Fault Location Techniques for Distributed Control Interconnection Networks

NATHANIEL J. DAVIS IV, STUDENT MEMBER, IEEE, WILLIAM TSUN-YUK HSU, AND HOWARD JAY SIEGEL, SENIOR MEMBER, IEEE

Abstract — **One class of networks suitable for use in parallel processing systems is the multistage cube network. This paper focuses on fault location procedures suitable for use in networks that employ distributed routing control through the use of routing tags and message transmission protocols. Faults occurring in the data lines can corrupt message routing tags transmitted over them and thereby cause misrouting of messages. Protocol lines (used in handshaking between network sources and destinations), if faulty, can prevent a message path from being established or can cause the path to "lock up" once transmission of data has begun. These faults have more pronounced effects on the network performance than faults previously considered for centralized routing control systems. The single-fault location procedures presented form a logical superset to those of the centralized control systems (where message routing is dictated by the actions of a global control unit) and can be adapted for use in both circuit and packet switching networks.**

Index Terms — **Circuit switching, cube network, distributed processing, fault location, generalized cube, interconnection networks, multimicroprocessor systems, parallel processing, PASM.**

I. Introduction

With the advent of very large scale integrated circuit technology, relatively inexpensive hardware systems and subsystems are now readily available. The result has been the greater use of multiple-processor system designs that employ processing elements, operating in parallel, to achieve high levels of computational power. The ability of these parallel systems to continue operations, despite the occurrence of faults, is of critical importance.

One class of interconnection networks suitable for use in parallel processing systems is the multistage cube network [15]. This class includes the omega [9], the indirect binary n-cube [13], the baseline [6], and the generalized cube [15]. The cube network is not, however, fault tolerant. Any single point failure in the network will prevent some source–destination pair of functional subsystems from communicating. Fault tolerance can be introduced into the network through the use of one or more "extra" stages of switches [1], [3], [12]. Effective use of the redundant paths in the network, available as a result of the extra stages, requires that each source know the exact location of any network faults.

Previous work in fault location has concentrated on networks operating under a centralized control scheme [2], [6], [7], [11]. Systems such as PASM [14] and Ultracomputer [8] implement the network using a distributed control methodology. Network faults in a distributed system, especially faults occurring in the interconnecting links within the network, can cause much more severe errors in network operations than could a similar fault in a centralized control system (a result of message misrouting due to the corruption of data tags). In addition, faulty protocol lines (used in handshaking between network sources and destinations) can prevent a path from being established or can cause the path to "lock up" once the transmission of data has begun. In this paper, the single-fault location procedures necessary for distributed control networks are considered. These procedures form a logical superset to those of the centralized systems. This work has been motivated by the implementation of the PASM system prototype [4].

In Section II, the system and network models are defined and the fault model is presented. Section III overviews the centralized control fault location procedures of [6]. An outline of the testing procedure for use in a distributed control network is presented in Section IV. Potential network faults, their ensuing effects on the network, and the testing procedure outputs they would produce are presented in Section V. Section VI discusses fault location techniques based on the output responses of the testing procedures. Procedures for testing broadcast connections are overviewed in Section VII. Section VIII summarizes these results.

II. The Interconnection Network Model

Consider a parallel processing system consisting of N functional subsystems where $N = 2^n$. The subsystems will be assumed to be *processing elements (PE's)*, processors paired with their own local memories. The interconnection network will have N inputs (sources) and N outputs (destinations). PE i will be connected to network input i and output i. The multistage cube network [15] consists of n stages with each stage being composed of $N/2$ two-by-two interchange boxes. Interchange boxes in stage i pair I/O lines with link

Manuscript received February 1, 1985; revised May 30, 1985. This work was supported by the Rome Air Development Center under Contract F30602-83-K-0119. A preliminary version of this paper was presented at the IEEE 1985 International Conference on Parallel Processing, St. Charles, IL, Aug. 1985.

N. J. Davis IV was with the PASM Parallel Processing Laboratory, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907. He is now with the Department of Electrical Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH 45433.

W. T. -Y. Hsu and H. J. Siegel are with the PASM Parallel Processing Laboratory, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

0018-9340/85/1000-0902$01.00

labels that differ only in the ith bit position. The same labeling is used for both the input and output lines connected to an interchange box. A multistage cube network is shown in Fig. 1, with $N = 8$. Each data path through the network will be m bits wide where m is a function of the system hardware used. Circuit switched data transmission is assumed, where a complete path linking the source and destination PE's must be established before the actual data transfer can begin.

Two different approaches can be used to govern the manner in which connections are made in the network. Using centralized control, a global network controller mediates between message requests and establishes the desired network connections. In contrast, distributed control removes this serial bottleneck by allowing the individual interchange boxes to establish their own connections based on the use of *routing tags* associated with each message. Distributed routing control using *destination address* routing tags [9] is assumed in this paper. The complete routing information is contained in a $2n$ bit routing tag: an n bit *broadcast mask* and an n bit *destination tag* equal to the binary expansion of the destination address. Interchange boxes in stage i examine bits i of both the broadcast masks and the routing tags for the messages at their input ports and make the switching connections accordingly. If a mask bit is "1," then a broadcast to both outputs is to be performed. If the mask bit is "0," then the routing tag specifies the desired connection pattern. A "0" in bit i of the routing tag indicates a connection to the upper output port is desired, while a "1" indicates connection to the lower output port. Conflicting connection requests can be resolved using conflict resolution schemes such as those discussed in [5], [10]. A message on the data lines will have the following format. The first word of the message will contain the message routing tag. The remaining words constitute the actual data that are to be transmitted. The set of data lines is assumed to include parity bit lines.

Message transmission in the network is controlled through the use of two types of asynchronous protocols. A message request/grant protocol is used to establish a path connecting source and destination PE's. The message request is the combination of the routing tag and a message request signal REQ. "Data available–data received" handshaking signals are used to transmit data between the input/output ports that interface the PE's to the network. The network interface ports are assumed to have the ability to validate message routing and data transmission (e.g., through the use of parity bits or checksum bits and the comparison of the routing tag information to known destination addresses). Thus, the complete information path will consist of the data lines (including the parity bit lines), the message REQ line, the data available line, the message grant line, and the data received line.

Blockages and potential faults in the network are detected by the source PE's when the anticipated return protocol signals are not received. The return protocol signals will not be generated if the message is blocked within the network or if the destination PE detects an error (e.g., a parity error or the destination tag does not match the destination PE's address). This process can be facilitated through the use of watchdog timers and/or nonacknowledge signals. Detection of a possible fault can initiate the execution of the fault location diagnostic routine.

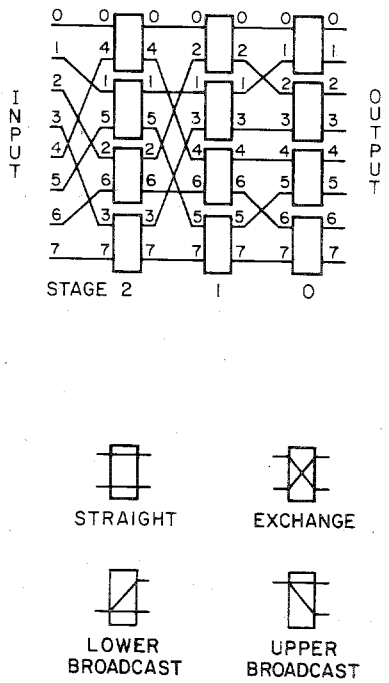
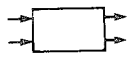
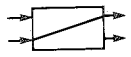
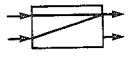
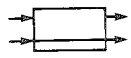
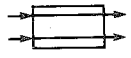
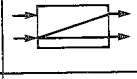
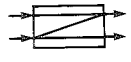
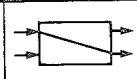
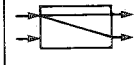
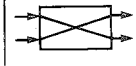
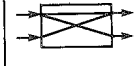
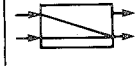
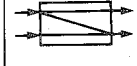
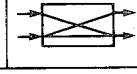
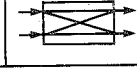


Fig. 1. A multistage cube network, with $N = 8$, and the allowable box settings.

A fault within the interconnection network can occur in either an interchange box or in one of the interconnection links within the information paths. All faults will be assumed to be nontransient. As delineated in [6], there are 16 possible ways of connecting the inputs of an interchange box to its outputs, as shown in Table I. A faulty box can become stuck in one particular invalid state, regardless of the routing information at its input ports, or it can respond incorrectly (but consistently) to the routing information. For an example of the latter situation, it is possible for a box to respond correctly when the straight state (S_{10}) is requested, but incorrectly when the exchange state (S_5) is requested. Furthermore, a faulty box may enter one incorrect state (say, S_{11}) in response to a straight state request, and enter a different incorrect state (say, S_{12}) in response to an exchange state request. Thus, a box fault occurs when an interchange box enters an incorrect state.

Any network fault that corrupts data on an information path will be called a link fault. A link fault occurs in an information line when it becomes stuck at either logical "0" or "1," regardless of the actual input signal that is applied to it. The actual location of the fault can be in the link itself, an interchange box, or in the hardware interfaces of the interchange boxes that the link connects. (Depending on the box implementation, some such faults within a box may be detected as box faults.) The fault model will allow link faults to occur in either the data lines themselves or the lines carrying the protocol and parity bit signals. As will be seen in Section IV, link faults in distributed control systems can create a large number of network errors not found in centralized control systems. These errors are due to the corruption of routing tags as they are transmitted over faulty links and the ensuing misrouting and blocking of the messages.

TABLE I
16 POSSIBLE STATES FOR A TWO-BY-TWO INTERCHANGE BOX

State	Interchange Box Setting	State	Interchange Box Setting
S_0		S_8	
S_1		S_9	
S_2		S_{10}	
S_3		S_{11}	
S_4		S_{12}	
S_5		S_{13}	
S_6		S_{14}	
S_7		S_{15}	

III. FAULT LOCATION IN CENTRALIZED CONTROL NETWORKS

In [6], [7], Feng, Wu, and Zhang present a comprehensive method for detecting and locating both link and box faults in a centralized control interconnection network. A three-phase testing strategy is developed to detect faults within the network. The testing of the network is performed in an SIMD (synchronous) operating mode under the direction of a global system control unit.

In phase 1, all interchange boxes are set to the straight connection. A logical "0" and a logical "1" are transmitted over each data link through the network. By comparing the received output words (at the destinations) to the known correct output specified by the system control unit, the presence of a fault can be readily determined. In phase 2, the process is repeated with the interchange boxes set to exchange.

The two-phase process requires the transmission of exactly four data words and has been shown to detect all single faults within the network for the straight and exchange settings. The fault location can be obtained by comparing the paths on which detected errors occurred or through the use of a binary tree search algorithm that can isolate most faults (or, in a few cases, indicate a localized region of two adjacent boxes and their connecting link) in no more than $\max(12, 6 + 2\lceil \log_2(\log_2 N)\rceil)$ tests. Complete details of the testing procedures can be found in [6].

Once the link faults and faults in the straight and exchange box settings have been located, testing for the broadcast settings can be performed. Each stage is individually set to upper (lower) broadcast, while the remaining stages are set to either straight or exchange. Faults that resulted from the broadcast operation can be quickly pinpointed within the broadcast stage. Including the detection of faults in the broadcast settings in the procedure of [6] requires a total of $\max(28, 4 \log_2 N + 8, 14 + 2\lceil \log_2 N(\log_2 N)\rceil)$ tests to, in most cases, locate the fault or, in a few cases, localize the faulty region to two adjacent boxes and their connecting link [7].

IV. NETWORK FAULT DETECTION IN DISTRIBUTED CONTROL SYSTEMS

The set of possible fault patterns in a distributed control network is more complex than that of a centralized control network. This is because the routing tags which direct the path establishment through the network and the data transmission protocol lines are carried over the same information paths as the data. For example, a link fault may produce an erroneous routing tag which, in turn, may cause the message to be misrouted — an error that is not possible in a centralized control system.

The procedures and methodology described here are for detecting and locating single faults in a distributed control network. They are based on the network model of Section II, but can be adapted for other cube-type interconnection networks and for different formats of the data path and protocol lines. Specific examples will refer to a multistage cube network, with 16 PE's and a 16 bit data path. The routing tag word will contain the 4 bit destination address in bits 3–0 of the word and a 4 bit broadcast mask in bits 11–8. The other bits in the word (bits 7–4 and 15–12) are not used. There are two parity bits, a high-order bit for bits 15–8, and a low-order bit for bits 7–0 of the 16 bit word. This format is similar to the routing tag format used in the PASM prototype network [4].

The fundamental testing procedure remains similar to that described in [6], [7]. To check for link stuck-at faults, each set of links carrying data or parity bits must have two bitwise complementary words transmitted over it. Procedures for isolating these types of faults will be discussed in Section V. To test for faults in the interchange boxes, we shall first attempt to set each box to the valid states S_{10} and S_5 (straight and exchange). Broadcast settings will be evaluated separately. Faults will be detected by examining the test patterns which propagate through the network and are received by the destination PE's, and by combining this information with the blockage/timeout information available from the source PE's.

The basic testing procedure for straight and exchange settings is divided into two phases. In each phase we shall attempt to detect if there are one or more faulty paths and, through their intersection, isolate the faulty component. In *phase 1*, all boxes are preset to the "straight" setting through the actions of routing tags submitted to the network by the source PE's. Each source PE sends its own address as the destination address tag (bits 4–15 are set to zero). Call

the process of sending routing tags through the network to preset all paths the *setup*.

If some block or routing error is detected, all paths are immediately dropped (by negating REQ), and phase 2 of the test is begun. If, however, no block or routing error occurred, each PE will begin the *data transfer* testing subphase of phase 1. The first data word to be transmitted will be the bitwise complement of the routing word. If again no error is detected, a second data word, with different parity, is sent through the network to ensure the links carrying the parity bits have been tested properly. This is necessary because, for a data path with an even number of bits (as assumed here), the parity bit values will remain the same for the routing tag word in setup and the first data word, since complementing a word with an even number of bits does not change its parity bit. The second data word is formed by complementing bits 0 and 8 of the first data word, i.e., the low-order bit from each byte.

As an example of this phase of the testing procedure, consider a 16 PE system. PE 6 would have a phase 1 routing tag word of 0000000000000110 with parity bits 00. The first data word would be 1111111111111001 with parity bits 00. The second data word would be 1111111011111000 with parity bits 11. Every data and parity link in the information path will have had both logical "0" and logical "1" signal levels transmitted over it.

In *phase 2*, the setup procedure involves having each source PE send the complement of its address as the destination tag. This is equivalent to requesting that all interchange boxes be set to exchange (setting S_5). As in phase 1, all unused bits are set to "0" for convenience. If no block or routing error occurs in the setup subphase, the PE's will send as the first data word the bitwise complement of the routing word.

If no error is detected with the transmission of the first data word, an extra data word with different parity is sent through the network to ensure that the links carrying the parity bits have been properly tested. This second data word is formed in the same way as that in phase 1, i.e., by complementing bits 0 and 8 of the first data word. Continuing with the example from phase 1, PE 6 would have a phase 2 routing tag word of 0000000000001001 with parity bits 00. The first data word would be 1111111111110110 with parity bits 00. The second data word would be 1111111011110111 with parity bits 11.

The destination's network interface verifies the message routing during setup (by examining the destination address portion of the setup word) and the parity for every received word. If there are no errors, it then generates the return protocol signals for the source PE.

V. The Effects of Network Faults

Faults occurring in a network that uses distributed control can cause much more serious operational errors than in a comparable centralized control network. Faults and their concomitant error patterns generated by the two phases of the straight and exchange setting tests are discussed below for both link and box faults.

A. Error Patterns for Link Stuck Faults

Two types of link faults can occur: faults in the data path or parity bit links, or faults in the links which carry the message protocol signals. Table II is a complete listing of the errors caused by each type of link fault. In the table, columns "one phase" and "other phase" record the received error signals (if any) and do not necessarily correspond to the phase 1–phase 2 testing sequence.

1) Type 1: Faults on the Data Path and Parity Bits: Functionally, Type 1 faults can be divided into five cases.

Case 1: A link fault occurs in a bit that is not used by the routing word. No matter how the unused bits are scrambled, messages will still be routed to their correct destinations. Since all unused bits are set to "0" in setup, if a link carrying one of these bits is stuck at "1," there will only be one parity error in each test phase (Type 1, Case 1a of Table II). If a link carrying the bit is stuck at "0," no error will occur during setup. When the first data word is sent through the network, all unused bits are set to "1." Parity checks will then result in one error in each phase (Type 1, Case 1b of Table II).

Case 2: A link fault occurs in a routing bit after that bit has already been examined for routing purposes. For example, if a link carrying bit 3 of the destination tag has a stuck-at fault between stage 1 and stage 0, the message routing will not be affected (for routing purposes, bit 3 would have already been examined at stage 3). This type of fault produces the same fault patterns as Case 1 (Type 1, Case 2a and Type 1, Case 2b of Table II).

Case 3: In this case, a link fault occurs in a routing bit before that bit has been examined in the setup process. Since a link of this nature carries a "1" in the setup of one phase and a "0" in the setup of the other, in one of the phases there will be no setup error, but an error will occur when transferring the first data word. In the other phase, there will be an error in setup. Two routing possibilities may happen. The erroneous bit may successfully request an erroneous path and, in turn, block an otherwise good path — causing a block and a routing error (Type 1, Case 3b of Table II) — or, if the good path has already been established, the erroneous path will be blocked and the only error in that phase would be a block (Type 1, Case 3a of Table II).

Case 4: A fault occurs on a link carrying a parity bit. Here, a link carrying a parity bit is stuck at "1" or "0." In both test phases, this will either be detected during setup or when transmitting one of the two data words. There will either be errors in both phases when doing data transfers (Type 1, Case 4a of Table II), routing errors in setup in both phases (Type 1, Case 4b of Table II), or an error in the setup in one phase and an error in data transfer in the other phase (Type 1, Case 4c of Table II) (routing error here refers to a parity error in the setup word, not a misroute).

Case 5: A link carrying a broadcast bit is stuck so that a broadcast operation is erroneously requested. The erroneous broadcast request may succeed in neither, one, or both of the two test phases (Type 1, Case 5 of Table II). The observed errors depend on whether the broadcast setting is successfully set up (which will result in a parity error detected at a destination and a blockage of the correct path of another mes-

TABLE II
FAULT PATTERNS FOR LINK STUCK FAULTS

Fault patterns for Link Stuck Faults					
Case	Faulty Link	one phase		other phase	
		setup	data transfer	setup	data transfer
TYPE 1					
1a 1b	unused bits	E OK	E	E OK	E
2a 2b	destination bit, does not affect routing	E OK	E	E OK	E
3a 3b	destination bit, affects routing	B EB		OK OK	E E
4a 4b 4c	parity bit	OK E E	E	OK E OK	E E
5a 5b 5c	broadcast bit asserted, affects routing	B EB EB		B EB B	
TYPE 2					
1	message request stuck negated	B		B	
2	message grant stuck negated	B		B	
3	data available stuck negated	E		E	
4	data received stuck negated	E		E	
5a 5b	message request stuck asserted	OK EB	OK	EB EB	
6	message grant stuck asserted	OK	OK	OK	OK
7	data available stuck asserted, before stage 0	OK	E	OK	E
8	data available stuck asserted, after stage 0	E		E	
9	data received stuck asserted, after stage n-1	OK	E	OK	E
10	data received stuck asserted, before stage n-1	E		E	

Fault pattern notation:
"OK" -- No errors or blocks in that testing subphase.
"E" -- Only 1 PE detects a routing or parity error.
"B" -- Only 1 PE detects a block.
"EB" -- One PE detects a routing error, 1 or more PEs detects blocks.

sage), or if the broadcast request is blocked at the box in which the broadcast would be performed (by the other box input correctly establishing its path first and thus preventing the broadcast setting).

2) Type 2: Link Faults in Control Links: There are four control lines which can be stuck at asserted or negated: message request, message grant, data available, and data received.

Case 1: A link carrying a message request signal is stuck in the negated state. To set up an interchange box, a message's request signal must be asserted. If not, the message will not propagate through the box, causing a perceived blocking error to be detected. This will occur in both test phases (Type 2, Case 1 of Table II).

Case 2: A link carrying a message grant signal is stuck in the negated state. A perceived blocking error will be detected in each phase (Type 2, Case 2 of Table II).

Case 3: A link carrying a data available signal is stuck in the negated state. A routing error will occur in each phase because the assertion of the data available signal is never detected by the destination port, and as a result, the data received signal is never returned to the source PE (Type 2, Case 3 of Table II).

Case 4: A link carrying a data received signal is stuck in the negated state. This will result in one routing error in each phase since no data received signal returns to indicate that the routing tag has been received (Type 2, Case 4 of Table II).

Case 5: A link carrying a message request signal is stuck in the asserted state. Because message request is always asserted, the portion of the last path that is after the fault location (and established before the link failed) will not be dropped and will remain held through the fault isolation procedure. Depending on what this last path was, there are several different fault patterns. If the held path consists of all straight or all exchange box settings, there will be no routing errors or blocks in one phase, but a routing error and a block in the other phase (Type 2, Case 5a of Table II). Note that the new routing tags generated by the source PE will be treated as nontag data items by the incorrectly held path, causing the routing error. If the held path consists of a combination of straights and exchanges, there will be a routing error and one or more blocks in each phase (Type 2, Case 5b of Table II).

Case 6: A link carrying the message grant signal is stuck in the asserted state. This will produce no errors or blocks in the normal fault location procedure. The only way to detect this is to deliberately attempt to set up paths which will be blocked in the network and check for the signal being stuck (Type 2, Case 6 of Table II).

Cases 7 and 8: A link carrying the data available signal is stuck at the asserted state. Assume that the data available signal is edge-sensitive and that active low logic is being used in the network implementation (both reasonable assumptions for typical port handshaking signals). The errors that can be generated will depend on whether the stuck link is before or after stage 0 (i.e., whether it is the network output link or not).

In Case 7, a data available link is stuck at asserted before stage 0 (recall from Fig. 1 that stage 0 is the network output stage). An edge is still produced on subsequent links when the path is first set up. Hence in both phases, no error or block occurs in setup, but errors will occur when transmitting the first data word since the destination port will not receive the required negated-to-asserted edge on the data available line to gate the data word in (Type 2, Case 7 of Table II).

For Case 8, a data available link is stuck at asserted after stage 0 (on the line connecting stage 0 to its respective network–destination interface port). As a result of the fault, the port never detects an edge transition being produced on this control link. In each phase, there will be one routing error only. The fault patterns generated here are identical to those in Case 3 (Type 2, Case 8 of Table II).

Cases 9 and 10: A link carrying the data received signal is stuck at asserted. Similarly to Cases 7 and 8, the errors generated depend on whether the bad link is before stage $n - 1$.

In Case 9, assume that a data received link is stuck at asserted after stage $n - 1$ (the input stage of the network). As with the data available signal, assume that the data received signal is edge-sensitive. An edge will be produced when the path is first established. Hence, in both phases, no error or block occurs in setup, but errors will occur when transmitting the first data word since the source port will not receive the required negated-to-asserted edge on the data received line (Type 2, Case 9 of Table II).

For Case 10, let a data received link be stuck asserted

before stage $n-1$. Since the stuck link is between stage $n-1$ and the source port, no edge is ever produced on this bad link. In each phase, there will be one routing error only (Type 2, Case 10 of Table II).

B. Error Patterns for Interchange Box Faults

Interchange box faults in interconnection networks with distributed routing schemes are handled in much the same way as are Feng and Wu's switching element faults in [6]. Differences lie primarily in the additional effects of the m bit data path and the routing and protocol schemes, not addressed in [6]. This class of faults will be described briefly with emphasis being placed on these differences.

Initially, two groups will be considered: a faulty state when S_{10} is the desired state and a faulty state when S_5 is desired (broadcast states are discussed in Section VII). The possible fault patterns for interchange box faults in these states are summarized in Table III. In the analysis, it is assumed that a box fault will affect all lines of an input port in the same way.

1) Faulty State in S_{10}: This refers to the condition where the combination of the routing tag and REQ signals requests setting a box to S_{10}, but, because of a fault in the internal logic of the box, it is set to some other state instead. From Table I, there are 15 possible erroneous states. Erroneous states S_0, S_1, S_2, S_4, S_5, and S_8 are straightforward—messages are either misrouted or blocked, and error-checking hardware at the PE's detects routing errors or blocks. For erroneous states S_3 and S_{12}, one of the messages requesting passage through the faulty box is blocked, while the other message is sent to both output lines. The effect of this will be a blocked message and a routing error signal from the second incorrect output. The box input port hardware is assumed to perform a logical AND operation on the returning protocol signals from each of the box output ports to detect the presence of an error condition (i.e., message received logically 0) from either output link. The error signal is, in turn, propagated towards the respective source PE. The error patterns resulting from these eight erroneous states are similar to those of [6].

The remaining seven erroneous states, S_6, S_7, S_9, S_{11}, S_{13}, S_{14}, and S_{15}, may involve changes to the routing tags, and therefore, new considerations come to bear. The routing tags can become corrupted when one of the messages at the output links of the faulty box is the result of the two input messages overwriting each other. The effect of having two input bits write to the same output bit is defined in a way similar to that of Feng and Wu in [6]. If the two input bits are identical, the output bit will be equal to either one of the inputs. However, if the input bits are different, the output bit will always be a "1" or always a "0," i.e., an overwritten bit position always sticks at the same value. When two m bit routing words are transferred to the same output link, some bits in the resulting tag may be scrambled by the overwrite. Depending on the message at the output link of the faulty box, there are two resultant cases.

In the first case, the message is changed by the overwrite so that an error in the bits of the setup word which specify the destination address is generated. A routing error is detected.

TABLE III
FAULT PATTERNS FOR INTERCHANGE BOX FAULTS

Interchange Box Faults					
Desired Setting is S_{10}			Desired Setting is S_5		
Erroneous	Observed Errors		Erroneous	Observed Errors	
	Setup	Data Transfer		Setup	Data Transfer
S_0	2B		S_0	2B	
S_1	EB		S_1	B	
S_2	B		S_2	EB	
S_3	EB		S_3	EB	
S_4	EB		S_4	B	
S_5	2E		S_6a S_6b	2E OK	2E
S_6a S_6b	2E OK	2E	S_7a S_7b	2E OK	2E
S_7a S_7b	2E E		S_8	EB	
S_8	B		S_9a S_9b	2E OK	2E
S_9a S_9b	2E OK	2E	S_{10}	2E	
$S_{11}a$ $S_{11}b$	2E OK	2E	$S_{11}a$ $S_{11}b$	2E E	
S_{12}	EB		S_{12}	EB	
$S_{13}a$ $S_{13}b$	2E E		$S_{13}a$ $S_{13}b$	2E OK	2E
$S_{14}a$ $S_{14}b$	2E OK	2E	$S_{14}a$ $S_{14}b$	2E E	
$S_{15}a$ $S_{15}b$	2E OK	2E	$S_{15}a$ $S_{15}b$	2E OK	2E

Fault pattern notation:
"OK" -- No errors or blocks in that testing subphase.
"E" -- Only 1 PE detects a routing or parity error.
"2E" -- Two PEs detect routing or parity errors.
"B" -- Only 1 PE detects a block.
"2B" -- Two PEs detect a block.
"EB" -- One PE detects a routing error, 1 or more PEs detects blocks.

For example, if a stage i box is in erroneous state S_6, bit i of the setup word using the lower input should be a "1." Assume the two setup words entering the box are merged so that bit i is overwritten and is stuck at "0." Thus, a destination with a "1" in its ith address bit position will receive a setup word with a "0" in the ith bit position. The source PE's requesting a path through the faulty interchange box will detect this error since the error condition is propagated back to both PE's.

In the second case, the message can be changed by the overwrite so that an error in the destination address part of the setup word is not generated as a result of the overwritten bits (e.g., for a stage i box, bit i of the message at the lower output of S_6 in Table I should be "1" and is stuck at the same value because of the overwrite). The proper return protocol signals are propagated back to both of the affected source PE's. In this case, no routing error or block is detected in the setup for S_6, S_9, S_{11}, S_{14}, and S_{15}. However, the first data word is a bitwise complement of the routing tag, while the overwritten bits always stay at the value they took in the setup. An error will result from the parity check, and both source PE's receive the error signal. The result is two routing errors. In the setup for S_7 and S_{13}, one of the messages is misrouted regardless of whether the overwritten message was scrambled. For example, the message at the upper output of S_7 in Table I is misrouted, and the lower input performs a logical AND operation on both returning protocol signals and informs the PE connected to the lower input of the error. In these two states, a single routing error in the setup will result.

2) Faulty State in S_5: This refers to the condition where

the combination of the routing tag and REQ signals requests setting the interchange box to S_5, but the box is set to some other state instead. As for the case where the desired state was S_{10}, there are 15 possible erroneous states. The way in which fault patterns are generated is very similar to those of S_{10}. Refer to Table III for a complete description of the fault patterns generated by each type of fault.

VI. Isolation of Single Faults

Tables II and III summarize the errors resulting from the different types of faults after the first two phases of the testing procedure. The principle behind the testing procedure is to determine the fault type, i.e., whether a link or an interchange box is faulty, and the location of the fault. This is done by recording the path on which faults are detected in each phase and intersecting the faulty paths.

Notice that in the condition denoted by "*EB*" in the tables, i.e., one PE detects a routing error with one or more PE's detecting blocks, only the path which resulted in the routing error is considered to be the faulty path. This is because the blocked path may be fault free, but it was blocked by a misrouted message. If no routing errors are detected in a test phase but blocks occurred (i.e., in the conditions denoted by "*B*" and "2*B*"), paths with blocks are considered faulty paths. This is because there is no detected error in the network which could have caused a fault-free path to be blocked. The exception to this rule is for faults belonging to Group 3, as explained below. In the conditions denoted by "2*E*," two paths are considered to be faulty paths.

Looking over the tables of results for link faults and box faults, the faulty responses can be divided into several groups. These are considered below.

A. Group 1: Two Faulty Paths in Either the Setup or Data Transfer of a Single Phase

This refers to situations where the error conditions denoted by "2*E*" and "2*B*" are detected, either in the setup or in the data transfer. Referring to Tables II and III, these conditions will only be registered if an interchange box is faulty, i.e., if the desired state is S_{10} and the erroneous state is S_0, S_5, S_6, S_7a, S_9, S_{11}, $S_{13}a$, S_{14}, and S_{15} or if the desired state is S_5 and the erroneous state is S_0, S_6, S_7, S_9, S_{10}, $S_{11}a$, S_{13}, $S_{14}a$, and S_{15}. Since two faulty paths were registered in a single phase, intersection of these two paths will pinpoint the faulty box.

B. Group 2: No Setup Errors in One or Both Phases, Data Transfer Error in Phase(s) Where No Setup Error Occurred

In this group, no setup errors occur in one or both setup subphases. Where the setup is valid, a single error will be detected in the ensuing data transfer. This group includes link faults of Type 1, Cases 1b, 2b, 3, and 4a and c, and Type 2, Cases 7 and 9. Interchange box faults are not in Group 2 since they never generate a single error in data transfer in one or both phases (see Table III).

Referring to Table II, a Group 2 link fault always generates one faulty path in each phase. Since fault patterns of this type can be definitely identified as being caused by link faults, intersection of the faulty path obtained in phase 1 and that obtained in phase 2 will isolate the faulty link.

C. Group 3: No Anomaly in One Phase, Only One Faulty Path in the Other Phase

This group includes all those conditions where no anomaly (i.e., no routing error, parity error, or block) was detected in one phase, but an error and/or block in setup or data transfer was detected in the other phase. Hence, if faulty test patterns of this group were detected, only one faulty path will be registered and further tests are necessary to isolate the fault.

From Table II, only one type of link fault might produce fault patterns of this group: Type 2, Case 5a. The fault pattern produced here is the condition denoted by "*EB*." If the fault were in an interchange box, several types of faults would generate patterns belonging to this group.

a) The faulty box would work normally if it were set to S_{10} in the phase 1 test, but in the phase 2 test, instead of being set to S_5, it is set to S_1, S_2, S_3, S_4, S_8, $S_{11}b$, S_{12}, or $S_{14}b$. Of these eight possibilities, S_1, S_4, $S_{11}b$, and $S_{14}b$ do not produce the condition denoted by "*EB*" and are thus easily recognized as box faults instead of a link fault of Type 2, Case 5a.

b) The faulty box would work normally if it were set to S_5 in the phase 2 test, but in the phase 1 test, instead of being set to S_{10}, it is set to S_1, S_2, S_3, S_4, S_7b, S_8, S_{12}, or $S_{13}b$. Of these eight possibilities, S_2, S_8, S_7b, and $S_{13}b$ do not produce the condition denoted by "*EB*" and are thus easily recognized as box faults instead of a link fault of Type 2, Case 5a.

For the "*EB*" fault conditions in this group, both paths are considered faulty, and the location of the faulty component is exactly pinpointed by the intersection of the two paths on which the routing/parity error and the block lie. If the "*EB*" condition is caused by link fault Type 2, Case 5a, the intersection of the faulty path and the blocked path will give the interchange box immediately after the stuck link. If more than one block is detected, each blocked path is intersected with the path containing the routing/parity error. The component(s) obtained by the intersection that is closest to the input stage of the network is considered to be the faulty component. Hence, the faulty component is either the interchange box indicated by the intersection of the two bad paths, or an input link on that box.

For the remaining faults in this group (the non-"*EB*" cases), a binary tree search algorithm can be used to isolate the fault. See the discussion given in [6] under the section on "Switching Element Faults," Case 1.

D. Group 4: Anomalies in the Setup of Both Phases

In this group, one faulty path is generated in the setup of both phase 1 and phase 2. This group covers all the remaining link faults except Type 2, Case 6, and also the interchange box faults in which a particular box is set to an erroneous state(s) in both phase 1 and phase 2. The three error conditions possible in the setup are "*E*," "*EB*," and "*B*." All possible combinations of these conditions are summarized in Table IV. From Table II, it can be observed that link faults which generate fault patterns in the setup of both

TABLE IV
GROUP 4 FAULT PATTERNS

Fault Patterns for Group 4		
Subgroup	phase 1 setup	phase 2 setup
1	E	E
2	E	EB
3	E	B
4	EB	E
5	EB	EB
6	EB	B
7	B	E
8	B	EB
9	B	B

phases always produce identical fault patterns in both phases. Hence, combinations of fault patterns which are not identical in both phases (Table IV, Subgroups 2, 3, 4, 6, 7, and 8) are immediately identifiable as interchange box faults. Since two faulty paths are obtained after the two-phase test, the intersection of the faulty paths will locate either the faulty box or a pair of interchange boxes connected by a link. The latter condition and subcases 1, 5, and 9 of Table IV require special procedures to further isolate the faulty component. These procedures, which involve trying to pinpoint the fault by setting boxes in different stages to different valid states, are described in detail in [6] and are an abbreviation of their tree search algorithm.

E. Group 5: No Anomalies Detected

In this group, no errors are detected in either the straight, the exchange, or the broadcast testing (discussed in Section VII) phases. This could indicate the presence of one remaining fault type—a Type 2, Case 6 link fault (a message grant protocol line is stuck asserted). Knowing the source PE that requested the diagnostic testing and the path that it was trying to set up when the initial network error was detected (the fault was in that particular path), the location of the fault can be determined. Blocking paths can be systematically established in the network that are known to conflict with the path containing the fault. If the faulty path and a blocking path intersect before the fault location, the grant signal should become negated (as a result of the blockage). If the paths intersect after the fault location, the grant signal will remain asserted, despite the path being blocked. The fault is identified as being in a particular link when a block in the preceding interchange box causes the grant signal to be negated, while a block in the succeeding box does not negate the signal. The search for the fault location can be performed in a binary tree search fashion and will require $O(\log_2 n)$ steps to complete.

F. Fault Location Summary

After the first two phases of the testing, it is possible to detect all link faults and single straight and exchange faults. The location of the component can be determined by examining the combination of error signals or, at worst case, narrowed to a pair of interchange boxes and their connecting links. Group 4 faults present the most difficult combination of error patterns. In those cases where the fault has not been completely isolated, the special procedures described in [6] are necessary to further isolate the fault.

VII. TESTING FOR BROADCASTS

If no anomalies are detected in the two-phase test for the straight and exchange settings, the broadcast states will be validated. Since possible link faults have been checked for, only interchange box faults need to be considered.

The basic procedure is very similar to that used in [7]. Each stage will in turn be set to both the upper and the lower broadcast states (S_{12} and S_3), with all other stages being set to straight for simplicity. Since the broadcasting stage is known, any anomaly immediately pinpoints the fault.

The testing procedure is divided into two subphases for each stage. Suppose stage i is being tested. In the setup subphase for upper broadcast testing, all boxes in the stage i are set to S_{12}. Each of the $N/2$ PE's with a "0" in bit i of its address sends its own address as the destination address, and a broadcast tag which is all "0"s except for bit i, which would be a "1." If no anomaly is detected, the PE's perform the data transfer subphase by sending the bitwise complement of the message header in order to check for overwrites (discussed below).

In the setup subphase of lower broadcast testing, all boxes in the stage being tested are set to S_3. Each of the $N/2$ PE's with a "1" in bit i of its address sends its own address as the destination address, and a broadcast tag which is all "0"s except for bit i, which would be a "1." If no anomaly is detected, the PE's perform the data transfer subphase by sending the bitwise complement of the message header.

In either subphase, the presence of an erroneous overwrite state can be detected by having the $N/2$ previously idle PE's attempt to establish paths using all straight connections. An overwrite state would not block the new straight request, as should normally happen. Instead, data from both inputs would be overwritten and the ensuing parity errors would be detected.

In summary, the stagewise validation of the network for broadcasting would take $O(n)$ tests where n is the number of stages. Because the links and the interchange boxes not performing broadcasts are assumed to be fault free, any anomaly would immediately isolate the faulty box.

VIII. SUMMARY

In this paper, an existing fault detection and location procedure for centralized routing control networks has been overviewed and extended for use in distributed routing control systems. Networks that employ distributed routing control transmit both message routing and protocol information and data through the network. The errors that could occur in these networks were analyzed and shown to be a superset of the errors occurring in a centrally controlled system. Faults occurring in the data lines could corrupt the message routing tags transmitted over them and thereby cause the misrouting of messages. Additionally, protocol lines used in the handshaking between source–destination PE pairs, if faulty, could

prevent a message path from being established or could cause the path to become "locked up" once the transmission of data has begun. A fault detection and location procedure, patterned after the ones presented in [6], [7], was developed. The procedure is executed in three phases where each phase involves the transmission of routing control information to set up the desired network connections and the transmission of data words to test the integrity of the paths. Response patterns to the test messages were derived for link faults in the data path as well as in the protocol links. Interchange box faults and their associated fault patterns were also investigated. Faults were detected by the source PE's, as perceived routing and/or parity errors, rather than through the inspection of received data at the destinations as in [6], [7].

While the procedures described in this paper were specifically targeted to a circuit switched network implementation, the approach could be modified for packet switched networks using similar control protocol structures. In packet switching, protocol lines connect interchange boxes in adjacent stages rather than the sources and destinations in circuit switching. As a result, the effects of faults in these lines will tend to be more localized than in the circuit switched networks discussed in this paper.

REFERENCES

[1] G. B. Adams III and H. J. Siegel, "The extra stage cube: A fault-tolerant interconnection network for supersystems," *IEEE Trans. Comput.*, vol. C-31, pp. 443–454, May 1982.

[2] D. P. Agrawal, "Testing and fault tolerance of multistage interconnection networks," *IEEE Comput.*, vol. 15, pp. 41–53, Apr. 1982.

[3] C.-Y. Chin and K. Hwang, "Packet switching networks for multiprocessors and data flow computers," *IEEE Trans. Comput.*, vol. C-33, pp. 991–1003, Nov. 1984.

[4] N. J. Davis IV and H. J. Siegel, "The PASM prototype interconnection network," in *Proc. 1985 Nat. Comput. Conf.*, July 1985, pp. 183–190.

[5] ——, "The performance analysis of partitioned circuit switched multistage interconnection networks," in *Proc. 12th Symp. Comput. Architecture*, June 1985, pp. 387–394.

[6] T.-Y. Feng and C.-L. Wu, "Fault-diagnosis for a class of multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-30, pp. 743–758, Oct. 1981.

[7] T.-Y. Feng and Q. Zhang, "Fault diagnosis of multistage interconnection networks with four valid states," in *Proc. Fifth Int. Conf. Distrib. Comput. Syst.*, May 1985, pp. 218–226.

[8] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer—Designing an MIMD shared-memory parallel computer," *IEEE Trans. Comput.*, vol. C-32, pp. 175–189, Feb. 1983.

[9] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.*, vol. C-24, pp. 1145–1155, Dec. 1975.

[10] M. Lee and C.-L. Wu, "Performance analysis of circuit switching baseline interconnection networks," in *Proc. 11th Symp. Comput. Architecture*, June 1984, pp. 82–90.

[11] W. Y.-P. Lim, "A test strategy for packet switching networks," in *Proc. 1982 Int. Conf. Parallel Processing*, Aug. 1982, pp. 96–98.

[12] K. Padmanabhan and D. H. Lawrie, "A class of redundant path multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-32, pp. 1099–1108, Dec. 1983.

[13] M. C. Pease III, "The indirect binary n-cube microprocessor array," *IEEE Trans. Comput.*, vol. C-26, pp. 458–473, May 1977.

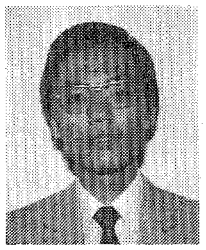
[14] H. J. Siegel, L. J. Siegel, F. C. Kemmerer, P. T. Mueller, Jr., H. E. Smalley, Jr., and S. D. Smith, "PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Trans. Comput.*, vol. C-30, pp. 934–947, Dec. 1981.

[15] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*. Lexington, MA: Lexington Books, 1985.

Nathaniel J. Davis IV (S'82) was born in Alexandria, VA, on February 13, 1954. He received the B.S. degree in 1976 and the M.S. degree in 1977, both in electrical engineering, from Virginia Polytechnic Institute and State University, Blacksburg, and the Ph.D. degree in electrical engineering in 1985 from Purdue University, West Lafayette, IN.

He is currently a Captain in the U.S. Army Signal Corps and is an Assistant Professor with the Department of Electrical Engineering, Air Force Institute of Technology, Wright–Patterson Air Force Base, OH. While attending Purdue University, he was a Graduate Research Assistant and worked on the development of the PASM parallel processing system. His previous assignments within the U.S. Army have been as an Instructor at the Air Force Institute of Technology and as a Communication-Electronics Officer with the 35th Signal Brigade (Corps) (Airborne), Fort Bragg, NC. His research interests include computer architecture, communications networks, interconnection networks, parallel processing, and system modeling and performance analysis.

Dr. Davis is a member of the IEEE Computer Society and the Eta Kappa Nu, Tau Beta Pi, and Sigma Xi honorary societies.



William Tsun-Yuk Hsu was born in Hong Kong on March 6, 1962. He received the B.S. degree in 1983 and the M.S. degree in 1985, both in electrical engineering, from Purdue University, West Lafayette, IN, and is now beginning work towards the Ph.D. degree at Purdue University.

He has worked as a Graduate Research Assistant on the design of the PASM parallel processing system. His research interests include computer architecture, parallel processing, interconnection networks, and graph theory.

Mr. Hsu is a member of the Eta Kappa Nu and Phi Kappa Phi honorary societies.

Howard Jay Siegel (M'77–SM'82) was born in New Jersey on January 16, 1950. He received the B.S. degree in electrical engineering and the B.S. degree in management from the Massachusetts Institute of Technology, Cambridge, MA, in 1972 and the M.A. and M.S.E. degrees in 1974 and the Ph.D. degree in 1977, all in electrical engineering and computer science, from Princeton University, Princeton, NJ.

In 1976 he joined the School of Electrical Engineering, Purdue University, West Lafayette, IN, where he is currently a Professor and Director of the PASM Parallel Processing Laboratory. His research interests include parallel/distributed processing, computer architecture, and image and speech understanding.

Dr. Siegel has published over 100 technical papers and is the author of the book *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies* (Lexington Books, 1985). He has served as Program Co-Chairperson of the 1983 International Conference on Parallel Processing, as the General Chairman of the Third International Conference on Distributed Computing Systems (1982), as an IEEE Computer Society Distinguished Visitor, as Chairman of the IEEE Computer Society Technical Committee on Computer Architecture (TCCA), as Chairman of the ACM Special Interest Group on Computer Architecture (SIGARCH), and as a Guest Editor of the IEEE TRANSACTIONS ON COMPUTERS. He is currently an Area (Associate) Editor of the *Journal of Parallel and Distributed Computing*. He is a member of the Eta Kappa Nu and Sigma Xi honorary societies.