

Purdue University Purdue e-Pubs

Department of Electrical and Computer
Engineering Technical Reports

Department of Electrical and Computer
Engineering

10-1-1987

Destination Tag Routing Techniques Based on a State Model for the IADM Network

Darwen Rau
Purdue University

Jose A. B. Fortes
Purdue University

Howard Jay Siegel
Supercomputing Research Center

Follow this and additional works at: <https://docs.lib.purdue.edu/ecetr>

Rau, Darwen; Fortes, Jose A. B.; and Siegel, Howard Jay, "Destination Tag Routing Techniques Based on a State Model for the IADM Network" (1987). *Department of Electrical and Computer Engineering Technical Reports*. Paper 580.
<https://docs.lib.purdue.edu/ecetr/580>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

Destination Tag Routing Techniques Based on a State Model for the IADM Network

Darwen Rau
Jose A. B. Fortes
Howard Jay Siegel

TR-EE 87-39
October 1987

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

Destination Tag Routing Techniques Based on a State Model for the IADM Network¹

Darwen Rau and Jose A. B. Fortes
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

Howard Jay Siegel
Supercomputing Research Center
4380 Forbes Blvd.
Lanham, MD 20706

ABSTRACT

A "state model" is proposed for solving the problem of routing and rerouting messages in the Inverse Augmented Data Manipulator (IADM) network. Using this model, necessary and sufficient conditions for the reroutability of messages are established, and then destination tag schemes are derived. These schemes are simpler, more efficient and require less complex hardware than previously proposed routing schemes. Two destination tag schemes are proposed. For one of the schemes, rerouting is totally transparent to the sender of the message and any blocked link of a given type can be avoided. Compared with previous works that deal with the same type of blockage, the time×space complexity is reduced from $O(\log N)$ to $O(1)$. For the other scheme, rerouting is possible for any type of link blockage. A universal rerouting algorithm is constructed based on the second scheme, which finds a blockage-free path for any combination of multiple blockages if there exists such a path, and indicates absence of such a path if there exists none. In addition, the state model is used to derive constructively a lower bound on the number of subgraphs which are isomorphic to the Indirect Binary N-Cube network in the IADM network. This knowledge can be used to characterize properties of the IADM networks and for permutation routing in the IADM networks.

Index terms - cube network, data manipulator network, destination-tag routing, fault tolerance, interconnection network, multiprocessor, parallel processing, state model.

¹This research was supported in part by the National Science Foundation under Grant DCI-8419745, by the Innovative Science and Technology Office of the Strategic Defense Initiative Organization and was administered through the Office of Naval Research under contract No. 00014-85-k-0588, and by the Supercomputing Research Center under contract MDA904-85-C-5027.

1. Introduction

This paper discusses novel and efficient techniques for routing and rerouting messages in the Inverse Augmented Data Manipulator (IADM) network [9]. These results are based on a new approach, the "state model," which characterizes and correlates the topologies of the IADM and Indirect binary n -cube networks, and leads to efficient exploitation of the redundancy available in the IADM network.

Considerable research has been dedicated to the design of multistage interconnection networks for multiprocessor systems. The class of data manipulator networks, introduced in [3], includes, among others, the Augmented Data Manipulator (ADM) network [17], the IADM network [9] and the Gamma network [13][14]. The IADM network and the ADM network differ only in that the input side of one of them corresponds to the output side of the other and vice versa. The Gamma and the IADM networks are topologically equivalent; however, they use switches of different types. Each 3×3 crossbar switch used in the Gamma network can connect simultaneously all three inputs to all three outputs whereas each switch used in the IADM network can connect only one of its three inputs to one or more of its three outputs. The main interest of this paper is the study of the IADM network; both the one-to-one and permutation routings are considered. The schemes proposed for routing and rerouting messages in the IADM network are also applicable to the Gamma network.

Perhaps the most popular class of multistage networks is the multistage *cube-type* networks such as the Indirect Binary N -Cube [15], Omega [6], Baseline [20], Generalized Cube [18], STARAN flip [2] and a special case of SW-Banyan [4] networks. Among the main advantages of these networks are their very efficient destination tag routing schemes, partitionability, $O(N \log_2 N)$ cost and

ability to pass useful permutations [16]. Some results of this paper are based on characteristics of the Indirect Binary N-Cube network (hereon referred to as the *ICube network*). Since the cube-type networks mentioned above are all topologically equivalent [16][17][20][21], the results in this paper are also relevant to any of them.

The ICube network is composed of $n = \log N$ stages labeled from 0 to $n-1$. Each stage consists of $2N$ connection links and N interchange (switches) boxes. The structure of the network is such that two input links of an interchange box differ only in the i -th bit of their labels; the upper links have a "0" in the i -th bit and the lower links have a "1." Figure 1 illustrates an ICube network of size $N=8$ and two possible states of an interchange box, "straight" and "exchange." Since this paper considers only one-to-one and permutation routing, broadcast states are not shown.

The IADM network is composed of n stages labeled from 0 to $n-1$. Each stage consists of $3N$ connection links and N switching elements. An extra column of switches is appended at the end of the last stage as the output switches and is referred to as stage n . Each switch j at stage i has three output links to switches $(j-2^i) \bmod N$, j and $(j+2^i) \bmod N$ of the succeeding stage. Each switch selects one of its input links and connects it to one or more output links. Figure 2 illustrates an IADM network of size $N=8$.

In a multistage interconnection network, the path connecting the source of a message to its destination is determined by a routing scheme that specifies the switching state of each switch in the path. Routing schemes are considerably simpler for the cube-type networks than for the data manipulator-type networks. In cube-type networks, the interchange box at stage i needs to examine the i -th bit of the binary representation of the destination address of an

incoming message. If the i -th bit is 0, then the upper output of the box is taken. If the i -th bit is 1, the lower output of the box is taken. These schemes are known as *destination tag routing schemes* [6] and are extremely efficient and simple to implement. Unlike cube-type networks, in the IADM and other data manipulator-type networks there are several paths between any source s and destination d ($s \neq d$) and each switching element has at least three switching states. Previously proposed routing schemes [9][10][13] for the IADM network can be thought of as *distance tag schemes*; that is, they require calculation of the distance from source to destination in order to generate routing and rerouting tags. The rerouting schemes in these works are basically finding an alternate representation, which specifies an alternate routing path, for the distance.

McMillen and Siegel [9] proposed three dynamic rerouting techniques for the IADM network for avoiding faulty or blocked $\pm 2^i$ (nonstraight) links. The first and the second schemes require that switches be capable of performing two's complement and $+2^i$ addition operations, respectively. The third scheme requires one extra tag bit which is dynamically updated as the message propagates toward the destination. In [10], the work of [9] was expanded, and a single-stage look-ahead scheme was proposed to avoid certain type of straight link faults. This improved scheme also requires two's complement operations.

Parker and Raghavendra [13] used redundant number representation and proposed an algorithm capable of finding all routing paths, which, effectively, are the redundant number representations for the distance between the source and the destination. Because of the complexity of the algorithm, the cost of computation is prohibitively large so that it is infeasible to implement the algorithm in order to achieve dynamic routing [19]. In addition, although the algorithm can generate all routing tags for any distance, there is no specific work on

rerouting schemes in [13][14].

Lee and Lee [7] proposed signed bit difference tag and destination tag local control algorithms for the ADM and IADM networks that require no computation for the distance between the source and the destination. But their local control algorithms can only find one routing path for each source and destination pair. If the need for rerouting arises, they still resort to the distance tag schemes to find alternate paths.

Past research has shown interesting relationships between data manipulator and cube-type networks. For example, because it is possible to embed the Generalized Cube network in the ADM network [1][17], the set of interconnections implementable by the ADM network is a superset of that of the Generalized Cube network. This fact and the existence of multiple paths between any source s and destination d ($s \neq d$) in the ADM network suggests that the ADM network can be thought of as a fault-tolerant Generalized Cube network. Analogously, the IADM network can be regarded as a fault-tolerant ICube network². Since the permutations realizable by cube-type networks are well studied, the identification of possible embeddings of the ICube network in the IADM network can help characterize the permutation capabilities of this network. A contribution to the precise understanding of these notions is made in this paper; it consists of the identification of a large number of distinct subgraphs of the IADM network that are isomorphic to the ICube network.

Section 2 of this paper introduces a state model to describe and correlate topologies of the ICube network and the IADM network. Necessary and

² While topologically equivalent, the ICube and Generalized Cube I/O ports are addressed so that their inter-relationship is the same as that of the IADM and ADM network, i.e. the input and output sides are interchanged.

sufficient conditions to perform rerouting in the IADM network are derived in Section 3. In Section 4 two routing and rerouting schemes are proposed based on the theory developed in Section 3, together with a discussion of their merits and implementation considerations. A universal rerouting algorithm is proposed in Section 4, which can deal with any combination of multiple link blockages. A class of subgraphs in the IADM network that are isomorphic to the ICube network are identified in Section 6, and it is shown how to reconfigure the IADM network under certain link faults to pass the cube-admissible permutations. Finally, Section 7 summarizes the results presented in this paper.

2. State Model Descriptions for the ICube and IADM Networks

Multistage networks can be modeled as graphs by treating interchange boxes (also called switching elements) and links of the network as nodes and edges of the graph, respectively. Another equivalent graph model [1][8] results if interchange boxes are associated with edges, and links with nodes. Both models are exemplified in Figures 1 and 3 for the ICube network. The IADM network is shown in Figure 2 according to the first model. The design of switches based on both models is discussed in [11]. Clearly, the ICube network in Figure 3 can be regarded as being a subgraph of the IADM network in Figure 2. Henceforth, the second model is always assumed when referring to the ICube network (i.e. Figure 2) and the first model is assumed when dealing with the IADM network.

With respect to these graph models, the nodes and the edges of the graph refer to the switches and the links of the networks, respectively. The number of switches at each stage of a network is denoted N and $n = \log_2 N$ refers to the number of stages. The switches of each stage are labeled from 0 to $N-1$ from the top to the bottom. Any integer j has a binary representation

$j_0 j_1 \cdots j_{n-1}$, where j_{n-1} is the most significant bit and n denotes the number of bits. The notation j_p/q means the bits of j starting at j_p and ending at j_q , where $p \leq q$. Bit \bar{j}_i is 1's complement of bit j_i . Throughout this paper, j and $j+a$, where a is some constant, are reserved to represent labels of switches. Also modulo N arithmetic is assumed, e.g. $j+a$ implies $(j+a) \bmod N$. The notation $j \in S_i$ is used to indicate that a switch j belongs to stage i and $(j' \in S_i, j'' \in S_{i+1})$ is used to represent a link at stage i joining $j' \in S_i$ and $j'' \in S_{i+1}$. A sequence of switches of contiguous stages $(j' \in S_i, j'' \in S_{i+1}, \cdots, j''' \in S_{i+k})$ is used to represent a path from $j' \in S_i$ to $j''' \in S_{i+k}$.

Notation and terminology required for the characterization of network topologies and destination tag routing schemes are introduced next. A switch j of stage i is an *even_i* switch if $j_i = 0$ and an *odd_i* switch if $j_i = 1$. Figure 2 identifies *even_i* and *odd_i* switches at different stages of the IADM network of size $N=8$. Define the functions ΔC_i and $\Delta \bar{C}_i$ that represent connection links at stage i as

$$\Delta C_i(j, t_i) = \begin{cases} 0 & \text{if } j \text{ is an even}_i \text{ switch and } t_i=0, \\ & \text{or if } j \text{ is an odd}_i \text{ switch and } t_i=1 \\ -2^i & \text{if } j \text{ is an odd}_i \text{ switch and } t_i=0 \\ +2^i & \text{if } j \text{ is an even}_i \text{ switch and } t_i=1 \end{cases}$$

$$\Delta \bar{C}_i(j, t_i) = -\Delta C_i(j, t_i)$$

Also, define the functions $C_i(j, t_i) = j + \Delta C_i(j, t_i)$ and $\bar{C}_i(j, t_i) = j + \Delta \bar{C}_i(j, t_i)$. These definitions imply the following lemma of fundamental importance to the results of this paper.

Lemma 2.1

$$C_i(j, t_i) = j_{0/i-1} t_i j_{i+1/n-1}$$

$$\bar{C}_i(j, t_i) = j_{0/i-1} t_i q_{i+1/n-1}$$

for some value of $q_{i+1/n-1}$ which depends on j and t_i .

Proof: If j is an *even* _{i} switch and $t_i = 0$, then $C_i(j, t_i) = \bar{C}_i(j, t_i) = j$. If j is an *odd* _{i} switch and $t_i = 1$, then $C_i(j, t_i) = \bar{C}_i(j, t_i) = j$. If j is an *odd* _{i} switch and $t_i = 0$, then $C_i(j, t_i)$ results from subtracting 1 from j_i . Since j is an *odd* _{i} switch, $j_i = 1$, no borrow is generated and all remaining bits of j are unchanged; however, $\bar{C}_i(j, t_i)$ adds 1 to j_i , changing the i -th bit to 0 and altering some of the bits in positions $i+1, \dots, n-1$ due to carry propagation. Similar reasoning applies when j is an *even* _{i} switch and $t_i = 1$. \square

The notation and terminology just introduced can now be used to describe the networks of interest in this paper. The following description for a network in terms of ΔC_i , $\Delta \bar{C}_i$, C_i and \bar{C}_i is called the *network state model*.

The ICube network is composed of n stages labeled from 0 to $n-1$. Each stage consists of $2N$ links and N switches. An extra column of switches is appended at the end of the last stage as the output switches (Figure 3) and is denoted S_n . A switch $j \in S_i$ is connected to switches $C_i(j, t_i) \in S_{i+1}$, for $0 \leq i \leq n-1$, $0 \leq j \leq N-1$, and $t_i = 0$ or $t_i = 1$. When using destination tags, switch $j_i \in S_i$ routes a message to switch $C_i(j, d_i) \in S_{i+1}$ where d_i is the i -th bit of the address of the message destination.

The IADM network is composed of n stages labeled from 0 to $n-1$. Each stage consists of a column of N switches and $3N$ connection links. An extra column of switches is appended at the end of the last stage as the output switches and is denoted S_n . A switch $j \in S_i$ is connected to switches $C_i(j, t_i) \in S_{i+1}$ and $\bar{C}_i(j, t_i) \in S_{i+1}$ for $0 \leq i \leq n-1$, $0 \leq j \leq N-1$, and $t_i = 0$ or $t_i = 1$. In other words, three links connect a switch $j \in S_i$ to the switches $(j-2^i)$,

j and $(j+2^i)$ at stage $i+1$. Sometimes $+2^i$ and -2^i are used to represent links $(j \in S_i, (j+2^i) \in S_{i+1})$ and $(j \in S_i, (j-2^i) \in S_{i+1})$, respectively. The terms a *straight link* refers to link $(j \in S_i, j \in S_{i+1})$ and a *nonstraight link* refers to links $\pm 2^i$.

According to the model, two types of switches, *even_i* and *odd_i*, are required in the IADM and ICube networks. Figure 4 illustrates the connection links of a pair of *even_i* and *odd_i* switches for an ICube and an IADM network of size $N=8$. The ΔC_i function describes the ICube connections. For the IADM network, the connection links can be described by the union of the functions ΔC_i and $\Delta \bar{C}_i$. In practice, *even_i* and *odd_i* switches can be identical and easily programmed (at power-up or system configuration time) to behave differently.

There are two possible routing behaviors (or states) for each switch in an IADM network. A switch is said to be in *state C* if the routing is decided in accordance with the function $C_i(j, t_i)$ and it is in the *state \bar{C}* if the function $\bar{C}_i(j, t_i)$ applies. On the whole, the link on which a message is routed depends on whether the switch is an *even_i* or *odd_i* switch, in state *C* or \bar{C} , and the value of tag bit t_i . Also the term *state of the network* is used to denote collectively the states of all switches in the network.

The notion of switch state is only conceptual; it can be implemented by designing the switches with actual logic states as well as by using tags with n added bits specifying the states of the switches on the routing path. In Section 4, these and other aspects of the actual implementation of the proposed schemes are discussed in detail.

3. Theory behind the State-Based Destination Tag Routing Schemes

Based on the framework developed in Section 2, routing problems in the IADM network are now examined. It is clear that when every switch in the IADM network is in state C , the IADM network behaves like an ICube network and, therefore, the destination address $d_{0/n-1}$ can be used as a routing tag, i.e. $t_i = d_i$. More generally, the following theorem can be proven.

Theorem 3.1 Let $d = d_{0/n-1}$ be the destination in the IADM network to which a message is to be sent. Then $t = d_{0/n-1}$ is the unique destination routing tag to the destination d regardless of state of the IADM network.

Proof: Consider an arbitrary tag $f_{0/n-1}$ and assume that the IADM network is in an arbitrary state. Let $t_{0/n-1} = f_{0/n-1}$. Then each switch will route the incoming message to either $C_i(j, f_i)$ or $\bar{C}_i(j, f_i)$. From Lemma 2.1, it can be reasoned by induction that, at stage i , $(C_i(j, f_i))_{0/i} = (\bar{C}_i(j, f_i))_{0/i} = f_{0/i}$; at the last stage, $C_{n-1}(j, f_{n-1}) = \bar{C}_{n-1}(j, f_{n-1}) = f_{0/n-1}$. Thus the address of the destination of the message is the same as the routing tag. This proves both the validity and the uniqueness of $d_{0/n-1}$ as a routing tag. \square

It is implicit in the reasoning underlying Theorem 3.1 that any link on a given path results from the appropriate choice of the state of the corresponding switch, i.e. the use of "link" $\Delta C_i(j, t_i)$ results from setting $j \in S_i$ to state C and the use of "link" $\Delta \bar{C}_i(j, t_i)$ results from setting $j \in S_i$ to state \bar{C} . Thus, given a path to the destination d , there is at least one network state for which the use of d as the destination tag results in the routing of a message through that path.

The implication of Theorem 3.1 is that the use of a state model for the IADM network reduces the problem of finding alternate routing paths to that of

controlling the states of the switches in the network. Capitalizing on this idea, the following theorems show how alternate routing paths can be found in order to evade blockages in the network. A *straight link blockage* occurs if a straight link on the routing path is faulty or busy. A *nonstraight link blockage* is defined analogously. The third type of blockage, called *double nonstraight link blockage*, occurs if both nonstraight output links of a switch in the routing path are faulty or busy. A *switch blockage* occurs if the switch itself is busy or faulty. A switch blockage has the same effect as blocking all of the switch's input links and can be transformed into a link blockages problem accordingly. The discussion on rerouting in this paper is concerned only with link blockages.

Theorem 3.2 In the IADM network, a change of the state of switch $j \in S_i$ results in a different routing path to a destination d if and only if a nonstraight output link of j is used on the original routing path to d . Moreover, the other nonstraight output link of j is used on the new path.

Proof: Changing the state of j implies that the "link" $\Delta C_i(j, t_i)$ is used instead of $\Delta \bar{C}_i(j, t_i)$ or vice versa. However, if $\Delta C_i(j, t_i) = 0$ then $\Delta \bar{C}_i(j, t_i) = 0$ (i.e. both use a straight link) and vice versa. \square

With regard to the rerouting schemes proposed in this paper, the implications of Theorem 3.2 are twofold. First, the "if" part of the theorem implies that dynamic rerouting for a nonstraight link blockage can be achieved by changing the state of the switch whose output is the nonstraight link, which is equivalent to rerouting the message through the oppositely signed nonstraight link connected to the same switch. Thus, the same subset of destinations is reachable from the two switches whose input links are the two oppositely signed nonstraight links. Second, the "only if" part of the theorem implies that dynamic rerouting for a straight link blockage is impossible. This is true in

general since every routing path in the IADM network can be the result of setting the network to some state. Moreover, if a path from stage i' to stage i'' consists of all straight links connecting $j \in S_i$ and $j \in S_{i+1}$, $i' \leq i < i''$, then there exist no alternate routing paths from $j \in S_{i'}$ to $j \in S_{i''}$ for otherwise there would exist an alternate routing path branching from $j \in S_{i'}$ and ending at the destination. The only resort, if any at all, to bypass the straight link blockage is to backtrack to a switch connected to a nonstraight link on the routing path at some preceding stage and to reroute from that switch. It remains to show that an alternate routing path always exists, provided that such a nonstraight link exists. In fact, the existence of an alternate routing path partly results from Theorem 3.2, as stated in the next theorem. Figure 5 illustrates the situation in Theorem 3.3.

Theorem 3.3 Consider a routing path in the IADM network to a destination d that contains a blocked straight link at stage i . There exists at least one network state which results in an alternate routing path that avoids the same straight link blockage at stage i if and only if the original routing path to d contains a nonstraight link at stage $i-k$ for some k , $i \geq k > 0$.

Proof: See Appendix A1. \square

Previous work [7][9][13] implies only the "if" part of the theorem, i.e. the possibility of using nonstraight link of opposite sign in order to reroute a message in the case of a nonstraight link failure. However, the "only if" part of the theorem also implies that, in addition, it is not possible to devise a new rerouting scheme capable of avoiding a backtracking (or look-ahead) mechanism in order to deal with straight link blockages.

From Theorem 3.2, (for a given source/destination pair) if the straight output link of a switch is on some routing path, both nonstraight output links of

the switch cannot be used for routing; if one of the nonstraight output links of a switch is on some routing path, the other nonstraight link of the switch is also on another routing path and the straight link of the switch cannot be used for routing. So for a given switch, the output link blockages that affect paths from a given source to a given destination can only be (a) a nonstraight link blockage, (b) a straight link blockage or (c) the double nonstraight link blockage.³ Theorem 3.2 can be used to avoid case (a) a nonstraight link blockage and Theorem 3.3, case (b), a straight link blockage. If case (c) occurs, then Theorem 3.2 cannot be used to find a rerouting path. A backtracking scheme proposed later in Corollary 4.2 based on Theorem 3.3 can be adapted to overcome this type of blockage. The adapted backtracking scheme is based on Theorem 3.4, which is illustrated in Figure 6.

Theorem 3.4 Consider a routing path in the IADM network to a destination d that contains a switch at stage i whose both nonstraight output links are blocked. There exists at least one network state which results in an alternate routing path that avoids the same blocked nonstraight links at stage i if and only if the original routing path to d contains a nonstraight link at stage $i-k$ for some $k, i \geq k > 0$.

Proof: See Appendix A1. \square

³Physically it is possible to have any combination of blockages of the output links of a given switch. However, the possible routing paths for a given source/destination pair can be affected by either a straight link blockage or a double nonstraight link blockage in a given switch but never both types of blockage.

4. State-Based Routing and Rerouting Schemes

In this section, routing and rerouting schemes are discussed based on the theory developed in Section 3. As mentioned earlier, the novelty of the ideas in this paper lies in the state model of the routing behavior of each switch. In previously proposed approaches, routing is determined solely by tag bits. According to the state model, the switching action of each network element is conceptually determined by its relative position (i.e. an *even_i* or *odd_i* switch), its state (i.e. C or \bar{C}) and a destination tag bit (i.e. 0 or 1) (Figure 4). This conceptual separation of routing information makes it possible to devise the simple routing schemes described in this section.

In the first scheme, each switch is initially set up to behave as an *odd_i* or *even_i* switch. In addition, each switch can dynamically be set to one of the logical states C or \bar{C} . In other words, this scheme corresponds to a direct implementation of the conceptual view of switch states. Destination tags are used and, according to Theorem 3.1, the state of the network is transparent to the sender of the message since it only affects the path of the message and not its destination. Consequently, rerouting is also transparent in the sense that it results from a change in the network state. In practice, the implementation can be such that, for instance, state C (or \bar{C}) is used as the default state for each switch in the IADM network and the switch regards the other nonstraight link as a spare link for rerouting; if a nonstraight blockage is detected, then the switch changes state to \bar{C} (or C) so that the spare link is used instead. This scheme is called the *Self-Repairing State-Based Destination Tag (SSDT)* scheme.

Rerouting is useful not only when one nonstraight link in a switch is faulty or busy, but also if both nonstraight links are busy. For example, when considering a packet switching environment, rerouting may be desirable as a means of

balancing the message load throughout the network. The scheme proposed here is well suited for this purpose. Assume that each nonstraight link has an associated buffer (queue). When both nonstraight links are busy due to message traffic congestion, a switch can choose which nonstraight buffer to assign a message to (i.e. which state to associate with that queued message), based on the number of messages present in the buffers in order to evenly distribute the message load to the nonstraight links.

The proposed SSDT scheme has the advantages that it uses simple n -bit destination tags and is capable of rerouting messages when blockages occur in nonstraight links. In addition, rerouting of a message is transparent to its sender since the path of the message is determined by the state of the network. For a given destination tag, the routing behavior of each switch on a possible path is determined by the state of the switch, i.e. the SSDT scheme is fully distributed and rerouting is done dynamically. Each switch requires a negligible amount of extra hardware for the detection of blocked links and the representation of two possible states.

The second scheme is called the *Two-Bit State-Based Destination Tag (TSDT)* scheme and it uses $2n$ -bit routing tags, which specify both the destination of the message and the states of switches on the corresponding path. The TSDT scheme has the advantage that rerouting is possible when blockages occur for straight as well as nonstraight links.

As with the first scheme, the TSDT scheme assumes that each switch is appropriately initialized to behave as an *odd_i* or *even_i* switch. Each "digit" of the routing tag is represented by two bits b_{n+i} and b_i , called the *state bit* and the *destination bit*, respectively. For this scheme, the state of a switch of stage i is specified by b_{n+i} : if $b_{n+i}=0$, the switch is in state C and if $b_{n+i}=1$, the

switch is in state \bar{C} . For all i , $0 \leq i \leq n-1$, $b_i = d_i$. In general, if j is an *even* _{i} switch, $b_i b_{n+i} = 00$ and $b_i b_{n+i} = 01$ direct the message through a straight link, $b_i b_{n+i} = 10$ through link $+2^i$ and $b_i b_{n+i} = 11$ through link -2^i ; if j is an *odd* _{i} switch, $b_i b_{n+i} = 10$ and $b_i b_{n+i} = 11$ directs the message through a straight link, $b_i b_{n+i} = 01$ through link $+2^i$ and $b_i b_{n+i} = 00$ through link -2^i . In general, given a switch, the destination bit specifies use of a straight link or a non-straight link while the state bit determines the choice of the positive or the negative link (if the chosen link is a nonstraight link). Since state information is carried by the routing tag, switches are not required to determine and remember their own states, i.e. the design of the switches does not need to implement the logic states C and \bar{C} .

From Theorem 3.2, a nonstraight link blockage at stage i can be bypassed conveniently by complementing the i -th state bit while the destination bits remain unchanged. For convenience of reference, this is restated in terms of the TSDT scheme as Corollary 4.1 below.

Corollary 4.1 Let $b_{n/2n-1}$ and $b'_{n/2n-1}$ be the state bits of the routing tag and the rerouting tag, respectively, for the IADM network. In order to bypass a nonstraight link blockage at stage i , state bit b_{n+i} needs to be changed to \bar{b}_{n+i} . That is, $b'_{n/2n-1} = b_{n/n+i-1} \bar{b}_{n+i} b_{n+i+1/2n-1}$. \square

Figure 7 illustrates an example of routing from $s = 1$ to $d = 0$ in an IADM network of size $N = 8$. Let $b_{0/5} = 000000$ be the routing tag and $b'_{0/5}$ and $b''_{0/5}$ denote the rerouting tags. The original tag $b_{0/5} = 000000$ specifies the path $(1 \in S_0, 0 \in S_1, 0 \in S_2, 0 \in S_3)$. If $(1 \in S_0, 0 \in S_1)$ is blocked, the rerouting tag $b'_{0/5} = 000100$ is obtained by complementing b_3 , and link $(1 \in S_0, 2 \in S_1)$ is used for rerouting. This tag specifies the path $(1 \in S_0, 2 \in S_1, 0 \in S_2, 0 \in S_3)$. If $(2 \in S_1, 0 \in S_2)$ is also blocked, the rerouting tag $b''_{0/5} = 000110$ results from

complementing b'_4 , and link $(2 \in S_1, 4 \in S_2)$ is used for rerouting. This tag specifies the path $(1 \in S_0, 2 \in S_1, 4 \in S_2, 0 \in S_3)$.

As discussed in Section 3, a straight link blockage and a double nonstraight link blockage cannot be overcome easily; implementing a backtracking (or look-ahead) mechanism is a must in order to evade these types of blockages. Since all links in the routing path from stage $i-k+1$ to stage i consist of only straight links, backtracking of at least k stages is required to find the switch from which an alternate routing path branches. That is, at least k state bits need to be considered for change. Due to the similarity between Theorems 3.3 and 3.4, the TSDT schemes for finding the rerouting paths from Theorems 3.3 and 3.4 are exactly the same, which is stated as Corollary 4.2.

Corollary 4.2 Let $b_{n/2n-1}$ and $b'_{n/2n-1}$ be the state bits of the routing tag and the rerouting tag, respectively, for a source/destination pair in the IADM network. Let $i-k$ be the largest stage number for $i \geq k > 0$ such that a switch at stage $i-k$ is connected to a nonstraight link on the routing path. In order to bypass a straight link blockage or a double nonstraight link blockage at stage i , only state bits $b_{n+(i-k)/n+i-1}$ need to be changed; (i) $b'_{n/n+(i-1)} = b_{n/n+(i-k)-1} \bar{d}_{i-k/i-1}$ if the nonstraight link at stage $i-k$ of the original path is link -2^{i-k} , and (ii) $b'_{n/n+(i-1)} = b_{n/n+(i-k)-1} d_{i-k/i-1}$ if the nonstraight link at stage $i-k$ of the original path is link $+2^{i-k}$. The state bits $b'_{n+i/2n-1}$ have arbitrary values in both cases.

Proof: See Appendix A1. \square

The example in Figure 7 can be used to illustrate the TSDT scheme for (a) a straight link blockage and (b) a double nonstraight link blockage. (a) Again the tag $b_{0/5} = 000000$ specifies a path $(1 \in S_0, 0 \in S_1, 0 \in S_2, 0 \in S_3)$. If the straight link $(0 \in S_1, 0 \in S_2)$ is blocked, the rerouting tag can be 000110 which

specifies path $(1 \in S_0, 2 \in S_1, 4 \in S_2, 0 \in S_3)$ by having $b'_{3+0}b'_{3+1}b'_{3+2} = \bar{d}_0\bar{d}_1b_{3+2} = 110$. Since state bits $b'_{3+1}b'_{3+2}$ can be arbitrary, 000100, for example, is also a valid rerouting tag; it specifies path $(1 \in S_0, 2 \in S_1, 0 \in S_2, 0 \in S_3)$. (b) Let the tag $b_{0/5} = 000110$ specifies a path $(1 \in S_0, 2 \in S_1, 4 \in S_2, 0 \in S_3)$. If both nonstraight output links of $4 \in S_2$ are blocked, the rerouting tag $b'_{0/5}$ can be 000100 which specifies path $(1 \in S_0, 2 \in S_1, 0 \in S_2, 0 \in S_3)$ by having $b'_{3+0}b'_{3+1}b'_{3+2} = b_{3+0}d_1d_2$. Since state bits b'_{3+2} can be arbitrary, 000101 is also a valid rerouting tag which also specifies the same path.

The rerouting path computed from Corollary 4.2 is blockage-free from stage 0 to stage i . While the rerouting path is different from the original routing path from stage $i-k$ to stage i , the routing path from stage 0 to $i-k-1$ remains the same. This results from the fact that backtracking always proceeds backward along the original path until it stops at stage $i-k$, and the rerouting path only changes course from stage $i-k$ onwards. Although state bits $b_{n+i/2n-1}$ remain unchanged, the routing path from stage i to $n-1$ may still be altered due to the changes from stage $i-k$ to i . For example, in Figure 5, the switch on the original routing path at stage $i+1$ is $j \in S_{i+1}$ whereas the switch on the rerouting path at stage $i+1$ may be $(j+2^{i+1}) \in S_{i+1}$, which may further induce changes at higher-order stages.

In the TSDT scheme, the tag can be computed by the message sender which is assumed to know the location of faulty links and switches in the network. Thus, rerouting is transparent to the switches in the sense that the tag computed by the sender of the message simply avoids the usage of faulty links and switches. Therefore switches do not require any extra hardware for rerouting purposes. An alternative is to implement dynamic rerouting for the TSDT

scheme. Since backtracking is indispensable for avoiding a straight link blockage, it is required that each switch can detect the inaccessibility of any output port (connected to a switch at the next stage) and signal the presence of the blockage back to the switches of previous stages [10][12]. Whether rerouting is done by the sender or dynamically is an implementation decision which depends on how many stages of backtracking are allowed. When the sender computes the tag, it must be able to identify and track the switches and links on the corresponding routing and rerouting paths (the next paragraphs explain how this is done). If any of the switches or links in the path is known to the sender as being faulty, then the sender computes another tag by changing the state bits as described in Section 5.

Locating the switches on the routing path is straightforward. For a given source s and a destination d , the initial routing path can be specified by setting state bits $b_{n/2n-1} = 0_{n/2n-1}$ (a string of n 0's), equivalent to setting every switch in the IADM network to state C . Then every switch on the original path has label $d_{0/i-1}s_{i/n-1} \in S_i$, $0 \leq i \leq n-1$, since now the IADM network functions like an ICube network [6][15].

To find the switches on the rerouting path, let $j \in S_i$ be the switch whose output link is blocked. First consider the case where the blocked link is a non-straight link. It may be an (a) positive or (b) negative link. In case (a) the switch at stage $i+1$ reached by the positive link is $(j+2^i) \in S_{i+1}$ and, from Corollary 4.1, rerouting can be done through switch $(j-2^i) \in S_{i+1}$. In case (b) the switch at stage $i+1$ reached by the negative link is $(j-2^i) \in S_{i+1}$ and, from Corollary 4.1, rerouting can be done through switch $(j+2^i) \in S_{i+1}$. Let the switch at stage $i+1$ on the rerouting path be $w_{0/n-1}$. The state bits $b_{n+(i+1)/n-1}$ remain intact (equal to 0's) because it corresponds to having every switch from

stage $i+1$ to $n-1$ remain in state C so that the IADM network from stage $i+1$ to $n-1$ can emulate the ICube network from stage $i+1$ to $n-1$. Thus, the bits l , $i+1 \leq l \leq n-1$, of the label of a switch on the rerouting path are $w_{l/n-1}$. From Lemma 2.1, bits 0 to $l-1$, $1 \leq l \leq i+1$, of the label of a switch on a path to destination $d_{0/n-1}$ must be $d_{0/l-1}$. Hence the switch on the rerouting path from stage $i+1$ to $n-1$ has label $d_{0/l-1}w_{l/n-1}$, $i+1 \leq l \leq n-1$.

Next consider the case where the blockage of $j \in S_i$ is a straight link blockage or a double nonstraight link blockage so that backtracking is necessary. There are two sub-cases for each type of blockage: (i) the nonstraight link found in backtracking is a negative link and (ii) it is a positive link. Here only sub-case (i) of the straight link blockage is considered; the other cases can be dealt with similarly. From the proof of Corollary 4.2 (case (i) only), the switch on the rerouting path is $(j+2^l) \in S_l$, $i-k \leq l \leq i$. The switch of stage $i+1$ on the rerouting path is $j \in S_{i+1}$ if $b'_{n+i} = 0$ and $j \in S_{i+1}$ is an *odd_i* switch or if $b'_{n+i} = 1$ and $j \in S_{i+1}$ is an *even_i* switch, and is $(j+2^{i+1}) \in S_{i+1}$ if $b'_{n+i} = 0$ and $j \in S_{i+1}$ is an *even_i* switch or if $b'_{n+i} = 1$ and $j \in S_{i+1}$ is an *odd_i* switch. The identification of switches on the rerouting path from stage $i+1$ to $n-1$ is done as in the case of a nonstraight link blockage described above.

The blocked link can be represented by the two switches joined by the link. Since every switch on the original routing path and the rerouting paths can be easily identified as described above, it can be readily determined whether or not the blocked link is on the current path.

In summary, for both SDT schemes, the binary representation of the destination address can be used directly as the routing tag. In the SSDT scheme, rerouting tags are not needed and in the TSDT scheme, rerouting tags result from simple bit complementing operations. In terms of complexity of the

computation for a rerouting tag, the SSDT scheme and the TSDT scheme for one instance of nonstraight link blockage require time×space complexity $O(1)$; an improvement over previous proposed schemes [9] dealing with rerouting for a nonstraight link blockage that require time×space complexity $O(\log N)$. In [10] a single-stage look-ahead scheme for rerouting of a straight link blockage was proposed; it requires use of two's complement to compute the positive and negative dominant tags so that the scheme has time×space complexity of $O(\log N)$. Note that the single-stage look-ahead rerouting scheme is valid only for some cases of the straight link blockage; it cannot be applied to any case of the straight link blockage. From Corollary 4.2, k -stage backtracking is needed for a straight link blockage and k bits of the state bits needs to be changed; thus the complexity of the TSDT scheme for a nonstraight link is $O(k)$. If only single-stage backtracking (corresponds to single-stage look-ahead) is necessary, rerouting can be done dynamically and the complexity is $O(1)$, an improvement over the scheme in [10].

5. A Universal Rerouting Algorithm for Multiple Blockages

The TSDT scheme can be applied to not only one instance of some blockage, but also can be applied repetitively each time a new blockage is encountered as the message propagates along. This section considers the derivation of an algorithm to deal with any case of multiple blockages. The backtracking schemes proposed in Corollary 4.2 find a rerouting path for a straight link blockage and a double nonstraight link blockage. Nevertheless, it is possible that blockages also exist on the rerouting path; then further backtracking to a lower-order stage is needed. Since this phenomenon can recur, repeated backtracking may be necessary due to blockages on the rerouting paths. The

algorithm BACKTRACK described next performs iterated backtracking to find an alternate routing path. It underlies a universal rerouting algorithm (called REROUTE) to be shown later that can find a routing path, if there exists any, to bypass multiple blockages in the network.

The inputs to algorithm BACKTRACK are the current routing path P , the stage number i where a blockage occurs, and state bits $b'_{n/2n-1}$ representing path P . The algorithm returns updated values of the state bits $b'_{n/2n-1}$ which specify a rerouting path that is blockage-free from stage 0 to stage i if such a rerouting path exists, or returns FAIL if the blockages on the current routing path and the rerouting paths eliminate the possibility of communication between the source and the destination. It is assumed that the blockage on the original routing path at stage i is a straight link blockage or a double non-straight link blockage and $j \in S_i$ is the switch whose output links are the blocked links. Informal explanations for the algorithm will be given following the algorithm and the correctness proof of this algorithm can be found in Appendix A2.

Algorithm BACKTRACK (and REROUTE) presumes existence of the knowledge of all blockages in the network. The network controller is responsible for collecting this information and maintaining a global map of blockages, which is accessible to every sender of the messages in order to compute a path to avoid the blockages. In addition, since it may take several iterations before a blockage-free path can be found or it can be concluded that no blockage-free paths exist, the sender of the message needs to maintain and update the locations of switches on the rerouting path in each iteration.

Algorithm BACKTRACK ($P, i, b'_{n/2n-1}$)

0: q = stage number where a blockage occur.

$q \leftarrow i$.

1: P = the current routing path.

Backtrack on path P from stage q to find a nonstraight link. If no nonstraight link exists at any preceding stage, return(FAIL); otherwise assign to r the stage number where the first nonstraight output link is found.

2: If the nonstraight link at stage r on the routing path is $+2^r$, assign flag *linkfound* value 0; if it is -2^r , assign *linkfound* value 1.

3: If *linkfound* = 0, $b'_{n/2n-1} \leftarrow b'_{n/n+r-1} d_{r/q-1} b'_{n+q/2n-1}$; if *linkfound* = 1, $b'_{n/2n-1} \leftarrow b'_{n/n+r-1} \bar{d}_{r/q-1} b'_{n+q/2n-1}$.

4a: This step applies only when the blockage at stage q on path P is a straight link blockage.

If *linkfound* = 0, set $b'_{n+q} = d_q$; if $((j-2^q) \in S_q, (j-2^{q+1}) \in S_{q+1})$ is blocked, change b'_{n+q} to \bar{d}_q ; furthermore, if $((j-2^q) \in S_q, j \in S_{q+1})$ is also blocked, return(FAIL). If *linkfound* = 1, set $b'_{n+q} = \bar{d}_q$; if $((j+2^q) \in S_q, (j+2^{q+1}) \in S_{q+1})$ is blocked, change b'_{n+q} to d_q ; furthermore, if $((j+2^q) \in S_q, j \in S_{q+1})$ is also blocked, return(FAIL).

4b: This step applies only when the blockage at stage q on path P is a double nonstraight link blockage.

If $((j-2^q) \in S_q, (j-2^q) \in S_{q+1})$ is blocked for *linkfound* = 0, or $((j+2^q) \in S_q, (j+2^q) \in S_{q+1})$ is blocked for *linkfound* = 1, return(FAIL).

5: Let \hat{Q} denotes the part of the rerouting path (specified by the tag in step 3) from stage $r+1$ to q from step 3.

If $\text{linkfound} = 0$, $\hat{Q} = ((j-2^{r+1}) \in S_{r+1}, \dots, (j-2^{q-1}) \in S_{q-1}, (j-2^q) \in S_q)$; if *linkfound* = 1,

$$\hat{Q} = ((j+2^{r+1}) \in S_{r+1}, \dots, (j+2^{q-1}) \in S_{q-1}, (j+2^q) \in S_q).$$

If a blockage occurs on path \hat{Q} , return(FAIL).

- 6: If $linkfound = 0$ and $((j-2^r) \in S_r, (j-2^{r+1}) \in S_{r+1})$ is blocked, or if $linkfound = 1$ and $((j+2^r) \in S_r, (j+2^{r+1}) \in S_{r+1})$ is blocked, go to step 7; else return($b'_{n/2n-1}$).
- 7: $j \leftarrow j+2^r, q \leftarrow r$.
- 8: Backtrack on path P from stage q to find a nonstraight link. If no nonstraight link exists at any preceding stage, return(FAIL); otherwise assign to r the stage number where the first nonstraight output link is found.
- 9: If $linkfound = 0$ and the nonstraight link at stage r is -2^r , or if $linkfound = 1$ and the nonstraight link at stage r is $+2^r$, return(FAIL).
- 10: If $linkfound = 0$, $b'_{n/2n-1} \leftarrow b'_{n/n+r-1} d_{r/q-1} b'_{n+q/2n-1}$; if $linkfound = 1$, $b'_{n/2n-1} \leftarrow b'_{n/n+r-1} \bar{d}_{r/q-1} b'_{n+q/2n-1}$. Go to step 4b.

Step 0 is the initialization step. From Theorems 3.3 and 3.4, an alternate path exists for avoiding a straight link blockage or a double nonstraight link blockage if and only if there exists a nonstraight link at some stage preceding stage r ; step 1 of the algorithm searches backward for such a nonstraight link. If not found, it results in premature termination of the algorithm, reflecting the fact that no alternate paths for rerouting exist. Step 2 is used to differentiate the cases when the nonstraight link at stage r found in the first backtracking is a positive link and when it is a negative link; flag $linkfound$ is assigned 0 for the former and 1 for the latter. If a nonstraight link exists at some stage preceding the blockages, in step 3, Corollary 4.2 is applied to find the stage bits specifying the rerouting path; cases (i) and (ii) in Corollary 4.2 correspond to $linkfound = 1$ and $linkfound = 0$, respectively, and q and r correspond to i

and $i-k$, respectively.

Steps 4a and 4b deal with the link blockage at stage q on the rerouting path computed in step 3. If the blockage of a switch at stage q on path P is a straight link, the possible rerouting links at stage q are two nonstraight links. In step 4a the default link is negative link if $linefound = 0$ and a positive link if $linkfound = 1$. If the default link is blocked, step 4a attempts to reroute the message through the other nonstraight link. If both nonstraight links are blocked, there exist no blockage-free paths. Step 4b applies if the blockage of a switch at stage q on path P is a double nonstraight link blockage. The rerouting path must use a straight link at stage q . If it is also blocked, no blockage-free path exists.

Step 5 checks blockages from stage $r+1$ to stage $q-1$ on the rerouting path; if any blockage falls on \hat{Q} , there exists no blockage-free path. In step 6, if the blockage falls in the link of stage r on the rerouting path, further backtracking is necessary. Otherwise (no blockages on the rerouting path), the algorithm terminates with the state bits specifying the rerouting path. Step 7 updates the stage number q and the switch label j where a blockage on the rerouting path occurs, initiating a new iteration of backtracking. Step 8 is the same as step 1, searching backward at lower-order stages again for a nonstraight link. Step 9 of the algorithm dictates that if the encountered nonstraight link in the first iteration of backtracking is a positive (or negative) link, the nonstraight link found in each subsequent iteration of backtracking must be also a positive (or negative) link; otherwise no blockage-free paths exist. If the condition in step 9 is satisfied, step 10, which is the same as step 3, computes a rerouting path. After the rerouting path is found, the algorithm returns to step 4b, to check for further blockages on the rerouting path.

For each source/destination pair, a link on some routing path for the source/destination pair is called a *participating* link. As a direct result of Theorem 3.2, the set of participating output links of a switch is composed of either its straight output link or both of its nonstraight output links, but never all of them. So the output link blockages of a switch, for a given source/destination pair, can only be a straight link blockage, a nonstraight link blockage, or a double nonstraight link blockage. Algorithm BACKTRACK deals with the first and third kind of blockages, and the second kind of blockage can be overcome by applying Corollary 4.1. Algorithm BACKTRACK and Corollary 4.1 can be used to form a universal algorithm capable of rerouting messages when multiple blockages exist in the IADM network. This algorithm, called REROUTE, returns state bits $b'_{n/2n-1}$ specifying a blockage-free rerouting path if one exists, or returns FAIL otherwise.

Algorithm REROUTE ($P, b'_{n/2n-1}$)

- 0: P = the original routing path.
 $b_{n/2n-1}$ = the routing tag specifying the original routing path.
 $b'_{n/2n-1}$ = the rerouting tag specifying the rerouting path.
 $b'_{n/2n-1} \leftarrow b_{n/2n-1}$.
- 1: Let i be the smallest stage number such that there exists a blockage at stage i on path P . If no blockages occur on path P , return($b'_{n/2n-1}$).
- 2: If the blockage at stage i on path P is a nonstraight link blockage and the other nonstraight link is not blocked, apply Corollary 4.1 to find state bits $b'_{n/2n-1}$ and go to step 4.
- 3: $b'_{n/2n-1} \leftarrow \text{BACKTRACK}(P, i, b'_{n/2n-1})$.

4: Q = the rerouting path specified by state bits $b'_{n/2n-1}$.

$P \leftarrow Q$ and go to step 1.

Step 0 is the initialization step. At the end of each iteration, a blockage-free path from stage 0 to stage i is found. Then a new iteration starts and i is given a new value in order to find a path avoiding the blockages at a higher-order stage. The only terminating conditions for algorithm REROUTE are that a return of FAIL from step 3 indicating that no blockage-free paths exist and the return from step 1 indicating a blockage-free path is found. Algorithm REROUTE is executed iteratively to evade blockages from lower-order to higher-order stages. The correctness of this algorithm follows from the correctness of algorithm BACKTRACK and Corollary 4.1.

6. Permutation Routing and Cube Subgraphs of the IADM Network

The results discussed so far are a consequence of the existence of spare nonstraight links in addition to the ICube network embedded in the IADM network. This section pursues this issue further by showing that there exist multiple distinct subgraphs in the IADM network, each called a *cube subgraph*, that are isomorphic to the ICube network. Two cube subgraphs are considered to be distinct if they differ in at least one link. As mentioned in the introduction of this paper, the cube-type networks have been studied extensively in the literature and shown to be topologically equivalent. Together with results from these studies, the knowledge of how to identify cube subgraphs can help the understanding of the capabilities of the IADM network and be useful for permutation routing in the IADM network.

Since each switch can be in state C or \bar{C} , there are as many as $2^{N \cdot n}$ ($= N^N$) network states, although each does not necessarily generate a unique

permutation. Setting a switch to a certain state indicates that one of its non-straight output links can be used for routing (i.e. it is *active*) while the other cannot. Thus, each network state can be associated with a subgraph of the IADM network which contains only the active links. When all switches in the IADM network are set to state C , the IADM network functions as an ICube network; this network state corresponds a cube subgraph. The constructive derivation of a lower bound for the number of cube subgraphs of the IADM network uses the two basic ideas discussed in the next paragraphs.

Since $+2^{n-1} \equiv -2^{n-1} \pmod{N}$, $C_{n-1}(j, t_{n-1}) = \bar{C}_{n-1}(j, t_{n-1})$, i.e. the state of each switch of stage $n-1$ is irrelevant in the sense that any switch at stage $n-1$ is always connected to the same two switches at stage n . Consequently, given any cube subgraph, there exist $(2^N - 1)$ subgraphs isomorphic to it which differ only in their choices of the nonstraight link $+2^{n-1}$ or -2^{n-1} at stage $n-1$. Therefore, the total number of distinct cube subgraphs is given by the product of 2^N and the number of distinct subgraphs of the IADM network from stage 0 to stage $n-2$ that are isomorphic to the same stages in the ICube network.

The calculation of the number of subgraphs in the first $n-1$ stages uses an idea similar to that proposed in [5] for reconfiguring the DR network so that it performs as a Generalized Cube network. All switches of the IADM network are logically relabeled by adding a constant x , $0 \leq x \leq N-1$ to the original labels, i.e. switch j becomes $j' = j + x$. By setting each switch to be an *even*, or *odd*, switch according to its new label and having all switches be in state C , a cube subgraph results for each relabeling. However, of the N possible subgraphs, only $\frac{N}{2}$ are distinct as far as the first $n-1$ stages are concerned. This result is stated in Theorem 6.1. A graphical interpretation of cube subgraph isomorphism for an IADM network of size $N=8$ is illustrated in Figure 8. In

Figure 8, each physical switch j acts as a logical switch $j' \equiv (j+1) \bmod 8$. The isomorphism to the ICube network can be easily visualized by moving switch 7 to the top of each stage as shown in the figure. Notice that setting some switch to state C according to its logical label may be equivalent to setting the switch to state \bar{C} according its original label. For instance, switch $0 \in S_0$ (logical label 1) is set to state \bar{C} in Figure 8.

Theorem 6.1 There exist at least $\frac{N}{2} \cdot 2^N$ distinct cube subgraphs in the IADM network.

Proof: See Appendix A1. \square

In order to reconfigure the IADM network to one of its cube subgraphs, each switch of stage i , for $0 \leq i \leq n-2$, needs to know the i -th bit of its logical label. This can be done by sending the same logical label to every switch in the same row at system reconfiguration time. Each switch is set as being an *odd_i* or *even_i* switch by examining the i -th bit of the logical label. All switches operate in state C according to its logical label with the exception of those at stage $n-1$ for which different states correspond to different subgraphs.

The results of this section can be used in different ways. One usage is in characterizing a class of permutations performable by the IADM network. Permutations passable by the ICube network are discussed in [15] and adaptable from [6]. Thus, the IADM network can perform all of these permutations plus the same set of permutations with a given x added to both the same source and destination labels, $0 \leq x < \frac{N}{2}$. Another use of the results of this section is that the IADM network can pass the permutations performable by the ICube network when the ICube network embedded in the IADM network experiences nonstraight link failures. This is done by incorporating a reconfiguration

function in the system that reassigns each switch j to $(j+x)$ and reconfiguring the IADM network to a corresponding cube subgraph which does not include the faulty nonstraight links. In [21] it is shown that any of the cube-type networks can pass the permutations performable by the others by incorporating appropriate reconfiguration functions. By the same token, the IADM network with a nonstraight link fault can also pass the permutations performable by the cube-type networks by including these reconfiguration functions in the system.

7. Concluding Remarks

One of the main contributions of this paper is the identification of destination tag routing schemes for the IADM network. They are simpler and more efficient than previously known approaches, thus requiring less complex switches and reducing message communication delays due to routing overhead. In the SSDT scheme rerouting can be done when nonstraight links fail and in the TSDT scheme both the straight and double nonstraight link blockages can be avoided. As for the SSDT scheme, routing and rerouting are transparent to the source and only negligible hardware and time are used by each switch for routing and rerouting purpose. These are considerable advantages over previously proposed schemes which do not use destination tags and require extra hardware or delays of $O(\log N)$ complexity instead of $O(1)$. In addition, previous works all deal only with certain types of blockages. Based on the TSDT scheme, a universal rerouting algorithm is derived, which is capable of avoiding any combination of multiple blockages if there exist a blockage-free path and indicating absence of such a path if there exists none. The rerouting capabilities of the new schemes can be readily used for fault-tolerance and load balancing purposes since they adequately exploit the redundancy available in the IADM

network.

Another contribution of this paper is the constructive derivation of a lower bound on the number of cube subgraphs of the IADM network. While it was previously known that the ICube network is a subgraph of the IADM network, this paper shows that there exist at least $\frac{N}{2} \cdot 2^N$ distinct cube subgraphs. This, combined with previous multistage cube network studies, can help characterize some of the permutations performable by the IADM network. As other use of the subgraph analysis, it is shown how to reconfigure the IADM network under nonstraight link faults to pass the cube-admissible permutations.

Perhaps the most fundamental contribution of this paper is that of the network state model used for the IADM and the ICube networks. The essence of this model is in the recognition that the routing action of each switch is conceptually dependent on its position in the network (topological information), its state (functional information), and the destination of the message (routing information). Topological information is fixed and, when using destination tags, the same can be said of routing information for a given message destination. Consequently, the routing path is solely determined by the state of the network. These basic concepts are applicable to networks other than those considered in this paper; the state model can help devise new designs, solve routing problems, and understand relationships among networks.

References

- [1] D. Agrawal, "Graph Theoretical Analysis and Design of Multistage Interconnection Networks," *IEEE Trans. Computers*, Vol. C-32, No. 7, July 1983, pp. 637-648.

- [2] K. E. Batcher, "The Flip Network in STARAN," *1976 Int'l Conf. Parallel Processing*, Aug. 1976, pp. 65-71.
- [3] T-Y Feng, "Data Manipulating Functions in Parallel Processors and their Implementations," *IEEE Trans. Computers*, Vol. C-23, No. 3, Mar. 1974, pp. 309-318.
- [4] L. R. Goke and G. J. Lipovski, "Banyan Networks for Partitioning Multiprocessor Systems," *1st Ann. Symp. Computer Architecture*, Dec. 1973, pp. 21-28.
- [5] M. Jeng and H. J. Siegel, "A Fault-tolerant Multistage Interconnection Network for Multiprocessor Systems Using Dynamic Redundancy," *6th Int'l Conf. Distributed Computing Systems*, May 1986, pp. 70-77.
- [6] D. H. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Trans. Computers*, Vol. C-24, No. 12, Dec. 1975, pp. 1145-1155.
- [7] D. Lee and K. Y. Lee, "Control Algorithms for the Augmented Data Manipulator Network," *1986 Int'l Conf. Parallel Processing*, Aug. 1986, pp. 123-130.
- [8] M. Malek and W. W. Myre, "A Description Method of Interconnection Networks," *IEEE Tech. Committee Distrib. Process.*, Quart. Vol. 1, Feb. 1981, pp. 1-6.
- [9] R. J. McMillen and H. J. Siegel, "Routing Schemes for the Augmented Data Manipulator Network in an MIMD System," *IEEE Trans. Computers*, Vol. C-31, No. 12, Dec. 1982, pp. 1202-1214.
- [10] R. J. McMillen and H. J. Siegel, "Performance and Fault Tolerance Improvements in the Inverse Augmented Data Manipulator Network," *9th Ann. Symp. Computer Architecture*, Apr. 1982, pp. 63-72.

- [11] R. J. McMillen and H. J. Siegel, "Evaluation of Cube and Data Manipulator Networks," *J. Parallel and Distributed Computing*, Vol. 2, No. 1, Feb. 1985, pp. 79-107.
- [12] K. Padmanabhan and D. H. Lawrie, "A Class of Redundant Path Multistage Interconnection Networks," *IEEE Trans. Computers*, Vol. C-32, No. 12, Dec. 1983, pp. 1099-1108.
- [13] D. S. Parker and C. S. Raghavendra, "The Gamma Network: A Multiprocessor Interconnection Network with Redundant Paths," *9th Ann. Symp. Computer Architecture*, Apr. 1982, pp. 73-80.
- [14] D. S. Parker and C. S. Raghavendra, "The Gamma Network," *IEEE Trans. Computers*, Vol. C-33, No. 4, Apr. 1984, pp. 367-373.
- [15] M. C. Pease, III, "The Indirect Binary n-Cube Microprocessor Array," *IEEE Trans. Computers*, Vol. C-26, No. 5, May 1977, pp. 458-473.
- [16] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*, Lexington Books, D. C. Heath and Company, Lexington, MA, 1985.
- [17] H. J. Siegel and S. D. Smith, "Study of Multistage SIMD Interconnection Networks," *5th Ann. Symp. Computer Architecture*, Apr. 1978, pp. 223-229.
- [18] H. J. Siegel and R. J. McMillen, "The Multistage Cube : A Versatile Interconnection Network," *IEEE Computer*, Vol. 14, Dec. 1981, pp. 65-76.
- [19] A. Varma and C. S. Raghavendra, "On Permutations Passable by the Gamma Network," *J. Parallel and Distributed Computing*, Vol. 3, No. 1, pp. 72-91, Mar. 1986.
- [20] C-L. Wu and T-Y. Feng, "On a Class of Multistage Interconnection Networks," *IEEE Trans. Computers*, Vol. C-29, No. 8, Aug. 1980, pp. 694-702.

- [21] C-L. Wu and T-Y. Feng, "The Reverse-Exchange Interconnection Network," *IEEE Trans. Computers*, Vol. C-29, No. 9, Sept. 1980, pp. 801-811.

Appendix A1

Proof of Theorem 3.3

The "only if" part follows immediately from Theorem 3.2. To prove the "if" part, let $j \in S_i$ be the switch whose straight output link is the blocked link on the routing path and $i-k$ be the largest stage number for $i \geq k > 0$ such that a switch at stage $i-k$ has a nonstraight output link on the routing path. (i) Assume that the nonstraight link at stage $i-k$ found in backtracking is link -2^{i-k} . Clearly, as illustrated in Figure 5, the path $((j+2^{i-k}) \in S_{i-k}, (j+2^{i-k+1}) \in S_{i-k+1}, \dots, (j+2^i) \in S_i, j \in S_{i+1})$ is a rerouting path for path $((j+2^{i-k}) \in S_{i-k}, j \in S_{i-k+1}, j \in S_{i-k+2}, \dots, j \in S_i, j \in S_{i+1})$. (ii) Assume that the nonstraight link at stage $i-k$ found in backtracking is link $+2^{i-k}$; similarly path $((j-2^{i-k}) \in S_{i-k}, (j-2^{i-k+1}) \in S_{i-k+1}, \dots, (j-2^i) \in S_i, j \in S_{i+1})$ is a rerouting path for path $((j-2^{i-k}) \in S_{i-k}, j \in S_{i-k+1}, j \in S_{i-k+2}, \dots, j \in S_i, j \in S_{i+1})$. \square

Proof of Theorem 3.4

The "only if" part again follows immediately from Theorem 3.2. To prove the "if" part, let notations i , $i-k$ and $j \in S_i$ be the same as those in the proof of Theorem 3.3. The proof is illustrated in Figure 6. From Theorem 3.2, $(j-2^i) \in S_{i+1}$ and $(j+2^i) \in S_{i+1}$ can reach the same subset of destinations so that it does not matter which is on the rerouting path. (i) Assume that the nonstraight link at stage $i-k$ found in backtracking is link -2^{i-k} . It is self-explanatory that path $((j+2^{i-k}) \in S_{i-k}, (j+2^{i-k+1}) \in S_{i-k+1}, \dots, (j+2^i) \in S_i, (j+2^i) \in S_{i+1})$ is a rerouting path for both paths $((j+2^{i-k}) \in S_{i-k}, j \in S_{i-k+1}, \dots, j \in S_i, (j+2^i) \in S_{i+1})$ and $((j+2^{i-k}) \in S_{i-k}, j \in S_{i-k+1}, \dots, j \in S_i, (j-2^i) \in S_{i+1})$. (ii) Assume that the nonstraight link at stage $i-k$ found in backtracking is link $+2^{i-k}$; similarly

path $((j-2^{i-k}) \in S_{i-k}, (j-2^{i-k+1}) \in S_{i-k+1}, \dots, (j-2^i) \in S_i, (j-2^i) \in S_{i+1})$ is a rerouting path. Note that the participating input link of $j \in S_i$ may be a non-straight link; however, this is just a special case for $k = 1$. \square

Proof of Corollary 4.2

First two lemmas are presented, which are to be used to prove Corollary 4.2.

Lemma A1.1 In the TSDT scheme, the links $+2^l$ and -2^l connected to a switch $j \in S_l$ are specified by tag bits $b_l b_{n+l} = \bar{j}_l j_l$ and $b_l b_{n+l} = \bar{j}_l \bar{j}_l$ respectively, and the straight link is specified by $b_l b_{n+l} = j_l j_l$ or $b_l b_{n+l} = j_l \bar{j}_l$.

Proof: Follow immediately from the definition for the TSDT scheme. \square

Lemma A1.2 (i) Let $j \in S_l$ and $(j+2^l) \in S_{l+1}$ be two switches joined by a positive nonstraight link $+2^l$ and they are on a path to the destination $d_{0/n-1}$. In the TSDT scheme, the routing tag can be set to $b_l b_{n+l} = d_l \bar{d}_l$ to control routing to send the message from $j \in S_l$ to $(j+2^l) \in S_{l+1}$. (ii) Let $j \in S_l$ and $(j-2^l) \in S_{l+1}$ be two switches joined by a negative nonstraight link -2^l and they are on a path to the destination $d_{0/n-1}$. In the TSDT scheme, the routing tag can be set to $b_l b_{n+l} = d_l d_l$ to control routing to send the message from $j \in S_l$ to $(j-2^l) \in S_{l+1}$.

Proof: Only proof for (i) is given and proof for (ii) is similar. From Lemma 2.1 and the proof for Theorem 3.1, the switch $j' (= j+2^l) \in S_{l+1}$ has the label $j'_{0/n-1} = d_{0/l-1} d_l w_{l+1/n-1}$, where $w_{l+1/n-1}$ depends on network state. So $j'_l = d_l$. Additionally, $j'_l = \bar{j}_l$ because $j' = j+2^l$. Hence $j_l = \bar{d}_l$. By Lemma A1.1, $b_l b_{n+l} = d_l \bar{d}_l$. \square

Proof of Corollary 4.2:

Only proofs of (i) for (a) a straight link blockage and for (b) a double non-straight link blockage are given; proofs of (ii) for cases (a) and (b) are similar. Since the destination bits always remain unchanged, only state bits need to be

considered. (a) This proof first derives the state bits controlling the rerouting path $Q^+ = ((j+2^{i-k}) \in S_{i-k}, (j+2^{i-k+1}) \in S_{i-k+1}, \dots, (j+2^i) \in S_i)$ in Figure 5 (which illustrates the proof of Theorem 3.3). Since the links on path Q^+ are all positive nonstraight links, by Lemma A1.2, $b'_{n+(i-k)/n+i-1} = \bar{d}_{i-k/i-1}$ represents the state bits for path Q^+ . In addition, by Theorem 3.2, the link of stage i on the rerouting path can be either link -2^i $((j+2^i) \in S_i, j \in S_{i+1})$ or link $+2^i$ $((j+2^i) \in S_i, (j+2^{i+1}) \in S_{i+1})$. Thus b'_{n+i} can be 0 or 1. (b) Notice that the rerouting paths from stage $i-k$ to stage i found in Theorem 3.3 and Theorem 3.4 are the same except the link of stage i on the rerouting path is a non-straight link in Theorem 3.3 (Figure 5) and it is a straight link in Theorem 3.4 (Figure 6). By Lemma A1.1, the state bit b'_{n+i} , which specifies the straight link at stage i in Theorem 3.4, can be 0 or 1. So the state bits specifying the rerouting path from stage $i-k$ to stage i are the same as those in (a). $b'_{n+(i+1)/2n-1}$ can be arbitrary because, regardless of the values of $b'_{n+(i+1)/2n-1}$, as long as the destination bits are $b_{0/n-1} = d_{0/n-1}$, the path can reach the destination $d_{0/n-1}$. \square

Proof of Theorem 6.1

Consider two cube subgraphs generated by adding x and y , respectively, to the original labels of all switches of the IADM network. It is shown that $x \bmod \frac{N}{2} \neq y \bmod \frac{N}{2}$ is a sufficient condition for these subgraphs to be distinct in the sense that they differ in at least one link of the first $n-2$ stages (it is also possible to show the necessity of this condition). To prove that the subgraphs are distinct, it is shown that, given the condition above, there exists some physical switch $j^* \in S_{n-2}$ such that $(j^* + x)$ and $(j^* + y)$ differ in their $(n-2)$ -th bit, i.e. the switch with logical label $(j^* + x)$ is an *even_i* switch and the switch with logical label $(j^* + y)$ is an *odd_i* switch, or vice versa. This implies

that a different nonstraight link is used and therefore the subgraphs are distinct. Let the h -th bit of $x_{0/n-2}$ and $y_{0/n-2}$ be the highest order bit such that $x_h \neq y_h$, i.e. $x_{h+1/n-2} = y_{h+1/n-2}$. Here $h \leq n-2$ since only the topology of the IADM network from stage 0 to stage $n-2$ is considered. Without loss of generality, assume that $x_h = 0$ and $y_h = 1$ and let $j^*_{0/n-2} = 0_{0/h-1}1\bar{x}_{h+1/n-2}$ (where $0_{0/h-1}$ is a string of h 0's). Then

$$(j^* + x)_{0/n-2} = x_{0/h-1}(0+1)1_{h+1/n-2} = x_{0/h-1}1_{h/n-3}1 \quad \text{and}$$

$$(j^* + y)_{0/n-2} = y_{0/h-1}(1+1)1_{h+1/n-2} = y_{0/h-1}0_{h/n-3}0$$

differ in the value of their $(n-2)$ -th bit. Therefore there exist $\frac{N}{2}$ distinct cube subgraphs when considering only the topology of the IADM network from stage 0 to stage $n-2$. For each of these $\frac{N}{2}$ cube subgraphs, there exist 2^N subgraphs of the IADM network which differ from it only in the choice of the nonstraight links at stage $n-1$. Thus, the IADM network contains at least $\frac{N}{2} \cdot 2^N$ distinct cube subgraphs. \square

Appendix A2: Proof of Algorithm BACKTRACK

Terminology and two lemmas are introduced first in order to lay the ground for the verification of algorithm BACKTRACK. Given a source and a destination, a switch on some routing path for the source/destination pair is called a *pivot*. Conversely, by the definition of a pivot, a path in the IADM network can reach the destination if and only if it passes through a pivot at each stage. The set of pivots at each stage varies with different source/destination pair and is characterized by the following lemma.

Lemma A2.1 Let \hat{k} be the smallest stage number for which there exists a non-straight link on at least one routing path from a given source $s_{0/n-1}$ to a given destination $d_{0/n-1}$ in the IADM network. For this source/destination pair, there is exactly one pivot at stage k' , $0 \leq k' \leq \hat{k}$, and there exists exactly two pivots at stage k'' , $\hat{k}+1 \leq k'' \leq n-1$. The pivot at stage k' is $d_{0/k'-1}s_{k'/n-1}$. The pivots of stage k'' are $d_{0/k''-1}s_{k''/n-1}$ and either $(d_{0/k''-1}s_{k''/n-1}+2^{k''})$ or $(d_{0/k''-1}s_{k''/n-1}-2^{k''})$.

Proof: By definition of \hat{k} , the routing paths from stage 0 to $\hat{k}-1$ consist of only straight links. From Theorem 3.2, there exists a unique path from stage 0 to stage \hat{k} and, therefore, the set of pivots at stage k' , $0 \leq k' \leq \hat{k}$, consists of exactly one pivot. Existence of exactly two pivots at stage k'' , $\hat{k}+1 \leq k'' \leq n-1$, and that their distance is $2^{k''}$ follow immediately from the single theorem in [13]. Since the IADM network functions like an ICube network when every switch in the IADM network is set to state C , $d_{0/k-1}s_{k/n-1} \in S_k$, $0 \leq k \leq n-1$, is on a routing path [6][15]; the lemma follows. \square

Lemma A2.1 captures a simple characteristic of routing in the IADM network and, for each source/destination pair, it allows the discussion to focus only on the behavior of the pivots at each stage. A pivot is *unreachable* if all its participating input links (defined in Section 3) are blocked, and it is *closed* if all its participating output links are blocked. A pivot of a lower-order stage can be closed due to the closure of pivots at higher-order stages. Likewise, a pivot of higher-order stage can be unreachable due to unreachability of pivots at lower-order stages. From the definition of a pivot, an important lemma which identifies the causes for the absence of blockage-free paths between a source/destination pair is stated as follows.

Lemma A2.2 In the IADM network, for a given source/destination pair, if all pivots of some stage are closed or unreachable, there exist no blockage-free paths for the source/destination pair. \square

Lemmas A2.1 and A2.2 describe the behavior of the switches and the links in the set of routing paths for each source/destination pair. These lemmas make it possible to ignore switches other than pivots and links other than participating links at each stage for a source/destination pair. These results greatly simplify the complexity of rerouting in the IADM network.

The correctness proof for algorithm BACKTRACK consists of two parts. First is that the path found by the algorithm is a valid path leading to the destination and capable of avoiding blockages in the network. Second is that algorithm BACKTRACK always finds a rerouting path if there exists any, which is equivalent to that algorithm BACKTRACK returns FAIL only if there exist no blockage-free paths. To prove these two parts, it requires examination of the conditions that terminate algorithm BACKTRACK.

The rerouting path found by the algorithm can route the message to the destination because the destination bits of the rerouting tags equal to the binary representation of the destination address. The rerouting path's ability to evade blockages is a natural consequence of Corollary 4.2, on which steps 3 and 10, the only steps in the algorithm that generate rerouting tags, are based. Notice that step 6 returns the rerouting tag if the rerouting path found from step 3 or 10 is blockage-free.

The steps that return FAIL are steps 1, 4a, 4b, 5, 8 and 9. Steps 1 and 8 return FAIL because no alternate routing paths exist. Steps 4a, 4b, 5 and 9 return FAIL because the communication between the source and the destination is broken due to the blockages in the network. So it is impossible for a

blockage-free path to exist without algorithm BACKTRACK finding it and not returning FAIL. Validity of steps 1 and 8 was discussed. Therefore, the proof for the second part is complete if steps 4a, 4b, 5 and 9 are verified.

Proof of steps 4a and 4b

In the following discussion for steps 4a and 4b, only the case where *linkfound* = 1 is explored; the cases where *linkfound* = 0 can be treated analogously. In Figure 5 (*linkfound* = 1 and $q = i$), the blockage at stage q on path P is a straight link blockage and the link at stage q on the rerouting path is chosen to be $((j+2^q) \in S_q, (j+2^{q+1}) \in S_{q+1})$ by setting $b'_{n+q} = \bar{d}_q$ (Lemma A1.2). A blockage in $((j+2^q) \in S_q, (j+2^{q+1}) \in S_{q+1})$ can be overcome by rerouting the message through the other nonstraight link $((j+2^q) \in S_q, j \in S_{q+1})$. This is done by complementing b'_{n+q} . If $((j+2^q) \in S_q, j \in S_{q+1})$ is also blocked, links $(j \in S_q, j \in S_{q+1})$, $((j+2^q) \in S_q, j \in S_{q+1})$ and $((j+2^q) \in S_q, (j+2^{q+1}) \in S_{q+1})$ are all blocked, thus both pivots at stage q , $j \in S_q$ and $(j+2^q) \in S_q$, are closed. Hence no blockage-free paths exist. The above explains step 4a. In Figure 6 (*linkfound* = 1 and $q = i$) both nonstraight links of $j \in S_q$ on path P , $(j \in S_q, (j-2^q) \in S_{q+1})$ and $(j \in S_q, (j+2^q) \in S_{q+1})$, are blocked and thus pivot $j \in S_q$ is closed. If $((j+2^q) \in S_q, (j+2^q) \in S_{q+1})$ is also blocked, pivot $(j+2^q) \in S_q$ is also closed. Because both pivots of stage q , $j \in S_q$ and $(j+2^q) \in S_q$, are closed, there exist no blockage-free paths. This explains step 4b. \square

The scope of the correctness proof for steps 6 and 10 is limited to the case where the first nonstraight link found in backtracking is -2^r (*linkfound* = 1) and assumes that the blockage at stage i is a double nonstraight link blockage. Discussions for the cases where link $+2^r$ is the first nonstraight link found in backtracking and where the blockage at stage i is a straight link blockage can be treated analogously.

An interesting property regarding the behavior of the pivots at each iteration of backtracking is discussed here. This is to be used in the correctness proof for steps 5 and 9. The discussions are associated with Figures 5 and 6 for $q = i$ and $r = i - k$. Since the links on path P from stage $r+1$ to $q-1$ are all straight links, by Theorem 3.2, there exist no alternate routing paths from $j \in S_{r+1}$ to $j \in S_q$. So the closure of $j \in S_q$ would effectively close every pivot $j \in S_l$, $r+1 \leq l \leq q-1$. Hence if $j \in S_q$ is closed, every $j \in S_l$, $r+1 \leq l \leq q$, is closed. Due to the closure of $j \in S_{r+1}$, $((j+2^r) \in S_r, j \in S_{r+1})$ is blocked. If $((j+2^r) \in S_r, (j+2^{r+1}) \in S_{r+1})$ is also blocked (step 6), both participating output links of $(j+2^r) \in S_r$ are blocked and thus $(j+2^r) \in S_r$ is closed. After j and q are updated in step 7 (i.e. $j \leftarrow j+2^r$, $q \leftarrow r$ so that $(j+2^r) \in S_r$ becomes $j \in S_q$), the same type of blockage recurs (i.e. both nonstraight output links of $j \in S_q$ are blocked and thus $j \in S_q$ is closed) as that which took place when the algorithm was first entered (i.e. $q = i$) and thus a new iteration of backtracking begins. For convenience of reference, the property described in this paragraph is formally restated as a lemma.

Lemma A2.3 In each iteration of backtracking in algorithm BACKTRACK, on path P every pivot $j \in S_l$, $r+1 \leq l \leq q$, is closed; if $((j+2^r) \in S_r, (j+2^{r+1}) \in S_{r+1})$ is also blocked, $(j+2^r) \in S_r$ is also closed. \square

Beginning from the second iteration of backtracking, the link of stage q on the rerouting path is always a straight link, since the blockage at the onset of each iteration of backtracking is always that both nonstraight output links of $j \in S_q$ are blocked (Figure 6). Hence only step 4b is concerned in checking the blockages of stage q on the rerouting path. As a result, in Figures 5 and 6 ($linkfound = 1$), the links on path P from stage r to stage i consist of only straight links and negative nonstraight links; correspondingly, the links on the

rerouting path from stage r to stage i consist of only straight links and positive nonstraight links. Similarly, for $linkfound = 0$, the links on path P from stage r to stage $i-1$ consist of only straight links and positive nonstraight links; correspondingly, the links on the rerouting path from stage r to stage $i-1$ consist of only straight links and negative nonstraight links.

Proof of step 5

Proof of step 5 is illustrated in Figure 6 for $q = i$ and $r = i-k$. Because of Lemma A2.2, it suffices to show that, in each iteration of backtracking, a double nonstraight link blockage at stage q and an additional link blockage in $((j+2^l) \in S_l, (j+2^{l+1}) \in S_{l+1})$, for some $l, r+1 \leq l \leq q-1$, effectively close pivot $j \in S_{l+1}$ and make $(j+2^l) \in S_{l+1}$ unreachable. From Lemma A2.3, on path P every pivot $j \in S_{l+1}, r+1 \leq l \leq q-1$, is closed. On path \hat{Q} , if a link blockage also occurs in $((j+2^l) \in S_l, (j+2^{l+1}) \in S_{l+1})$, pivot $(j+2^{l+1}) \in S_{l+1}$ becomes unreachable unless $j \in S_l$, the other pivot at stage l , is also connected to $(j+2^{l+1}) \in S_{l+1}$. This would occur only if link $+2^{l+1}$ is a legitimate link at stage l , i.e. $2^{l+1} = 2^n \equiv 0 \pmod{2^n}$ (a straight link). But $l \leq q-1 \leq (n-1)-1$ so that $l+1 \neq n$. $q \leq n-1$ since q is the stage number at which a output link is blocked and stage $n-1$ the last stage that has output links. Because pivot $j \in S_{l+1}$ is closed and $(j+2^{l+1}) \in S_{l+1}$ is unreachable, there exist no blockage-free paths. \square

Proof of step 9

From Lemma A2.3, at the end of each iteration of backtracking, $(j+2^r) \in S_r$ and $j \in S_{r+1}$ are closed. After a new iteration of backtracking starts and step 7 is executed, $(j+2^r) \in S_r$ is relabeled as $j \in S_q$ and $j \in S_{r+1}$ is relabeled as $(j-2^q) \in S_{q+1}$. So the condition that $j \in S_q$ and $(j-2^q) \in S_{q+1}$ are both closed is a priori in the beginning of the new iteration. Since $(j-2^q) \in S_{q+1}$, one of the

pivots at stage $q+1$, is closed, any rerouting path must pass through $(j+2^q) \in S_{q+1}$, the other pivot at stage $q+1$. It is shown below that such a rerouting path does not exist if the nonstraight link at stage r found in backtracking is $+2^r$. The proof is illustrated in Figure 9. The current routing path is $((j-2^r) \in S_r, j \in S_{r+1}, \dots, j \in S_q, (j-2^q) \in S_{q+1})$ and there exists a rerouting path $((j-2^r) \in S_r, \dots, (j-2^{q-1}) \in S_{q-1}, (j-2^q) \in S_q, (j-2^q) \in S_{q+1})$. Thus $(j-2^q) \in S_q$ and $j \in S_q$ are the two pivots at stage q . Since pivot $j \in S_q$ is closed, any rerouting path must pass through pivot $(j-2^q) \in S_q$. But $(j-2^q) \in S_q$ is not connected to $(j+2^q) \in S_{q+1}$ since link $+2^{q+1}$ is not a legitimate link at stage q , $0 \leq q < n-1$. Therefore, no paths that pass through $(j-2^q) \in S_q$ and $(j+2^q) \in S_{q+1}$ exist. Note that although further backtracking to a still lower-order stage is possible, as long as the nonstraight link at stage r found in backtracking is $+2^r$, the two pivots at stage q never change. That is, further backtracking will not result in a path that passes through $(j-2^q) \in S_q$ and $(j+2^q) \in S_{q+1}$. \square

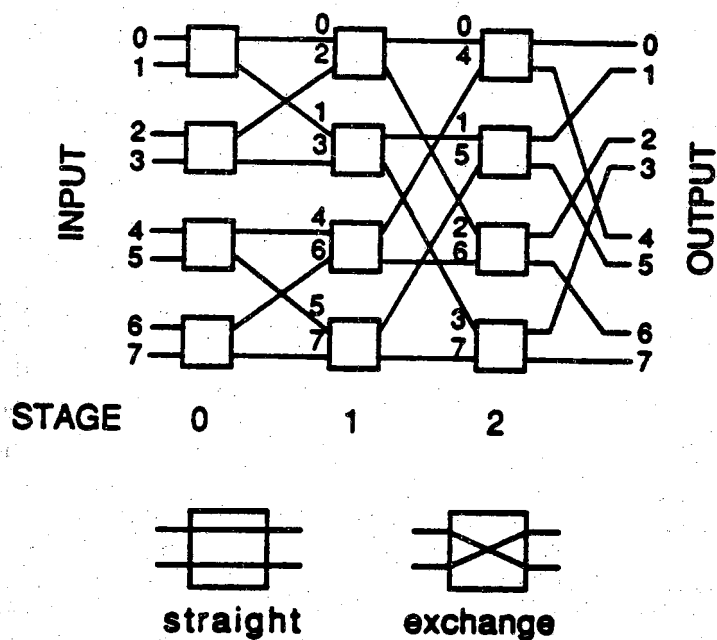


Figure 1. The Indirect Binary N-Cube (ICube) network for $N=8$ (according to the first graph model); two possible states for each box are shown (i.e. straight and exchange).

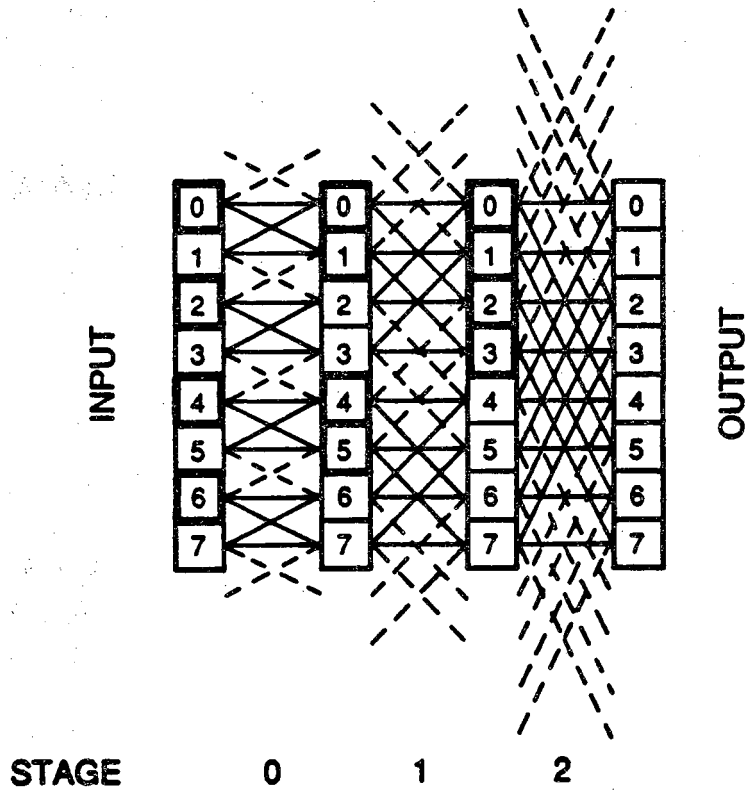


Figure 2. The IADM network for $N=8$ (according to the first graph model); *even*, and *odd*, switches, $0 \leq i \leq 2$, are enclosed with bold and regular edges respectively. The solid edges (links) show the ICube subgraph.

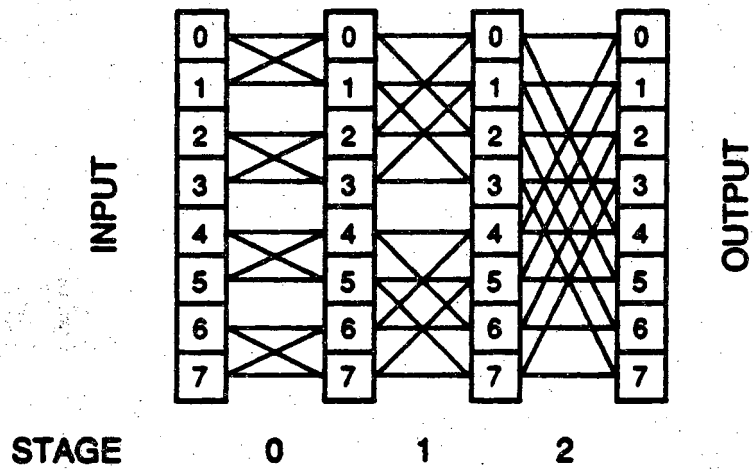


Figure 3. The Indirect Binary N-Cube (ICube) network for N=8 (according to the second graph model).

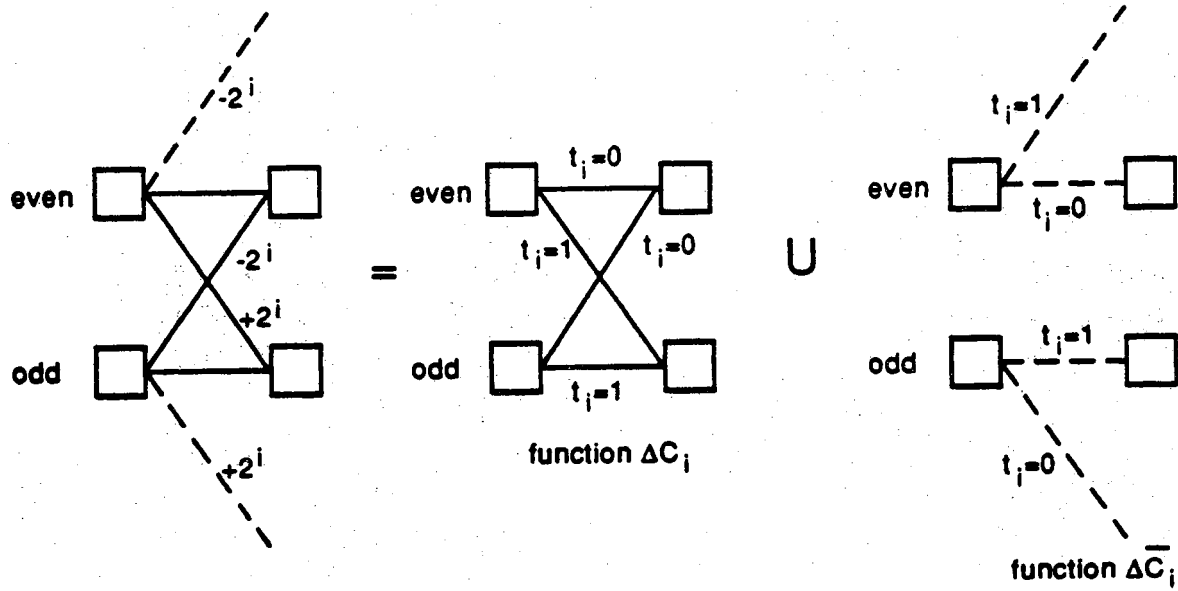


Figure 4. The connection links of stage i of the ICube network can be described by the function ΔC_i . The connection links of stage i of the IADM network can be described by the union of the functions ΔC_i and $\Delta \bar{C}_i$.

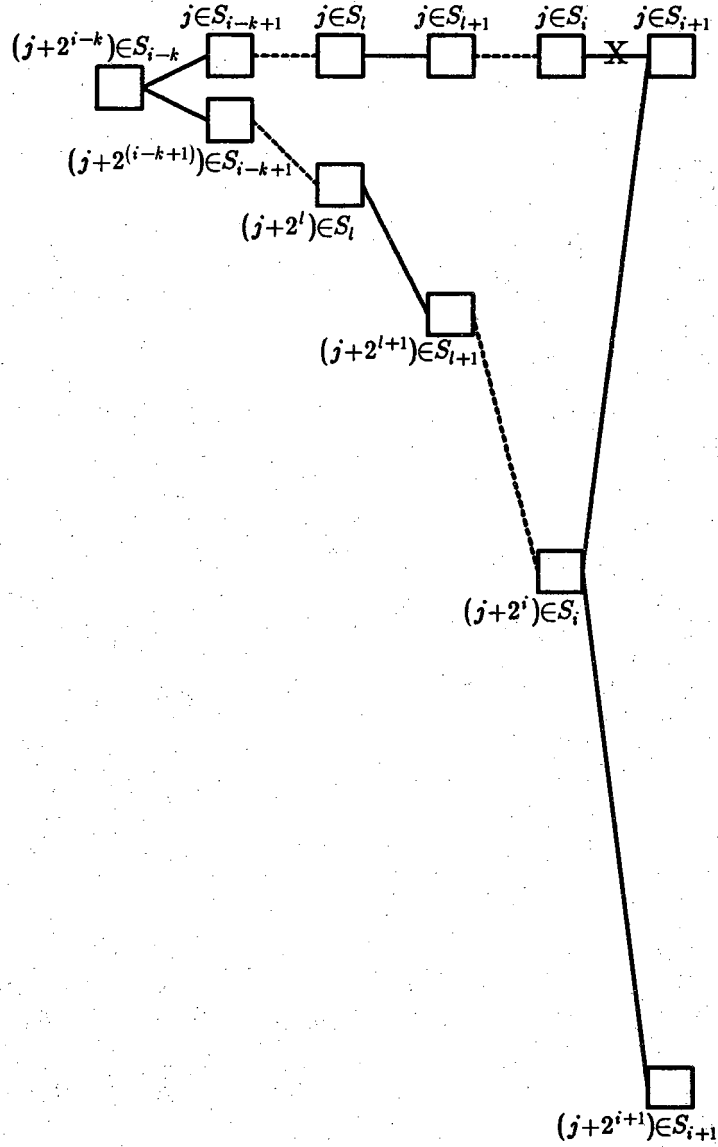


Figure 5. Rerouting for a straight link blockage in $(j \in S_i, j \in S_{i+1})$. Path $((j+2^{i-k}) \in S_{i-k}, j \in S_{i-k+1}, \dots, j \in S_{i+1})$ is a segment of the original path; $((j+2^{i-k}) \in S_{i-k}, (j+2^{i-k+1}) \in S_{i-k+1}, \dots, (j+2^{i'}) \in S_i, j \in S_{i+1})$ and $((j+2^{i-k}) \in S_{i-k}, (j+2^{i-k+1}) \in S_{i-k+1}, \dots, (j+2^{i'}) \in S_i, (j+2^{i+1}) \in S_{i+1})$ are the rerouting paths for it.

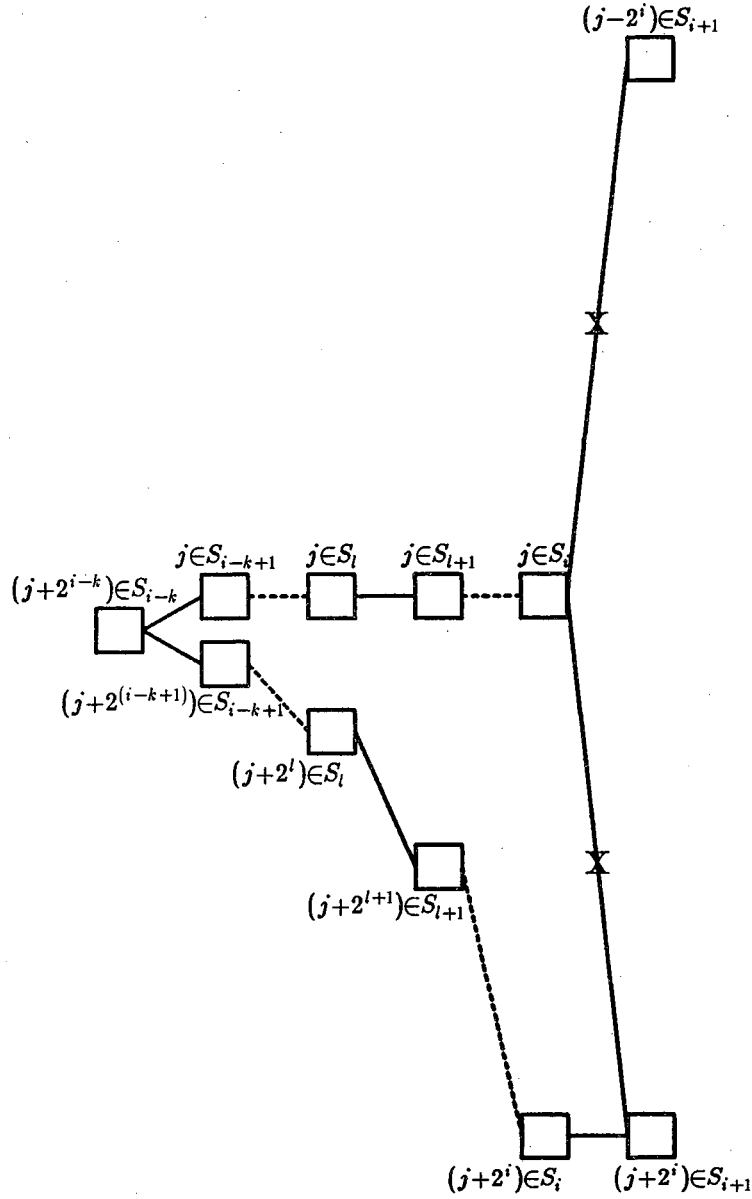


Figure 6. Rerouting for a double nonstraight links blockage in $(j \in S_i, (j-2^i) \in S_{i+1})$ and $(j \in S_i, (j+2^i) \in S_{i+1})$. Path $((j+2^{i-k}) \in S_{i-k}, (j+2^{i-k+1}) \in S_{i-k+1}, \dots, (j+2^i) \in S_i, (j+2^i) \in S_{i+1})$ is a rerouting path for both paths $((j+2^{i-k}) \in S_{i-k}, j \in S_{i-k+1}, \dots, j \in S_{i+1}, (j-2^i) \in S_{i+1})$ and $((j+2^{i-k}) \in S_{i-k}, j \in S_{i-k+1}, \dots, j \in S_{i+1}, (j+2^i) \in S_{i+1})$.

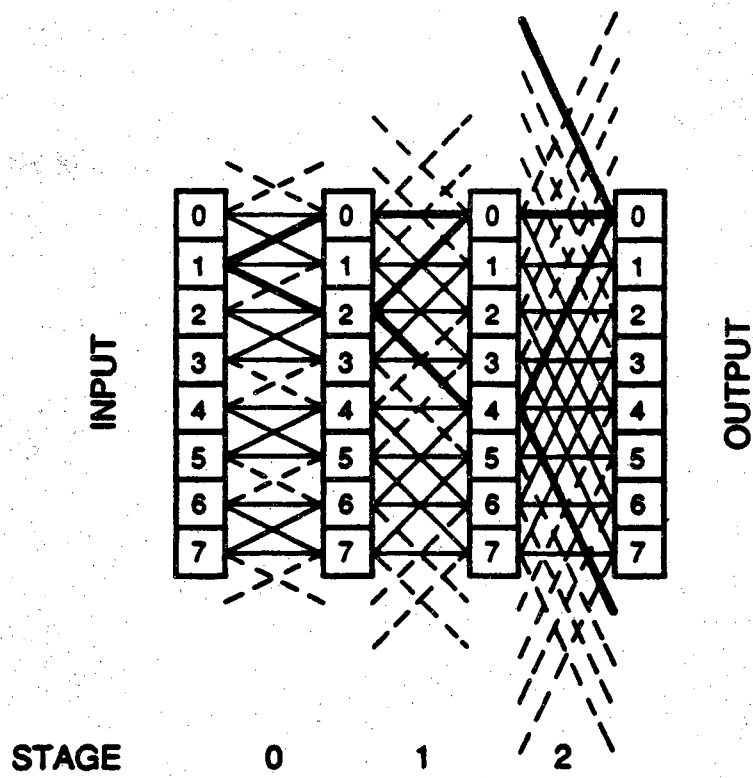


Figure 7. All routing paths from $1 \in S_0$ to $0 \in S_3$ in an IADM network of size $N=8$.

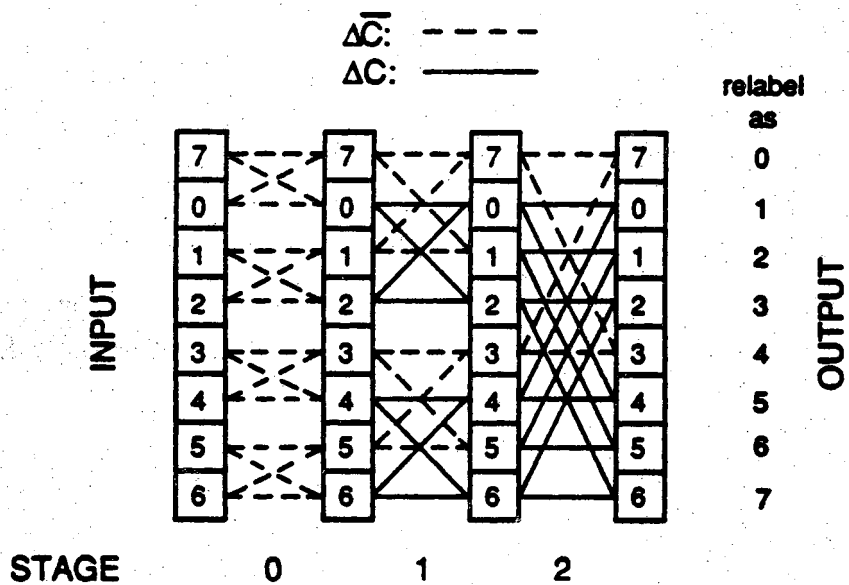


Figure 8. A cube subgraph generated by relabeling each switch j to $(j+1) \bmod 8$ for an IADM network of size $N=8$.

