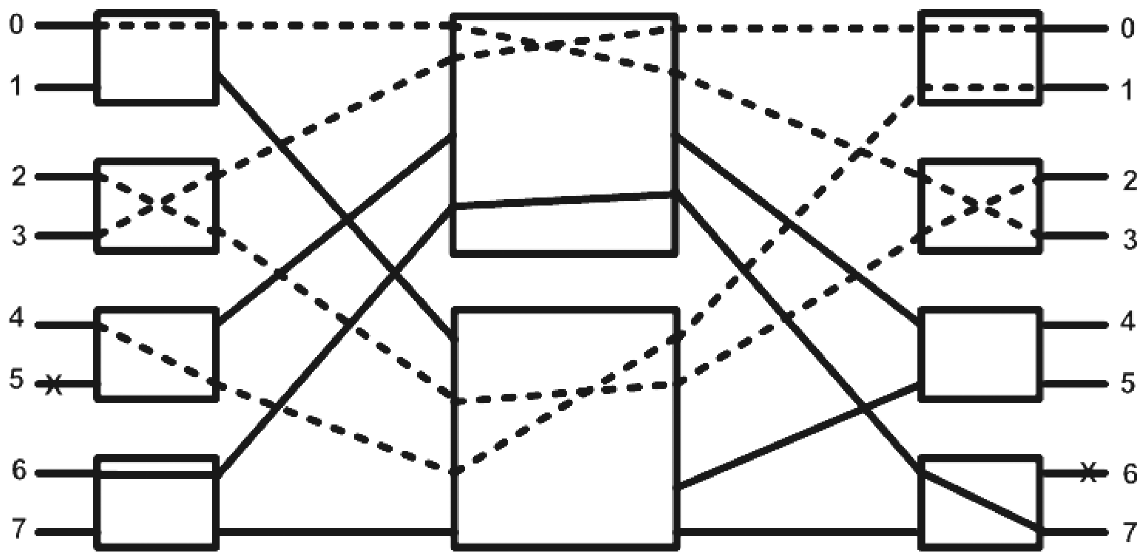


Figure 4.1: *A network having a chain with length* 4

Definition 4.3.2 Blocking State: *A blocking state inside a network results when more than one request is competing for the same output link. This puts the network into a state where paths rearrangement of ongoing communication are required to bring the network into a state where it is possible to establish path for the blocked request.*

For a 3-stage network blocking state can arise because of unavailability of connecting link from one middle stage switching element to the requested output switching element. For longer depth networks, it can happen because of the fact that required inner stage link is already occupied by some other request. For example in Fig. 4.1, the request by input 5 for connection to output 6 is blocked because of the current state of the network.

Definition 4.3.3 Isolated Paths: *If two inputs from the same switching element in the input stage are connected to two different output stage switching elements, and they are the only requests for those switching elements, the two paths concerned are called isolated paths. Similarly if two inputs of two different switching elements in the input stage are connected to the same output stage switching element, and they are the only requests from those switching elements they are called isolated paths.*

In Fig. 4.2 two isolated paths are from inputs 6 and 7.

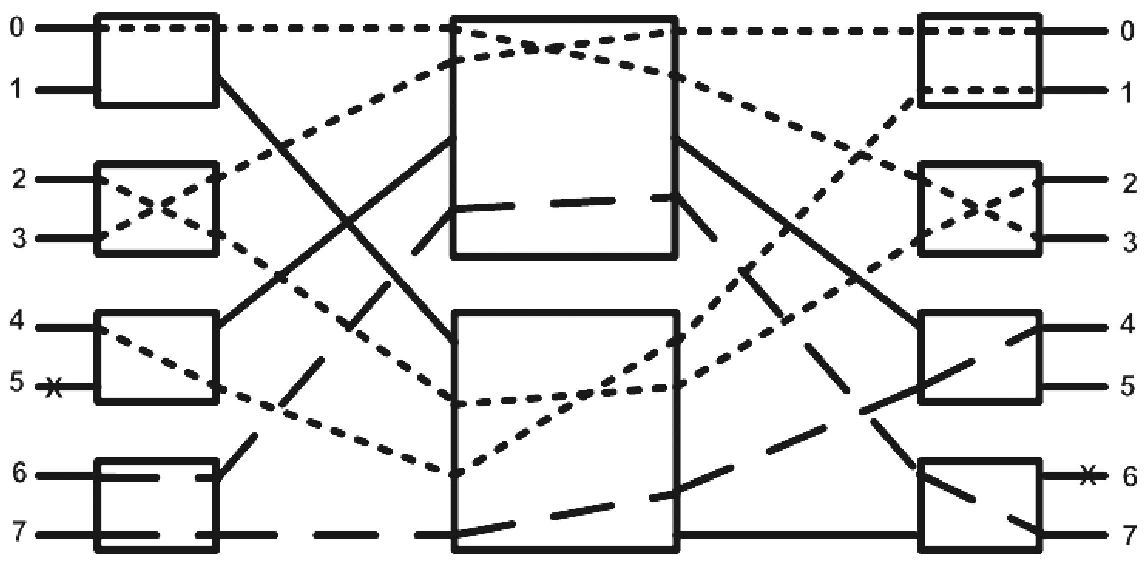


Figure 4.2: *Two isolated links from inputs* 6 *and* 7

4.4 General Rearrangement Scenario

This section describes the blocking scenario and rearrangement process in rearrangeable networks. Blocking resolution in a rearrangeable network has two basic operations. The first one is to identify the chain related to the blocked request and the other one is to rearrange the chain and setup a path for the blocked request.

Fig 4.2 demonstrates a blocking scenario for a request from the input 5 and the output 6. The Network goes into blocking state because input switching element 2 has one free link going to the upper middle stage switching element (U). Similarly, output switching element 3 has one free link coming from the lower middle stage switching element (L). There are no free links available sharing same middle stage switching element that can establish a path for the request. To setup a path for the request the network goes through rearrangement and finds path for the new request. Table 4.1 explains the input-output path map before rearrangement for the scenario presented in Fig 4.2, where the symbol "X" indicates input request is blocked.

Now let's identify the paths forming chains and also the isolated paths from the Fig. 4.2. Input-output paths forming the chain are as follows: $0 \rightarrow \mathrm{U} \rightarrow 3$,

Table 4.1: *Path map for input-output request.*

Input	Before Rearrangement		After Rearrangement	
	Network	**Output**	**Network**	**Output**
0	U	3	U	3
2	L	2	L	2
3	U	0	U	0
4	L	1	L	1
5	X	6	U	6
6	U	7	L	7
7	L	4	U	4

$2 \rightarrow \mathrm{L} \rightarrow 2$, $3 \rightarrow \mathrm{U} \rightarrow 0$ and $4 \rightarrow \mathrm{L} \rightarrow 1$, and the existing isolated paths are $6 \rightarrow \mathrm{U} \rightarrow 7$ and $7 \rightarrow \mathrm{L} \rightarrow 4$. To unblock the blocked request $5 \rightarrow 6$, the connecting paths forming the chain or the isolated paths need to be rearranged. Number of isolated paths is smaller than exiting chain length, hence they are selected for the required rearrangement. Once the isolated paths are rearranged, two links are available that share upper middle stage switching element, hence a path is established between the input 5 and the output 6. Table 4.1 also shows the paths map after required rearrangements.

The rearrangement process is carried out by treading off the path continuity of existing communications. Repackable network is used to overcome this path continuity issue of rearrangeable networks and discussed in detail in next sections.

4.5 Principle of Repackable Networks

The complexity of a repackable network is a function of the minimum possible chain length . This also depends on whether operations are performed on all effected paths simultaneously, or are performed sequentially which requires intelligent selection of paths.

4.5.1 Network Architecture

Repackable networks that can perform simultaneous reswitching, accommodate the maximum length of the smallest chain. On the other hand for sequential reswitching it requires intelligent algorithm to select candidate paths to be reswitched. Figure 4.3 shows a general block diagram of a symmetric repackable network capable of simultaneous reswitching. This network has N inputs and outputs and uses 2×3 switching elements and bypass network of size $\frac{N}{2} \times \frac{N}{2}$ at the outer most stage. The architecture of the network follows a recursive structure, hence each subnetworks is built with similar switching elements as well as similar bypass structure. The complexity of the bypass networks is $O(N^2)$, which is similar to that of crossbar network of size N. This repackable network is not cost effective, as the crosspoint count exceeds that of a crossbar network of similar size. Thus simultaneous reswitching topology is not an efficient repackable structure for symmetric rearrangeable networks. Hence possible sequential reswitching structures from symmetric rearrangeable networks will be looked into detail in this thesis.

4.5.1.1 Sequential Reswitching Structures

Sequential reswitching structure requires intelligent identification of the required bypass links. The minimum number of bypass links required is repoted in the literature to be two [186]. In sequential reswitching, the paths in the chain are reswitched consecutively, so that the bypass network only requires capacity for two bypass links. Paths in the smallest chain are numbered following a topdown or bottom up approach. Then paths with the odd numbers are reswitched first then the even ones or vice versa. Detail of this reswitching algorithm can be found in [182, 186]. Fig. 4.4 shows a repackable network having two bypass links. Sequential structure saves considerable number of crosspoints in building a symmetric repackable network, detail on the number of requirement crosspoints

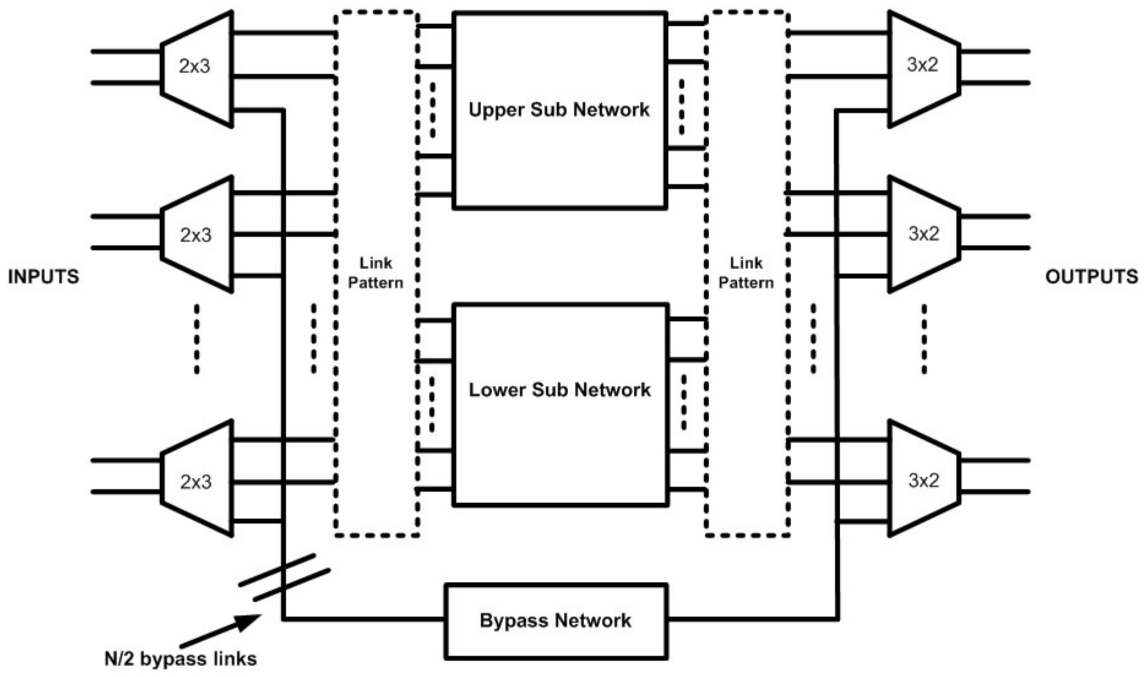


Figure 4.3: *Symmetric repackable network for simultaneous reswitching.*

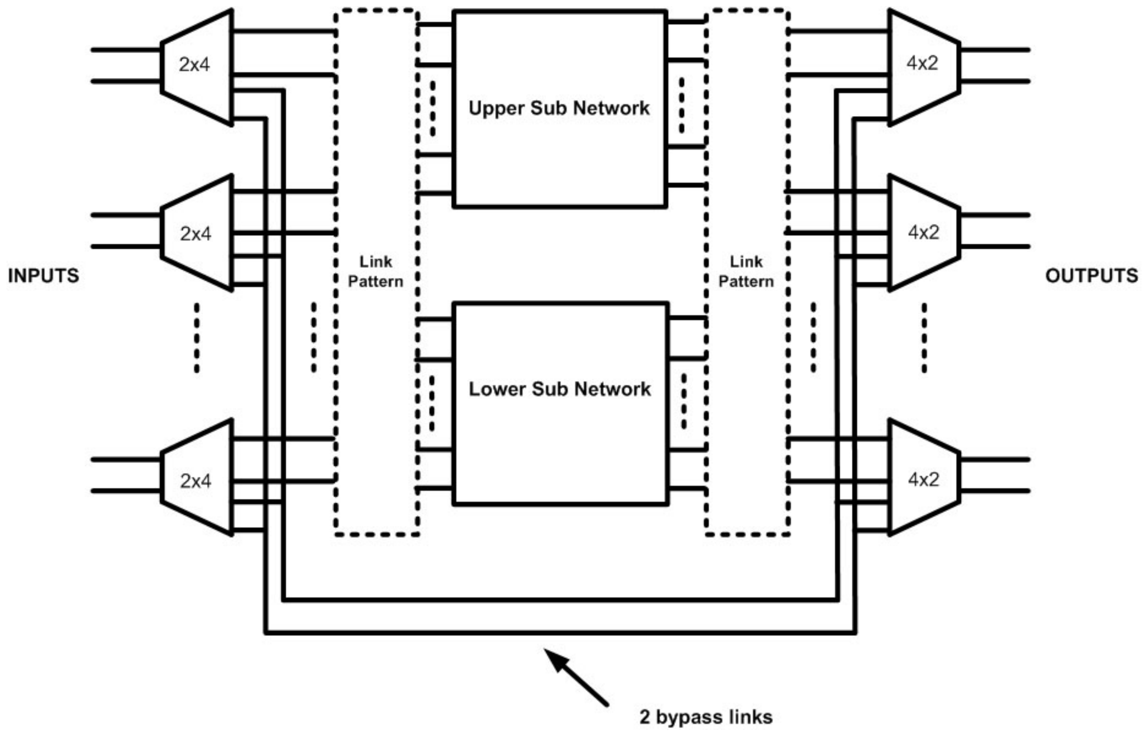


Figure 4.4: *Repackable network with* 2 *bypass links.*

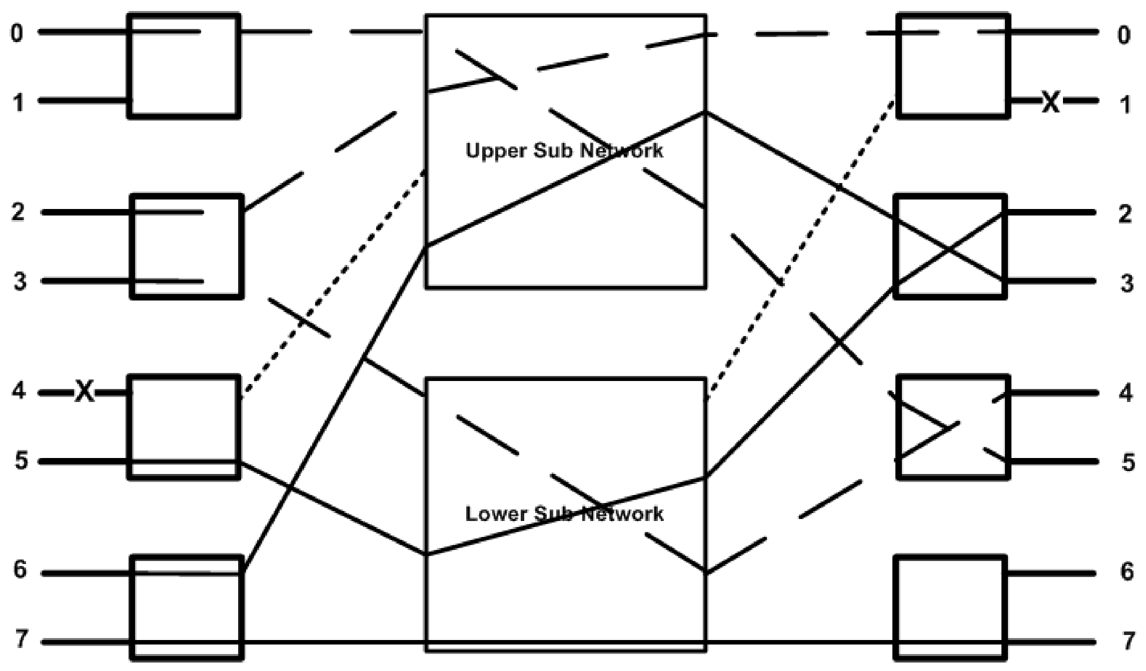


Figure 4.5: *Blocking state in the network for request* $4 \to 1$.

discussed in Section 4.6. The operation of sequential reswitching is illustrated in the example below:

Example: To start with, let's considre a network with $N = 8$ and in a blocking state as in Fig. 4.5. This particular state of the network has two chains of similar length:

- $0 \to \mathrm{U} \to 5, 3 \to \mathrm{L} \to 4, 2 \to \mathrm{U} \to 0$ and
- $5 \to \mathrm{L} \to 2, 6 \to \mathrm{U} \to 3, 7 \to \mathrm{L} \to 7$

The state of the network presented requires rearrangement to find paths for the new request $4 \to 1$, as there are no two links sharing a common middle stage switching element that can be used for the blocked request. In this case the first chain has been selected for necessary sequential reswitching with repackable network having two bypass links. The paths are numbered and the second path in the chain is chosen for reswitching and put through the bypass path. This state of the network can described as below:

- $0 \rightarrow \mathrm{U} \rightarrow 5$
- $2 \rightarrow \mathrm{U} \rightarrow 0$ and
- $3 \rightarrow \mathrm{Bypass} \rightarrow 4$

Putting the path through the bypass link allows the rearrangeable network to do the following rearrangements:

- $2 \rightarrow \mathrm{L} \rightarrow 0$ and
- $0 \rightarrow \mathrm{L} \rightarrow 5$

These rearrangements allows reswitching the request $3 \rightarrow 4$ through upper middle stage switching elements. These set of operations allows two free links that are sharing upper middle stage switching element, hence a path has been established for request $4 \rightarrow 1$. Final path map is described in Table 4.2.

Table 4.2: *Path map for input-output requests after reswitching.*

Input	Network	Output
0	L	5
3	U	4
2	L	0
4	U	1
5	L	2
6	U	3
7	L	7

4.5.1.2 Modification to Sequential Reswitching Structures

This section discusses possible modifications to the sequential reswitching structure in repackable networks. This will highlight the link level modifications that can reduce the required number of crosspoints in building the networks. This modification will address the reduction on number of links required in the bypass network. Number of bypass links required in a repackable network is directly related to the maximum length of the smallest chain. Maximum length of a chain in a blocking state can be as long as $\frac{(N-2)}{2}$. This is due to that fact that there can be at most $(N-2)$ active requests in the network to have a blocking state in the network for any new request. In worst case there will be only two chains with equal length, discussed in Section 3.1. Table 4.3 shows the worst case chain lengths for different network size.

Table 4.3: *Worst case chain length.*

Network Size	Length
8	3
16	7
32	15
64	31
128	63
256	127
512	255
1024	511

For a chain length of over 3, reswitching algorithm selects two paths with consecutive odd or even numbering from two different switching elements and puts them through the bypass links. This allows other two paths of those switching elements to rearrange in the network and then the paths in the bypass links are put back to their new positions in the rearrangeable network. But with a chain length of 3, reswitching algorithm has two options, one is to put two even numbered paths on the bypass links or put the odd numbered one to the bypass link. Putting the odd numbered one to one of the bypass links gives other two paths free links to switch their middle stage switching elements. This process only requires one bypass link rather than two. This shows that networks of size $N = 8$ give the option to modify the sequential reswitching link structure.

Inner most stages of these networks are built with 8×8 subnetworks, as a result replacing inner two bypass links repackable networks with one will save considerable number of crosspoints. A network of size $2^{\log N} \times 2^{\log N}$, will have a total of $2^{(\log N-3)}$ subnetworks of size 8×8. Fig 4.6 shows repackable network

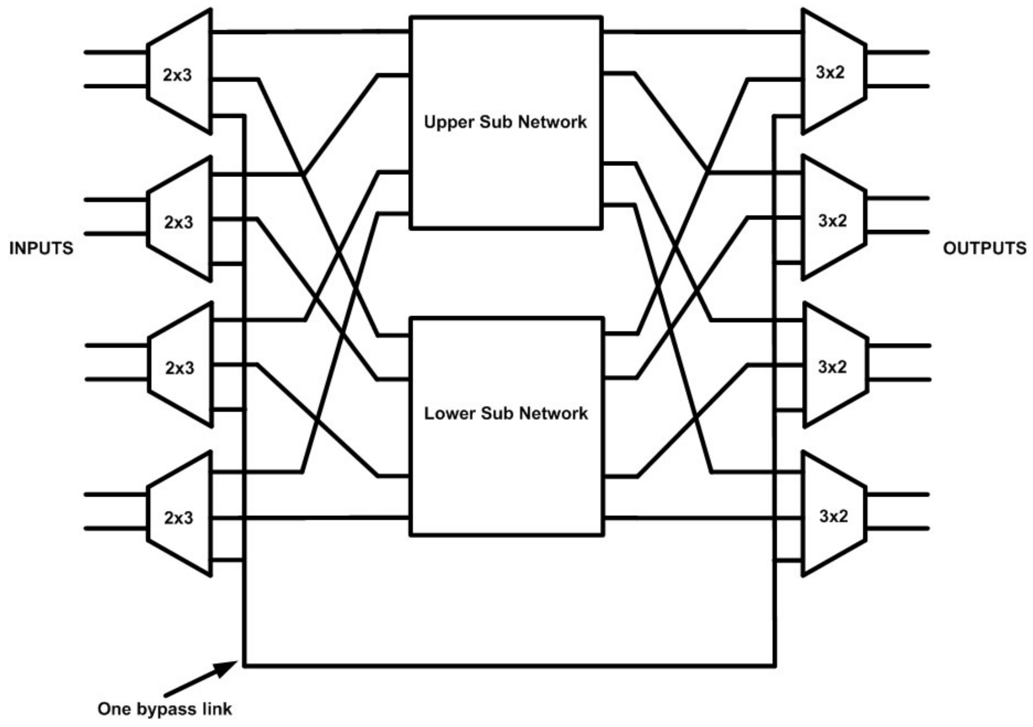


Figure 4.6: *Repackable network with one bypass link.*

build with one bypass link. The next section discusses details on the required number of crosspoints for different repcakable networks.

4.6 Crosspoint Requirements

Required crosspoints count is not an accurate metric of implementation cost for today's communication switches and routes, since components are rarely be implemented as discrete entities. However in the absence of implementation specific information on cost, crosspoints count requirements are useful indicators of overall switch complexity. An in-depth study of switch depth and the size of the building blocks has been presented in [73]. According to that study, an optimal network should have a depth of $(2M-1)$, where M is the sum of the prime factors of N (the number of switch inlets). It is also was recommended that the middle stage should have larger building blocks compared to the other stages. Using a recursive construction technique to build symmetric rearrangeable networks require $(4N\log N - 2N)$ crosspoints. The required crosspoint count a for strict-sense nonblocking network is $(6N^{3/2} - 3N)$. The required crosspoint count per input varies depending on the switching element size.

Networks built with larger switching elements have stages smaller than networks built with 2×2 switching elements, this allows the input signals to travel less number of stages to reach output port. In general a network of input size N has $(2\log_k N - 1)$ stages, where $k = \text{switching elements input size}$. This construction also saves required number of crosspoints per input in the network (but it increases routing complexity). To understand the required number of crosspoints comparisons, lets assume a network with $N = 2048$ built with 2×2 switching elements. This network will have 21 stages in total with a total required crosspoints of 86016, which gives 42 crosspoints per input. A network with $N = 2187$ constructed with 3×3 switching elements will have 13 stages. This network will require a total of 85293 crosspoints, requiring 39 crosspoints per input. A network built with 4×4 switching elements and $N = 4096$, will have a total of 11 stages. This network will require a total of 180224 crosspoints, requiring 44 crosspoint per input. Another network built with 5×5 switching elements and $N = 3125$ will have a total of 9 stages. This network will consume a total of 140625 crosspoints, requiring 45 crosspoints per input. Detail on networks built with large switching elements is discussed in Chapter 6.

For symmetric rearrangeable networks built exclusively using 2×2 switching elements, the required crosspoints count per input can be reduced by using 4×4 crossbar switches in the middle stage. This also reduces the network stages by a factor of 2. A crossbar network equivalent to 4×4 rearrangeable network is shown in Fig. 4.7. The crosspoint count in Fig. 4.7 (a) is $8 \times 3 = 24$, and in Fig. 4.7 (b) it is 4×4. Thus 8 crosspoints are saved, i.e, two crosspoints per input. The crosspoint count per input for a rearrangeable network built from 2×2 switching elements with $2^{\log N}$ input-output is $(4N\log N - 2N)$. Each network has a total of $2^{\log N - 2}$ subnetworks built with 4×4 rearrangeable networks.

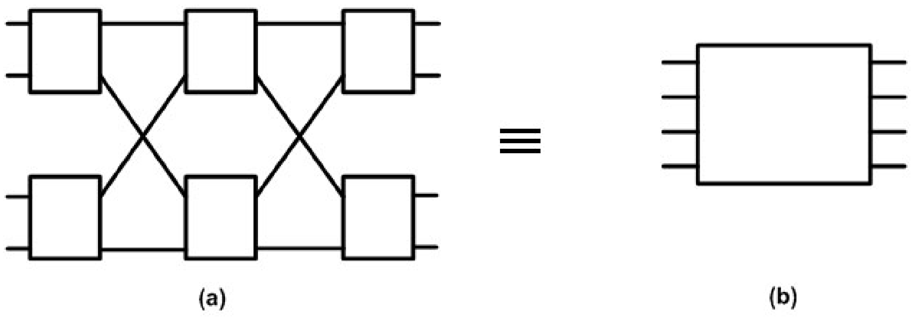


Figure 4.7: *(a).* 4×4 *rearrangeable network. (b).* 4×4 *crossbar*

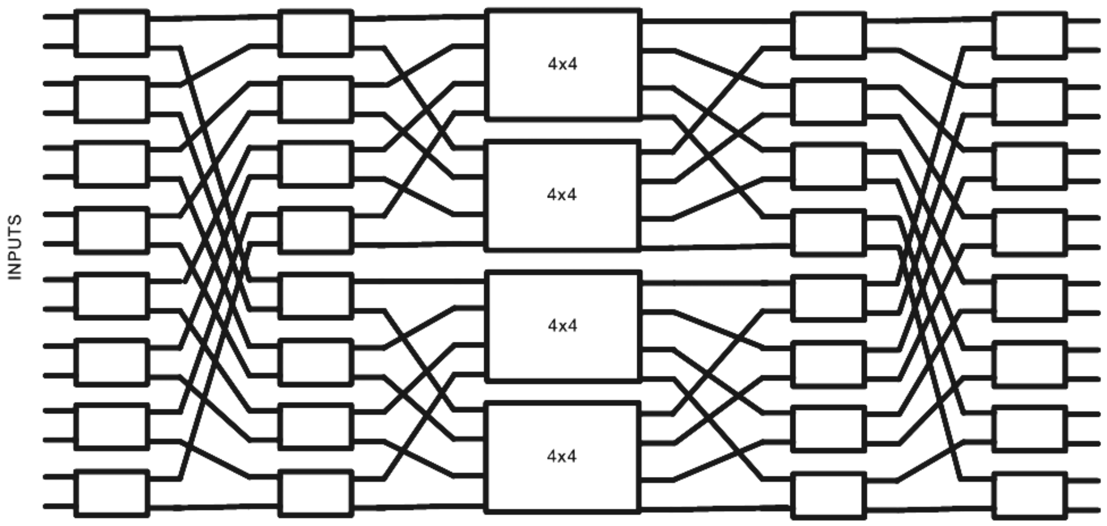


Figure 4.8: 16×16 *Beneš network with* 4×4 *crossbar in the middle stage*

For example, for a network size of $N = 16$, a recursive construction results in each input requiring 14 crosspoints, but with (4×4) crossbar in the middle stages this count reduces to 12. Similarly for $N = 32$, the numbers are 18 and 16 respectively. Replacing middle stage recursive structure with 4×4 crossbar gives overall required crosspoints as in Eqn 4.1, and Fig 4.8 shows a modified 16×16 network.

$$\begin{aligned} C_{rearr} &= (4N\log N - 2N) - (8 \times 2^{\log N - 2}) \\ &= (4N\log N - 2N) - 2^{\log N + 1} \end{aligned} \tag{4.1}$$

Using 2 bypass links for outer subnetworks and one for all 8×8 subnetworks and 4×4 crossbars to replace the 3 middle stages will require a total number of required crosspoints, given by Eqn 4.2 and shown in Fig 4.10, these networks will be called networks with *mixed bypass*(mb) links. In these networks first $(\log N - 3)$ stages will have two bypass links and the following stage will have only one bypass link. Next three middle stages will be replaced by 4×4 crossbars, hence the equation can be given as:

$$\begin{aligned} C_{mb} &= 8N(\log N - 3) + 6N + 16 \times 2^{\log N - 2} \\ &= 8N\log N - 18N + 2^{\log N + 2} \end{aligned} \tag{4.2}$$

The crosspoints requirement for a repackable network with two bypass links (with rearrangeable middle stages) given by Eqn 4.3:

$$\begin{aligned} C_{2b} &= 4N(2\log N - 2) + 2N \\ &= 8N\log N - 6N \end{aligned} \tag{4.3}$$

Crosspoints requirement per input has been given in Table 4.4. In this table a comparison is made among rearrangeable networks, repackable networks with mixed bypass which is combination of 2 and 1 bypass links also with repackable network with 2 bypass links have been carried out. It shows considerable savings in the required number of crosspoints per input in using mixed bypass method rather than 2 bypass.

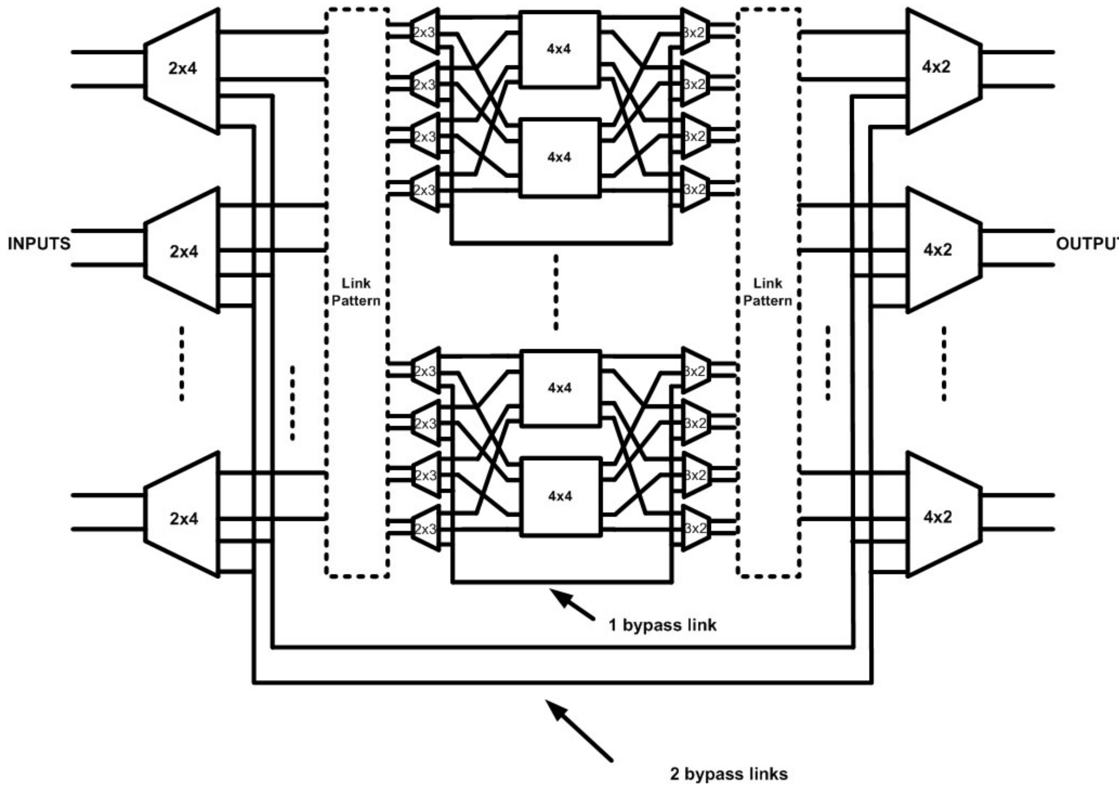


Figure 4.9: *Mixed bypass links repackable network.*

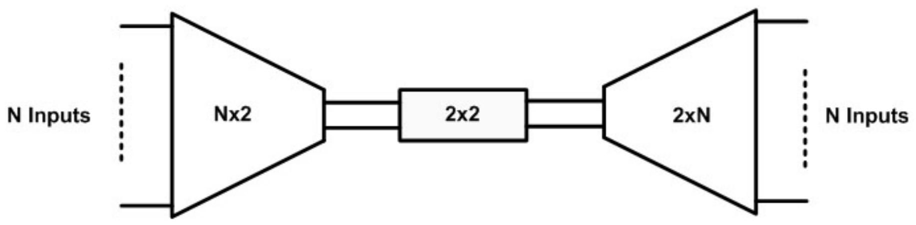


Figure 4.10: *Logical representation of bypass links.*

4.7 Hardware Structure of the Bypass links

Bypass link structures proposed in Section 4.5.1 are logical structure. Actual physical structure will be different because of the two characteristics fan-in and fan-out in shared bus architecture. Fan-in is the limit on the inputs that a logic gate can handle similarly fan-out is the limit on the number of logic gates that one single gate can drive. A logical equivalent representation of the by pass links can be given as in Fig. 4.10.

A block diagram for the actual physical structure of the two bypass links in the outermost stages is presented in this section. The fan-in and fan-out figure is technology dependent. The value 256 has been taken as the maximum number of inputs(outputs) in a concentrators (expanders) for the purpose of illustration and is representative of the capabilities of fan-in and fan-out at the time of writing [187–190]. A block diagram of hardware structure for the two bypass links is given is Fig 4.11. Design of the bypass paths involve placing the concentrators (expanders) arranged in columns. Lets assume that $l = 256$, so the number of stages in the bypass structure can be given as $2\log\frac{N}{l} + 2$. Outputs from stage k drives concentrators in stage $k + 1$. From the middle stage output from a single concentrator dives the single expander. This expander output drives the expanders in next higher stages and so on. Fig 4.11 shows the arrangement of outermost stage for 2 bypass links repackable network designed for $N = 1024$ network. Obviously in the innermost stages there will be multiple copies of similar structures for implementing inner repackable topologies of smaller sizes.

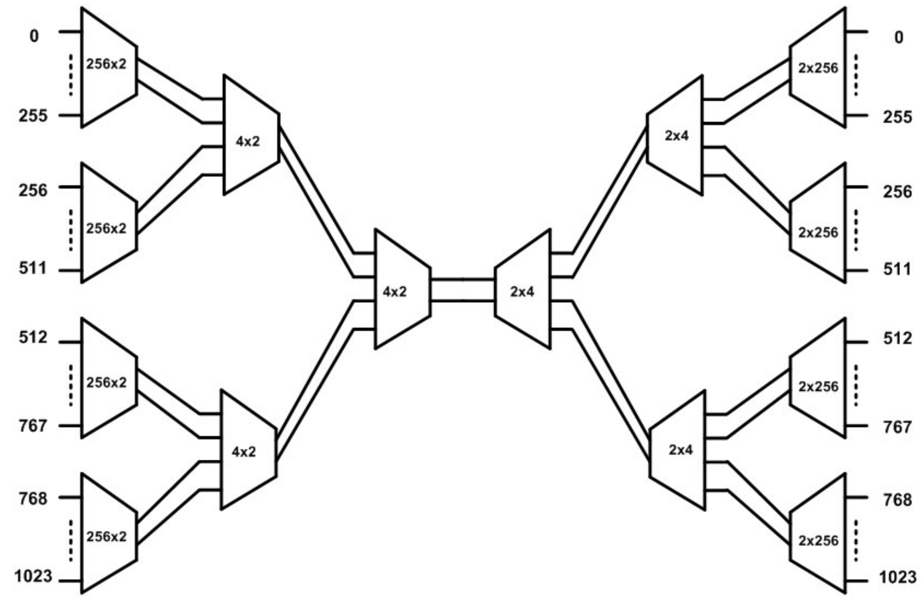


Figure 4.11: *Hardware structure of* 2 *bypass links.*

In the first stage of outermost bypass links implementation there will be $\frac{N}{256}$ concentrators of size 256×2 each. Second stage have $\frac{N}{2\times256}$ concentrators of size 4×2, next stage have a configuration of $\frac{N}{4\times256}$ concentrators of size 4×2. The

Table 4.4: *Crosspoints requirement per input for various size of networks.*

Input	Rearrangeable	Mixed Bypass		2−bypass	
		logical	256−constraint	logical	256−constraint
16	14	18	-	26	-
32	18	26	-	34	-
64	22	34	-	42	-
128	26	42	-	50	-
256	30	50	-	58	-
512	34	58	62	66	70
1024	38	66	74	74	82

configuration is recursive in the other half of the structure, so that half contains similar number of expanders with reverse size. Required crosspoints count for the designed structure in the outermost stage is 4144. In the inner stages of the network this design unless the subnetwork size reduces to 256. Fig. 4.4 shows the required crosspoints comparison when the bypass links are implemented in hardware for both mixed bypass and $2-$bypass structure. Results in the table show that mixed bypass require less hardware than $2-$bypass repackable networks.

4.8 Summary

Symmetric rearrangeable networks show promise for use in optical crossconnect (OXC) networks. Suitable switch elements of compact size and low insertion loss are emerging (e.g., using MEMS technology [191]) and the use of multi-stage techniques has been validated experimentally (e.g., the thermo-optic 32 x 32 matrix switch [192]). Multi-stage switch architectures have also been proposed in the related area of on-chip optical interconnection [193]. If these networks are to be agnostic to the signal format of the data they bear, they must treat all signals as analog signals, and thus require path continuity. In other words, unless the timing of a discrete-timed or sampled-data signal being transported through the network is known, there is no safe time at which the signal path can be disconnected other than during connection setup and teardown. The techniques described in this chapter allow the Beneš architecture (which features the minimum known number of stages for a fully-connected switch) to be modified to feature path continuity during rearrangement with only a modest increase in network complexity (less than twice to be precise), thus allowing a repackable network to be constructed at relatively low cost. Such a network provides performance indistinguishable from that of a strictly non-blocking network, provided requests arrive at a rate sufficiently low to allow the computation of the required rearrangement to repack the network , and for the sequence of switch state changes required to effect the rearrangement to be executed.

Major findings of this chapter listed below:

- This chapter provides a detail analysis of repackable network topology. Studies on the increase of required addition hardware have been discussed.
- A new repackable network topology has been presented that requires less hardware than the methods proposed in the literature. A detail comparison on the required hardware for various sizes of network have been presented for both the proposed method and the method in the literature.
- A hardware structure for implementing bypass links in the repackable networks.

CHAPTER 5

HYBRID ROUTING ALGORITHM

Beneš rearrangeable networks provide multiple paths and allow the network state to be rearranged to make way for new requests. Deterministic routing methods that guarantee finding conflict-free routing paths are complex. An alternative is to use a routing algorithm that is hybrid in nature. This can reduce the time complexity of the routing process compared to fully deterministic routing, but are not guaranteed to be non-blocking. The design of such an algorithm with acceptable blocking performance is considered in this chapter.

5.1 Preliminaries

Some notation used throughout this chapter to describe the state variables used in the algorithm code is presented below:.

Definition 5.1.1 (Subnetwork). *The subnetwork variable indicates which subnetwork is the source of an incoming signal. If* $\mathrm{Subnetwork} = 0$*, the signal is coming from the upper subnetwork and for* $\mathrm{Subnetwork} = 1$ *the signal emerges from the lower subnetwork.*

Definition 5.1.2 (STATE). *STATE is an array that holds the state of all the 2×2 switching elements in the network. A switching element can occupy one of three states. The state of the switching element at position $[j,k]$ in the network is recorded as $STATE[j,k]$ where $j = \lfloor \frac{i}{2} \rfloor$ where $0 \leq i \leq N-1$, and $0 \leq k \leq (2\log N - 2)$. If* $\mathrm{STATE}[j,k] =$ *$NULL$, the switching element is actually unconfigured. If* $\mathrm{STATE}[j,k] = 0$ *the switching element is set to perform straight through switching and when* $\mathrm{STATE}[j,k] = 1$ *it will perform a cross operation (i.e. will connect its lower input to its higher output and vice versa).*

Definition 5.1.3 (Forward Routing.) *Forward routing is the establishment of a routing path from an input i , $0 \leq i \leq (N-1)$, to the requested output port $P(i)$, where P is the input permutation. Fig 5.1(a) shows the routing from input i to output $P(i)$, $0 \leq i \leq (N-1)$, where the nodes represent the switching elements in the input and output stages, and the path through the stages is represented by a straight line.*

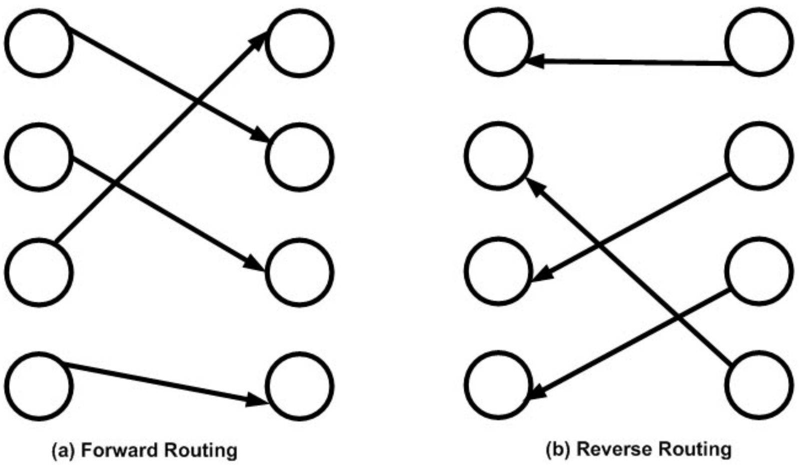


Figure 5.1: *Forward and reverse routing*

Definition 5.1.4 (Reverse Routing.) *The establishment of a path from output $P^{-1}(i)$ to input i, $0 \leq i \leq (N-1)$, is termed reverse routing. Fig 5.1(b) shows the routing from output port $P^{-1}(i)$ to input i , $0 \leq i \leq N-1$.*

Definition 5.1.5 (Neighbour Port.) *The neighbor port $Ne(i)$ to port i is the port adjacent to it in a switching element. $Ne(i) = i + 1$, if i is even otherwise it is $(i - 1)$. Evidently:*

$$Ne(i) = \begin{cases} i+1, & if\ i\ is\ even \\ i-1, & if\ i\ is\ odd \end{cases}$$

Definition 5.1.6 *(Port Mapping.) As a signal passes through the network, its address changes , in the sense that the port number in a given stage at which the signal is present differs, starting with address i at the network input, and ending with address $P(i)$ at the network output. The changes in address can be regarded as due to* port mapping. *Two types of port mapping can occur, those caused by the switching elements, and those caused by the link patterns.*

A signal presenting at input i_k in stage k will emerge at output O_k where,

$$O_k = \begin{cases} i_k, & if\ STATE[j,k] = 0 \\ Ne(i_k), & if\ STATE[j,k] = 1 \\ & with\ j = \lfloor \frac{i}{2} \rfloor \end{cases}$$

In other works the input will mapped to the output with the same address if the relevant switching element is in the straight configuration, and to its neighbour output if the switching element performs a cross operation.

The mapping performed by the link patterns differ in the first $\log N - 1$ stages of the network and the remaining stages, because of the symmetric arrangement of the link patterns. During the mapping between output port O_k of stage k and the corresponding input port in stage $k + 1$ as $M_k(O_k)$ and using a binary representation for the perfect shuffle and inverse perfect shuffle [62], it follows that:

$$M_k(O_k) = \begin{cases} (b_l \dots b_{l-k+1}\, b_0\, b_{l-k} \dots\ b_1, & 0 \le k < \log N - 1) \\ (b_l b_{l-1} ... b_{k+2} b_k b_{k-1} ... b_0 b_{k+1}), & \log N - 1 \le k < 2\log N - 2) \end{cases}$$

where the binary representation of O_k is $(b_l\ b_{l-1}\ \dots\ b_0)$ and $l = \log N - 1$.

The reverse mapping must be known to perform reverse routing. This maps input port i_{k+1} to the corresponding input port in stage k. Evidently this is port $M^{-1}(i_{k+1})$.

Definition 5.1.7 *Port Occupancy. The port occupancy, $B_k^{O_k}$ indicates whether output port O_k in stage k is "busy". It is initialized to zero, and is set to one whenever a path is assigned to it, to indicate that anyother path using that path will result in blocking.*

5.2 A Formal Description of the Looping Algorithm

This section documents the general looping algorithm for networks built with 2×2 switching elements. The looping algorithm proposed by Opferman and Tsao-Wu [125], must be applied recursively $(\log N - 1)$ times for a network with $(2\log N - 1)$ stages and N inputs and outputs. There is no formal representation of this recursive use of the looping algorithm is in the literature. Any analysis of the algorithm complexity requires such a description. Hence Algorithm 1 shows the general looping algorithm using the notation presented in Section 5.1, for a network of size N. Given a permutation $P_{0:(N-1)}$ to be implemented, the algorithm starts with input 0 and carry on setting all the switching elements between stages 0 and $(2\log N - 2)$.

Algorithm 1 starts by setting all the switching elements of the networks in to NULL state at Line 3. Line 7 calls function UPDATEPERM(), which modifies P to reflect input at stage k and requested output at stage $2\log N - 2 - k$. In Line 10, algorithm checks the status of requested switching element for requested output port $P(i)$ at stage $2\log N - 2 - k$. If that switching element is in a NULL state, algorithm calls function ROUTELOOP() to set the switching elements at stage k and $2\log N - 2 - k$. An example of the execution of the looping algorithm is given in Example 5.2.1.

Algorithm 1 : Algorithm Looping

```
INPUT: Permutation $P_{0:N-1}$.
OUTPUT: Matrix state of size $\frac{N}{2} \times (2\log N - 1)$.
 1: for $k = 0$ to $2\log N - 2$ do
 2:     for $j = 0$ to $\frac{N}{2} - 1$ do
 3:         STATE$[j, k]$ ←NULL
 4:     end for
 5: end for
 6: for $k = 0$ to $\log N - 2$ do
 7:     UPDATEPERM(k)
 8:     for $i = 0$ to $N - 1$ do
 9:         if STATE$[\lfloor P(i)/2 \rfloor, 2\log N - 2 - k]$=NULL then
10:             ROUTELOOP(i,k)
11:         end if
12:     end for
13: end for
```

5.2.1 Example

Let $N = 8$ and the input permutation $P_{(0:7)} = (0\ 6\ 4\ 5\ 2\ 1\ 3\ 7)$. The mapping between inputs and outputs is thus $0 \rightarrow 0$, $5 \leftarrow 1$, $6 \rightarrow 3$, and so on. Initially let's set $\text{STATE}[0, 0] = 0$, hence $\text{Subnetwork} = 0$ and the requested port is even, so it follows that $[(P(0)\ mod\ 2) \oplus Subnetwork] = 0$, and $\text{STATE}[0, 4]$ will be 0 or straight through. Because $\text{STATE}[0, 4] = 0$, the signal goes through the lower subnetwork from output port 1 at SE[0,4] to input port 5 at SE[2,0]. The values for variables set $\text{STATE}[2, 0]$ to 0. This process continues for all the ports of the switching elements at stage 0 and 4. Fig 5.2 shows the switching element settings in stage 0 and $(2logN - 2)$, this same process is applied in two sub-networks and hence a path is established between all input-output ports pairs.

Although the looping algorithm grantees zero blocking probability, it comes at the cost of a recursive switching element setup. A permutation of size N is divided into two $\frac{N}{2}$ parts and one part follows forward routing (from input towards

Function 2 : UPDATEPERM(k)

```
1:  if $k = 0$ then
2:      Return //Noting to do.$P$ is for outermost stages.
3:  else
4:      for $i = 0\ to\ \frac{N}{2} - 1$ do
5:          if STATE[$\lfloor \frac{i}{2} \rfloor, k$] $= 1$ then //Port mapping by SE
6:              $s(i) \leftarrow N_e(i)$
7:          else
8:              $s(i) \leftarrow i$
9:          end if
10:         $s(i) \leftarrow M_k((s(i))$ //Port mapping by link pattern
11:         if STATE[$\lfloor \frac{i}{2} \rfloor, 2\log N - 2 - k$] $= 1$ then
12:             $d(i) \leftarrow Ne(P(i))$
13:         else
14:             $d(i) \leftarrow P(i)$
15:         end if
16:         $d(i) \leftarrow M^{-1}_{2\log N - 2 - k}(d(i))$
17:     end for
18:     for $i = 0\ to\ N - 1$ do
19:         $P(s(i)) \leftarrow d(i)$
20:     end for
21:     Return
22: end if
```

the outputs) and the other half does the inverse rooting (from output towards input). Inverse routing takes up most of the time as it requires the algorithm to determine the available neighbour ports and then determine the corresponding input port. The increase in execution time with N shows that the routing process is time consuming, which makes it unscalable. Section 5.7 demonstrates the execution time of this algorithm for various values of N. The required execution time of the looping algorithm motivates looking for an alterative routing method which might give sub-optimal blocking performance but requires less time to execute. Such an alternative method is described in Section 5.3

5.3 Random Routing

Random routing method is proposed in [141], where the decision making is based for the random probability of setting the switching elements to a straight through

Function 3 : ROUTELOOP(i, k)

```
1: STATE[$\lfloor i/2 \rfloor, k$] $\leftarrow 0$ //Determine Subnetwork
2: home$\leftarrow i$
3: if [STATE[$\lfloor i/2 \rfloor, k$] $\oplus$ $(i\ mod\ 2)] = 0$ then
4:     Subnetwork $\leftarrow 0$
5: else
6:     Subnetwork $\leftarrow 1$
7: end if
8: repeat//Until a chain is routed
9:     if [$(P(i)\ mod\ 2) \oplus Subnetwork] = 0$ then
10:        STATE[$\lfloor P(i)/2 \rfloor, 2\log N - 2 - k] \leftarrow 0$
11:    else
12:        STATE[$\lfloor P(i)/2 \rfloor, 2\log N - 2 - k] \leftarrow 1$
13:    end if
14:    if S thenubnetwork= $1$ //Convert Subnetwork
15:        Subnetwork$\leftarrow 0$
16:    else
17:        Subnetwork$\leftarrow 1$
18:    end if
19:    $O \leftarrow Ne(P(i))$
20:    $i \leftarrow P^{-1}(i)$
21:    if $[(i\ mod\ 2) \bigoplus Subnetwork] \leftarrow 0$ then //Set input SE state
22:        STATE[$\lfloor i/2 \rfloor, k] \leftarrow 0$
23:    else
24:        STATE[$\lfloor i/2 \rfloor, k] \leftarrow 1$
25:    end if
26:    if Subnetwork= $1$ then
27:        Subnetwork$\leftarrow 0$
28:    else
29:        Subnetwork$\leftarrow 1$
30:    end if
31:    $i \leftarrow Ne(i)$
32: until (i=home)
```

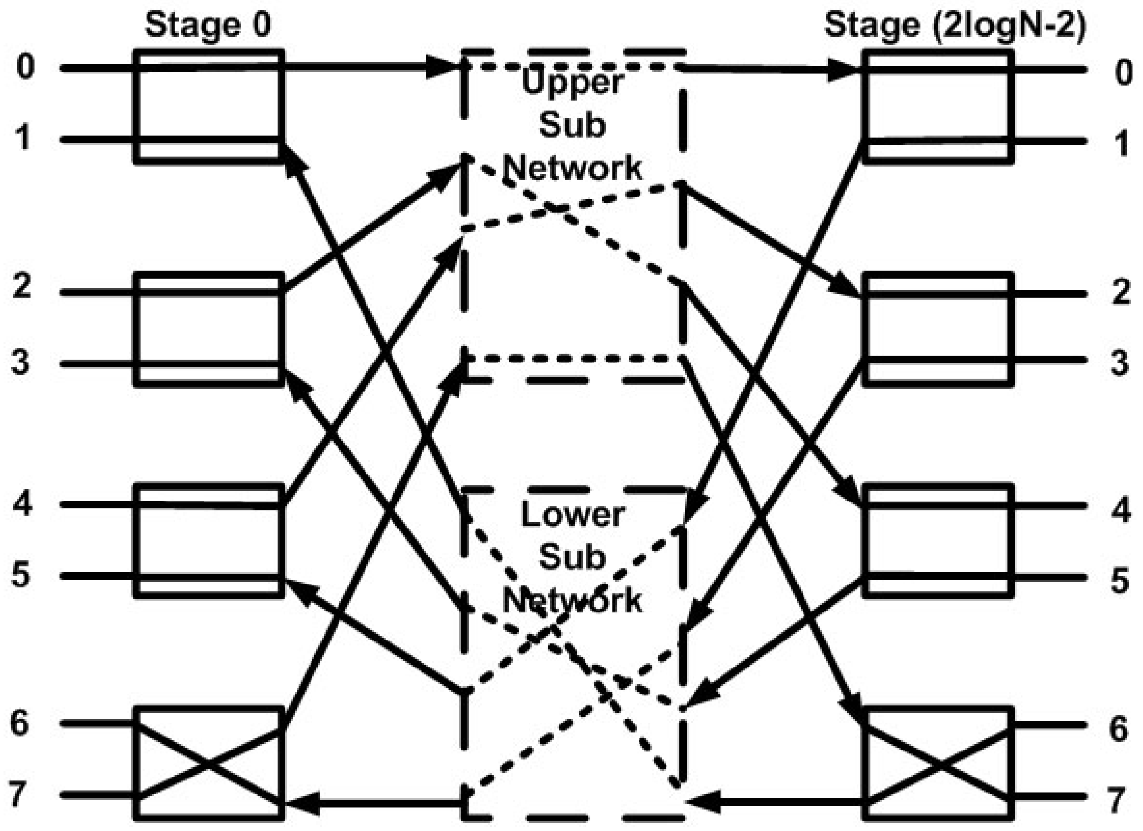


Figure 5.2: *Switching element settings in stage* 0 *and* $(2logN - 2)$

or cross state. This eliminates the need of using a complex mathematical method (such as graph theory, matrix manipulation, etc), and does not requires recursive decision making like the looping algorithm. This method provides non-zero blocking probabilities, but gives a quick routing solution. Fig 5.3 shows the performance of random routing for networks of various sizes.

The simulated results for random routing clearly show that this cannot be used in any practical application because of poor blocking performance. So this method of routing can not be considered as an alternative to the looping algorithm. This highlights the need for an algorithm that can provide a lower blocking probability than random routing but much faster execution time than the looping algorithm. Such an algorithm is presented in next section.

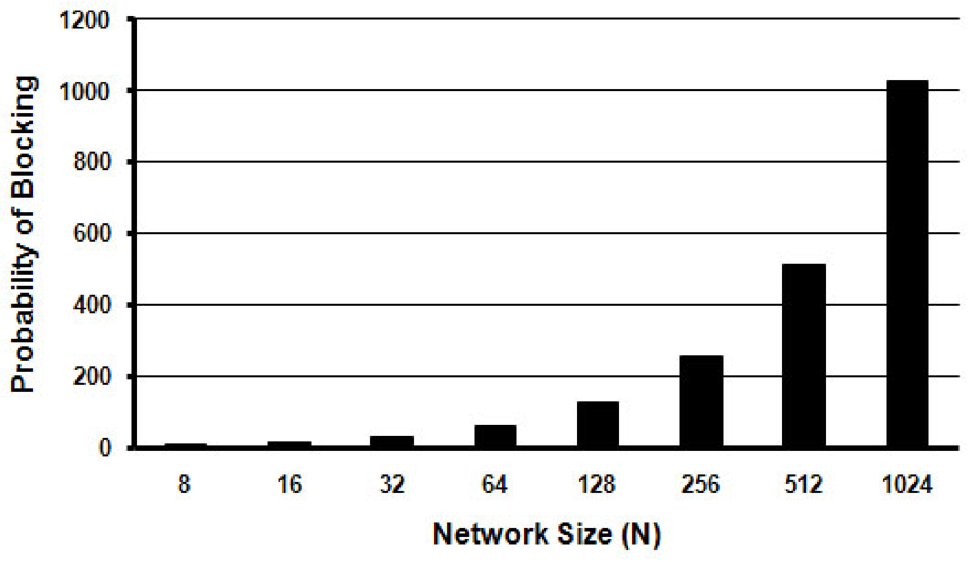


Figure 5.3: *Blocking probability for random routing.*

5.4 Adaptive Routing

This section studies the possibilities of designing an algorithm that can provide sub-optimal blocking performance, superior than the random algorithm. Also the execution time is quicker than that of the looping algorithm.

The design of the algorithm follows the looping algorithm, in applying forward and reverse routing. In the distributed part of this algorithm it selects paths depending on the state of the switching element. If it is in an idle state, it is set to the straight through state and the signal goes to the next stage. If the switching elements is already set to a state, the unused output port is used to go to the next stage. Once the signal finds a middle stage switching element, the rest of the routing is bit controlled. Any conflict that may occure is resolved by choosing an alternative connecting path. Detail of the routing methods are given in Section 5.4.1 and 5.4.2

5.4.1 Forward Routing Phase

The Forward routing part of this algorithm contains number of functions that work in conjunction. Algorithm Forward initialized all the switching elements in to a NULL state and call function FORWARD which controls the routing. Variable i_k is the input to switching element SE$[\lfloor M(i_k)/2 \rfloor, k]$ at stage k, where $M(i_k)$ is the mapping port of i_k at stage k. In line 19, j_k holds the temporally physical port number in the underlying hardware, that i_k maps to, in the next stage.

Algorithm 4 : Algorithm Adaptive

```
INPUT: Permutation $P_{0:N-1}$
OUTPUT: Matrix state of size $\frac{N}{2} \times (2\log N - 1)$.
1: for $k = 0$ to $2\log N - 2$ do
2:     for $j = 0$ to $\frac{N}{2} - 1$ do
3:         STATE$[j, k]$ ←NULL
4:     end for
5: end for
6: $k \leftarrow 0$
7: $i_k \leftarrow 0$
8: FORWARD($i_k$, $k$)
```

The variables $B_k^{O_k}$ in BITFRD or in BITREV set to 1 when output port O_k or input port j_k is already used. If variable R in Line 13 or 18 is TRUE current loop exists and in Line 25 BITFRD calls function CONFLICTFRD to resolve any conflict in the switching stages because of the operations in previous stages. In BITFRD, line 33 randomly selects an unused input at stage 0 if SE$[\lfloor P(i_k)/2 \rfloor, k]$ has no free output port. Variable B is a counter that counts $(\log N - 1)$ bits of the binary digits equivalent to the requested output port number and is decremented after processing of each stage in the bit controlled part. C stores the binary presentation of requested the output port number.

CONFLICTFRD (i_k) finds alternative path for a request i_k that is blocked at stage k. Once a function call occurred, this function starts searching for an alternative path for request $i_k \rightarrow P(i_k)$ from stage $\log N - 2$. If it is not successful in finding an alternative path, it goes back one stage and looks again. If the search ends up at stage 0, the function drops the request as no alternative path is avail-

Function 5 : FORWARD ($i_k,\ k$)

```
1:  if $i_k = NULL$ then
2:      Exit
3:  end if
4:  if $k = 0$ then
5:      if STATE$[\lfloor i_k/2 \rfloor, k] = NULL$ then
6:          STATE$[\lfloor i_k/2 \rfloor, k] \leftarrow 0$
7:          $j_k \leftarrow M(i_k)$
8:      else
9:          if STATE$[\lfloor i_k/2 \rfloor, k] = 0$ then
10:             $j_k \leftarrow M(i_k)$
11:         else
12:             $j_k \leftarrow Ne(M(i_k))$
13:         end if
14:         $k \leftarrow k + 1$
15:     end if
16:     repeat
17:         if STATE$[\lfloor M(i_k)/2 \rfloor, k] = NULL$ then
18:             STATE$[\lfloor M(i_k)/2 \rfloor, k] \leftarrow 0$
19:             $j_k \leftarrow M(i_k)$
20:         else
21:             if STATE$[\lfloor M(i_k)/2 \rfloor, k] = 0$ then
22:                 $j_k \leftarrow M(i_k)$
23:             else
24:                 $j_k \leftarrow Ne(M(i_k))$
25:             end if
26:         end if
27:         $B_k^{j_k} \leftarrow 1$;$k \leftarrow k + 1; M(i_k) \leftarrow j_k$
28:     until ($k <$ log$N - 1$)
29:     if $k =$ (log$N - 1$) then
30:         CALL BITFRD($i_k,\ M(i_k)$)
31:     end if
32: end if
```

Function 6 : BITFRD $(i_k,\ M(i_k))$

```
1: $B \leftarrow (\log N - 1)$
2: $C[\log N - 1] \leftarrow Binary\ (P(i_k))$
3: **repeat**
4:     **if** STATE$[\lfloor M(i_k)/2 \rfloor, k] = NULL\ \ \&\ \ [C[B] \oplus (M(i_k)\ mod\ 2)] = 0$ **then**
5:         STATE$[\lfloor M(i_k)/2 \rfloor, k] = 0\ \ \&\ \ j_k \leftarrow M(i_k)$
6:     **else**
7:         STATE$[\lfloor M(i_k)/2 \rfloor, k] = 1\ \ \&\ \ j_k \leftarrow Ne(M(i_k))$
8:     **end if**
9:     **if** STATE$[\lfloor M(i_k)/2 \rfloor, k] \neq NULL$ **then**
10:        **if** $C[B] \oplus (M(i_k)\ mod\ 2) = 1\ \ \&\ \$ STATE$[\lfloor M(i_k)/2 \rfloor, k] = 0$ **then**
11:            Blocking at SE$[\lfloor M(i_k)/2 \rfloor, k]$
12:            $R \leftarrow TRUE$
13:            Exit Loop
14:        **else**
15:            **if** $C[B] \oplus (M(i_k)\ mod\ 2) = 0\ \ \&\ \$ STATE$[\lfloor M(i_k)/2 \rfloor, k] = 1$ **then**
16:                Blocking at SE$[\lfloor M(i_k)/2 \rfloor, k]$
17:                $R \leftarrow TRUE$
18:                Exit Loop
19:            **end if**
20:        **end if**
21:    **end if**
22:    $B \leftarrow B - 1$
23:    $k \leftarrow k + 1\ \ \&\ \ M(i_k) \leftarrow\ j_k$
24: **until** $(B < 0)$
25: **if** $R = TRUE$ **then**
26:     CALL CONFLICTFRD($i_k$)
27: **else** $B_k^{j_k} \leftarrow 1$
28: **end if**
29: **if** $B < 0$ **then**
30:     $O_k \leftarrow Ne(P(i))$
31:     **if** $B_k^{O_k} = 1$ **then**
32:         $k \leftarrow 0$
33:         $i_k \leftarrow Random(B_k^{i_k} \neq 1)$
34:         Call FORWARD $(i_k,\ k)$
35:     **else**
36:         Call REVERSE $(O_k, k)$
37:     **end if**
38: **end if**
```

able for that request. Once a request is dropped, the function calls algorithm FORWARD which starts with another free input at stage 0.

Function 7 : CONFLICTFRD(i_k)

```
1: $k \leftarrow \log N - 2$
2: if $[M^{-1}(B_k^{i_k}) \oplus Ne(M^{-1}(B_k^{i_k}))] = 1$ then
3:     if STATE$[\lfloor M^{-1}(i_k)/2 \rfloor, k] = 0$ then
4:         STATE$[\lfloor M^{-1}(i_k)/2 \rfloor, k] \leftarrow 1$
5:     else
6:         STATE$[\lfloor M^{-1}(i_k)/2 \rfloor, k] \leftarrow 0$
7:     end if
8: else
9:     $k \leftarrow k - 1$
10:    if $k < 0$ then
11:        drop the request ($i_k$)
12:        $i_k \leftarrow$ Random ($B_k^{i_k} \neq 1$)
13:        Exit Loop
14:    end if
15: end if
16: if then$(i_k = NULL)$
17:    CALL FORWARD ($i_k \leftarrow Random(B_k^{i_k} \neq 1), 0$)
18: else
19:    CALL FORWARD ($i_k, k$)
20: end if
```

5.4.2 Reverse Routing Phase

Algorithm REVERSE uses the neighbour port of $P(i)$ that is O_k as an input at stage $2\log N - 2$. The basic method is similar to that of algorithm FORWARD, the only difference being that algorithm REVERSE starts from the output stage and proceeds towards stage 0. This algorithm checks the status of target switching element and if it is NULL sets it to the straight through position.

If the switching element is already set, it uses the unused port to reach the next lower stage. In a conflicting or blocking situation, algorithm REVERSE calls function CONFLICTREV to overcome any blocking by selecting an alternative path for the blocked request. As with function CONFLICTFRD, this function also drops requests where there are no alternative paths.

Function 8 : REVERSE (O_k, k)

```
repeat
    if $k = (2\log N - 2)$ then
        if STATE$[\lfloor O_k/2 \rfloor, k] = 0$ then
            $j_k \leftarrow M(O_k)$
        else
            $j_k \leftarrow M(Ne(O_k))$
        end if
    else
        if STATE$[\lfloor M(O_k)/2 \rfloor, k] = NULL$ then
            STATE$[\lfloor M(O_k)/2 \rfloor, k] \leftarrow 0$
            $j_k \leftarrow M(O_k)$
        else
            if STATE$[\lfloor M(O_k)/2 \rfloor, k] = 0$ then
                $j_k \leftarrow M(O_k)$
            else
                $j_k \leftarrow Ne(M(O_k))$
            end if
        end if
    end if
    $B_k^{O_k} \leftarrow 1$;$M(O_k) \leftarrow j_k$;$k \leftarrow k - 1$
until $(k > (\log N - 1))$
if $k = (\log N - 1)$ then
    CALL BITREV($O_k$, $M(O_k)$)
end if
```

Function 9 : BITREV ($O_k,\ M(O_k)$)

```
$B \leftarrow (\log N - 1)$
$C[\log N - 1] \leftarrow Binary\ (P(O_k^{-1}))$
**repeat**
    **if** STATE$[\lfloor M(O_k)/2 \rfloor, k] = NULL$ & $[C[B] \oplus (M(O_k)\ mod\ 2)] = 0$ **then**
        STATE$[\lfloor M(O_k)/2 \rfloor, k] = 0$ & $j_k \leftarrow M(O_k)$
    **else**
        STATE$[\lfloor M(O_k)/2 \rfloor, k] = 1$ & $j_k \leftarrow Ne(M(O_k))$
    **end if**
    **if** $(M(O_k)\ mod\ 2) \oplus C[B] = 1$ & STATE$[\lfloor M(O_k)/2 \rfloor, k] = 0$ **then**
        Blocking at SE$[\lfloor M(O_k)/2 \rfloor, k]$
        $R \leftarrow TRUE$
        Exit Loop
    **else**
        **if** $(M(O_k)\ mod\ 2) \oplus C[B] = 0$ & STATE$[\lfloor M(O_k)/2 \rfloor, k] = 1$ **then**
            Blocking at SE$[\lfloor M(O_k)/2 \rfloor, k]$
            $R \leftarrow TRUE$
            Exit Loop
        **end if**
    **end if**
    $k \leftarrow k - 1$ & $M(O_k) \leftarrow j_k$
    $B \leftarrow B - 1$
**until** $(B < 0)$
**if** $R = TRUE$ **then**
    CALL CONFLICTREV $(O_k, k)$
**end if**
**if** $B < 0$ **then**
    $i_k \leftarrow Ne(P^{-1}(O_k))$
    **if** $B_k^{i_k} = 1$ **then**
        $i_k \leftarrow Random(B_k^{i_k} \neq 1)$
        CALL FORWARD$(i_k, k)$
    **else**
        CALL FORWARD $(i_k, k)$
    **end if**
**end if**
```

Function 10 : CONFLICTREV(O_k, k)

```
1: $k \leftarrow \log N$
2: if $[M^{-1}(B_k^{O_k}) \oplus Ne(M^{-1}(B_k^{O_k}))] = 1$ then
3:     if STATE$[\lfloor M^{-1}(O_k)/2 \rfloor, k] = 0$ then
4:         STATE$[\lfloor M^{-1}(O_k)/2 \rfloor, k] \leftarrow 1$
5:     else
6:         STATE$[\lfloor M^{-1}(O_k)/2 \rfloor, k] \leftarrow 0$
7:     end if
8: else
9:     $k \leftarrow k + 1$
10:    if $k > (2\log N - 3)$ then
11:        drop the request $(O_k)$
12:        Exit Loop
13:    else
14:        return $(O_k, k)$
15:    end if
16: end if
17: CALL FORWARD $(i_k \leftarrow Random(B_k^{i_k} \neq 1), 0)$
```

FUNCTION CONFLICTREV is similar to that of function CONFLICTFRD, but works in the reverse direction. Also when there us a failure to find an alternative path this function, after dropping the blocked request, calls algorithm FORWARD to start with any unprocessed input at stage 0.

5.4.3 Example

Fig. 5.4 shows the network setup using adaptive routing method for a permutation $P_{(0:7)} = (5\ 1\ 7\ 4\ 3\ 2\ 0\ 6\ )$ where dotted circles indicate conflict and hence alternative path search was required. Algorithm FORWARD starts with request $1 \rightarrow 5$, where the routing tags for first 3 switching stages are generated using this algorithm. Once the algorithm reaches stage 2, hence it calls algorithm BITFRD for bit controlled routing in the forward direction and reaches switching element 2, hence output port 5. Algorithm REVERSE starts from output port 4 and finds a path for request $4 \rightarrow 3$. The rest of the routing procedure involves a repetition of algorithm FORWARD and REVERSE along with the corresponding bit controlled part. In the reverse routing phase, request $1 \rightarrow 1$ finds a conflict at stage 2, and

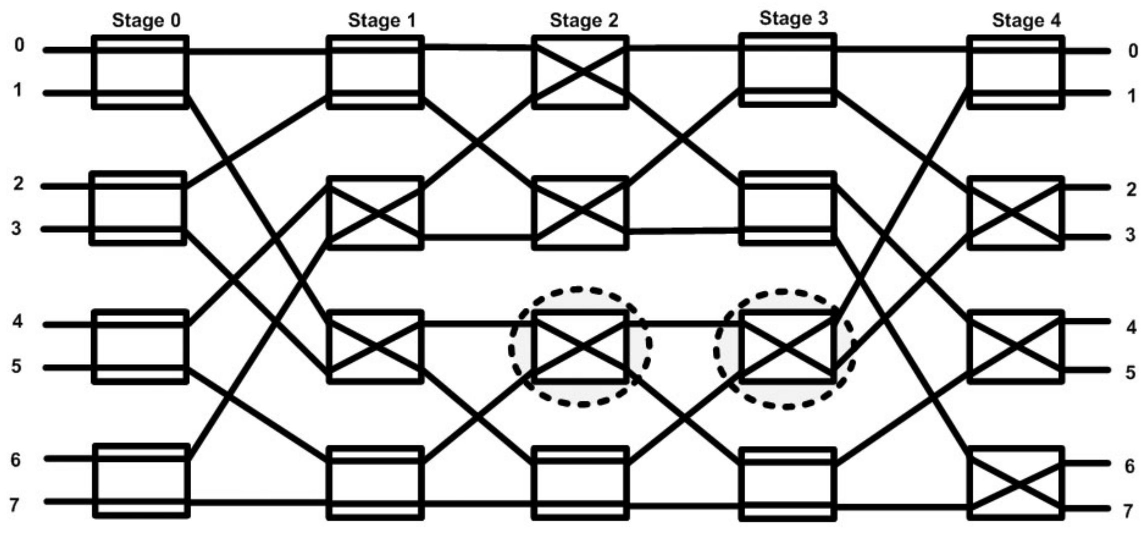


Figure 5.4: *Network setup using adaptive method for* $P_{(0:7)} = (5\ 1\ 7\ 4\ 3\ 2\ 0\ 6\ )$.

hence function CONFLICTREV is called to search for any alternative route, and so to resolve the conflict. The dotted circle indicates the conflicts in the network.

5.5 Simulation Results

This section illustrates the results for various network sizes using the method proposed in Section 5.4 and also compares them with the blocking probabilities of random routing and the execution time of the looping algorithm. All the simulators have been tested in an Intel(R) Core(TM)2 Quad 2.40 GHz CPU computer with a memory of 8GB. Simulation studies have been carried in this thesis to show the performance of the proposed algorithm in this chapter. Fig 5.5 shows the blocking probability for random routing and adaptive routing. Results show considerable improvements in the blocking probabilities for the adaptive method. This suggest the effectiveness of the adaptive routing in preference to random routing.

Fig 5.6 compares the execution time of the looping algorithm to that of the adaptive routing. The two graphs shows similar results with the adaptive method

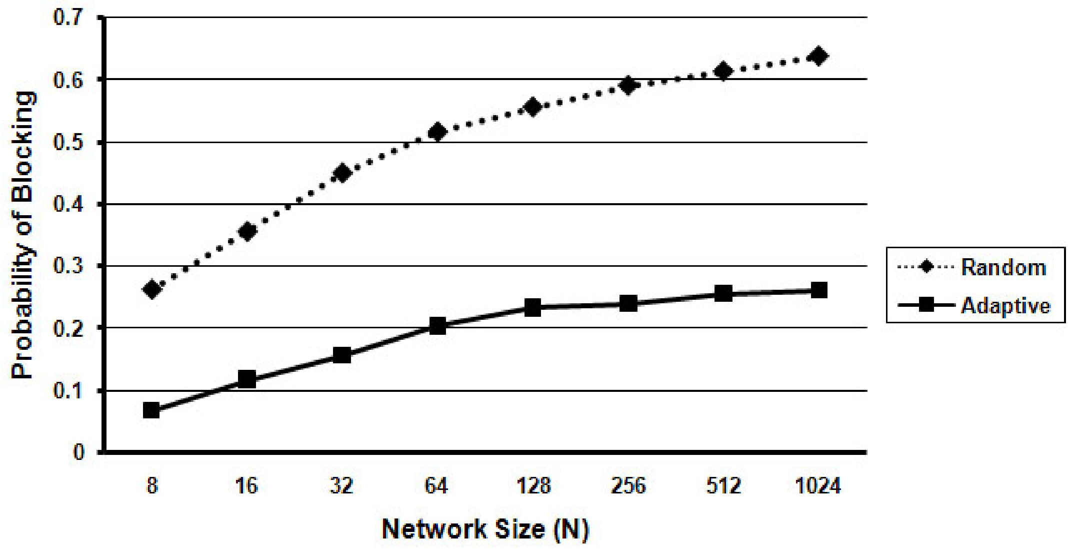


Figure 5.5: *Probability of blocking using random and adaptive routing .*

requiring longer than the looping algorithm for larger N. These outcomes point out that the adaptive method can give better blocking performance than the random routing, but it fails to satisfy the need for an algorithm that is faster than the looping algorithm. This drawback of the adaptive routing requires further investigation so as to obtain improvements in its execution time. The next section proposes some modifications to the method that reduce the execution time considerably without increasing the blocking performance.

5.6 Hybrid Routing

Further studies on the adaptive method shows that the required number of path search per input taking up a long time and is the factor that limits its performance compared to the looping algorithm. This motivates a class of routing algorithm which combines two approaches:

- An "outer" algorithm, which is applied to the outermost r stages, $0 \leq r \leq (\log N - 2)$, of the network (this will typically be the looping algorithm).

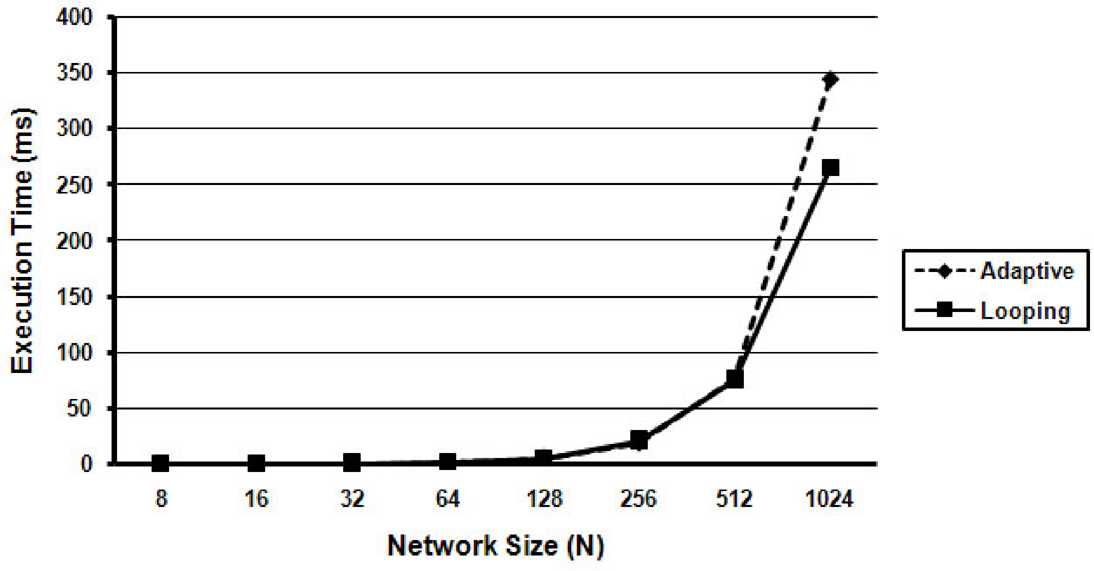


Figure 5.6: *Execution time for adaptive and the looping algorithm.*

- An "inner" algorithm, which is applied in all the remaining stages. This algorithm trades off performance against execution time.

An obvious choice for the inner algorithm is the random algorithm, but this gives poor performance as shown in the performance graph in Section 5.3. The algorithm proposed in Section 5.4 shows promising blocking performance but suffers from high execution time. This is due to the fact that for large networks this method searches all possible alternative paths before dropping a request. One solution to this is to limit reduce the required number of paths searched by using the looping algorithm in the outer most stages of the network. This would allow the inner routing method to search only one subnetwork for possible alternative paths, and reduces required path search count.

The design of the hybrid algorithm requires modifications to the algorithms and functions proposed in Section 5.4, and they are listed below:

- Set $k \leftarrow 1$ in Algorithm 5 at Line 6.
- Set $B \leftarrow (\log N - 2)$ in function 6 at Line 1.
- In function CONFLICTFRD, if $k < 1$, then the function exits.

- Set $k \leftarrow (2\log N - 3)$ in function 8 at Line 2.
- Set $B \leftarrow (\log N - 2)$ in function 9 at Line 1
- In function CONFLICTREV, if $k > (2\log N - 3)$, then the function exits.

5.7 Simulation Results

The performance of the methods described are measured using three different metrics: blocking probabilities, required path search and setup time. The variable r for hybrid routing is set to 1, which means the hybrid method checks for possible alternative paths in to stage 1 of the network. The results have been compared with Random routing, Adaptive routing and with the looping algorithm. Fig 5.7 shows the performance graphs for a full input occupancy network. The results show that hybrid and adaptive have similar blocking probabilities throughout the observation window. The blocking probabilities shows little variation for smaller values of N where hybrid routing has a slight edge over the adaptive approach. Comparing these two methods with the Random routing shows that both hybrid and adaptive outperforms Random routing for all values of N by a large margin. So for full a occupancy network when probability of blocking is the performance measuring tool, hybrid and adaptive give similar performance with few exceptions, but overall they are always superior than Random routing.

Fig 5.8 and 5.9 show the performance of the three methods for 50% and 75% input occupancies. These two graphs show a reduction in the blocking probability as the number of active inputs is less compared to full occupancy network. This opens more unoccupied routes inside the network resulting in less chance of paths overlapping. The nature of the curve is almost identical to that in fig. 5.7. As in the full occupancy scenario, Random routing has the highest loss probabilities.

The required number of path searches is another performance analysis met-

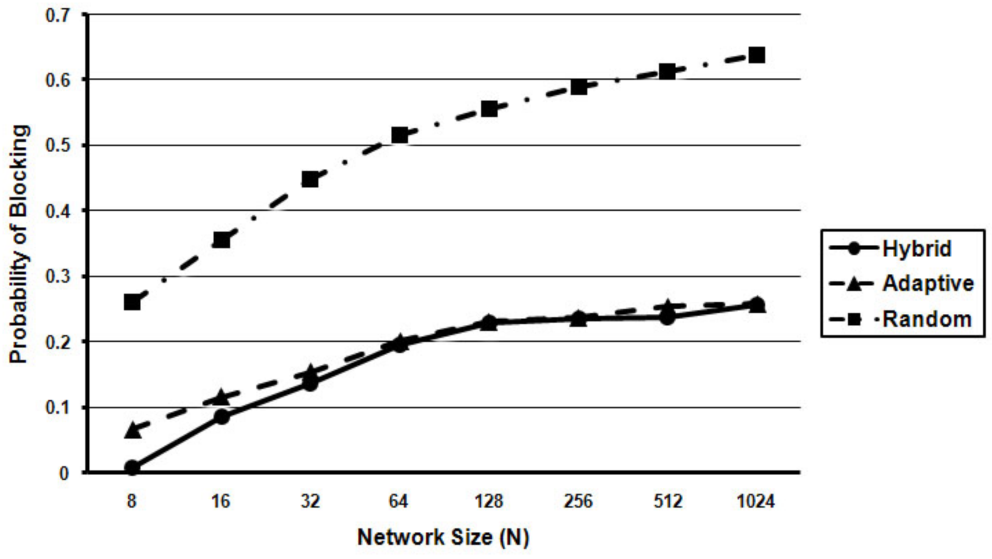


Figure 5.7: *Performance of three different methods for full input occupancy.*

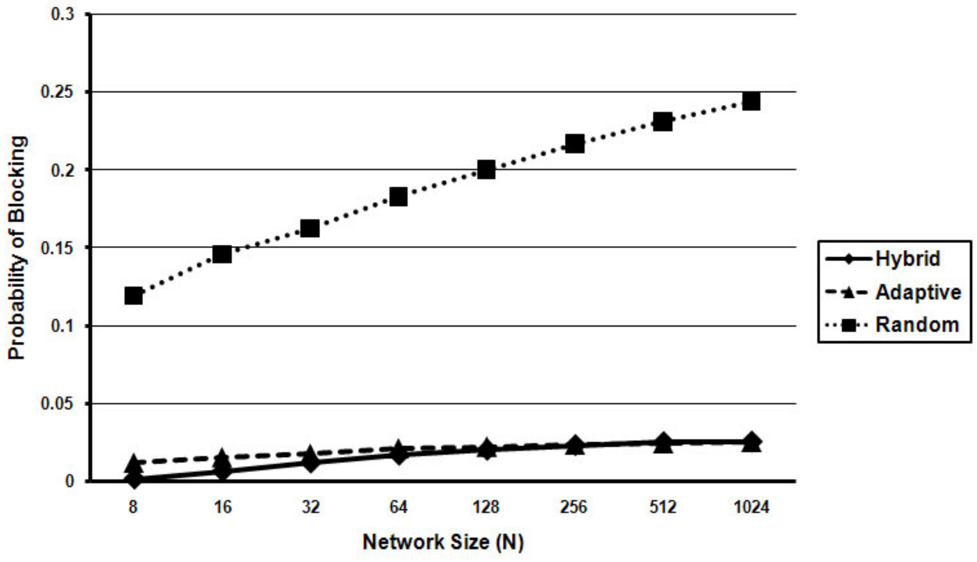


Figure 5.8: 50% *input occupancy graphs for three methods*

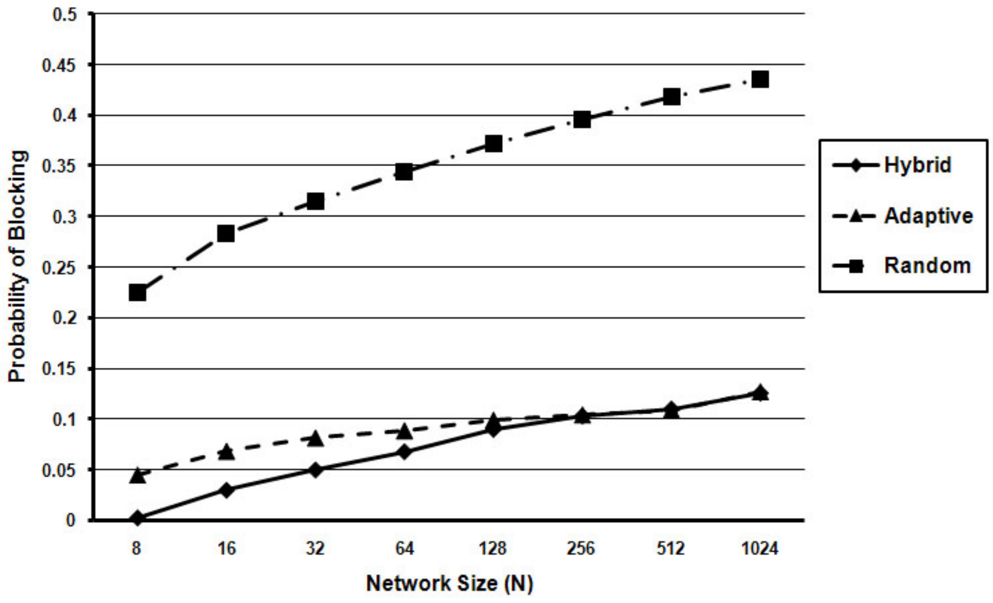


Figure 5.9: 75% *input occupancy graph for three methods.*

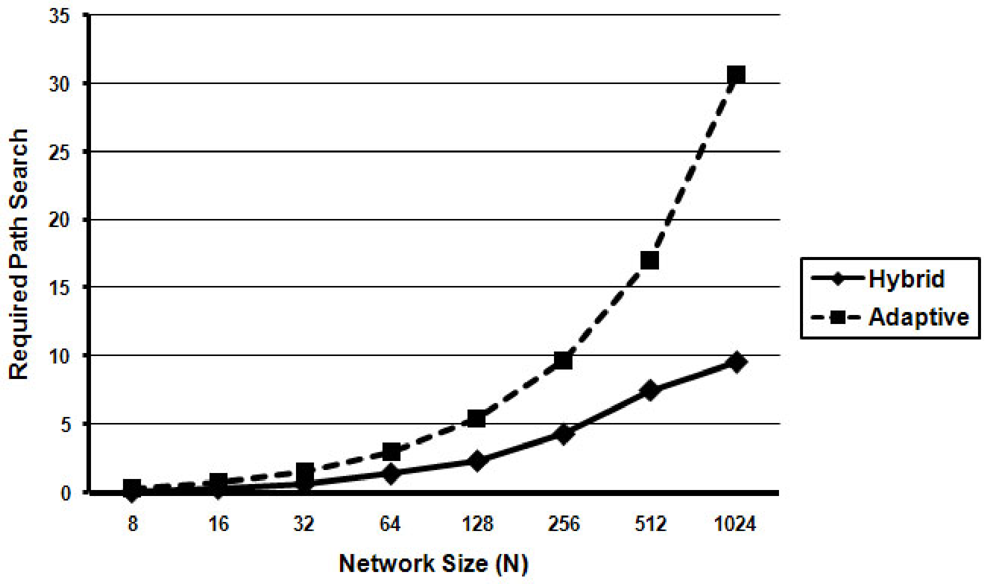


Figure 5.10: *Path search graph for a full occupancy network.*

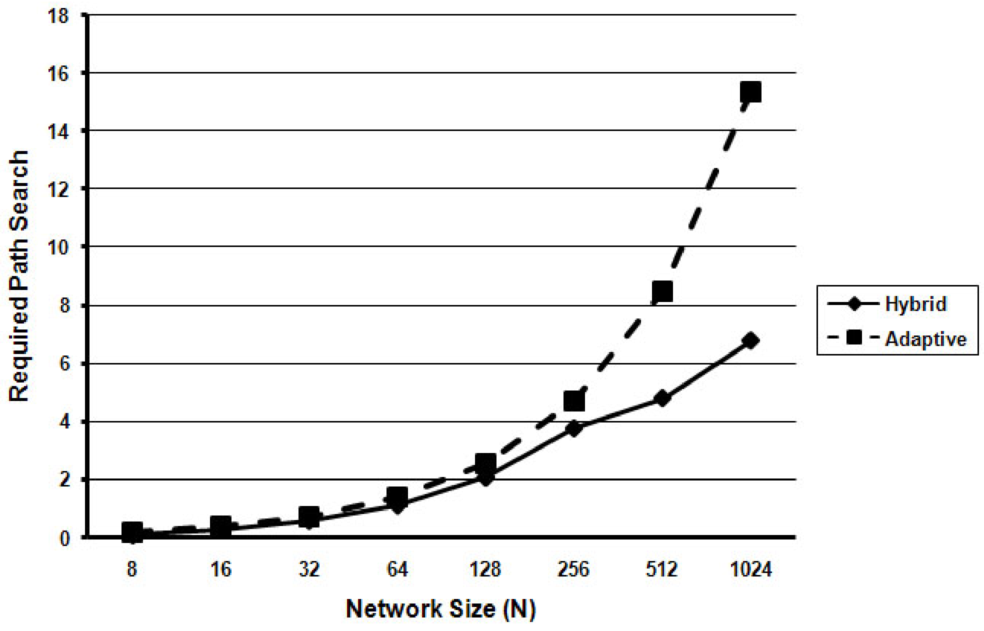


Figure 5.11: *Path search graph for a* 50% *input occupancy network.*

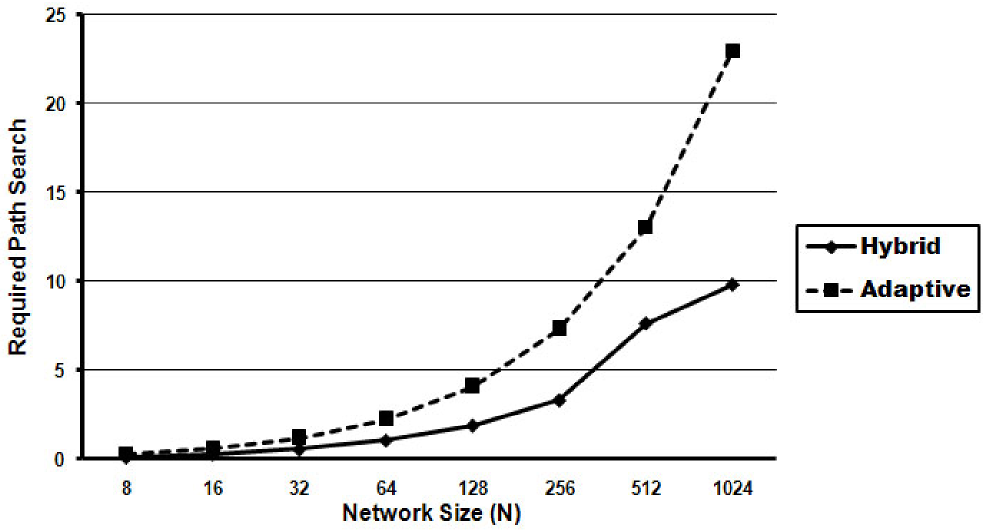


Figure 5.12: *Path search graph for a* 75% *input occupancy network.*

ric. Only hybrid routing and adaptive have been considered in this analysis as Random routing does not use alternative path searching in the case of internal blocking. The path searching count is an important tool to measure algorithm complexity as it is a measure of the time it takes to configure the network. To continue with the results, Fig 5.10 shows the curves indicating the average required number of path searches for both methods for a full occupancy network. Unlike the blocking probability curves, these two curves show huge differences in path search count. For example for $N = 512$, adaptive routing requires almost two times as many path searches as hybrid routing. Fig. 5.11 and 5.12, shows the result for various proportional of active inputs.

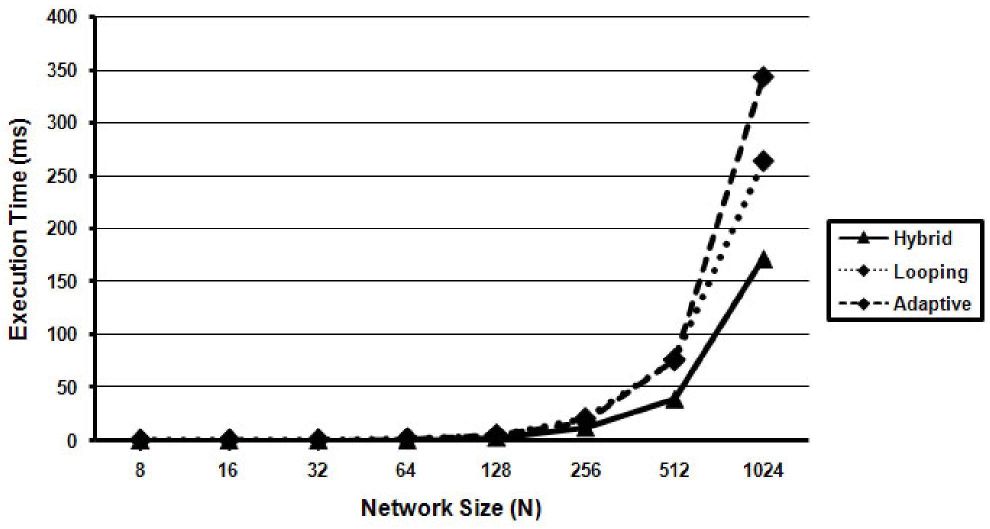


Figure 5.13: *Execution time for full occupancy network.*

The execution time of the two methods is another important measure. Here the execution time of looping algorithm is also taken into consideration. Fig. 5.13 shows the result when the two algorithms applied in a network with full input occupancy. The figure suggests that for smaller networks (for example $N = 64$) there is almost no observable difference in the two routing algorithms. But significant differences can be seen for larger networks. For example, when $N = 1024$ the time difference of more than 150 ms. Also the hybrid algorithm takes less

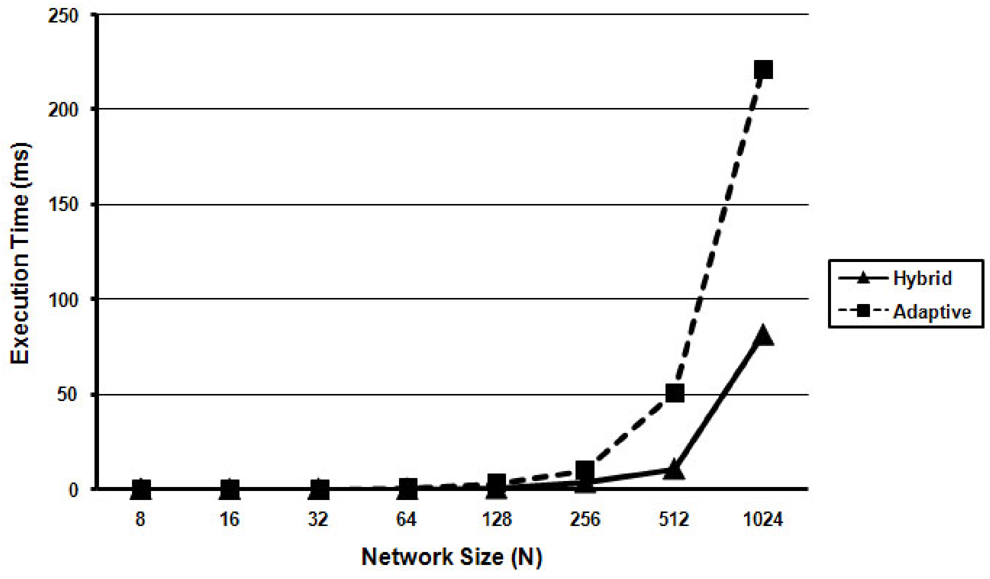


Figure 5.14: *Execution time for a* 50% *input occupancy network.*

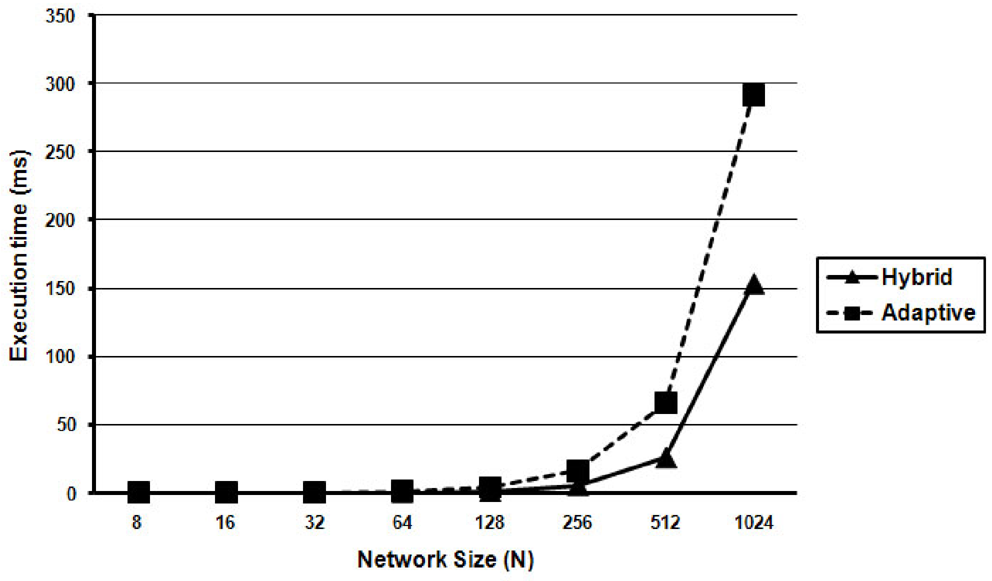


Figure 5.15: *Execution time for a* 75% *input occupancy network.*

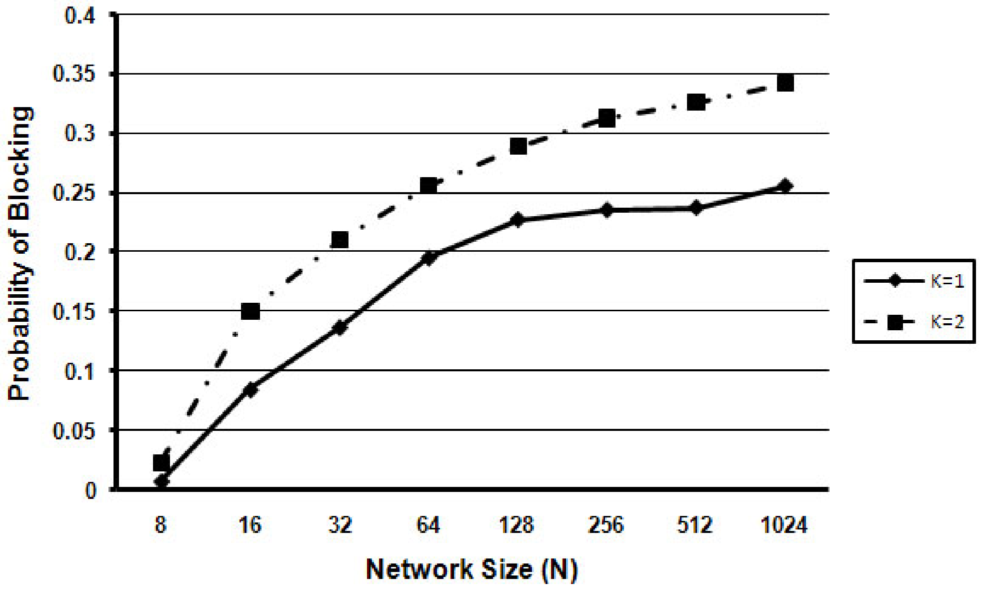


Figure 5.16: *Probability of blocking for hybrid routing with $k = 1$ and $k = 2$.*

time to execute for larger network than the looping algorithm. Fig. 5.14 and 5.15 show the execution time for 50% and 75% input occupied networks respectively. According to these graphs there is also no visible timing difference between these three methods, but for a large input count they show considerable timing differences.

Hybrid routing is modification in functions CONFLICTFRD and CONFLICTREV to reduce the required path search. In function CONFLICTFRD the condition $k < 0$ is changed to $k < 2$, similarly in function CONFLICTREV condition $k > (2\log N - 2)$ is changed to $k > (2\log N - 3)$. These two changes force the algorithm to to check fewer alternative paths, as it has smaller networks to do the checking than the original from where it started routing. Fig 5.16 shows the outcome of these changes to the original one. The result shows increase in the blocking probabilities. Because of the considerable difference between the blocking probabilities, further results (such as execution time, putting partial load to the network) have not been considered for analysis with the proposed modifications.

5.8 Time Complexity Analysis

In this section a detailed complexity analysis of the algorithm is carried out. Hybrid routing algorithm uses the looping algorithm and the adaptive algorithm mentioned in Section 5.4. Main function of this algorithm is FORWARD(). Function FORWARD() controls the routing by making calls to other supporting functions. In this analysis worst case situation for each function will be identified first for both forward and reverse routing. For forward routing this algorithm has three functions that determine the routing paths in forward direction. These functions are:

- FORWARD(): Function FORWARD() in worst case can run $(\log N - 2)$ times to go from stage 1 to stage $(\log N - 2)$.
- BITFRD(): From stage $(\log N - 1)$ to stage $(2\log N - 3)$ routing is carried out by BITFRD() which runs for $(\log N - 1)$ time for each request. In case of a conflict in BITFRD() part of the routing, Function BITFRD() exits by making a call to function CONFLICTFRD().
- CONFLICTFRD(): In the worst case function CONFLICTFRD() runs for $(\log N - 2)$ times, if no available path is found for the request, it is dropped. Function CONFLICTFRD() exits by making a call to function FORWARD(), which starts the routing with another request.

Now time complexity for individual functions will be determined. Conflict free requests are routed using this algorithm as link in self routing method [60]. This is because the decision making is entirely dependent on the status of the switching elements. A request ends up in a switching element at stage k in the distributed part of the network takes decision depending on the status of that switching elements. So no extra computation is required. It is obvious that in the distributed part of the network, there will be no internal conflict, as there are multiple paths available for a request. Conflicts will only occur at the bit control part, as there are multiple paths available to go to the requested output port, but only one of them is conflict free. So when a conflict arises in the bit control part functions CONFLICTFRD() determines the conflict free path. In a situation of unavailability of path, the request is dropped by CONFLICTFRD().

CONFLICTFRD() function loops for $(\log N - 2)$ times in the worst case, hence its time complexity is in $O(\log N)$ for each request, and if there is no alternative path it drops the request and calls function FORWARD() to start routing with a new request. If CONFLICTFRD() is successful in finding a new path, it sends the value of the stage k along with the input number to function FORWARD(). FORWARD() routes the request upto stage $(\log N - 2)$, then its bit controlled routing controlled by BITFRD(). BITFRD() itself runs for $(\log N - 1)$ times so its an $O(\log N)$ function. Function FORWARD() loops for $(\log N - 2)$ time, as a result it is an $O(\log N)$ function for individual request. So for forward routing this algorithm has worst case time complexity of $O(\log N)$.

For reverse routing this algorithm has three functions:

- REVERSE(): Function REVERSE() in worst case can run $(\log N - 2)$ times to go from stage $(2\log N - 3)$ to stage $(\log N - 1)$.
- BITREV(): From stage $(\log N)$ to stage 1 routing is carried out by BITREV() which runs for $(\log N - 1)$ times for each request. In case of a conflict in BITREV() part of the routing, Function BITREV() exits by making a call to function CONFLICTFRD().
- CONFLICTREV(): CONFLICTREV(): In the worst case function CONFLICTREV() runs for $(\log N - 2)$ times, if no available path is found for the request, it is dropped. Function CONFLICTREV() exits by making a call to function FORWARD(), which starts the routing with another request.

The time complexity of reverse routing function can determined from their worst case run time. For all the function in reverse routing time complexity is in

$O(\log N)$. Hence overall time complexity for reverse routing is $O(\log N)$ for each request. So overall time complexity of the algorithm for one request in the forward direction and one in the reverse direction is $O(\log N) + O(\log N) \approx O(\log N)$. So for N active requests overall complexity of the algorithm is $O(N \log N)$.

This matches the best complexities in setting Beneš network using single processor as described in Table 3.2. Reduction on the execution can be achieved by processing independent operations of the algorithm in parallel. The algorithm presented in this chapter rely on dependent decision making, hence making it fully parallel is not possible. But it is possible to assign independent jobs to different processing units.

Once the looping algorithm phase is finished in the hybrid routing algorithm, the algorithm works within two different subnetworks in the network. Routing in these two subnetworks is independent of each other and decision making in one subnetwork dose not affect the decision making in the other subnetwork. In the adaptive part of hybrid routing, the algorithm searches for appropriate middle stage switching element and then from middle stage its bit controlled routing. Once a conflict arises in the subnetwork, the algorithm overcomes the conflict by searching alternative middle stage switching elements. Instead of using only one processor, two processors can be assigned for each sub network. One processor determines the appropriate middle stage switching elements and the other one can create the record for the switching element status in the network. As a result four processors can be used to make the algorithm some what parallel. This will reduce the execution time close to half of a serial routing.

5.9 Summary

This chapter studies the benefits of applying sub-optimal routing algorithms in a symmetric rearrangeable network. The proposed method is unique in the sense that it is a hybrid method for routing. Comparing the method with random routing shows that proposed algorithm gives much better performance for any number of active inputs. Also the results of adaptive routing shows that it has similar blocking probabilities to the hybrid routing, but it requires more path searches and hence more time to setup the network. Results show that hybrid routing also gives faster execution time that the looping algorithm. The proposed hybrid routing has the blocking probabilities associated with it, but this method of routing is faster than other popular method. In the serial processing, proposed method has a time complexity of $O(N\log N)$, which is the minimum for Beneš networks reported in the literature. The hybrid method is thus the best compromise among all mentioned alternatives. The proposed method is not fully blocking free, but it's execution time makes it viable in application where the cost of determining blocking-free paths is excessive. This is particularly import for SOC application where the looping algorithm is still preferred as discussed in Section 3.7.

Major findings of this chapter are listed below:

- Unlike the requirement of full permutation availability of other methods such as in methods [128, 135, 146, 150] this method can realized partial permutations, i.e this method can realize $O(m) \leq O(N)$ requests. This is always beneficial when all the inputs and outputs are not busy.

- When there is not internal conflict in the network, this method works in a way similar to the self routing method.

- Even though to overcome internal conflict this method uses extended path search, still overall time complexity matches best complexity reported in the literature.

- With addition of four processor to route requests in two different sub networks, this can achieve faster performance than with a uni-processor system. This method can give better performance than the parallel routing algorithm proposed in [150]. In [150] a total of p processors are used, where

$p \leq \frac{N}{\log N}$, and achieves a time complexity of $O(N\log N)$ to set up the network. So the method proposed in this chapter is as work efficient as the work efficient algorithm proposed in [150], while require much less hardware to implement.

- This method can be of particular importance in SOC applications. In SOC, the looping algorithm or some form of looping algorithm are still in place. According to the simulation results method proposed in this chapter has much faster execution time, hence will be of particular interest in SOC applications. Also this method can be application in digital subscriber loop applications as mentioned in Section 3.14.

CHAPTER 6

MATRIX BASED ROUTING ALGORITHM

In this chapter, a new routing algorithm is presented for rearrangeably non-blocking networks which can work for partial permutations and can give parallel time complexity that matches best known upper bound reported in the literature.

6.1 Introduction

Non blocking routing algorithms for rearrangeable networks give ideal performance at the cost of computationally intense mathematical models, which take a long time to calculate non blocking paths for each input. Section 5.2 contained discussion of the routing complexities of the looping algorithm. It was shown in that chapter that the looping algorithm suffers from poor scalability, because of the recursive switching elements setup method it applies to setup paths.

Algorithm proposed by Çam and Fortes [148–150], claims to be work efficient when it is implemented in a Parallel Random Access Machine (PRAM) machine. PRAM is built with p (where p is very large) processors with each processor having its own processor ID. All the processors are exposed to a single common shared memory and communication is carried in a single instruction stream, multiple data stream (SIMD) [194, 195] fashion. Each instruction stream takes unit time in the PRAM structure regardless of the processor's number and each processor has a flag that indicates whether the processor is busy or idle. The PRAM model is not a realistic model of available hardware. One of the biggest drawbacks with PRAM is its constant memory access time, as this model suggests p to be a large number, so in the physical implementation these processors will occupy some physical space and the location of the memory access time cannot be the same for all processors. Another issue is with the concurrent reads and writes operation mentioned in PRAM model. With current memory structure, there is a limit on the number of concurrent read and write, which also suggest that it is not possible to perform read and write simultaneously by all the processors. A third issue is that the shared memory has a capacity of $O(N)$, where N is the number of network inputs and outputs. If the memory is on a shared bus, this will limit the speed of execution. If the memory is implemented with an independent path from each processor to each memory element, the complexity of the required interconnect equals that of the network to which the routing algorithm is being applied. Since PRAM model is not practical, so the method of routing in [150] is yet to prove itself to be work efficient.

Another important issue with most of the deterministic routing algorithms is that they require full permutations to make routing decisions. If full permutation is not available, inactive request pairs require dummy request so that the routing algorithm can make decision. Algorithm proposed in [134, 135] introduced the idea of partial permutations for Beneš networks. It has been reported in the literature that minimum parallel time complexity for any Beneš networks routing algorithm is $(\log^2 N)$ as shown in Tabel 3.2. Using algorithms in [134, 135] it is possible to minimize the time complexity to $O(\log^2 \overline{m} + \log N)$ for active requests

$\overline{m} < N$. Issues with this algorithm is that it is required to determine open and close chains before making a blocking free routing decisions. Which increases the calculation complexity of the overall method.

This chapter contains a description of a new and simpler solution for rearrangeable network routing. Simulation results of this method will be used to compare its performance with that of the looping algorithm. This new method will be able to make the routing decision in case of inactive pair of request without the need for dummy request. Also a new proposal for routing in optical domain will also be investigated which was not presented in methods proposed in [134, 135, 148–150].

6.2 Conceptual Basis of the Algorithm

The state of the network will be represented below by set of matrices and a set of sub-matrices. The networks under investigation have a recursive construction [68]. So the target is to formulate a method that can set the switching elements in the outer stages without any blocking, to apply the same method to the next outermost stages with appropriate modifications to the network size and link numbering, etc, since the network can then be regarded as comprising two subnetworks. The network state is recorded using a matrix representation, where rows represent the input switching elements and columns represents output switching elements. Subnetworks are represented by a matrix of lower dimension, applying this representation recursively to reach the inner most stages.

In developing a deterministic routing algorithm, its execution time is always a major concern. The goal is an algorithm that gives better performance than the looping algorithm. Developing the algorithm requires that the network is abstracted as a mathematical model such as a matrix or graphical model. A graphical model could use vertices to represent switching elements, while the edges between the vertices represent the connecting links between switching elements.

Alternatively network stages can be represented as a set of matrices, where each matrix represents each stage and each cell of the matrices determines state of corresponding switching elements.The matrix model is more easily adapted to larger networks, and so was chosen as the abstraction model for this algorithm. This is so because graphical model can represent the connectivity between stages, it can not tell us about the relevant routing decision unless some kind of labelling is used. This model starts by populating the matrix cells corresponding to the input-output requests with binary values (where a 0 represents the straight through position and a 1 is the crossover position in actual implementation), starting by setting the first available switching element to the straight through position. This method gives rise to temporary blocking states in the sub-matrix, because of the constraint that there can be only a single 1 in corresponding row and column, same goes for a value of 0. This is due to the fact that blocking will occur in the underlying network if any row or column of the sub-matrix contains more than a single 0 or 1. This blocking is resolved by doing rearrangements of the cell values. The required reconfiguration cost when a blocking occurs is evaluated in this chapter. Two rearrangement methods have been tried to optimized the reconfiguration cost (as explained in Section 6.3.1). This chapter contains detailed descriptions of the proposed algorithms along with simulation results.

6.2.1 Preliminaries

The notation used in this chapter is defined below:

Definition 6.2.1 (Sub-matrix) *Stages k and $(2\log N-2-k)$ of the switching network is represented by 2^k sub-matrices, where $0 \leq k \leq (\log N-2)$. Configuring the switching element settings at each stage equivalent to putting the values 0 or 1 in the cells of the sub-matrices. The value(s) at cell$[i,j]_k$, where $0 \leq i,j \leq (\frac{N}{2}-1)$, determines the routing tag(s) for input(s) to the switching element,$SE_{k,i}^{\lfloor \frac{i}{2} \rfloor}$, where $0 \leq i \leq (N-1)$, in actual underlaying hardware.*

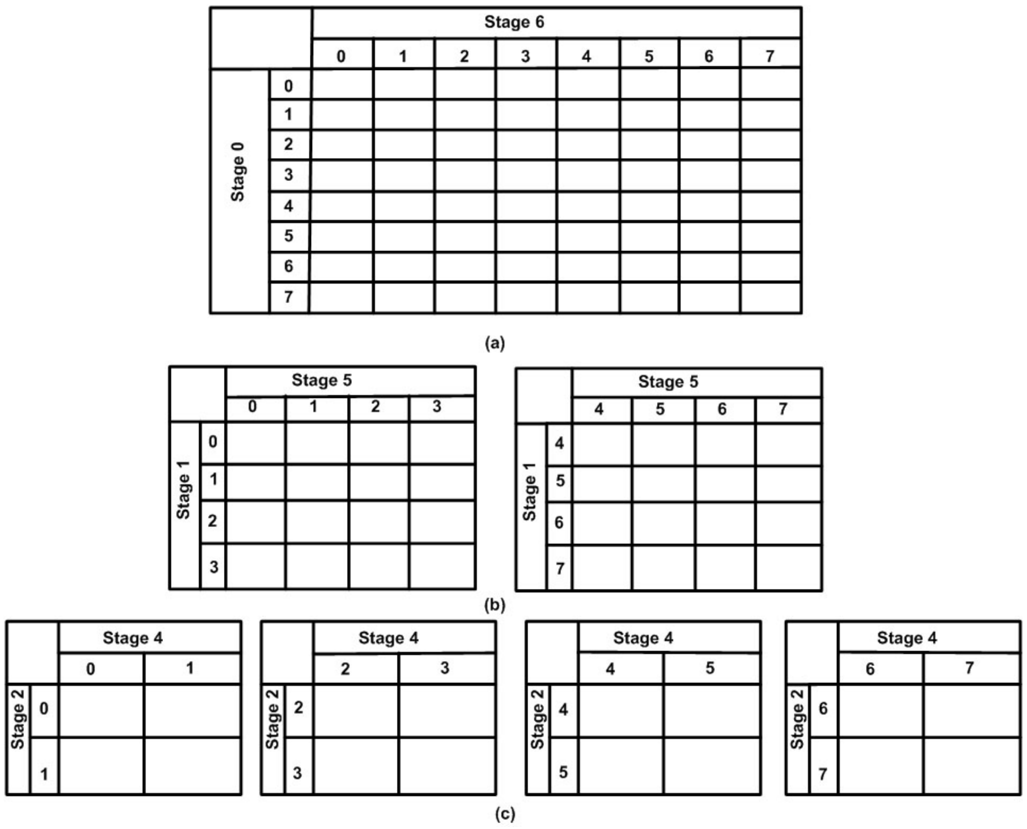


Figure 6.1: *Sub-matrices format for a* 16×16 *Beneš Network.*

Fig 6.1 shows the sub-matrices for a 16×16 network. In the figure each row represents input stage switching elements and each column represents output switching elements. For example, at stage $k = 0$ a set of requests such as $4 \rightarrow 7$ and $5 \rightarrow 10$ would point to CELL$[2, 3]_0$ and CELL$[2, 5]_0$ respectively with values that set the switching element. Values in these cells determines routing tags for input to the switching elements, $SE^2_{0,4}$ and $SE^2_{0,5}$. Such a matrix is called Paull's matrix and details can be found in [100].

Definition 6.2.2 (Blocked Sub-matrix State) *Each row and column in every sub-matrix must contain a single* 0 *and a single* 1*. Otherwise the sub-matrix is in a blocking state. Since routing tags are generated from these sub-matrices, a blocking state inside a sub-matrix will create a situation in the actual hardware where more than one input will compte for a single link, and hence a blocking will occur inside the network.*

In Fig. 6.2, CELL$[6, 2]$ indicates a blocking state in the sub-matrix indicated by "*". This is because CELL$[6, 1]$ has a 1, as a result CELL$[6, 2]$ can only have a

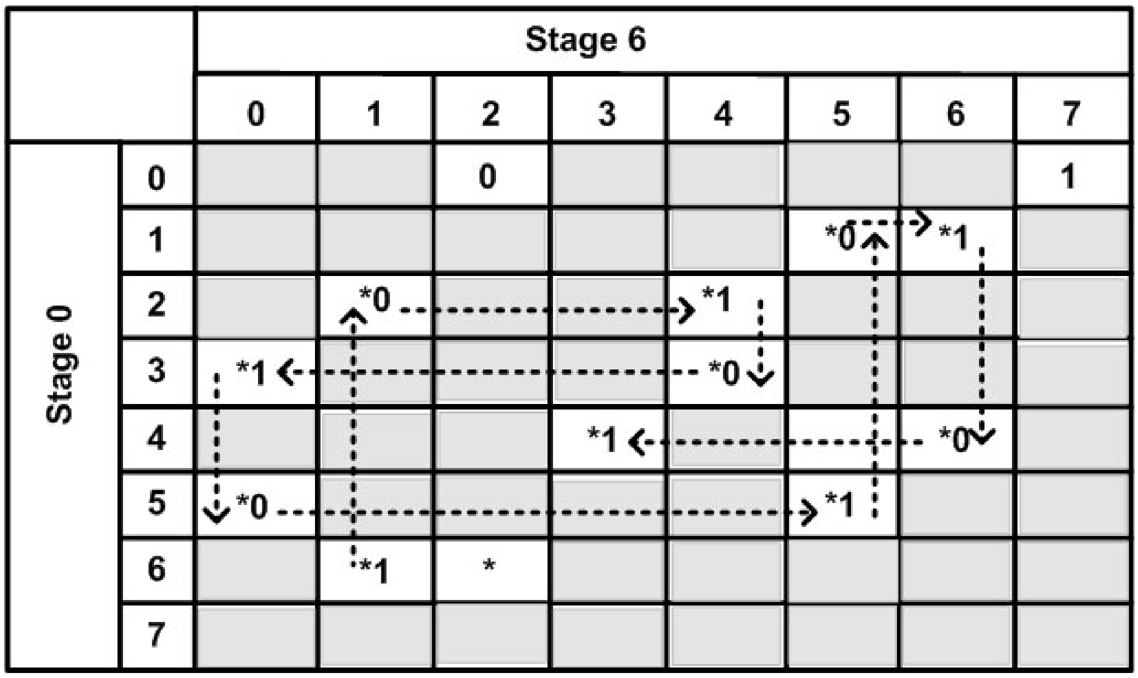


Figure 6.2: *One circuit with length* 11*, where gray boxes represents NULL value.*

0. But in CELL$[0, 2]$ there is a 0, which means in CELL $[6, 2]$ either a 0 or a 1 puts the sub-matrix into a blocking state. In the actual underlaying architecture this demonstrates a situation where two inputs in a switching element have the same routing tags. They are thus trying for the same link to reach to the next stage, and forcing the network into a blocking state.

Definition 6.2.3 (Circuits) *Circuits are used to identify the cells where cell values need to be rearranged, and thus the switching elements settings need to be changed. The length of a circuit is the total number of cells whose contents must be changed to perform the rearrangement. If a candidate circuit has r cells in it then the length of the circuit is r.*

Fig. 6.2 shows a blocking state by indicating the affected cells with the sign "*". The arrows indicate the circuit that needs to be rearranged. The size of this circuit (which in general may not be the smallest) is 11. So as to overcome the blocking in the stage, changes need to made on these 11 cells following the arrows. In other words, the cells indicated by the arrows have to alter their values either from 1 to 0 or from 0 to 1 as indicated by the "*".

Definition 6.2.4 (Balanced State) *Balanced state is the blocking free state of the network. A binary matrix can be called balanced when it has $N = 2^{\log N}$ rows, where each column has an equal number of $1's$ and $0's$ and consecutive $\log N$ columns form binary representation of the set $\{0, 1, \ldots, N-1\}$. These two different balanced matrices create a balanced state in the network, which means no two requests are contending for the same connecting link at the same time in the network, hence no blocking occurs. If C_i, where $0 \leq i \leq (logN - 2)$, are column vectors for the distributed part of the network, then for a conflict free routing each C_i has to be balanced as well as the matrix created from the concatenation of the $C_i's$.*

For example, let C_0 and C_1 be two binary column vectors as shown in fig. 6.3. If C_0 and C_1 are individually balanced then concatenation of C_0 and C_1 must be balance for routing tags to be conflict free [150]. This is because having an unbalanced matrix will create a situation in the actual hardware where two inputs in a switching element have the same routing tags. This means two inputs will try to use the same output link, as a result create a blocking in the network. The concept of neighbour ports as described in 5.1 can be used to explain the routing tags in a balanced column. Two rows corresponding to two inputs in the same switching element will have binary values that are the compliment of each other. So if one input has a 1 in the relevant bit position of its routing tag, its neighbour input port must have to have a value of 0 in the corresponding position to ensure a conflict free routing in the network.

6.3 Symmetric Rearrangeable Networks Routing

This section describes the routing method proposed in this chapter.

6.3.1 Basic Routing Algorithm

Variables used throughout this algorithm as follows:

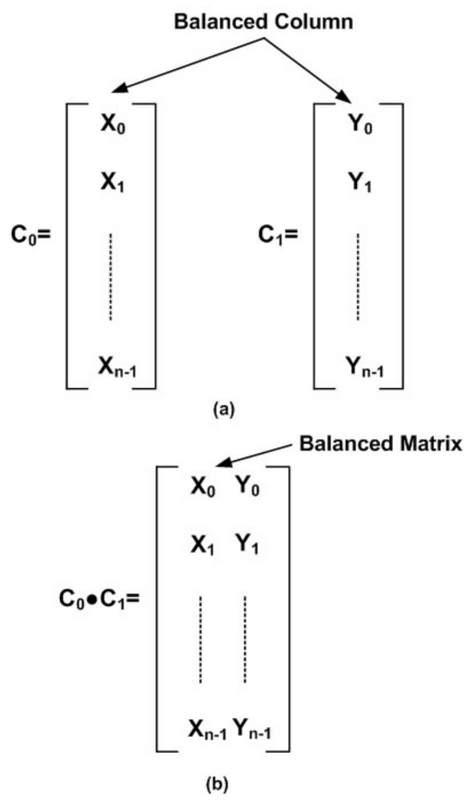


Figure 6.3: *(a) Balanced Columns. (b) Balanced Matrix.*

- $Tag[j, k]$: A target cell at position $[j, k]$ in a matrix that holds the routing tag for input i. Where $j = \lfloor \frac{i}{2} \rfloor$ and $k = \lfloor \frac{P(i)}{2} \rfloor$.
- B_1 : Stores the sum of all the values in the cells of row j. Initially set to NULL.
- B_2 : Stores the sum of all the values in the cells of column k. Initially set to NULL.
- (R', C') : Variables store temporary row and column numbers.

6.3.2 Example

Let us consider $N = 16$ and a random permutation $P_{(0:15)} = ($ 5 14 11 12 9 2 8 1 13 6 0 10 3 4 15 7). Since the switch size is 16×16, there is a single sub-matrix for stage 0 with a size of 8×8. For stage 1 there are two submatrices of size 4×4, stage 2

Algorithm 11 : Algorithm Matrix

```
INPUT: Permutation $P_{0:N-1}$
OUTPUT: Column matrix.
1: for $k = 0$ to $2\log N - 2$ do
2:     for $j = 0$ to $\frac{N}{2} - 1$ do
3:         Tag$[j, k]$ ←NULL
4:     end for
5: end for
6: $i \leftarrow 0$
7: CALL ROUTING(i)
```

Function 12 : ROUTING(i)

```
1: $Flag[j] \leftarrow NULL$
2: $Flag[k] \leftarrow NULL$
3: if $i > (N - 1)$ then
4:     Exit
5: else
6:     repeat
7:         if $Flag[j] = Flag[k] = NULL$ then
8:             $Tag\ [j, k] \leftarrow 0$
9:             $Flag[j] = Flag[k] \leftarrow 0$
10:        end if
11:        if $Flag[j] = Flag[k] = 0$ then
12:            $Tag\ [j, k] \leftarrow 1$
13:            $Flag[j] = Flag[k] \leftarrow 1$
14:        end if
15:        if $Flag[j] = Flag[k] = 1$ then
16:            $Tag\ [j, k] \leftarrow 0$
17:            $Flag[j] = Flag[k] \leftarrow 0$
18:        end if
19:        if $Flag[j] \bigoplus Flag[k] = 0$ then
20:            CONFLICT at $Tag\ [j, k]$
21:            $R' \leftarrow j$
22:            $C' \leftarrow k$
23:            Exit Loop
24:        end if
25:        $i \leftarrow i + 1$
26:        $Flag[j] = Flag[k] \leftarrow NULL$
27:    until $(i > (N - 1))$
28: end if
29: $Flag[j] \leftarrow B_1$
30: $Flag[k] \leftarrow B_2$
31: CALL CONFLICTROW($R'$, $C'$, $B_1$, $i$) & CONFLICTCOLUMN
    ($R'$, $C'$, $B_2$, $i$)
```

has submatrices of size 2×2, Fig 6.4 shows the sub-matrices and the execution of the algorithm. According to the given permutation, input 0, which in underlying hardware is, $SE^0_{0,0}$, requests connecting path to output switching element, $SE^2_{6,5}$ in actual hardware. To generate routing tag for this request, Set $Tag[0, 2] \leftarrow 0$ in sub-matrix for stage 0, which means in actual implementation input 0 will exit from out put 0 of switching element SE^0_0 to go to the next stage, this sets SE^0_0 in straight through state. Similarly for input set $Tag[0, 7] \leftarrow 1$, as 0 or 1 can be used only once in a row or column, and row 0 already has a 0. Continue this process till a conflict aries at row 5 of the stage 0 sub-matrix. Row 5 corresponds to input 10 and 11 in the given permutation requesting output port 0 and 10 respectively. The conflict aries as a result of having a 1 at Tag[5,0], and a 0 at Tag[1,5], which satisfies the condition $B_1 \bigoplus B_2 = 0$ in algorithm Matrix. In this conflicting state of the sub-matrix, the algorithm goes through rearrangements of cell values forming a circuit. Cells involved in the circuit are identified as Tag[5,0], Tag[3,0], Tag[3,4], Tag[2,4] and Tag[2,1]. To resolve the conflict, cell values involved in the circuit have been rearranged. After resolving the conflict, the algorithm starts with input 12, and continue the process to generate a column matrix for stage 0 routing tags for the given permutation.

At stage 1 this algorithm is applied in two sub-matrices that represent two subnetworks and hence the generates the routing tag column matrix for stage 1. Similarly algorithm Matrix is applied to 4 sub-matrices in generate in stage 2 and routing tags column matrix is generated.

6.4 Simulation Results

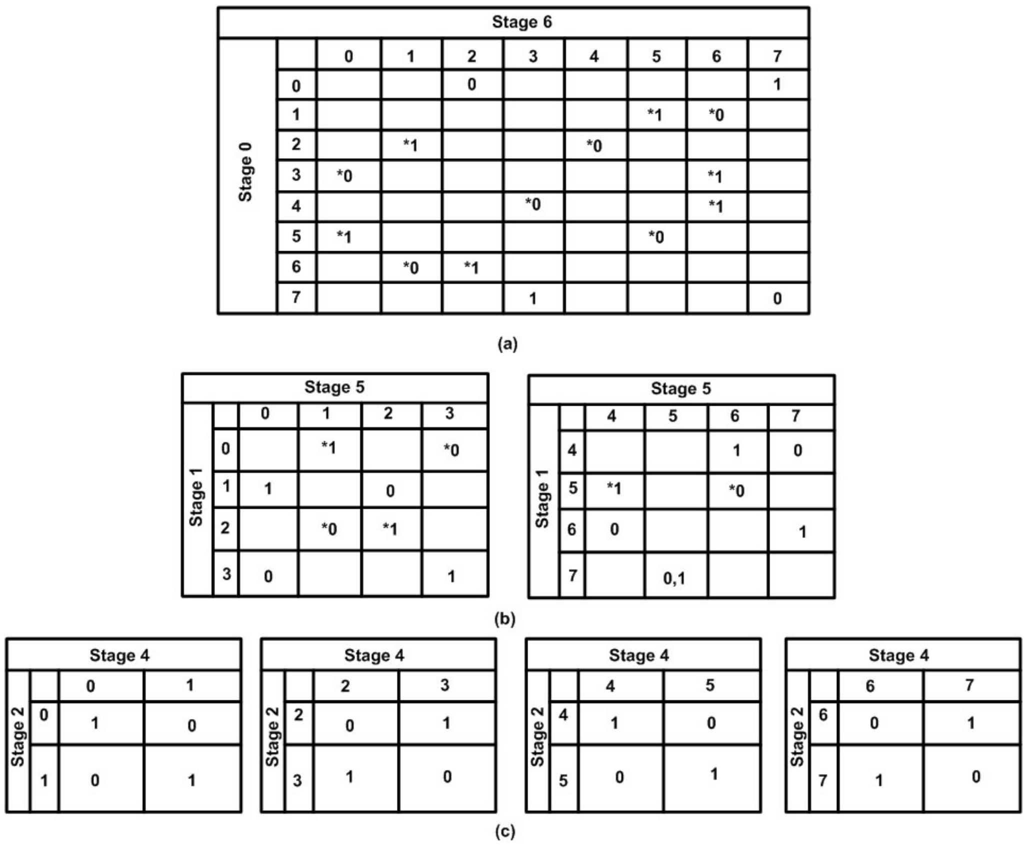
This section gathers simulation results from various size of networks. This section also discuss the reconfiguration cost of generating conflict free routing tags, which is column matrix for each switching stages.

Function 13 : CONFLICTROW($R',\ C',\ B_1,\ i$)

```
1: $Tag[R', C'] \leftarrow \overline{B_1}$ (Complement of $B_1$)
2: $Flag[R'] \leftarrow \overline{B_1}$
3: Find column $k' \neq C'$ at row $R'$ where $Tag[R', k'] = \overline{B_1}$
4: $Tag[R', k'] = B_1$
5: $Flag[C'] \leftarrow B_1$
6: $C' \leftarrow j'$
7: Find row $r' \neq R'$ at column $C'$ where $Tag[r', C'] = B_1$
8: $Tag[r', C'] \leftarrow \overline{B_1}$
9: $Flag[R'] \leftarrow \overline{B_1}$
10: $R' \leftarrow r'$
11: Find row $r' \neq R'$ for any other $\overline{B_1}$
12: if no other $B_1$ then
13:     $i \leftarrow i + 1$
14:     ROUTING(i)
15: else
16:     $R' \leftarrow r'$
17:     Goto 1
18: end if
```

Function 14 : CONFLICTCOLUMN (R', C', B_2, i)

```
1: $Tag[R', C'] \leftarrow \overline{B_2}$ (Complement of $B_2$)
2: $Flag[C'] \leftarrow \overline{B_2}$
3: Find row $r' \neq R'$ at column $C'$ where $CELL[r', C'] = \overline{B_2}$
4: $Tag[r', C'] = B_2$
5: $Flag[R'] \leftarrow B_2$
6: $R' \leftarrow r'$
7: Find column $k' \neq C'$ at row $R'$ where $CELL[R', k'] = B_2$
8: $Tag[R', k'] \leftarrow \overline{B_2}$
9: $Flag[R'] \leftarrow \overline{B_2}$
10: $C' \leftarrow k'$
11: Find column $k' \neq C'$ for any other $\overline{B_2}$
12: if no other $B_2$ then
13:     $i \leftarrow i + 1$
14:     ROUTING(i)
15: else
16:     $C' \leftarrow k'$
17:     Goto 1
18: end if
```



		Stage 6							
		0	1	2	3	4	5	6	7
Stage 0	0			0					1
	1						*1	*0	
	2		*1			*0			
	3	*0						*1	
	4				*0			*1	
	5	*1					*0		
	6		*0	*1					
	7				1				0

(a)

		Stage 5			
		0	1	2	3
Stage 1	0		*1		*0
	1	1		0	
	2		*0	*1	
	3	0			1

		Stage 5			
		4	5	6	7
Stage 1	4			1	0
	5	*1		*0	
	6	0			1
	7		0,1		

(b)

		Stage 4	
		0	1
Stage 2	0	1	0
	1	0	1

		Stage 4	
		2	3
Stage 2	2	0	1
	3	1	0

		Stage 4	
		4	5
Stage 2	4	1	0
	5	0	1

		Stage 4	
		6	7
Stage 2	6	0	1
	7	1	0

(c)

Figure 6.4: *Sub-matrices status after execution of the algorithm.*

6.4.1 The Routing Tag and Its Validity

This method allows a submatrix to have only one 0 and one 1 in each row and each column. The method explained in Section 6.3.1 creates 2^i sub-matrices at each stage where $0 \leq i \leq (\log N - 2)$. The values in the sub-matrices created for stage i, $0 \leq i \leq (\log N - 2)$, represents routing tags for the inputs to the switching elements at that stage. Once a sub-matrix is populated with routing tags for all the input-output requests it represents in a stage, these tags are used to set up the switching elements in the stage of the underlying hardware without any blocking. A sub-matrix is balanced when it has no conflicts, i.e there is no duplication of values in the same row or column. Since the state inside a sub-matrix is balanced, so the concatenated matrix having values taken from all the sub-matrices and having $(\log N - 2)$ columns and $(N - 1)$ rows will always create a balanced matrix. In the concatenated matrix, column 0 corresponds to the values

$$C_0 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \quad C_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad C_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$$C = C_1 \bullet C_2 \bullet C_3 = \begin{bmatrix} 0\ 0\ 0 \\ 1\ 0\ 0 \\ 0\ 1\ 1 \\ 1\ 1\ 1 \\ 0\ 0\ 0 \\ 1\ 1\ 1 \\ 1\ 0\ 0 \\ 0\ 1\ 1 \\ 0\ 1\ 1 \\ 1\ 0\ 0 \\ 0\ 0\ 0 \\ 1\ 1\ 1 \\ 0\ 0\ 0 \\ 1\ 0\ 0 \\ 0\ 1\ 1 \\ 1\ 1\ 1 \end{bmatrix} \quad C_T = \begin{bmatrix} 0000001 \\ 1001001 \\ 0100010 \\ 1100101 \\ 0011111 \\ 1111010 \\ 1010111 \\ 0111000 \\ 0101101 \\ 1000000 \\ 0010110 \\ 1110011 \\ 0001011 \\ 1011100 \\ 0110100 \\ 1101110 \end{bmatrix}$$

Figure 6.5: *Balanced matrices.*

taken from the sub-matrix for stage 0, column 1 from the stage 1 and so on. Each row will have two values for two inputs of switching elements pointing to the row number.The values at position (i, j), where $0 \leq i \leq (N-1)$ and $0 \leq j \leq (\log N - 2)$, correspond to the routing tag for input i at stage j.

6.4.2 Example

For a 16×16 network the proposed method is used to generate the routing tags for randomly generated permutation . The generated permutation is $P_{(0:15)} = (\ 1\ 9\ 2\ 5\ 15\ 10\ 7\ 8\ 13\ 0\ 6\ 3\ 11\ 12\ 4\ 14\ )$. Fig 6.5 shows three column vectors gen-

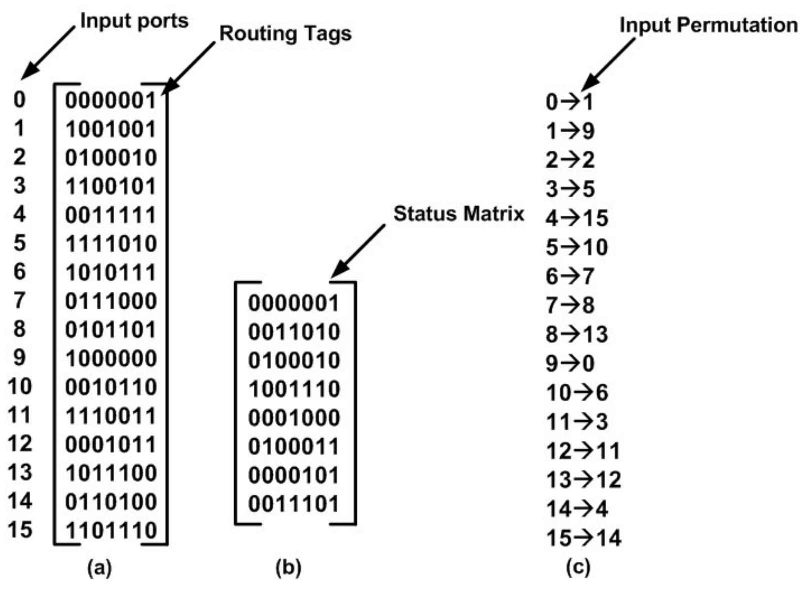


Figure 6.6: *Routing tags and status matrix for a given permutation*

erated from sub-matrices between stage 0 and 2. The rows of vector C_0 contain the values used to set the switching elements at stage 0. Similarly, C_1 and C_2 dictate the settings of switching elements at stages 1 and 2 respectively. All the column vectors are balanced. The matrix C is constructed by concatenating C_0, C_1 and C_2. Matrix C is also a balanced matrix. Each row of matrix C gives the routing tags for the corresponding switching element input. A concatenation of $(\log N - 1)$ balanced vector will give rise to a $N \times (\log N - 1)$ balanced matrix. The binary output matrix C_O itself is a balanced matrix since it is one-to-one request permutation. So $C \cdot C_O = C_T$ will create a balanced state in the network which is the condition for a conflict free routing, even though the combined matrix is not balanced. The sign $\cdot$ indicates a concatenation.

To further verify the correctness of the proposed method, for a random permutation generated routing tags are tested using a tag validator. This validator takes the generated routing tags and then it works in the reverse direction to generate the original permutation. First it generates the status matrix of the switching elements in the switch after a conflict free routing. If the simulator is successful in generating the status matrix, the next step is to generate the original permutation for the given routing tags. The generated status matrix is then used to generate the original permutation. If the permutation matches the original permutation, it proves the correctness of the proposed method. Fig 6.6 shows the routing tags, the status matrix for these tags and the generated permutation which matches the original permutation.

6.4.3 Reconfiguration Cost

The reconfiguration cost inside a sub-matrix to unblock a blocked state can be defined as the required number of cell value rearrangements to make a sub-matrix balanced in the event of a conflict. It is important to know the reconfiguration cost for various sizes of networks. Using only row-first rearrangement process may not always be able to select the smallest length circuit for rearrangement. To investigate the circuit length for a column-first rearrangement also and compare the result with row-first. If these two rearrangement processes provide two different length of the circuit, then the smallest one is selected for rearrangement. A counter stores the required circuit lengths using CONFLICTROW and CONFLICTCOLUMN, and selects the smallest circuit where cell values need to be altered.Simulations for networks of various sizes from $N = 8$ to $N = 1024$ have been executed to determine the reconfiguration cost. Since the reconfiguration cost is directly proportional to the request setup time, it is desirable to keep it to as minimum as possible. Fig 6.7 shows the results of Algorithm 12 using the two rearrangement process Algorithm 13 and Algorithm 14 after selecting the smallest circuit for rearrangement.

The execution time is an important aspect of an algorithm's performance. The execution time of the proposed algorithm compared with that of the looping algorithm in Fig. 6.8. The performance difference between the two methods is in the range of fraction of milliseconds. For larger values of N, such as $N = 1024$, the difference in their execution time is significant. This difference is due to the

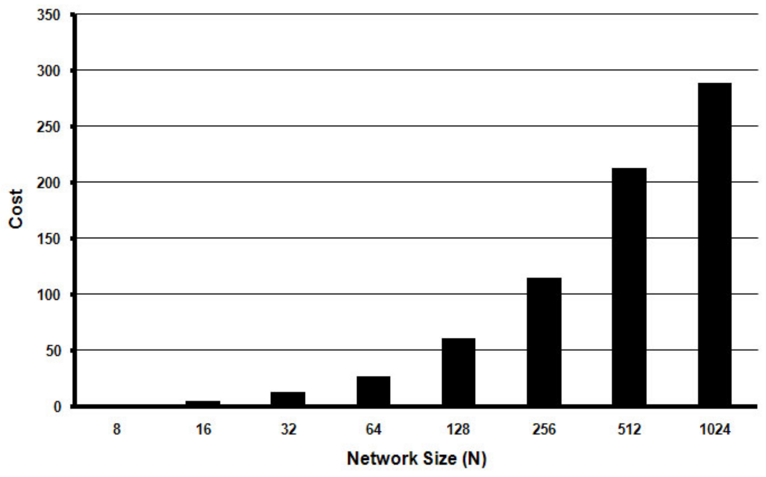


Figure 6.7: *Reconfiguration cost.*

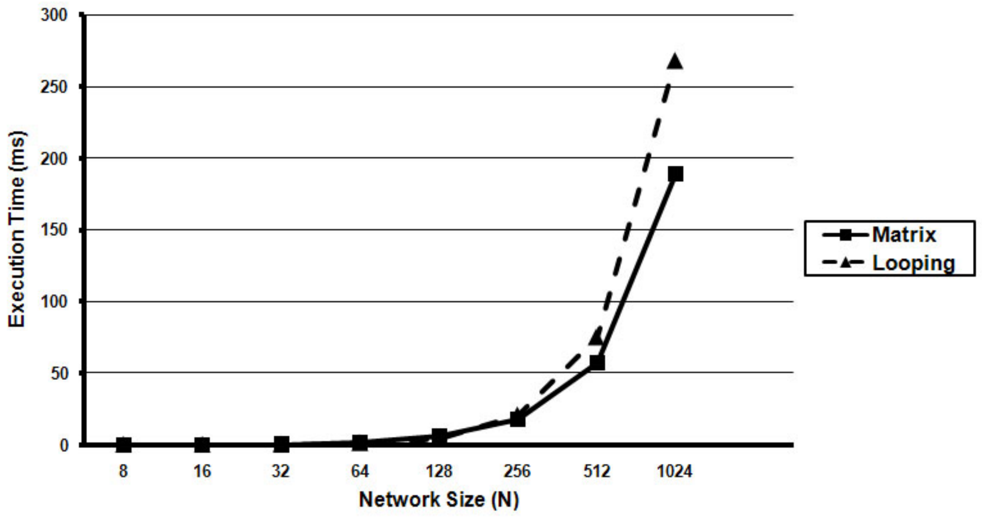


Figure 6.8: *Execution time for Looping and Matrix based algorithm.*

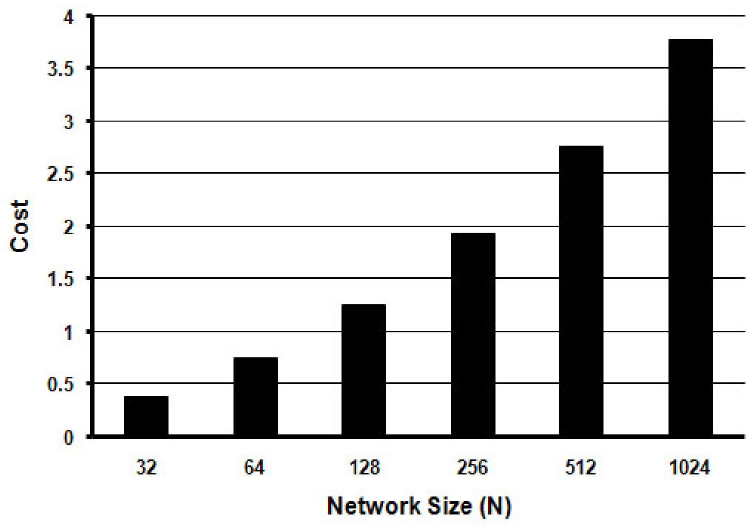


Figure 6.9: *Reconfiguration cost per input for different sizes of network.*

fact that the inner stages of the network are collections of smaller subnetworks, and as a result the proposed method takes less time to generate routing tags for the inner stages. On the other hand, in the looping algorithm decision making is uniform and hence it takes longer to setup. Fig. 6.9 shows reconfiguration cost per input for different sizes of network. It shows reconfiguration cost increases approximately logogrammatically.

6.5 Complexity Analysis

This section studies the time complexities for both serial and parallel implementation of the algorithm.

Before going into the detail of complexity analysis, it is first required to show that this method can work for partial permutations,i.e for request sets where not all the input output pairs have active requests. To do so lets assume a permutation $P_{0:15} = (5\ 7\ 1\ x\ 2\ x\ 13\ 10\ x\ 11\ 9\ 12\ 15\ 0\ 4\ 8)$, where x indicate the inactive requests. Fig 6.5 shows the matrix status for stage 0 for the given partial permutation P. As it can be seen that the only conflict occurs for request $15 \rightarrow 8$, and to resolve the conflict it requires three cells to rearrange their binary values. This shows that the method can determine routing tags for partial permutation. This is an important feature because most of the Beneš networks routing algorithms only work if all inputs have an output request.

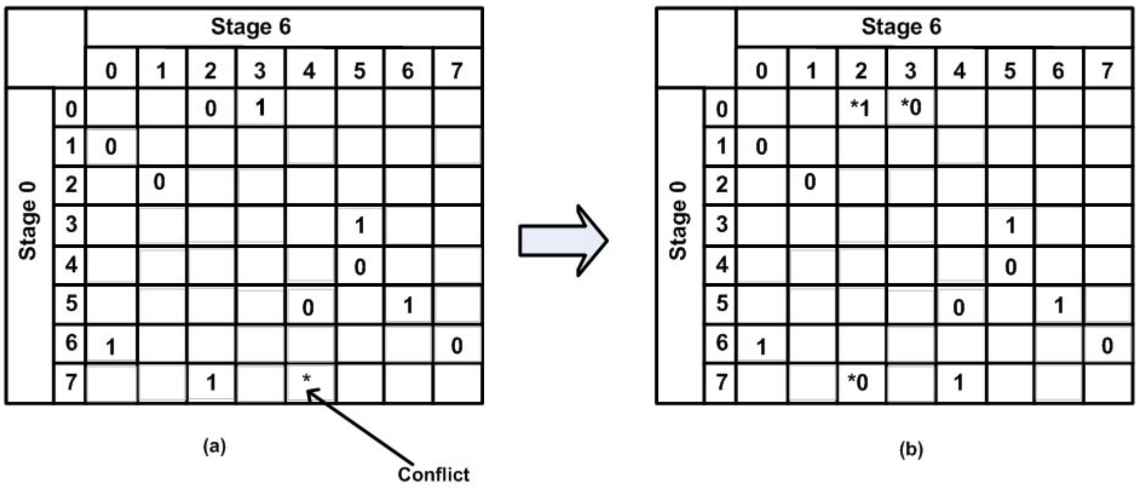


Figure 6.10: *Matrix status for permutation* $P_{0:15} = (5\ 7\ 1\ x\ 2\ x\ 13\ 10\ x\ 11\ 9\ 12\ 15\ 0\ 4\ 8)$

To do the overall complexity analysis of the algorithm it is required to calculate the complexity of each functions that are involved in the algorithm. Functions that are involved in routing are:

- ROUTING(): This is the main function that controls the routing algorithm. Routing runs for N times for each stage in the network for each N inputs.
- CONFLICTROW(): In a situation of conflicting row in a sub-matrix, this function rearranges the value of the cells in a circuit. In worst case this function needs to change $\frac{N}{2}$ cells for a conflict.
- CONFLICTCOLUMN(): In a situation of conflicting column, this function changes $\frac{N}{2}$ cell values in worst case.

Determining the tag for each input in ROUTING() takes $O(1)$ time so for N input it is $O(N)$ at every stage. Once there is any conflict in ROUTING(), execution of the loop ends and it calls two functions CONFLICTROW() and CONFLICT-COLUMN(). These two functions overcomes the conflict by selecting the smallest circuit. In a worse case situation minimum length of the circuit will be $\frac{N}{2}$, hence these functions will have a time complexity of $O(\log N)$. Function ROUTING() runs for $O(\log N)$ stages of the network for each request. This gives an overall time complexity of the algorithm in $O(N\log N)$.

Required execution time can be reduced if parallel implementation is applied. To make the algorithm work in parallel, minor modification is required in the algorithm proposed in Section 6.3.1. In Section 6.3.1 algorithm, each conflict is resolved when they occur in the execution. This makes the algorithm inadequate for full parallel implementation. So the proposed modification is to do the conflict resolution after setting the switching elements in the stage and then rearrange the setting of the conflicting switching elements. This is done by continuing populating the matrix representing each subnetwork with binary values avoiding column conflicts(even if this may cause row conflict). This modification eliminates the need of frequent conflict resolution that may occur. Having N processing elements this algorithm can set up switching elements in each stage with a time complexity of $O(1)$. As the length of the circuit is $\frac{N}{2}$, and length of maximum cell search will be $\frac{N}{2}$. So to search the cells will take $O(\log N)$ time with searching algorithm [196]. Depth of the network is in $O(\log N)$, hence overall complexity of the algorithm will be $O(\log^2 N)$ in a completely connected parallel structure with N processors. This is the minimum complexity for processing Beneš network routing in parallel [133, 146, 197] using $O(N)$ processing elements.

The proposed method can also work for $m \leq N$ requests. Hence for uniprocessor system the overall complexity will be $O(\overline{m}\log N)$ where $\overline{m} = \rho N$. I will consider the case where $\overline{m} < N$. This corresponds to a network with a large number of idle inputs. The complexities of an efficient algorithm will scale in proportional to $\overline{m}$. Also for completely connected parallel machine, for $\overline{m}$ active requests only $\overline{m}$ processors are needed rather than N. With only $\overline{m}$ active requests in the network, maximum length of a circuit will be $\frac{\overline{m}}{2}$, hence $\overline{m}$ processors are enough to break a circuit and rearrange the cell values in each stage.

With $\overline{m}$ active requests in the network, maximum length of a circuit will be $\lfloor \frac{\overline{m}}{2} \rfloor$, which gives a search time of $O(\log\overline{m})$. So with $\overline{m}$ active requests and completely connected network with $\overline{m}$ processors parallel time complexity of the algorithm is $O(\log N.\log\overline{m})$, which is the minimum upper bound than any other algorithm reported in the literature for unicast $\overline{m}$ active requests.

The method of rearranging cell values is similar to that of graph 2 coloring. Each cell can be considered as a vertex and two neighbour vertices are one cell in the same row and one cell in the same column. Edges connecting the vertices to that vertex will have two different colore. This is similar to the routing algorithm condition of having no duplicate binary values in the same row or column. As the modified algorithm continues to populate matrix cell with binary(even with possibilities of row conflict), at the end of the populating process, circuits are identified. This process will take $O(\log N)$ time with a searching algorithm. Then similar to the graph 2 coloring cells in the circuits are assigned binary values such that it overcomes the conflict. Using a PRAM computer with N processors and N nodes graph 2 coloring can be done in $\log N$ time [150, 198]. At each stage the routing algorithm spend $O(\log N)$ time to find conflict free routing tags. So with PRAM machine with N processors and $O(\log N)$ stages the proposed algorithm can be executed in $O(\log^2 N)$ time to determine conflict free routing tags. Similar arguments can be made for $\overline{m} < N$ active request pairs, and show that the time complexity of the algorithm will be $O(\log\overline{m}.\log N)$.

$O(\log^2\overline{m} + \log N)$ in [134] is the best reported complexity in the literature for Beneš networks using partial permutations in parallel domain. The complexity of the algorithm proposed in this chapter is less than that of the best case complexity reported in [134] for a range of values of ρ. Let $f(N) = O(\log\overline{m}.\log N)$ and $g(N) = O(\log^2\overline{m} + \log N)$. Let's find the value of ρ for which:

$$\begin{aligned} f(N) &= g(N) \\ &\Leftrightarrow \log\overline{m}.\log N = (\log^2\overline{m} + \log N) \\ &\Leftrightarrow \log(\rho N).\log N = (\log^2(\rho N) + \log N \\ &\Leftrightarrow b(a+b) = (a+b)^2 + b \quad [where\ a = \log\rho\ , b = \log N] \\ &\Leftrightarrow a^2 + ab + b = 0 \qquad (6.1) \end{aligned}$$

Eqn. 6.1 has roots:

$$a = \frac{-b \pm \sqrt{b^2 - 4b}}{2} \qquad (6.2)$$

Substituting for a and b in Eqn. 6.2, the crossover point for ρ:

$$\rho = 2^{\frac{1}{2}[(\log^2 N - 4\log N)^{\frac{1}{2}} - \log N]} \qquad (6.3)$$

Determining $\lceil\overline{m}\rceil$ and $\lceil\log\overline{m}\rceil$ using Eqn. 6.3 gives the minimum value for which the proposed algorithm in this chapter outperforms the algorithm in [134]. For large N, ρ approaches to 0.5.

6.6 Routing in Optical Domain

This section will overview the application of matrix based routing algorithm for the switching in optical domain along with the time complexities of the algorithm after the necessary modifications.

Routing in optical domain is much more critical than in electrical domain because of the noticeable crosstalk effect. Crosstalk occurs when two waveguides transmits part of their signal power to one another. For example, lets assume that one switching element is set to a cross state and two active inputs carnying signals A and B are in its upper and lower inputs respectively. In a crosstalk situation certain amount of signal power from A will be coupled to output of B and output of A will have some signal power of B. When output signal is affected by the crosstalks taken place in the switching element, it is termed as first order crosstalk [199, 200], in this thesis the keyword crosstalk will indicate first order crosstalk.

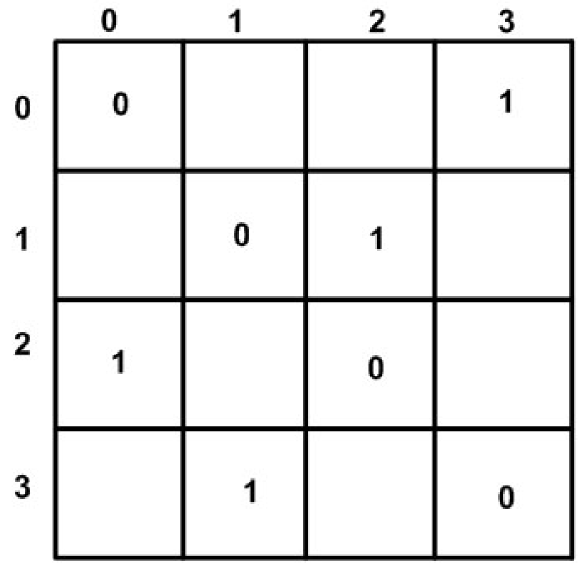
In order to overcome the crosstalk effect three approaches are in place, *space dilation*, *time dilation* and *wavelength dilation*. In space dilation approach, concurrent switching of two different signals are avoided through a single switching element [201–203]. In other words only one input is active in each switching elements in the network at a time. Which requires increase in the number of switching elements, in general this is done by adding multiple plane of the network. In time dilation approach input permutation is divided into two half and one half is routed at a time to avoid crosstalk [181, 204]. Clearly hardware complexities grows for space dilation and time complexity for time dilation approach. In wavelength dilation, different wavelengths are used for active switching inputs are wavelength converters are used in the switching elements [205]. In this chapter, focus will be on the time dilation and space dilation approaches of routing.

Any permutation can be break into two separate permutation by using the concept of semi permutation described in [121]. It has been shown in [123, 206] that for an optical Beneš networks, each semi permutation can be realized in one pass. Graph 2 coloring based approach for generating semi permutation has been presented in [198]. In that approach it has been ensured that two signals entering an output or exiting from an input switching element have different colors. Hence signals with same colors forms a semi permutation. And using time dilation approach each permutation is routed through the network, hence it takes two pass to route a complete permutation using time dilation approach.

Using matrix based routing algorithm proposed in this chapter, semi permutation can be generated for an $N \times N$ optical Beneš routing in time dilation approach. It has been mentioned in Section 6.3.1, that there can not be any repetition of binary values in the same row or column when generation routing tags for input output requests. This method works in stage by stage fashion, i.e after generating routing tags for stage k, the algorithm goes on to generate tags for stage $k + 1$ and so on. Also any conflict in the cell value is resolved using conflict resolution method once initial value of the routing tags are put in to the matrix cells. After the conflict resolution, each value in a matrix cell indicates routing tags for input and output stage for an input output request. It has been shown in Section 6.5 that with $O(N)$ processors in completely connected parallel computer it takes $O(\log N)$ time to generate routing tags for each stage. Two routing tags at input or output stage switching element will be complement of each other when there is no conflict in the stage. These two values can be consider as two colors with the two requests entering or exiting to/from any switching element. Once the matrix for each stage are configured, two binary values will be used to generate the semi permutation. Requests having same binary value as routing tag forms a semi permutation. So request with routing tag 0 will be one set of semi permutation and another set comprises the requests having 1 as routing tag. Fig 6.6 shows a matrix for stage 0 for permutation $P_{0:7} = (0\ 7\ 3\ 5\ 1\ 4\ 2\ 6)$. From the matrix two semi permutation will be $P_1 = \left(\begin{smallmatrix} 0 & 2 & 5 & 7 \\ 0 & 3 & 4 & 6 \end{smallmatrix}\right)$ and $P_2 = \left(\begin{smallmatrix} 1 & 3 & 4 & 6 \\ 7 & 5 & 1 & 2 \end{smallmatrix}\right)$.

As it is shown in previous section that generating routing tags using matrix based routing method takes $(N\log N)$ serial time and $O(\log^2 N)$ parallel time for $O(N)$ active inputs having $O(\log N)$ depth network with $O(N)$ completely connected processors, hence generating semi permutation for all the stages in the network will have similar time complexities in both parallel and serial execution. The matrix decomposition serves two purpose in routing, one it helps generating the semi permutations and second it also provides the routing tags for each input at each stage. Once the semi permutations are in place, using time dilation approach, in two passes any $N!$ permutations can be routed in the network.

Using space dilation in Beneš networks any permutation can be realized in a



	0	1	2	3
0	0			1
1		0	1	
2	1		0	
3		1		0

Figure 6.11: *Routing tag matrix for stage* 0.

single pass by using multiple plans of the network. Lets assume two semi permutations are P_1 and P_2, each of these two permutations are feed into two plans of the space dilated Beneš network where crosstalk free routing is carried out. These networks contain two copies of the same network, hence compromising hardware complexities for saving execution time. The structure of the network requires little modifications than regular Beneš networks. Outermost stages of these networks have spelters and combiners. In the input stage a total of N 1×2 spelters are there to feed the input request at the appropriate plan of the network. Similarly at output stage the network has a total of N 2×1 combiners that combines the output coming from two plans. Having two separate plans for routing allows crosstalk free routing in optical domain using semi permutation from the partial permutation at each stage. Fig 6.6 shows a space dilated optical Beneš network. The time complexity of the routing algorithm in optical domain remains the same as in the electrical domain.

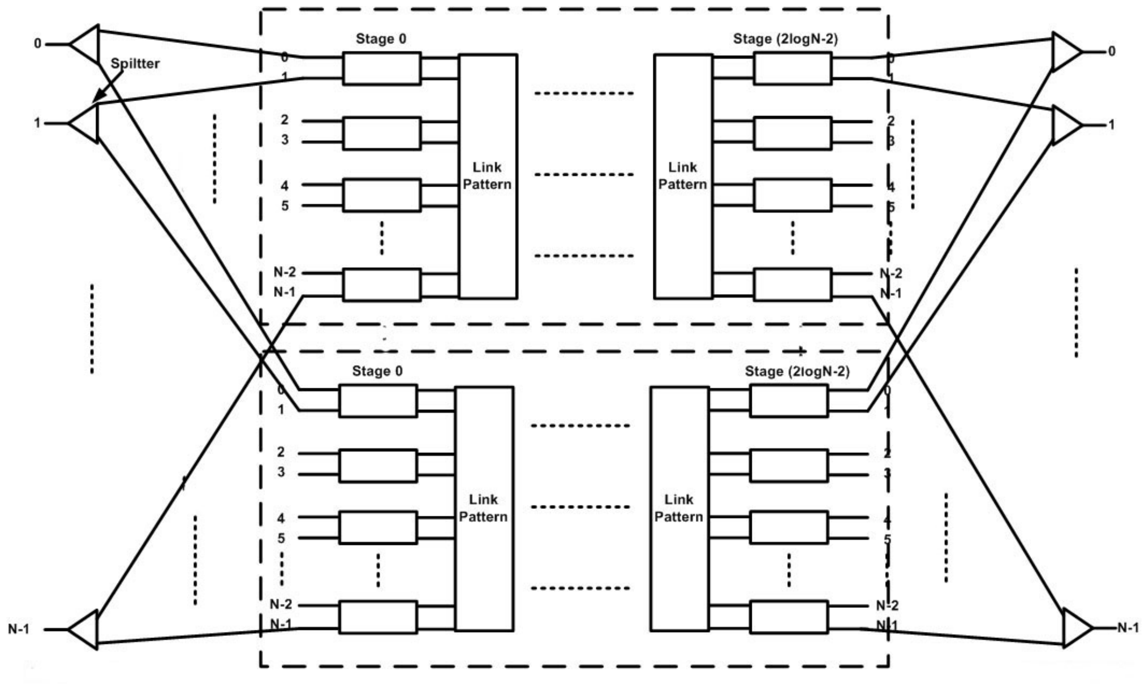


Figure 6.12: *Space dilated Optical Beneš network.*

6.7 Networks with Large Switching Elements

This section investigates possible modifications to classic topology of symmetric rearrangeable networks built with 2×2 switching elements, also such as suing larger switching elements. The classic topology will be called as Topology A in rest of this chapter.

6.7.1 Reducing Network Depth

Fig 3.3 shows a Topology A network of size $N = 16$. This network can be reduced in depth by using 4×4 switching elements in each stage, to obtain a topology B network. This reduces the total number of stages in the network without increasing crosspoints count.

The depth of the networks for where $\log N$ is even can be reduced more than others. Networks for which $\log N$ is odd, will be called Topology C networks. Table 6.1 shows the network depth using Topologies A, B and C. In the table, Topology B shows a considerable decrease in the network depth compared to

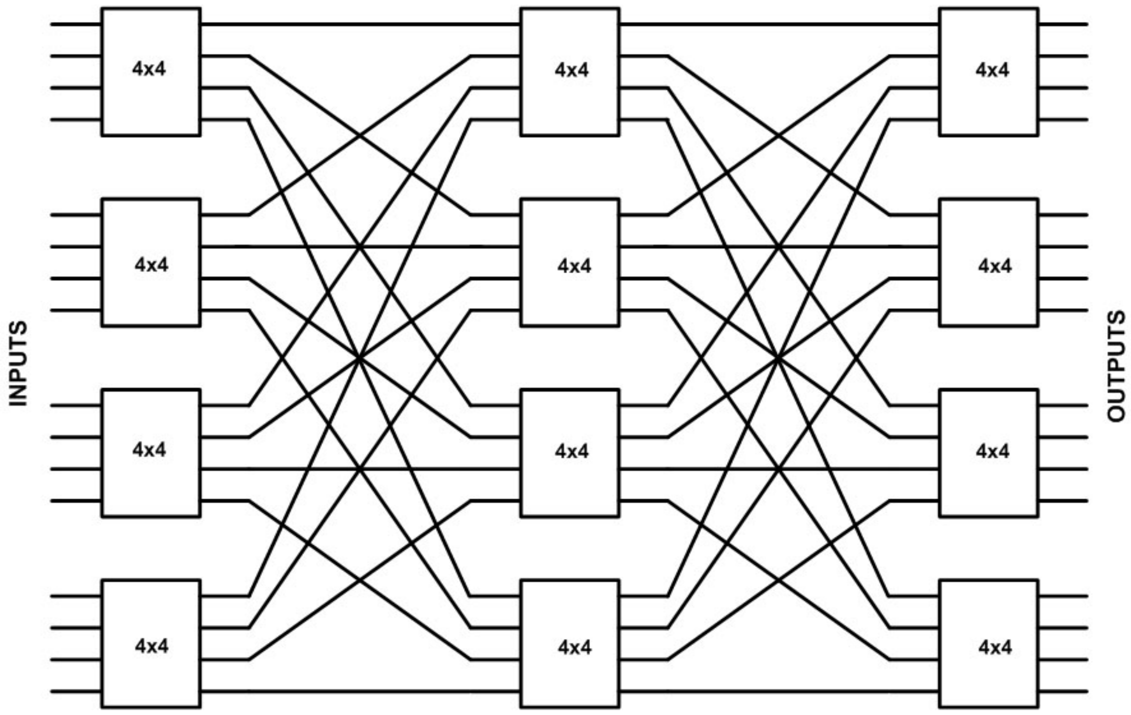


Figure 6.13: *A* 16×16 *symmetric network built with* 4×4 *switching elements*

the other two topologies. In Topology B, where $\log N$ is even (such as $N = 16,\ 64,\ 256\ and\ 1024$) network depth is reduced to less than half that of Topology A. Fig. 6.13shows structure of a 16×16 symmetric rearrangeable network using Topology B. The link patterns at stages 0 and 1 of Fig. 6.13 can be denoted as $i \rightarrow \lfloor \frac{i}{4} \rfloor + 4i \bmod 4$, where i is the input number. Networks belonging to Topology C, use two different link patterns. In the outermost stages, they have the same link patterns as Topology A, and in other stages they use the link pattern of Topology B.

The standard routing algorithms used for Topology A networks will not work for Topologies B and C. This is because, switching elements larger than 2×2 can no longer be controlled by binary bits and require more complex decision making. For Topology C networks, where the outer stages are built with 2×2 switching elements and the inner stages are made of 4×4 elements, routing through these network requires two different algorithms. One routes in 2×2 stages of switching and the other applies for stages built with 4×4 switching elements. An alternative would be to use the looping algorithm, as the operating

Table 6.1: *Comparison of network depths.*

N	Topology A	Topology B	Topology C
8	5	x	3
16	7	3	x
32	9	x	5
64	11	5	x
128	13	x	7
256	15	7	x
512	17	x	9
1024	19	9	x

principle of the algorithm allows it to work for networks with multiple switch sizes. An other alternative would be to use the algorithm proposed in Section 6.3, which gives better performance for large Topology A networks than does to the looping algorithm.

In the case of a heterogenous network, switching stages are built with different sized switching elements and generally the middle stage should be built with switching elements larger than in other stages. If the total number of inputs of the network is N, then N can be given as $N = \prod_{i=1}^{n} P_i$, where $\{P_i\}$ is the set of prime factors of N and n is the total factors of N and $P_i \geq P_{i-1}$. This network can be built with a total of $(2n-1)$ stages, where stages $i = 1$ to n use $P_i \times P_i$ switching elements, and where the switching elements sizes in stages $(n-i)$ and $(n+i)$ are the same for $i = 1$ to $n-1$. The number of switching elements per stage is $\frac{N}{P_i}$, in stages i and $(2n-i)$, and each switching element has a crosspoint count of P_i^2, $1 \leq i \leq n$. Hence the overall crosspoint count per input for the network can be given as $2\sum_{i=1}^{n-1} P_i + P_n$.

Table 6.2 show an example of a network of size $N = 120$, built with heterogenous switching elements. In the table it can be seen that smaller middle stage switching elements(≤ 10) give the minimum crosspoint count. Longer switching element sizes than 2×2 are considered below:

Table 6.2: *Crosspoint count for networks with heterogenous switching elements*

factors, $N = 120$	**crosspoints per input**
2,2,2,3,5	$2(3 \times 2 + 3) + 5 = 23$
2,3,4,5	$2(2 + 3 + 4) + 5 = 23$
4,5,6	$2(4 + 5) + 6 = 24$
8,15	$2 \times 8 + 15 = 31$
2,5,12	$2(2 + 5) + 12 = 26$
10,12	$2 \times 10 + 12 = 32$
2,2,5,6	$2(2 + 2 + 5) + 6 = 24$
2,2,3,10	$2(2 + 2 + 3) + 10 = 24$
2,2,2,15	$2(2 + 2 + 5) + 15 = 27$
3,5,8	$2(3 + 5) + 8 = 24$
2,5,12	$2(2 + 5) + 12 = 26$
2,3,20	$2(2 + 3) + 20 = 30$
2,2,30	$2(2 + 2) + 30 = 38$
2,60	$2 \times 2 + 60 = 64$
3,40	$2 \times 3 + 40 = 46$
5,24	$2 \times 5 + 24 = 34$
4,30	$2 \times 4 + 30 = 38$
6,20	$2 \times 6 + 20 = 32$

6.8 Networks with 3×3 Switching Elements

This section considers networks that can be built using 3×3 switching elements. Obviously binary control does not apply in these networks, nor can widely popular the looping algorithm. One reason for the looping algorithm not being applicable in these networks is that it works for only networks that have an even number of middle switches. Networks built with 3×3 switching elements have an odd number of switches in the middle stage and hence the looping algorithm is not applicable. The routing algorithm proposed in Section 6.3.1 has been extended below so that it can work for such networks. Fig. 6.14 shows a 27×27 symmetric rearrangeable network built with five stages of 3×3 switching elements. Networks build with 3×3 switching elements have a total of $(2M - 1)$ stages where M is the number of prime factors of N [73]. The overall crosspoint count is then $3N(2M - 1)$.

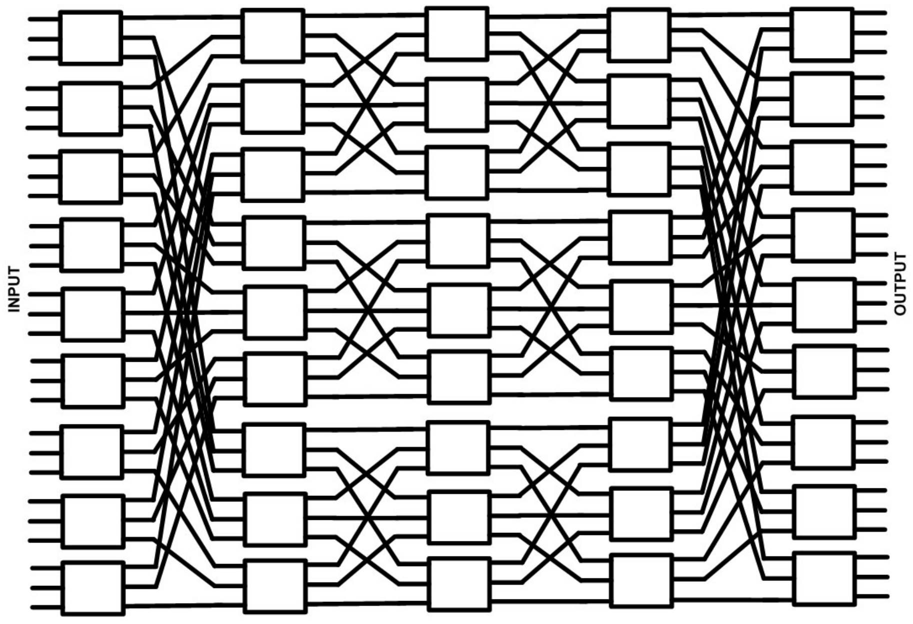


Figure 6.14: *A 27×27 symmetric network build with 3×3 switching elements*

These networks have two major advantages over networks built with 2×2 switching elements, one being that they have fewer stages and another that they reduce the overall crosspoint count. Tables 6.3 and 6.4 show the number of crosspoints for networks built with 2×2 and 3×3 switching elements respectively. In general the number of crosspoints required is reduced by a factor of approximately $\frac{\log 2}{\log k}$ when a switch is built with $k \times k$ switching elements rather than 2×2. This assumes that all switch inputs are used, i.e $N = k^{\log_k N}$.

Table 6.3: *Crosspoint for networks using 2×2 switching elements*

N	total stages	overall crosspoints	crosspoints per input
8	5	80	10
16	7	224	14
32	9	576	18
64	11	1408	22
128	13	3328	26
256	15	7680	30
512	17	17408	34
1024	19	38912	38

A 3×3 switching element has more possible states than a 2×2 element. It has 6 different active states where all the inputs are active, and one idle state which

Table 6.4: *Crosspoint count for networks for* 3×3 *switching elements*

N	total stages	overall crosspoints	crosspoints per input
9	3	81	9
27	5	405	15
81	7	1701	21
243	9	6561	27
729	11	24057	33
2187	13	85293	30
6561	15	295245	45

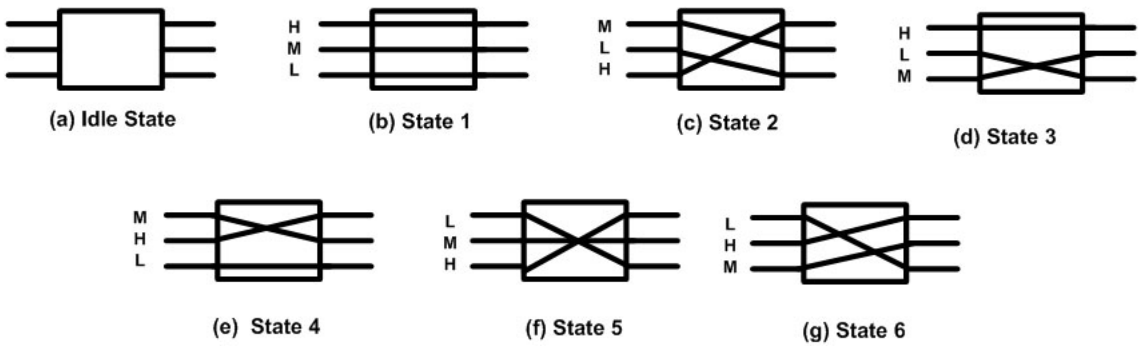


Figure 6.15: *Different states of a* 3×3 *switching elements*

means all the inputs are idle. There can be other active states in a switching element when a fraction of its inputs are active and rest are idle. In this discussion only switching element states with all inputs active and one idle state will be taken into consideration. The control inputs can be denoted as $(H,\ M,\ L)$ where the letters indicate the upper,middle and lower output ports respectively.This control inputs decides the connecting output port for the input port. Control inputs connect input to an output port following the rule in Eqn 6.4, where $I_0\ I_1,\ I_2$ are the input ports to a switching element and $O_0\ O_1,\ O_2$ are its output ports. Fig. 6.15 shows various active states of a 3×3 switching element.

$$I_0, I_1, I_2 = \begin{cases} O_0, & \text{if control input H} \\ O_1, & \text{if control input M} \\ O_2, & \text{if control input L} \end{cases} \tag{6.4}$$

6.9 Routing Algorithm

The routing method of section 6.3 requires major modifications so as to adapt it to networks using 3×3 switching elements. Network stages are represented by a set of sub-matrices. Each row and column representing 3 inputs and outputs respectively of the underlaying hardware. Each stage is represented by 3^k matrices, where k is the stage number and $0 \leq k \leq (M-2)$. Each matrix is populated using the control input values $(H,\ M,\ L)$. These values must be chosen so that there is no duplication of values in any row or column. This ensures nonblocking routing in the actual hardware.

Since there are three choices rather than two in these networks, an order must be specified to apply in populating the cells. Populating each cells of sub-matrix is carried out below following the rule in Eqn 6.5

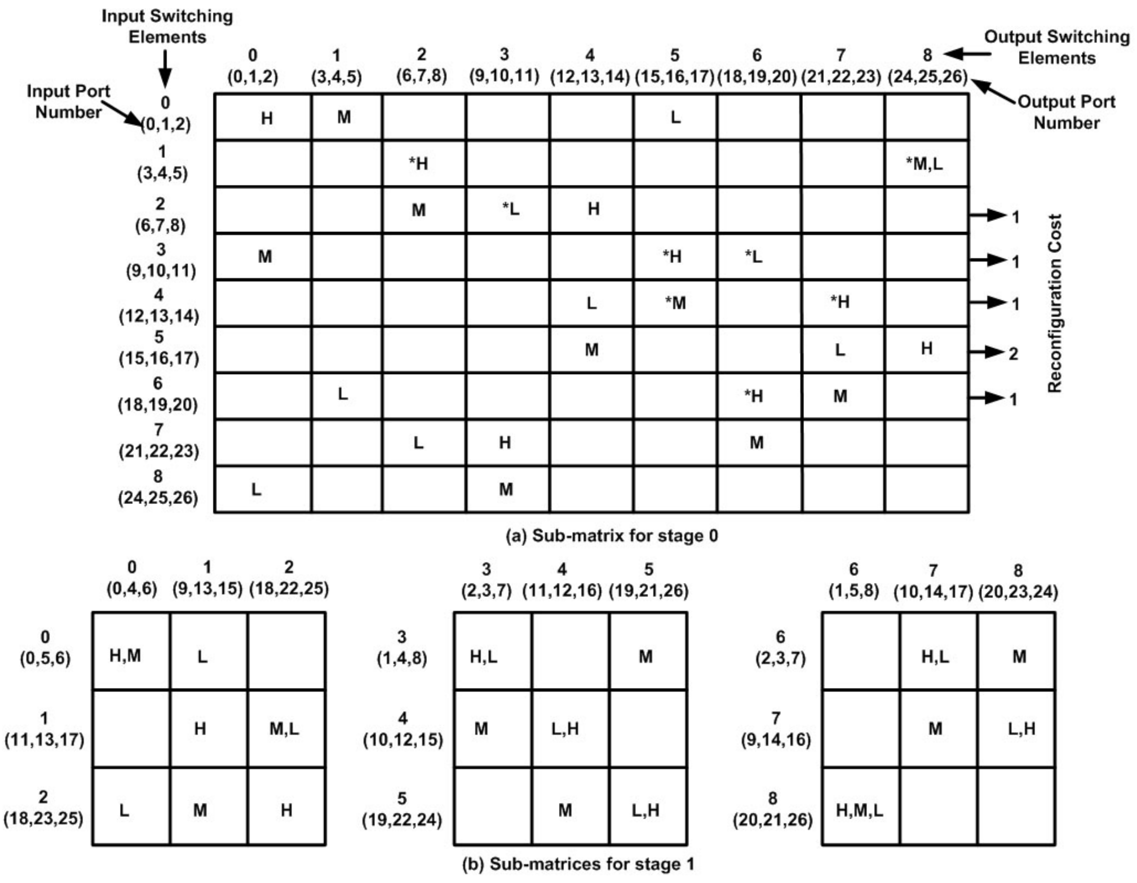
$$Tag[j,k] = \begin{cases} H, & if\{Row[j] \cup Column[k] \notin H\} \\ M, & if\{Row[j] \cup Column[k] \notin M\} \\ L, & if\{Row[j] \cup Column[k] \notin L\} \\ & Where\ j = \lfloor i/3 \rfloor\ ,\ k = \lfloor P(i)/3 \rfloor\ and\ 0 \leq i \leq (N-1) \end{cases} \tag{6.5}$$

In case of a conflict between two cell values between row$[j]$ and column$[k]$, the algorithm rearranges some of the cell values to overcome the conflict. Two different cell value rearrangements have been carried out to determine the smallest circuit length. Eqn 6.6 shows the rule for rearranging the values in the row and Eqn 6.7 shows the rule for rearranging the values in the column first. In these equations $j = \lfloor i/3 \rfloor$ and $k = \lfloor P(i)/3 \rfloor$. Depending on the smallest number of rearrangements cell values are changed. This process is similar to that detailed in Algorithm 13 and Algorithm 14.

$$Tag[j,k] = \{x|x = Row[j] \cap \{H, M, L\}\} \tag{6.6}$$

$$Tag[j,k] = \{x|x = Column[k] \cap \{H, M, L\}\} \tag{6.7}$$

To further illustrate the method assume that a permutation $P_{0:26} = (0\ 3\ 17\ 24\ 26\ 6\ 13\ 10\ 7\ 20\ 2\ 15\ 16\ 22\ 14\ 12\ 23\ 25\ 18\ 21\ 5\ 8\ 19\ 9\ 11\ 4\ 1)$ is to be routed. Fig 6.16 shows the execution of the algorithm for this permutation. Cells with a $(*)$ sign indicate rearrangements, where the initial cell values have been changed to resolve a conflict.


Input Switching Elements
Output Switching Elements
Input Port Number
Output Port Number
Reconfiguration Cost
(a) Sub-matrix for stage 0
(b) Sub-matrices for stage 1

	0 (0,1,2)	1 (3,4,5)	2 (6,7,8)	3 (9,10,11)	4 (12,13,14)	5 (15,16,17)	6 (18,19,20)	7 (21,22,23)	8 (24,25,26)	Reconfiguration Cost
0 (0,1,2)	H	M				L				
1 (3,4,5)			*H						*M,L	
2 (6,7,8)			M	*L	H					1
3 (9,10,11)	M					*H	*L			1
4 (12,13,14)					L	*M		*H		1
5 (15,16,17)					M			L	H	2
6 (18,19,20)		L					*H	M		1
7 (21,22,23)			L	H			M			
8 (24,25,26)	L			M						

	0 (0,4,6)	1 (9,13,15)	2 (18,22,25)
0 (0,5,6)	H,M	L	
1 (11,13,17)		H	M,L
2 (18,23,25)	L	M	H

	3 (2,3,7)	4 (11,12,16)	5 (19,21,26)
3 (1,4,8)	H,L		M
4 (10,12,15)	M	L,H	
5 (19,22,24)		M	L,H

	6 (1,5,8)	7 (10,14,17)	8 (20,23,24)
6 (2,3,7)		H,L	M
7 (9,14,16)		M	L,H
8 (20,21,26)	H,M,L		

Figure 6.16: *Execution of algorithm*

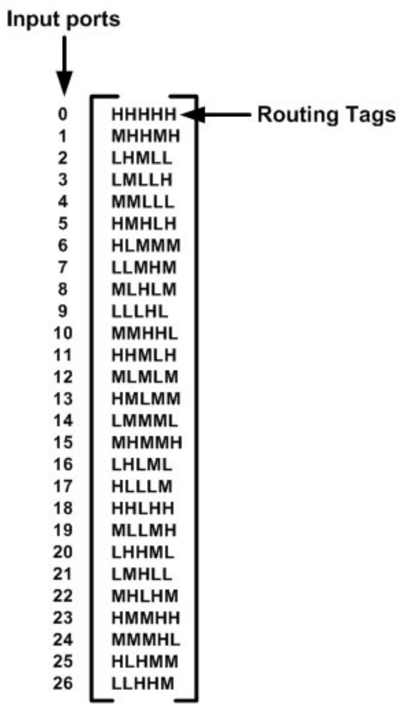


Figure 6.17: *The Routing tags obtained for the permutation* $P_{0:26}$

6.10 Simulation Results

This section provides simulation results for various values of N, where the metric used is reconfiguration cost.

Fig 6.17 shows the routing tags for the permutation used in Section 6.9. Fig. 6.18 shows the reconfiguration cost for the worst case rearrangement. Fig. 6.19 shows the reconfiguration cost per input for various values of N. This figure shows a gradual increase in the reconfiguration cost with increasing network size. The reconfiguration cost is higher than a network built with 2×2 switching elements. This illustrates the fact that finding conflict free routing paths for networks built with large switching elements require complex decision making.

6.11 Summary

This chapter proposes a routing algorithm that can provide zero probability of blocking in symmetric rearrangeable networks. A simple matrix decomposition

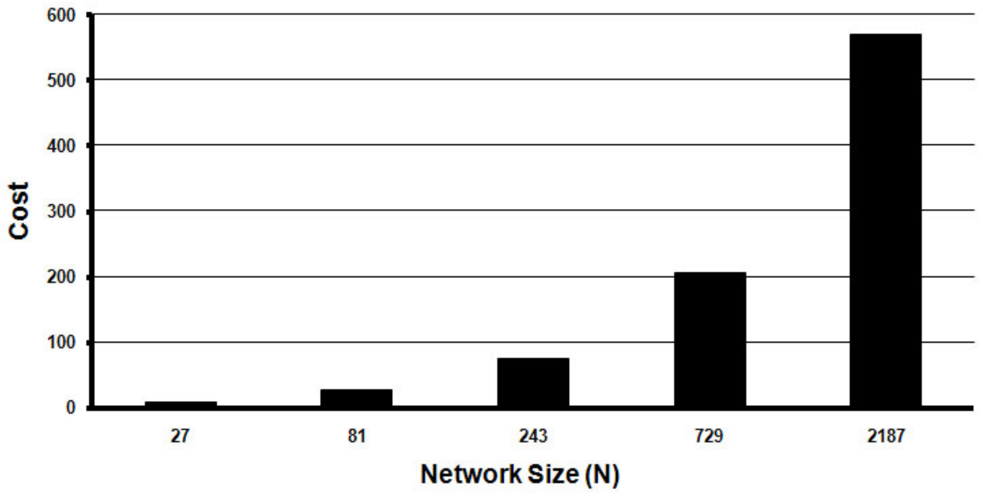


Figure 6.18: *Maximum reconfiguration cost for different sizes of network*

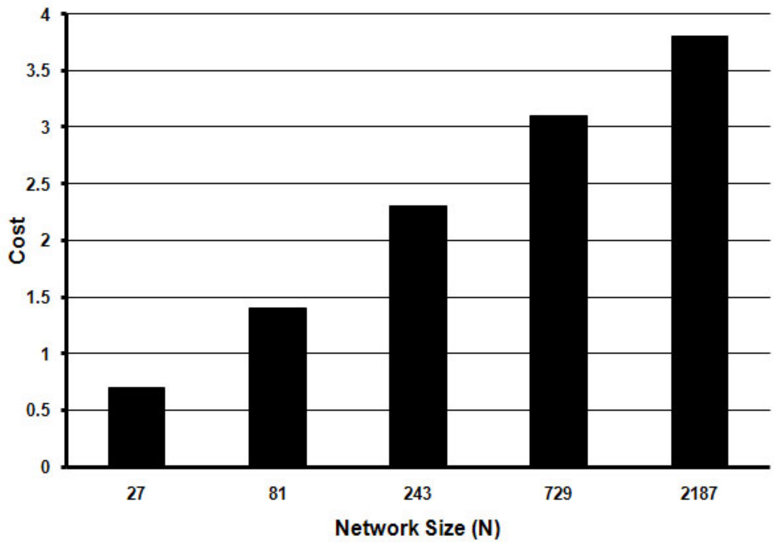


Figure 6.19: *Reconfiguration cost per input for different size of networks*

is used to map the actual network into a mathematical format. Two different rearrangement methods have been proposed and simulated. The result shows considerable reductions in the number of rearrangements. The method has almost similar execution time to that of the looping algorithm for small networks but for larger networks the proposed method has lower execution time than the looping algorithm. Parallel version of the algorithm and it time complexities have been described in detail. A proposal of using this method for optical domain switching has been described.

This chapter also discussed the issues of designing symmetric rearrangeable networks built with large and heterogenous switching elements. It is shown that networks built with 2×2 switching elements can be reduced in their depth by substituting 4×4 switching elements. Networks built with heterogenous switching elements show that a middle stage built with switching elements size ≤ 10 gives a lower crosspoint count per input than other alternatives. The analysis has been limited to networks built with 2×2, 3×3 and 4×4 switching elements. The extension to switching element sizes of grater size is not worthwhile, under the assumption that the implementation cost of an switching element is proportional to the square of the number of inputs. More complex modules of switch cost may justify the use of larger switching element sizes, although routing is much more complex in such case. This can be seen from the execution time for networks built with 3×3 switching elements. A network built with 3×3 switching elements gives the network designer a choice between long execution time and smaller network depth with limited hardware cost.

Major findings of this chapter are listed below:

- Proposed method dose not uses recursive method as in the looping algorithm, which is the base for all available routing methods reported in the literature [132, 137, 146, 150, 207].

- This method can work for partial permutation, in other words where some of the input remain inactive. Without major modifications algorithms such as the looping can not work for partial permutation and algorithm such as [128, 135, 146, 150] can only work with full permutations.

- It has parallel time complexity that matches other parallel methods in Table 3.2 for N active inputs. For partial permutations it has a time complexity of $O(\log N . \log \overline{m})$ in compared to $O(\log^2 \overline{m} + \log N)$ in [134]. Presented method can be used to generate semi permutations and used in optical domain. With the offered time complexity, this method is of particular interest in large scale optical switch routing for today's high performance communication networks.

- Minor modifications in the switching element setting scheme allows this method to be applicable for network having odd number of middle stage switching element and non binary decision making. Conventional routing methods such the general looping algorithm or algorithms such as [128, 132, 137, 146, 150, 207]designed for networks built with 2×2 switching elements can not work on these modified networks as decision making on these methods are controlled by binary bits.

- In serial processing, proposed method gives theoretical time complexity for N pair of request that is equal to all the other available routing method and which is the minimum that can be achieved for Beneš network in any single processor system. Simulation results also show that for large network it has better execution time than the looping algorithm.

CHAPTER 7

CONCLUSION AND FUTURE WORK

Switching technologies have changed much over time. Core switching networks and methods for routing through these networks play an important role in current communication networks. A special class of communication networks has been considered in this thesis. These are called symmetric rearrangeable networks. These networks are suggested for use in future communication networks; especially in the optical domain, system on chip (SoC), network on chip (NOC) and DSL applications.

7.1 Contributions

The overall contributions of this thesis are summarised below:

- A new design for implementing multistage symmetric repackable networks of minimum cost;
- A new design of a hybrid routing algorithm for symmetric rearrangeable

networks;

- A new algorithm for symmetric rearrangeable networks with zero probability of blocking routing;
- A new deign for symmetric rearrangeable networks built with large switching elements than 2×2.

It is not desirable to break the existing communication links in the process of finding an alternative path for a blocked request. To address this issue, this thesis proposed a repackable topology for rearrangeable networks. These repackable networks will identify the target paths that need to be rearranged and will transfer them to a bypass link before breaking their existing communication paths. This thesis proposes the minimum bypass links, possible number of link required to achieve a repackable network.

A hybrid routing algorithm has been proposed for symmetric rearrangeable networks. The hybrid method uses the looping algorithm in the outermost part of the network. In the inner part of the network, the routing decision is made logically based on the status of the switching element the signal is passing through. The execution time of this new method is superior to other popular methods. Zero blocking probability cannot be achieved with this method, but the blocking rate is much better for existing suboptimal other blocking algorithms. Mathematical time complexity shows that worst case time complexity of this algorithm is bounded by the limit set in the literature.

Deterministic methods suffers from the drawbacks of poor scalability and high execution time. In this thesis I proposed and implemented a new deterministic routing method that makes the routing decision faster than the looping algorithm for bigger networks, and gives similar performance to the looping algorithm for smaller networks. This algorithm has also been extended for use in networks built with bigger switching elements. This method can work for partial permutations without the need for dummy requests. Time complexity of this method is bounded by the limit set in the literature for full permutation for both serial and parallel processing domain. For partial permutation it has complexities better that other comparable methods. This method can also be used in the optical domain with networks having planar topologies and uses semi permutations.

Symmetric rearrangeable network built with large switching elements have also been considered in this research. It has been shown that networks built with 2×2 switching elements can be reduced in depth by using 4×4 switching elements. This process does not affect overall crosspoint count. Symmetric rearrangeable networks built with 3×3 switching elements have been also studied in this work. Since binary decision making is no longer valid for these networks, with increase in the size of the switching elements decision making gets more and more complex. This is due to the fact that with increase in inputs-outputs in a switching element possible options of routing also increases. Hence takes more time to make a valid decision.

7.2 Future Work

The research findings presented in this thesis may be extended in a number of ways, some of which are discussed below. In the hybrid routing algorithm, a possible extension is to support group addressing. This would make it viable to generate a new permutation if a request is blocked. The new permutation would assign the blocked port to a new output in the same group. Once a new permutation has been identified another attempt can be made to route the new permutation. Group addressing can be of relevant interest in systems where an interconnect is used to share resources among processing elements. It will then be possible to assign an unused resource to a processing element if no route to the initially assigned resource is available. This would provide an improvement in the overall blocking performance at the expense of an addition overhead. Another modification can be to implement some of the hybrid routing steps in parallel. Obviously in that case major modifications in the path search process of the algorithm would be required, to minimise the interactions between parallel routing procedures.

Since decision making in networks with large switching elements is quite complex, the use of suboptimal routing methods in these networks should be studied. The hybrid algorithm proposed in this thesis can be applied in such networks. In the future, building a switch using large switching elements may cost considerably less than the square-law model of switch cost indicates, and efficient methods of routing for such networks will be required.

It would be an interesting challenge to apply the methods proposed in this thesis to some real world networks. As it is outside the scope of this thesis, only results from simulation have been presented in this thesis. The destination of requests modelled in the thesis assumed on equal probability of selecting output ports, but simulation based on, permutation patterns captured from real world traffic would allow the network and its routing algorithm to be adapted to real applications. Any prospect of some collaborative research where there is a chance to apply these methods in real hardware would be of great interest.

BIBLIOGRAPHY

[1] Daesun Oh and Keshab K. Parhi. Low-complexity switch network for reconfigurable ldpc decoders. *IEEE Trans. Very Large Scale Integr. Syst.*, 18:85–94, January 2010.

[2] William Stallings. *Data and computer communications (8th ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

[3] S. Weinstein and P. Ebert. Data transmission by frequency-division multiplexing using the discrete fourier transform. *IEEE Transactions on Communication Technology*, 19(5):628 –634, 1971.

[4] D.T. Harvatin and R.E. Ziemer. Orthogonal frequency division multiplexing performance in delay and doppler spread channels. In *IEEE 47th Vehicular Technology Conference, 1997*, volume 3, pages 1644 –1647 vol.3, May 1997.

[5] Ivan B. Djordjevic and Bane Vasic. Orthogonal frequency division multiplexing for high-speed optical transmission. *Opt. Express*, 14(9):3767–3775, May 2006.

[6] C. Antal, J. Biro, T. Henk, and G. Matefi. Performance evaluation of a time division multiplexing method applicable for dynamic transfer mode net-

works. In *Fifth IEEE Symposium on Computers and Communications, 2000. Proceedings. ISCC 2000.*, 2000.

[7] D.M. Spirit, A.D. Ellis, and P.E. Barnsley. Optical time division multiplexing: systems and networks. *IEEE Communications Magazine*, 32(12):56 –62, December 1994.

[8] Mario M. Freire and Henrique J. A. da Silva. Performance assessment of high density wavelength division multiplexing systems with dispersion supported transmission at 10 gbit/s. *IEEE Symposium on Computers and Communications*, 0:318, 1997.

[9] M. Koshiba. Wavelength division multiplexing and demultiplexing with photonic crystal waveguide couplers. *Journal of Lightwave Technology*, 19(12):1970 –1975, December 2001.

[10] Qianfan Xu, Brad Schmidt, Jagat Shakya, and Michal Lipson. Cascaded silicon micro-ring modulators for wdm optical interconnection. *Optics Express*, 14(20):9431–9435, 2006.

[11] G. Jacobsen and P. Wildhagen. A general and rigorous wdm receiver model targeting 10-40-gb/s channel bit rates. *Journal of Lightwave Technology.*, 19(7):966, 2001.

[12] B.G. Lee, B.A. Small, Qianfan Xu, M. Lipson, and K. Bergman. Characterization of a 4 nbsp; times; nbsp; 4 gb/s parallel electronic bus to wdm optical link silicon photonic translator. *IEEE Photonics Technology Letters*, 19(7):456 –458, april1, 2007.

[13] Bell labs innovation briefs. *Bell Labs Technical Journal*, Spring 1997.

[14] K. Grobe, M. Wiegand, and J. McCall. Optical metropolitan dwdm networks an overview. *BT Technology Journal*, 20:27–44, October 2002.

[15] S. Mysore, R. Villa, and G. Beveridge. Performance of broadband dwdm networks. In *Electronic-Enhanced Optics, Optical Sensing in Semiconductor Manufacturing, Electro-Optics in Space, Broadband Optical Networks, 2000. Digest of the LEOS Summer Topical Meetings*, 2000.

[16] Ashwin Gumaste and Tony Antony. *Dwdm network designs and engineering solutions*. Cisco Press, 2002.

[17] S.D. Personick and W.O. Fleckenstein. Communications switchingfrom operators to photonics. *Proceedings of the IEEE*, 75(10):1380–1403, Oct. 1987.

[18] Youngsong Mun and Hee Yong Youn. Performance modeling and evaluation of circuit switching using clos networks. *IEEE Transactions on Computers.*, 43(7):854–861, 1994.

[19] M. Gerla, E. Leonardi, F. Neri, and P. Palnati. Routing in the bidirectional shufflenet. *IEEE/ACM Transactions on Networking*, 9(1):91 –103, feb 2001.

[20] D. Huynh, H. Kobayashi, and F. F. Kuo. Design issues for mixed media packet switching networks. In *AFIPS '76: Proceedings of the national computer conference and exposition June 7-10, 1976,*, pages 541–549, New York, NY, USA, 1976. ACM.

[21] P. Kirstein. The early history of packet switching in the uk [history of communications]. *IEEE Communications Magazine,*, 47(2):18–26, February 2009.

[22] John O. Limb and Dolors Sala. A protocol for efficient transfer of data over hybrid fiber/coax systems. *IEEE/ACM Transactions on Networking.*, 5(6):872–881, 1997.

[23] E.W.M. Wong, A.K.M. Chan, and T.-S.P. Yum. Analysis of rerouting in circuit-switched networks. *IEEE/ACM Transactions on Networking*, 8(3):419 –427, June 2000.

[24] E.W.M. Wong, A.K.M. Chan, and T.-S.P. Yum. Re-routing in circuit switched networks. In *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1373 –1379 vol.3, April 1997.

[25] H. Yamada, H. Kataoka, T. Sampei, and T. Yano. High-speed digital switching technology using space-division-switch lsi's. *IEEE Journal on Selected Areas in Communications,*, 4(4):529–535, Jul 1986.

[26] N. Yamanaka, S. Kikuchi, M. Suzuki, and Y. Yoshioka. A 2 gb/s expandable space-division switching lsi and network architecture for gigabit-rate broad-band circuit switching. *IEEE Journal on Selected Areas in Communications,*, 8(8):1543–1550, Oct 1990.

[27] Emil Hopner and Michael Allen Patten. The digital data exchange–a space-division switching system. *IBM J. Res. Dev.*, 28(4):444–453, 1984.

[28] S. Kikuchi and N. Yamanaka. An expandable time-division circuit switching lsi and network architecture for broadband isdn. *IEEE Journal on Selected Areas in Communications,*, 14(2):328–336, Feb 1996.

[29] H. Inose, Y. Yoshida, Y. Yasuda, and Z. Koono. A time slot interchange system in time-division electronic exchanges. *IEEE Transactions on Communications Systems*, 11(3):336 –345, 1963.

[30] H. Obara. Efficient parallel time-slot interchanger for high-performance sdh/sonet digital crossconnect systems. *Electronics Letters*, 37(2):81 –83, January 2001.

[31] Pablo Molinero-Fernández and Nick McKeown. Performance of circuit switching in the internet. *Journal on Optical. Networking.*, 2(4):83–96, 2003.

[32] Rajendran Parthiban, Christopher Leckie, Andrew Zalesky, Moshe Zukerman, and Rodney S. Tucker. Cost comparison of optical circuit-switched

and burst-switched networks. *Journal of Lightwave Technology.*, 27(13):2315–2329, 2009.

[33] M. Molle and G. Watson. 100base-t/ieee 802.12/packet switching. *IEEE Communications Magazine,*, 34(8):64 –73, August 1996.

[34] D. Russell. Alternative strategies for managing virtual circuit failure and recovery in local area computer networks. *IEEE Transactions on Communications,*, 30(6):1450–1454, Jun 1982.

[35] R.-H. Hwang and J.F. Kurose. On virtual circuit routing and re-routing in packet-switched networks. In *ICC '91 IEEE International Conference on Communications Conference Record., 1991.*, pages 1318–1323 vol.3, Jun 1991.

[36] Emmanouel A. Varvarigos and Jonathan P. Lang. A virtual circuit deflection protocol. *IEEE/ACM Transactions on Networking.*, 7(3):335–349, 1999.

[37] Lorenzo Aguilar. Datagram routing for internet multicasting. In *SIGCOMM '84: Proceedings of the ACM SIGCOMM symposium on Communications architectures and protocols*, pages 58–63, New York, NY, USA, 1984. ACM.

[38] Louis Pouzin. Virtual circuits vs. datagrams: technical and political problems. In *AFIPS '76: Proceedings of the national computer conference and exposition, June 7-10, 1976,*, pages 483–494, New York, NY, USA, 1976. ACM.

[39] W. J. Buchanan. *The handbook of data communications and networks*. Kluwer Academic Publishers, Norwell, MA, USA, 2005.

[40] J. Kanzow. Berkom-a global b-isdn communication system. In *Subscriber Loops and Services, 1988. Proceedings, ISSLS 88., International Symposium on*, pages 10–13, Sep 1988.

[41] Parviz Kermani and Leonard Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks (1976)*, 3(4):267 – 286, 1979.

[42] James W. Dolter, P. Ramanathan, and Kang G. Shin. Performance analysis of virtual cut-through switching in harts: A hexagonal mesh multicomputer. *IEEE Transactions on Computers*, 40:669–680, 1991.

[43] Ronald I. Greenberg and H.-C. Oh. Universal wormhole routing. *IEEE Transactions On Parallel And Distributed Systems*, 8(3):56–63, 1993.

[44] Lionel M. Ni and Philip K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26:62–76, 1993.

[45] Prasant Mohapatra. Wormhole routing techniques for directly connected multicomputer systems. *ACM Computing Surveys*, 30:374–410, 1998.

[46] George N. Rouskas and Lisong Xu. Optical packet switching, 2004.

[47] P. Pavon-Marino, J. Garcia-Haro, J. Malgosa-Sanahuja, and F. Cerdan. Optical packet switching fabrics comparison under scwp/shwp operational modes. *IEEE Symposium on Computers and Communications,*, 0:547, 2003.

[48] Farid Farahmand, Jason Jue, Vinod Vokkarane, Joel J. P. C. Rodrigues, and Mario M. Freire. A layered architecture for supporting optical burst switching. In *AICT-SAPIR-ELETE '05: Proceedings of the Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop*, pages 213–218, Washington, DC, USA, 2005. IEEE Computer Society.

[49] Hossam M. H. Shalaby. A simplified performance analysis of optical burst-switched networks. *Journal of Lightwave Technologyl.*, 25(4):986–995, 2007.

[50] Rajesh R. C. Bikram and Vinod M. Vokkarane. Tcp over optical burst switching: To split or not to split? *Journal of Lightwave Technology.*, 27(22):5208–5219, 2009.

[51] T. Feng. Fault tolerance in rearrangeable networks. In *TENCON 90. 1990 IEEE Region 10 Conference on Computer and Communication Systems*, pages 399–403 vol.1, Sep 1990.

[52] G. M. Masson, G. C. Gingher, and S. Nakamura. A sampler of circuit switching networks. *Computer*, 12(6):32–48, 1979.

[53] H.J. Siegel, W.G. Nation, C.P. Kruskal, and Jr. Napolitano, L.M. Using the multistage cube network topology in parallel supercomputers. *Proceedings of the IEEE*, 77(12):1932 –1953, dec 1989.

[54] Kenneth J. Thurber. *Distributed Processor Communication Architecture*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1979.

[55] Masayuki Murata Hideo Miyahara Yuji Oie, Tatsuya Suda. Survey of switching techniques in high-speed networks and their performance. *International Journal of Satellite Communications.*, 9(5):285–303, 1991.

[56] Assaf Shacham. *Architectures of Optical Interconnection Networks for High Performance Computing*. VDM Verlag, Saarbrücken, Germany, Germany, 2008.

[57] M. Collier. A systematic analysis of equivalence in multistage networks. *Journal of Lightwave Technology*, 20(9):1664–1672, Sep 2002.

[58] Chwei-King Mok and Nader F. Mir. An efficient interconnection network for large-scale computer communications applications. *Journal of Network and Computer Applications*, 23(2):59 – 75, 2000.

[59] L. Rodney Goke and G. J. Lipovski. Banyan networks for partitioning multiprocessor systems. *SIGARCH Computer Architecture News*, 2(4):21–28, 1973.

[60] D. H. Lawrie. Access and alignment of data in an array processor. *IEEE Transactions on Computers.*, 24(12):1145–1155, 1975.

[61] H. S. Stone. Parallel processing with the perfect shuffle. *IEEE Transactions on Computers.*, 20(2):153–161, 1971.

[62] Chuan-Lin Wu and Tse-Yun Feng. On a class of multistage interconnection networks. *IEEE Transactions on Computers*, C-29(8):694–702, Aug. 1980.

[63] C T A Lea. The load-sharing banyan network. *IEEE Transactions on Computers.*, 35(12):1025–1034, 1986.

[64] G. B. Adams and H. J. Siegel. The extra stage cube: A fault-tolerant interconnection network for supersystems. *IEEE Transactions on Computers.*, 31(5):443–454, 1982.

[65] A. Hopper and J. Wheeler. Binary routing networks. *IEEE Transactions on Computers.*, 28(10):699–703, 1979.

[66] K. Padmanabhan and D. H. Lawrie. A class of redundant path multistage interconnection networks. *IEEE Transactions on Computers.*, 32(12):1099–1108, 1983.

[67] Manoj Kumar and J. R. Jump. Performance of unbuffered shuffle-exchange networks. *IEEE Trans. Comput.*, 35(6):573–578, 1986.

[68] C. P. Kruskal and M. Snir. A unified theory of interconnection network structure. *Theory of Computer Science*, 48(1):75–94, 1986.

[69] Karanjeet Singh Kahlon. Sandeep Sharma, BansalP.K. On a class of multistage interconnection network in parallel pprocessing. *International Journal of Computer Science and Network Security*, 8(5), May 2008.

[70] Fong-Chih Shao and A. Yavuz Oruç. Efficient nonblocking switching networks for interprocessor communications in multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems .*, 6(2):132–141, 1995.

[71] Paul Feldman, Joel Friedman, and Nicholas Pippengers. Wide-sense non-blocking networks. *SIAM Journal on Discrete Mathematics.*, 1(2):158–173, 1988.

[72] Italo Busi and Achille Pattavina. Strict-sense non-blocking conditions for shuffle/exchange networks with vertical replication. In *INFOCOM*, pages 126–133, 1998.

[73] V. E. Benes. Mathematical theory of connecting networks. *Mathematics in Science and Engineering*, 17, 1965. New York: Academic.

[74] C.J. Smyth. Nonblocking photonic switch networks. *IEEE Journal on Selected Areas in Communications*, 6(7):1052 –1062, aug 1988.

[75] Yuanyuan Yang and Jianchao Wang. Wide-sense nonblocking clos networks under packing strategy. *IEEE Transactions on Computers.*, 48(3):265–284, 1999.

[76] F. H. Chang, J. Y. Guo, and F. K. Hwang. Wide-sense nonblocking for multi-logd n networks under various routing strategies. *Theoretical Computer Science.*, 352(1):232–239, 2006.

[77] W. Kabacinski and M. Michalski. Wide-sense nonblocking log2 $(n, 0, p)$ switching networks with even number of stages. In *IEEE International Conference on Communications*, volume 2, pages 1058–1062 Vol. 2, May 2005.

[78] G. Danilewicz, W. Kabacinski, M. Michalski, and M. Zal. Wide-sense nonblocking multiplane baseline switching networks composed of $d \times d$ switches. In *IEEE International Conference on Communications*, pages 6386–6391, June 2007.

[79] Yao-Ming Yeh and Tse-yun Feng. On a class of rearrangeable networks. *IEEE Transactions on Computers.*, 41(11):1361–1379, 1992.

[80] Nabanita Das. More on rearrangeability of combined (2n - 1)-stage networks. *Journal of Systems Architecture.*, 51(3):207–222, 2005.

[81] Nabanita Das, Krishnendu Mukhopadhyaya, and Jayasree Dattagupta. O(n) routing in rearrangeable networks. *Journal of Systems Architecture.*, 46(6):529–542, 2000.

[82] C. S. Raghavendra and Rajendra V. Boppana. On self-routing in benes and shuffle-exchange networks. *IEEE Transactions on Computers.*, 40(9):1057–1064, 1991.

[83] Zhen Chen, Zeng-Ji Liu, and Zhi-Liang Qiu. Bidirectional shuffle-exchange network and tag-based routing algorithm. *IEEE Communications Letters*, 7(3):121 – 123, march 2003.

[84] Jose Duato, Sudhakar Yalamanchili, and Ni Lionel. *Interconnection Networks: An Engineering Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[85] Charles E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.*, 34:892–901, October 1985.

[86] Andr DeHon. Practical schemes for fat-tree network construction. In *Advanced Research in VLSI: International Conference 1991*, pages 307–322. MIT Press, 1991.

[87] C. Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32:406–424, 1953.

[88] D. Slepian. Two theorems on a particular crossbar switching network. *Unpublished memorandum*, 1952.

[89] AM Duguid. Structural properties of switching networks. *Brown University Progress Report BTL-7*, 1959.

[90] M.C. Paull. Reswitching of connection networks. *Bell System Technical Journal 41*, 1962. pp. 833855.

[91] Rudolf G. Schehrer. On non-blocking multi-stage switching networks with rearrangement or repacking. *AEU - International Journal of Electronics and Communications*, 61(7):423 – 432, 2007.

[92] V. E. Benes. Permutation groups ,complexes and rearrangeable connecting networks. *Bell System Technical Journal*, 43:1619–1640, 1964.

[93] Abraham Waksman. A permutation network. *Journal of ACM*, 15(1):159–163, 1968.

[94] J. Konicek, T. Tilton, A. Veidenbaum, C. Zhu, E. Davidson, R. Downing, M. Haney, M. Sharma, P. Yew, P. Farmwald, D. Kuck, D. Lavery, R. Lindsey, D. Pointer, J. Andrews, T. Beck, T. Murphy, S. Turner, , and N. Warter. The organization of the cedar system. In *ICPP*, volume I. pp, 1991. 4956.

[95] Ching-Yi Lee and A. Yavuz Oruç. A fast parallel algorithm for routing unicast assignments in benes networks. *IEEE Transactions on Parallel and Distributed Systems.*, 6(3):329–334, 1995.

[96] K.Y. Lee. On the rearrangeability of a (2log n-1) stage permutation network. *IEEE Transactions on Computers.*, 34(5), 1985. pp.412425.

[97] Amitabha Chakrabarty, Martin Collier, and Sourav Mukhopadhyay. Adaptive routing strategy for large scale rearrangeable symmetric networks. *International Journal of Grid and High Performance Computing (IJGHPC)*, 2(2):53–63, 2010.

[98] C.Y. Lee. Analysis of switching networks. *The Bell System Technical Journal*, 34(6):1287 – 1315, Nov. 1955.

[99] C. Jacobaeus. A study on congestion on link systems. *Ericsson Technics*, 1950.

[100] Joseph Yu Hui. *Switching and Traffic Theory for Integrated Broadband Networks*. Kluwer Academic Publishers, Norwell, MA, USA, 1990.

[101] D. K. Hunter. Switching systems. *Encyclopedia of Information Technology*, 2000.

[102] D. G. Cantor. On non-blocking switching networks. *Networks*, 1(4):367–377, 1971.

[103] Riccardo Melen and Jonathan S. Turner. Nonblocking multirate networks. *SIAM Journal On Computing*, 18:301–313, 1989.

[104] Ren Kaixin and Gu Naijie. Permutation capability of optical cantor network. In *Parallel and Distributed Computing, Applications and Technologies, 2007. PDCAT '07. Eighth International Conference on*, pages 398 –404, dec. 2007.

[105] Ning Wang, Liren Liu, and Yaozu Yin. Cantor network, control algorithm, two-dimensional compact structure and its optical implementation. *Appl. Opt.*, 34(35):8176–8182, Dec 1995.

[106] Rajgopal Kannan. The kr-benes network: a control-optimal rearrangeable permutation network. *IEEE Transactions on Computers*, 54(5):534 – 544, may 2005.

[107] David M. Koppelman and A. Yavuz Oru. A self-routing permutation network. *Journal of Parallel and Distributed Computing*, 10(2):140 – 151, 1990.

[108] Ching Yuh Jan and A. Yavuz Oruç. Fast self-routing permutation switching on an asymptotically minimum cost network. *IEEE Trans. Comput.*, 42:1469–1479, December 1993.

[109] H. Cam and J.A.B. Fortes. A fast vlsi-efficient self-routing permutation network. *IEEE Transactions on Computers*, 44(3):448 –453, mar 1995.

[110] R. Melen. A general class of rearrangeable interconnection networks. *IEEE Transactions on Communications*, 39(12):1737 –1739, dec 1991.

[111] F. Bernabei, A. Forcina, and M. Listanti. On non-blocking properties of parallel delta networks. In *INFOCOM '88. Networks: Evolution or Revolution, Proceedings. Seventh Annual Joint Conference of the IEEE Computer and Communcations Societies, IEEE*, pages 326 –333, mar 1988.

[112] F. Bernabei and M. Listanti. Generalized parallel delta networks: a new class of rearrangeable interconnection networks. In *INFOCOM '89. Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies. Technology: Emerging or Converging, IEEE*, pages 219 –226 vol.1, apr 1989.

[113] Kyungsook Yoon Lee. On the rearrangeability of $2(\log n) - 1$ stage permutation networks. *IEEE Transactions on Computers*, C-34(5):412 –425, may 1985.

[114] H. El-Sayed and A. Youssef. Performance of the r-truncated benes networks under randomized routing algorithms. In *Parallel and Distributed Systems, 1997. Proceedings., 1997 International Conference on*, pages 104 –108, dec 1997.

[115] Abdou Youssef and Bruce W. Arden. A new approach to fast control of r^2xr^2 $3-$stage benes networks of rxr crossbar switches. In *ISCA*, pages 50–59, 1990.

[116] JIANG Xiaohong, HO Pin-Han, SHEN Hong, and HORIGUCHI Susumu. A class of benes-based optical multistage interconnection networks for crosstalk-free realization of permutations(fiber-optic transmission for communications). *IEICE transactions on communications*, 89(1):19–27, 2006-01-01.

[117] Xiaohong Jiang, Hong Shen, M.R. Khandker, and S. Horiguchi. Vertically

stacked benes networks for crosstalk-free permutation. In *Cyber Worlds, 2002. Proceedings. First International Symposium on*, pages 255 – 260, 2002.

[118] Yuanyuan Yang, Jianchao Wang, and Yi Pan. Permutation capability of optical multistage interconnection networks. *Journal of Parallel and Distributed Computing*, 60(1):72 – 91, 2000.

[119] R. Spanke. Architectures for guided-wave optical space switching systems. *IEEE Communications Magazine*, 25(5):42 – 48, may 1987.

[120] R. A. Spanke and V. E. Benes. N-stage planar optical permutation network. *Appl. Opt.*, 26(7):1226–1229, Apr 1987.

[121] Kenneth P. Bogart. *Introductory Combinatorics*. Halsted Press, New York, NY, USA, 1986.

[122] Dongwan Shin Junyu Peng Rainer Domer Daniel D. Gajski Andreas Gerstlauer, Gunar Schirner. System-on-chip component models. *Technical Report,University of California, Irvine, 2006*.

[123] Yuanyuan Yang, Jianchao Wang, and Yi Pan. Permutation capability of optical multistage interconnection networks. *J. Parallel Distrib. Comput.*, 60:72–91, January 2000.

[124] Daesun Oh and Keshab K. Parhi. Area efficient controller design of barrel shifters for reconfigurable ldpc decoders. In *ISCAS*, pages 240–243, 2008.

[125] D.C. Opferman and N.T. Tsao-Wu. On a class of rearrangeable switching networks part i: Control algorithm. *Bell System Technical Journal*, 50:579–1, 600, 1971.

[126] S. Andresen. The looping algorithm extended to base 2^t rearrangeable switching networks. *IEEE Transactions on Communications,*, 25(10):1057–1063, Oct 1977.

[127] David Nassimi and Sartaj Sahni. A self routing benes network. In *ISCA '80: Proceedings of the 7th annual symposium on Computer Architecture*, pages 190–195, New York, NY, USA, 1980. ACM.

[128] D. Nassimi and S. Sahni. A self-routing benes network and parallel permutation algorithms. *IEEE Transactions on Computers.*, 30(5):332–340, 1981.

[129] David Nassimi and Sartaj Sahni. An optimal routing algorithm for mesh-connected parallel computers. *Journal of ACM*, 27(1):6–29, 1980.

[130] C P Schnorr and A Shamir. An optimal sorting algorithm for mesh connected computers. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, pages 255–263, New York, NY, USA, 1986. ACM.

[131] A. Yavuz Oruç and M. Mittal. Setup algorithms for cube-connected parallel computers using recursive karnaugh maps. *IEEE Trans. Comput.*, 40:217–221, February 1991.

[132] S. T. Huang and S. K. Tripathi. Self-routing technique in perfect-shuffle networks using control tags. *IEEE Trans. Comput.*, 37:251–256, February 1988.

[133] D. Nassimi and S. Sahni. Parallel algorithms to set up the benes permutation network. *IEEE Transactions on Computers.*, 31(2):148–154, 1982.

[134] Ching-Yi Lee and A. Yavuz Oruç. A fast parallel algorithm for routing unicast assignments in benes networks. *IEEE Transactions on Parallel and Distributed Systems.*, 6(3):329–334, 1995.

[135] Ching-Yi Lee and A. Yavuz Oruc. Fast parallel algorithms for routing one-to-one assignments in benes networks. In *Proceedings of the 1993 International Conference on Parallel Processing - Volume 03*, ICPP '93, pages 159–166, Washington, DC, USA, 1993. IEEE Computer Society.

[136] T.-Y. Feng and S.-W. Seo. A new routing algorithm for a class of rearrangeable networks. *IEEE Transactions on Computers*, 43(11):1270 –1280, nov 1994.

[137] Seung-Woo Seo and Tse-yun Feng. A general inside-out routing algorithm for a class of rearrangeable networks. In *ICPP '94: Proceedings of the 1994 International Conference on Parallel Processing*, pages 17–20, Washington, DC, USA, 1994. IEEE Computer Society.

[138] Tse-Yun Feng and Linjiang Ma. A routing algorithm for modified omega+omega interconnection networks. In *Proceedings of the The 6th International Conference on Parallel Interconnects*, PI '99, pages 117–, Washington, DC, USA, 1999. IEEE Computer Society.

[139] M. K. Kim, H. Yoon, and S. R. Maeng. On the correctness of inside-out routing algorithm. *IEEE Transactions on Computers.*, 46(7):820–823, 1997.

[140] Dan M. Marom and David Mendlovic. Comment on "a new routing algorithm for a class of rearrangeable networks", 1997.

[141] Joseph Yu Hui. *Switching and Traffic Theory for Integrated Broadband Networks*. Kluwer Academic Publishers, Norwell, MA, USA, 1990.

[142] Nabanita Das, Krishnendu Mukhopadhyaya, and Jayasree Dattagupta. O(n) routing in rearrangeable networks. *J. Syst. Archit.*, 46:529–542, April 2000.

[143] Nabanita Das and Jayasree Dattagupta. Two-pass rearrangeability in faulty benes networks. *J. Parallel Distrib. Comput.*, 35:191–198, June 1996.

[144] T.T. Lee and S.Y. Liew. Parallel routing algorithms in benes-clos networks. *IEEE Transactions on Communications*, 50(11):1841 – 1847, nov 2002.

[145] J. Lenfant. Parallel permutations of data: A benes network control algorithm for frequently used permutations. *IEEE Transactions on Computers.*, 27(7):637–647, 1978.

[146] Tony T. Lee and Soung-Yue Liew. Parallel routing algorithms in benes-clos networks. In *Proceedings of the Fifteenth annual joint conference of the IEEE computer and communications societies conference on The conference on computer communications - Volume 1*, INFOCOM'96, pages 279–286, Washington, DC, USA, 1996. IEEE Computer Society.

[147] T.T. Lee and S.Y. Liew. Parallel routing algorithms in benes-clos networks. *IEEE Transactions on Communications*, 50(11):1841 – 1847, nov 2002.

[148] H. Cam and J.A.B. Fortes. Frames: a simple characterization of permutations realized by frequently used networks. *IEEE Transactions on Computers*, 44(5):695–697, May 1995.

[149] Hasan Çam. Rearrangeability of $(2n-1)$-stage shuffle-exchange networks. *SIAM Journal on Computing.*, 32(3):557–585, 2003.

[150] Hasan Çam and José A. B. Fortes. Work-efficient routing algorithms for rearrangeable symmetrical networks. *IEEE Transactions on Parallel and Distributed Systems.*, 10(7):733–741, 1999.

[151] N. Linial and M. Tarsi. Interpolation between bases and the shuffle-exchange network. *Europian Journal of Combinatorics*, 10:29–39, 1989.

[152] Fred S. Roberts. *Graph theory and its applications to problems of society*. Society for Industrial Mathematics, 1987.

[153] J. C. Fournier. Combinatorics of perfect matchings in plane bipartite graphs and application to tilings. *Theory of Computer Science*, 303(2-3):333–351, 2003.

[154] T.M. DuBois, B. Lee, Yi Wang, M. Olano, and U. Vishkin. Xmt-gpu: A pram architecture for graphics computation. In *37th International Conference on Parallel Processing, 2008. ICPP '08.*, pages 364–372, Sept. 2008.

[155] Steve Furber. *ARM System-on-Chip Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2000.

[156] Drew Wingard. Micronetwork-based integration for socs. In *In Proceedings of the 38th Design Automation Conference*, pages 673–677, 2001.

[157] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Comput. Surv.*, 38, June 2006.

[158] Kanishka Lahiri, Sujit Dey, and Anand Raghunathan. Evaluation of the traffic-performance characteristics of system-on-chip communication architectures. In *Proceedings of the The 14th International Conference on VLSI Design (VLSID '01)*, VLSID '01, pages 29–, Washington, DC, USA, 2001. IEEE Computer Society.

[159] Pierre Guerrier Alain and Alain Greiner. A generic architecture for on-chip packet-switched interconnections. pages 250–256, 2000.

[160] Pierre Guerrier and Alain Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '00, pages 250–256, New York, NY, USA, 2000. ACM.

[161] William J. Dally and Brian Towles. Route packets, not wires: on-chip inteconnection networks. In *Proceedings of the 38th annual Design Automation Conference*, DAC '01, pages 684–689, New York, NY, USA, 2001. ACM.

[162] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35(1):70 –78, jan 2002.

[163] Jiang Xu, Wayne Wolf, Joerg Henkel, Srimat Chakradhar, and Tiehan Lv. A case study in networks-on-chip design for embedded video. In *Proceedings of the conference on Design, automation and test in Europe - Volume 2*, DATE '04, pages 20770–, Washington, DC, USA, 2004. IEEE Computer Society.

[164] R. G. Gallager. Low density parity check codes. *IRE Trans. Inf. Theory*, IT-8(1):21–28, jan 1962.

[165] D. J. C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Trans. Inf. Theory*, 45(2):399431, jan 1999.

[166] Maurizio Martina, Guido Masera, Hazem Moussa, and Amer Baghdadi. On chip interconnects for multiprocessor turbo decoding architectures. *Microprocessors and Microsystems - Embedded Hardware Design*, 35(2):167–181, 2011.

[167] Federico Quaglio Guido Masera and Fabrizio Vacca. Low density parity check codes. *IEEE Transactions On Circuits And SystemsII: Express Briefs*, 54(6):542–546, june 2007.

[168] Martin Coiller. Private communications.

[169] L. Y. Lin, E. L. Goldstein, and R. W. Tkach. Free-space micromachined optical switches with submillisecond switching time for large-scale optical crossconnects. In *Wavelength Division Multiplexing Components*, page 152. Optical Society of America, 1999.

[170] Shi-Sheng Lee, Long-Sun Huang, Chang-Jin Kim, and Ming C. Wu. Free-space fiber-optic switches based on mems vertical torsion mirrors. *Journal on Lightwave Technology*, 17(1):7, 1999.

[171] J. Leuthold, Juerg Leuthold, Associate Member, Pierre andre Besse, Emil Gamper, Marcus Dlk, Stefan Fischer, Georg Guekos, and Hans Melchior. All-optical mach-zehnder interferometer wavelength converters and switches with integrated data- and control-signal separation scheme. *Journal on Lightwave Technology*, 17:1056–1066, 1999.

[172] H. Scott Hinton. *An introduction to photonic switching fabrics*. Plenum Press, New York, NY, USA, 1993.

[173] Mistuhiro Makihara, Makoto Sato, Fusao Shimokawa, and Yasuhide Nishida. Micromechanical optical switches based on thermocapillary integrated in waveguide substrate. *Journal on Lightwave Technology.*, 17(1):14, 1999.

[174] Victor Li, Chun Yin Li, and P. K. A. Wai. Alternative structures for two-dimensional mems optical switches. *Journal on Optical Networking.*, 3(10):742–757, 2004.

[175] De-Gui Sun, Ying Zha, Tiegen Liu, Ying Zhang, Xiaoqi Li, and Xiuhua Fu. Demonstration for rearrangeable nonblocking 8×8 matrix optical switches based on extended banyan networks. *Optics Express*, 15(15):9347–9356, 2007.

[176] Guido Maier and Achille Pattavina. Design of photonic rearrangeable networks with zero first-order switching-element-crosstalk. *IEEE Transactions on Communications*, 49:1268–1279, 2001.

[177] Gangxiang Shen, Tee Hiang Cheng, Chao Lu, Teck Yoong Chai, and Sanjay K. Bose. A novel rearrangeable non-blocking architecture for 2d mems optical space switches. *Optical Networks Magazine*, November-December 2002.

[178] Guomei Zhu and Geng-Sheng (G. S.) Kuo. A novel integrated multistage 2-d mems optical switch with spanke–benes architecture. *Journal of Lightwave Technology.*, 26(5):560–568, 2008.

[179] Xiaohua Ma and Geng-Sheng Kuo. A novel integrated multistage optical mems-mirror switch architecture design with shuffle benes inter-stage connecting principle. *Optics Communications*, 242(1-3):179 – 189, 2004.

[180] Shou-Heng Chen, Kuang-Chao Fan, Tien-Tung Chung, and Yao-Joe Joseph

Yang. A $n \times n$ architecture for 2-d mirror-type optical switches. *Journal of Lightwave Technology.*, 27(14):2843–2851, 2009.

[181] G. Maier and A. Pattavina. Design of photonic rearrangeable networks with zero first-order switching-element-crosstalk. *IEEE Transactions on Communications*, 49(7):1268 –1279, jul 2001.

[182] Rudolf G. Schehrer. On switching networks with rearrangement or repacking. *AEU - International Journal of Electronics and Communications*, 60(2):103 – 108, 2006.

[183] Martin H. Ackroyd. Call repacking in connecting networks. *IEEE Transactions on Computers.*, 27(3):589–591, 1979.

[184] Andrzej Jajszczyk and Grzegorz Jekel. A new concept-repackable networks. *IEEE Transactions on Computers.*, 41(8), 1993.

[185] A. Jajszczyk. Nonblocking, repackable, and rearrangeable clos networks: fifty years of the theory evolution. *IEEE Communications Magazine*, 41(10):28 – 33, oct 2003.

[186] Rudolf G. Schehrer. On non-blocking multi-stage switching networks with rearrangement or repacking. *AEU - International Journal of Electronics and Communications*, 61(7):423 – 432, 2007.

[187] M. Alioto and G. Palumbo. Interconnect-aware design of fast large fan-in cmos multiplexers. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 54(6):484 –488, june 2007.

[188] Hsu-Wei Huang, Cheng-Yeh Wang, and Jing-Yang Jou. Optimal design of high fan-in multiplexers via mixed-integer nonlinear programming. In *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*, ASP-DAC '04, pages 280–283, Piscataway, NJ, USA, 2004. IEEE Press.

[189] M. Alioto, G. Di Cataldo, and G. Palumbo. Optimized design of high fan-in multiplexers using tri-state buffers. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 49(10):1500 – 1505, oct 2002.

[190] Ming-Bo Lin. On the design of fast large fan-in cmos multiplexers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(8):963 –967, aug 2000.

[191] Kuang-Chao Fan, Wu-Lang Lin, Li-Hung Chiang, Shou-Heng Chen, Tien-Tung Chung, and Yao-Joe Yang. A 2x2 mechanical optical switch with a thin mems mirror. *Journal on Lightwave Technology.*, 27(9):1155–1161, 2009.

[192] S. Sohma, T. Watanabe, N. Ooba, M. Itoh, T. Shibata, and H. Takahashi. Silica-based plc type 32 x 32 optical matrix switch. In *Optical Communications, 2006. ECOC 2006. European Conference on*, pages 1 –2, sept. 2006.

[193] A.W. Poon, Xianshu Luo, Fang Xu, and Hui Chen. Cascaded microresonator-based matrix switch for silicon on-chip optical interconnection. *Proceedings of the IEEE*, 97(7):1216 –1238, july 2009.

[194] Ujval J. Kapasi, Scott Rixner, William J. Dally, Brucek Khailany, Jung Ho Ahn, Peter Mattson, and John D. Owens. Programmable stream processors. *Computer*, 36(8):54–62, 2003.

[195] A J Krygiel. Synchronous nets for single instruction stream multiple data stream computers. pages 186–193, 1986.

[196] Thomas T. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 1990.

[197] E. Lu and S.Q. Zheng. Parallel routing algorithms for nonblocking electronic and photonic switching networks. *IEEE Transactions on Parallel and Distributed Systems*, 16(8):702 – 713, aug. 2005.

[198] Enyue Lu. *PhD Thesis:Parallel Algorithms For High Performance Switching in Communication Networks*. The University Of Texas At Dallas, August, 2004.

[199] Chunming Qiao, R.G. Melhem, D.M. Chiarulli, and S.P. Levitan. A time domain approach for avoiding crosstalk in optical blocking multistage interconnection networks. *Journal of Lightwave Technology*, 12(10):1854 –1862, oct 1994.

[200] J. Zhou, M.J. O'Mahony, and S.D. Walker. Analysis of optical crosstalk effects in multi-wavelength switched networks. *IEEE Photonics Technology Letters*, 6(2):302 –305, feb 1994.

[201] Yi Pan, Chunming Qiao, and Yuanyuan Yang. Optical multistage interconnection networks: new challenges and approaches. *IEEE Communications Magazine*, 37(2):50 –56, feb 1999.

[202] Chunming Qiao. A universal analytic model for photonic banyan networks. *IEEE Transactions on Communications*, 46(10):1381 –1389, oct 1998.

[203] Xiaojun Shen, Fan Yang, and Yi Pan. Equivalent permutation capabilities between time-division optical omega networks and non-optical extra-stage omega networks. *IEEE/ACM Trans. Netw.*, 9:518–524, August 2001.

[204] Enyue Lu and S. Q. Zheng. Fast reconfiguration algorithms for ti space, and wavelength dilated optical benes networks. *Int. J. Parallel Emerg. Distrib. Syst.*, 22:39–58, January 2007.

[205] J. Sharony, K.W. Cheung, and T.E. Stern. Wavelength dilated switches (wds)-a new class of high density, suppressed crosstalk, dynamic wavelength-routing crossconnects. *IEEE Photonics Technology Letters*, 4(8):933 –935, aug 1992.

[206] Yuanyuan Yang and Jianchao Wang. Optimal all-to-all personalized ex-

change in a class of optical multistage networks. *IEEE Trans. Parallel Distrib. Syst.*, 12:567–582, June 2001.

[207] Seung-Woo Seo, Tse yun Feng, and Yanggon Kim. Conflict resolutions in the inside-out routing algorithm. *International Conference on Parallel Processing,*, 1:0026, 1996.