Order Number 8920578

High performance and high reliability multistage interconnection networks

Liu, Jyh-Charn, Ph.D.

The University of Michigan, 1989

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

HIGH PERFORMANCE AND HIGH RELIABILITY MULTISTAGE INTERCONNECTION NETWORKS

by

Jyh-Charn Liu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in The University of Michigan
1989

Doctoral Committee:

Professor Kang G. Shin, Chairman
Associate Professor John R. Birge
Professor John P. Hayes
Professor Ronald J. Lomax
Assistant Professor Pinaki Mazumder

RULES REGARDING THE USE OF

MICROFILMED DISSERTATIONS

Microfilmed or bound copies of doctoral dissertations submitted to The University of Michigan and made available through University Microfilms International or The University of Michigan are open for inspection, but they are to be used only with due regard for the rights of the author. Extensive copying of the dissertation or publication of material in excess of standard copyright limits, whether or not the dissertation has been copyrighted, must have been approved by the author as well as by the Dean of the Graduate School. Proper credit must be given to the author if any material from the dissertation is used in subsequent written or published work.

To my parents and my wife

ACKNOWLEDGEMENTS

I am most grateful to my thesis advisor, Professor Kang G. Shin, for his persistent inspiration, encouragements, and suggestion of challenging problems. His guidance and continuous support is the key element to the completion of this dissertation. His enthusiasm and dedication on research has always inspired me to pursue perfection throughout my dissertation research.

I would like to thank other committee members, Prof. John Birge, Prof. John Hayes, Prof. R. Lomax, and Prof. P. Mazumder for their constructive comments. I also like to thank B. J. Moganhan for her help on the preparation of this document. Frequent technical discussions with my colleagues at RTCL make this research a very pleasant experience.

My wife's moral support has been essential for me to cope with the frustrations of a graduate student. My parents' sacrifice to support my education always remind me of their love and care.

Finally, financial support for this dissertation research by the Office of Naval Research under contract N00014-85-0122 and the NASA under Grants NAG-1-296 and NAG-1-492 is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

LIST OF APPENDICES

Appendix

CHAPTER 1

INTRODUCTION

1.1 Research Background

Many high performance microprocessors implemented with very large scale integration (VLSI) technology have successfully entered the commercial market in recent years [54, 29, 30]. Multiprocessor systems built with these (relatively) low cost microprocessor chips have the potential to outperform conventional single instruction single data (SISD) supercomputers if computational parallelism can be exploited. The basic idea of parallel computing is to decompose a large task into subtasks, which are then executed in parallel by processors in a multiprocessor system. Since subtasks need to communicate with each other, a task's completion time is determined by its subtasks' execution time and their communication delay.

While it is relatively easy to increase computing power by adding more processors to a multiprocessor system, it is often very expensive to build a high performance communication network. Owing to their high performance, *multistage interconnection networks* (MINs) are studied in this dissertation. A typical system with a MIN which interconnects single-chip processors and memory modules is shown in Fig. 1.1. Since hardware cost is a major concern in system design, our first research topic is to improve the cost-effectiveness of MINs. In addition to the improvements on cost-effectiveness, our second research topic is to improve

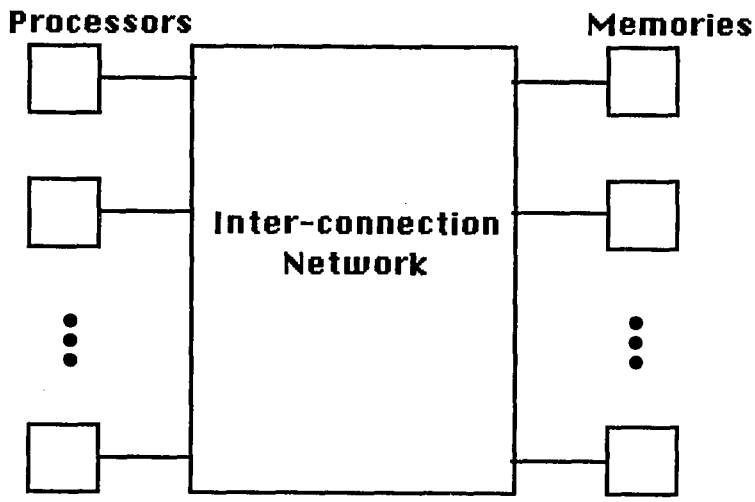


Figure 1.1: A multiprocessor system.

the reliability of MINs, because the large number of processors and/or memory modules connected by a MIN may become useless when the network fails. Since the architectures for improving performance/cost-effectiveness and reliability are compatible with each other, high network performance and reliability are expected when these two architectures are integrated in one network.

Research related to important features of MINs like performance analysis, fault-tolerance, and combinatorial capability, is surveyed next. An excellent introduction to the classification of interconnection networks can be found in [36]. MINs may be designed to be circuit switched or packet switched. In general, packet-switching MINs are suitable for short messages, whereas circuit-switching MINs are more suitable for massive long data streams [36, 88, 2, 112]. Thus, both types of networks have their unique advantages and disadvantages. System performance,

programming languages, and construction parameters have been extensively simulated to study the feasibility of MIN-based systems [28]. Several experimental prototypes have also been built, e.g., TRAC at the University of Texas Austin [82], the NYU Ultracomputer [48], the PASM computer at Purdue University [94, 14, 48], the Starnet [111], the Ceder project at the University of Illinois [43], and the RP3 project at IBM[80], to name a few. These prototypes can provide realistic system parameters for clock skew minimization [33, 41, 40], modular design [27, 39, 48, 14], and design optimization [53, 58] of large scale MINs.

The next important feature of MINs is their combinatorial capability. Many network topologies have been proposed such as perfect-shuffle [100], omega [62], flip [6], banyan [45], rearrangeable [26, 8], data manipulator [35], and indirect binary N-cube [78]. Permutation capability and routing control of different network topologies have been of intensive research interest [101, 19, 99, 115, 74, 100]. Wu and Feng showed that a class of important networks, such as modified data manipulator, baseline, flip, perfect shuffle, indirect binary N-cube, and regular SW Banyan networks, are topologically equivalent [110]. Boolean and arithmetic functions have also been incorporated into MINs [35].

Throughput analysis is one of the most active research topics. Benes introduced basic mathematical models of MINs [9]. Circuit-switching MINs have been analyzed with probabilistic models [11, 77, 114, 12], or Markov models [85, 102, 111]. It is intractable to analyze packet switching MINs due to the excessive number of system states. To simplify the analysis, it was usually assumed that stages in a MIN are independent, and queues in each switch have an infinite capacity [23]. These assumptions may yield good approximations when the actual buffer size is very large, or the network has light traffic[59, 60], but the dependency between stages cannot be ignored when the buffer size is small, and the network traffic is heavy [18, 57, 16].

In addition to the fault-tolerant design of switches [69, 44], the fault-tolerance capability

of MINs is improved by providing redundant paths to the system. Graph models have been proposed to denote paths in a MIN, and redundant paths can be derived from the redundancy graph [4, 76, 13, 86, 25, 84, 86, 63]. Some researchers proposed to replicate links without increasing the number of switches [25]. Redundant paths can also be obtained by adding additional stages to the network. Siegel *et al.* proposed to add one extra stage of switches and links while retaining the same combinatorial capability [1]. It has been proposed to decompose a packet-switching MIN into fanout, transferring, and merging stages [23, 24]. Analysis showed that very little performance improvement can be gained when more than two extra stages are added. Thus, adding more stages to a MIN will improve its fault-tolerance capability but decrease its throughput.

Two major processes, fault diagnosis and fault recovery[96], are needed for improving network reliability. Since fault diagnosis is the most expensive and difficult process, we will focus on issues related to it in this dissertation. A system's reliability will be degraded seriously if the system does not have efficient fault-detection mechanisms[91, 105]. In particular, single faults should be detected and repaired as soon as possible before they are accumulated into multiple faults, which are far more difficult to handle [92, 66].

It is difficult to test MINs, because they have a large number of components and I/O ports, and each I/O port is connected to an independent processor. Any testing method should have the following features: (1) applicable when a large number of processors and/or network components are faulty, (2) extensible to large scale networks, (3) can be easily handled by each processor, and (4) can be applied to multi-path networks. Fault masking [61, 108, 98, 46, 51] and coding techniques [42, 67] can effectively combine fault detection and fault recovery processes, but are too expensive for MINs due to the large number of components in MINs. Thus, we will develop easily testable architectures [10, 109] for periodical network testing during normal operation, and for fast fault diagnosis in case the system operation has been

stopped. Testing of circuit-switching baseline MINs with 2×2 switches has been studied thoroughly [38, 3, 37]. Shen *et al.* [64] modeled a CSN using 2×2 switches as an iterative logic array (ILA), which can be tested with a constant number of patterns. Leu and Agrawal used dynamic full accessibility [90, 89] to develop a high level diagnosis method for MINs [5]. Lim proposed the use of testing packets to exercise a network periodically [68]. Several researchers have proposed adaptive testing strategies for packet switching MINs with different topologies [106, 21, 71, 75, 20, 70, 104, 103]. Since adaptive procedures usually imply centralized testing evaluation and require human assistance, such a strategy is very inefficient for multiprocessor systems.

1.2 Organization of the Dissertation

To develop cost-effective high performance and high reliability MINs, three major research topics, as listed in Fig. 1.2, have been investigated in this dissertation. The correlation of these three topics are introduced as follows.

The cost of a MIN with N input/output ports is $N \log_2 N$. The primary technique to reduce the network size is to group processors/memory-modules into *clusters* to reduce the number of I/O ports of the network. When the clustering technique is applied to a packet switching MIN, the number of processors (memory-modules) in a cluster can be determined by matching the bandwidth of processor (memory) clusters with the bandwidth of buffers in input (output) stage[1]. On the other hand, there is no buffer in a circuit switching network. It will be shown in **Chapter 2** that the clustering technique alone can only provide very limited performance improvement in circuit switching MINs. A more sophisticated *network overlapping and memory interleaving* (NOMI) technique can improve the system performance at reduced cost.

[1] As will be seen in Chapter 4, each stage in the network may have a different bandwidth.

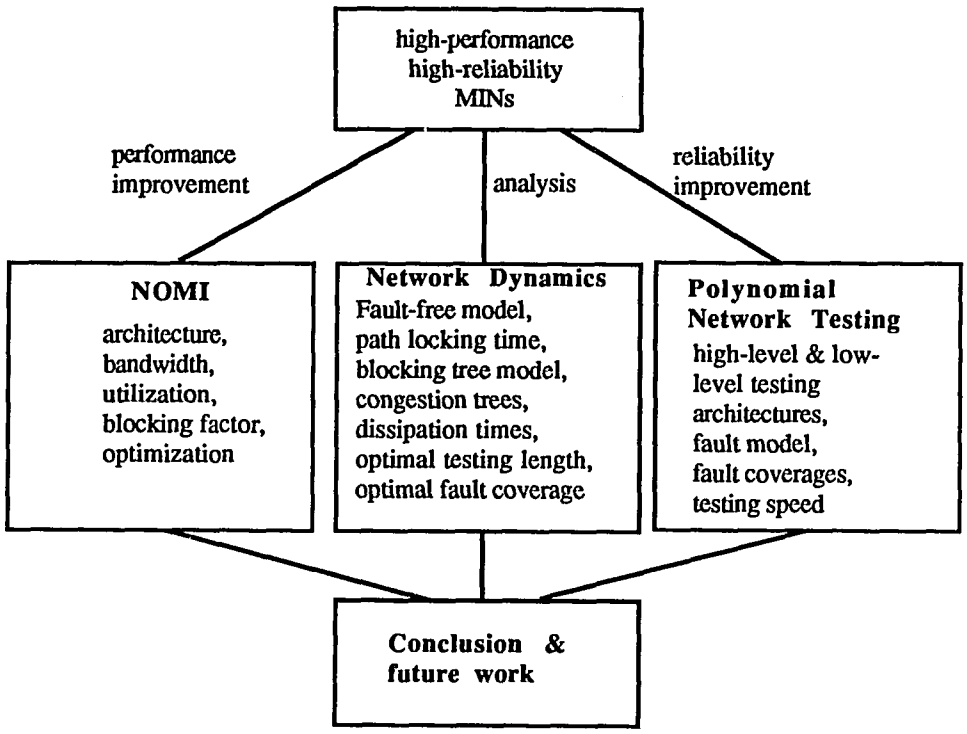


Figure 1.2: Block diagram of three research topics.

When the network size is reduced by the clustering technique, the performance loss caused by a failed network component is more serious, because more processors will be affected by the failed component. As mentioned earlier, since most fault masking and coding methods are too expensive for MINs, we will develop easily testable architectures for packet switching MINs in **Chapter 3**. The first easily testable architecture supports high-level network testing, which is periodically applied to the network during normal operation. The second architecture supports low-level network testing, which is for fast fault diagnosis after stopping normal operation completely. These two architectures can be easily modified to be used in circuit switching networks, because circuit switching networks can be considered as packet switching networks with no buffers.

In a packet-switching network, a path needs to be locked up before it is tested by the high-level testing method. To avoid excessive performance loss during network testing, it is important to understand the network dynamics such as the probability of the network being congested, and the dissipation time of a congested network. Thus, the analysis of network dynamics and the optimization of concurrent testing strategies are studied in **Chapter 4.** The first objective is to minimize the performance loss caused by testing. Performance loss is minimized by applying test patterns in small batches, not in a long stream, to avoid blocking an excessive number of packets. The other objective is to trade the mean fault detection time for fault coverage. This objective is justified because (1) fault coverage increases with the length of test patterns, and (2) the probability of system crash increases with the mean fault detection time.

Several concluding remarks are made in **Chapter 5.**

CHAPTER 2

NETWORK OVERLAPPING WITH MEMORY INTERLEAVING

2.1 Introduction

In conventional MINs,[1] a physical path between a source and destination must be established for data transmission. As shown in Fig. 2.1, a conventional MIN contains a *forward* network and a *backward* network. Requests/data generated by processors are routed through the forward network to memory modules, and service results are returned to the processor through the backward network. The utilization rate in conventional MINs is low, because a forward path and a backward path must be locked up simultaneously before the service of a request is completed.

Pipelined MINs have been proposed to improve network performance [95, 41]. A burst of data can be transmitted once a pipeline in the network is established. However, when the size of data burst is not large, the performance of a pipelined MIN could be worse than a conventional MIN due to setup overhead. Both asynchronous and synchronous multiplexing have been widely used in telecommunication systems [55, 7]. Asynchronous control schemes are less useful for large systems for their long delay. Although the synchronous multiplexing technique has been widely used in telephone switches [72, 55, 7], the depth of multiplexing

[1] We focus on circuit switching MINs in this chapter, thus, the term "circuit switching " will be omitted.

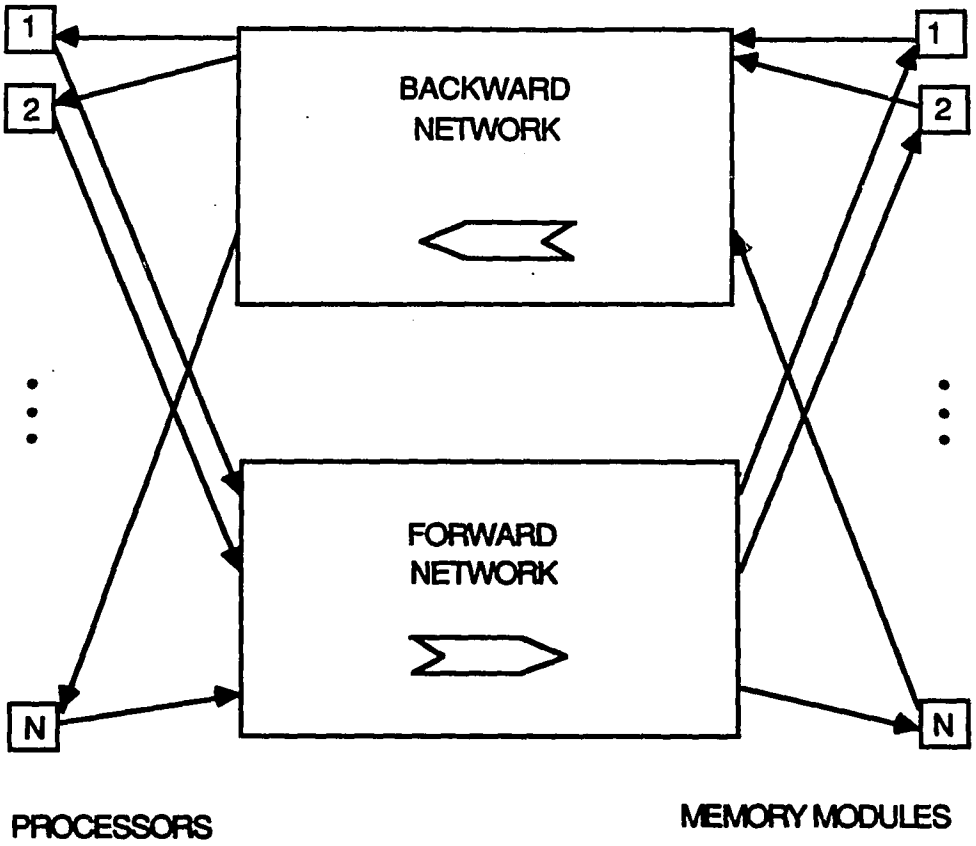


Figure 2.1: An $N \times N$ multiprocessor system connected by a conventional MIN.

in telephone switches is limited so as to guarantee the quality of voice signals. The *network overlapping* and *memory interleaving* (NOMI) technique proposed in this chapter is a type of synchronous multiplexing for large multiprocessor systems. For convenience, a MIN equipped with the NOMI technique is called an *Overlapped Circuit Switching Multistage Interconnection Network* (OCSMIN).

The NOMI technique may increase or decrease the mean system waiting time in an OCSMIN based system. To decrease system waiting time, the size of switches can be increased to lower the network's blocking probability. To improve the network performance at a reduced cost, a branch and bound procedure is developed to find optimal combinations of switch size with the depth of overlapping/interleaving.

The rest of this chapter is organized as follows. Section 2.2 describes operating principles of the NOMI technique. Hardware cost and performance of conventional MIN and OCSMIN are compared in Section 2.3 and Section 2.4, respectively. Optimization procedures are discussed in Section 2.5. Concluding remarks on the NOMI technique are given in Section 2.6.

2.2 Operational Principles of NOMI

Clustering is the first step in the design of OCSMINs. A *processor cluster* pc_i is a set of processors p_{ij}, $1 \leq j \leq w$, and an interface unit pn_i between these processors and the network, i.e., $pc_i \equiv \{pn_i,\ p_{ij} |\ 1 \leq j \leq w\}$ for all $1 \leq i \leq N'$, where N is the number of processors in the system, $N' = N/w$ is the *height* of the network, and w is the number of processors in a cluster. The processor subsystem is the collection of processor clusters, $PS \equiv \{pc_i |\ 1 \leq i \leq N'\}$. Similarly, the memory subsystem MS is the collection of memory clusters, i.e., $MS \equiv \{mc_i |\ 1 \leq i \leq N'\}$, where $mc_i \equiv \{mn_i,\ m_{ij} |\ 1 \leq j \leq w\}$, m_{ij} is the j-th memory module, and mn_i is the interface unit at cluster i, respectively. Note that a conventional MIN can be considered as an OCSMIN with $w = 1$ and $N = N'$.

conventional MIN can be considered as an OCSMIN with $w = 1$ and $N = N'$.

Fig. 2.2 shows a hierarchical system structure containing processor and memory clusters. A $\lceil \log_r N' \rceil$ stage network must be used to connect processor and memory clusters. In the forward network, a *forward switch* is denoted by $FS_{ij}(m,l)$, where ij is the coordinate of the switch, and m and l represent the m-th input port and l-th output port, respectively. (m and l will be omitted whenever they do not cause any ambiguity.) Similarly, in the backward network, $BS_{ij}(m,l)$ denotes a *backward switch* where m represents the m-th output port and l the l-th input port. The set of switches and links in the network can be represented by $\{FS_{ij},\ BS_{ij} \mid 1 \le i \le N',\ 1 \le j \le k\}$, and $\{FL_{ij},\ BL_{ij} \mid 1 \le i \le N',\ 1 \le j \le k+1\}$, respectively, where FL_{ij} (BL_{ij}) is a link in the forward (backward) network. The forward and backward networks are *topologically identical* if they have identical network structures except for the direction of routing.

Definition 1 : When the forward and backward networks are topologically identical, the switch BS_{ab} is called the *partner* of FS_{cf}, $BS_{ab} \equiv \Pi(FS_{cf})$, when $a = c$, $b = f$, and BS_{ab} is set up for later servicing of all the requests passing through FS_{cf}.

The partner of a forward (backward) link, which is also a backward (forward) link, can be defined in a similar way. Interconnections between processors and memory clusters can be represented by the *Interconnection Relation* :

$$IR \equiv \{(pc_i,\ {}_iFP_j,\ mc_j,\ {}_jBP_i) \mid 1 \le i,\ j \le N'\},$$

where ${}_iFP_j \equiv \{(pn_i, FS_{1x_1}(i_1,o_1),\ FL_{1y_1}, FS_{2x_2}(i_2,o_2),\ FL_{2y_2}, \cdots,\ FS_{kx_k}(i_k,o_k),\ FL_{k+1y_{k+1}},\ mn_j)\}$ is the set of all paths from pc_i to mc_j. Similarly, ${}_jBP_i \equiv \{(\ pn_i,\ BS_{1x_1}(l_1,r_1),\ BL_{1y_1},\ BS_{2x_2}(l_2,r_2),\ BL_{2y_2}, \cdots, BS_{kx_k}(l_k,r_k),\ BL_{k+1y_{k+1}},\ mn_j\ )\}$ represents all possible paths from mc_j to pc_i. In addition to the relation *IR*, a dynamic model is needed to distinguish conventional MINs from OCSMINs. At time instant l, the request pattern $\Lambda(l)$ is

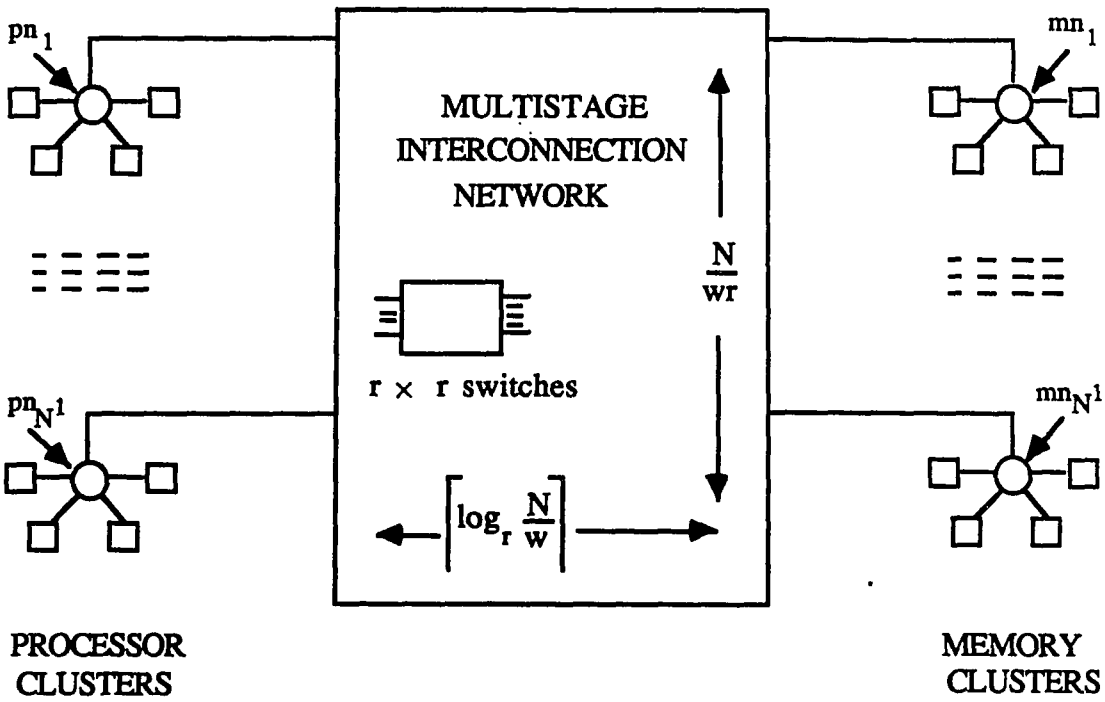


Figure 2.2: The hierarchical structure of a multiprocessor system.

represented by an N'−tuples $\{(a_1(t),\ a_2(t),\cdots,\ a_{N'}(t))|\ a_i(t) \in \{0,1,\cdots,N'\},\ 1 \le i \le N'\}$, where $a_i(t) = j$ if pc_i requests to access mc_j at time t, and $a_i(t) = 0$ if no request is made by pc_i. For each request $a_i(t)$, there is a corresponding *forward path* in the forward network,

$$ {}_iFP_j(t) = \begin{cases} {}_jFP_j & \text{if} a_i(t) = j \text{ and the request is granted} \\ \phi & \text{if } a_i(t) = 0 \text{ or not all resources available at time t} \end{cases} $$

and the corresponding *backward path*

$$ {}_jBP_i(t) = \begin{cases} {}_jBP_i & \text{if} a_i(t) = j \text{ and the request is granted} \\ \phi & \text{if } a_i(t) = 0 \text{ or not all resources available at time t} \end{cases} $$

We can define the *Forward Interconnection Pattern*,

$$ FIP(t) \equiv \{\ {}_iFH_j(t) \mid 1 \le i,\ j \le N'\}\ , \text{ where } {}_iFH_j(t) = (a_i(t),\ {}_iFP_j(t),\ mc_j). $$

and the *Backward Interconnection Pattern* as

$$ BIP(t) \equiv \{\ {}_jBH_i(t) \mid 1 \le i,\ j \le N'\}\ , \text{ where } {}_jBH_i(t) = (a_i(t),\ {}_jBP_i(t),\ mc_j). $$

At time t, $FIP(t)$ and $BIP(t)$ are established paths on the forward and backward networks, respectively. In a conventional MIN, ${}_jBH_i$ and its partner ${}_iFH_j$ are simultaneously locked up for a set of requests, and thus, $BIP(t) = \Pi(FIP(t))$, $\forall t$. Operations of MINs can be represented by the *Total Interconnection Pattern*: $TIP(t) \equiv FIP(t) \times BIP(t)$ $\equiv \{(a_i(t),\ {}_iFP_j(t),\ mc_j,\ {}_jBP_i(t))\}$.

Definition 2 : For $1 \le i,\ j \le N'$, the function $CSF : A(t) \to TIP(t)$ is the interconnection function of a conventional MIN, such that

(1) $CSF(a_i(t)) = {}_iFH_j(t) \times {}_jBH_i(t)$,

(2) ${}_iTIP_j(t)$ is unique for each $a_i(t)$, and ${}_{i_1}FP_{j_1}(t) \cap {}_{i_2}FP_{j_2}(t) = \phi \quad \forall\ i_1 \neq i_2$,

(3) ${}_iBP_j(t) = \Pi({}_jFP_i(t))$.

When ${}_{i_1}FP_{j_1} \cap {}_{i_2}FP_{j_2} = \phi$, requests $a_{i_1}(t)$ and $a_{i_2}(t)$ do not have resources contention. In a conventional MIN, as illustrated in Fig. 2.3, components locked in the forward network are idle during periods T2 and T3, and gates and links locked in the backward network are idle during T1 and T2. To increase resource utilization rate, we can relax the total interconnection relation CSF into a new operation OVF:

Definition 3 : The function $OVF : A(t) \rightarrow TIP(t)$ is an interconnection function of the OCSMIN satisfying following conditions:

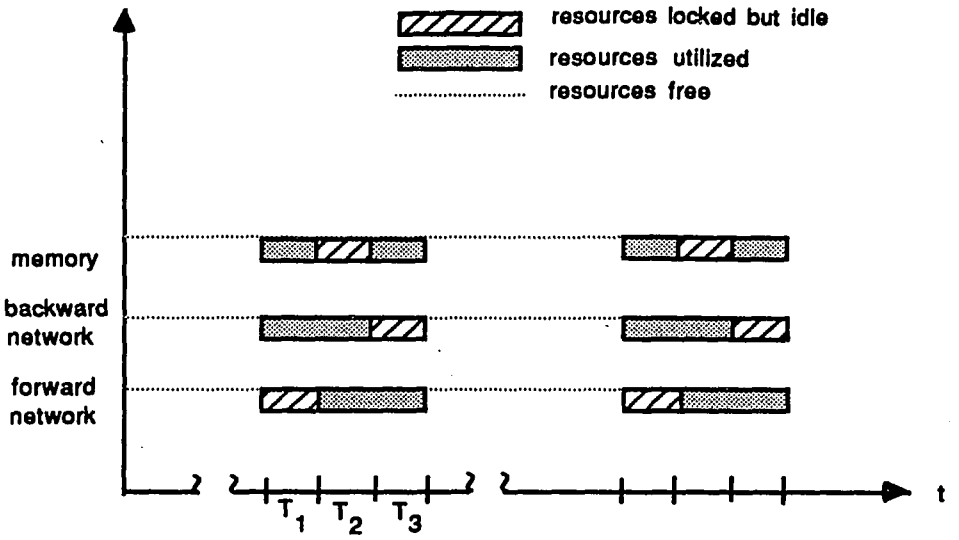
(1) $OVF(a_i(t)) = {}_iFH_j(t) \times {}_{j'}BH_{i'}(t)$, i and j need not be equal to i' and j',

(2) ${}_iTIP_j(t)$ is unique for each $a_i(t)$, and ${}_{i_1}FP_{j_1}(t) \cap {}_{i_2}FP_{j_2}(t) = \phi$ if $i_1 \neq i_2$.

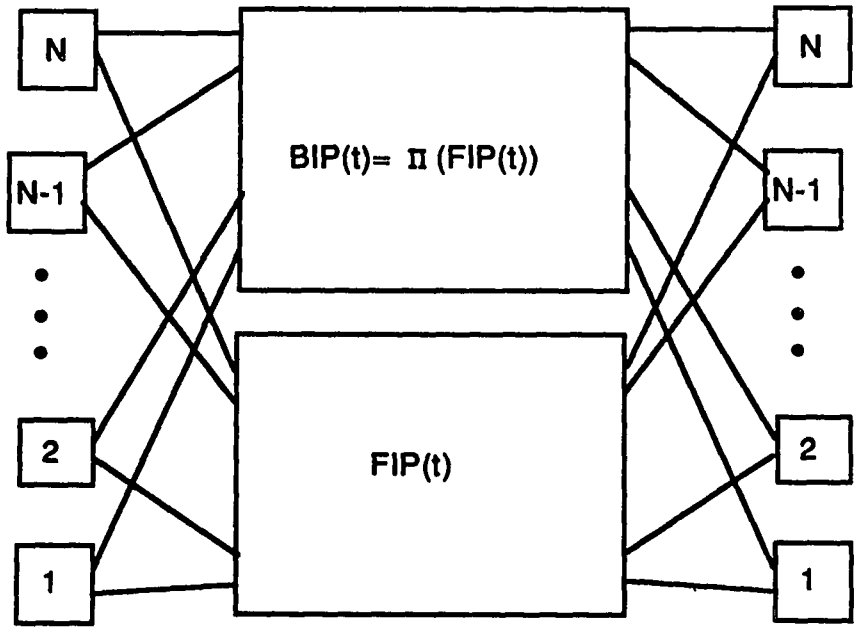
(3) ${}_jBP_i(t) = \Pi({}_iFP_j(t - \Delta t))$ for all t and some fixed Δt.

Functions CSF and OVF describe operational principles of conventional MINs and OCSMINs, respectively. The timing chart and interconnection function of an OCSMIN are plotted in Fig. 2.4. Note that at time t, a processor/memory interface may appear in both ${}_iFH_j$ and ${}_jBH_i$ while serving different requests. The maximum number of requests that can be routed through one path within Δt is called the number of *phases, w*, or the depth of NOMI. The *network cycle* or *network propagation delay* is the time to route one request through the network.

The NOMI technique reserves the partner of a established forward path for use after a delay of Δt, and an established path in forward network is immediately released after the processor's data are transmitted into the destination memory module.



(a)



(b)

Figure 2.3: **Illustration of a conventional MIN. (a) A timing chart. (b) Its interconnection function.**

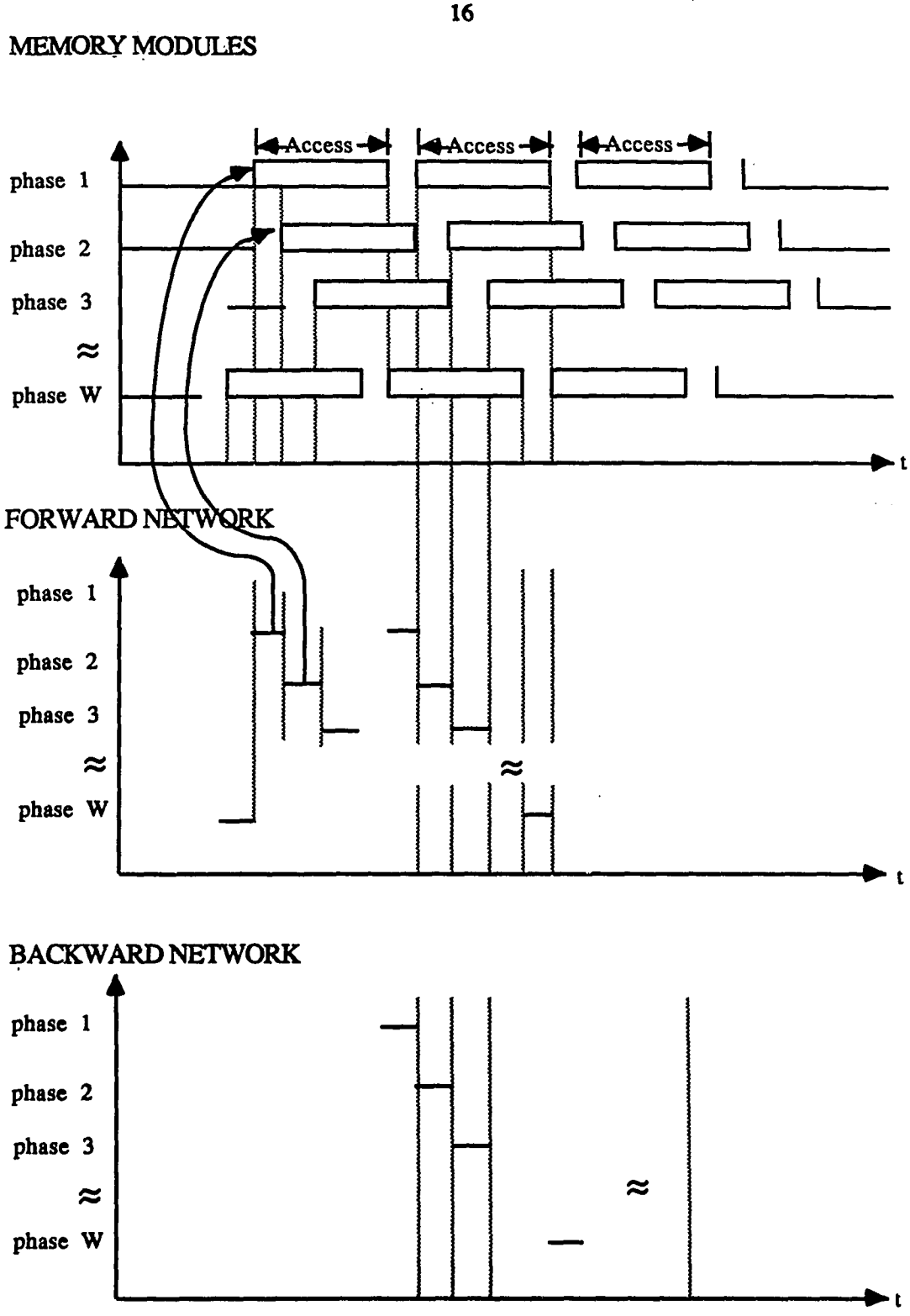


Figure 2.4: **Illustration of an OCSMIN. (a) A timing chart.**

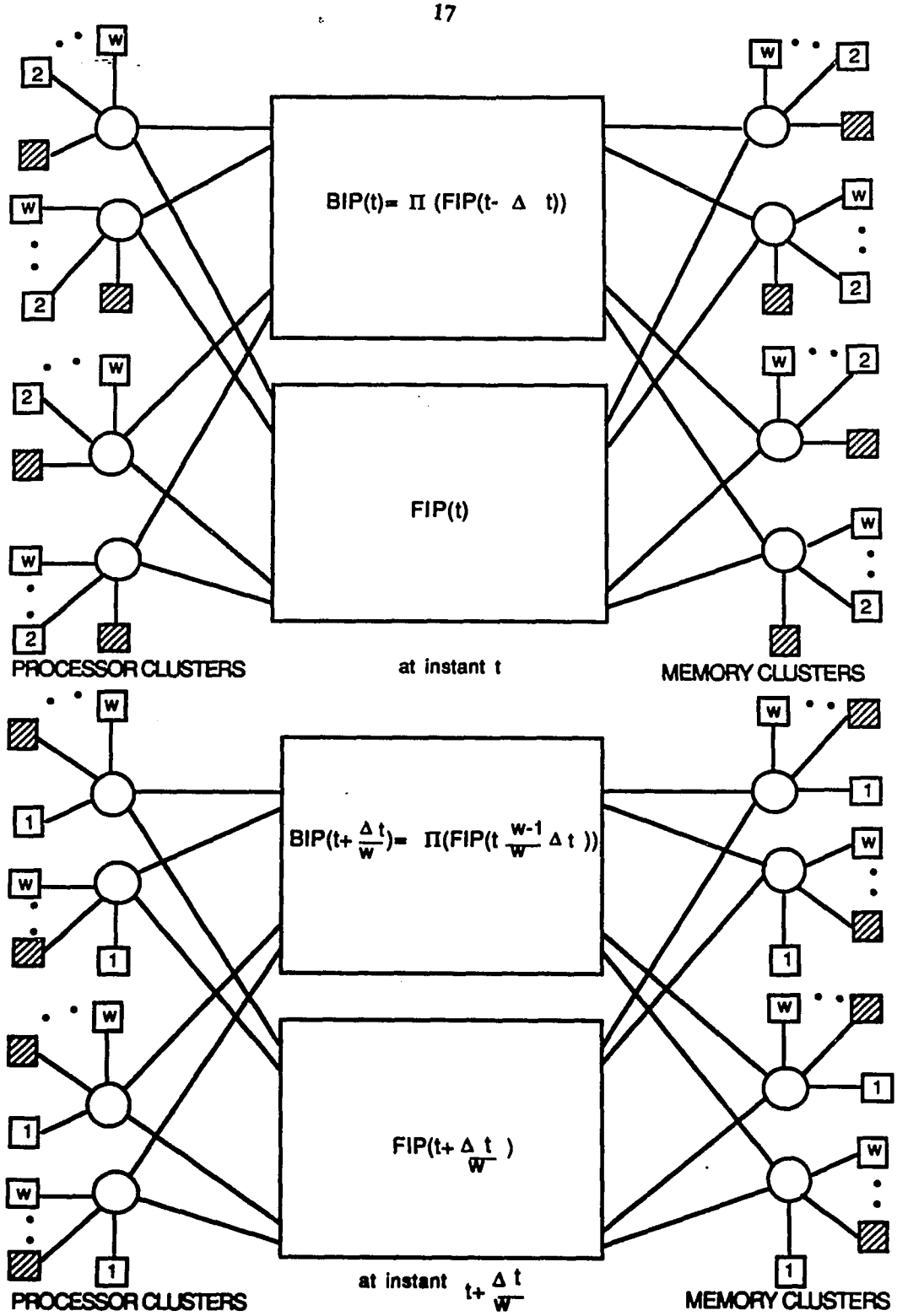


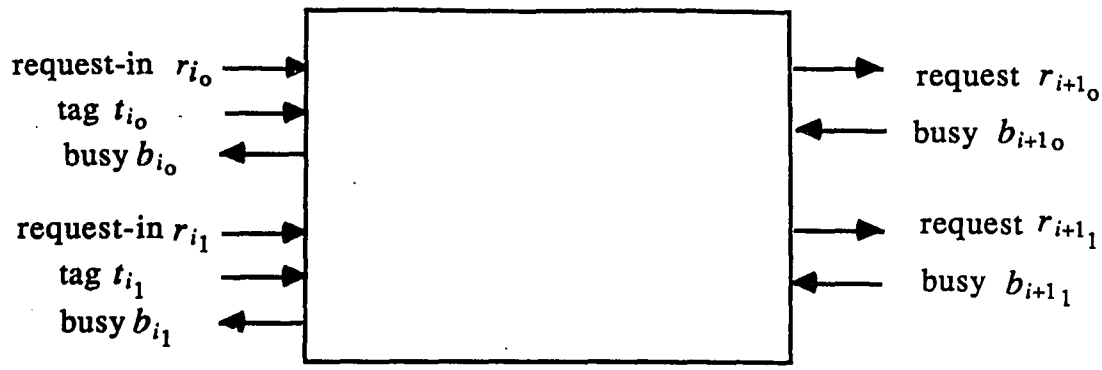
Figure 2.4(b)The interconnection function of an OCSMIN.

OVF can be implemented by the NOMI technique if and only if $\Delta t = T_M$, and $w \times T_D = T_M$, where T_D is the network propagation delay, and T_M is the memory cycle time. When $w \times T_D = \Delta t = T_M$, w requests are serviced in a memory cluster in a time period T_M. Completed services are routed in the same order through the backward network to the originating processors. Thus, ${}_jBP_i(t + T_M)$ is the partner of ${}_iFP_j(t)$ for all t. When one completed service is being routed through the backward path, its partner path is available for other requests, thus satisfying all three conditions of the function OVF.

Let T_P be the mean request time between requests of processors, the number of processors within one processor cluster is $N_P \equiv \frac{T_P}{T_D}$, where T_D is the network propagation delay. For simplicity, it is assumed that one memory cycle is equal to w network cycles, and each cluster is composed of w processors or memory modules. In a cluster, only one processor (or memory module) can access the network at a time. Thus, memory modules in a cluster are *physically* interleaved into w phases, ph_i, $1 \leq i \leq w$. However, processors in one cluster are *logically interleaved* due to their random request time.

To minimize the path establishing time, instead of using handshaking protocols between switches [107], a processor's phase number can be transmitted to acknowledge the completion of service. A message/request will be retransmitted if it is not acknowledged within Δt. A comparison between handshaking and non-handshaking methods is shown in Fig. 2.5. Since w requests may be routed through any I/O port of a switch during one memory cycle T_M, $w - 1$ switching states can be stored in $w - 1$ registers. Fig. 2.6 shows the different designs to support OCSMINs and conventional MINs.

Static random access memories (SRAMs) and dynamic random access memories (DRAMs) are the most popular main memory components for commercial machines. Contemporary commercial DRAM chips are standardized to use multiplexed two phase addressing: row (*ras*) and column selection (*cas*). On the other hand, single phase operation is still a standard



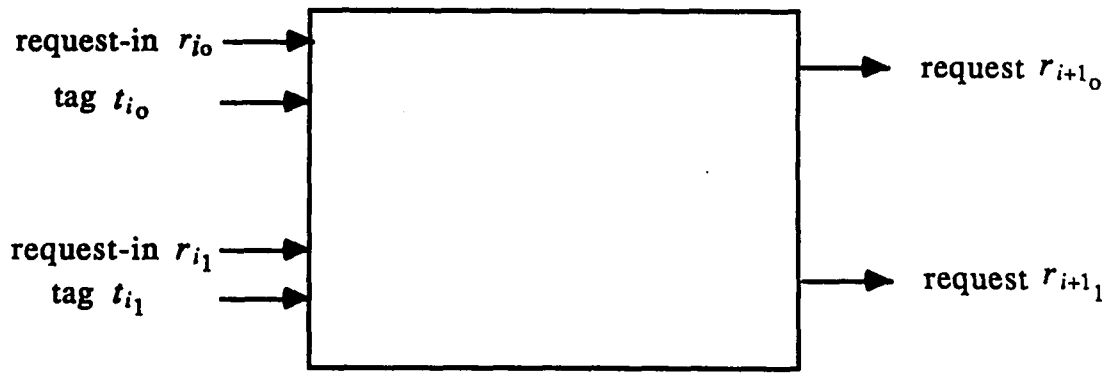
$$b_{i_0}=r_{i_0}(b_{i+1_0}\bar{t}_{i_0}+b_{i+1_1}t_{i_0})$$

$$b_{i_1}=r_{i_0}r_{i_1}(t_{i_1}t_{i_0}+\bar{t}_{i_1}\bar{t}_{i_0})+\bar{r}_{i_0}r_{i_1}(t_{i_1}b_{i+1_1}+\bar{t}_{i_1}b_{i+1_0})$$

$$r_{i+1_0}=r_{i_0}\bar{t}_{i_0}+r_{i_1}\bar{t}_{i_1}(t_{i_0}+\bar{r}_{i_0})$$

$$r_{i+1}=r_{i_0}t_{i_0}+r_{i_1}t_{i_1}(\bar{t}_{i_0}+\bar{r}_{i_0})$$

(a)



$$r_{i+1_0}=r_{i_0}\bar{t}_{i_0}+r_{i_1}\bar{t}_{i_1}(t_{i_0}+\bar{r}_{i_0})$$

$$r_{i+1}=r_{i_0}t_{i_0}+r_{i_1}t_{i_1}(\bar{t}_{i_0}+\bar{r}_{i_0})$$

(b)

Figure 2.5: Examples of different communication protocols. (a) The handshaking protocol. (b) The nonhandshaking protocol.

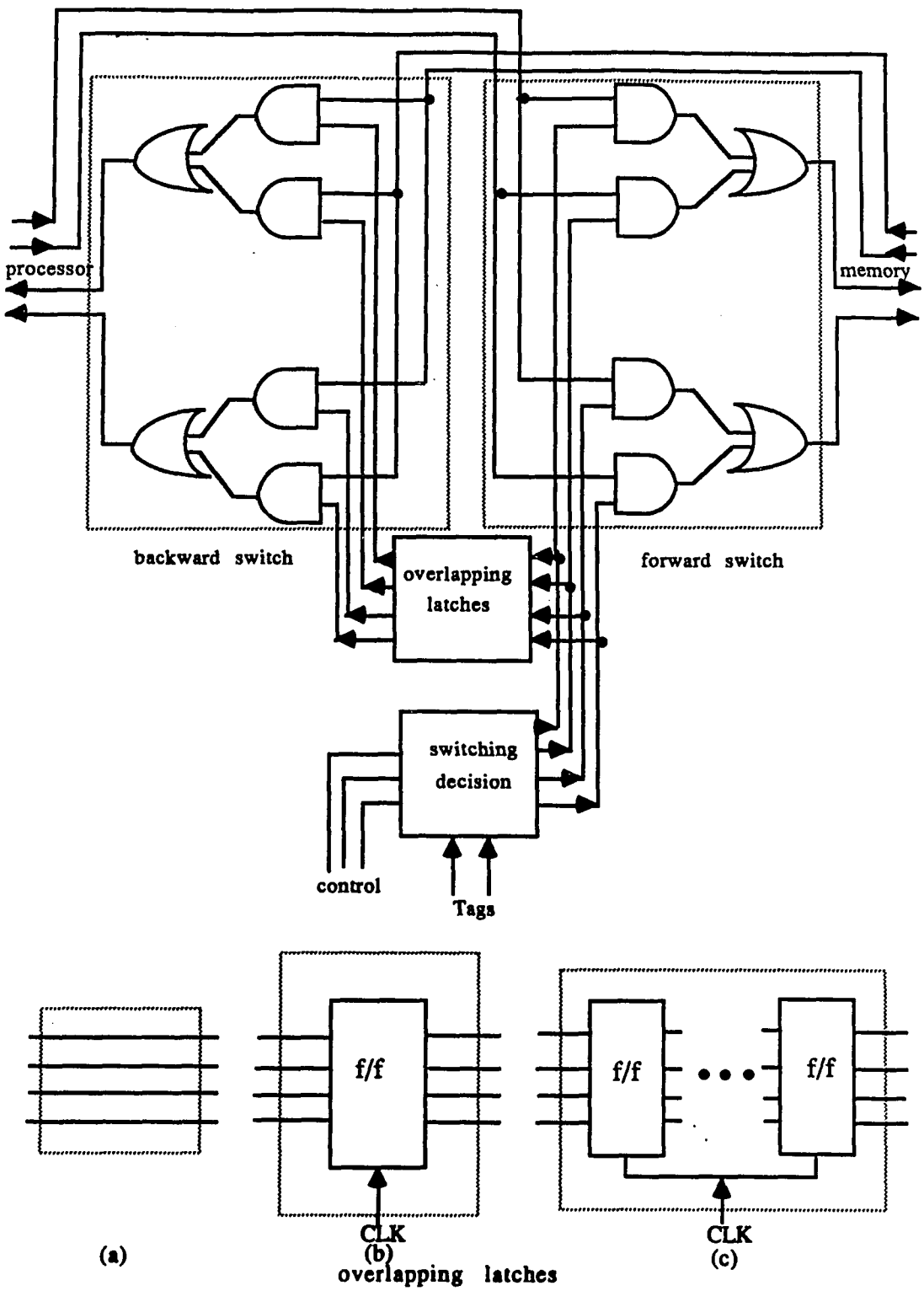


Figure 2.6: Required circuitry for supporting (a) a conventional MIN, (b) a two-way overlapped network, and (c) a w-way overlapped network.

for commercial SRAMs [83]. Both DRAMs and SRAMs can be easily used in the NOMI technique, except when the depth of overlapping is 2. Figs. 2.7 and 2.8 illustrate two-way NOMI techniques for DRAMs and SRAMs, respectively.

2.3 Cost Analysis

The primary advantage of the NOMI technique is that it has the potential to improve system performance at reduced cost. To have a fair comparison with conventional systems, the network cost is defined as follows.

Definition 4 : The *cost* of a conventional MIN (or an OCSMIN) is defined to be the number of data gates and links. The *cost factor* is the ratio of the number of data gates to the number of processors.

The cost and cost factor of a conventional MIN using $r \times r$ switches are $rN \log_r N$ and $r \log_r N$, respectively. In an OCSMIN with $r \times r$ switches, the total number of gates is $r\lceil \frac{N}{w} \rceil \lceil \log_r \frac{N}{w} \rceil$. The total number of gates is changed by $rN\lceil \log_r N \rceil - r\lceil \frac{N}{w} \rceil \lceil \log_r \frac{N}{w} \rceil$, and the cost factor of an OCSMIN is $\frac{r}{w}\lceil \log_r \frac{N}{w} \rceil$. Although the logic circuit and the number of links increases with the network size as $O(N \log N)$, the link length still grows in $O(N^2)$ complexity [32]. Since only the number of reduced links has been taken into account, we have a conservative cost analysis. As an example, 5120 2×2 switches (20480 gates) are required for a conventional 10-stage MIN, while only 1024 2×2 switches (4096 gates) are needed, if the network is overlapped in four ways. Modifications to existing designs to implement the NOMI technique include: (1) additional $w - 1$ registers at each switch, (2) a network clock w times faster than the *memory clock*, (3) a w-phase clock for each memory cluster, and (4) interface units in processors or memory modules.

Note that resource contention among processors increase when processors are clustered. The performance impact of resource contention caused by clustering can be neutralized by

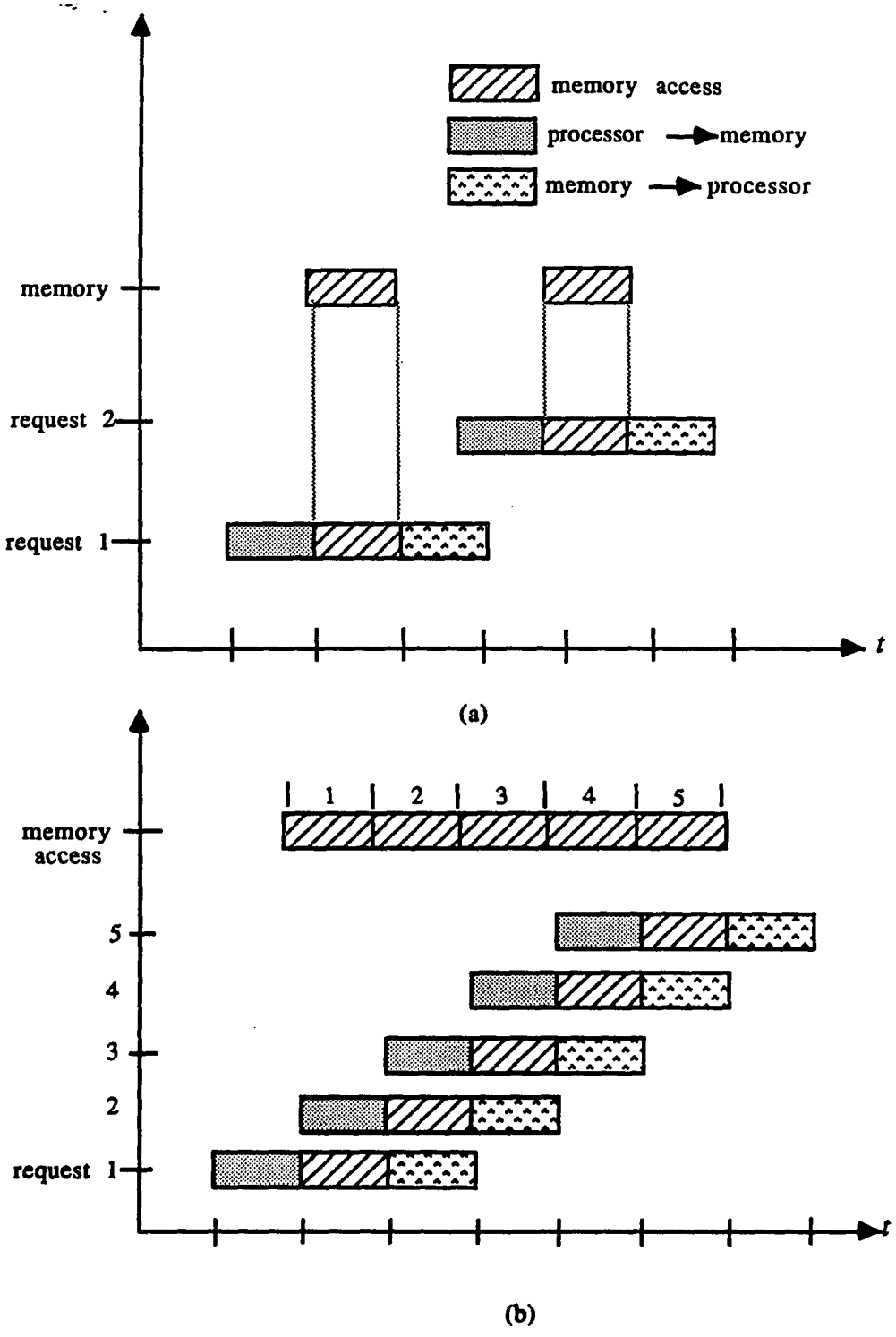


Figure 2.7: Comparison of SRAM operations in (a) a conventional MIN, and (b) in an OCSMIN.

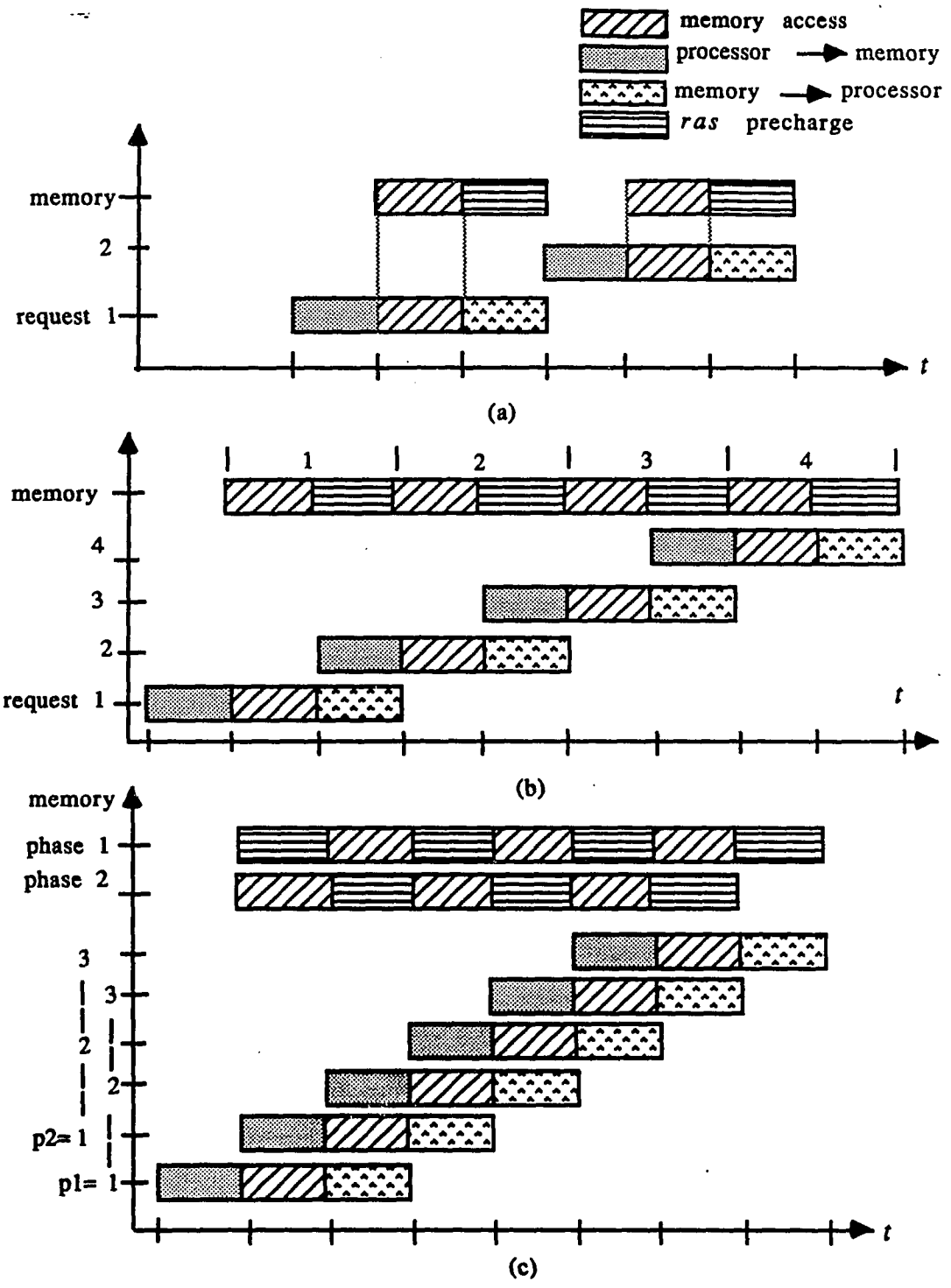


Figure 2.8: **Different modes of network operations with DRAM. (a) Conventional network. (b) Simple network overlapping with memory interleaving. (c) Overlapped network with memory interleaving.**

increasing switch size. The following theorem describes necessary conditions to reduce system cost while we increase switch size (in order to improve system performance) in the NOMI technique.

Theorem 1 : The cost of an OCSMIN can be monotonically decreased when

(1) a conventional MIN is replaced by an OCSMIN,

(2) switch size is increased, and

(3) the relative increase in the overlapping (or interleaving) depth is greater than that of the switch size.

Proof: The cost C_{orig} is $rN\lceil \log_r N \rceil$ for an $N \times N$ system connected by a network using $r \times r$ switches. The cost of a new network, C_{new}, with overlapping depth w and $r' \times r'$ switches, is $r'\frac{N}{w}\lceil \log_{r'} \frac{N}{w} \rceil$. Since $r' > r$, $\lceil \log_r N \rceil \geq \lceil \log_{r'} \frac{N}{w} \rceil$. We can now consider the factors rN in C_{orig} and $r'\frac{N}{w}$ in C_{new}. Applying the third condition to these factors, we get $\frac{r}{r'} < \frac{w}{w'}$ and thus $\frac{r'}{w'}N < \frac{r}{w}N$. It follows that $C_{orig} \geq C_{new}$. ∎

Corollary 1 : The cost of an OCSMIN satisfying Theorem 1 is always less than or equal to the cost of a bidirectional MIN.

Proof: In a bidirectional network, the forward and backward networks are essentially combined into one. The NOMI technique *cannot* be applied to bidirectional networks. Thus, the cost of a bidirectional MIN is $\frac{rN}{2}\lceil \log_r N \rceil$.

The lowest possible NOMI depth is 2 for an OCSMIN. The cost of a two-way overlapped OCSMIN is $\frac{rN}{2}\lceil \log_r N \rceil$. Since the network cost decreases monotonically when Theorem 1 holds, OCSMINs are always cheaper than bidirectional networks. ∎

2.4 Performance Analysis

The performance of conventional MINs is compared with the performance of OCSMINs

in this section. In addition to conventional performance parameters, i.e., blocking factors and network bandwidths, we introduce two additional parameters called the *mean system access time* and *execution/access parallelism* in this section.

2.4.1 Blocking Factors and Bandwidths

Numerous models have been proposed for evaluating blocking factors of the conventional MINs [77, 114, 85, 11, 65]. For its simplicity and accuracy, the model proposed in [77] with the following assumptions is used:

A1 Requests generated by processors are independent and uniformly distributed.

A2 Switches in the network have synchronous operations, and it takes one network cycle to establish a path. In each network cycle, a processor generates a new request with probability P.

A3 A blocked request is discarded, and new requests will be generated independently.

Although assumption A3 is apparently unrealistic, it greatly simplifies the analysis, and has been validated by simulation [77]. With the above assumptions, the blocking factor at stage i can be expressed as

$$P_{i+1} = 1 - (1 - \frac{P_i}{r})^r \tag{2.1}$$

where P_i is the probability that a request appears at an output port of an $r \times r$ switch located at stage i, and P_0 is the request rate of a processor cluster. Let $k \equiv \log_r N$ be the number of stages and P_{C_k}(or P_{O_k}) be the probability that a request can pass through the k stages of the conventional MIN (OCSMIN). The network pass-probabilities of conventional MINs and OCSMINs are $P_{A_C} = \frac{P_{C_k}}{P}$ and $P_{A_O} = \frac{P_{O_k}}{P}$, respectively.

In conventional MINs, the second sub-cycle (*ras precharge*) of a DRAM can be completed when the data is being transmitted back to the processor. Thus, memory bandwidths of

DRAMs and SRAMs are essentially the same, as shown in Fig. 2.9. The memory bandwidths are $\frac{1}{2T_D+T_{M_S}}$ and $\frac{1}{\max(T_{M_S},T_D)}$ for conventional MINs and OCSMINs, respectively, where T_D is the network delay, and T_{M_S} is the memory cycle time of an SRAM.

Little performance improvement can be achieved if only network overlapping (without memory interleaving) is implemented. For example, the throughput of an OCSMIN with no interleaved DRAM memory modules is one half lower than the throughput of a conventional network with SRAMs. The memory bandwidths of conventional MINs and OCSMINs are $\frac{1}{2T_D+T_{M_D}}$ and $\frac{1}{2\max(T_{M_D},T_D)}$, respectively.

Bandwidths of a k-stage conventional MIN and an l-stage OCSMIN, where $l = \lceil \log_r \frac{N}{w} \rceil$, are as follows.

$$BW_C = \frac{N \times P_{A_k}}{T_M+2T_D}$$
$$BW_O = \frac{1}{T_M} \times \frac{N \times P_{A_l}}{w} \times w = \frac{N}{T_M} \times P_{A_l},$$

where BW_C and BW_O are bandwidths of conventional MINs and OCSMINs, P_{A_l} is the pass probability of an l-stage network, and w is the depth of interleaving. The maximum utilization rates of conventional MINs and OCSMINs are $\frac{2T_D}{T_M+4T_D}$ and 100 percent, respectively.

2.4.2 System Access Time and Access Parallelism

We now discuss the impact of the NOMI technique on system waiting time, which is the mean time to establish a path and then access the memory module.

In the system access time analysis, a partially established path is assumed to be immediately released when a request is rejected [65]. After a path is established in a conventional MIN, there is a $\frac{T_M}{2}$ average delay before a memory module can actually be accessed. Similarly, in an OCSMIN, an average delay of $\frac{T_M}{2}$ is expected before a processor can access the network. Thus, the mean system access time can be expressed as

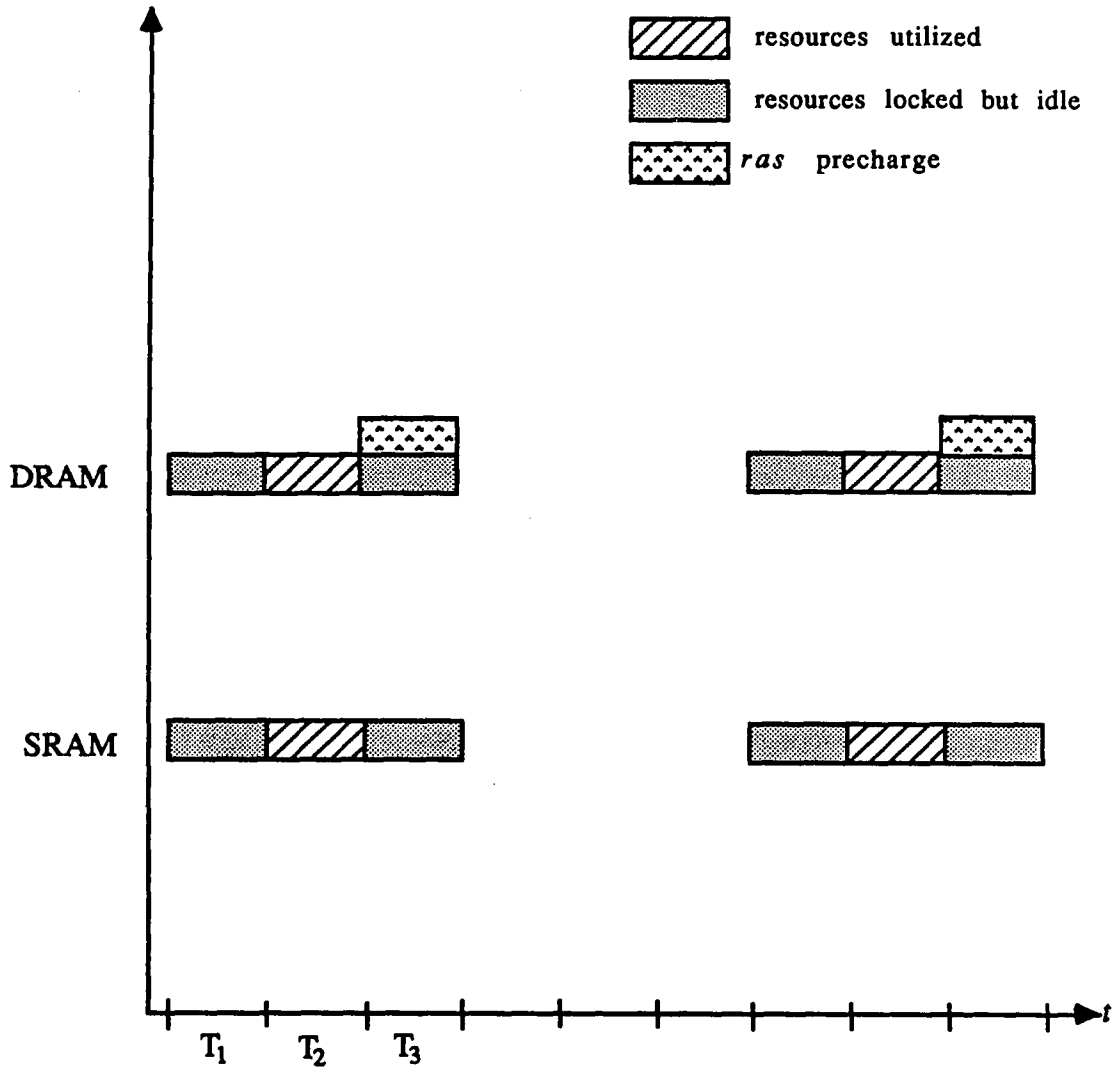


Figure 2.9: Comparison of DRAM and SRAM operations in a conventional MIN.

$$\begin{aligned}\overline{T}_O &= \int_0^1 [\ 2T_D + \frac{1}{2}T_M + P_A T_M + (1-P_A)P_A 2T_M + (1-P_A)^2 P_A 3T_M \cdots] dF(P) \\ &= \int_0^1 [2T_D + \frac{1}{2}T_M - T_M P_A \frac{\partial}{\partial P_A} \sum_{n=1}^{\infty}(1-P_A)^n] dF(P) \\ &= \int_0^1 [2T_D + (\frac{1}{P_A} + \frac{1}{2})T_M] dF(P) \end{aligned} \tag{2.2}$$

and

$$\begin{aligned}\overline{T}_C &= \int_0^1 [T_M + \frac{1}{2}T_M + P_A 2T_D + (1-P_A)P_A(2T_D + T_D) + \cdots] dF(P) \\ &= \int_0^1 [\frac{3}{2}T_M + \frac{P_A}{1-P_A}T_D\{\sum_{n=1}^{\infty} n(1-P_A)^{n-2}\}] dF(P) \\ &= \int_0^1 [\frac{3}{2}T_M + T_D\frac{1+P_A}{P_A}] dF(P) \end{aligned} \tag{2.3}$$

where $\overline{T}_O$ and $\overline{T}_C$ are the mean system access time in OCSMINs and conventional MINs, respectively, and $F(P)$ is the distribution function of the request rate.

P_A is a monotonically decreasing function of the number of stages in the network. T_D is determined by both the switch delay and the cable delay. The mean system access time difference between a conventional MIN and an OCSMIN is

$$\begin{aligned}\overline{T}_C - \overline{T}_O &= \tfrac{3}{2}T_{M_C} + T_{D_C}\tfrac{1+P_C}{P_C} - (2T_{D_O} + \tfrac{T_{M_O}}{P_O}) \\ &= T_{D_C}(1 + \tfrac{1}{P_C} - 2d) + T_{M_C}(\tfrac{3}{2} - \tfrac{m}{P_O})\end{aligned}$$

where $d \equiv \frac{T_{D_O}}{T_{D_C}}$ and $m \equiv \frac{T_{M_O}}{T_{M_C}}$. The mean system access time of an OCSMIN is smaller than that of the conventional MIN if $d < \frac{1+P_C}{2P_C}$ and $m < \frac{3P_O}{2}$. For example, in crossbar networks the network pass probability quickly converges to a constant (≈ 0.7) [77], and $P_C < 1$, $d < 1$. Thus, NOMI can always be applied to crossbar networks without increasing the mean system access time.

The last performance parameter to be considered is the degree of parallelism in system execution/access. In an OCSMIN based system, every memory module can be immediately

accessed after its current cycle, and processors can execute their own instructions independently. Thus, the access parallelism of the memory subsystem and the execution parallelism of the processor subsystem are the same as that of a conventional MIN based system. However, the NOMI technique does affect processors' access parallelism.

A processor cluster can be taken as a $w \times w$ switch, because each network access phase in a cluster can be treated as an output port of a switch. Thus, at each phase ph_i, the probability of a request appearing on the interface pn_i is $P_{ph_i} = 1 - (1 - \frac{P}{w})^w$. We can now get the access parallelism P_t, by plugging P_{ph_i} in the above equation into P_0 of Eq. 2.1, where t is the number of stages. The NOMI technique must be carefully adjusted to improve the access parallelism and avoid performance penalty. Access parallelism in three different networks connecting up to 1024 processors/memory modules are shown in Fig. 2.10. As shown in Fig. 2.10, the access parallelism of an OCSMIN with a four way NOMI using 4×4 switches is higher than the access parallelism of a conventional MIN with 2×2 switches, and is close to the access parallelism of a conventional MIN with 4×4 switches.

2.5 NOMI Optimization

The NOMI technique can achieve different design goals by adjusting the depth of overlapping/interleaving and switch size. Since MINs have integral number of stages, and most other design parameters have continuous values, all the related optimization problems are mixed integer nonlinear programming problems. The optimization problem is solved by a branch and bound technique, which is illustrated by two examples.

2.5.1 Combinatorial Capability

When N processors communicate with one another, the number of possible permutations is $N!$. For a large N, $N!$ can be approximated by the Stirling's equation, $N! \approx N^N e^{-N} \sqrt{2\pi N}$.

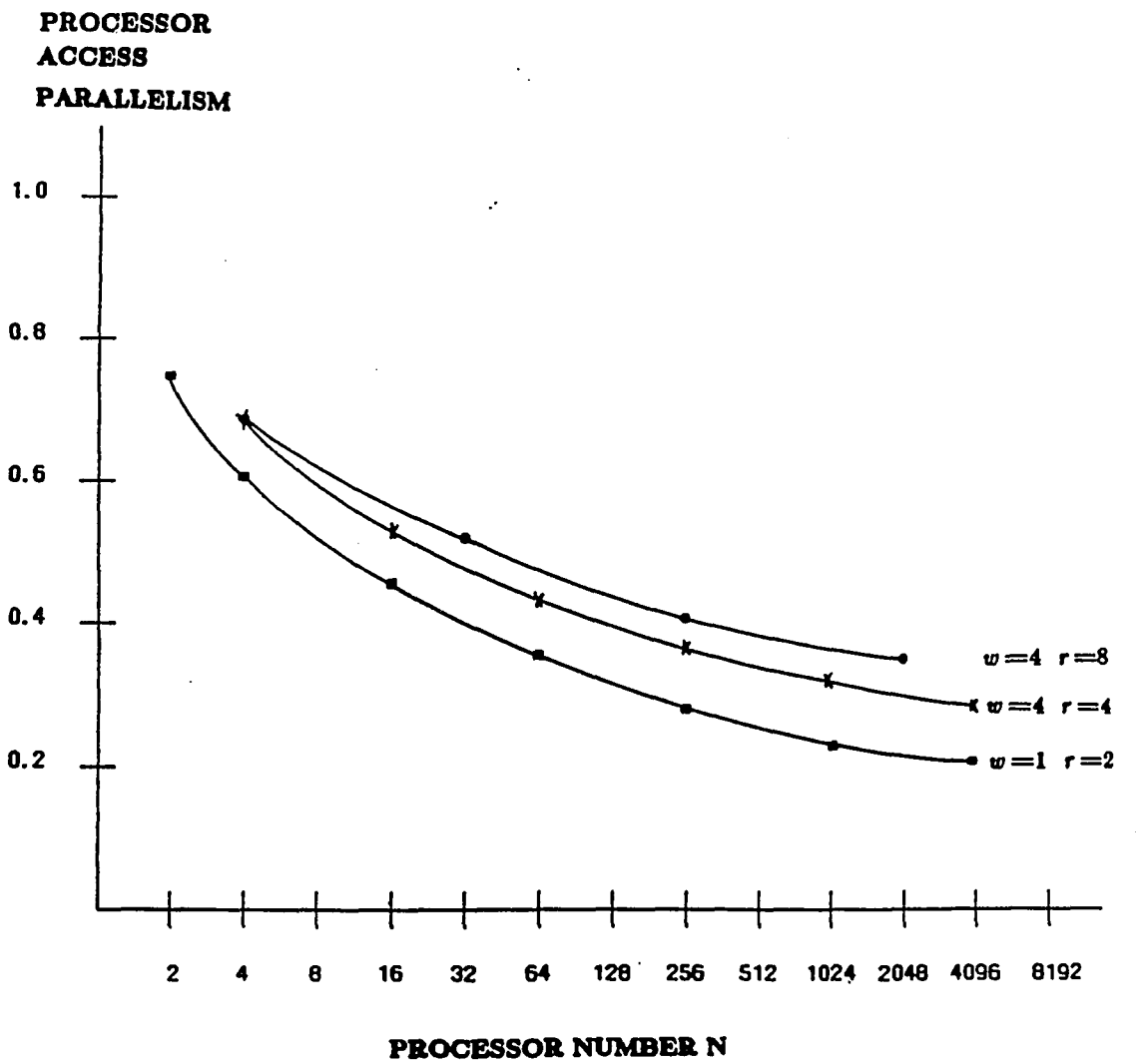


Figure 2.10: Comparison of access parallelisms between conventional MIN and OCSMIN when p=1.

In a conventional MIN, the total number of conflict-free permutations $|P_N^r|$ is $(r!)^{\frac{N}{r}\log_r N}$. The ratio of conflict-free permutations to the total number of permutations ($N!$) decreases to zero even for small N. For example, $\frac{|P_{16}^2|}{16!} = 0.0002$, and $\frac{|P_{32}^2|}{32!} \approx 4.6 \times 10^{-12}$.

We now derive the number of conflict-free permutations, $|L_N^{w,r}|$, in a network implementing w-way NOMI technique with $r \times r$ switches. When the depth of NOMI is w, one system cycle is composed of w phases. In the first phase, there are $w^{\frac{N}{w}}$ combinations of processors that may present service requests. In the second phase, there are $w-1$ selections for each processor cluster, and thus there are $(w-1)^{\frac{N}{w}}$ selections for the system. By induction, there are $(w!)^{\frac{N}{w}}$ combinations of processors to be interconnected in one system cycle. At each phase, the network can perform $(r!)^{\frac{N}{wr}\log_r \frac{N}{w}}$ different permutations. Thus, $|L_N^{w,r}| = (w!)^{\frac{N}{w}}$ $(r!)^{\frac{N}{wr}\log_r \frac{N}{w}}$.

The combinatorial capabilities of various networks change too radically to be included in the objective function of the optimization problem. For example, $1024! \approx 10^{2640}$, $|P_{1024}^2| = 10^{1541}$, $|L_{1024}^{2,2}| \approx 10^{847}$, $|L_{1024}^{4,4}| \approx 10^{707}$, $|L_{1024}^{8,8}| \approx 10^{761}$. Thus, the combinatorial capability is used as a constraint in the optimization problem to prevent any excessive degradation of the network's combinatorial capability.

2.5.2 Optimal Design

Many switches can be used to implement the NOMI technique of a specific overlapping/interleaving depth. A *configuration wr* is defined as the combination of the overlapping depth w and the switch size r. All feasible configurations form a *configuration space* Ω. The memory and processor subsystems are assumed to have fixed operational parameters, i.e., the memory access time, the instruction execution speed, etc.

Recall that the cost of a network is $C_{r,w} = r\lceil\frac{N}{w}\rceil\lceil\log_r\lceil\frac{N}{w}\rceil\rceil$. The network cost monotonically decreases with the overlapping depth. When $N' = \lceil\frac{N}{w}\rceil$ is fixed, and for $r_i \in \Omega$, $s \in I^+$,

$\log_{r_i} N' \in I^+$, and $0 < \log_{r_i} N' - \log_{r_i+s} N' < 1$, we have $\lceil \log_{r_i} N' \rceil = \lceil \log_{r_i+s} N' \rceil$. That is, $C_{r,w}$ is a sawtooth function, and r_i is a local minimal point when $\lceil \log_{r_i} N' \rceil = \log_{r_i} N'$. Given a fixed w, to find the global minimal cost we only need to examine local minimal points $r_1, r_2, \cdots, r_{max}$, where r_{max} is determined from the upper bound of cost. r_i's can be derived recursively by $r_{i+1} = e^{\frac{k_i}{k_i - 1} \log r_i}$. When the number of stages is decreased by one, the network cost is changed by $\Delta C_{w,r_i} = N' ((r_i - r_{i+1})k_i + r_{i+1})$, where $r_i - r_{i+1}$ is the increased switch size before decreasing the number of stages by one, and k_i is the number of stages. Since $\Delta C_{w,r_i}$ is neither positive nor negative, r_1 through r_{max} must be exhaustively searched for the global minimal cost.

To demonstrate the optimization technique, mean system access time in Eq. 2.3 is used as the first example objective function. The mean system access time is composed of two parts: the fixed network propagation $2T_D$, and the random network delay $\frac{1}{2}T_M + \int_0^1 \frac{T_M}{P_{A_{wr}}} dF(P)$, where $P_{A_{wr}} = P_k/P$ is the *pass rate* of the network, P_k is the probability that a request can be transmitted through a k-stage network, and P is the probability that a request occurs at a certain phase.

The first constraint in the above optimization problem is the network propagation delay, which is composed of *active* components' (switches') delay Y, and *passive* components' (cables') delay [14, 81]. The cable delay is determined by the physical size of the network. Most network topologies will shuffle cables, i.e., the passive network delay is proportional to the height of the network $\lceil \frac{N}{w} \rceil$. Thus, cable delay can be expressed as $\delta \lceil \frac{N}{w} \rceil$, where δ is the unit length delay of the cable used. Reliability sets the other constraint in the optimization problem: when one cluster fails, the loss of computation power should be bounded by a constant b. The formal problem statement is then given as follows.

Problem OD: $\min_{wr \in \Omega} 2T_{D_{wr}} + \frac{1}{2}T_M + \int_0^1 \frac{T_M}{P_{A_{wr}}} dF(P)$

subject to constraints :

(1) $T_{D_{wr}} = k_{wr}(Y + \delta\frac{N}{w})$

(2) $P_{i+1} = 1 - (1 - \frac{P_i}{r})^r$

(3) $P_{A_{wr}} = P_{k_{wr}}/P$

(4) $T_M \geq wT_{D_{wr}}$

(5) $C_{spec} \geq C_{wr} \geq r\lceil \frac{N}{w} \rceil \log_r \lceil \frac{N}{w} \rceil$

(6) $b \geq \frac{w}{N}$

(7) $|L_N^{w,r}| \geq |L_{spec}|$.

This problem is essentially a mixed integer-nonlinear programming problem. Given Y, δ, T_M, and b, a branch-and-bound technique is presented here to find the optimal combination of w and r.

S0. $r_{inc} = 0$, $w_{inc} = 0$, $\bar{T}_{inc} = \infty$.

S1. Compute an upper bound of w by $w_{max} = \lfloor Nb \rfloor$.

S2. $w = 2$.

S3. Obtain an upper bound of the stage number by solving $k_{max}[Y + \lceil \frac{N}{w} \rceil \delta] \leq T_{D_{max}}$. $T_{D_{max}} = \frac{T_M}{w}$.

S4. $r_{min} = \inf_r \{\lceil \log_r \lceil \frac{N}{w} \rceil \rceil \leq k_{max}\}$, and $r_{max} = \sup_r \{ r\lceil \frac{N}{w} \rceil \lceil \log_r \lceil \frac{N}{w} \rceil \rceil \leq C_{spec}\}$.

S5. **If** $r_{max} < r_{min}$ **then** w is infeasible and go to S8 **else** $r = r_{min}$.

S6. $\bar{T}_{wr} = 2T_{D_{wr}} + \frac{1}{2}T_M + \int_0^1 \frac{T_M}{T_{A_{wr}}} dF(P)$. **If** $\bar{T}_{wr} \leq \bar{T}_{inc}$ **then** $\bar{T}_{inc} = \bar{T}_{wr}$, $w_{inc} = w$, $r_{inc} = r$.

S7. $r := r + 1$. **If** $r \leq r_{max}$ **then** go to S6.

S8. $w := w + 1$. **If** $w \leq w_{max}$ **then** go to S3.

S9. **If** $w_{inc}r_{inc} = 0$ **then** no feasible solution **else** $\bar{T}_{opt} = \bar{T}_{inc}$, $w_{opt} = w_{inc}$, $r_{opt} = r_{inc}$.

For example, when $N = 1024, \delta = 0.01, Y = 8, T_M = 300$, it is found that the optimal network cost occurs when $w = 10$ and $r = 5$, and the corresponding mean system access time is 373.1.

Network cost can be used as another objective function of the optimization problem. With the same constraints as the above example, the optimal network cost is only 7.5 percent of a conventional network based on 2×2 switches. The optimal mean system access time is 363.2 ns when $w = 3, r = 3$. The mean system access time of the network with the optimal network cost is three percent higher than the optimal mean system access time. On the other hand, the cost of the network with the optimal mean system access time is three times higher than the network with the optimal network cost. With the above information, the designer can select a network design with either optimal network performance or optimal network cost.

2.6 Conclusion

In this chapter we have proposed a cost-effective high performance architecture for circuit switching MINs. This technique is useful in matching bandwidths of subsystems in a large multiprocessor system. The drastic reduction in the number of switches/gates makes the use of large switches more attractive. The NOMI technique also shortens the network delay because of the smaller number of stages required. The cost analysis provides a lower bound for the improvement made by NOMI since it did not consider the length of data links.

When the network size is reduced, the impact of a network component's failure will be more serious than in a conventional network. To improve the network reliability, it is important

to detect and remove faults in the network as soon as possible. To achieve this goal, two new architectures are developed in the next Chapter.

Appendix 2.A: List of Symbols

pc_i	Processor cluster i.
mc_i	Memory cluster i.
p_{ij}	Processor located in processor cluster i with phase number j.
m_{ij}	Memory module located in memory cluster i with phase number j.
pn_i	Network interface unit of processor cluster i.
mn_i	Network interface unit of memory cluster i.
r	The size, or the number of input/output ports, of switches.
w	Depth of overlapping, which is always associated with the memory interleaving of the same depth.
OCSMIN	Overlapped Circuit Switching Multistage Interconnection Network.
NOMI	Network Overlapping and Memory Interleaving.
$FS_{ij}(k,m)$	A forward switch located at (i,j) and its input port k, and output port m are used to form an interconnection.
$BS_{ij}(k,m)$	A backward switch located at (i,j) and its output port k, and input port m are used to form an interconnection.
FL_{ij}	A forward link located at (i,j).
BL_{ij}	A backward link located at (i,j).
$A(t)$	The request vector appears on the processor subsystem at time t.
$\Pi(E)$	The partner of resource E on the forward (or backward) network.

${}_iFP_j$	The set of resource on the forward network to support a path from processor cluster i to memory cluster j.
${}_iBP_j$	The set of resource on the backward network to support a path from processor cluster i to memory cluster j.
Δt	The time interval before the partner of a forward switch (link) is locked after the forward switch (link) is locked.
$CSF(t)$	Interconnection function of a conventional circuit switching network.
$OVF(t)$	Interconnection function of an overlapped circuit switching network.
T_M	Memory subsystem (memory modules and interface) cycle time.
T_D	Network propagation delay.
T_P	Processor's memory-request cycle time.
N	Number of processor (or memory) modules.
$N' = \frac{N}{w}$	Number of processor (or memory) clusters.
P	Processor request rate, $0 \leq P \leq 1$ at each network cycle.
P_{A_r}	Network pass probability for a network with switch size r.

CHAPTER 3

POLYNOMIAL TESTING OF PACKET SWITCHING NETWORKS

3.1 Introduction

Large interconnection networks are made feasible by the advances in VLSI technology. Since VLSI technology greatly degrades testability, an interconnection network must have a structure that is easily testable.

A *path* and a *route* for a source-destination pair are defined as follows. A path is a physically-established communication medium between the source and destination to transfer a request/data. A route is a logical path which can transfer a request from a source to its destination without total dedication to it; resources on a route are time-shared among several packets. In a circuit switching MIN, the path from a source to its destination is physically set up *a priori* and dedicated to a request until the request is completely serviced. By contrast, no complete physical path is established *a priori* for a request in a packet switching network (PSN). A packet-switching multistage interconnection network (PSMIN) is composed of a large number of links and switches with buffers. Each PSMIN switch is essentially an $r \times r$ crossbar, in which a queue is placed at each input port to store packets. A request/message is decomposed into several packets, each of which is independently transferred through an available route.

Many PSNs have an undesirable effect called the *routing dynamic*: the order of arrival of packets at the destination may be different from the order of their transmission from the source. Although PSMINs can be designed not to have the routing dynamic, the routing dynamic will be considered in our testing method to provide better versatility. Clearly, a PSMIN with the routing dynamic is an asynchronous sequential machine. Although a sequential machine can be fully tested with a checking sequence derived from its state-transition table [15], no feasible checking sequence seems to be derivable for large scale asynchronous sequential machines like PSMINs. Functional testing is an alternative to prove the correctness of some of the machine's functions within a finite time period.

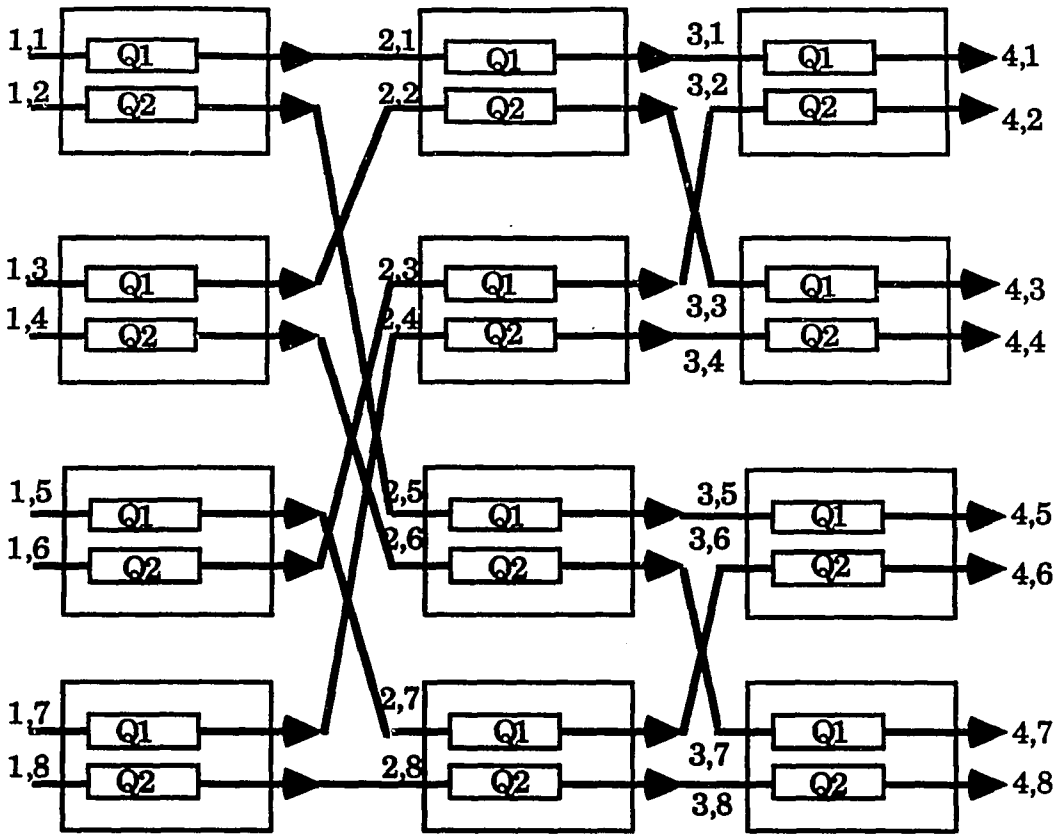
Several researchers have proposed functional testing procedures for specific networks. A comprehensive method for diagnosing base-line circuit switching MINs with 2×2 switches was introduced by Feng and Wu [38], and a simplified version of their fault model and the corresponding testing strategy can be found in [3]. Lee and Shen modeled a circuit switching MIN using 2×2 switches as an ILA [64]. Agrawal and Leu used the dynamic full accessibility of MINs to test their connectivity [5]. Several high level testing strategies for a general PSMIN have also been studied [21, 71, 75, 68, 20, 70, 104, 31], most of which are adaptive procedures requiring human assistance.

Most existing methods are centralized and off-line, i.e., the whole network is tested off-line by *one* tester. Since there are $\frac{N}{r} log_r N$ switches in a PSMIN, the complexity of the network testing problem is $O(\frac{N}{r} log_r N)$. Centralized testing methods are usually very inefficient for large networks, because the problem needs to be handled by the tester grows exponentially with the size of the network. To improve testing efficiency, we propose two testing strategies: *high-level* and *low-level testing*. In the high-level, every processor can serve as a tester to test part of the network; thus, there are N testers for the network. Assuming that testers are homogeneous, the complexity of the testing problem in each tester is reduced to $O(\frac{1}{r} log_r N)$. In the low-level

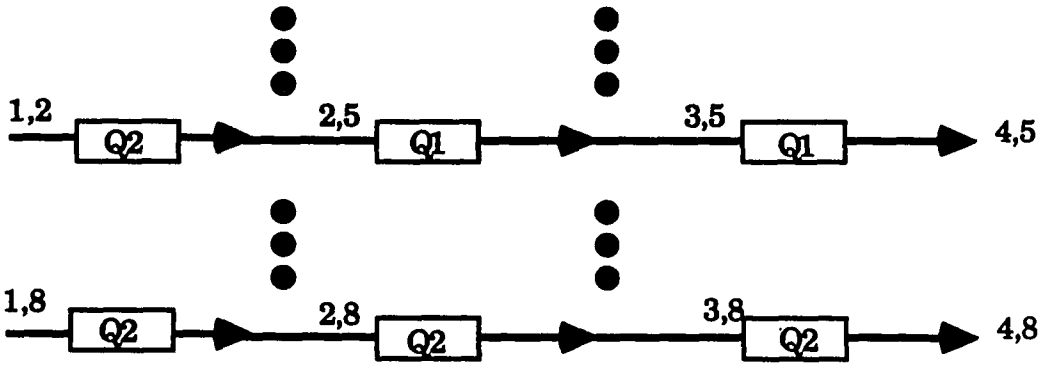
testing, switches are used as testers and designed to have autonomous testing capability [73]. In other words, the complexity of the testing problem in each tester is determined by the switch size, and is independent of the network size. The high-level testing strategy can test the network concurrently, but may have a lower fault coverage than the low-level testing. On the other hand, the low-level testing strategy is an off-line testing method with a small testing time and high fault coverage.

The high-level testing is based on the topology and functions of the network. To eliminate the routing dynamic, network operations are first synchronized. Then, an $N \times N$ blocking network is decomposed into N^2 routes, $NT = \{RT_{ij} \mid 1 \leq i,j \leq N\}$, where RT_{ij} is the route from source i to destination j. RUT_{ij} is the route RT_{ij} under test, and the testing processors are the processors connected to the route under test (RUT). In the high-level testing, faults in RUT_{ij} are tested without stopping the normal operations on $NT - \{RUT_{ij}\}$, where $\{RUT_{ij}\} \in NT$ and $\{RUT_{ij}\} \neq NT$. To test a route without interrupting, or being interrupted by, normal operations, the testing processors should be able to lock/unlock the RUT. Locking a route prevents unexpected packets from entering the route. As shown in Fig. 3.1, a route can be viewed as a cascaded shift register array. The register array can then be easily modified into *divisors*, *multipliers* or other similar structures for *polynomial testing*.

Each switch functions as a tester in the low-level testing. To obtain high fault coverage with a small testing time, each switch is designed to have self-testing capabilities. A switch is composed of buffers, a routing control unit (RCU) and output ports consisting of multiplexers-demultiplexers (MU/DEXes). Since thorough testing of the RCU may require an intractable testing length, an on-line checker is proposed to detect malfunctions in the RCU. For the rest of the network, queues are first self-tested by polynomial generation and comparison. If the queues are fault-free, they are then used to generate test patterns for links and MU/DEXes. The testing responses of switches at one stage are verified at the next stage.



(a) A Baseline PMIN with switch permuation E_0



(b) The corresponding cascaded shift register array of the PMIN.

Figure 3.1: A baseline PSMIN with switch permutation E_0 and the corresponding cascaded shift register arrays.

The rest of this chapter consists of four sections. The polynomial operations necessary for our testing method are first reviewed. Then, network fault models are introduced, and then testable designs and the corresponding testing methods for the high and low-level testing are presented. Finally, concluding remarks are given.

3.2 Polynomial Testing Principles

Basic polynomial operations and their implementations are briefly discussed below. Use of the polynomial ring, $GF(2)[x]$, is well-known for error-control codes [47]. Only those properties useful for testing PSNs will be introduced below for completeness.

Definition 1 : A polynomial $P_b(x) = \sum_{i=0}^{n} b_i x^i$ in $GF(2)[x]$ is said to be a *bit polynomial* if each of its coefficients is a bit, i.e., $b_i \in \{0, 1\}$, $0 \leq i \leq n$. A *word polynomial* is the one whose coefficients are words instead of bits, i.e., $P_w(x) = \sum_{i=0}^{n} w_i x^i$, where for every $i \in I_n = \{0, 1, \cdots, n\}$, $w_i =$ *ONE or ZERO* , and *ONE* is a b-bit vector of *arbitrary* pattern and *ZERO* $=$ $\overline{ONE}$, i.e., *ZERO* is bit-wise complemented to *ONE* . Thus, any two words with maximum Hamming distance can be used as *ONE* and *ZERO* , respectively.

For notational convenience, let $W_n(x)$ denote a polynomial $\sum_{i=0}^{n} c_i x^i$, $c_i = 1$ or *ONE* , $\forall\ i \in I_n$. $\overline{P}(x) = \sum_{i=0}^{n} \bar{c}_i x^i = P(x) \oplus W_n(x)$ is the complement of $P(x)$, and the symbol "$\oplus$" represents the addition in $GF(2)$. Unless otherwise specified, we will use the term "polynomial" to represent both bit and word polynomials. The mechanisms to manipulate polynomials are called their *calculators*. The contents of a calculator before operating on its input are called the *initial state*, which will always be assumed, for clarity of presentation, to be all zeros. A calculator with the zero initial state is called an *inert linear machine* [56]. When a word polynomial operation is applied to a faulty circuit, the closure property of $GF(2)[x]$ may not hold. However, when a word polynomial is applied to a non-faulty

circuit, the resulting polynomial belongs to $GF(2)[x]$. Calculators are more hardware-efficient if *ONE* and *ZERO* are composed of all 1's and 0's, respectively, because for each operation every bit will require an identical circuit.

3.2.1 Operations on Polynomials

A *periodic* polynomial with period p is the series $\sum_{i=1}^{\infty} c_i x^i$ where $c_i = c_{i+p}$, $\forall\, i \in \mathbf{I}$, and $\mathbf{I}$ is the set of integers. It can be generated by a linear (or nonlinear) feedback shift register (LFSR) called a *polynomial generator* (PG). Registers in a PG can be implemented by different types of flip-flops, and apparently different test patterns are needed for different implementations. However, as shown in Appendix 1, at most two inputs are needed to detect faults in a master-slave SR flip-flop. Since the high-level testing deals with the network topology, we will consider only the input and output stuck-at faults of registers, i.e., not the stuck-at faults inside registers. However, the same test patterns can test all the faults in those registers implemented with the master-slave flip-flops shown in Appendix 1.

Two polynomials, $P_1(x) = \sum_{i=0}^{n} c_{1,i}x^i$ and $P_2(x) = \sum_{i=0}^{n} c_{2,i}x^i$, are *equal* iff $c_{1,i} = c_{2,i}$, $\forall\, i \in \mathbf{I}_n$. Two polynomials can be compared for equality by XOR gates. The following operations are useful for our discussion.

Addition and Boolean:

Let $\{P_j(x) = \sum_{i=0}^{n} c_{j,i}x^i\}$ be k polynomials in $GF(2)[x]$, $P_3(x)$ is the addition of $P_1(x)$ and $P_2(x)$, denoted by $P_3(x) = \sum_{i=0}^{n} c_{3,i}x^i = P_1(x) \oplus P_2(x)$, if for each $i \in \mathbf{I}_n$, $c_{3,i} = c_{1,i} \oplus c_{2,i}$. Addition can be implemented with XOR gates. If a Boolean operation Δ is applied to $P_1(x)$, $P_2(x)$,$\cdots P_k(x)$, the resulting polynomial $P(x) = \sum_{i=0}^{n} c_i x^i$ is calculated by $c_i = c_i^1 \,\Delta\, c_i^2 \,\Delta\, \cdots \,\Delta\, c_i^k$, $\forall\, i \in \mathbf{I}_n$, and Δ is a bit-wise operation when c_i is a word. Only AND and OR, two most important operations, will be considered in this chapter.

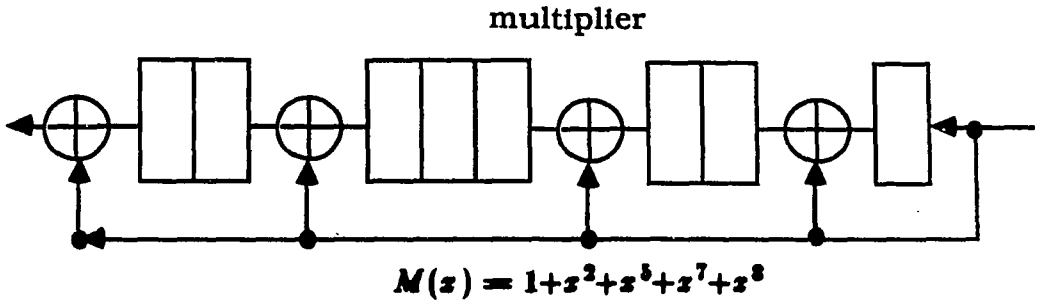
Division and Multiplication

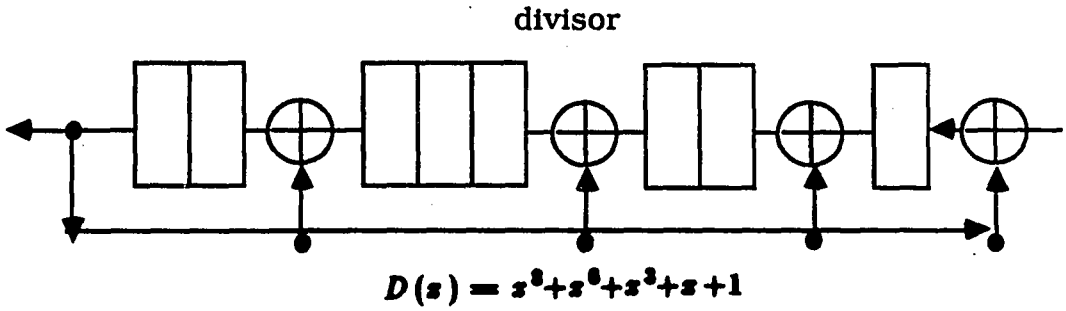
If $P(x) = \sum_{i=0}^{n} p_i x^i$ and $M(x) = \sum_{i=0}^{n} m_i x^i$ in $GF(2)[x]$, the multiplication of $M(x)$ (multiplier) to $P(x)$ (multiplicand) is $P_3(x) = P(x)M(x) = \sum_{i=0}^{n} p_{3,i} x^i$, where $\forall\ i \in \mathbf{I}_n$, $p_{3,i} = p_i m_0 \oplus p_{i-1} m_1 \oplus \cdots \oplus p_1 m_{i-1} \oplus p_0 m_i$. On the other hand, given two polynomials $P(x)$ (dividend) and $D(x) = \sum_{i=0}^{r} d_i x^i \neq 0$ (divisor) in $GF(2)[x]$ there exist two polynomials $Q(x)$ and $R(x)$ in $GF(2)[x]$ such that $P(x) = D(x)Q(x) + R(x)$ where $R(x) = 0$ or $deg\ R(x) < deg\ D(x)$. In this process, $P(x)$ is said to be *divided* by $D(x)$, yielding a quotient $Q(x)$ and a remainder $R(x)$.

A *bit divisor* (*multiplier*) divides (multiplies) an input stream by a fixed bit polynomial. Similarly, a *word divisor* (*multiplier*) performs divisions (multiplications) between two word polynomials. In a word polynomial divisor/multiplier (PDM), operands are *ONE* or *ZERO* instead of 1 or 0. It has logic operations similar to those of a bit PDM, but special mechanisms are necessary to preserve the properties of the polynomial ring.

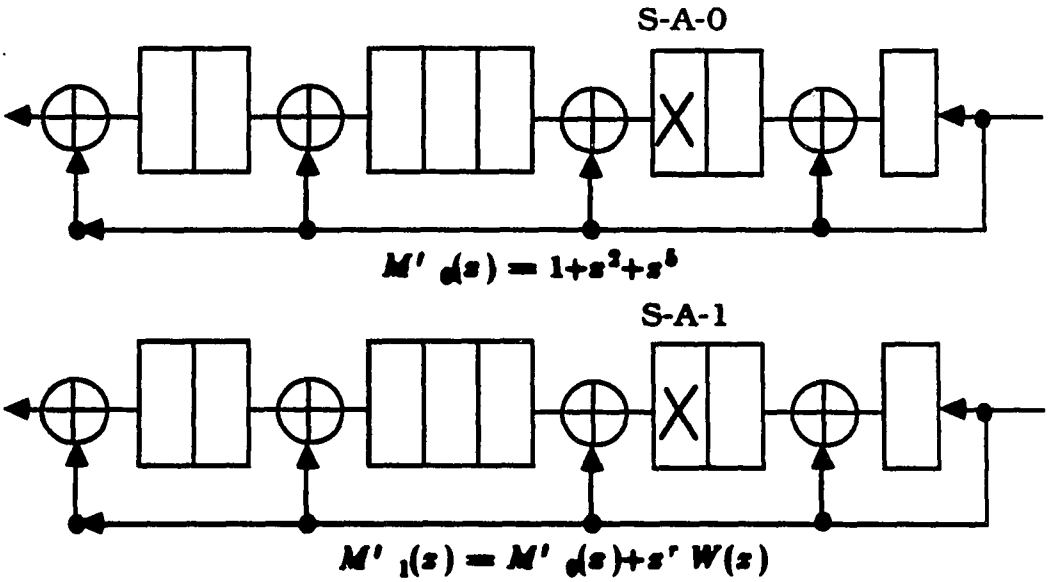
The final contents of a PDM will henceforth be represented by $R(x)$, the input stream by $P(x)$ and the output stream by $Q(x)$. The general structures of a bit divisor and a bit multiplier are shown in Fig. 3.2(a). $M(x)$ and $D(x)$ in Fig. 3.2(a) are $1 + x^2 + x^5 + x^7 + x^8$ and $x^8 + x^6 + x^3 + x + 1$, respectively. The lowest order position is located in the input (output) port of the divisor (multiplier). There is an XOR gate, denoted by $\oplus$, at the D-type flip-flop's (DFF's) output of stage i only when m_i or d_i is 1. A *block* B_i is the collection of DFFs between the $(i-1)$-th and i-th XOR gates, counting from the lowest order position, in a PDM. Thus, a PDM is composed of a set of blocks, $\{B_i\}$. Let the order (the number of stages) of B_i be r_i. In the multiplier of Fig. 3.2(a), $r_1 = 2$, $r_2 = 3$, $r_3 = 2$, and $r_4 = 1$.

Since a RUT is to be transformed into a polynomial calculator for testing, the effects of DFFs' Multiple Stuck-At (MSA) faults on a PDM are discussed as follows. A MSA fault, f_M, in a block B_i is composed of multiple Single Stuck-At (SSA) faults, i.e., $f_M = \{f_s^i\}$,





(a) A normal multiplier and divisor



(b) Two faulty multipliers

Figure 3.2: The structure of faulty and non-faulty multipliers and divisors

where f_s^i is a SSA fault in B_i. Let k be the faulty position nearest to the output port of B_i. Then, $f_s^k \in f_M$ will block the effects of all the other SSA faults in f_M. Such a f_s^k is called the *leading* SSA fault in B_i. There are $2r_i$ possible leading faults in B_i, and, thus, there are $2r_i$ distinguishable stuck-at faults in the block, where r_i is the number of stages in B_i.

A s-a-0 FL changes the attached XOR gates into null operators. Thus, for a multiplier, $M'(x) = \sum_{i=1}^{k} m_i' x^i$, where $m_i' = 0$ if the FL attached to the XOR gate at x^i is stuck at 0, and $m_i' = m_i$ otherwise.

Lemma 1 [56] : The impulse response of a multiplier is $m_0 m_1 ... m_n 0...0$, where the impulse polynomial is $P_I(x) = 10...0$.

Clearly, an unknown multiplier can be uniquely identified by its impulse response, and the multiplication of $P_I(x)$ to $M(x)$ can be viewed as a discrete convolution between them.

Lemma 2 : When a PDM is an inert machine and a s-a-0 fault occurs in B_k, the multiplier $M(x) = \sum_{i=0}^{n} m_i x^i$ is changed to a new multiplier, $M'(x) = \sum_{i=0}^{k-1} m_i x_i$.

Lemma 3 : Let l be the number of fault-free DFFs between B_i's output and the leading faulty DFF. Then, when the leading faulty DFF^l in B_k is s-a-1, $Q(x) = M'(x)P(x) \oplus x^{r'} W_n(x)$, where $r' = l + \sum_{j<k} r_j$ and $M'(x)$ is the new multiplier whose highest order position is located at the leading faulty DFF in B_k.

Proof: Let the input (or output) of B_i be I_i (or O_i). Then we have $I_{i-1} = O_i \oplus P(x)$ and $O_{i-1} = x^{r_{i-1}} I_{i-1}$. When DFF^l is s-a-1, $O_k = W_n(x)$. Thus, $I_{k-1} = P(x) \oplus x^l W_n(x)$ and $O_{k-1} = x^{r_{k-1}}(P(x) + x^l W_n(x))$. By induction, we can show that $Q(x) = M'(x) P(x) \oplus x^{r'} W_n(x)$. ∎

When x^k of a divisor is s-a-1, the output $Q(x) = P'(x)/D'(x)$, where $P'(x) = \{x^l (\sum_{i=k+1}^{r} n_i x^{i-k}) \oplus W_l(x)\}$, $\sum_{i=k+1}^{r} n_i x^{i-k}$ is the initial state of the divisor and l is the poly-

nomial length that is sufficient for testing, and $D'(x)$ the new divisor with its lowest order position at the output of B_k. Similarly, a s-a-0 fault at x^k makes $Q(x)$ periodic, i.e., $Q(x) = x^l \left(\sum_{i=k+1}^{r} n_i x^{i-k} \right) / D'(x)$, where the degree of the faulty divisor is $r_d = \sum_{j>k} r_j$. Note that the output $Q(x)$ is independent of the input stream. The structures of s-a-0 and s-a-1 multipliers are shown in Fig. 3.2(b). The resulting $M'_0(x)$ (for s-a-0) and M'_1 (for s-a-1) are $1 + x^2 + x^5$ and $M'_0 \oplus x^r W(x)$, respectively.

For testing purposes, it is assumed that every DFF on a route can be simultaneously set to *ZERO* by an external signal. Signature analysis examines $R(x)$ after the testing polynomial $P(x)$ is applied to a circuit under test. The final contents of each DFF must be directly read out for signature analysis. Unfortunately, this will greatly increase the number of I/O terminals of a network. Thus, signature analysis or other similar methods requiring direct access to DFFs are not followed here and interested readers are referred to other articles such as [97].

The proposed high-level testing method is to diagnose the network by appropriate operations on the output stream. After the testing polynomial $P(x)$ is applied to a RUT, a fault f_i changes $Q(x)$ into $Q_i(x)$, where $Q(x)$ (or $Q_i(x)$) is the correct (faulty) output polynomial of the RUT. The procedure is then to find a testing polynomial $P(x)$ and an operation Θ_{f_i} such that $\Theta_{f_i}(P(x), Q(x)) = Q_i(x)$. The combination of $P(x)$, its output $Q(x)$ and the operation Θ_{f_i} is called a *testing routine* for the fault f_i.

3.3 Fault Models

A PSN is composed of links and switches. There are $r!$ possible interconnection patterns within an $r \times r$ switch. There are then $(r!)^{\frac{N}{r}\log_r N}$ different conflict-free interconnection patterns in an $N \times N$ PSMIN. Links' stuck-at faults are equivalent to stuck-at faults of the switches to which they are attached. Thus, only switch faults are considered for the high-level testing. That is, link stuck-at faults are implicitly included in the switch fault models.

Permanent *multiple stuck-at*, *delay*, *merging*, *partial setting*, *blocking*, *broadcasting* and *misrouting* faults are all considered in this chapter. A MSA fault occurs when one or more signal lines are fixed at 0 or 1. A delay fault occurs when the operation speed of some component(s) is slower than the specified and, thus, erroneous operations result. A partial setting fault occurs when some of the identical components in a unit do not provide the same operation as the others. A blocking fault occurs when an appropriate route within a switch cannot be established for a request. A handshake signal deadlock is an example of blocking fault. A switch has a merging (broadcasting) fault when two or more input (output) ports are connected to one output (input) port. A misrouting fault represents the case when packets are mis-directed to incorrect output ports. Stuck-line faults at gate level are tested at the low-level testing.

3.4 PSMIN Diagnosis

we present testable designs and testing methods on the basis of the polynomial operations and the fault models introduced in Sec. 3.2 and Sec. 3.3, respectively. The network is designed such that all signal lines have only two states, i.e., 1 or 0, whether or not they are used to transfer data. The output port of a switch is a combination of multiplexers and demultiplexers (MU/DEXes). A MU/DEX is basically composed of AND and OR gates. When multiple requests are assigned to an output port, a combination of OR/AND functions among the requests will take place.

3.4.1 High-Level Diagnosis

Assume that the PSMIN under test connects N sources and N destinations and is built with $r \times r$ switches. The number of stages in the PSMIN is $k \equiv \log_r N$. To describe the PSMIN's topology and permutation, the input (output) ports of all switches in each stage are

vertically indexed. The number assigned to an input (output) port is called its *global index*. For each $r \times r$ switch, there is a one-to-one correspondence between the global index and the input/output port number: $f_i(j) = m$, where j is the port number of the i-th switch at a stage, and m is the port's global index. A *link permutation* T_i, $1 \leq i \leq k$, is a one-to-one mapping from the output ports at stage $i-1$ to the input ports at stage i. On the other hand, a *switch permutation*, $E_m^i : f_i(j) - f_i((j+m) \bmod r)$, is a one-to-one mapping from input ports of a switch to its output ports, $0 \leq m \leq r-1$. For simplicity, all the switches on the RUT are assumed to have an identical permutation, i.e., $i_1 = i_2$, for all $E_m^{i_1}, E_m^{i_2} \in RUT$, and E_m will henceforth be used to denote E_m^i. More general cases than this can be easily derived by using the actual permutation at each stage. To allow for simultaneous diagnosis and normal operation during the high-level testing, the testing processors should be equipped with complete information of link and switch permutations.

Testable Design

Links are passive components and can be treated as data paths of switches, whereas switches make all switching decisions and also contain memory elements. To make the network easily testable, switches are designed to have two operational modes: *normal* and *testing* modes.

As mentioned in the Introduction, a RUT can be viewed as a cascaded shift register array. A FL and XOR gates must be added to transform a 1-bit wide RUT into a bit PDM. Since links are the predominating cost factor of a PSMIN, the link overhead in improving testability must be kept as small as possible. A tracer in each switch is thus proposed to minimize the width of FL. A tracer is composed of a testing pattern masker and mapper, a feedback/feed-forward selector (F-selector) and a modulo TWO adder, where TWO = { *ONE* , *ZERO* }. The masker examines if bits of the testing pattern are identical and maps the testing pattern from *ONE* (*ZERO*) to 1(0) for the FL. The mapper transforms 1(0) to *ONE* (*ZERO*) to use the

adder. The F-selector determines the transmission direction of FL.[1] An adder is necessary for each switch to form a block on a route for data path diagnosis.

Four possible operational states, S, A, X and N, are assigned to a switch when the network is being tested. Once a switch in a RUT is in state S, the switch will not allow any packets, except those from the same RUT, to enter the RUT, and the operations of switches on the route are synchronized. State S can be taken as a sub-operation of the other states, because the tracer in the other states is activated *and* switch operations are synchronized. When the switch is in state N, only FL and the F-selector are activated. When a switch at stage i is in state A, the F-selector blocks the FL signals from stage $i+1$, and the current switch's output is led to the FL. When the switch is in state X, the data on FL is mapped, by the mapper, from 1 (0) to *ONE* (*ZERO*), and the logic operation $BU_0 \leftarrow P_{in} \oplus FL$ is performed at the input of the queue, where BU_0 is the input of the queue and P_{in} the input packet. Fig. 3.3 shows these switch operations in different states. The logic diagram in Fig.3.4 shows a switch design example of the high-level testing.

A switch can enter/exit the testing mode by command packets. Two formats, *data packets* and *command packets*, are used to control the switch operations. A command packet is composed of routing tags and a command array {**CA(1)**, $\cdots$,**CA(k)**}, where k is the number of stages of the network and **CA(i)** is a 2-bit command word associated with stage i. A switch at stage i will enter states S, A, N and X, when **CA(i)** = 00, 11, 10 and 01, respectively. The type of packets can be identified by a one-bit flag in each packet. As shown below, this testing method can also identify a misinterpreted command array (by a faulty switch).

Theorem 1 : All misinterpreted command packets can be tested in one testing routine.

Proof: Once a RUT is transformed into a multiplier, the test pattern for misinterpreted com-

[1] The F-selector can be eliminated if the RUT is to be transformed into either a multiplier or divisor, but not both.

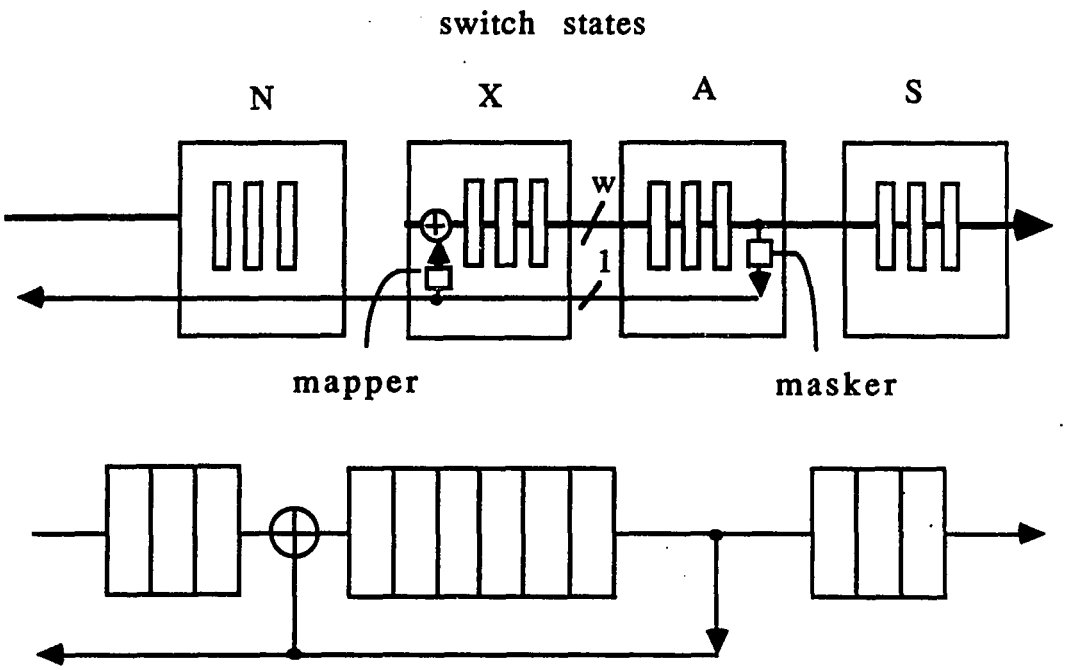


Figure 3.3: Switches on a RUT and the corresponding word divisor

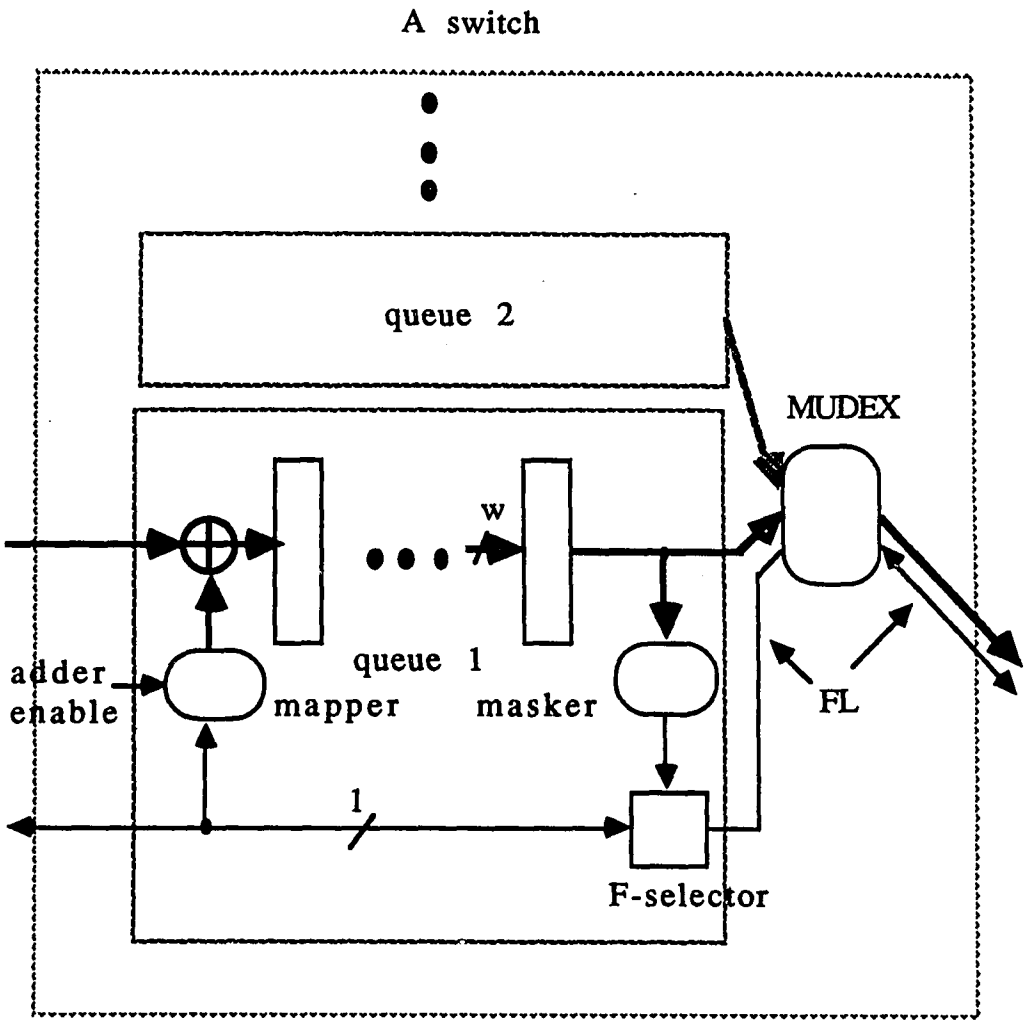


Figure 3.4: A testable design of switches for concurrent testing.

mand packets becomes an impulse polynomial. From Lemma 1, $M(x)$ of the RUT can be uniquely identified. ■

Data Path Stuck-at Faults

All switches are in state X when data path stuck-at faults are being tested. A SSA fault at the high-level represents a stuck-at fault(s) in a single switch. But a MSA fault at the high-level implies stuck-at faults in more than one switch. In a conventional approach, upon detection of a fault on some route, test patterns must be submitted from processors on different routes to locate the fault. It is shown below that the fault location with the polynomial testing is much easier than that with the conventional approach.

SSA Faults:

Every switch is set to an identical permutation. When $r \times r$ switches are used, r different switch permutations, $\{E_i \mid 0 \leq i \leq r-1\}$, are necessary to test every data path within a switch. For any input port of a switch, its data paths to all the output ports are included in $\{E_i \mid 0 \leq i \leq r-1\}$. Thus, in these r permutations every data path from each input port to every output port is tested.[2] The procedure can be generalized as follows: in testing routine m, the switch permutation E_m, $0 \leq m \leq r-1$, is performed first. Then, the connection of source i to destination j is specified by $j = T_k E_m T_{k-1} E_m \cdots T_2 E_m T_1(i)$. The special case of $r = 2$ allows data path stuck-at faults to be detected in two permutations, each of which is composed of two steps [38].

Theorem 2 : When a locked RUT_{ij} is configured as a multiplier, a SSA fault on the data path can be located by processor j in one testing routine.

Proof: The testing polynomial for the data path SSA fault is $W_n(x)$, where n is the total length

[2] Only r permutations are needed to test a data path, although $r!$ permutations are required to test the routing functions.

of buffers on RUT_{ij}. As discussed earlier, RUT_{ij} can be expressed as $M_{ij}(x) = \sum_{i=0}^{n} m_i x^i$. The output at the destination j becomes $Q(x) = \sum_{l=0}^{n} m_l x^l\, W_n(x)$. $Q(x)$ should then have the format of $1\cdots10\cdots01\cdots$, where a $0(1) \rightarrow 1(0)$ transition takes place at each position of an XOR gate on RUT_{ij} and the number of consecutive 1's (0's) in the i-th block is the size of B_i. For example, the output stream of the multiplier in Fig. 3.2(a) is 10011100. When M_{ij} changes to $M'_{ij} \neq M_{ij}$ due to a SSA fault, there must be at least one i such that $m_i \neq m'_i$, $1 \leq i \leq k$, by Lemmas 2 and 3. When the number of $0 \rightarrow 1$ transitions is m_f, the faulty switch can be located by $s_f = (\prod_{i=0}^{m_f} E^{-1}\, T^{-1}_{m_f - i})\,(j)$, where T_i^{-1} is the inverse of permutation T_i. ∎

MSA Faults:

A MSA fault on a data path cannot be determined in one testing routine. However, the polynomial testing can be applied to a sequential repairing procedure which locates and then replaces leading faulty switches/links in each testing routine.

Theorem 3 : A MSA fault on a data path can be repaired in k testing routines, where k is the number of stages of the network.

Proof A MSA fault is the collection of multiple SSA faults. When the testing polynomial $W_n(x)$ is applied to a PDM, $Q(x)$ is uniquely determined by the type (s-a-0 or s-a-1) and location of the leading stuck-at fault. In other words, the lowest-order faulty switch can be located in each testing routine, regardless of the cardinality of the multiple fault. Since there are k switches on a route, at most k steps are required to repair the network. ∎

Delay Faults:

A delay fault on a data path is detectable when its operational speed is at least one clock cycle slower than specified.

Theorem 4 : A single delay fault of longer than one clock cycle can be located in one testing routine.

Proof: The polynomial $P_E^1(x) = \sum_{i=0}^{k} x^{2i}$ can detect all delay faults. However, a polynomial $P_E^m(x) = \sum_{i=0}^{k} x^{(m+1)i}$ can be used to distinguish a $1 - 0$ transition delay fault of m clock cycles from delay faults of less than m cycles. When an m unit delay fault occurs and $P_E^m(x)$ is applied, the faulty switch's output becomes $W(x)$. By forming a PDM on RUT_{ij}, a delay fault can be located in one testing routine. A testing polynomial for $0 - 1$ delay transitions is complemented to become $P_E^m(x)$ and the output is $\overline{W}(x)$. ■

Like MSA faults, a multiple delay fault composed of different delay lengths can be repaired in k testing routines.

Routing Faults

Methods for locating routing faults are studied in this subsection. Switches are set to state S when routing functions are tested.

Merging and Broadcasting Faults:

Depending on the implementation details, a merging fault can be located in one testing routine when appropriate polynomials are applied. A Δ-*merging* fault occurs when a Δ (i.e., AND or OR) operation results from the merging of two or more switch input/output ports.

Consider the effect of the OR merging first. For two routes RUT_{i_1} and RUT_{i_2}, they will topologically intersect in at most one switch when the network is not redundant.

Theorem 5 : For a given permutation, a multiple OR-merging fault can be located in one testing routine for both distributed and centralized routing control PSMINs.

Proof The testing polynomial at processor j is $P_j^N(x) = \sum_{i=1}^{N} c_i x^i$, where $c_j = ONE$ and $c_i = ZERO$, $\forall\ i \neq j$. First, consider the case when two RUTs are merged. The two routes

from i_1 and i_2 under the given permutation intersect at most once. When the intersecting switch has an OR-merging fault, and the testing polynomials $P_{i_1}^N(x)$ and $P_{i_2}^N(x)$ are applied, there will be an OR operation between these two polynomials. Without loss of generality, $P_{i_1}^N(x)$ can be assumed to be merged into $P_{i_2}^N(x)$, i.e., $P_{i_2}'(x) = P_{i_1}^N(x)\ OR\ P_{i_2}^N(x)$. Since there is no overlap of the positions containing 1's in both $P_{i_1}^N(x)$ and $P_{i_2}^N(x)$, new information on the merging fault is added to $P_{i_2}'(x)$. Applying the XOR operation between $P_{i_2}'(x)$ and $P_{i_2}^N(x)$ at the destination of $P_{i_2}'(x)$, we get $P_{i_2}(x) = P_{i_2}'(x) \oplus P_{i_2}^N(x)$. A nonzero resulting polynomial implies that some polynomial is merged into $P_{i_2}^N(x)$. The switch with the merging fault is determined by the topology. That is, $P_{i_1}^N(x)$ merges with $P_{i_2}^N(x)$ at $S(i_f, j_f)$, where $S(i_f, j_f)$ is the the j_f-th switch located at stage i_f , when $(j_f - 1)r = \prod_{i=1}^{i_f} E_m T_i(i_1)$ $- \prod_{i=1}^{i_f} E_m T_i(i_1) \bmod r$ and $(j_f - 1)r = \prod_{i=1}^{i_f} E_m T_i(i_2) - \prod_{i=1}^{i_f} E_m T_i(i_2) \bmod r$. It is easy to see that no information will be lost when multiple merging faults occur. Thus, all multiple merging faults can be determined in one testing routine. ■

If merging faults are assumed to be independent of the interconnection pattern, they can be located in one testing routine. Otherwise, we need $r!$ tests to set each switch to every interconnection pattern for fault location. The *AND-merging* fault can be diagnosed by the same method with the testing polynomial, $\bar{P}_j^N(x)$.

A broadcasting fault at one input port of a switch implies a merging fault at the output port of the broadcast data path. Thus, broadcasting faults can be located by the same procedure used for testing merging faults.

Misrouting Faults:

There are $r!$ possible permutations in an $r \times r$ switch. To locate a misrouting fault, the testing polynomial $P_i(x)$ for source i must be unique.

Theorem 6 : One testing routine is sufficient to locate a multiple misrouting fault for both

distributed and centralized routing control PSMINs.

Proof: The testing polynomial for merging faults can also be used for testing misrouting faults. $kr!$ permutation calculations are required in each testing routine. Given a permutation $j = T_k E T_{k-1} E \cdots E T_1(i)$, a misrouting fault results when E becomes E', where $E' \neq E$ is a faulty permutation. The fault locating procedure is to find E' of a faulty switch. For a given processor j which receives an incorrect polynomial, all possible permutations have to be calculated to find E' of the faulty switch. Since each switch has $r!$ permutations, we need $kr!$ inverse permutations to locate the faulty switch. ■

A misrouting fault may be caused by either the misdecoding of a routing tag in the RCU of a faulty switch, or a stuck-at link/switch which transmits the routing tag before the routing tag is actually decoded.

Blocking Faults:

As mentioned earlier, the network is designed such that there are only two logic values, i.e., 0 and 1, in all signal lines. When a blocking fault occurs, a data path cannot be utilized, even though it is available.

Theorem 7 : A blocked data path in a centralized routing control PSMIN can be located in one testing routine.

The proof of this theorem is straightforward. In a centralized routing control network, a locked route can be established even when its data path is blocked. Since the output of a blocked switch is fixed at 1 or 0, it has the same output as a stuck-at data path. It is much more difficult to locate a blocking fault in a distributed routing control network, because routing tags and data are blocked at the same time. It can be located by a binary search which requires $\log_2 k$ testing routines.

Partial Setting Faults:

When a data path is partially stuck, the testing procedures with multipliers can still be applied. Test patterns, however, must be determined by the design details of the masker and the mapper. In case of a partial fault, unaffected data bits have correct outputs but the stuck-at bit needs the same testing procedures as described above. In such a case, we have to examine faulty bit(s) instead of a faulty word.

Pattern Generation

Test patterns are generated by pattern generators $\{G_i\}$ which are processors or dedicated hardware mechanisms. The cost of pattern generators is one of the most important factors for evaluating the performance of a testing method. Only two testing patterns, $W_n(x)$ and $\{P_i^N(x)\}$, need to be generated for the high-level testing. Both patterns can be easily generated when G_i's are *ringed* through a single bit control line. Denote the input and output of the ring in G_i by $D_{i(in)}$ and $D_{i(out)}$, respectively. $D_{N(out)}$ is connected to $D_{1(in)}$, and $D_{i(out)}$ is connected to $D_{i+1(in)}$, $\forall\ 1 \leq i \leq N-1$. To generate $\{P_i^N(x)\}$, the ring is initialized as: $D_{1(in)} = 1$, $D_{i(in)} = 0$, $\forall\ i,\ i \neq 1$. Operations of G_i at the k-th clock cycle are given as:

$$\text{OP1.} P_i(k) = \begin{cases} \textit{ONE} \text{ when } D_{i(in)} = 1 \\ \textit{ZERO} \text{ when } D_{i(in)} = 0\ , \end{cases}$$

$$\text{OP2.} D_{i(out)}(k) \leftarrow D_{i(in)}(k)\ ,$$

where $P_i(k)$ is the pattern generated by G_i at the k-th clock cycle. The other test pattern, $W_n(x)$, can be easily generated by the initialization $D_i(out) = \textit{ONE}$, $\forall\ i \leq N$, and applying OP1 and OP2 in each pattern generator. For a given permutation, there are only rk possible merging faults on a route and the above testing polynomial is thus not optimal for testing OR-merging faults. For testing OR-merging faults, the length of the testing polynomial can be reduced to rk, when $P_k(x) \neq P_j(x)$ for any pair of polynomials $P_j(x)$ and $P_k(x)$ intersecting in a switch under a given permutation. However, the testing polynomial allows merging

and misrouting faults to be tested simultaneously, and, thus, simplifies testing procedures. Moreover, G_i has a very simple structure and can be easily applied to various interconnection networks.

Testing Complexity

It is important to consider the testing complexity of the high-level testing. The length of test pattern for data path stuck-at faults and misinterpreted command packets is km, where m is the queue length in each switch.[3] The calculation of a misinterpreted command packet is straightforward, because the coefficients of the multiplier can be identified directly from the output stream. The stuck-at-a faults at the inputs of XOR gates, to which the FL are connected to, can be tested by an all zero polynomial, and its testing length is km. To test single data path stuck-at (delay) faults, we need one testing routine which is composed of at most k steps of inverse permutations. At most k testing routines are necessary to repair all multiple data path stuck-at faults, and each testing routine needs k inverse permutations. Thus, a total of $k^2 + k + 2$ inverse permutations is needed for data path diagnosis.

For routing faults, the test pattern length is N. One testing routine is sufficient to identify all merging and broadcasting faults. To locate a merging (broadcasting) fault, two RUTs are needed at a time. Since there are k switches on a RUT and each switch needs $r!$ inverse permutations, $k^2r!$ inverse permutations are required to locate a merging (broadcasting) fault. Finally, $kr!$ inverse permutations are required to locate the misrouting faults.

The high-level testing is quite general to handle various circuit implementations and locate faults without completely stopping the normal operations of the network. The testing time varies with the size of the network. Note, however, that the high-level testing may not detect all possible faults for different circuit implementations. When the high-level testing fails to

[3] The queue lengths need not be identical.

locate some faults, a fast off-line testing method with high fault coverage needs to be called for. The low-level testing described below is to meet this very need.

3.4.2 Low-Level Testing

A switch is composed of data paths and a RCU. Data paths consist of links, queues and MU/DEXes. A pool of buffers, BU_i^j, $1 \leq i \leq m$, in a switch constitutes the j-th queue of the switch, where m is the number of buffers within the queue. A buffer can store one w−bit packet. There are then at least $Nwm \log_r N$ memory bits in an $N \times N$ PSMIN built with $r \times r$ switches, and a CSN is the special case of $m = 0$. Let BU_0^j and BU_{m+1}^j denote respectively the input and output ports of a switch. It is shown in Fig. 3.5 that these buffers are cascaded, or *C-connected*, and formally described by $CN : BU_i - BU_{i+1}$, where "—" denotes an interconnection within a queue, called an *interlink*.

Different implementations of registers need different test patterns. We use random testing to test the queues. However, when specific test patterns like the one in Appendix I is needed, they are easy to generate. In each switch, queues are tested by generation and comparison of polynomials. For the generation of a polynomial we can use the natural structure of a queue. The basic idea is to convert the queue into two PGs. A queue can be taken as a $w \times m$ matrix M in which each column is a buffer of w DFFs. Note that DFFs in each row j (collection of the $j-th$ DFFs of m buffers), $1 \leq j \leq w$, of the matrix are cascaded by its natural structure. Assuming w to be even, two PGs, PG_1 and PG_2, are formed by properly cascading the rows of M.

Two symmetric PGs can be obtained by (1) horizontally halving the buffers in the queue, (2) connecting $M(i+1,1)$ to $M(i,m)$ $\forall\ i \leq \frac{w}{2}$ for PG_1, and $M(i+1,m)$ to $M(i,1)$, $\forall\ i \geq \frac{w}{2}+1$ for PG_2, (3) identically connecting registers' outputs to the feedback XOR gates in PG_1 and PG_2, and (4) connecting the output of the XOR gate outputs of PG_1 and PG_2

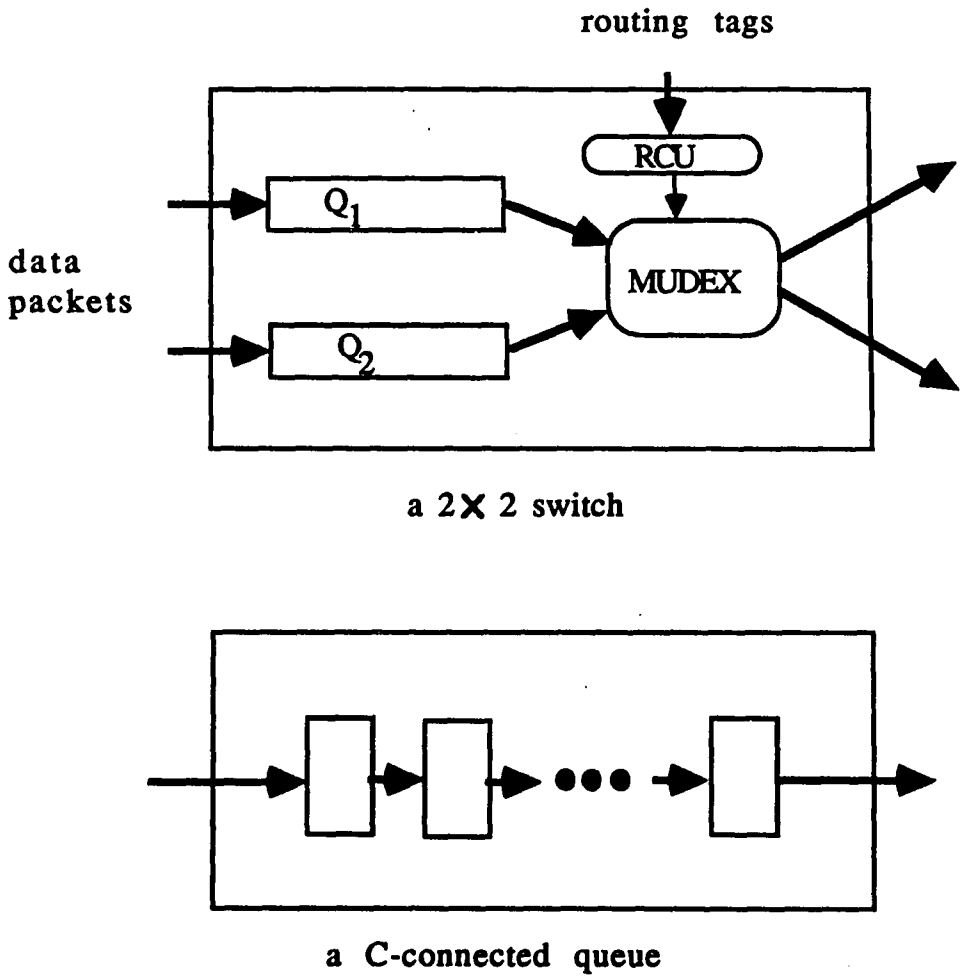


Figure 3.5: The structure of a 2 x 2 switch and a C-connected queue.

to $M(1,1)$ and $M(\frac{w}{2}+1,1)$, respectively. It is well-known that the maximum period of the output stream of a PG can be obtained when $2^l - 1$ is a prime number, where l is the PG's length, and the PG's characteristic function is irreducible [47]. A fault is detectable when it yields different output sequences in the two PGs.

The PGs' outputs form a 1-out-of-2 codeword when an inverter is added to one XOR gate's output. An XOR gate with n inputs needs $n+1$ test patterns when n is odd; on the other hand, three test patterns are sufficient for an XOR gate with an even number of inputs. The test patterns for the XOR gate with an odd and an even number of inputs are $\{0\cdots0,\ 10\cdots0,\ 010\cdots0,\ \cdots,0\cdots01\ \}$, and $\{\ 0\cdots0,\ 1\cdots1, I_s\}$, respectively, where I_s is any input with an odd number of 1's. The test patterns for the XOR gate of a PG can be easily generated by setting the PG's initial state. Since every component in the PGs is tested, there is no hard core in this design.

When two symmetric PGs are used, unidirectional stuck-at faults in a buffer cannot be detected. To solve this problem, PG_1 can be modified such that the outputs $M(i,\frac{m}{2}),\ \forall\, i \le \frac{w}{2}$, are connected to the XOR gate whose output is then connected to $M(1,\frac{m}{2}+1)$. Although the physical interconnection of $M(1,1)$ to $M(\frac{w}{2},m)$ is different from that of $M(\frac{w}{2}+1,1)$ to $M(w,m)$, both PG_1 and PG_2 still have an identical structure. Such a modification can now detect the unidirectional faults mentioned above. Symmetric and asymmetric PG configurations are illustrated in Figs. 3.6 and 3.7, respectively.

The optimal testing length of a PG and its fault coverage are important performance parameters. Any DFF in the MSA fault model can be s-a-1, s-a-0 or fault-free. To evaluate the MSA fault coverage of the proposed method, we only need to consider the type and position of leading faulty DFFs in a block. Consider a pair of leading faulty DFFs, s_1 and s_2, which are in x^i and x^j positions of PG_1 and PG_2, respectively. The effects of faults in s_1 and s_2 can be distinguished only when they yield different outputs for at least one clock cycle. We

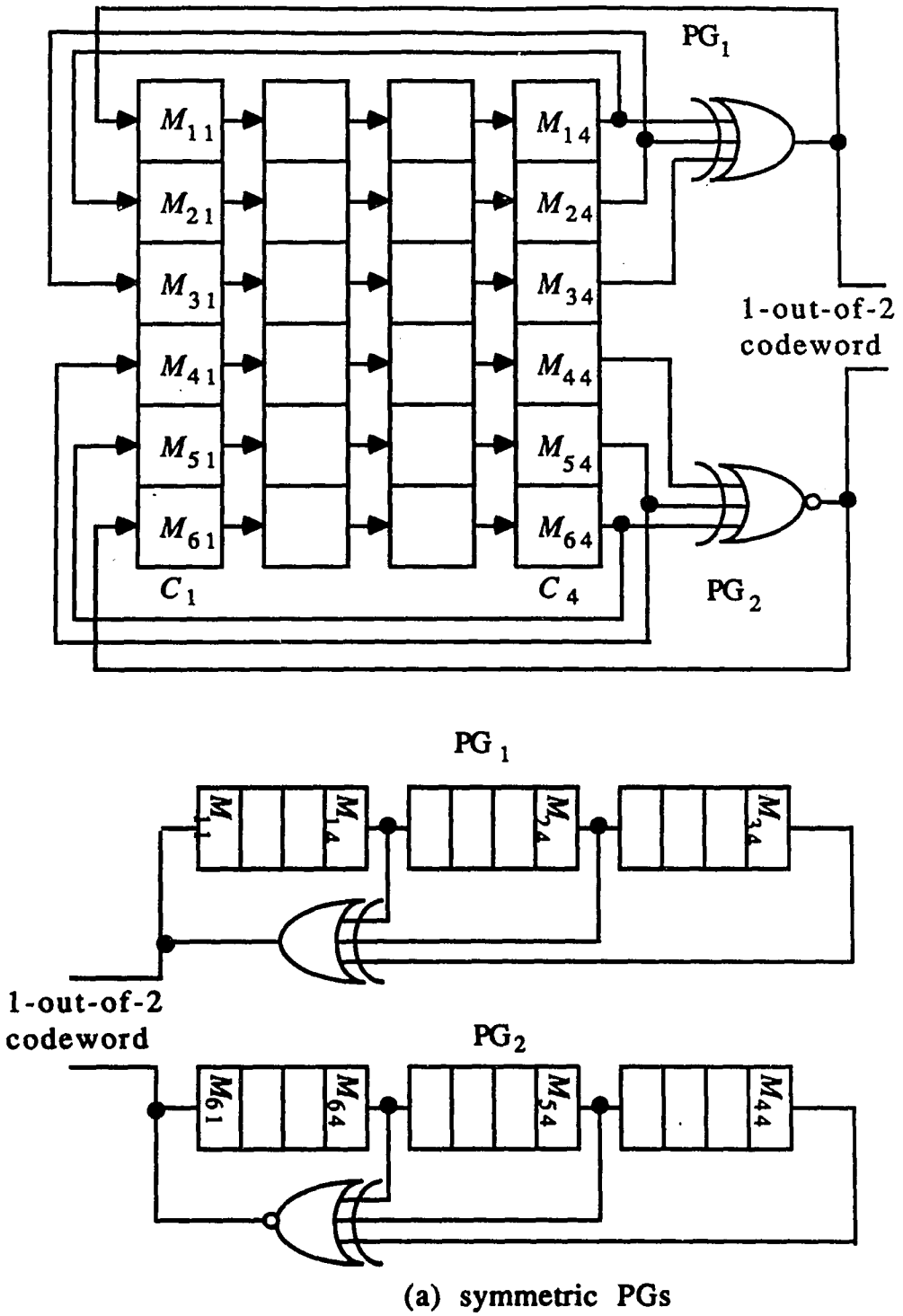


Figure 3.6: A queue converted into two symmetric PG's.

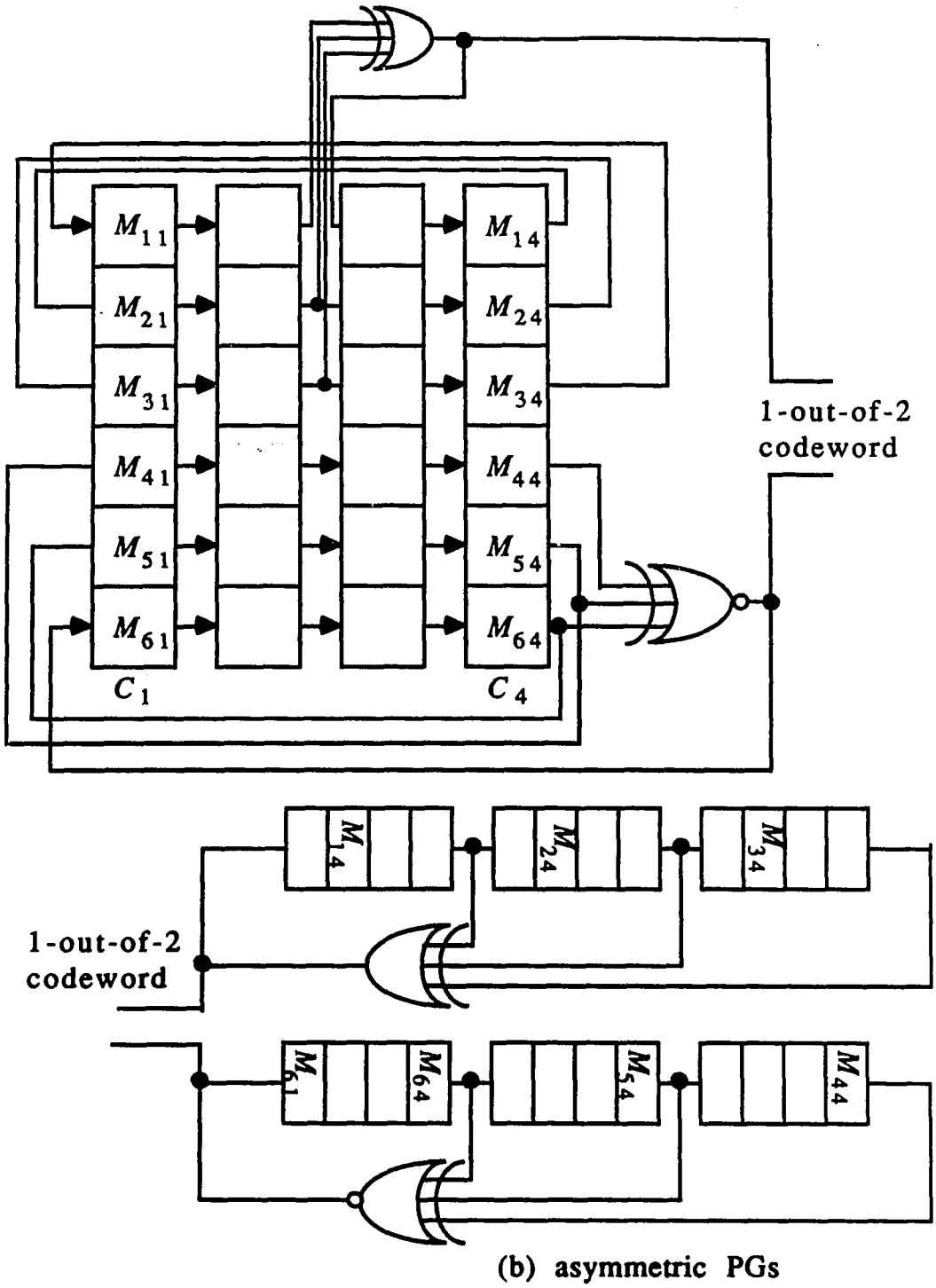


Figure 3.7: A queue converted into two asymmetric PG's.

begin with the simplest special case of the MSA fault model, i.e., the SSA fault model.

Theorem 8 : All SSA faults are detectable, and the maximum testing length is $r+1$, where r is the order of the PG.

Proof: A SSA fault in a PG is detectable when it generates an output different from that of the other PG. Let the initial state of the PG be $\sum_{i=1}^{r} n_i x^i$, where $n_1 = 1$ and $n_i = 0$, $\forall\, i \neq 1$. When a s-a-0 is located at output of x^i, n_1 is falsely inverted at the i-th shift. The fault cannot be revealed during the first $i-1$ clock cycles, because the s-a-0 is the same as the preset value of a fault-free circuit. The worst case occurs when the s-a-0 is located at the output of x^r, and, thus, r is the maximum testing length.

When a s-a-1 is located at the output of x^i, it will change the parity of the output immediately when it propagates to a feedback line. The worst case occurs when feedback lines emanate from x^1 and x^r, and the s-a-1 is present at the input of x^1. The output of the faulty PG is the same as the non-faulty one until the $r+1$-th clock cycle. Thus, the maximum testing length for SSA faults is $r+1$. ■

To calculate the MSA fault coverage, the position and type of leading faulty DFFs must be considered. Each DFF can be s-a-1, s-a-0 or fault-free, and the number of MSA faults in a queue is $3^{mw} - 1$. Due to the fault masking effect, the actual computing time is $K\prod_{i=1}^{k}(2r_i+1)^2$, where r_i is the order of block B_i and K the computing time required for each iteration. As shown in Table 3.1, various testing strategies are simulated to examine their MSA fault coverages.

The initial state of each simulation is $n_1 = 1$, $n_i = 0$, $\forall\, i \neq 1$. From the simulation results, the following three conjectures are made:

Conjecture 1: The fault coverage is dominated by the number of feedback lines. It monotonically increases with the number of feedback lines. The length

of a PG has little effect on the fault coverage.

Conjecture 2: For a given PG of length r, and l_f feedback lines, the MSA fault coverage attains a maximum when feedback lines are located at $x^1, x^2, \cdots x^{l_f-1}$, and x^r.

Conjecture 3: The MSA fault coverage increases with the number of testing routines, each of which uses a different initial state. The optimal testing length for MSA faults is r for a given initial state and a feedback configuration.

For a given PG configuration and the initial state, theoretically, 2^r shifts are required to exercise all the states of the PG. However, our simulation results show that testing lengths are rarely required to be longer than the length of the PG. Although the choice of an initial state affects the fault coverage, the number and location of feedback lines are the dominating factors in the fault coverage. It is shown in Fig. 3.8 that about 65% of detectable MSA faults are immediately detected for most cases. From Theorem 8 and the above conjectures, each testing length is found to be $r + 1$.

Unlike the off-line testing of data paths, a faulty RCU can be detected on-line. A *RCU checker* is proposed to detect faults in the RCU using its output signals. A RCU has an $r \log_2 r$-bit input and an r^2-bit output. The RCU output signals are denoted by E_{ij}, $1 \leq i, j \leq r$, where $E_{ij} = 1$ if queue j is connected to output port i, and $E_{ij} = 0$ otherwise. For any fixed k, $\{E_{ik}\}$ or $\{E_{ki}\}$, $1 \leq i \leq r$, forms a 1-out-of-r codeword. Thus, $2r$ 1-out-of-r self-checking checkers, one for each $\{E_{ik}\}$ or $\{E_{ki}\}$, are needed to detect all non-codeword outputs.

The outputs of the RCU and queues are the inputs of the MU/DEX to which they are connected. The RCU and queues can be tested first using the above procedures. If they are fault-free, then the MU/DEX and the links connected to the MU/DEX are tested by using the RCU and queues to generate test patterns for the the MU/DEX and its links. For output

feedback (f_i)	Stage No.(k)								
	3	4	5	6	7	8	12	16	20
k,0	.728	.715	.710	.709	.708	.708	.708	.708	.708
k,1,0	.780	.788	.793	.794	.794	.795	.795	.795	
k,2,1,0	.827	.828	.831	.831	.832	.831			

(a) Fault coverage (C) of different numbers and locations of feedback lines.

f_1, f_2, f_3	8,1,0	8,2,0	8,3,0	8,4,0	8,5,0	8,6,0
C	.795	.785	.780	.777	.770	.746

(b) k=8 with three feedback lines.

$C^{(k,f_1',f_2')}$; $f_1 = k, f_2 = 0$									
3,2,1	4,2,1	5,2,1	6,2,1	7,2,1	8,2,1	9,2,1	10,2,1	11,2,1	12,2,1
.868	.876	.877	.876	.875	.875	.875	.875	.875	.879
8,1,0	8,3,1	8,3,2	8,4,1	8,4,3	8,5,1	8,5,3	8,5,4	8,6,1	8,6,3
.875	.876	.876	.879	.877	.886	.885	.879	.900	.899
8,6,5	8,7,0	8,7,1	8,7,5	8,7,6	16,1,2	18,10,9	19,10,9	20,11,10	21,12,11
.874	.801	.875	.883	.869	.875	.875	.875	.875	.875

(c) PG's tested twice by two feedback configurations.

$C^{(k,in_2=2^n)}$; first initial condition $in_1 = 2^0$									
3,1	3,2	4,1	4,2	4,3	5,1	5,2	5,3	5,4	6,1
.762	.809	.748	.764	.805	.745	.749	.764	.803	.743
6,2	6,3	6,4	6,5	7,1	7,3	7,5	7,6	8,1	8,2
.745	.749	.764	.803	.743	.745	.764	.803	.743	.745
8,4	8,6	8,7	9,1	9,3	9,5	9,7	9,8	10,1-5	10,6
.745	.764	.803	.743	.743	.745	.764	.803	.743	.745
10,7	10,8	10,9	12,1-7	12,8	12,9	12,10	12,11	16,1-11	16,12
.749	.764	.803	.743	.745	.749	.764	.803	.743	.745
16,13	16,14	16,15							
.749	.764	.803							

(d) Feedback lines at k,0, and the PGs are tested twice with two different initial states.

Table 3.1: The fault coverage of MSA faults under different conditions.

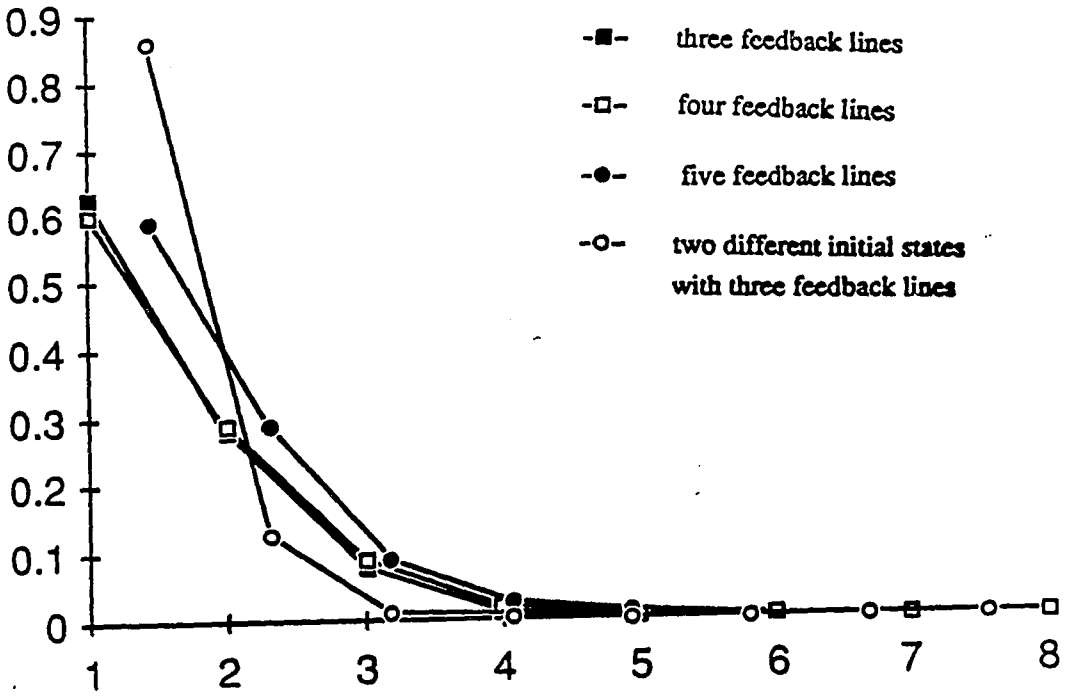


Figure 3.8: Detected-faults/detectable-faults vs. number of shifts when r=8.

verification, the streams from the MU/DEXes of stage i are transmitted through the links and then verified at stage $i+1$ with special mechanisms.

Before we develop the test method for MU/DEXes and links, it is necessary to find the test patterns of the $r \times 1$ multiplexer shown in Fig. 3.9, where E_i and D_i are the enable and data of the i-th input, respectively. The $r \times 1$ multiplexer is implemented by r two-input AND gates and an OR gate.

Lemma 4 : All SSA faults in the multiplexer of Fig. 3.9 can be detected in $r+2$ steps.

Proof: After fault collapsing, the faults that need to be tested are: (1) s-a-0 and s-a-1 primary output, i.e., output of the OR gate in Fig. 3.9, (2) s-a-0 A_i, $\forall\ i \leq r$ in Fig. 3.9, and (3) s-a-1 $D_i(E_i)$, $\forall\ i \leq r$. Test patterns can be derived as follows.

PT(1): $E_i D_i = 10, \forall\ i \leq r,$

PT(2): $E_i D_i = 01, \forall\ i \leq r,$

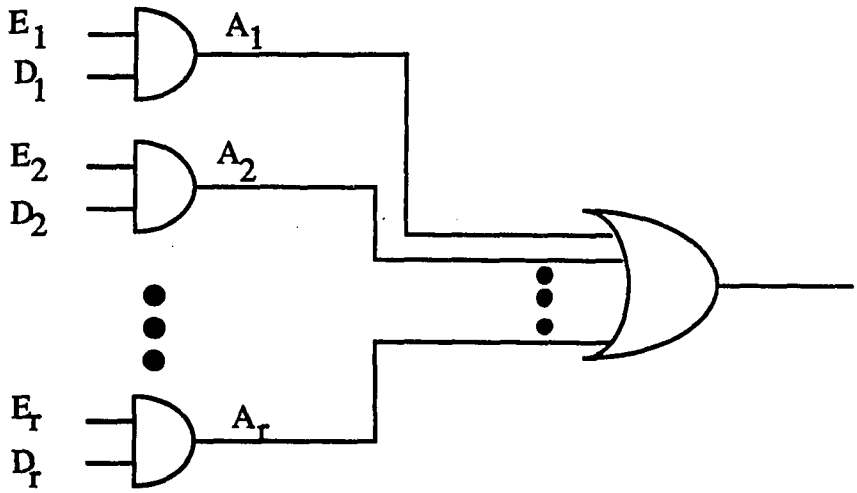
PT(3): $E_1 D_1 = 11$, and $E_i D_i = e_i d_i$ for $i \neq 1$

PT(4): $E_2\ D_2 = 11$, and $E_i D_i = e_i d_i$ for $i \neq 2$

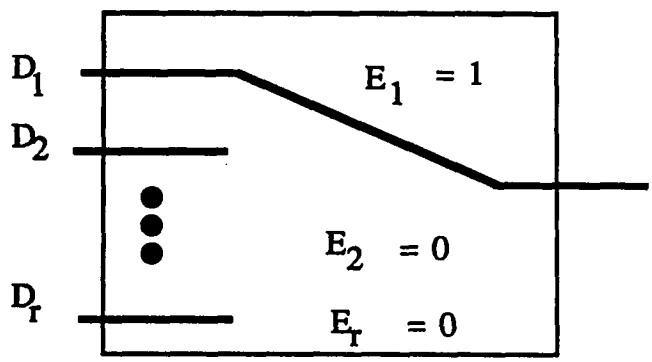
... ...

PT(r+2): $E_r D_r = 11$, and $E_i D_i = e_i d_i$ for $i \neq r$, where $d_i = 0$ or $e_i = 0, \forall\ 1 \leq i \leq r$.

An $r \times r$ MU/DEX connects r queues' outputs to r links. The MU/DEX can be implemented by two level AND and OR gates, where each MU/DEX's output port is basically a multiplexer. An example design of MU/DEX is shown in Fig. 3.10, where E_{ij} is the enable signal from the RCU to route the packet at queue j to output port i. E_{ij} fans-out to w branches to simultaneously enable the w bits of queue j.



(a) logic diagram



(b) functional diagram

Figure 3.9: **The logic and functional diagrams of a multiplexer with r data inputs and r enable signals.**

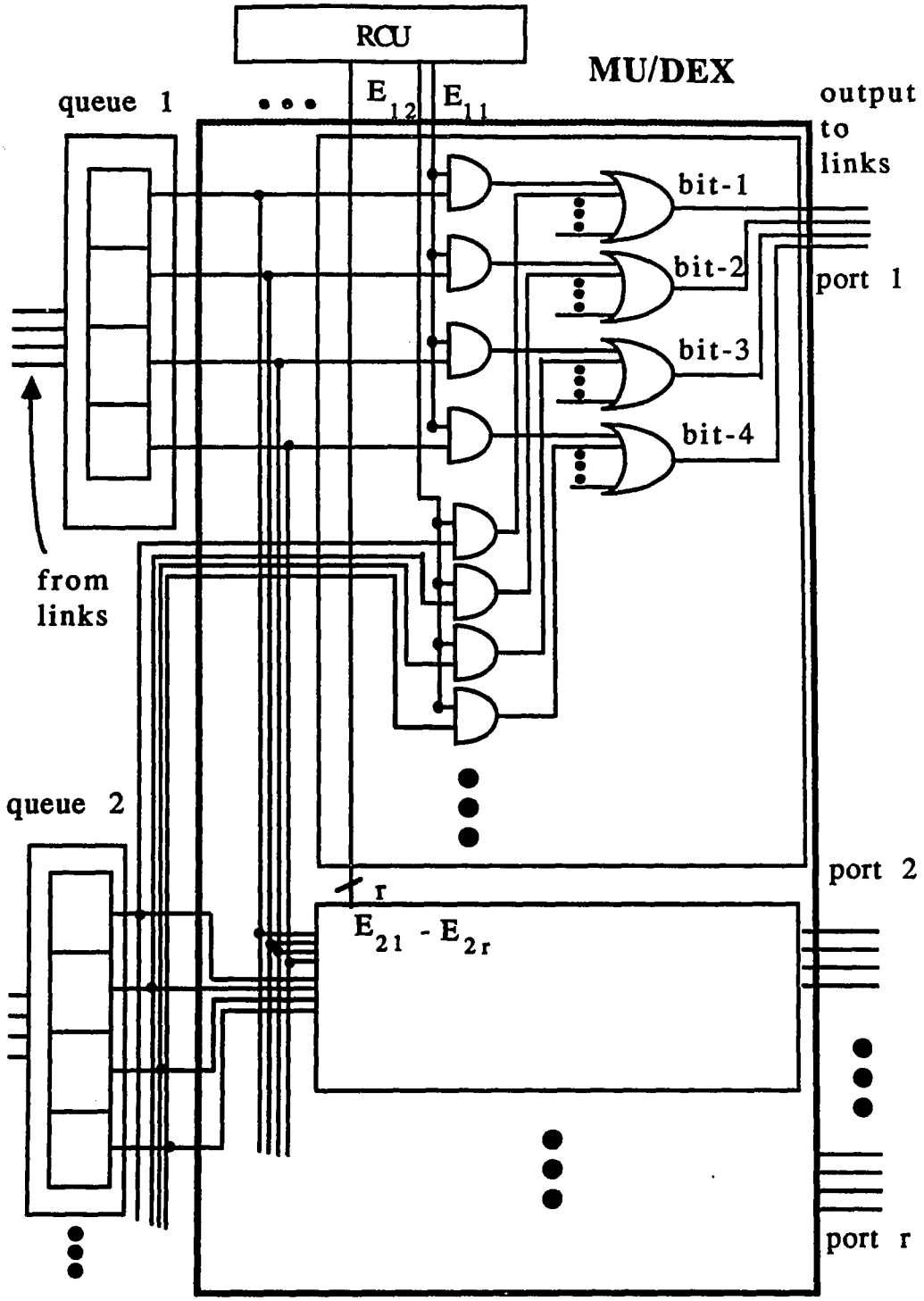


Figure 3.10: An example of the MU/DEX in an $r \times r$ switch.

Theorem 9 : Any SSA fault on links or MU/DEXes can be tested in $r + 2$ clock cycles.

Proof: Since operations to be applied to each of the w bits of a packet are identical, it is sufficient to discuss only one bit of the packet. Each output port of the 1-bit MU/DEX is an $r \times 1$ multiplexer, and there are a total of r multiplexers in an MU/DEX. Test patterns derived in Lemma 4 can be directly applied to test the MU/DEX. However, it is important to minimize the test length when one selects test patterns. The proposed testing procedures are as follows: At clock cycle 1, all the RCU's outputs are set to 1 and the queues' outputs to 0. Queue outputs are fixed at 1 for the rest of the procedures. At cycle 2, all the RCU's outputs are set to 0. During the remaining r cycles, the RCU performs permutation $i \rightarrow (i + j - 1)\ MOD\ r$ at cycle j, $3 \leq j \leq r + 2$. When the network uses distributed routing control, the queues for storing routing tags can be used to generate the desired routing requests to the RCU. By this permutation and the data queue setting, the r multiplexers in a MU/DEX are tested simultaneously. The testing procedures are shown in Fig. 3.11. ∎

The MU/DEX's output stream is two 0's followed by r 1's. Since both 0 and 1 appear at each switch's output, and thus, at each link, the links can be tested without introducing any additional cost. For test verification, it should be noted that all links in the network have identical outputs. Thus, the comparison method to verify the test results of queues can be applied similarly. Without loss of generality, the number of links from each switch is assumed to be even. Half of the links are connected to the primary inputs of a fan-out free XOR tree, and the rest are connected to the primary inputs of the other fan-out free XNOR tree. The outputs of the two fan-out free networks form a 1-out-of-2 codeword. A design example for this method is given in Fig. 3.12 (a).

It has been shown in [49] that a linear function implemented by two-input XOR gates needs at most four test patterns. The test patterns can be recursively derived from the primary output of the XOR(XNOR) tree to the primary inputs. Assume that a linear function P_n of n variables

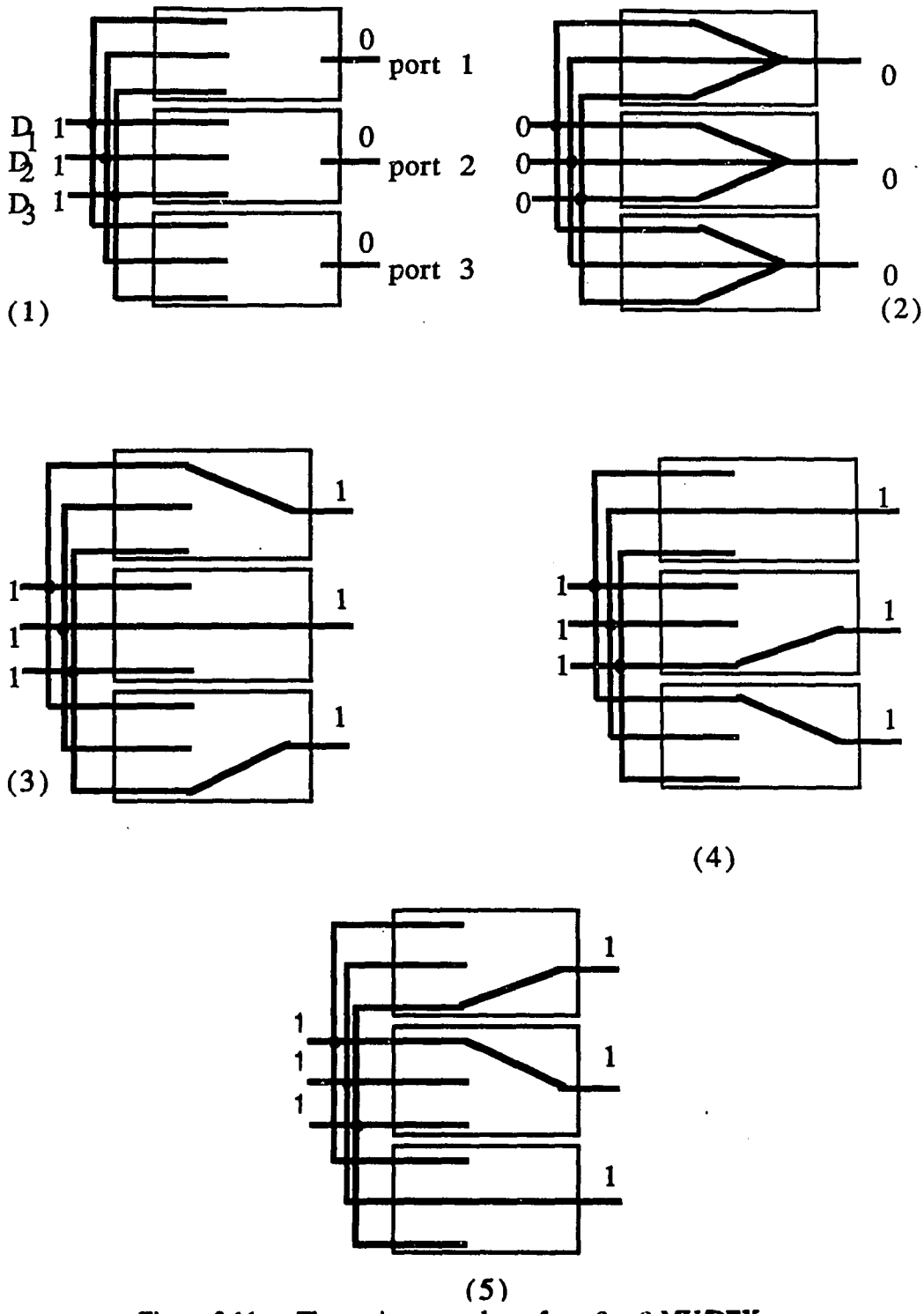


Figure 3.11: The testing procedures for a 3 x 3 MU/DEX.

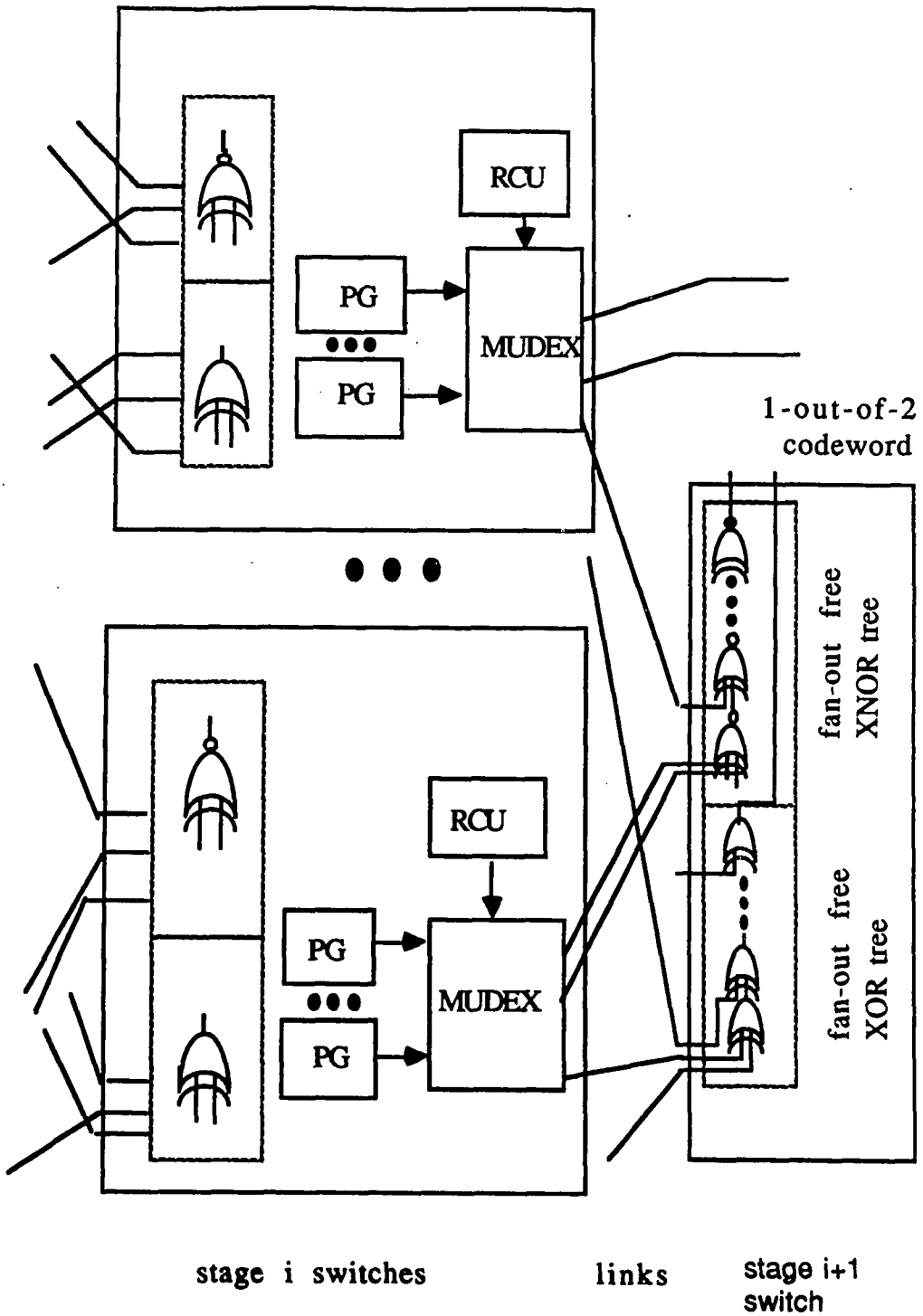


Figure 3.12: (a)Verification of testing response by comparison and signature analysis.

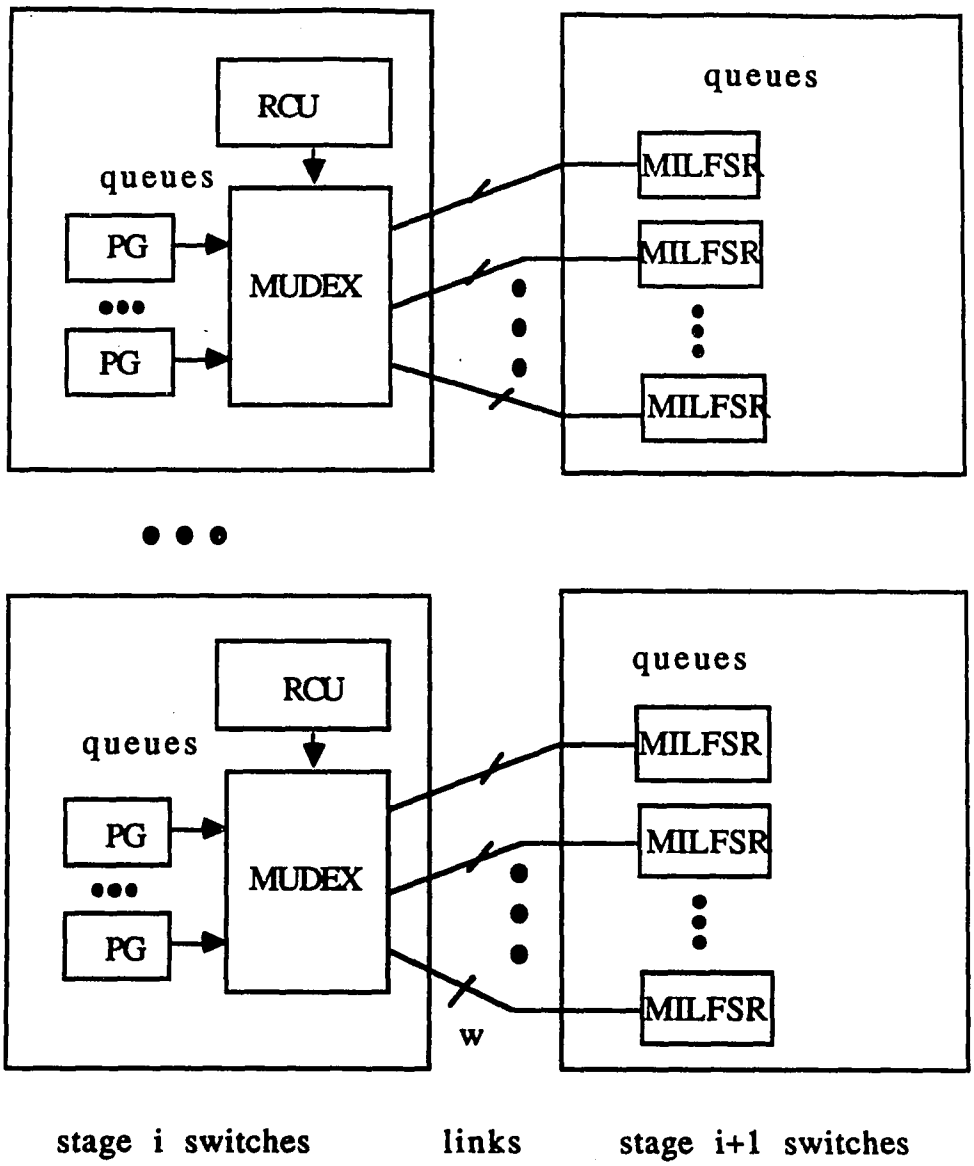


Figure 3.12(b) Verification of testing outputs with MILFSR's.

is implemented by an XOR tree as in Fig. 3.12(a). Then, P_n can be recursively expressed by $P_n = x_n \oplus P_{n-1}$, where x_n is the n-th primary input (variable), and P_{n-1} is the linear function implemented by the sub-network excluding the primary output XOR gate and the primary input x_n. To test the primary output gate, it is sufficient to have $P_{n-1}x_n = 00, 01, 10, 11$. The input stream in x_n is then 0101, and P_{n-1} should be 0011. We want to derive a test pattern which can be easily generated, e.g., all inputs are identical, or, only one or two inputs are different from others. Thus, for $P_{n-1} = 0011$ and $P_{n-1} = P_{n-2} \oplus x_{n-1}$, we set $x_{n-1} = 0101$ (as x_n), and thus, $P_{n-2} = 0110$. It can be shown by induction that $x_i = 0101, \forall i \neq 1$, and $x_1 = 0110$ (0011) when the number of gates is even (odd).

It is now clear that we can eliminate the hard core in the XOR and XNOR trees when their test patterns are applied. Assume that $\frac{w}{2}$ links are connected to inputs x_1 ... $x_{\frac{w}{2}}$ of the XOR tree. To test the XOR (XNOR) tree we need to add one more input x_0 to the tree, and x_0 is controlled by the BIT. Since the output stream of MU/DEX testing is composed of two 0's and r 1's, the XOR (XNOR) tree can be tested simultaneously with the MU/DEXes and links when $0011\cdots$ ($0110\cdots$) are simultaneously applied to x_0 by the BIT. It requires wr XOR and XNOR gates in each switch to verify the test response. When the number of XOR (XNOR) gates is too high, the testing method can be decomposed into two phases as follows. In phase one(two), all the queues in even(odd)-stage switches serve as pattern generators and those in odd(even)-stage switches serve as multiple input linear feedback shift registers (MILFSRs). To test MU/DEXes and links, the outputs of the MILFSRs are compared in a way similar to the case of testing queues. Thus, the network can be tested in two phases, each phase requiring $r + 2$ clock cycles. An example design showing such a strategy is given in Fig. 3.12(b).

3.5 Conclusion

A two level testing strategy is developed in this chapter. The high-level testing uses

processors as testers to test the network concurrently with normal network operations. On the other hand, in the low-level testing strategy, switches functions as testers to off-line test the network in a very short time period.

The first step to ease the high-level testing is to synchronize switches' operations. Then, the network is tested with different polynomial operations. Although only a few functional fault models and their corresponding testing procedures are discussed, the basic principle can be easily extended to detect other faults. For example, it can be used for *contention resolving* testing. When two or more requests in a switch competing for the same output port, the RCU should grant only one. Using the testing polynomials, $P_i^N(x)$ and conflicting routing requests, an $r \times r$ switch can be tested in $r^r - r!$ testing routines. Obviously, contention resolving testing is very expensive, when r is large.

Most of network faults can be diagnosed in a distributed manner by processors with the proposed testing methods. During each session of testing, a processor need not communicate with others except the simultaneous submission of polynomials. When the packet width is more than one bit, one of the data links can be used as a feed-forward line (but not as a feedback line). However, when a data line is to be transformed into a feedforward line, a multiplexer should be used to bypass the queue that the link is connected to. One masker, a mapper, and an adder of w bits are required for each data queue. For an $r \times r$ switch, at least r^2 logic gates are needed to implement the MU/DEX, and the overhead, relative to the combinational circuit of the switch to implement the tracer, is $\frac{2}{r}$. It should be noted that the overhead is an upper bound, because hardware for the data buffers are not considered.

The goal of the low-level testing is to obtain high fault coverage with a small testing time. Queues are self-tested first by converting them into polynomial generators. Furthermore, test patterns for the MU/DEX can also be generated by the PGs. For a C-connected queue, it needs w extra interlinks to convert a queue into PGs. It is very inefficient to test the RCU, and r

1-out-of-r codeword checkers are proposed to monitor the outputs of the RCU. Finally, the MU/DEXes and links are tested simultaneously. Testing responses are verified by using XOR and XNOR trees. It is shown that the network testing time is independent of the network size and the design method can be applied to various types of switch. For the low-level testing, comparators in the switches are the predominating overhead, which is $\frac{mr}{r^2} = \frac{1}{r}$ for the combinational circuits of switches, and no extra links are needed.

In this chapter we have focused only on the development of testing strategies. Although the high-level testing can test the network concurrently with normal network operations, a *congestion tree* (defined in the next chapter) might be formed if we lock up a path for a long time period. To prevent formation of congestion trees, which may cause excessive performance loss, an optimal testing period must be decided. The network dynamics under concurrent testing and the tradeoff between the performance penalty and expected fault-detection time are the subject to be analyzed in the next chapter.

Appendix 3.A: Fault Coverage of Polynomial Testing

Fault coverage of polynomial testing may vary with different circuit implementations of the network. To obtain a concrete figure on its fault coverage of the low-level testing method, consider an example LFSR design in Fig. 3.13. The basic structure of a shift-register is essentially a master-slave S/R flip-flop. The i-th gate in a flip-flop is denoted as G_i, and its output and j-th input (indexed from up to down) by GO_i and GI_i^j, respectively. It should be noted that latches usually have two outputs Q and $\bar{Q}$. However, since the number of links is a major concern in the network design, only the Q output of the slave-latch will be used, and $\bar{Q}$ is ignored. In other words, a fault is detectable only when an erroneous response can be observed at the Q output of the slave latch. During normal operations, the S and R inputs are in the form of $d\bar{d}$, 11, $d\bar{d}$, 11, ..., where $d \in$0,1. It should be noted that the input 11 is inserted automatically when CLK=0(1) at the master(slave)-latch, and the outputs of a latch are read out when SR=11. 01, 10, and 11 at the SR inputs are referred to as r, s, and b, respectively.

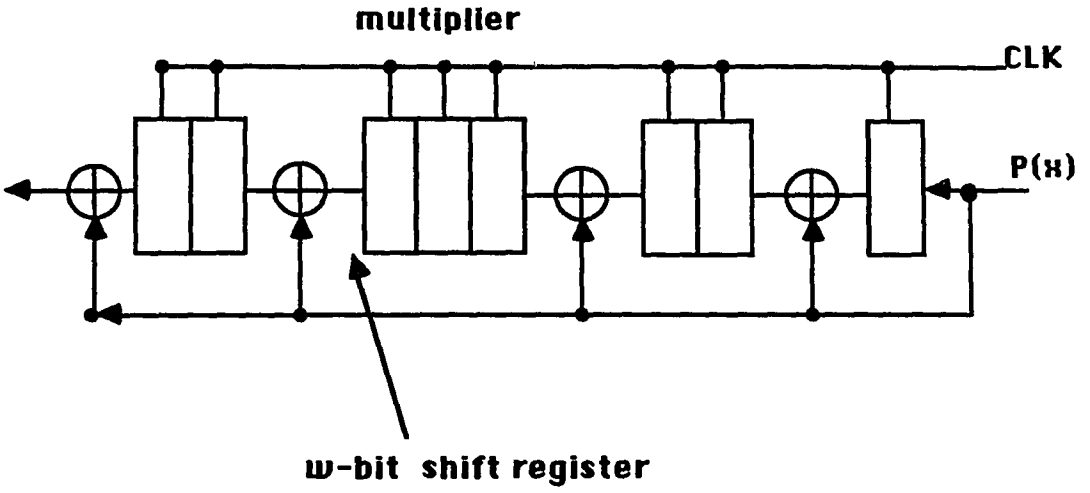
To derive its test set, the slave-latch composed of G_7, G_8 is considered first. Using the D-algorithm, test patterns for faults in the latch are summarized as follows.

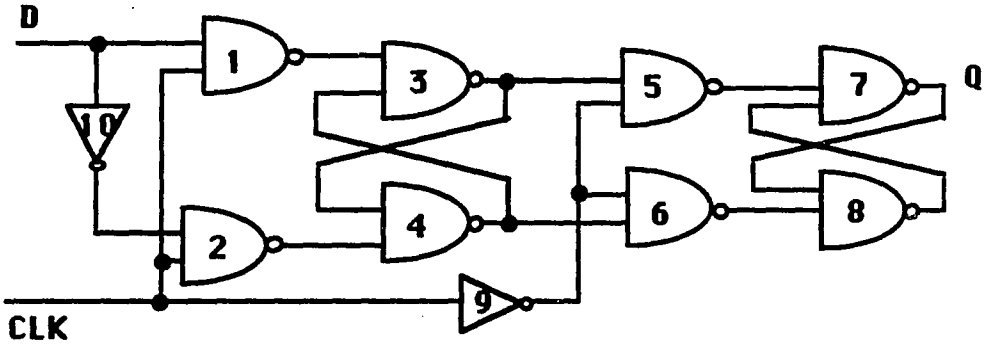
faults/nodes	GI_7^1	GI_7^2	GO_7	GI_8^1	GI_8^2	GO_8
s-a-0	sb	sb	rb	rb	rb	sb
s-a-1	sbrb	sbrb	sb	sb	rbsb	rb

Testing the master-latch is more complicated, because the slave-latch must be in an appropriate initial state, i.e., output, to propagate the erroneous response of the master-latch to the Q output of the slave-latch. For example, when the $(Q, \bar{Q}) = (0, \bar{D})$ at the master-latch, [4] the initial state of the slave-latch must be $(1, 0)$ to obtain D at the output of the slave-latch. Otherwise, the slave-latch's output is 01, meaning that the fault is not tested. In our case the master and slave latches have identical test patterns.

Test patterns for G_5 and G_6 are summarized below:

[4] They are inverted to $(1, D)$ before entering the slave-latch.





An one-bit shift register implemented by a master-slave S/R flip/flop

Figure 3.13: An example LFSR implemented with master-slave SR latches.

faults/nodes	GI_5^1	GI_5^2	GO_5	GI_6^1	GI_6^2	GO_6
s-a-0	sbrb	sbrb	sb	rbsb	rbsb	rb
s-a-1	rb	rbsb	sbrb	sbrb	sb	rbsb

It can be shown that the above pattern can test G_1, G_2 simultaneously. GI_{10} s-a-1 and GO_{10} s-a-0 can be tested by sbrb. The only undetectable faults are GI_{10} s-a-0 and GO_{10} s-a-1, because the erroneous responses can only propagate to the $\bar{Q}$ output of the slave-latch. GI_9 s-a-1 and GO_9 s-a-0 will block the transmission of data, and thus, sbrb is sufficient to test them. GI_9 s-a-0 and GO_9 s-a-1 faults do not cause logic faults, and thus, not detectable by the D-algorithm. However, the *memory* of the register is lost when the above two faults occur, i.e., the latch fails to hold the data for a specific period. Assume that the high (low) period t_T of testing clock is three times slower than the latch's transition time t_d. Then, such faults can be tested by the polynomial testing method. For example, when GO_9 is stuck-at-1, the data at the input of the register will be shifted to the slave-latch's output after $2t_d$. At the third t_d, the data is erroneously shifted into the next register. Thus, the shift register fails to perform the $delay$ function. Occurrence of such faults in a queue implies that the length of queue is reduced. Since the polynomial testing can identify the configuration of a LFSR, all such faults can be detected.

Although the polynomial testing is developed as a high level testing, it can clearly detect all the detectable stuck-at faults at the logic level of registers. Since only two of the 56 faults are undetectable in a register, and all other faults, i.e., stuck-at faults on the XOR gates and feedforward lines, are detectable, the lower bound of the polynomial testing method is 96%.

Appendix 3.B: List of Symbols

A,N,S,X	Four states of a switch on the route under test.
B_i	The i-th block in a PDM.
BU_i	The i-th buffer in a queue.
$D_{i(in)}$, $D_{i(out)}$	The ring link input (output) of the i-th pattern generator G_i.
DFF	A D type flip-flop with a single input and a single output.
E_m	Switch permutation, $E_m^{i+1} : (ir + j) \rightarrow f(ir + j + m \ MOD \ r)$, where $f(ir + j)$ and $f(ir + j + m \bmod r)$ are the global indices of the j-th input port and the $j + m \bmod r$-th output port of the $(i + 1)$-th switch, respectively, where r is the switch order, and — is the interconnection. E_m^i is denoted as E_m when all switches have the same permutation.
E_m^{-1}	The inverse of the switch permutation E_m.
E_{ij}	An RCU output which enables queue j to be connected to output port i.
$GF(2)$[x]	The polynomial ring.
I, I_n	I is the set of integers and $\mathbf{I}_n = \{0, 1, 2 \cdots, n\}$.

M	An $m \times w$ matrix representing m buffers in a queue of w bits each. The i-th row of M is the i-th bits of all buffers and the j-th column of M is the j-th buffer in the queue.
$M_{ij}(x)$	The multiplier formed by the route connecting source i to destination j.
MUDEX	An $r \times r$ MU/DEX is the combination of r multiplexers and r demultiplexers. It is used to direct packets in a PSMIN switch.
ONE, ZERO	Coefficients of a word polynomial, $ONE = \overline{\text{ZERO}}$.
$P(x)$, $\bar{P}(x)$	A polynomial and its complement.
$P_j^N(x)$	$P_j^N(x) = \sum_{i=0}^{N} c_i x^i$, where $c_j = 1$, $c_i = 0$, $i \neq j$, is a test pattern submitted by processor j.
$P_E^m(x)$	$P_E^m(x) = \sum_{i=0}^{k} x^{(m+1)i}$, the test polynomial for m unit delay faults.
PDM	Polynomial multiplier or divisor.
PG	Polynomial generator.
PSMIN	Packet switching multistage interconnection network.
$Q(x)$	The product of $P(x)$ and $M(x)$, or the quotient of $P(x)/D(x)$, or the output of a PDM.
$R(x)$	The remainder of the operation that $D(x)$ divides $P(x)$ or the final contents of a PDM.
RCU	Routing control unit.
RUT	Route under test.

r_i	The length or order of B_i.
SSA	Single stuck at fault.
T_i	The link permutation at stage i.
$W_k(x), W(x)$	$W_k(x) = \sum_{i=0}^{k} x^i$. $W(x) = W_\infty(x)$.
$\sum_{i=0}^{r} n_i x^i$	The initial state of a PDM or a PG.

CHAPTER 4

ANALYSIS AND OPTIMIZATION OF CONCURRENT NETWORK TESTING

4.1 Introduction

It is sometimes very expensive to suspend the operation of a switching network for the purpose of off-line fault diagnosis. For example, signal processing applications require large amounts of data to be transmitted through the network, and in telephone switching systems, the revenue loss caused by a system stoppage for fault diagnosis could be unacceptably high. Concurrent testing strategies, which test a part of the network at a time while letting the other parts function normally, can alleviate or remedy this problem.

As will become clear later, concurrent testing of a network is much more complex than that of memory or processors. When a processor or memory undergoes concurrent testing, no other system components need to be involved with the testing. Thus, it is relatively easy to estimate the performance loss caused by concurrent processor/memory testing. By contrast, testing the interconnection network of a multiprocessor system is complex and difficult because the network is shared by many processors, I/O devices and memory modules. Unless the network is equipped with a centralized control mechanism, unexpected packets may enter a path under test (PUT), making it impossible to evaluate testing outcomes correctly. Thus, for the purpose of concurrent network testing, special care must be taken to prevent unexpected packets from

entering the PUT. We proposed in the previous chapter a new method, called the high-level testing, to resolve this problem by synchronizing and locking up switches on the PUT. After a path is locked up, it can only accept testing packets from the testing processor, which is the only processor connected to the input of the PUT. Locking up switches prevents unexpected packets from entering the path, and synchronizing switch operations is necessary to reduce the testing complexity. Since it requires at most one additional processor to evaluate testing outcomes, the high-level testing minimizes interactions between processors, and thus, greatly simplifies the testing problem. Moreover, it has been shown in Chapter 3 that only a few modifications to existing switch designs are needed to support the high-level testing. However, we addressed in Chapter 3 only the architectural aspects of the concurrent high-level testing without any quantitative analysis of its performance. The performance analysis and optimization of the concurrent high-level testing are the subject of this chapter.

The concurrent high-level testing can be applied to networks with arbitrary topologies and switches, and/or redundant paths. An $r \times r$ switch in the network has r queues, each of which is served on a first–in–first–out (FIFO) basis. Packets that are submitted to the network by processors other than the testing processor and must pass through the locked path will be blocked. After a packet is blocked at a queue, other packets that must pass through the queue will also be blocked. Hence, when a path is locked for a long time, all of its source queues—those queues that have some paths to reach the locked path—will eventually be blocked. When all the source queues of a node are blocked, they form a *congestion tree*.

Every packet in a congestion tree cannot advance because packets in front of it are stuck. Performance loss may become substantial when a congestion tree is formed in the network for a long time. In order to avoid the formation of congestion trees and shorten the mean fault detection time, it is essential to analyze the network dynamics under concurrent testing. Congestion trees are somewhat similar to hot spots in common memory access [81, 22, 116].

However, packets stuck in a congestion tree cannot advance at all until the locked path is released, whereas packets in a hot spot zone move slowly without stopping.

As their name implies, concurrent testing procedures have to be executed according to a pre-planned schedule during the network's normal operation. Hence, to analyze the behavior of a network under concurrent testing, one has to know the network's steady-state performance. Numerous approximation methods have been proposed to model the performance of circuit and packet switching networks. Probabilistic models and their extensions are reported to yield a close approximation to the performance of circuit switching networks [77, 114, 12]. Markov chain models are usually used for the performance analysis of packet/circuit switching networks [34, 102, 23, 59, 60, 117]. Kruskal *et. al* derived a closed form solution for the asymptotic analysis of circuit switching networks, and packet switching networks with a large buffer size [57]. Due to the complexity involved, it is in general intractable to accurately analyze packet switching networks. We propose a *reduced network* model to ease this problem.

One can avoid the formation of congestion trees by controlling the time of testing a PUT. This implies that the probability distribution of time to block an arbitrary node be an important parameter for the analysis of a congestion tree. We develop several algorithms to derive this distribution. A congestion tree is said to be *dissipated* when all the stuck packets exit from the tree. The time to dissipate a congestion tree is another important performance parameter. Unfortunately, no simple analytic method appears to be derivable that can be used to calculate the probability distribution of tree dissipation time. Simulation is thus used to obtain the mean tree dissipation time, instead of attempting to derive the distribution of tree dissipation time.

The optimal fault coverage is derived to minimize performance loss and maximize testing efficiency. Finally, it is essential to avoid the repetition of testing the same network component; otherwise, the network testing time may become excessively long. We must therefore find the optimal *testing rings*, the testing sequence that minimizes the repetition of testing the same

component.

The rest of this chapter is organized as follows. Section 4.2 discusses network operations and the underlying testing method. The network dynamics under concurrent testing are analyzed in Section 4.3, and optimization of testing strategies is the subject of Section 4.4. This chapter concludes with Section 4.5.

4.2 Network Organization and Testing

Before analyzing the concurrent network testing, it is necessary to describe the basic network architecture and the underlying testing method.

4.2.1 Network Organization and Operations

The network under consideration is a packet switching multistage interconnection network (PSMIN) which connects N processors, $\{P_i,\ 0 \leq i \leq N-1\}$, to N memory modules, $\{M_i,\ 0 \leq i \leq N-1\}$, where memory module M_i is a partner of processor P_i, but is allowed to be accessed by any other processor in the system. When $r \times r$ switches are used in a blocking PSMIN with a unique path between every processor–memory pair, there are $k = \log_r N$ stages of switches in the network. Each switch has r queues and an arbiter, where the r queues are located at the r inputs of the arbiter. Packet transmissions are synchronous; that is, an arbiter grants/denies each request for routing a packet, and the packet, if granted, is routed from one stage to the next stage in one *network cycle*. Connection of the r outputs of each arbiter to the next stage is determined by the network topology. A switch (arbiter) located at the i-th row of stage j is denoted by $S_{i,j}$ ($AR_{i,j}$). $S_{i,j} \equiv \{Q_{(i-1)r,j}, Q_{(i-1)r+1,j}, \cdots, Q_{ir-1,j}, AR_{i,j}\}$, where $Q_{(i-1)r+\ell,j}$ is the ℓ-th queue of $S_{i,j}$, $0 \leq \ell \leq r-1$.

The network topology provides a unique path between every processor–memory pair. Interconnections between queues of adjacent stages can be described by *backward* (Γ) and

forward (Δ) relations. Physically, $\Gamma(Q_{j,i})$ ($\Delta(Q_{j,i})$) is the set of all immediate source (destination) queues of $Q_{j,i}$ in the previous (next) stage. For any $Q_{j,i}$, $\Gamma(Q_{j,i}) = \{ Q_{\ell_1,i-1}, Q_{\ell_2,i-1}, \cdots, Q_{\ell_r,i-1}\}$. $\Gamma^2(Q_{j,i}) \equiv \bigcup_{m=1}^{r} \Gamma(Q_{\ell_m,i-1})$, and $\Gamma^n(Q_{j,i}) \equiv \bigcup_{m=1}^{r} \Gamma^{n-1}(Q_{\ell_m,i-1})$. Similarly, $\Delta(Q_{j,i}) = \{ Q_{\ell_1,i+1}, Q_{\ell_2,i+1}, \cdots, Q_{\ell_r,i+1}\}$, and $\Delta^n(Q_{j,i}) \equiv \bigcup_{m=1}^{r} \Delta^{n-1}(Q_{\ell_m,i+1})$. Since a processor can access any memory module, $\Gamma^k(M_i) = \{P_j,\ 0 \leq j \leq N-1\}$, and $\Delta^k(P_i) = \{M_j,\ 0 \leq j \leq N-1\}$, $0 \leq i \leq N-1$. A ***destination tree*** with M_i as its root, denoted by TR_{M_i}, is formed by a collection of queues, $\bigcup_{j=1}^{k} \Gamma^j(M_i)$. A *source tree* with P_i as its root, denoted by TR_{P_i}, is formed by a set of queues, $\bigcup_{j=1}^{k} \Delta^j(P_i)$. Source or destination trees can be defined similarly with respect to queues instead of processors or memory modules. For a queue $Q_{\ell,n}$ at stage n, the notation $TP_{\ell,n}$ will be used to denote $TR_{Q_{\ell,n}} = \bigcup_{i=0}^{n} \Gamma^i(Q_{\ell,n})$ whose height is the same as the associated stage number, n. $TP_{\ell,n}$ will be written as TP_n whenever the value of ℓ becomes immaterial in our discussion.

Each tree $TP_{\ell_m,m}$ in the PSMIN can be modeled by a labeled graph G, in which nodes denote arbiters and edges are queues and links. Fig. 4.1 shows an example destination tree and its corresponding graph model. Having introduced the network organization, we now briefly discuss the operational principle of the network-level testing.

4.2.2 High-Level Testing

The network-level testing requires locking up only a small number of network components and processors for each testing session. The lock-up procedure is executed by routing *locking packet*(s) through a path to be tested. Note that the polynomial testing method can detect/locate the lock-up faults caused by mis-routing. Once a path is locked up, it will accept packets only from the testing processor, which is connected to the input of the locked path.

Stuck-at faults in a path $H_{i,j}$ can be located by three test patterns: the impulse, the all 1's and the all 0's streams. If no fault is found during the application of these patterns, the set of

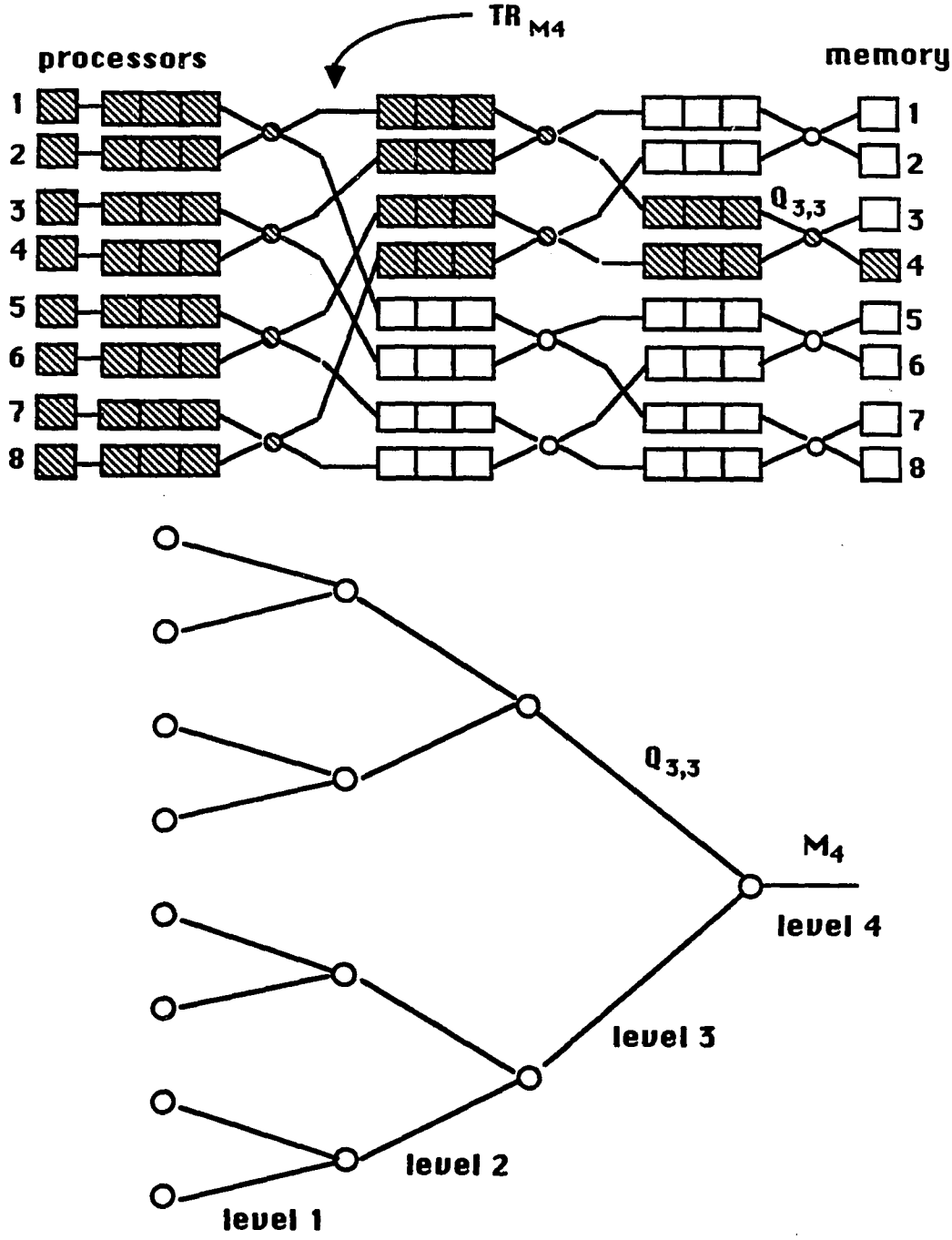


Figure 4.1: An example destination tree and its corresponding graph model for an 8×8 PSMIN.

switches $\{S_{m_1,1}, S_{m_2,2}, \cdots, S_{m_k,k}\}$ may be tested further for other faults, where $S_{m_\ell,\ell}$ is the ℓ-th switch on $\Pi_{i,j}$. For example, to test for merging and broadcasting faults in switch $S_{i,j}$, two paths intersecting at $S_{i,j}$ must be simultaneously locked up. Thus, any two processors P_ℓ and P_m such that $P_\ell, P_m \in \Gamma^{ij}(S_{i,j})$ can function as testers of $S_{i,j}$. To test $S_{i,j}$, two test patterns, T_ℓ and T_m, must be submitted simultaneously from P_ℓ and P_m, respectively. As shown in Chapter 3, these test patterns can be represented as $T_\ell = \sum_{\ell=0}^{wk-1} a_\ell X^\ell$ and $T_m = \sum_{\ell=0}^{wk-1} b_\ell X^\ell$, where $a_i = \bar{b}_i \ \ \forall i$, w is the number of buffers in each queue, k the number of stages, and X a dummy variable.

When the locking packet is routed through a queue, and thus a switch, we can lock up either the queue or the switch. When the queue is selected to be locked up, two intersecting paths can be locked up independently, because they run through different output ports of the switch where they intersect. If the entire switch, instead of a queue, is selected, two paths' lock-up procedures must be synchronized to avoid the possibility of deadlock. A deadlock occurs when two paths' lock-up speed is different, where the slower lock-up operation can never be completed. To avoid deadlock, we must either synchronize the routing of the two locking packets, or must design switches so that they can be locked up only after two locking packets enter a switch. The first type of lock-up operation is much simpler than the second type, and thus, will be used in the rest of this paper.

As mentioned earlier, concurrent testing is performed according to a predefined schedule. Further, the system is assumed to be in steady-state when a testing session starts. We shall develop a network model in the next subsection to facilitate the analysis and simulation of steady-state system performance before delving into the analysis of the network behavior under concurrent testing.

4.2.3 Probability Distribution of Queue Lengths

A PSMIN can be described by an Nk–tuple, $\Theta = (Y_{0,1}, Y_{1,1}, \cdots, Y_{N-1,k})$, where $Y_{i,j}$ is a random variable (or state) denoting the number of packets in $Q_{i,j}$. The steady-state joint probability distribution of Θ can be obtained by solving the balance equation $\mathbf{v} = \mathbf{P}\mathbf{v}$, where $\mathbf{v}$ is a vector representing the steady-state distributions of $Y_{i,j}$'s and $\mathbf{P}$ the state transition matrix. Since the number of system states increases exponentially with the network size, i.e., $(w+1)^{Nk}$, it is necessary to reduce the number of states in order to make system analysis/simulation manageable. A *reduced network* model is proposed to meet this need.

Processors' request patterns are assumed to be independent and identically distributed (i.i.d.), and as a result, address bits generated by processors are also i.i.d. It is further assumed that arbiters grant routing requests based on a geometric distribution [77, 57], and at stage i of PSMIN, the parameter of the geometric distribution is $P_{out,i}$. When all arbiters' output ports have the same routing priority, the packets' interarrival times at arbiters' output ports are i.i.d., and the number of packets waiting in each of the queues at one stage is i.i.d. That is, queues within a stage are *statistically identical*, and so are arbiters. It makes no difference which buffer of the previous (next) stage a packet is coming from (going to). Consequently, for the purpose of steady-state performance analysis, one can assume, without loss of generality, that all the output links of a switch are logically connected to the input ports of another switch. (Note that these output links are actually connected to the input ports of several different switches.) This simplification will reduce the original PSMIN to N/r independent and identical subnetworks. The reduced network model deals with each of these subnetworks (i) which consists of k stages of $r \times r$ switches connecting r processors to r memory modules, and (ii) in which link connections between adjacent stages are represented by the identity permutation.

The reduced network model can be simplified further as follows. Since queue lengths in a

switch are i.i.d., one random variable is sufficient to represent the number of packets in each stage. The reduced network model can thus be characterized by a k-tuple $\Theta' = (Y_1, Y_2 \cdots Y_k)$, where Y_i is the random variable denoting the number of packets in any one of the queues at stage i. Let the probability of $Y_i = \ell$ be $P_{i,\ell}$, and w be the number of buffers in each queue. Then, the number of states of each queue is $w + 1$, where the state of a queue is defined by the number of packets in the queue. Note that Y_i's are often assumed to be independent as in [34]. However, this assumption is acceptable only when the network has light traffic, or w is very large.

Elements of the transition matrix **P** are determined by the operational principles of arbiters, and the number of buffers in each queue. For a non-full queue at stage i, the probability of a packet entering the queue is $P_{in,i} = 1 - \left(1 - \frac{1-P_{0,i-1}}{r}\right)^r$. The packet at the head of a non-empty queue can move to the next stage only when the next destination switch is not full, and the packet's routing request is granted by the associated arbiter. Assuming that packet transmissions in the network are synchronized, at any network cycle we get

$$\begin{aligned} P_{out,i} &= (1 - P_{w,i+1}) \sum_{j=0}^{r-1} \binom{r-1}{j} (1 - P_{0,i})^j P_{0,i}^{r-1-j} r \left(\sum_{k=0}^{j} \binom{j}{k} \left(\frac{1}{r}\right)^k \left(\frac{r-1}{r}\right)^{j-k} \frac{1}{k+1} \right) \\ &= r\frac{1-P_{w,i+1}}{1-P_{0,i}} \left(1 - (1 - \tfrac{1-P_{0,i}}{r})^r\right) = r\frac{1 - P_{w,i+1}}{1 - P_{0,i}} P_{in,i}. \end{aligned}$$

For a queue at stage i we have

$$\begin{aligned} P_{j,i} &= P_{in,i}(1 - P_{out,i})P_{j-1,i} + P_{out,i}(1 - P_{in,i})P_{j+1,i} + \\ &\quad (P_{in,i}P_{out,i} + (1 - P_{in,i})(1 - P_{out,i}))P_{j,i}, \quad \text{if } 0 < j < w \\ P_{w,i} &= P_{in,i}(1 - P_{out,i})P_{w-1,i} + (1 - P_{out,i})P_{w,i} \\ P_{0,i} &= P_{out,i}(1 - P_{in,i})P_{1,i} + (1 - P_{in,i})P_{0,i}. \end{aligned}$$

Let $\tau_i = \frac{P_{in,i}(1-P_{out,i})}{P_{out,i}(1-P_{in,i})}$, then $P_{j,i} = \tau_i^{j-1} P_{1,i}$, $1 \le j \le w-1$, $P_{0,i} = \frac{1}{\tau_i}(1 - P_{out,i})P_{1,i}$, and $P_{w,i} = \tau_i(1 - P_{in,i})P_{w-1,i}$. To find the solutions for $P_{i,j}$'s, one must solve the normalization

equation $\sum_{i=1}^{k}\sum_{j=0}^{w} P_{i,j} = 1$, that is,

$$\sum_{i=1}^{k}\left(1+\frac{1-\tau_i^{-(w+1)}}{(1-P_{in,i})(1-\frac{1}{\tau_i})}-\frac{P_{out,i}}{\tau_i^{w}(1-P_{in,i})}\right)P_{w,i}=1.$$

Let $P_{w,i} = \tau_i P_{w-1,i}$ and $P_{0,i} = \frac{1}{\tau_i} P_{1,i}$. Then, the normalization equation is simplified to $\sum_{i=1}^{k} \frac{1-\tau_i^{w+1}}{1-\tau_i} P_{0,i} = 1$. Our simulation results in the next section show that when either the network has light traffic or w is very large, arbiters at different stages have nearly identical parameter values, i.e., $\tau_i \approx \tau_j, \forall i,j$. However, when each queue has only a small number of buffers or the network has heavy traffic, dependence among first few stages cannot be ignored. The solutions of $P_{i,j}$'s appear to be intractable under such a condition, because τ_i's are completely different from one another, and the required solution procedure is extremely tedious. Consequently, we use the reduced network model in our simulation to derive steady-state network performance. Once the steady-state performance is determined,[1] we can analyze the network behavior under concurrent testing.

4.3 Network Behavior Under Concurrent Testing

There are several interesting analysis and optimization problems associated with the concurrent network-level testing. We shall investigate such important performance parameters as path locking time, and times to form and dissipate a congestion tree in the PSMIN.

4.3.1 Time to Form a Congestion Tree

The most important network dynamic parameter under concurrent testing is the probability distribution of time to form a congestion tree. However, it is very complex and difficult to derive this distribution due mainly to the excessive number of different locations where packets may reside at. Hence, the analysis of time to form a congestion tree is decomposed

[1] These results will be shown in the next section together with other simulation results.

into the following four steps. First, components of a congestion tree are modeled. Second, the distribution of time to block the root of an arbitrary tree is derived. The third step is to reduce the computational complexity associated with the derivation of the distribution of root blocking time. Finally, the probability distribution of time to block an arbitrary node is derived using the distributions of root blocking times. The probability distribution of time to form a congestion tree can then be derived from the distributions of node and root blocking times. Each of these four steps is discussed below in detail.

A. Modeling of a Congestion Tree

A locked path $\Pi_{i,j}$ is composed of a set of queues $\{Q_{\ell_n,n}\}_{n=1}^{k}$, where $Q_{\ell_n,n}$ is the locked queue at stage r on the path. As shown in Fig. 4.2, there are k disjoint subtrees $\{TP_j\}_{j=1}^{k}$ within $TR_{M_j} - \{\Pi_{i,j}\}$, where $Q_{\ell_n,n} \in \Pi_{i,j}$, $1 \leq n \leq k$, is the root of TP_n. Each such TP_n is called a *blocking tree*, because the root of TP_n is locked up, and thus, packets may build up in this tree. A blocking tree becomes a congestion tree when all the queues in the tree are filled with packets that cannot advance.

TP_n's associated with $\Pi_{i,j}$ are disjoint, i.e., $TP_\ell \bigcap TP_m = \emptyset$, $\forall \ell \neq m$, $1 \leq \ell, m \leq k$, because $\forall\ \ell < m$, (1) there is only one path from a queue in TP_ℓ to another queue in TP_m via the locked path $\Pi_{i,j}$, and (2) there is no path from TP_m to TP_ℓ. When $\Pi_{i,j}$ is locked up for testing, the testing processor and the output of $\Pi_{i,j}$ are P_i and M_j, respectively. Except for the $\Pi_{i,j}$'s queue in the first stage of the network, M_j and all other queues on $\Pi_{i,j}$ are shared between P_i and other processors. Since the queues in a locked path $\Pi_{i,j}$ can receive packets only from its testing processor P_i, all packets that are generated by processors other than P_i and need to use $\Pi_{i,j}$ and M_j will be blocked until the testing of $\Pi_{i,j}$ is completed. Those packets that need to enter a queue Q_{i_m,j_m} on $\Pi_{i,j}$ are called *blocking packets* of Q_{i_m,j_m} or $\Pi_{i,j}$. Once a packet gets stuck at a source queue $Q_{x,y}$ on $\Pi_{i,j}$, all subsequent packets entering

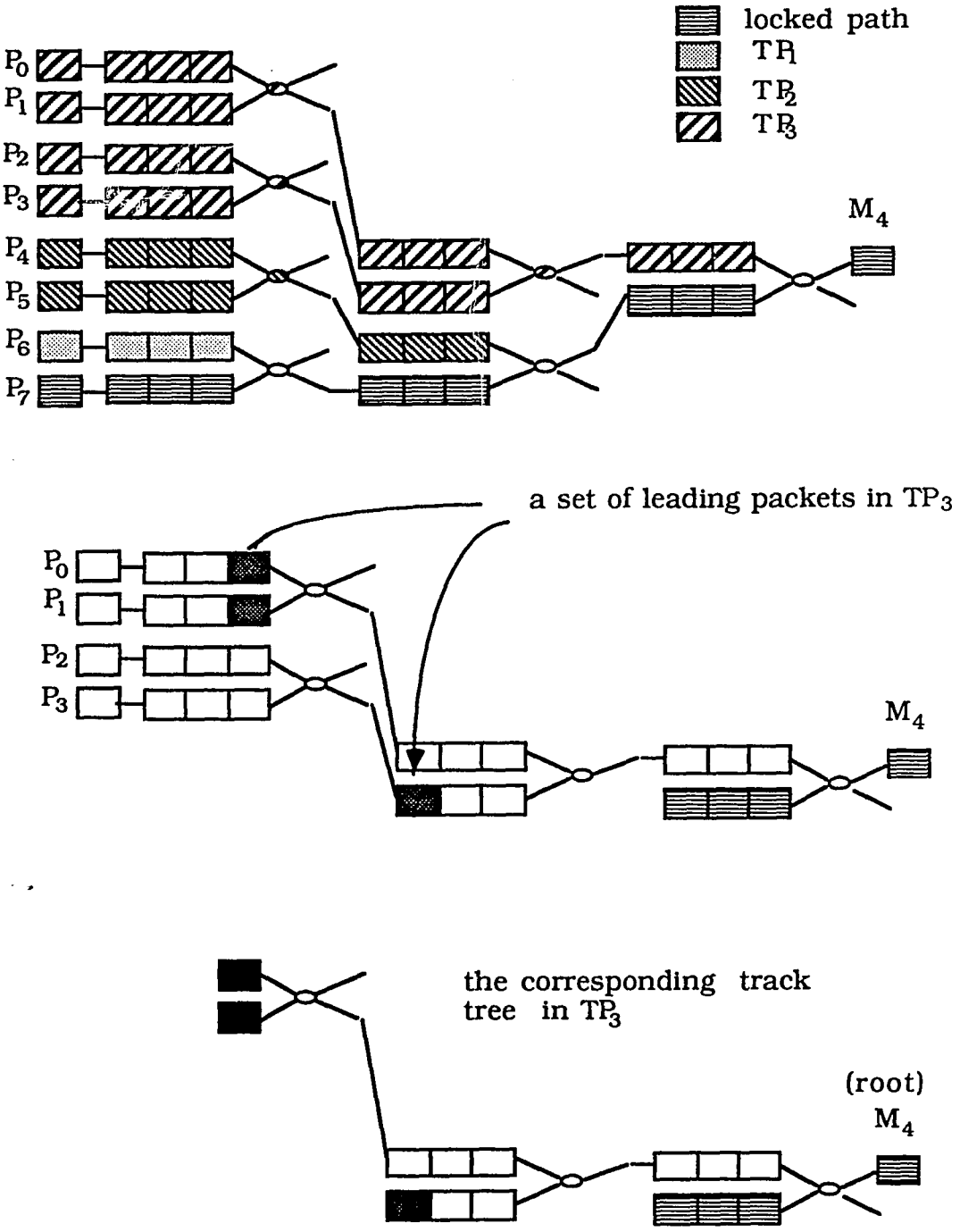


Figure 4.2: Disjoint subtrees in TR_{M_4}.

$Q_{x,y}$ will be stuck, and thus become the blocking packets of $Q_{x,y}$. Such a $Q_{x,y}$ (or $H_{i,j}$) will not be mentioned specifically when we refer to "blocking/leading packets" (to be described later) as long as it does not cause any ambiguity.

Those switches having no path to $H_{i,j}$ are called *free switches*.[2] Free switches can function normally during the testing of $H_{i,j}$. However, all free switches will eventually empty their queues for the following reason. Since TR_{M_j} contains all processors as its leaves, and all the queues in TR_{M_j} will eventually be blocked, no packet can be submitted to the free switches after the tree is completely blocked. Consequently, free switches do not have any impact on the network performance of our interest, and thus, will not be discussed any further.

A blocking packet is called a *leading* packet if there is no other blocking packet ahead of it. By definition, there must be no path available among the queues holding leading packets; otherwise, one leading packet would be ahead of another. Each set of disjoint subtrees within a blocking tree can be made to correspond to a set of leading packets by placing each leading packet at the root of one of the disjoint subtrees.

One can observe that in some packet switching networks such as telephone systems, packets are generated independently, and enter the network constantly. Based on this observation, every processor in the PSMIN is assumed to have the capability of generating an infinite number of packets over an infinite time period.[3] Those packets not generated yet in a blocking tree may in future block the root of a subtree just as already-existing packets may. Thus, to derive the probability distribution of a root's blocking time, we have to consider both blocking packets already in the network as well as those to be generated in future. Arrivals of packets are assumed to follow a geometric distribution with parameter p. The probability of a new blocking packet to be generated by a processor in tree TP_{i_r,j_r} at the n-th network clock cycle is

[2] However, there could be paths from $H_{i,j}$ to free switches.

[3] If processors do not have this capability, the required analysis will be much simpler than the one presented here.

$\sum_{i=0}^{n} \binom{n}{i} (1-p)^{n-i} p^i (1-(\frac{1}{r})^{jr})^{i-1} (\frac{1}{r})^{jr}$. Moreover, the probability of the i-th blocking packet to be generated by the processor at the n-th network clock cycle is $\binom{n-1}{i-1}(1-pr^{-jr})^{n-1-i}(\frac{p}{r})^i$, where $n \geq i$.

For a set of n leading packets $\{PK_i\}_{i=1}^{n}$ located at n different queues of a blocking tree TP_x, the set of leading packets' paths to the root of TP_x can be viewed as a *track tree*. The track tree for $PK_1, PK_2, \cdots, PK_n$ is simply $TP_x - \bigcup_{i=1}^{n} TP_i$, where TP_i is the tree whose root is the queue that holds PK_i. Two track trees are distinct when their corresponding sets of leading packets are different. Fig. 4.3 shows the sets of track trees in TP_1, TP_2 and TP_3, with $r = 2$. We shall henceforth use the terms "a set of leading packets", "a set of disjoint subtrees", and "a track tree", synonymously. Let D_b denote the set of distinct track trees in TP_b. When $Q_{i_m,m} \in H_{i,j}$ is locked up, a set of leading packets, $A = \{PK_1, PK_2, \cdots, PK_n\} \in D_m$, will have a race in the track tree to reach the root, $Q_{i_m,m}$, of TP_m. The number of leading packets decreases monotonically as the race progresses. Once a leading packet reaches the root, it will be the only leading packet in the tree.

Starting from the root of a k-level tree, each level of the tree is indexed as $k, k-1, \cdots, 1$. The *lowest common node* (LCN) of PK_i and PK_j, denoted by $LCN(PK_i, PK_j)$, is the node at the lowest level that can be reached by both PK_i and PK_j. Thus, $LCN(PK_i, PK_j)$ is the node at which one of the two packets will lead the other. Similarly, $LCN(PK_1, \cdots, PK_n)$ is the LCN of n leading packets, $\{PK_1, \cdots, PK_n\}$. In a track tree, every node having more than one child is the LCN of some leading packets.

Given any non-complete tree A, one can find another track tree A' with the same probability distribution of root blocking time. A' can be found by exchanging locations of two hertergeneous subtrees (i.e., subtrees with different structures) under a LCN. That is, given a set of n track trees $A_{CT} = \{A_\ell \mid \ell = 1, \cdots, n,\ F_{X_1|A_i} = F_{X_1|A_j},\ i \neq j,\ 1 \leq i, j \leq n\}$ where

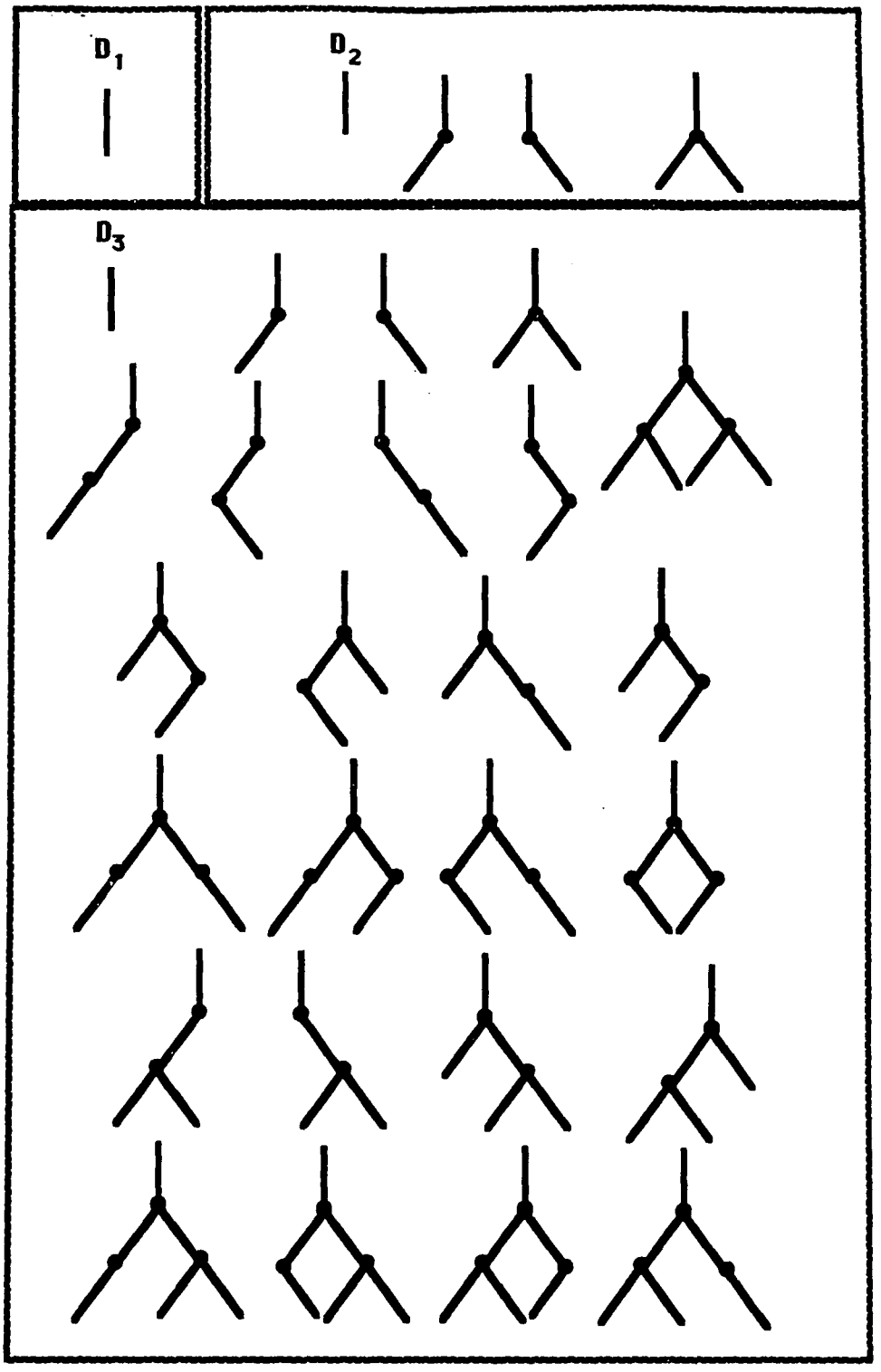


Figure 4.3: The set of track trees in TP_1, TP_2, and TP_3, respectively.

X_1 is the root's blocking time, we get $F_{X_1|A_{CT}} = |A_{CT}|\ F_{X_1|A_i}$, $\forall A_i \in A_{CT}$. Fig. 4.4(a) shows an example set of track trees A_1, A_2, A_3, A_4 that have an identical distribution of root blocking time.

Lexicographical ordering of trees in A_{CT} is used to uniquely identify each of the trees in A_{CT}. There are several ways to define the lexicographical ordering of trees in A_{CT}. However, any meaningful lexicographical ordering should be able to distinguish any two different trees by their lexicographical orders.

Consider an example game to determine the lexicographical order of two track trees, A_i and A_j in A_{CT}. Before the game starts, A_i and A_j are changed to two complete trees[4] T_i and T_j by adding new nodes and branches to A_i and A_j, and each newly added branch/node is labeled as a "fake" branch/node (Fig. 4.4(a)). Let two knights (packets) K_i and K_j traverse nodes of T_i and T_j in preorder, respectively. K_i (K_j) can always challenge K_j (K_i) as long as K_i (K_j) is not at a fake node, and wins the game if K_j (K_i) cannot challenge back (because it is at a fake node). Challenges will be dismissed and the game will continue if K_i and K_j challenge each other at the same time. The game is ended when one of the two knights wins, or there are no more nodes to visit. If K_i loses the game to K_j, A_i is said to be *lexicographically smaller* than A_j, denoted by $A_i \prec A_j$. $A_i = A_j$ if the two knights have a tie at the end of the game. This example lexicographical ordering can always distinguish two different trees, and $A_i = A_j$ only when they have an identical structure. An example set of track trees with their lexicographical orders are shown in Fig. 4.4(a).

Since the lexicographical ordering can distinguish all trees in A_{CT}, it can also be used to distinguish arbitrary trees. However, $T_i \prec T_j$ does not indicate which tree has more branches or nodes. An arbitrary tree $A_i \in A_{CT}$ is said to be *lexicographically maximal* if $A_i \succeq A_j$, $\forall A_j \in A_{CT}$. A tree T_m is lexicographically maximal if (1) all its subtrees

[4] In a complete tree, all but leaf nodes have r children.

are lexicographically maximal, and (2) at an arbitrary node Q of T_m, subtrees below Q are arranged in lexicographically descending order from left to right. Using these properties, an arbitrary tree can be converted to its lexicographically maximal form (called the *canonical tree* CT) by the following simple recursive routine.

```
lex_max(TP_{j,n})
/*TP_{j,n} is the input tree of height n to be converted*/
begin
    if(n > 1) then
        begin
            i = 1;
            while(i ≤ r) do
                lex_max(TP_{j_i,n-1}); /*go down one more level*/
                i=i+1;
                end_do
        end
    sort TP_{j1,n-1}, ..., TP_{jr,n-1} in lexicographical order;
end
```

B. Distribution of Root Blocking Time

We now derive the probability distribution of time to block the root of a blocking tree. Let T_i, $1 \leq i \leq m$, be the time required to route a leading packet PK_i to the root of a tree, and X_i be the time the i-th blocking packet arrives at the root, i.e., $X_1 < X_2 \cdots < X_w$, and $X_1 = \min(T_1, T_2, \cdots, T_m)$ is the root's blocking time. $(X_1, X_2 \cdots, X_w)$ is called the set of *order statistics*, representing the arrival times of the first w leading packets. The probability distribution of X_1, denoted by F_{X_1}, can be derived by enumerating all possible locations of leading packets, and then summing up the probability of the root to be blocked by each leading packet.

Times to route leading packets to the root of a track tree are not always independent because they must traverse some common queues and arbiters before reaching the root. However, two leading packets PK_i and PK_j in a same track tree have independent routing times before one of them reaches $LCN(PK_i,PK_j)$. Let Q_{i_r,j_r} be the root of the tree under consideration, and the path between PK_n and the root contain queues $Q_{i_n,j_n},\cdots,Q_{i_r,j_r}$. For a set of leading packets $PK_1, PK_2,\cdots,PK_m$ be located at $Q_{i_1,j_1}, Q_{i_2,j_2},\cdots,Q_{i_m,j_m}$, respectively, it takes time T_n for PK_n to travel from Q_{i_n,j_n} to Q_{i_r,j_r}. T_n can be expressed as the summation of times for PK_n to traverse all the queues and arbiters between Q_{i_n,j_n} and Q_{i_r,j_r}, i.e., $T_n = T_{i_r,j_r} + \cdots + T_{i_n,j_n}$, where T_{i_m,j_m}, $m = r,\cdots,m$, is the time for a packet to route through Q_{i_m,j_m}. Then,

$$\begin{aligned} X_1 &= \min(T_{i_r,j_r} + \cdots + T_{i_1,j_1},\ T_{i_r,j_r} + \cdots + T_{i_2,j_2},\cdots,\ T_{i_r,j_r} + \cdots + T_{i_m,j_m}) \\ &= T_{i_r,j_r} + \min(I_1,\ I_2,\cdots,I_m) \end{aligned} \tag{4.1}$$

where $I_\ell = \cdots + T_{i_\ell,j_\ell}$, $1 \le \ell \le m$. Since T_{i_r,j_r} is independent of $\min(I_1,\cdots,I_m)$, we get $f_{X_1} = f_{T_{i_r,j_r}} * f_{\min(I_1,\cdots,I_m)}$. Furthermore, for m independent random variables X_j's, $1 \le j \le m$, $F_{\min(X_1,X_2,\cdots,X_m)} = 1 - \prod_{i=1}^{m}(1 - F_{X_i})$. Thus, if I_i's in Eq. (4.1) are not independent, they should be combined into groups $G_1, G_2,\cdots,G_\ell$, so that I_i and I_j will be in the same group if and only if there is at least one common variable between I_i and I_j. The probability distribution of each group G_i is evaluated separately, and then combined as $F_{\min(I_1,I_2,\cdots,I_m)} = 1 - \prod_{i=1}^{\ell}(1 - F_{G_i})$. As was done in Eq. (4.1), the probability distribution of each F_{G_i} can be calculated by extracting common variables recursively.

Having calculated $F_{X_1|A}$, one can compute $F_{X_1} = \frac{1}{|D_b|}\sum_{A\in D_b} F_{X_1|A}$, where $|D_b|$ denotes the number of ways that leading packets may reside in different queues, and is derived by the following theorem.

Theorem 1 : When every queue is composed of only one buffer, the number of sets of disjoint subtrees (distinct track trees) in a blocking tree TP_b, $b > 1$, is

$$\begin{aligned} |D_b| &= (1 + |D_{b-1}|)^r \\ &= (1 + \cdots + (1 + (2^{r-1} \underbrace{)^r)^r \cdots)^r}_{b-1} \end{aligned}$$

Proof: $|D_b|$ is derived by induction. When $b = 1$, there are $r - 1$ unlocked queues in TP_1, and each queue may or may not have a leading packet in it. However, the case that none of the branches has leading packets must be excluded, because processors can generate infinite number of packets. Thus, $|D_1| = 2^{r-1} - 1$. Suppose TP_{b-1} has $|D_{b-1}|$ track trees. Since TP_b is created by linking r copies of TP_{b-1} to a new root, D_b can be generated from r copies of D_{b-1}. None or one of the elements of D_{b-1} can be assigned to a child of the root of TP_b. Thus, there are $(1 + D_{b-1})$ combinations of different elements in each child of the root, and $|D_b| = (1 + |D_{b-1})^r$. ■

Corollary 1 : When each queue is composed of $w > 1$ buffers, the number of sets of disjoint subtrees in TP_b is

$$\begin{aligned} |D_1| &= (w + 1)^{r-1} - 1 \\ |D_b| &= (w + |D_{b-1}|)^r, \quad b > 1 \end{aligned}$$

Proof: A queue is either full or empty if it has only one buffer. When $b = 1$, and each queue has w buffers, a queue may have none or one leading packet located in one of its w buffers. Thus, $|D_1| = (w + 1)^{r-1} - 1$. Assuming that TP_{b-1} has $|D_{b-1}|$ track trees, TP_b is created by linking r copies of TP_{b-1} to a new root. D_b can be generated from r copies of D_{b-1}. One element of D_{b-1} can be assigned to a child of the new root, or the leading packet may

be located in one of the w buffers in a new queue. Thus, there are $w + D_{b-1}$ combinations of different elements in each child of the new root, and $|D_b| = (w + |D_{b-1})^r$. ■

In enumerating all possible locations of leading packets, one only needs to consider which queues, instead of which buffers, leading packets may reside at, because the effects of buffer size have already been included in the distributions of (blocking/leading) packets' queue routing times. Thus, although Corollary 1 counts the number of track trees in TP_m's with multiple-buffer queues, one should use Theorem 1, instead of Corollary 1, to compute $|D_b|$.

C. Reduction of Computational Complexity

The number of disjoint subtrees in a blocking tree increases rapidly with its number of levels. Note that many cases have been redundantly counted in the calculation of the distribution of root blocking time. Let D^b_{CT} denote the set of canonical trees in TP_b, then $|D^b_{CT}| \ll |D_b|$ as will be shown below in Theorem 2. One needs much less effort to compute F_{X_1} with $F_{X_1} = \frac{1}{|D_b|} \sum_{CT_i \in D^b_{CT}} |CT_i| F_{X_1|CT_i}$ than directly computing $F_{X_1} = \frac{1}{|D_b|} \sum_{A \in D_b} F_{X_1|A}$.

Theorem 2 : When each queue is composed of one buffer, the number of canonical trees in TP_b of height b is

$$|D^b_{CT}| = \binom{|D^{b-1}_{CT}| + r}{r} \tag{4.2}$$

where $D^1_{CT} = \binom{r+1}{r}$, and each node (except for those nodes located at the lowest level) has r branches to the next lower level.

Proof: The theorem is proven by induction. TP_1 has only one canonical tree. TP_2 can be generated by linking the roots of TP_1's to a new root (with new links). Leading packets in TP_2 may be located at its root, or at i of its r leaves, where $1 \le i \le r$. Thus, there are $r + 1$ canonical trees in TP_2. Assuming that there are $|D^i_{CT}|$ canonical trees in TP_i, we now

build TP_{i+1} from r TP_i's. Since TP_{i+1} contains r TP_i's, and each subtree has $|D^i_{CT}| + 1$ canonical trees, $|D^{i+1}_{CT}| = \binom{|D^i_{CT}|+r}{r}$ follows. ∎

Table 4.1 shows the number of track trees and canonical trees for $b = 1$ to $b = 5$ when $r = 2$. As mentioned earlier, the number of canonical trees in a tree is the same as that of different forms of probability distributions in the tree.

b	1	2	3	4	5	6	7
$\|D_b\|$	1	4	25	676	458329	$\approx 2.1 \times 10^{11}$	$\approx 4.4 \times 10^{22}$
$\|D^b_{CT}\|$	1	3	10	66	2278	$\approx 2.6 \times 10^{6}$	$\approx 3.36 \times 10^{12}$

Table 4.1: The computational complexity of D_b and D^b_{CT}.

Canonical trees can be uniquely encoded by a data structure called a *mark*. The encoding sequence is obtained by assigning a level number to each branch of a tree, and branch level numbers are put into the mark when the tree is traversed in preorder. To decode a mark $i_1 i_2 i_3 \cdots i_n$ to its corresponding canonical tree, a cursor is used to point at the current digit of the mark, and a *branch generator* to generate branches according to the current and previous digits. At the beginning, the cursor is located at the leftmost digit of the mark, and the branch generator points at the root of the tree. The new branch is labeled with i_1, the branch generator is moved to the bottom of the branch, and the cursor is shifted to i_2. The rest of the procedure can be expressed by the following steps. At i_j of the mark, $1 < j < n$, we either add a branch at the current node if $i_j < i_{j-1}$, or move the branch generator to the top of the current branch and then add a new branch there if $i_j = i_{j-1}$, or trace back to the top of the first branch, b_x, which has the same label as i_j, and a new branch is added at the top of b_x, if $i_j > i_{j-1}$. Then, the cursor (branch generator) is moved to the next digit (bottom of the new branch), and the added branch is labeled with i_j. Fig. 4.4(b) shows the decoding sequence of mark 321121.

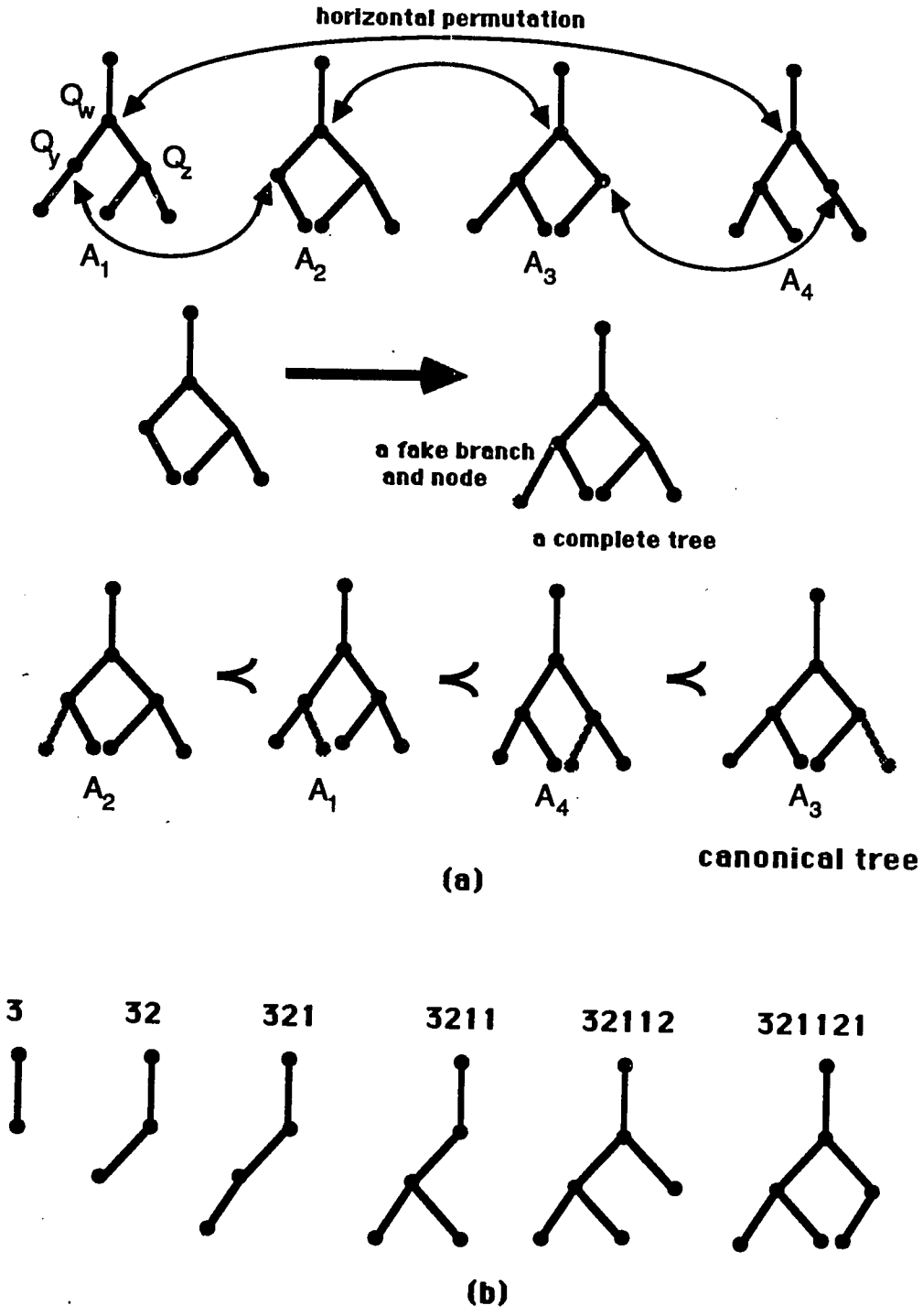


Figure 4.4: (a) A set of track trees and their lexicographical orders. (b) The canonical tree's mark and its decoding sequence.

We now present an algorithm to create the set of canonical trees in TP_k with a given mark. Similarly to the techniques used in proving Theorem 2, the set of marks in TP_i is generated from the set of marks in TP_{i-1}.

Algorithm: C 1 : D^k_{CT} generator

C = $\{a_1, a_2, \cdots, a_r\} = \{\underbrace{11\cdots1}_{r}, \cdots, 11, 1\}$; /*C is the set of marks D^{i-1}_{CT}*/

$i = 2$;

while $(i \leq k)$ **do**

$C' = \emptyset$; /* the new set of marks D^i_{CT}*/

$j = 1$;

while$(j \leq |C|)$ **do**

$m = r$;

while$(m \geq 1)$ **do**

$CT = i$; /*a new mark always starts with i*/

CT=$CT\,\underbrace{a_j a_j \cdots a_j}_{m}$;

add CT as the last element of C';

if$(m < r)$ **then** new_mark(CT, m, j);

m=m-1;

end_do

j=j+1;

end_do

$C = C'$;

i=i+1;

end_do

new_mark(CT, n, j)

begin

if $(n < r)$ **then**$k = j + 1$;

while $(k \leq |C|)$ **do**

$x = r - n$;

while $(x \geq 1)$ **do**

CT=$CT\underbrace{a_k a_k \cdots a_k}_{x}$;

add CT as the last element of C';

if $(x + n < r)$ **then** new_mark(CT, x, a_k);

$x = x - 1$;

end_do

$k = k + 1$;

end_do

end

The above algorithm can be verified by using r=2 and k=3 to generate D^3_{CT} ={3, 32, 3212, 32112, 322, 321, 321121, 32121, 3211, 3211211}. For a given mark CT, we can calculate $|A_{CT}|$ recursively by the following theorem.

Theorem 3 : In a canonical tree CT whose mark is $m_n = nm^{i_1}_{n-1,1} \cdots m^{i_k}_{n-1,k}$, where $m^j_{n-1,i}$ means j copies of $m_{n-1,i}$, there are $|A_{CT}|$ different subtrees with the same mark, and $\sum_{j=1}^{k} i_j \leq r$, $|A_{CT}| = \prod_{\ell=1}^{k} ({}^{r-\sum_{j=0}^{\ell-1} i_j}_{i_n})|m_{n-1,\ell}|$, where $|m_{n-1,\ell}|$ is the number of different subtrees with mark $m_{n-1,\ell}$.

Proof: Since F_{X_1} remains invariant under the horizontal permutation of subtrees of a track tree, when the locations of $m_{n-1,i}$'s are permuted, each location represents a track tree with the same F_{X_1}. Since $m_{n-1,i}$ can be further decomposed into $m_{n-2,k}$'s, $|m_n|$ can be derived recursively, thus leading to $|A_{CT}| = \prod_{\ell=1}^{k} ({}^{r-\sum_{j=0}^{\ell-1} i_j}_{i_n})|m_{n-1,\ell}|$. ∎

D. General Cases

F_{X_1} of TP_i, whose root is $Q_{j_i,i}$, is derived by the techniques discussed thus far. F_{X_2}, $F_{X_3}, \cdots, F_{X_w}$ of TP_i can be derived similarly after some pre-processing steps. We now derive F_{X_ℓ}, $1 < \ell \leq w$, where w is the number of buffers in each queue. The basic idea is that for a set Z of blocking packets, and the subset B of leading packets in Z, we first remove

a leading packet from B. The removed leading packet represents the first blocking packet reaches the root with routing time X_1. After removing one leading packet, blocking packets under it may now qualify to be new leading packets. For the leading packet removed from $Q_{k_j,k}$, there are $|D'_k| = |D_k| - 1$ ways of locating new leading packets below $Q_{k_j,k}$. Let N_B be the set of possible leading packets after removing one leading packet from B, and X'_1 be the time the first leading packet arrives at the root of the new track tree, then $X'_1 = X_2$. Thus, $F_{X_2|B} = F_{X'_1|B} = \frac{1}{|D'_k|} \sum_{B' \in N_B} F_{X'_1|B'}$. The following two pre-processing steps must be applied $\ell - 1$ times recursively for the calculation of F_{X_ℓ}: (1) remove one leading packet from B, and (2) find new leading packets below the packet removed in (1), and form a new track tree. We must enumerate all possible blocking packets' locations in the above steps.

The queue located at the root $Q_{j_i,i}$ is full at time X_w, after which leading packets of $Q_{j_i,i}$ and the r subtrees $TP_{j_1,i-1}, \cdots, TP_{j_r,i-1}$ will start to be blocked. The probability distribution of time to block the root of $TP_{j_\ell,i-1}$, $1 \le \ell \le r$, can be derived independently of others.

Given a set of packets B, let $C_\ell \subset B$ be the subset of blocking packets of $Q_{j,i}$ in $TP_{j_\ell,i-1}$. At time X_w, i.e., when the root of TP_b is full, x of the blocking packets in C_ℓ may have entered $Q_{j_i,i}$. (Note that packets from subtrees other than TP_i may also enter the root.) To calculate the probability distribution of time to block the root of TP_ℓ, $\ell < i$, we must enumerate all possible ways in which these x blocking packets of $Q_{j,i}$ can be removed from TP_ℓ. After the root of TP_ℓ becomes full with blocking packets, the r subtrees of TP_ℓ will have a condition similar to that of TP_ℓ with $Q_{j_i,i}$ as its root.

Let the queues between an arbitrary queue N_ℓ and the root $Q_{j_i,i}$ be $N_\ell, N_{\ell+1}, N_{\ell+2}, \cdots, N_n = Q_{j_n,n}$. Given a set of packets C in TP_{N_ℓ}, the following algorithm is used to calculate the probability distribution of time to block N_ℓ in TP_n. The basic idea of this algorithm is to recursively enumerate all different cases of packets in C to block N_i's, and then calculate the probability distribution for each case.

Algorithm: C 2 : $F_{X_{nw+1|C}}$ calculator

begin

find B_n, the set of blocking packets of N_n, from C;

find A_n, the set of leading packets of N_n, from B_n;

$i = 0$

while($i \leq w$) **do**

RM(A_n, B_n, i, N_n); /*remove i leading packets of N_n from B_n*/

$i = i + 1$;

end_do

end

RM(A_e, B_e, i, N_e) /* A_e is the set of leading packets of N_e

i is the number of blocking packets to be deleted from B_e, and

N_e is the current queue being worked on*/

begin

if($i > 0$) **then**

begin

$j = 1$;

while($j \leq |A_e|$) **do**

remove a leading packet of TP_{N_e} from A_e;

find a new A_e of TP_{N_e} based on current B_e, A_e, and C

RM($A_e, B_e, i - 1, N_e$); /*Remove next leading packets*/

j=j+1;

end_do

end

else

begin

if($w > 0$) **then**

begin

find B_{e-1} from C, B_e, and A_e;

find A_{e-1} based on current A_e, B_{e-1}, and C;

$k = 0$;

```
            while(k < m) do
                RM(Λ, k, N_{w-1}); /*Remove leading packets of N_{w-1}*/
                k=k+1;
            end_do
        end
        else begin
                calculate F_{ND|Λ};
                F_ND = F_ND + F_{ND|Λ};
            end
    end
end
```

The above algorithm calculates the probability distribution of time to block an arbitrary node under a certain workload. Since every processor can generate an infinite number of packets over an infinite period, we can remove as many blocking packets as needed from any track tree. It is intractable to calculate the probability distribution of time to block an arbitrary node in a large tree since a very large number of possible locations of the node must be accounted for. The mean root blocking time and the mean tree congestion time — the time all nodes on the tree are blocked — under different workloads, are plotted in Figs. 4.5 and 4.6, respectively.

4.3.2 Path Locking and Tree Dissipation Times

After releasing a locked path, it will take time to dissipate blocked packets. A tree is said to be *dissipated* when all packets stuck in the tree are routed out of the tree. No simple analytical method appears to exist which can determine the probability distribution of tree dissipation time, because the dissipation sequence of packets in cascaded queues depends strongly on each other. Thus, simulation is used to analyze the path locking time and the tree dissipation time,

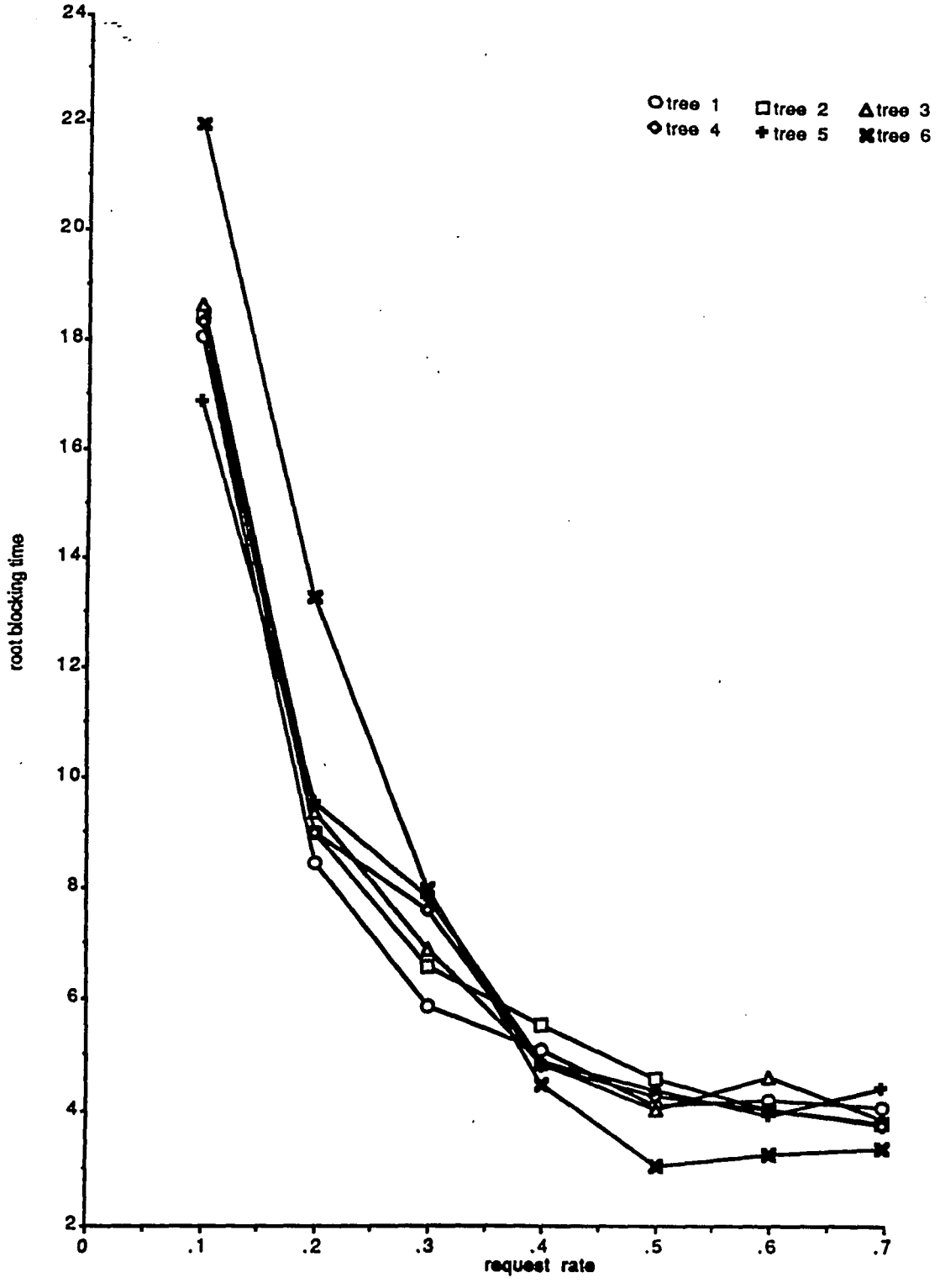


Figure 4.5: The mean root blocking time of 6 trees under different workloads, when N=64 and switch size= 2×2.

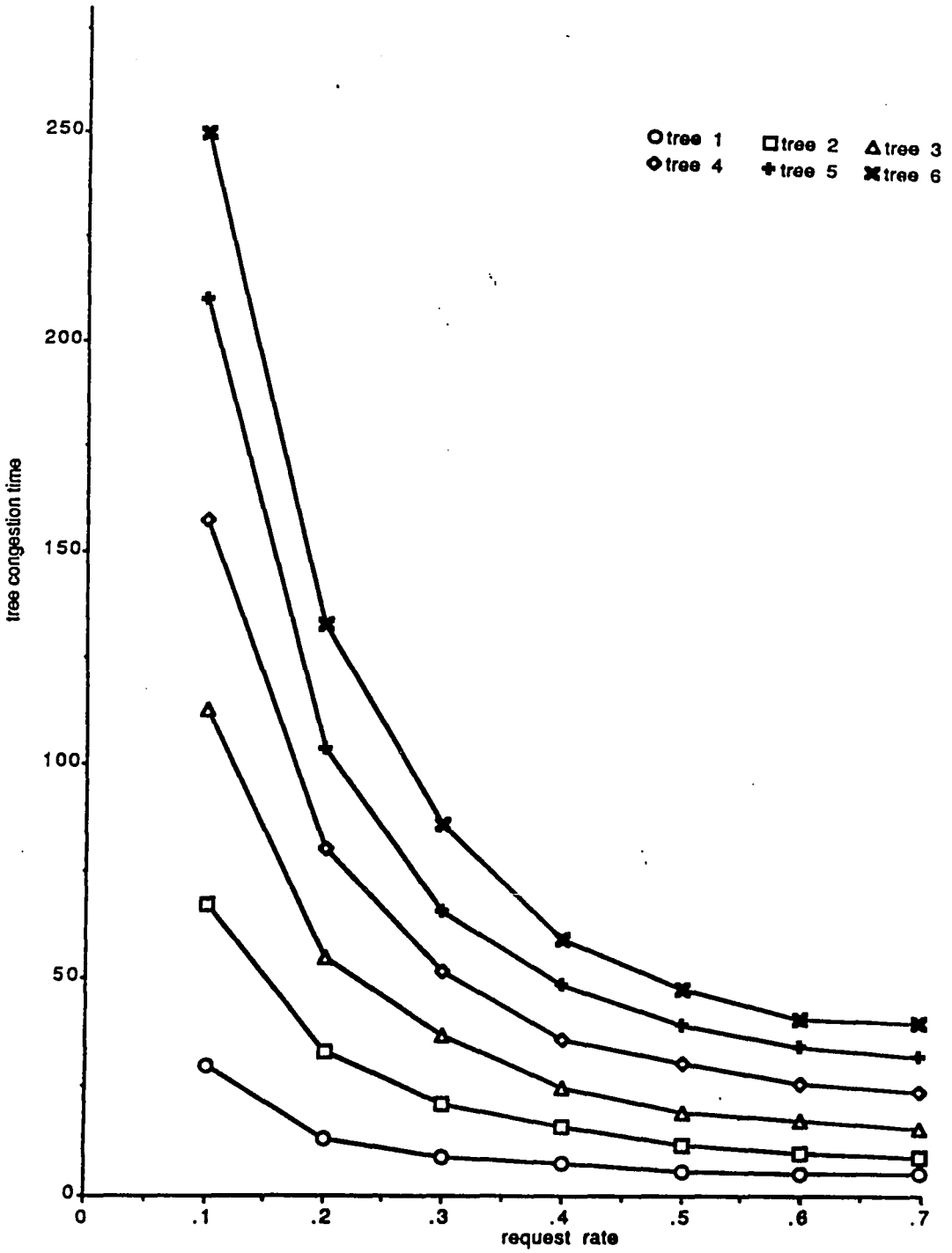


Figure 4.6: The mean tree congestion times under different workloads when N=64 and switch size= 2 x 2.

and the results are analyzed with isotonic regression [87].

Operations of both regular and reduced networks are simulated, where packets are generated according to a geometric distribution with parameter p. Memory modules are assumed to be always available and thus can be accessed at any time. After generating a new packet, it is submitted to the network if the network has an empty buffer available. It takes one network cycle for a packet to cross one stage (link transmission time), and a packet must spend at least one network cycle in each queue after entering it. A locking packet must travel through k stages to reach its destination. Thus, a path's locking time can be expressed as $L_p = k + \ell(p)$, where $\ell(p)$ is a random variable determined by the processors' packet generation rate p. A path's locking time is the same as a regular packet's routing time, because all packets are assumed to have the same priority.

The example system we simulated consists of 64 processors, two buffers in each queue, and 2×2 switches. As expected, the normal operations of regular and reduced networks exhibited identical packet routing times. We simulated only the testing of a single path, but testing of other paths can be similarly simulated. The mean path locking time (regular packet routing time) and tree dissipation time under different workloads (packet generation rates) and a fixed testing procedure are plotted in Figs. 4.7 and 4.8, respectively. When network traffic is heavy, the packet routing time in the last few stages (near memory modules) of the PSMIN is relatively insensitive to the network's input or processors' request rate. The *capacity threshold* of a PSMIN is thus defined as the packet generation rate, above which packets start to build up at the input stages of the network. Other stages of the network are fairly independent of the packet generation rate.

The network can be modeled by a fluid flow process when it has heavy traffic. That is, the traffic in those stages near memory modules is at the network's capacity, and a large number of packets build up at the first few stages of the network. Thus, a more accurate analysis

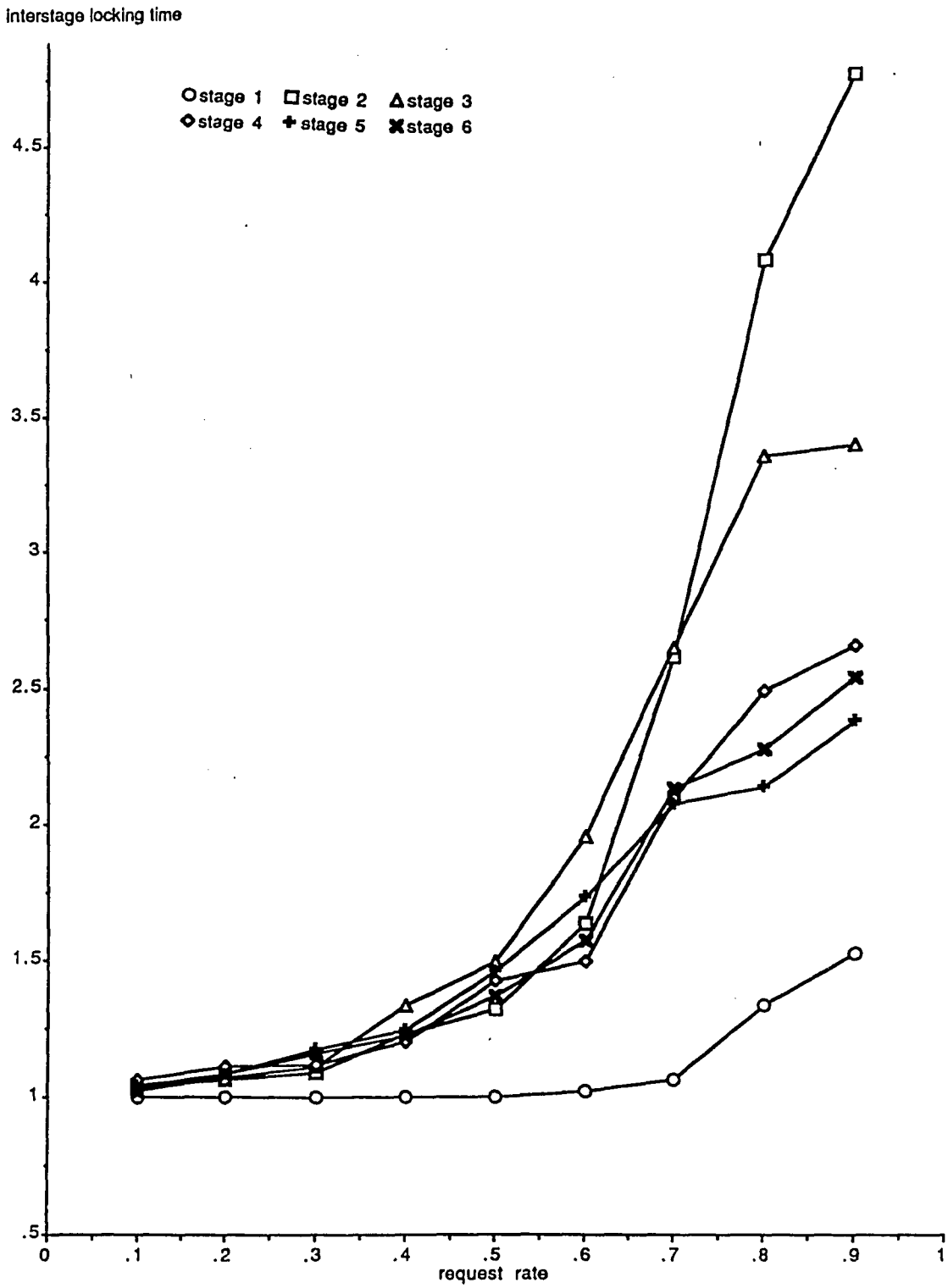


Figure 4.7: The mean path locking time of a PSMIN with $N = 64$, switch size= 2×2, and 2 buffers in each queue.

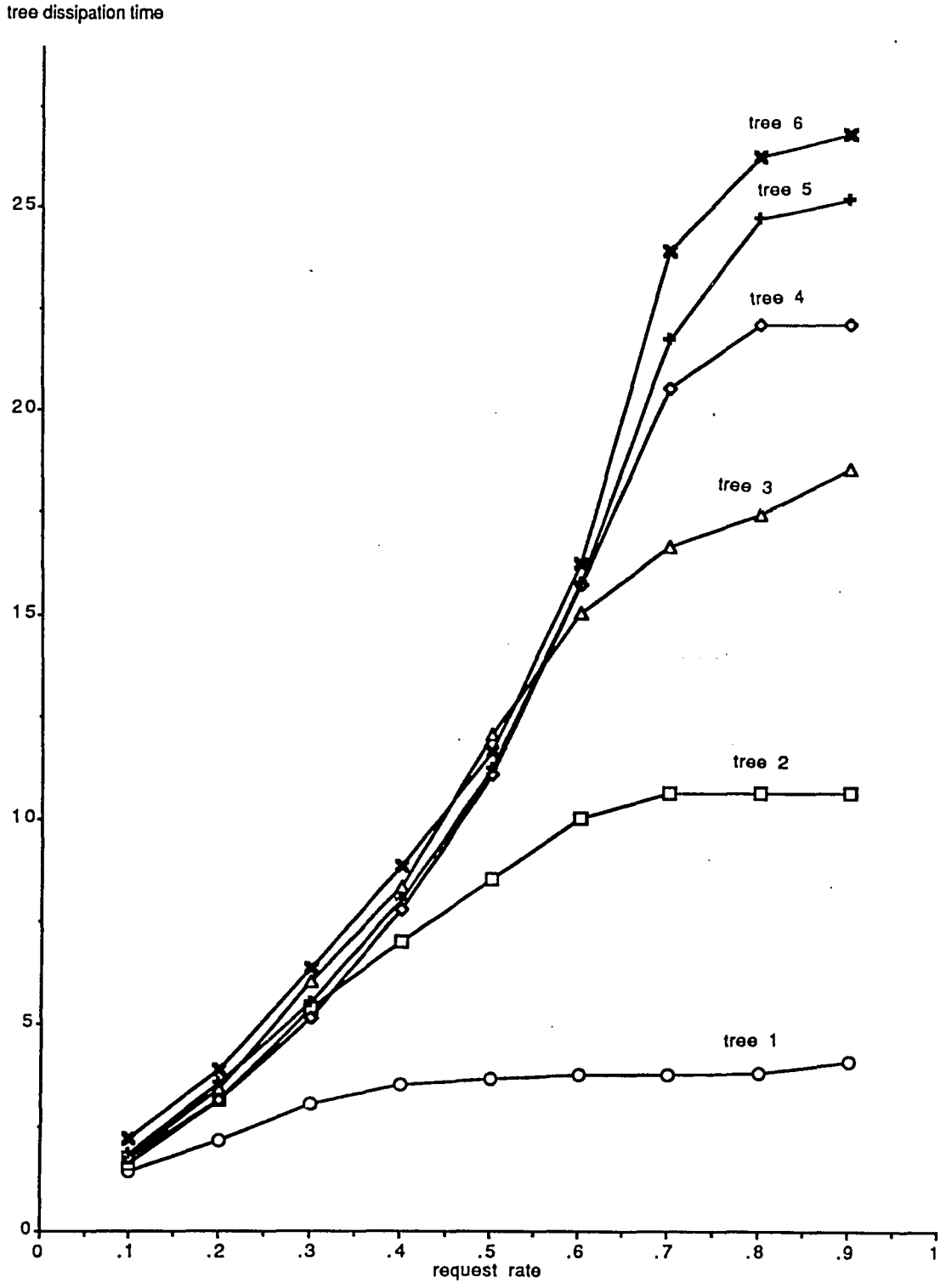


Figure 4.8: The mean tree dissipation time of a PSMIN with $N = 64$, switch size= 2×2, 2 buffers in each queue, and the testing length = 12.

should divide the network into three parts: the input, middle and output stages. In the middle stages of the network, queues have almost identical input and output rates. The output (input) rate of the first few stages is the I/O rate of the middle stages (the actual network input).

The network's congestion and dissipation speed under different testing lengths are simulated with the packet generation rate set at 0.4. As shown in Fig. 4.9, queues in the first two stages get congested and dissipated quickly. In those stages near memory modules, as mentioned above, their tree dissipation times are almost identical. Congestion probabilities at the various nodes are also plotted in Fig. 4.10.

4.4 Optimal System Testing Strategies

Network performance changes dramatically with system workloads, and thus, the parameters to be optimized (in some sense) must be changed accordingly. As a demonstrative example, the testing strategy in the previous section is optimized in this section. The performance penalty induced by the testing of a path in the PSMIN includes the path locking time, the waiting time of stuck packets in the network, and the time of dissipating congested packets. Let T be the length of testing procedure, $D_p^i(T)$ be the mean dissipation time of congestion tree TR_i, and $W_p^i(T)$ be the waiting time of packets stuck in congestion tree i, where p is the parameter of the geometric distribution describing the generation of packets by processors. When a set of network level testing procedures is applied, one can reduce the dissipation time by decreasing the network traffic. The optimization problem can be formulated as:

$$\begin{aligned} \min_{p'} \quad & L_{p'} + \sum_{i=1}^{k} \left(D_{p'}^i(T) + (r^{i+1} - 1)(W_p^i(T) - W_{p'}^i(T)) \right) \\ \text{subject to} \quad & 0 \le p' \le p \end{aligned}$$

where $L_{p'}$ is the lock-up overhead. Although the above optimization problem can be easily solved, it is in general difficult to control the system workload, and thus, its solutions are not practically useful. Thus, we formulate and solve an alternative problem which minimizes

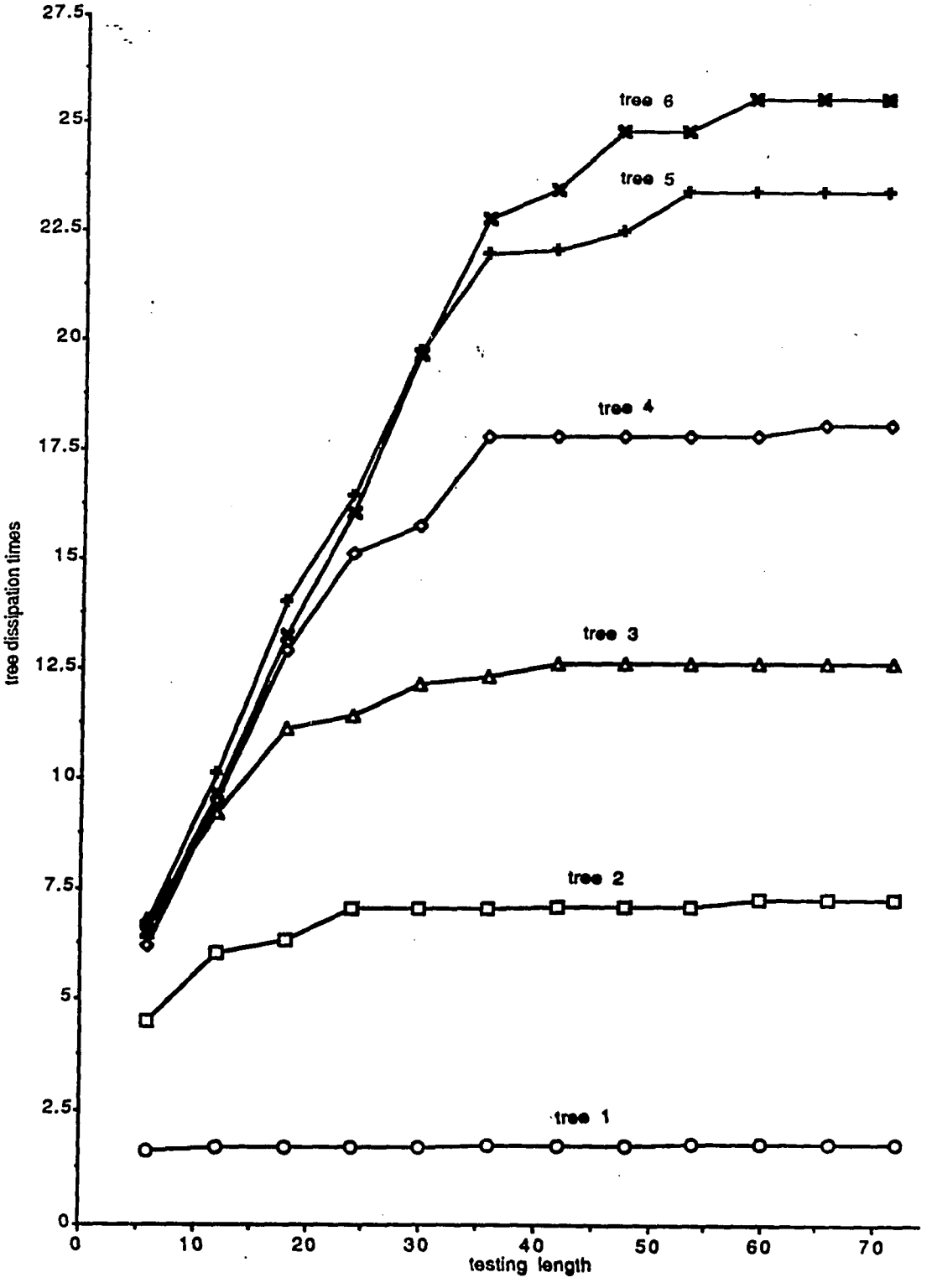


Figure 4.9: The mean tree dissipation times of a 6-stage PSMIN with different testing lengths, switch size= 2×2, and packet generation rate=0.4.

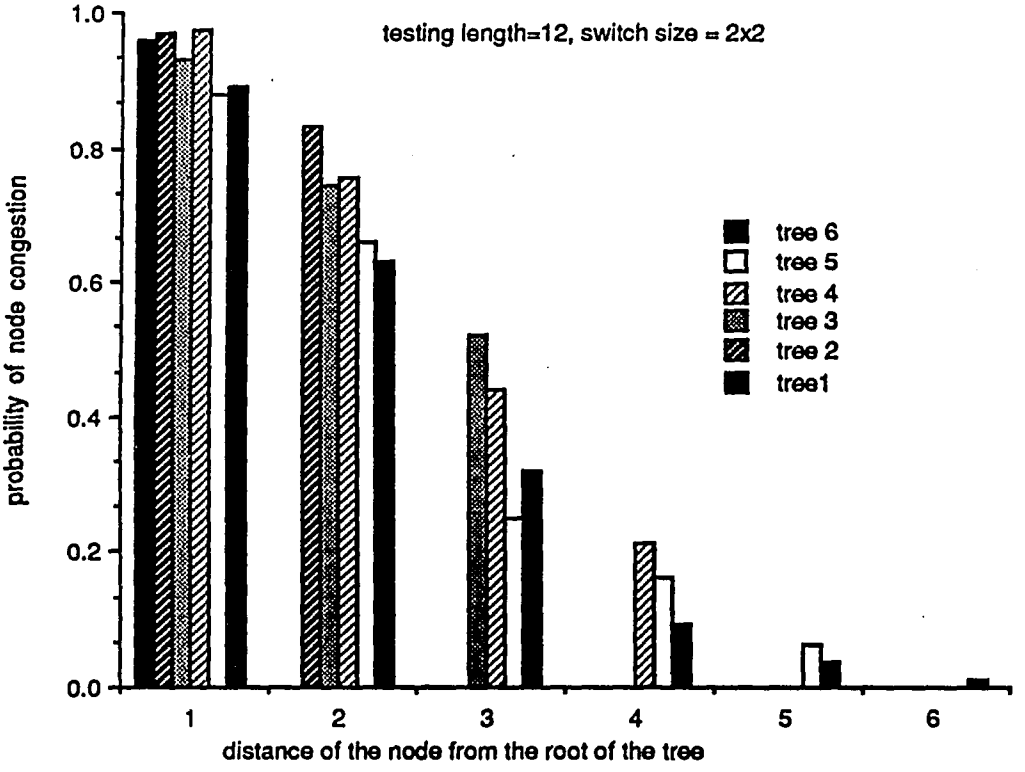


Figure 4.10: The probability of nodes being blocked when the testing length is 12.

the performance penalty of concurrent testing under a fixed workload and testing procedures. That is, given a set of m testing procedures to achieve the required fault coverage, how many testing procedures should be applied in each batch so that the performance penalty may be minimized. The optimization problem is then

$$\min_{n \in \mathbf{I}^+} \quad Z(n) = \lceil \tfrac{m}{n} \rceil L_p + \lceil \tfrac{m}{n} \rceil \sum_{i=1}^{k} (r^{i+1} - 1)(D_p^i(n) + W_p^i(n))$$

$$\text{subject to} \quad 1 \leq n \leq m,$$

where L_p is the lock-up overhead.

According to our simulation results, the mean dissipation times of TR_i can be approximated by

$$D_p^1(\ell) = 1.3$$

$$D_p^2(\ell) = 6.75$$

$$D_p^3(\ell) = \begin{cases} 6.25 + (\ell - 1)2.5 & \ell \leq 3 \\ 11.25 & \ell > 3 \end{cases}$$

$$D_p^4(\ell) = \begin{cases} 6.25 + (\ell - 1)2.5 & \ell \leq 5 \\ 17.5 & \ell > 5 \end{cases}$$

$$D_p^5(\ell) = \begin{cases} 6.25 + (\ell - 1)2.5 & \ell \leq 6 \\ 21.25 & \ell > 6 \end{cases}$$

$$D_p^6(\ell) = \begin{cases} 6.25 + (\ell - 1)2.5 & \ell \leq 7 \\ 23.75 & \ell > 7 \end{cases}$$

where ℓ is the number of testing procedures. D_p^i are saturated to $D_i^{\max}$ (the second line of each D_p^i shown above), when the test pattern is longer than some threshold. $W_p^i(n)$ is assumed to be negligible if $D_p^i \neq D_i^{\max}$. Otherwise, $W_p^i(n) = (r^{i+1} - 1)n$. It is clear that one should not hold up a path too long in each testing session because too many blocking packets may build up in the tree. On the other hand, if each session is too short, the path lock-up overhead

relative to the actual testing time (thus fault coverage) will be too high. Given m and k, the key features in this problem are: (1) when n is the smallest integer satisfying $\lceil \frac{m}{n} \rceil = k$, k will be the smallest integer satisfying $\lceil \frac{m}{k} \rceil = n$, and (2) for a set of integers $\{n_i \mid i = 1, \cdots, k, \lceil \frac{m}{n_i} \rceil = k,\ n_1 < n_2 < \cdots < n_k\}$, we have $Z(n_1) \leq Z(n_2) \leq \cdots \leq Z(n_k)$. Thus, we only have to compare $Z(n_i)$'s where n_i's are the minimal integers satisfying $\lceil \frac{m}{n_i} \rceil = k$. We can solve this optimization problem by the following algorithm.

Algorithm: C 3

B0: $v_{\min} := \infty,\ n := 1$

B1: $k := \lceil \frac{m}{n} \rceil$

B2: **if**$(Z(k) < v_{\min})$ **then** $v_{\min} := Z(k)$

B3: **if**$(Z(\lceil \frac{m}{k} \rceil) < v_{\min})$ **then** $v_{\min} := Z(\lceil \frac{m}{k} \rceil)$

B4: $n := n + 1$; **if**$(n \geq k)$ **then** stop **else** go to B2.

The problem does not have any solution if $v_{min} = \infty$ at the end of execution of the above algorithm. Otherwise, the optimal solution is stored in $v_{\min}$. As shown in Fig. 4.11, the optimal batch size tends to decrease with the number of buffers in each queue.

4.4.1 Optimal Testing Length

In conventional system reliability analysis, fault detection mechanisms are often assumed to have instantaneous response. That is, when a fault occurs, the system can immediately determine whether it should restart the whole task, or roll back to the last checkpoint [93]. This assumption may hold only for those systems with very expensive fault detection and masking mechanisms, but is not valid for systems with imperfect (and also much cheaper)

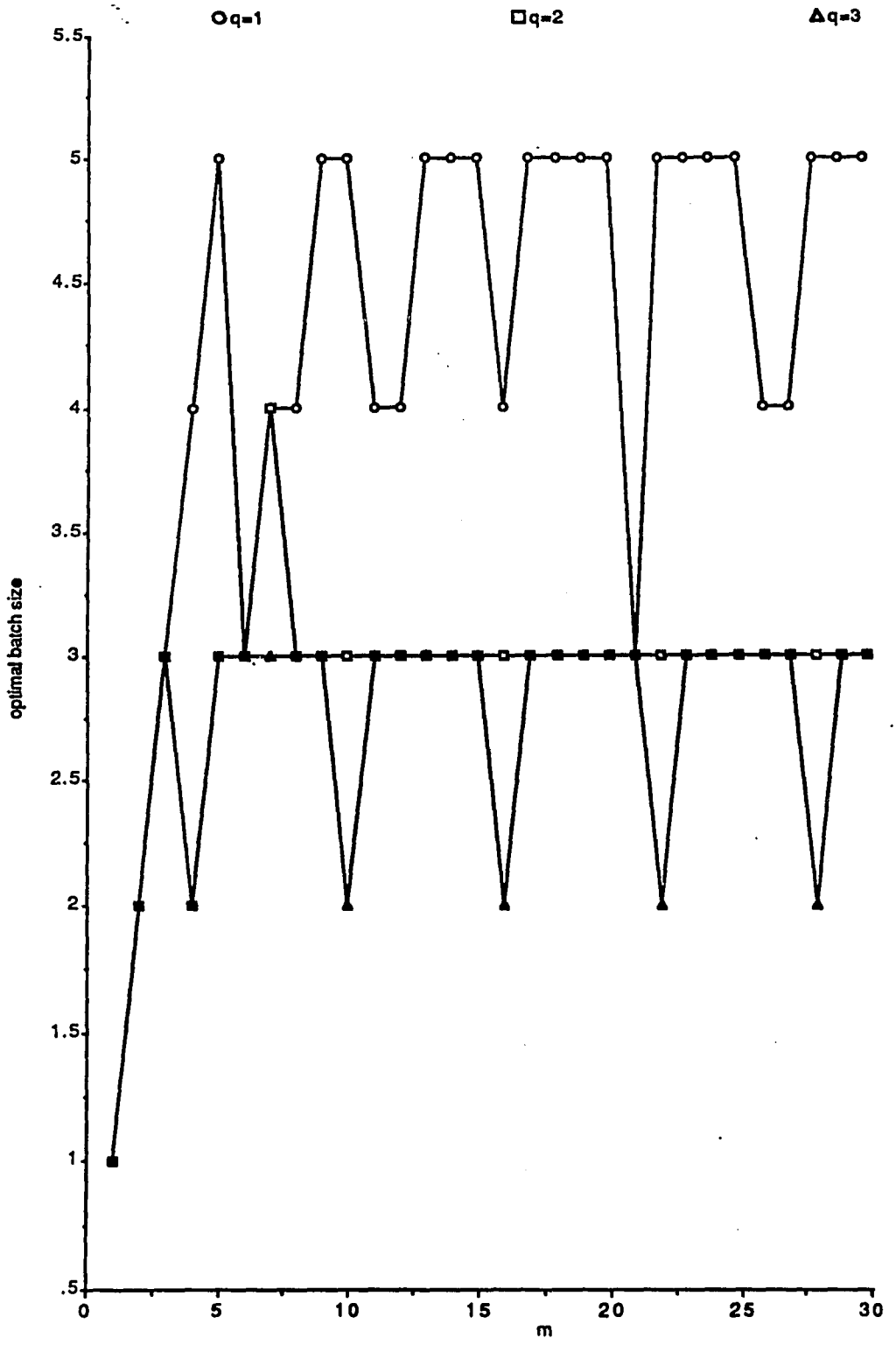


Figure 4.11: Optimal batch sizes with different queue and test pattern lengths.

fault detection mechanisms. Note that different applications impose different requirements of fault coverage and fault detection time. For example, when a detectable fault occurs, the probability of a successful rollback is strongly dependent on the fault detection time, because the probability of a fault causing multiple errors increases with its detection time.

The constraint on the mean performance loss sets an upper bound on the frequency of testing. Thus, the time interval between two successive testing procedures is constrained by the performance loss induced by the application of these testing procedures. Based on this fact, we want to optimize fault coverage subject to a performance loss constraint. The cost function to be optimized takes into account the mean fault detection time, called *latency cost*, and fault coverage. Since the length of test pattern monotonically increases with fault coverage, the mean fault detection time increases monotonically with fault coverage. It should be noted that frequent testing of the system with low coverage test patterns does not improve system reliability, because a large portion of faults are undetectable in such a case.

Let T_δ be the length of test pattern or the test application time to achieve coverage δ, and let fault arrivals follow an exponential distribution with rate λ. The probability of a fault occurrence during $(t_f, t_f + h]$ is $P_f(t_f) = = e^{-\lambda t_f}\lambda(h + \frac{o(h)}{h}) \approx e^{-\lambda t_f}\lambda h$ for a small h. The mean fault detection time is

$$\begin{aligned} E(T_D) &= \int_0^{T_\delta} e^{-\lambda t}\lambda(T_\delta - t)dt \\ &= T_\delta(1 - e^{-\lambda T_\delta}) - \frac{1}{\lambda} + (T_\delta + \frac{1}{\lambda})e^{-\lambda T_\delta}. \end{aligned} \tag{4.3}$$

Several examples with $\delta = 1 - e^{1-T_\delta}$, $\delta = 1 - \frac{1}{T_\delta}$, or $\delta = 1 - \frac{1}{\log_e T_\delta}$, are considered to examine the different levels of difficulty associated with concurrent testing. In Fig. 4.12, these three example conditions are denoted by cvg1, cvg2, and cvg3, respectively. Linear ($LC(t) = t$) and exponential ($LC(t) = e^t$) latency costs indicate different levels of importance of the fault detection time. When the fault detection time is deemed very important, the latency

cost can be expressed as

$$LC(t) = \begin{cases} t, & \text{if the fault is detectable} \\ G_{max} & \text{if the fault is undetectable.} \end{cases}$$

If $\delta = 1 - e^{1-T_\delta}$, the testing cost becomes

$$\begin{aligned} C_1(T_\delta) &= \delta \int_0^{T_\delta} e^{-\lambda t_f} \lambda e^{T_\delta - t_f} dt_f + (1-\delta) G_{max} \\ &= \delta e^{T_\delta} \frac{\lambda}{\lambda+1} (1 - e^{-(\lambda+1)T_\delta}) + (1-\delta) G_{max} \\ &\approx (1 - e^{1-T_\delta}) \lambda (e^{T_\delta} - e^{-\lambda T_\delta}) + e^{1-T_\delta} G_{max}. \end{aligned} \tag{4.4}$$

If $\delta = 1 - \frac{1}{T_\delta}$, we get

$$C_2(T_\delta) = (1 - \frac{1}{T_\delta}) \lambda (e^{T_\delta} - e^{-\lambda T_\delta}) + \frac{1}{T_\delta} G_{max}.$$

As shown in Fig. 4.12 with $G_{max} = 5$, when the system is easy to test ($\delta = 1 - e^{T_\delta}$), one can obtain very high coverage with a short test pattern. Although high fault coverage is easily achievable with long, extensive test patterns, the associated testing cost may become too high to be practically useful. A similar condition occurs when the system is difficult to test (e.g., $\delta = 1 - \frac{1}{T_\delta}$), except that C_2 is inherently larger than C_1. In these two cases, the exponential latency cost sets an upper bound for the length of test pattern.

The other class of examples is when the latency cost is linear, such as

$$LC(t) = \begin{cases} t, & \text{if the fault is detectable} \\ G_{max} & \text{if the fault is undetectable.} \end{cases}$$

If $\delta = 1 - e^{1-T_\delta}$, the testing cost becomes

$$C_3(T_\delta) = (1 - e^{1-T_\delta}) \left(T_\delta (1 - e^{-\lambda T_\delta}) - \frac{1}{\lambda} + (T_\delta + \frac{1}{\lambda}) e^{-\lambda T_\delta} \right) + e^{1-T_\delta} G_{max}.$$

If $\delta = 1 - \frac{1}{T_\delta}, T_\delta > 1$, the testing cost is

$$C_4(T_\delta) = (T_\delta - 1) \left(T_\delta (1 - e^{-\lambda T_\delta}) - \frac{1}{\lambda} + (T_\delta + \frac{1}{\lambda}) e^{-\lambda T_\delta} \right) + \frac{G_{max}}{T_\delta}.$$

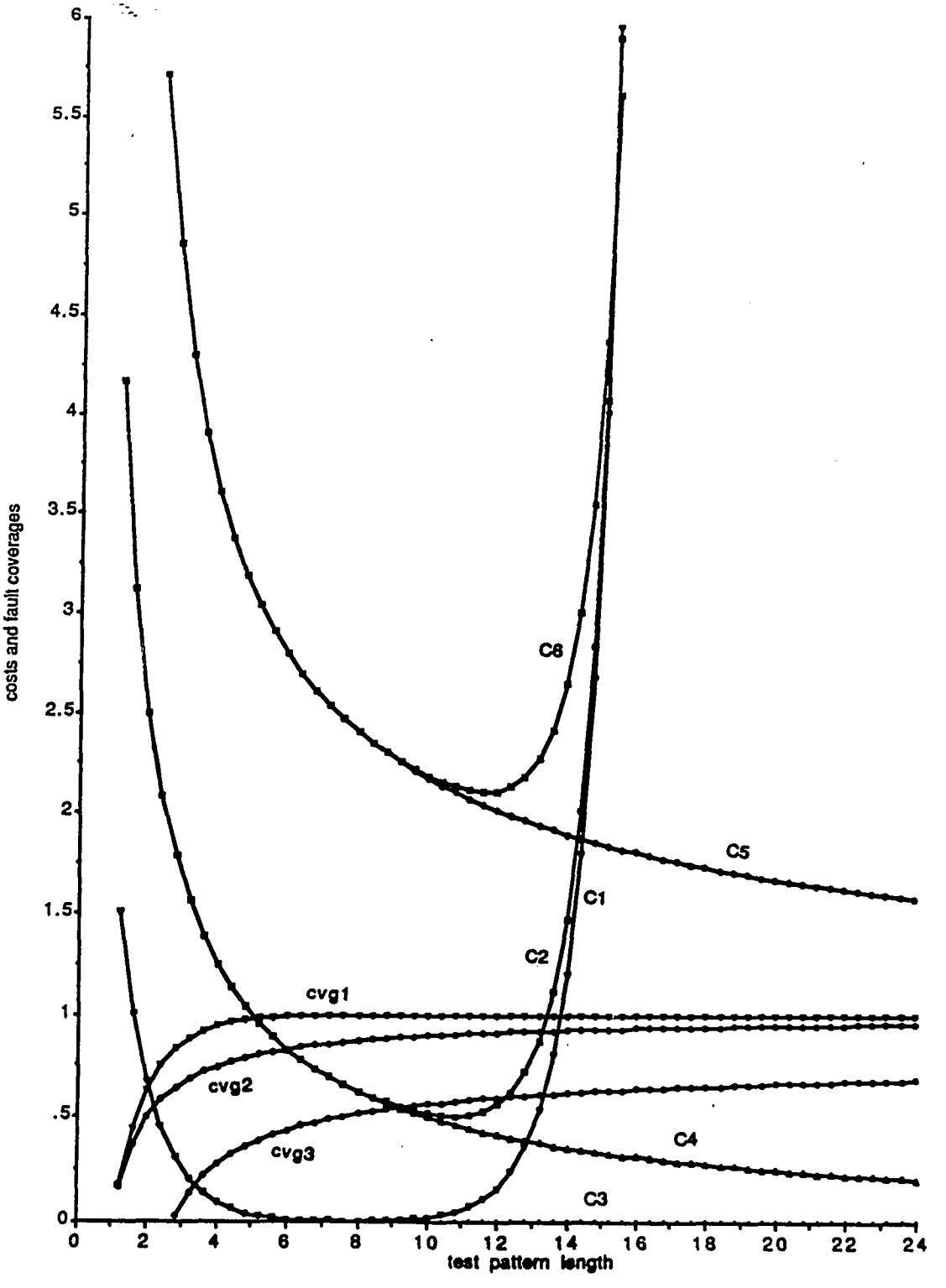


Figure 4.12: Testing costs under different testing lengths when $G_{max} = 5$ and $\lambda = 10^{-6}$.

The testing cost monotonically decreases within any practically interesting length of test pattern, as shown in Fig. 4.12.[5] $C_3(T_\delta)$ denotes the most desirable condition under which very high fault coverage can be obtained quickly, and the latency cost is linear. When $\delta = 1 - \frac{1}{T_\delta}$, $C_3(C_4)$ becomes a monotonically decreasing function of T_δ. Due to the poor quality of testing in this case, $C_3(C_4)$ is much higher than $C_1(C_2)$.

When the test pattern is too short, the cost contributed by undetectable faults is the dominating factor of the cost function. On the other hand, when the test pattern is excessively long, the latency cost will become the dominating factor. When the system is very hard to test, e.g., cvg3, the latency cost will be very high. $C5$ and $C6$ in Fig. 4.12 represent the linear and exponential testing costs, respectively.

4.4.2 Optimal Testing Rings

A receiver of testing packets can determine the correctness of the packet's format and its routing path without interacting with the sender, because the receiver can identify the source of the packet, and the path that the packet traversed. It is crucial to create *testing rings* to minimize the required number of testing procedures. In a testing ring, a processor (sender) sends testing packets, and testing outcomes are evaluated by another processor (receiver). Then, the receiver passes the testing packets to the next processor. Except for the first sender, no processor is allowed to be visited twice in a testing ring. A testing ring is completed when the testing packet returns to the original sender. Then, the next testing ring is created, and the same procedure will be repeated, until the network is completely tested.

The longest testing ring occurs when the testing packet does not return to the original sender until all processors have been visited. It occurs when processor i sends the testing packet to processor j such that $j = i + q \bmod N$, and the largest common divisor between N

[5] For clarity, the test pattern length longer than 26 is not shown.

and q is 1. The longest testing ring is usually not optimal, because many path segments may be tested more than once. When the size of switch is $r \times r$, any optimal testing procedure should test each of the r^2 paths in a switch exactly once. Imagining that all processors are testing the network, an optimal testing procedure results when all switches have *conflict free* permutations (packets in queues of the same switch enter different output ports). For example, an optimal testing ring occurs when all switches have the same permutation, $i \rightarrow i + q \bmod r$, $0 \leq q \leq r-1, \forall i$. Since each processor is connected to one input port of a switch, and the input port can be connected to its r output ports, each processor has to be used to test (different parts of) the network r times.

The switch permutations mentioned above can be set up when processors send packets to their appropriate destinations. For example, when the network topology is baseline, and the size of switch is 2×2, the interconnection rules are (1) $(a_k a_{k-1} \cdots a_1)_i \rightarrow (a_k \cdots a_{i+1} 0\, a_{i-1} \cdots a_2)_{i+1}$, where i is the stage number, if $a_1 = 0$, and (2) $(a_k a_{k-1} \cdots a_1)_i \rightarrow (a_k \cdots a_{i+1} 1\, a_{i-1} \cdots a_2)_{i+1}$ if $a_1 = 1$. Optimal testing rings can be created when all switches have *straight* or *cross* settings. When switches are set to have straight (cross) interconnections, testing rings are formed as follows: (1) Starting with processor P_0, $P_i = (a_1 a_2 \cdots a_k)$ sends the testing packet to processor $R_i = (a_k \cdots a_2 a_1)$ ($R_i = (\bar{a}_k \bar{a}_{k-1} \cdots \bar{a}_1)$ for cross connections) and vice versa, (2) After P_i receives the testing packet from R_i, P_i notifies processor P_{i+1} to start the next testing ring. If P_{i+1} has already acted as a receiver before, it simply passes the testing task to P_{i+2}.

Another example is flip networks, where the link permutation is a perfect shuffle. P_i sends the testing packet to itself or P_{N-i} through the network. When a processor submits testing packets to itself, switches will have straight interconnections, i.e., upper (lower) input port is connected to upper (lower) output port. On the other hand, all switches have the cross connection, i.e., the upper (lower) input port is connected to the lower (upper) output port,

when P_i submits testing packets to P_{N-i}. If the testing packet does not return to the sender in a pre-specified time-out period, either the path $\Pi_{i,j}$ or the receiver j is faulty. If P_i acts as the sender and receiver at the same time, any faulty component can be identified solely by itself. When P_i and P_j test two paths $\Pi_{i,j}$ and $\Pi_{j,i}$, P_j can inform P_i the testing results of $\Pi_{i,j}$ through $\Pi_{j,i}$, and vice versa. Thus, except for the switch $S \in \Pi_{i,j} \bigcap \Pi_{j,i}$, all other faulty queues on $\Pi_{i,j}$ and $\Pi_{j,i}$ can be detected and located by both processors in at most three steps.

4.5 Conclusion

Concurrent testing strategies for packet switching networks are analyzed and optimized in this chapter. Most conventional reliability analyses focus on systems with fault masking capability, yet little work has been done on concurrent fault detection mechanisms. Important network parameters such as path locking time, and tree congestion time, are analyzed and/or simulated.

The off-line part of the polynomial testing method, the *low-level* testing, is not discussed in this chapter. Analysis of the low-level testing is straightforward, because the network's normal operation is completely stopped for testing, and then resumed after testing. Testable design of the low-level testing makes it very effective but less flexible than the high-level testing. Testable design changes with the circuit implementation, and may pose high hardware overhead for circuit switching networks. Note that a common property between the high-level testing and low-level testing is that, when they have the same fault coverage and all the testing procedures are applied to the system within the same period of time, they will have the same mean fault detection time. Thus, selection between the high-level and the low-level methods should be based on the cost and reliability requirements.

The probability distribution of an arbitrary node being blocked is derived by a systematic

method. The probability distribution of time for the root of a tree to get blocked is first derived, and then the computational complexity is reduced by avoiding repetitive calculations of the same distributions. Then, the distribution of time to block the second buffer in the same queue can be derived by removing one of the leading packets and adding some newly arriving leading packets. For an arbitrary node in the tree, we must recursively remove from and add leading packets to different nodes. The probability distribution of node congestion in the tree can be systematically derived by the proposed method.

The required computation grows quickly as we move from the root to lower levels of the tree. In such a case simulations are used to derive mean path locking times, mean root congestion times, and mean tree dissipation times.

The proposed method can be used to derive the probability distribution of communication delays in networks like a (hexagonal) mesh, or an N-cube. For example, when the method is used to derive the communication delay of hexagonal mesh networks utilizing the shortest path routing [17], all possible shortest paths are spanned into a tree with unwanted branches removed.

Testing is not free, and unless the system has an easily testable structure, a more realistic assumption is that the length of test pattern monotonically increases with fault coverage. Since the fault detection time is very important for fault recovery, it is necessary to trade fault coverage for mean fault detection time. Examples in Section 4.4 show that when the mean fault detection time is extremely important, only fault masking can meet the system requirement. However, for most other applications, we can easily achieve the desired fault coverage and significantly improve the system reliability with concurrent testing strategies.

Appendix 4.A: List of Symbols

A, PK_i	$A = \{PK_1, PK_2, \cdots, PK_m\}$ is a set of m leading packets, PK_i, $i = 1, \cdots, m$. It can also be represented as a track tree.
A_{CT}	The set of track trees with the same distribution of root blocking time.
$AR_{i,j}$	The i-th arbiter at stage j.
$B_d^j(n)$	The dissipation time of the j-th tree when the packet generation rate is p, and the length of test pattern is n.
CT, A_{CT}	CT is a canonical tree. A_{CT} is the set of track trees with the same canonical tree CT.
D_b	Sets of disjoint subtrees in a tree of height b.
$D_{CT}^k, \lvert D_{CT}^k \rvert$	A canonical tree of height k, and $\lvert D_{CT}^k \rvert$ is the number of components in D_{CT}^k.
$\mathbf{I}^+$	The set of positive integers.
LCN	The lowest common node of two or more leading packets.
P_i	Processor i.
M_i	Memory module i, which is the partner of P_i.
$Q_{i,j}$	The i-th queue at stage j.

$S_{i,j}$ The i-th switch at stage j.

T_i, X_j T_i is the time for PK_i to reach the root of a tree. X_j is the time when the j-th leading packet reaches the root of a tree. $(X_1, X_2, \cdots, X_m)$ is called the *order statistics*.

$T_i \preceq T_j$ T_i is lexicographically less than or equal to T_j.

$TP_{\ell,m}$ $TP_{\ell,m} = TR_{Q_{\ell,m}}$, the destination tree of $Q_{\ell,m}$. $TP_{\ell,m}$ is abbreviated as TP_m whenever the value of ℓ is immaterial.

TR_{M_i} A destination tree, $TR_{M_i} = \bigcup_{j=0}^{k} \Gamma^j(M_i)$.

$\Gamma(Q_{i,j})$ The source queues of $Q_{i,j}$.

$\Delta(Q_{i,j})$ The destination queues of $Q_{i,j}$.

Θ, Θ' $\Theta = (Y_{0,1}, Y_{1,1} \cdots, Y_{N-1,k})$. $Y_{i,j}$ is a random variable denoting the number of packets in $Q_{i,j}$ of PSMIN before reduction. $\Theta' = (Y_1, Y_2 \cdots, Y_k)$, where Y_i is a random variable denoting the number of packets in Q_i of the reduced network model.

δ, T_δ δ is fault coverage, and T_δ is the length of test pattern to achieve δ.

CHAPTER 5

CONCLUSION

The major contributions of this dissertation are three new network architectures and the analysis of network dynamics. A discussion of these architectures along with potential applications of the completed work for future research are summarized in this chapter.

First of all, the NOMI technique presented in Chapter 2 is shown to be a useful design technique to improve system performance at reduced cost. We have shown that the conventional memory interleaving technique should be combined with network overlapping to improve system performance at reduced cost. The proposed branch and bound optimization procedure can either optimize system cost subject to performance constraints, or vice versa.

The other two new architectures, discussed in Chapter 3, are aimed at detecting faults efficiently to improve network reliability. These two architectures support the high-level concurrent testing and low-level off-line testing of packet-switching MINs, respectively. In the low-level testing strategy, the network can be tested off-line in a short time period with very high fault coverage. On the other hand, the high-level testing strategy requires minimal amounts of interactions between processors, and is suitable for those systems with non-stop operations.

The last major contribution is the analysis of network dynamics during high-level network testing. In the high-level network testing, packets will be blocked if they have to traverse

through a locked path. If a path is locked up for a long time period, a congestion tree, in which all packets are blocked, will eventually be formed. The probability distribution of the time to block an arbitrary node in the network under test has been derived. The computational complexity in calculating the probability distribution of node blockage is drastically reduced by the proposed canonical tree model. Other network dynamics like the mean tree dissipation time and mean path locking time have been studied via extensive simulation. We also developed a reduced network model to reduce the network simulation time substantially. Finally, optimal testing coverage has been derived by trading the mean fault detection time for fault coverage.

Efficient testing methods are essential to achieve high system reliability. Architectures proposed in this dissertation are intended to fill the gap between on-line fault detection and off-line fault diagnosis. Circuit-switching MINs do not have buffers, and thus, do not form congestion trees. Concurrent testing in OCSMINs can be easily implemented by periodically assigning one of the w phases for testing. Since network operations are synchronous and every cluster knows which of the w phases is dedicated to testing, all the routing faults mentioned in Chapter 3 can be easily tested. It will be more expensive to perform the low level testing in OCSMINs due to the extra hardware needed for pattern generation. However, the hardware overhead can be reduced by having one single pattern generator for each stage. It is clear that the clustering/NOMI technique, developed in Chapter 2, and testable architectures, developed in the Chapter 3, can be easily combined in both circuit switching and packet switching networks. When a network component fails, the excessive performance loss caused by a smaller network size (resulting from the clustering or NOMI technique) is minimized by detecting and repairing the fault by the easily testable architectures. Furthermore, applying the clustering or NOMI technique to a network with an easily testable architecture will make its hardware overhead negligible.

The network dynamics analysis presented in Chapter 4 may become very useful for the

design of packet-switching telecommunication systems. The extremely high capacity of optical fibers must be matched with very high bandwidth, e.g., 1 Gbit/s or higher, packet-switching MINs [79, 113, 52], to make broadband integrated services digital networks (BISDNs) feasible. Data, digitized voice and video signals are packetized for transmission via BISDNs. Testing large scale packet-switching MINs used in BISDNs is especially challenging, because stopping normal network operations for testing will result in loss of a large fraction of their bandwidth. Furthermore, fault symptoms will change dynamically due to continuous input packets. When a parallel packet transmission format is used [79], our high-level testing strategy can be easily applied to the design of packet-switching MINs. On the other hand, when a serial packet transmission format is employed, the easily-testable architecture for the low-level testing can be applied with a low hardware overhead.

Several different latency cost functions have been considered in optimizing testing strategies. The latency cost is dependent on a system component's functions and the system workload. An interesting future research topic is to characterize latency cost functions for different system components under different workloads. Characterization of latency cost is particularly crucial for real-time systems that must complete tasks before their hard deadlines.

Most faults in circuit switching networks do not cause serious performance loss. On the other hand, packet switching MINs are more prone to system crash due to network congestion and component failures. For example, when a blocking fault occurs in a switch located at the last stage of the MIN, the whole network could be paralyzed once a congestion tree with the faulty switch as its root is formed.

The performance impact of congestion trees is determined by the number of packets blocked in the tree, the blocking times of these (blocked) packets, and the intensity of incoming network traffic. Since losing a few (voice) packets does not degrade the speech quality significantly[50], the simplest method to dissipate a congestion tree is to discard blocked packets. This strategy

can be implemented by setting time-out periods for packets. Detection of congestion trees, along with the dissipation of congestion trees, are deemed vital to the success of next generation large scale telecommunication switching systems.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] G. B. Adams and H. J. Siegel, "The extra stage cube: A fault tolerant interconnection network for supersystems," *IEEE Trans. Comput.*, vol. C-31, no. 5, pp. 443–454, May 1982.

[2] A. K. Adiga and S. R. Deshpande, "Evaluation of effectiveness of circuit based and packet based interconnection networks via petri net models," *Proceeding of Int'l Conference on Parallel Processing*, pp. 533–541, 1987.

[3] D. P. Agrawal, "Testing and fault tolerance of multistage interconnection networks," *Computer*, pp. 41–53, Apr. 1982.

[4] D. P. Agrawal, "Graph theoretical analysis and design of multistage interconnection networks," *IEEE Trans. on Comput.*, vol. C-32, no. 7, pp. 637–648, Jul. 1983.

[5] D. P. Agrawal and J.-S. Leu, "Dynamic accessibility testing and path length optimization of multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-34, no. 3, pp. 255–266, Mar. 1985.

[6] K. E. Batcher, "The flip network in staran," *Proceeding of Int'l Conference on Parallel Processing*, pp. 65–71, Aug. 1976.

[7] J. Bellamy, *Digital Telephony*, John Wiley & Sons, 1982.

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

[9] V. E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic, New York, 1965.

[10] R. Bennetts, *Design of Testable Logic Circuits*, Addison-Wesley, 1984.

[11] D. P. Bhandarkar, "Analysis of memory interference in multiprocessors," *IEEE Trans. Comput.*, vol. C-24, no. 9, pp. 897–908, Sep. 1975.

[12] L. N. Bhuyan, "An analysis of processor-memory interconnection networks," *IEEE Trans. Comput.*, vol. C-34, no. 3, pp. 279–283, Mar. 1985.

[13] L. N. Bhuyan and D. P. Agrawal, "Design and performance of generalized interconnection networks," *IEEE Trans. Comput.*, vol. C-32, no. 12, pp. 1081–1090, Dec. 1983.

[14] R. Bianchini and J. R. Bianchini, "Wireability of an ultracomputer," *NYU Ultracomputer note #43*, 1982.

[15] M. A. Breuer and A. D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science, Rockville, MD, 1976.

[16] M. S. C. P. Kruskal and A. Weiss, "On the distribution of delays in buffered multistage interconnection networks for uniform and nonuniform traffic," *Proceeding of Int'l Conference on Parallel Processing*, pp. 215–219, 1984.

[17] M.-S. Chen, K. G. Shin, and D. Kandlur, "Addressing, routing and broadcasting in hexagonal multiprocessors," *IEEE Trans. on Comput.*, 1988 (in press).

[18] P. Chen et al., "Interconnection networks using shuffles," *Computer*, pp. 55–64, Dec. 1981.

[19] V. Cherkassky and M. Malek, "On permuting properties of regular rectangular sw-banyan," *IEEE Trans. Comput.*, vol. C-34, no. 6, pp. 542–546, Jun. 1985.

[20] V. Cherkassky and E. Opper, "Fault diagnosis and permuting properties of cc-banyan networks," *Proc. of Real Time Systems Symposium*, pp. 175–183, 1984.

[21] V. Cherkassky, E. Opper, and M. Malek, "Reliability and fault diagnosis analysis of fault-tolerant multistage interconnection networks," *Digest of Papers, FTCS-14*, pp. 246–251, 1984.

[22] A. Chin, *Congestion Control in Routing Networks*, Master Thesis, MIT, Massachusetts, 1986.

[23] C. Y. Chin and K. Hwang, "Connection principles for multipath packet switching networks," *12th Int'l Symposium on Computer Architecture*, pp. 99–108, 1984.

[24] C. Y. Chin and K. Hwang, "Packet switching networks for multiprocessors and data flow computers," *IEEE Trans. Comput.*, vol. C-33, no. 11, pp. 991–1003, Nov. 1984.

[25] L. Ciminiera and A. Serra, "A fault-tolerant connecting network for multiprocessor systems," *Digest of Papers, FTCS-12*, pp. 113–122, 1982.

[26] C. Clos, "A study of nonblocking switching networks," *Bell Syst. Tech. Journal*, vol. 32, pp. 406–424, 1953.

[27] T. H. Cormen, "Efficient multichip partial concentrator switches," *Proceeding of Int'l Conference on Parallel Processing*, pp. 525–532, 1987.

[28] B. Corp., "Numerical aerodynamic simulation facility feasibility study final report," *NASA Contract Report CR-152285*, Mar. 1979.

[29] I. Corp., "Data sheet: Intel's personal supercomputer,", May 1985.

[30] N. Corp., "Ncube handbook,", 1985.

[31] N. J. Davis IV, W. T. Y. Hsu, and H. J. Siegel, "Fault location techniques for distributed control interconnection networks," *IEEE Trans. Comput.*, vol. C-34, no. 10, pp. 902–910, Oct. 1985.

[32] J. B. Dennis, "Data flow supercomputers," *Computer*, pp. 48–56, Nov. 1980.

[33] S. Dhar, M. A. Franklin, and D. F. Wann, "Reduction of clock delays in vlsi structures," *Intl. Conf. Computer Design*, pp. 778–781, 1984.

[34] D. M. Dias and J. R. Jump, "Analysis and simulation of buffered delta networks," *IEEE Trans. Comput.*, vol. C-30, no. 4, pp. 273–282, Apr. 1981.

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

[36] T. Y. Feng, "A survey of interconnection network," *Computer*, pp. 12–27, Dec. 1981.

[37] T. Y. Feng and I. P. Kao, "On fault-diagnosis of some multistage networks," *Digest of Papers, FTCS-12*, pp. 99–101, 1982.

[38] T. Y. Feng and C. L. Wu, "Fault-diagnosis of a class of multistage interconnection networks," *IEEE Trans. on Comput.*, vol. C-30, no. 10, pp. 743–758, Oct. 1981.

[39] J. P. Fishburn and R. A. Finkel, "Quotient networks," *IEEE Trans. Comput.*, vol. C-31, no. 4, pp. 288–295, Apr. 1982.

[40] M. A. Franklin, "Pin limitations and partitions of vlsi interconnection networks," *IEEE Trans. Comput.*, vol. C-30, no. 4, pp. 283–290, Apr. 1981.

[41] M. A. Franklin, S. A. Kahn, and M. J. Stucki, "Design issue in the development of a multiprocessor communication network," *Proc. 6th Annual Sypm. on Computer Architecture*, pp. 182–187, 1979.

[42] W. K. Fuchs, J. A. Abraham, and K. H. Huang, "Concurrent error detection in vlsi interconnection networks," *Digest of Papers, FTCS-13*, pp. 309–315, 1983.

[43] D. D. Gajski, D. J. Kuck, D. H. Lawrie, and A. Samesh, "Cedar- a large scale multiprocessor," *Proceeding of Int'l Conference on Parallel Processing*, pp. 524–529, 1983.

[44] C. J. Georgiou, "Fault-tolerant crosspoint switching networks," *Digest of Papers, FTCS-14*, 1984.

[45] L. D. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," *Proceedings of the 1st Annual Symp. on Computer Architecture*, pp. 21–28, 1973.

[46] J. Goldberg et al., *Development and Analysis of SIFT*, NASA Langley Research Center, Hampton, VA 23665, Feb. 1984.

[47] S. W. Golomb, *Shift Register Sequences*, Holden-Day, Inc., 1967.

[48] A. Gottlieb et al., "The nyu ultracomputer- designing an mimd shared memory parallel computer," *IEEE Trans. Comput.*, vol. C-32, no. 2, pp. 175–189, Feb. 1983.

[49] J. P. Hayes, "On realizations of boolean functions requiring a minimal or near-minimal numbers of tests," *IEEE Trans. on Comput.*, vol. C-20, no. 12, pp. 1506–1513, Dec. 1971.

[50] A. Hills and K. Scott, "Perceived degradation effects in packet speech," *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. ASSP-5, no. 5, pp. 699–701, May 1987.

[51] A. L. . Hopkins et al., "Ftmp - a highly reliable fault tolerant multiprocessor for aircraft," *Proc. IEEE*, vol. 66, no. 10, pp. 1221–1239, Oct. 1978.

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

[53] R. Huslende, "Optimal cost/reliability allocation in communication networks," *Digest of pappers, FTCS-13*, pp. 348–355, 1983.

[54] M. Inc., *MC68020 32-Bit Microprocessor User's Manual*, Prentice-Hall Inc., 1984.

[55] H. Inose, *An Introduction to Digital Integrated Communication Systems*, Univ. of Tokyo Press, Tokyo, 1979.

[56] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw Hill, 1978.

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

[58] C. P. Kruskal and M. Snir, "The importance of being square," *Conference Proceeding, Annual Int'l Symp. Computer Architecture*, pp. 91–98, 1984.

[59] M. Kumar, D. M. Dias, and J. R. Jump, "Switching strategies in a class of packet switching networks," *Proc. 10th Annual Symp. on Computer Architecture*, pp. 284–300, Dec. 1983.

[60] M. Kumar, D. M. Dias, and J. R. Jump, "Switching strategies in shuffle-exchange packet switching networks," *IEEE Trans. Comput.*, vol. C-34, no. 2, pp. 180–186, Feb. 1985.

[61] P. K. Lala, *Fault Tolerant and Fault Testable Hardware Design*, Prentice-Hall Inc., 1985.

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

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

[64] D. C. H. Lee and J. P. Shen, "Easily-testable (n,k) shuffle/exchange networks," *Proc. of Int'l Conf. on Parallel Processing*, pp. 65–70, 1983.

[65] M. Lee and C. L. Wu, "Performance analysis of circuit switching baseline interconnection network," *Proc. 10th Annual Symp. on Computer Architecture*, pp. 82–90, June 1984.

[66] Y. H. Lee and K. G. Shin, "Design and evaluation of a fault-tolerant multiprocessor using hardware recovery blocks," *IEEE Trans. on Comput.*, vol. C-33, no. 2, pp. 113–124, Feb. 1984.

[67] J. E. Lilienkamp, D. H. Lawrie, and P. C. Yew, "A fault tolerant interconnection network using error correcting codes," *Digest of Papers, FTCS-12*, pp. 123–125, 1982.

[68] W. Y.-P. Lim, "A test strategy for packet switching networks," *Proc. of Int'l Conference on Parallel Processing*, pp. 96–98, 1982.

[69] W. Lin and C. L. Wu, "Design of a 2 x 2 fault-tolerant switching element," *Proceeding of Conference, Comput. Archit.*, pp. 181–189, 1982.

[70] J. Y. Maeng, "Self-diagnosis of multistage network-based computer systems," *Digest of Papers, FTCS-13*, pp. 324–331, 1983.

[71] M. Malek and E. Opper, "Multiple fault diagnosis of sw-banyan networks," *Digest of Papers, FTCS-13*, pp. 446–449, 1983.

[72] , "Special issue on no.4 ess," *The Bell System Technical Journal*, Sep. 1979.

[73] E. J. McCluskey and S. Bozorgui-Nesbat, "Design for autonomous test," *IEEE Trans. Comput.*, vol. C-30, no. 11, pp. 866–875, Nov. 1981.

[74] D. Nassimi and S. Sahni, "Parallel permutation and sorting algorithms and a new generalized connection network," *Journal of ACM*, pp. 642–667, July 1982.

[75] E. Opper and M. Malek, "Real-time diagnosis of banyan networks," *Proc. of Real Time Systems Symposium*, pp. 27–36, 1982.

[76] K. Padmanabhan and D. H. Lawrie, "Fault tolerance schemes in shuffle-exchange type interconnection networks," *Digest of Papers, FTCS-13*, pp. 71–75, 1983.

[77] J. H. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Trans. Comput.*, vol. C-30, no. 11, pp. 771–780, Nov. 1981.

[78] M. C. Pease III, "The indirect binary n-cube microprocessor array," *IEEE Trans. Comput.*, vol. C-26, no. 5, pp. 458–473, May 1977.

[79] G. Perucca, P. Belforte, E. Garetti, and F. Perardi, "Research on advanced switching techniques for the evolution to isdn and broadband isdn," *IEEE J. Select. Areas Commun.*, vol. SAC-5, no. 8, pp. 1356–1364, Oct. 1987.

[80] G. F. Pfister et al., "The ibm research parallel processor prototype (rp3): Introduction and architecture," *Proc. IEEE 1985 Int. Conf. Parallel Processing*, Aug. 1985.

[81] G. F. Pfister and V. A. Norton, "Hot spot contention and combining in multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-34, no. 10, pp. 943–948, Oct. 1985.

[82] U. V. Premkumar et al., "Design and implementation of the banyan interconnection network in trac," *AFIPS conference Proceedings*, vol. 51, pp. 643–653, 1980.

[83] B. Prince and G. Due-Gundersen, *Semiconductor Memories*, John Wiley & Sons, 1983.

[84] C. S. Raghavendra and A. Varma, "Indra: A class of interconnection networks with redundant paths," *Proc. of Real Time Systems Symposium*, pp. 153–164, 1984.

[85] B. R. Rau, "Interleaved memory bandwidth in a model of a multiprocessor computer system," *IEEE Trans. Comput.*, vol. C-28, no. 9, pp. 678–681, Sep. 1979.

[86] S. M. Reddy and V. P. Kumar, "On fault-tolerant multistage interconnection network," *Proceeding of Parallel Processing Conference*, pp. 155–164, 1984.

[87] D. A. Schoenfeld, "Confidence bounds for normal means under order restrictions, with application to dose-response curves, toxicology experiments, and low dose extrapolation," *Journal of the American Statistical Association*, vol. 81, no. 393, pp. 186–195, March 1986.

[88] M. Schwartz, *Telecommunication Networks*, Addison-Wesley, 1987.

[89] J. P. Shen and J. P. Hayes, "Fault-tolerance of a class of connecting networks," *Proc. 7th Symp. Comput. Arch.*, pp. 61–71, May 1980.

[90] J. P. Shen and J. P. Hayes, "Fault-tolerance of dynamic-full-access interconnection networks," *IEEE Trans. Comput.*, vol. C-33, no. 3, pp. 241–248, Mar. 1984.

[91] K. G. Shin and Y. H. Lee, "Error detection process – model, design, and its impact on computer performance," *IEEE Trans. on Comput.*, vol. C-33, no. 6, pp. 529–540, June 1984.

[92] K. G. Shin and Y.-H. Lee, "Evaluation of error recovery blocks used for cooperating processes," *IEEE Trans. on Software Engineering*, pp. 692–700, 1984.

[93] K. G. Shin, T. H. Lin, and Y.-H. Lee, "Optimal checkpointing of real-time tasks," *IEEE Trans. on Computers*, vol. C-36, no. 11, pp. 1328–1341, Nov. 1987.

[94] H. J. Siegel et al., "Pasm: A partionable simd/mimd system for image processing and," *IEEE Trans. Comput.*, vol. C-30, pp. 934–947, Dec. 1981.

[95] H. J. Siegel and R. J. McMillen, "The multistage cube: A versatile interconnection network," *Computer*, pp. 65–76, Dec. 1983.

[96] D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Equipment Corp., Bedford, MA, 1982.

[97] J. E. Smith, "Measures of effectiveness of fault signature analysis," *IEEE Trans. Comput.*, vol. C-29, no. 6, pp. 510–514, June 1980.

[98] T. B. Smith and J. H. Lala, "Development and evaluation of a fault-tolerant multiprocessor (ftmp) computer volume i: Ftmp principles of operation," Technical report, NASA Contractor Report 166071, May 1983.

[99] D. Steinberg, "Invariant properties of the shuffle-exchange and a simplified cost-effective version of the omega network," *IEEE Trans. Comput.*, vol. C-32, no. 5, pp. 444–450, May 1983.

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

[101] T. H. Szymanski and V. C. Hamacher, "On the permutation capability of multistage interconnection networks," *IEEE Trans. on Comput.*, vol. C-36, no. 7, pp. 810–822, July 1987.

[102] S. Thanawastien and V. P. Nelson, "Interference analysis of shuffle/exchange networks," *IEEE Trans. Comput.*, vol. C-30, no. 8, pp. 545–556, Aug. 1981.

[103] S. Thanawastien and V. P. Nelson, "Optimal fault detection test sequences for shuffle/exchange networks," *Digest of Papers, FTCS-13*, pp. 442–445, 1983.

[104] S. Thanawastien and V. P. Nelson, "Diagnosis of multiple faults in shuffle/exchange networks," *Proc. of Real Time Systems Symposium*, pp. 184–192, 1984.

[105] W. N. Toy, "Fault-tolerant design of local ess processors," *Proceeding of the IEEE*, pp. 1126–1145, Oct. 1978.

[106] N. F. Tzeng and P. C. Y. D. Lawrie, "Fault diagnosis in a multiple-path interconnection network," *Digest of Papers, FTCS-16*, pp. 98–103, 1986.

[107] D. F. Wann and M. A. Franklin, "Asynchronous and clocked control structures for vlsi based interconnection networks," *IEEE Trans. Comput.*, vol. C-32, no. 3, pp. 284–293, Mar. 1983.

[108] J. H. Wensley et al., "Sift: Design and analysis of a fault-tolerant computer for aircraft control," *Proc. of IEEE*, vol. 66, no. 10, pp. 1240–1255, Oct. 1978.

[109] T. W. Williams and K. P. Parker, "Design for testability – a survey," *Proc. IEEE*, vol. 71, no. 1, pp. 98–112, Jan. 1983.

[110] C. L. Wu and T. Y. Feng, "Tutorial: Interconnection networks for parallel and distributed processing," *IEEE*, 1984.

[111] C. L. Wu, W. Lin, and M. C. Lin, "Distributed circuit switching starnet," *Proceeding of Parallel Processing Confernce*, pp. 26–33, 1982.

[112] M. Yasrebi, S. Deshpande, and J. C. Browne, "A comparison of circuit switching and packet switching for data transfer in two simple image processing algorithms," *Proceeding, 1983 Int'l Parallel Processing Conf.*, pp. 25–28, 1983.

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

[114] D. W. L. Yen, J. H. Patel, and E. D. Davidson, "Memory interference in synchronous multiprocessor systems," *IEEE Trans. Comput.*, vol. C-31, no. 11, pp. 1116–1121, Nov. 1982.

[115] P. C. Yew and D. H. Lawrie, "An easily controlled network for frequently used permutations," *IEEE Trans. Comput.*, vol. C-30, no. 4, pp. 296–298, Apr. 1981.

[116] P. C. Yew, T. F. Tzeng, and D. Lawrie, "Distributing hot-spot addressing in large-scale multiprocessors," *IEEE Trans. Comput.*, vol. C-36, no. 4, pp. 388–395, Apr. 1987.

[117] H. Yoon, K. Y. Lee, and M. T. Liu, "Performance analysis and comparison of packet switching interconnection networks," *Proceeding, 1983 Int'l Parallel Processing Conf.*, pp. 542–545, 1987.