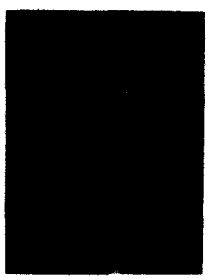
# Fault Recovery in Distributed Processing Loop Networks *

Raif M. YANNEY
*TRW Defense Systems Group, Redondo Beach, CA 90278, U.S.A.*

John P. HAYES
*Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, MI 48109, U.S.A.*

A graph model is introduced to formalize the fault recovery process in distributed loop networks. This model is applicable to centralized as well as distributed recovery. Key fault tolerance and recovery parameters including redundancy, fault model, recovery time, and recovery strategy are characterized. Centralized recovery strategies for a given fault-tolerant loop network are presented and analyzed. A distributed recovery strategy, which depends on the cooperation of a set of processors, is given, and its application to a new class of fault-tolerant loop networks is evaluated.

*Keywords:* Fault Tolerance, Fault Recovery, Recovery Strategy, Distributed Recovery, Loop Networks, Graph Models.



**Raif M. Yanney** is the Manager of Advanced Technology in the Systems Engineering and Development Division of TRW. He has been involved in the development of digital systems for the last 20 years. Before joining TRW in 1979, he was with Questron corporation, Hughes Aircraft company, and Cornell University. His research interest is in the areas of fault-tolerant systems and computer architecture.

Dr. Yanney holds a B.S.E.E. degree from Cairo University, a M.S.E.E. from Pratt Institute, and an Engineer degree and a Ph.D. from the University of Southern California, Los Angeles. He is a senior member of the IEEE and a member of the ACM and Sigma XI.

* This work was supported in part by the Office of Naval Research under Contract No. N15014-85-K-0531.

North-Holland
Computer Networks and ISDN Systems 15 (1988) 229–243

## 1. Introduction

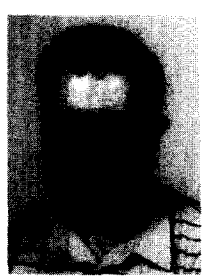
Fault-tolerant computer systems are often organized as a network of processors or computers, in which fault tolerance is achieved by dynamically reconfiguring around faulty units [1,2]. Three basic processes are needed to deal with faults in these systems: diagnosis, reconfiguration, and recovery. Of these, fault diagnosis has received the most research attention in the last two decades. Relatively little research has been reported on the formal modeling of reconfiguration and recovery [5,7,10,13].

A loop network may be defined as a closed communication channel with a set of processors and related devices that are attached to it by interface circuits [6]. Loop structures are widely used for distributed processing systems. Their main advantages are easy message routing and low implementation and expansion costs. They also are inherently fault-tolerant since

(a) two paths link every pair of processors; if one fails, communication can continue via the other path;

(b) a faulty processor can easily be bypassed.

The structure and behavior of such networks can become quite complex if extra processors and communication links are added to enhance the fault tolerance of the basic underlying loop net-



**John P. Hayes** is a professor in the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor. He is also the director of the Advanced Computer Architecture Laboratory at the university. Before joining the University of Michigan in 1982, he served on the faculty of the University of Southern California, Los Angeles. His research interest is in the areas of computer architecture, VLSI design, and fault-tolerant computing. He is the author of *Digital System Design and Microprocessors* (McGraw-Hill, 1984), *Computer Architecture and Organization* (2nd ed., McGraw-Hill, 1988) and over 90 technical papers.

Dr. Hayes received a B.E. degree from the National University of Ireland, Dublin, and his M.S. and Ph.D degrees from the University of Illinois, Urbana-Champaign. He is a fellow of IEEE and a member of ACM and Sigma Xi.

work. Recovery is also greatly complicated if the recovery mechanisms are distributed throughout the system.

Although the design of fault-tolerant loop networks has been previously studied [4,14], the reconfiguration and recovery mechanisms necessary for high levels of fault tolerance or distributed recovery do not seem to have been considered. In this paper we extend the graph-theoretical methodology for fault recovery presented in [5] and apply it to the analysis of both centralized and distributed recovery in loop networks. Section 2 summarizes the properties of the facility graph used to model loop networks and defines the recovery model within this framework. The properties of existing fault-tolerant loop networks [4] are discussed in Section 3. Efficient centralized recovery strategies are given and analyzed for those systems. Distributed recovery, which is intended for loop networks with no central supervisor, is introduced in Section 4. Distributed recovery depends on the cooperation of a set of nodes to execute the recovery strategy since each node is assumed to have only a limited amount of information about the system as a whole. The recovery problems encountered with such networks are analyzed, and an efficient fault-tolerant design and recovery procedures that circumvent these problems are presented.

## 2. Fault Recovery Model

A loop network is described here using a *facility graph* [4] which is an undirected labeled or unlabeled graph whose node represent the system components or facilities and whose edges represent interconnections between facilities. A *facility* is any node of the network that can fail independently of the remaining nodes. Two facility graphs $G$ and $H$ are *isomorphic* ($G = H$) if there exists a one-to-one correspondence between their node sets that preserves adjacency. If $G_1$ and $G_2$ are two facility graphs whose nodes are labeled by the same set of names $\{x_i\}$, then $G_1$ and $G_2$ are *L-isomorphic* ($G_1 =_L G_2$) if and only if there exists a one-to-one correspondence between their node sets that preserves labeling and adjacency. A *basic graph* is a labeled facility graph that represents the minimum system configuration needed to perform a certain set of tasks. A basic system by definition cannot tolerate any faults. A loop network containing $n$ processors is modeled by the basic graph $C_n$, which is a cycle or closed path with $n$ distinct nodes. The labels of $C_n$ represent tasks assigned to the nodes of $C_n$. A *redundant graph* $G_r$ with respect to a basic graph $C_n$ is an unlabeled graph that contains a subgraph isomorphic to $C_n$. $G_r$ is viewed as a redundant and possibly fault-tolerant realization of $C_n$. $G_r$ will also be written as $G_r[C_n]$ to indicate that $C_n$ is the underlying basic graph.

A *fault* $F_x$ is represented by the removal of the node $x$ from the redundant graph $G_r$. The removal of a node from the graph also requires the removal of all edges incident on the node. All nodes of $G_r$ are assumed to be of the same type and to have the same processing capability; hence, a faulty node can be replaced by any fault-free node that has the necessary edge connections. A set of $k$ faults affecting $k$ distinct nodes of $G_r$ is called a *k-fault*. $G_r[C_n]$ is *k-fault tolerant* ($k$-FT) [4] if the removal of any $k$ nodes and their associated edges from $G_r$ results in a graph that contains a subgraph isomorphic to $C_n$.

Reconfigurability is defined as a system's ability to change its physical or functional organization in response to change in the system's computational requirements or the occurrence of faults [9]. Recovery, on the other hand, comprises all actions that are initiated by the detection of a fault and concluded by

(a) resumption of normal operation (possibly in a degraded mode); or

(b) a systematic shutdown of the system.

Although the recovery process covers all actions, including reconfiguration, taken by a fault-tolerant system to circumvent the effect of a fault, we will reserve the term "recovery" to refer to reconfiguration processes that are expected to terminate successfully.

It is assumed that the systems of interest contain a mechanism for continuous self-diagnosis. The precise manner in which diagnosis is achieved is not of direct interest here. Normally the nodes in some subgraph $C_n' = C_n$ of $G_r[C_n]$ are engaged in computation, or are *active*, while the remaining nodes of $G_r$ are *inactive* or *spare* nodes. After the detection of a fault $F_x$ affecting a node $x$ in $C_n'$ the active subgraph changes from $C_n'$ to $C_n' - x$. In order for the system to recover, a new subgraph $C_n''$ that is isomorphic to the basic graph $C_n$ and has no faulty nodes must be found.

We look on reconfiguration as a binding between physical resources represented by the nodes of $G_r$ and logical resources or tasks represented by the nodes of $C_n$ [10]. Thus a task defines the *state* $s(x)$ of a node $x$. Every node $x$ of $G_r$ is assumed to be in one of $n+2$ possible states:

(a) $n$ active states denoted by $1, 2, \ldots, n$;

(b) an inactive or spare state denoted by 0; and

(c) a faulty state denoted by $-1$.

The state $S(G_r)$ of the $m$-node system $G_r$ is the $m$-tuple $S(x_1, \ldots, x_i, \ldots, x_m) = [S(x_1), \ldots, S(x_i), \ldots, S(x_m)]$ where $S(x_i)$ is the state of node $x_i$.

The state $S(G_r)$ defines a particular labeling of $G_r$, leading to the next definition.

**Definition 1.** Let $G_r[C_n]$ be a redundant system. A *configuration* $G_c$ of $G_r[C_n]$ is a labeled graph isomorphic to $G_r$ in which node $x_i$ is labeled with the state $S(x_i)$. A configuration $G_c$ is a *valid configuration* if it has exactly $n$ distinct active nodes, and contains a subgraph L-isomorphic to $C_n$.

If $G_c$ is valid, then $S(G_c)$ is called a *valid state* of $G_r[C_n]$.

Typically, every node of a subgraph $C_n'$ of a valid configuration $G_c$ is assigned one of the $n$ active states to make $C_n' =_L C_n$. The nodes of $G_c - C_n'$ which are not faulty are spare nodes and are assigned the state 0. The state $-1$ is assigned to any node in $G_c$ that develops a fault. A fault $F$ thus transfers $G_r[C_n]$ from a valid state $S$ to a faulty state $S_F$ by changing the state of all nodes affected by $F$ to $-1$.

The *reconfiguration problem* of interest may now be defined formally. Let $G_r[C_n]$ be a redundant graph that represents a fault-tolerant realization of a basic graph $C_n$. If $G_c$ is the initial valid configuration and a fault $F$ affecting $G_r[C_n]$ occurs, find a new configuration $G_c'$ of $G_r[C_n]$, if such a configuration exists, that contains a subgraph $C_n' =_L C_n$. The general reconfiguration problem involves finding a fault-free unlabeled subgraph $G_s$ of $G_r$ such that $G_s = C_n$, and then labeling it to obtain the subgraph $C_n' =_L C_n$. The problem of finding $G_s$ is the well-known subgraph isomorphism problem. While the general subgraph isomorphism problem is computationally very complex (NP-complete), efficient (polynomial time) algorithms are known for some special classes of graphs [8]. Efficient heuristic procedures are also known for the general case [11]. In this paper we impose restrictions on the systems of interest that effectively avoid the major difficulties of the subgraph isomorphism problem and also lead to fast recovery.

A *reconfiguration function* $R_F$ with respect to $F$ is a one-to-one mapping

$$R_F: \{S(G_r)\} \rightarrow \{S(G_r)\}$$

that transfers $G_r[C_n]$ from one state to another. Throughout this paper we consider only those reconfiguration functions that obey the following constraints:

(a) a node in an active state may be changed either to another active state or to the inactive state;

(b) a node in an inactive state may be changed only to an active state; or

(c) a node in a faulty state must remain in the faulty state, since repair or physical replacement of faulty nodes is not considered.

**Definition 2.** Let $S_0$ be a valid state of $G_r[C_n]$ and let $F$ be a fault that changes $S_0$ to $S_F$. A *p-step reconfiguration process* with respect to $S_0$, $F$ and a reconfiguration function $R_F$ is a state sequence

$$P(S_0, F, R_F) = S_F, S_1, S_2, \ldots, S_p$$

where $S_{i+1} = R_F(S_i) = R_F^i(S_F)$ for $i = 1, 2, \ldots, p$.

$P(S_0, F, R_F)$ is a *p-step recovery process* if $G_{cp}$ contains a subgraph L-isomorphic to $C_n$, where $G_{cp}$ is the configuration that corresponds to $S_p$.

It should be noted that one step in a recovery process corresponds to a single transition between two states of $G_r[C_n]$ as defined by $R_F$. A *reconfiguration strategy* $R$ for a set of faults $\{F\}$ is a set of reconfiguration functions $\{R_F\}$.

**Definition 3.** A *p-step recovery strategy* $R$ for $G_r$, $C_n$, $S(G_r)$ and $\{F\}$ is a reconfiguration strategy $\{R_F\}$, such that for every fault of the set $\{F\}$ and every valid state $S$ there exists $i \leq p$ such that

$$R_F^i(S_F) = S_i$$

where $S_i$ is a valid state. We say that $R$ *recovers* $G_r[C_n]$ from any fault of the set $\{F\}$.

A $k$-FT system can tolerate up to $k$ faults [4]. A system that achieves $k$-fault tolerance using a particular recovery strategy is called a *k-fault*

*recoverable system* with respect to that recovery strategy. Unless otherwise stated, $G_r$ may contain any valid initial configuration or state before a fault occurs.

**Definition 4.** A system $G_r[C_n]$ is *p-step k-fault recoverable* ( *p*-step *k*-FR)—or simply *k*-FR—with respect to the reconfiguration strategy $R$, if $R$ is a $p$-step recovery strategy with respect to any fault affecting at most $k$ nodes in $G_r$.

The recovery process often involves a considerable amount of information transfer among the system nodes. The number of fault-free nodes whose state is changed when reconfiguring the system to recover from a fault is taken as a measure of the system recovery time, and leads to the following definition.

**Definition 5.** A $k$-FT system $G_r[C_n]$ is *t-node recoverable* ($t$-NR) with respect to a recovery strategy $R$, if $R$ can recover the system from up to $k$ faults by changing the state of at most $t$ fault-free nodes.

Obviously, we must have $t \geqslant k$, and recovery can take place in at most $t$ steps. Note that $t$ represents the cumulative number of 1-node state changes occurring during the entire recovery process. We assume that each node changes state at most once during a recovery step.

In [5] a general class of $t$-FT $t$-NR, or simply $t$-NR, designs, denoted $G_t^{OPT}$, were specified and characterized. $G_t^{OPT}$ may be defined as follows for loop networks.

**Definition 6.** The *optimal t-NR redundant graph* $G_t^{OPT}$ realizing the basic graph $C_n$ is an $(n+t)$-node graph constructed as follows:

*Step* 1. Let $G_g$, the *generator graph* of $G_t^{OPT}$, be an $n$-node labeled graph L-isomorphic to $C_n$. Add $t$ spare nodes $x_{s1}, \ldots, x_{st}$ to $G_g$.

*Step* 2. Connect each spare node $x_{si}$ to every node in $G_g$ and to the other $t-1$ spares.

$G_t^{OPT}$ allows recovery to be achieved in the minimum time using the following simple recovery strategy $R_0$: replace each faulty node by the next available spare. (A more precise definition of $R_0$ can be found in [15].)

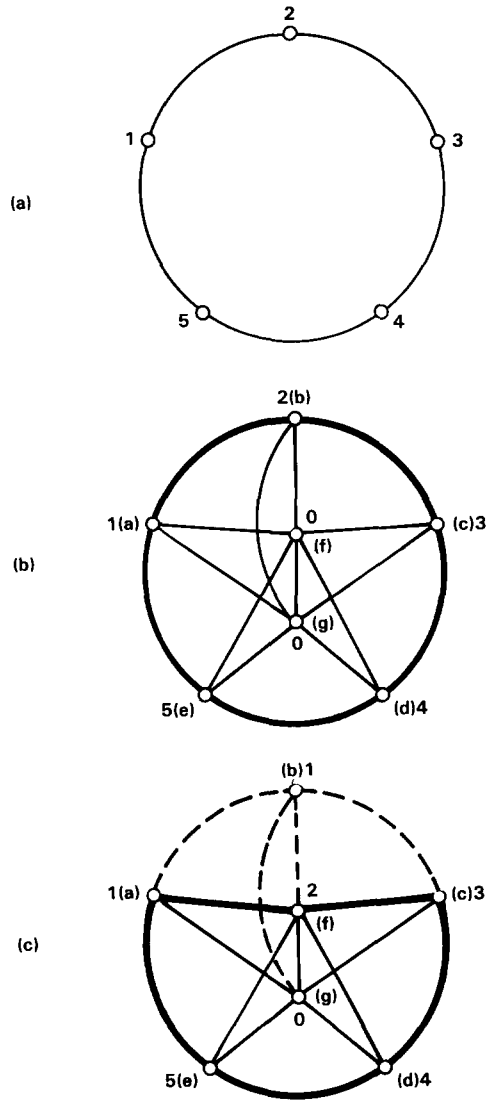



Fig. 1. (a) The basic cyclic graph $C_5$; (b) the optimal 2-NR redundant system $G_2^{opt}[C_5]$ with a valid configuration; (c) the system after recovery form a 1-fault.

**Example 1.** Figure 1(a) shows the labeled basic graph $C_5$ representing a 5-node loop network. The optimal 2-NR realization $G_2^{OPT}[C_5]$ of $C_5$ specified by Definition 6 appears in Fig. 1(b). The generator graph $G_g$ of $G_2^{OPT}[C_5]$ is shown in heavy lines. In a typical application $G_g$ is labeled as indicated so that $G_g =_L C_n$, and the figure constitutes a valid configuration of $G_2^{OPT}$. The corresponding valid state is

$$S_0(a, b, c, d, e, f, g) = (1, 2, 3, 4, 5, 0, 0).$$

Suppose that a 1-fault affecting node $b$ occurs, causing its state to change from 2 to $-1$. Applica-

tion of $R_0$ causes one of the spares to change its state from 0 to 2, thereby replacing the faulty node. As can be seen from Fig. 1(c), the recovered system contains a fault-free subgraph L-isomorphic to $C_5$. The system state at this point is

$$S_1(a, b, c, d, e, f, g) = (1, -1, 3, 4, 5, 2, 0).$$

A second faulty node can be similarly tolerated by a second application of $R_0$; recovery from a 2-fault using $R_0$ is therefore a 2-step process.

## 3. Centralized Recovery

While the general redundant design $G_t^{\mathrm{OPT}}[C_n]$ defined in Section 2 uses the minimum number of spare nodes ($k = t$) and has a very simple recovery strategy ($R_0$), it has the disadvantage of requiring a large number of edges, and consequently has nodes of relatively high degree. In [4] a class of optimal $k$-fault-tolerant realizations of $C_n$ denoted $C_{n,k}$ was defined which also employ $k$ spare nodes but have the minimum possible number of edges. This reduction in the number of redundant edges means that simple reconfiguration strategies like $R_0$ can no longer be used for recovery. We next present a recovery strategy $R_1$ for $C_{n,k}$ which shows that fast recovery is possible in such systems. Throughout this section we assume that recovery is centralized in a system supervisor that has complete information about the system's operational status and interconnection structure. The central supervisor is also able to transfer the state of a faulty node to any available fault-free node of the system. This implies that it must maintain backup files, check-point data, etc., concerning all active nodes of the system.

First we consider centralized recovery for $C_{n,k}$ where $k = 1$. (Note that the structure of $C_{n,k}$ is slightly simpler when $k \geq 2$ [4].) Figures 2(a) and (b) show $C_{n,1}$ for $n$ odd and even, respectively. Recovery in $C_{n,1}$ using $R_1$ depends on identifying nodes with degree 2 in the graph $C_{n,1} - x_i$, where $x_i$ is a faulty node of $C_{n,1}$. The edges incident on these nodes must be edges in the recovered system. These edges are identified and used systematically to build up segments of a cycle that is L-isomorphic to $C_n$. We now give a formal definition of the recovery strategy $R_1$ for the 1-FT case.

**Procedure 1:** Cycle recovery strategy $R_1$ for $C_{n,1}$.

*Step* 1. Let $S = (s_1, s_2, \ldots, s_{n+1})$ be the current valid state of $C_{n,1}$ and let the active subgraph be $C_n' =_{\mathrm{L}} C_n$. Let a fault $F$ affecting node $x_i$ occur that changes the state of $C_{n,1}$ from $S$ to $S_F$. If $x_i$ is not in $C_n$, implying that $s_i \in \{-1, 0\}$, then $R_1(S_F) = S_F$, i.e., $R_1$ leaves the system state unchanged.

*Step* 2. If $x_i$ is in $C_n'$, implying that $x_i$ is an active node, scan $S_F$ from left to right until a component $s_j = 0$ is found, and go to Step 3. If no $s_j = 0$ exists, then $R_1(S_F) = S_F$, and the recovery attempt is terminated unsuccessfully.

*Step* 3. Using Procedure 2 given below, generate the $n$-node path $x_1 \ldots x_i \ldots x_n$ that corresponds to the cycle $C_n''$.

*Step* 4. Change the states of the $n$ active nodes of the cycle $C_n'' = x_1 \ldots x_i \ldots x_n$ such that a node $x_i$ is assigned state $S(x_i) = i$. $C_n''$ is L-isomorphic to $C_n$.

Procedure 2, which is a subprocedure of Procedure 1, is used to find an unlabeled $n$-node cycle in the graph $C_{n,1} - x$. It generates iteratively edge-disjoint paths of a cycle. The endpoints of these paths are marked by an asterisk. If a node is marked by two asterisks, then the corresponding paths are concatenated to generate a larger path.

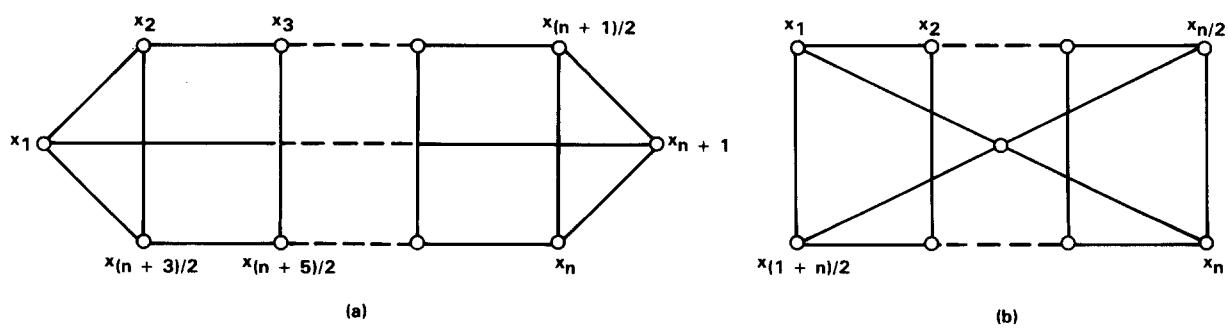



Fig. 2. The optimal 1-FT cyclic system: (a) $C_{n,1}$ for $n$ odd; (b) $C_{n,1}$ for $n$ even.

**Procedure 2:** To find a cycle $C_n$ in the graph $C_{n,1} - x$.

*Step* 1. Construct the adjacency matrix $A$ [3] of $C_{n,1} - x$.

*Step* 2. If any path generated so far has $n$ nodes, then it corresponds to the required $n$-node cycle $C_n$ and the procedure terminates.

*Step* 3. Scan the rows of $A$ from top to bottom. If any unmarked row $x_j$ of $A$ has exactly two 1's in columns $x_i$ and $x_k$, generate the 3-node path $x_i x_j x_k$. Delete column $x_j$ and row $x_j$ from $A$. Mark rows $x_i$ and $x_k$ with an asterisk. If no rows satisfying the forgoing conditions are found during the current pass through Step 3, go to Step 6; otherwise go to the next step.

*Step* 4. Scan $A$ from top to bottom. If any row, say $x_i$, has two asterisk marks, then $x_i$ is an endpoint of two paths. Concatenate these two paths to form a single path, and delete row $x_i$ and column $x_i$ from $A$.

*Step* 5. Scan $A$ from top to bottom. If any row, say $x_i$, has one asterisk mark and has a single 1-entry, say $a_{ij}$, then $x_i$ is an endpoint of the paths $P$ generated so far. Append the node $x_j$ adjacent to $x_i$ to the path $P$, delete column $x_i$ and row $x_i$ from $A$, mark row $x_j$ with an asterisk and go to Step 2.

*Step* 6. Scan $A$ from top to bottom. If any row, say $x_i$, with one asterisk mark has two 1-entries, then one of the two corresponding edges, say $x_i x_j$, is discarded since if added to $P$ it will create an $m$-node cycle, where $m < n$. Change both the $a_{ij}$ and $a_{ji}$ entries to 0 and go to Step 2.

**Example 2.** Figure 3(a) shows the optimal 1-FT graph $C_{8,1}$ and an initial valid configuration $G_c$. Suppose that node $i$ in state 8 becomes faulty, resulting in the invalid configuration of Fig. 3(b). Now consider the application of recovery strategy $R_1$ as defined by Procedure 1. The faulty state is

$$S_F(a, b, c, d, e, f, g, h, i) =$$
$$(1, 2, 3, 4, 5, 6, 7, 0, -1).$$

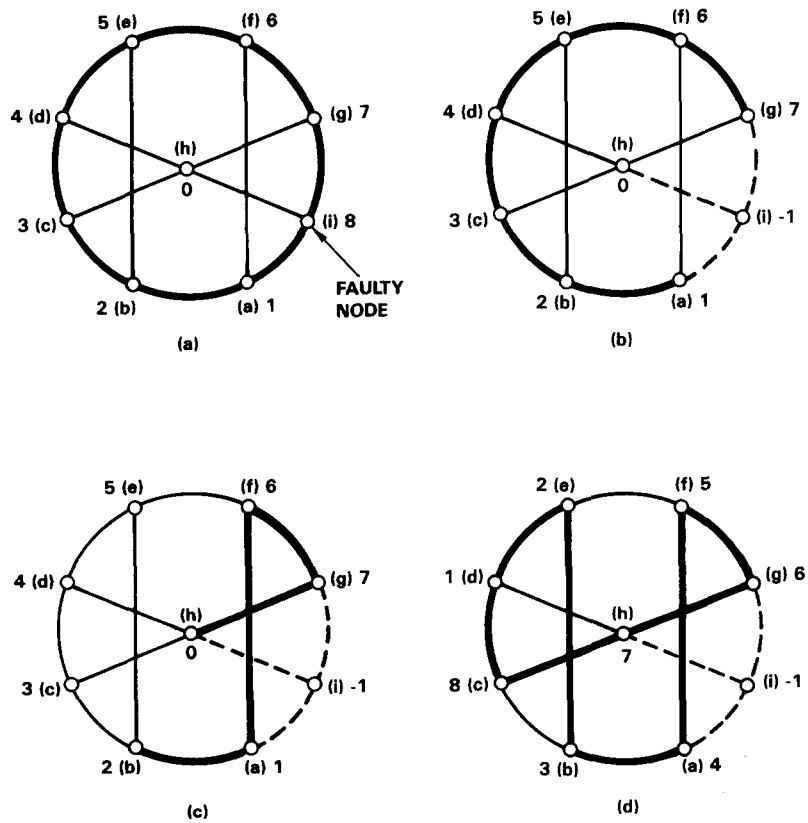



Fig. 3. The optimal 1-FT system $C_{8,1}[C_8]$: (a) the graph $C_{8,1}$ with a valid initial configuration; (b) configuration after fault $F_i$ occurs; (c) intermediate stage in recovery; (d) final recovered system.

Since $S(h) = 0$. Step 3 of the procedure invokes Procedure 2. The adjacency matrix $A$ of $C_{8,1} - i$ is

$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}.$$

Row $a$ has 1's in columns $b$ and $f$; therefore Step 3 of Procedure 2 yields the path $baf$. Row $a$ and column $a$ are now deleted, and the rows $b$ and $f$ are marked with asterisks. Row $g$ also contains two 1-entries, and yields the path $fgh$. We then delete the $g$ row and column, and marks rows $f$ and $h$ producing the following reduced adjacency matrix:

$$A = \begin{array}{c} \\ b \\ c \\ d \\ e \\ f \\ h \end{array} \begin{array}{c} \begin{array}{cccccc} b & c & d & e & f & h \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{l} \\ * \\ \\ * \ * \\ * \\ \end{array}.$$

Row $f$ has two asterisks; hence, according to Step 4 of Procedure 2, the paths $baf$ and $fgh$ are concatenated to form the new path $bafgh$ as shown by heavy lines in Fig. 3(c). Note that since edges $af$ and $fg$ are now included in the partial cycle, we conclude that the path $ef$ cannot be part of that cycle. Continued application of Procedure 2 yields the 8-node cycle $C_8''$ shown by heavy lines in Fig. 3(c). Step 4 of Procedure 1 now relabels $C_8''$ to make it L-isomorphic to the original basic graph $C_8$.

We next demonstrate the validity of recovery strategy $R_1$ by proving that Procedure 2 always identifies an $n$-node cycle in $C_{n,1} - x$.

**Theorem 1.** *Let $C_{n,1}$ be the optimal* 1-FT *realization of the cycle $C_n$, and let $x$ be any single faulty node in $C_{n-1}$. Procedure* 2 *finds an $n$-node cycle in $C_{n,1} - x$ that is isomorphic to $C_n$.*

**Proof.** The procedure determines whether or not an edge of $C_{n,1} - x$ is to form part of $C_n$ as follows:

(a) Step 3: If a node $x_i$ in the original graph $C_{n,1} - x$, or subsequently after edges are deleted, has degree 2, then both edges incident on $x_i$ must be edges of $C_n'' =_L C_n$. The graph $C_{n,1}$ can have no more than one node with degree 4; all other nodes have degree 3. Hence the fault affecting $x$ always leaves at least two nodes with degree 2, allowing Step 3 to be executed at least once to initiate Procedure 2.

(b) Step 4: If two paths are concatenated, any edges incident on the common endpoint that have not been selected so far can be eliminated; this is indicated by deleting the appropriate rows and columns of the adjacency matrix.

(c) Step 6: If adding an edge $x_i x_j$ to a selected path generates a cycle $C_m$, where $m < n$, then $a_{ij}$ and $a_{ij}$ are eliminated by changing the corresponding entry of the adjacency matrix from 1 to 0.

We now show that in every iteration through the procedure, at least one edge can be identified as either being a part of $C_n''$, or not being a part of $C_n''$. Consider the case where $n$ is even. The optimal 1-FT graph $C_{n,1}$ is shown again in Fig. 4(a). Suppose that the center node $x_{n+1}$ is active and is effectively removed by a fault. The resulting graph $C_{n,1} - x_{n+1}$ is shown in Fig. 4(b). In the first iteration through the procedure the paths $x_2x_1x_nx_{n-1}$ and $x_{(n-2)/2}x_{n/2}x_{(n+2)/2}x_{(n+4)/2}$ are identified as segments of $C_n''$. Also the edges $x_2x_{n-1}$ and $x_{(n-2)/2}x_{(n+4)/2}$ are identified as not being a part of $C_n''$. In subsequent iterations the other vertical edges are identified as not being part of $C_n''$ since they produce $m$-node cycles with $m < n$.

The only other fault we need to consider is the removal of a perimeter node $x_i$ from $C_{n,1}$, where $i \neq n+1$. We consider the representative case where $i = 2$. The graph $C_{n,1} - x_2$ is shown in Fig. 4(c). In the first iteration through the procedure the path $P = x_4x_3x_{n-2}x_{n-1}x_n$ is identified as a segment of $C_n''$. Also the edge $x_{n-2}x_{n-3}$ is identified as not being part of $C_n''$. In the second iteration the path $x_4x_{n-3}x_{n-4}$ is added to the path $P$. This implies that the edge $x_4x_5$ is not part of $C_n''$. This process continues until all edges of $C_n''$ have been identified. The proof for $n$ odd is similar. □

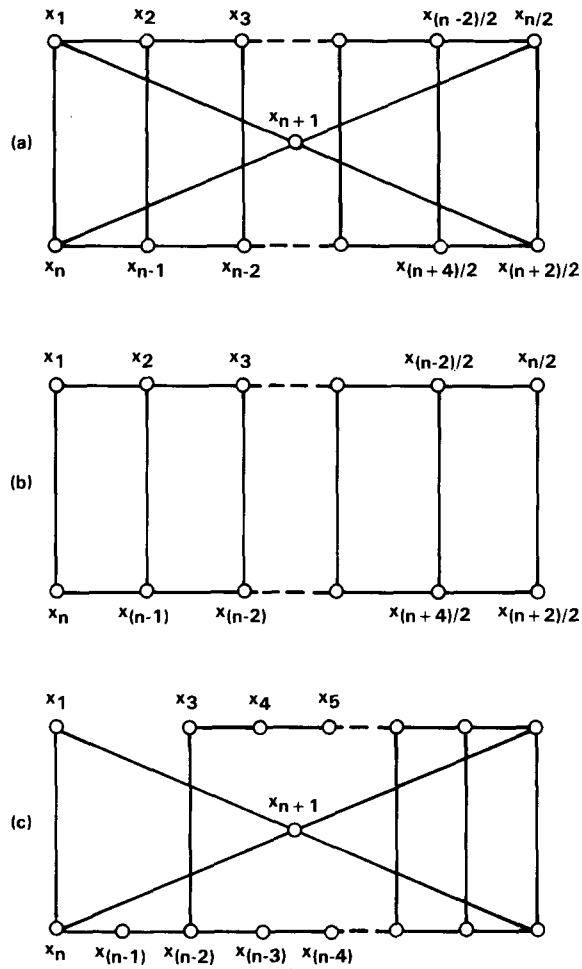



Fig. 4. Proof of Theorem 1: (a) the optimal 1-FT graph $C_{n,1}$; (b) $C_{n,1} - x_{n+1}$; (c) $C_{n,1} - x_2$.

Optimal $k$-FT loop networks $C_{n,k}$ where $k \geq 2$ are somewhat easier to characterize than the 1-FT case. We now discuss recovery in $C_{n,k}$ when $k \geq 2$ and $k$ is even.

**Procedure 3.** To construct an optimal $k$-FT realization $C_{n,k}$ of the cycle $C_n$ when $k = 2p$ is even [4].

*Step* 1. Form the cycle $C_{n+k}$, which is called the *generator cycle* of $C_{n,2p}$.

*Step* 2. Join every node $x_i$ of $C_{n+k}$ to all nodes at distance $j$ from $x_i$ in $C_{n+k}$, for all $j$ satisfying $2 \leq j \leq p+1$. The resulting graph is $C_{n,2p}$.

The *power graph* $G^m$ [3] of $G$ is constructed as follows: add edges to $G$ so that every node $x$ is connected to all nodes at distance $d \leq m$ from $x$. We refer to $G$ as the *generator graph* of $G^m$. It is obvious that the optimal $k$-FT graph $C_{n,2p}$ constructed using Procedure 3 is isomorphic to the cycle power graph $C_{n+k}^{p+1}$. Figure 5 shows the graph $C_{8,4}$ which is isomorphic to $C_{12}^3$, and according to [16] is a 4-FT realization of the cycle $C_8$.

We now define a recovery strategy $R_2$ for $C_{n,2p}$, which exploits certain properties of cycle power graphs. In the sequel we assume that the nodes of $C_{n,k}$ are named $x_1, x_2, \ldots, x_{n+k}$ according to their location in the generator cycle $C_{n+k}$. A set of $m$ consecutive nodes $x_i, x_{i+1}, \ldots, x_{i+m}$ in the generator cycle $C_{n+k}$ of $C_{n,k}$ is called an $m$-node *cluster*. An *$m$-fault cluster*, accordingly, is an $m$-fault that affects an $m$-node cluster. $R_2$ exploits the fact that any $k$-fault that may affect $C_{n,k}$ belongs to one of two basic classes. These fault classes are illustrated in Fig. 6 for the graph $C_{8,4}$ of Fig. 5. If an $m$-fault cluster affecting $C_{n,k}$ satisfies the relation $m \leq p$, then the subgraph consisting of the $n$ fault-free nodes is similar to the one shown in Fig. 6(a). The $n$ fault-free nodes are connected in a cycle around the perimeter of the faulty graph as shown by heavy lines. If $C_{n,k}$ has an $m$-fault cluster, where $m \geq p+1$, then every other $d$-node cluster satisfies the relation $d < p$. In this case the $n$-node fault-free subgraph of $C_{n,k}$ is similar to the graph of Fig. 6(b). Again the heavy lines show a subgraph isomorphic to $C_n$. $R_2$ uses the state vector of $C_{n,k}$ to label the first $n$ fault-free nodes with the states $1, 2, \ldots, n$. In the case of Fig. 6(a), it labels the nodes as they are first encountered. In

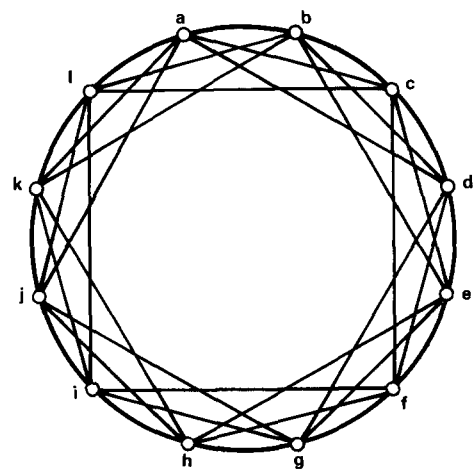



Fig. 5. The optimal 4-FT graph $C_{8,4}$.

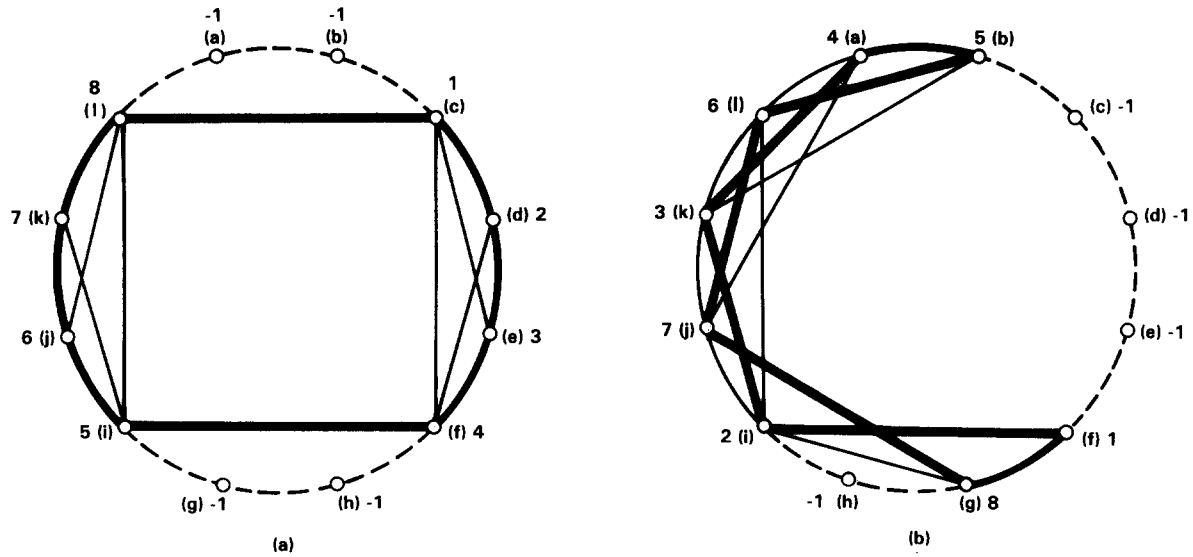



Fig. 6. Fault clusters in $C_{8,4}$: (a) two 2-fault clusters; (b) a 3-fault and a 1-fault cluster.

the case of Fig. 6(b) it labels the nodes alternately, skipping every second node encountered.

**Procedure 4:** Cycle recovery strategy $R_2$ for $C_{n,2p}$.

*Step* 1. Let $S(x_1, x_2, \ldots, x_{n+k}) = (s_1, s_2, \ldots, s_{n+k})$ be the current valid state of $C_{n,k}$ and let the active subgraph be $C_n' =_L C_n$. Let a fault $F$ affecting node $x_i$ occur that changes the state of $C_{n,k}$ from $S$ to $S_F$.

*Step* 2. If $x_i$ is not in $C_n'$, implying that $s_i \in \{0, -1\}$, then $R_2(S_F) = S_F$, i.e., $R_2$ leaves the system state unchanged.

*Step* 3. If node $x_i$ is in $C_n'$, implying that $x_i$ is an active node and $s_i \in \{1, 2, \ldots, n\}$, scan $S_F$ from left to right changing the state of every fault-free active node $x_i$ from $s_i$ to 0 to generate the state $S_0$. Set an index $I$ to 0.

*Step* 4. While $I \leqslant n$, scan $S_i$ from left to right until a component $s_k = 0$ is found. If there is a node $x_j$ in state $I$ adjacent to the node $x_k$, change $s_k$ from 0 to $I+1$ and $S_j$ to $S_{i+1}$. Increment $I$ and continue. If at any point no $s_k = 0$ is found, then $R_2(S_i) = S_i$, and the recovery attempts fails (no spares available). If at any point there is a component $s_k = 0$, but there is no node $x_j$ in state $I$ that is adjacent to the node $x_k$, go to Step 6.

*Step* 5. If the nodes in state 1 and $n$ are adjacent, then the $n$ nodes labeled 1, 2, ..., $n$ represent the cycle $C_n$, and the procedure terminates. Otherwise go to Step 7.

*Step* 6. Scan $S_i$ from left to right changing the state of every fault-free node to 0. Change the component $s_k$ from 0 to 1 to generate the state $S_1'$. Rotate $S_1'$ until the component $s_k$ becomes the first component in $S_1'$. Go to Step 8.

*Step* 7. Scan $S_n$ from left to right changing the state of the first $n$ fault-free nodes to 0 and change the state of every other fault-free node to a number $N > n$, yielding the state $S_0'$. Scan $S_0'$ from left to right to find a component $s_j = 0$. Change $s_j$ from 0 to 1 to generate the state $S_1'$. Set an index $I$ to 1.

*Step* 8. While $I \leqslant n$, scan $S_1'$ from left to right, starting at the component $s_i = I$, to find a pair of components $s_i = 0$ and $s_k = 0$. Change the state of $s_k$ from 0 to $I+1$ and $S_i'$ to $S_{i+1}$. Increment $I$ and continue. If at any time there is no pair of components satisfying the forgoing conditions, go to Step 9.

*Step* 9. While $I \leqslant n$, scan $S_i'$ from right to left to find a component $s_j = 0$. Change $s_j$ from 0 to $I+1$. Increment $I$ and continue.

*Step* 10. Scan $S_n$ from left to right changing every component $s_j = N$, if any, to 0. The nodes of the required cycle $C_n$ are now labeled 1, 2, ..., $n$.

**Example 3.** Again we consider the optimal 4-FT system $C_{8,4}$ shown in Fig. 5. Suppose that a 4-fault occurs affecting the nodes $c$, $d$, $e$, and $h$. We now use the recovery strategy $R_2$ to recover an 8-node

cycle $C_8$ from the faulty graph. Using Step 3 of Procedure 4, set the state of every fault-free node to 0 thus

$$S_0(a, b, c, d, e, f, g, h, i, j, k, l) = (0, 0, -1, -1, -1, 0, 0, -1, 0, 0, 0, 0).$$

Step 4 of the procedure causes $S_0$ to be scanned from left to right. The spare nodes are labeled $1, 2, \ldots, n$ in the order that they are encountered, yielding the following sequence of states:

$$S_1 = (1, 0, -1, -1, -1, 0, 0, -1, 0, 0, 0, 0),$$

$$S_2 = (1, 2, -1, -1, -1, 0, 0, -1, 0, 0, 0, 0).$$

At this point the next 0 encountered in $S_2$ is $s_6$ which corresponds to node $f$. Step 4 cannot assign node $f$ to state 3, since $f$ is not adjacent to node $b$, and $b$ is already assigned to state 2. Hence Step 6 is executed which changes the state of the first $n$ fault-free nodes in $S_2$ back to 0 and changes the state of $f$ to 1 to produce the state $S_1$:

$$S_1(a, b, c\, d, e, f, g, h, i, j, k, l) = (0, 0, -1, -1, -1, 1, 0, -1, 0, 0, 0, 0).$$

Then $S_1$ is rotated until the leftmost component corresponds to $f$ thus

$$S_1'(f, g, h, i, j, k, l, a, b, c, d, e) = (1, 0, -1, 0, 0, 0, 0, 0, 0, -1, -1, -1).$$

According to Step 8 of the procedure, we must now label every other 0-component in $S_i$ with the labels $1, 2, \ldots, n$. Step 8 therefore produces the following sequence of states:

$$S_2 = (1, 0, -1, 2, 0, 0, 0, 0, 0, -1, -1, -1),$$

$$S_3 = (1, 0, -1, 2, 0, 3, 0, 0, 0, -1, -1, -1),$$

$$S_4 = (1, 0, -1, 2, 0, 3, 0, 4, 0, -1, -1, -1).$$

Executing Step 9 of the procedure we scan $S_4$ from right to left labeling 0-entries in sequence:

$$S_5 = (1, 0, -1, 2, 0, 3, 0, 4, 5, -1, -1, -1),$$

$$S_6 = (1, 0, -1, 2, 0, 3, 6, 4, 5, -1, -1, -1),$$

$$S_7 = (1, 0, -1, 2, 7, 3, 6, 4, 5, -1, -1, -1),$$

$$S_8 = (1, 8, -1, 2, 7, 3, 6, 4, 5, -1, -1, -1).$$

The final state obtained by rotating the components of the state vector $S_8$ back to their normal positions is

$$S_8'(a, b, c, d, e, f, g, h, i, j, k, l) = (4, 5, -1, -1, -1, 1, 8, -1, 2, 7, 3, 6).$$

The nodes of $C_{8,4}$ are now labeled $1, 2, \ldots, n$ corresponding to $S_8'$. The valid configuration of the recovered system is shown in Fig. 6(b).

It can readily be shown that $R_2$ can always recover $C_{n,2p}$ from $2p$-faults [15]. $R_2$ can also be easily modified to deal with $C_{n,k}$ when $k$ is odd. We can therefore state the following result.

**Theorem 2.** *The optimal* $k$-FT *system* $C_{n,2p}$ *is* $k$-FR *with respect to the recovery strategy* $R_2$.

It is worth noting that $G_k^{\mathrm{OPT}}[C_n]$ and $C_{n,k}$ represent two extreme cases of $k$-FT realizations of $C_n$. $C_{n,k}$ employs the minimum number of edges consistent with having the minimum number of nodes, whereas $G_k^{\mathrm{OPT}}$ employs close to the maximal number of edges. Recovery in $G_k^{\mathrm{OPT}}$ via $R_0$ is the fastest possible in terms of the number of nodes that must change state. While $R_2$ is a fast recovery strategy for $C_{n,k}$, the spare interconnection structure of $C_{n,k}$ inherently requires a large number of nodes to change state, resulting in much longer recovery time. Other fault-tolerant loop networks and their (centralized) recovery strategies can be expected to fall between these extremes.

## 4. Distributed Recovery

In the fault-tolerant systems considered so far it has been assumed that recovery is directed by a single supervisor. Centralized recovery of this type has two drawbacks. First the supervisor is vulnerable, and needs added redundancy for its protection. The other drawback is the processing overhead of monitoring the operational status of all processors of the system. In this section the use of distributed recovery in loop networks is analyzed. Distributed recovery depends on the cooperation of a set of nodes (local supervisors) to execute the recovery function. Each node is assumed to have information about only a subset of nodes in its immediate vicinity. Here we restrict our attention to redundant systems in which each node $x$ has information only about the subgraph $N(x)$, called the *neighborhood* of $x$, which is the induced subgraph of $G_r$ containing $x$ and all nodes adjacent to $x$.

Distributed recovery requires several steps in which different nodes successively act as the local supervisor. It typically proceeds as follows:

(a) A fault $F$ occurs changing the system configuration or state from valid to invalid.

(b) An active node $x_i$ detects a faulty node $x_j$ in its neighborhood $N(x_i)$, assumes the role of local supervisor and initiates recovery. Usually $x_i$ attempts to find a spare node $x_k$ in $N(x_i)$ such that $x_k$ can assume the previous state $s_j$ of the faulty node $x_j$.

(c) If no suitable spare is available, either because no spares are present in $N(x_i)$, or because the available spares do not have the proper connectivity, $x_i$ makes an appropriate change to $N(x_i)$ and relinquishes the role of local supervisor,which must then be assumed by some other node.

It should be noted that the initial fault affecting node $x_j$ in Step (a) above corresponds to the absence of a node in state $s_j$. Subsequent reassignment of states during recovery may also lead to other states being temporarily missing from the system configuration. For example, if supervisor node $x_i$ changes the state of another node $x_k$ from $s_k$ to the state $s_j$ of the faulty node $x_j$, the old state $s_k$ of $x_k$ is unassigned These considerations lead to the following concept of an error condition.

**Definition 7.** Let $G_r[C_n]$ be a fault-tolerant system whose current configuration is $G_c$. An *error* $E(s_i)$ exists if $G_c$ has no active node in state $s_i$, where $s_i \in \{1, 2, \ldots, n\}$.

Thus error detection and recovery will be based on the identification of (local) configurations in which one of the $n$ active states is missing.

To allow a node $x_i$ to execute the recovery function when it acts as local supervisor, it must store certain limited information about the system $G_r[C_n]$, specifically:

(a) the structure of $N(x_i)$;

(b) the current state of all nodes in $N(x_i)$;

(c) the set of errors $\{E(s_i)\}$ of interest to $x_i$;

(d) certain backup files defining the tasks of the nodes in $N(x_i)$. When $x_i$ changes the state of another node $x_j$ from $s_j$ to $s_k$, $x_i$ uses these files to supply $x_j$ with the necessary information needed to perform the task $s_k$. $x_i$ periodically updates its backup files by polling the nodes of $N(x_i)$.

Several problems arise in distributed recovery that were not encountered with systems that use centralized recovery. An error may exist in a neighborhood that has no faulty nodes. A neighborhood may have no spare nodes available to replace faulty nodes. Valid local reconfiguration decisions may not lead to a valid global configuration. Storage requirements may be very great if a node is required to store backup files for every potentially faulty node in its neighborhood. If two nodes start reconfiguration at the same time, they may subsequently interfere with each other and prevent recovery from taking place.

To make above problems tractable, we impose the following general restrictions on system behavior.

*Assumptions Governing Distributed Recovery*:

(a) Any active node $x_i$ of $G_r$ in state $s_i$ may act as a local supervisor. It may only detect errors of a specified set $\{E_i\}$ that affect its neighborhood $N(x_i)$. It should be noted that we associate $\{E_i\}$ with the state $s_i$ rather than the node $x_i$.

(b) The set of errors $\{E_i\}$ and $\{E_j\}$ that can be detected by nodes $x_i$ and $x_j$, respectively, are disjoint.

(c) At any time at most one error may be present in the system configuration $G_c$.

(d) When reconfiguring the system in response to an error $E(s_k)$, a local supervisor may either assign $s_k$ to a spare node, or else change its own state to $s_k$.

These assumptions solve the problems listed earlier in the following ways. Assumptions (a)–(c) ensure that at any time only one error is present in the system, and only one node is acting as the local supervisor for this error. Assumption (d) ensures that all reconfiguration actions that do not produce immediate recovery, result in the creation of new errors that conform to Definition 7. This implies that at any time during recovery only one active state is missing from $G_c$. Assumption (c) also implies that only single faults are dealt with. Hence new faults may not occur while the system is recovering from a previous fault. Multiple faults must be treated as a sequence of single faults, with recovery taking place before the occurrence of a new single fault. Ensuring that global recovery is achieved via a series of local reconfiguration steps

requires restrictions to be placed on the system structure.

Next we define a class of efficient $k$-FR realizations of loop networks with respect to a distributed recovery strategy $R_3$, and show that they can be easily characterized by means of power graphs. If an $n$-node graph $G$ has a spanning subgraph $G_s$ isomorphic to the $n$-node cycle $C_n$, then $G$ is called a *Hamiltonian graph*, and $G_s$ is a *Hamiltonian cycle* [3]. Clearly if a redundant realization $G_r$ of $C_n$ is to tolerate a $k$-fault, the graph obtained by eliminating any $k$ nodes from $G_r$ must contain an $n$-node Hamiltonian subgraph. (Graphs with this property have been termed *k-Hamiltonian*.) It is easily shown that $C_{n+k}^{k+1}$ is a $k$-FT realization of $C_n$ which, however, is nonoptimal.

We now introduce a distributed recovery strategy $R_3$ for use with the redundant power graph $C_{n+k}^{k+1}[C_n] = G_r$. For $R_3$ we partition the possible errors into the $n$ disjoint sets $\{E(s_1)\}, \{E(s_2)\}, \ldots, \{E(s_n)\}$, each containing one error type. We also require the node in state $s_i$ to detect only the error $E(s_i - 1)$, where the subtraction is modulo-$n$. Accordingly, $x_i$ periodically polls the neighboring node $x_j$ that is in state $s(x_j) = s_i - 1$. If the error $E(s_i - 1)$ occurs, then $x_i$ changes its own state from $s_i$ to $s_i - 1$, generating the error $E(s_i)$. The recovery process propagates through $G_r$ until the node $x_k$ in state 1 detects the error $E(n)$, at which point $x_k$ activates a spare and the error is absorbed, thereby completing the recovery process. Procedure 5 below describes the recovery strategy $R_3$ formally. In this description it is assumed that the nodes of $G_r$ are labeled by $x_1$ through $x_{n+k}$ according to their positions on the generator cycle $C_{n+k}$.

**Procedure 5:** The distributed recovery strategy $R_3$.

*Step* 1. Let the initial state of $C_{n+k}^{k+1}[C_n]$ be $S_0(x_1, x_2, \ldots, x_{n+k}) = (1, 2, 3, 4, 5, \ldots, n, 0, 0, \ldots, 0)$. Let $S(x_1, \ldots, x_i, \ldots, x_{n+k}) = (s_1, s_2, \ldots, s_i, \ldots, s_{n+k})$ be the current valid state of $C_{n+k}^{k+1}[C_n]$, and let $G_c$ be the configuration corresponding to $S$. Suppose that a fault $F$ affecting a node $x_j$ occurs, which generates an error $E(s_j)$ and changes the state of $C_{n+k}^{k+1}$ from $S$ to $S_F$. If the fault affects a spare node, then $R_3$ leaves the system state unchanged.

*Step* 2. For $1 \leqslant s_i \leqslant n$, the node $x_i$ in state $s_i$ tests the node $x_j \in N(x_i)$ which should be in state $s_i - 1$. If $x_i$ detects the error $E(s_i - 1)$, then one of the following actions occurs:

(a) If the error $E(1)$ is present, then $x_i$, which is in state 2, scans the state vector $S(N(x_i))$ from left to right until a component $s_k = 0$ is found. $x_i$ then changes $s_k$ to 1 and the system recovers. If no $s_k = 0$ exists, then the recovery attempt fails (no spare nodes are available).

(b) If the error $E(s_i - 1)$ is present, where $2 \leqslant s_i - 1 \leqslant n$, then $x_i$ changes its own state from $s_i$ to $s_i - 1$.

(c) If the error $E(n)$ is present, then $x_i$, which is in state 1, scans the state vector $S(N(x_i))$ from left to right until a component $s_k = 0$ is found. $x_i$ then changes $s_k$ to $n$ and the system recovers. If no $s_k = 0$ exists, then the recovery attempt fails (no spare nodes are available).

It should be noted that since at most one error exists in the system at any time, only one node may change state each time $R_3$ is executed. In general, multiple steps are required for the system to recover.

**Theorem 3.** *The fault-tolerant system* $C_{n+k}^{k+1}[C_n]$ *is* $(n-1)k$*-step* $k$-FR *with respect to the recovery strategy* $R_3$.

**Proof.** First we prove that $C_{n+k}^{k+1}[C_n]$ is $k$-FR with respect to the recovery strategy $R_3$. Let the initial state be $S(x_1, x_2, \ldots, x_i, \ldots, x_{n+k}) = (1, 2, \ldots, i, \ldots, 0)$. $R_3$ may require a node $x_i$ in the initial state $s_i$ to change its state up to $k$ times during recovery from a $k$ fault. Thus $x_i$ may pass through the sequence of states $s_i - 1, s_i - 2, \ldots, s_i - k$ in response to a sequence of $k$ faults. A node $x_i$ can act as a local supervisor and assume the state $s_j + 1$. Since $x_i$ is adjacent to every node in the initial state $s_i - 1, s_i - 2, \ldots, s_i - k - 1$, it can act as the local supervisor required by $R_3$ for recovery from the $k$ faults. Hence, $C_{n+k}^{k+1}[C_n]$ is $k$-FR with respect to $R_3$. □

Next, let us calculate the maximum number of steps needed to recover the system from a fault. The worst case occurs when a node $x_j$ in state 2 fails. According to $R_3$, the node $x_k$ in state 3 initiates recovery and changes its own state from 3 to 2 in one step. Then the $n - 3$ nodes in states $s_i$,

for $i = 4, 5, \ldots, n$, have to change their states in turn, each change requiring one step. In the last step, the node in state 1 activates a spare node. Therefore, the maximum number of steps to recover from one fault is $n-1$. For $k$ faults, the number of steps is at most $(n-1)k$.

**Example 4.** Figure 7 shows the graph $C_8^3[C_6]$ which is a 2-FT redundant realization of $C_6$. By Theorem 3, $C_8^3[C_6]$ is also a 10-step 2-FR with respect to $R_3$. Suppose that node $c$ becomes faulty generating the error $E(3)$ and changing the state of the system from $S(a, b, c, d, e, f, g, h) = (1, 2, 3, 4, 5, 6, 0, 0)$ to $S_F = (1, 2, -1, 4, 5, 6, 0, 0)$. The propagation of the local supervisor is

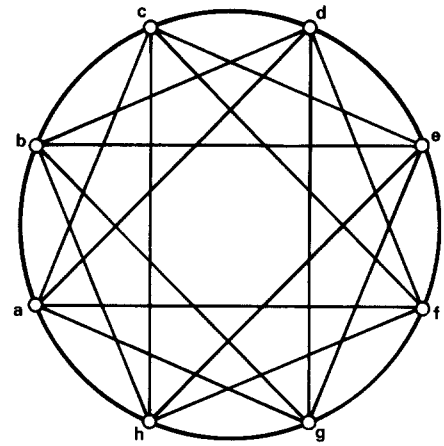



Fig. 7. The 2-FT graph $C_8^3[C_6]$.

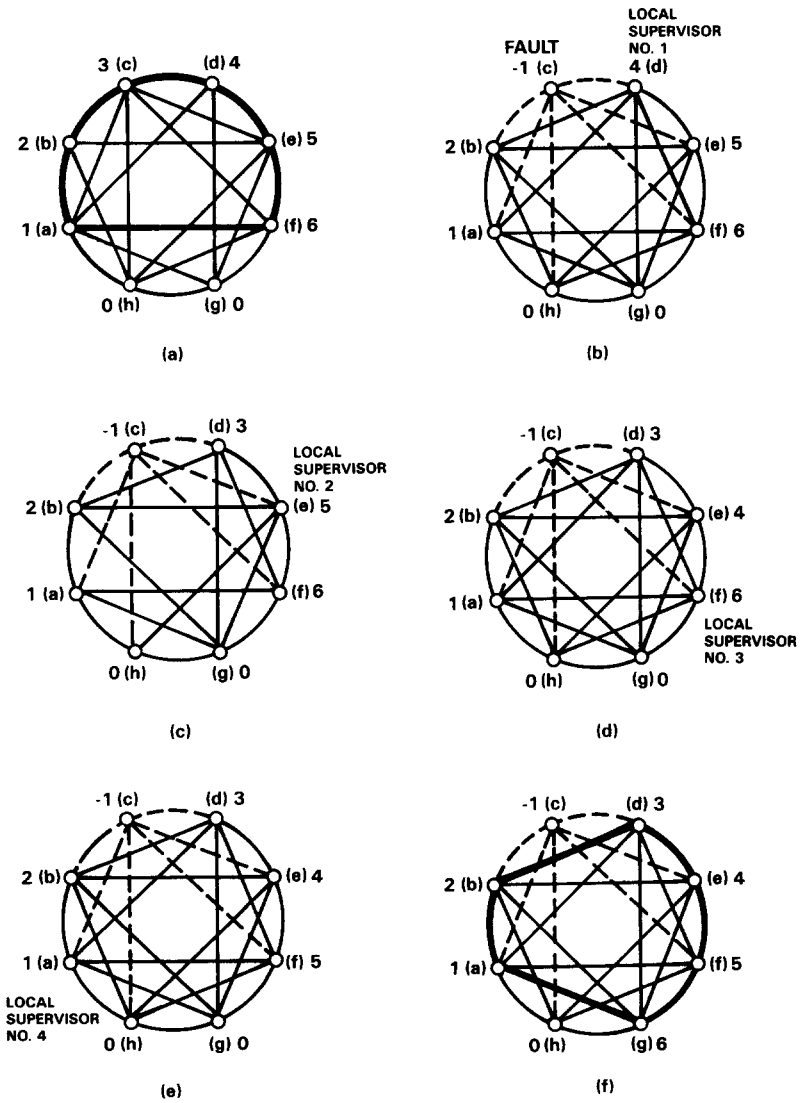



Fig. 8. Recovery propagation between neighborhoods on applying $R_3$ to $C_8^3[C_6]$.

illustrated by the series of configurations in Fig. 8. Figure 8(a) shows the initial configuration $G_c$. Figure 8(b) shows the invalid configuration with the error $E(3)$ present. Node $d$ in state 4 detects this error and becomes the local supervisor. Since $N(d)$ contains no spares, $d$ changes its own state from 4 to 3 generating the new error $E(4)$. The error $E(4)$ is in turn detected by the second local supervisor, node $e$. Node $e$ changes its own state from 5 to 4 producing the error $E(5)$. Node $f$ detects this error and changes its own state from 6 to 5 generating the error $E(6)$. Node $a$ scans $S(N(a))$ and finds that node $g$ is the first available spare in $N(a)$. Node $a$ then changes $S(g)$ from 0 to 6, so that the error is absorbed by the spare node $g$, and the recovery process is complete. The final valid configuration is shown in Fig. 8(f). The corresponding final state of the system is $S = (1, 2, -1, 3, 4, 5, 6, 0)$.

**Example 5.** Figure 9(a) shows a fault-tolerant multi-microcomputer system, the Basic Fault-

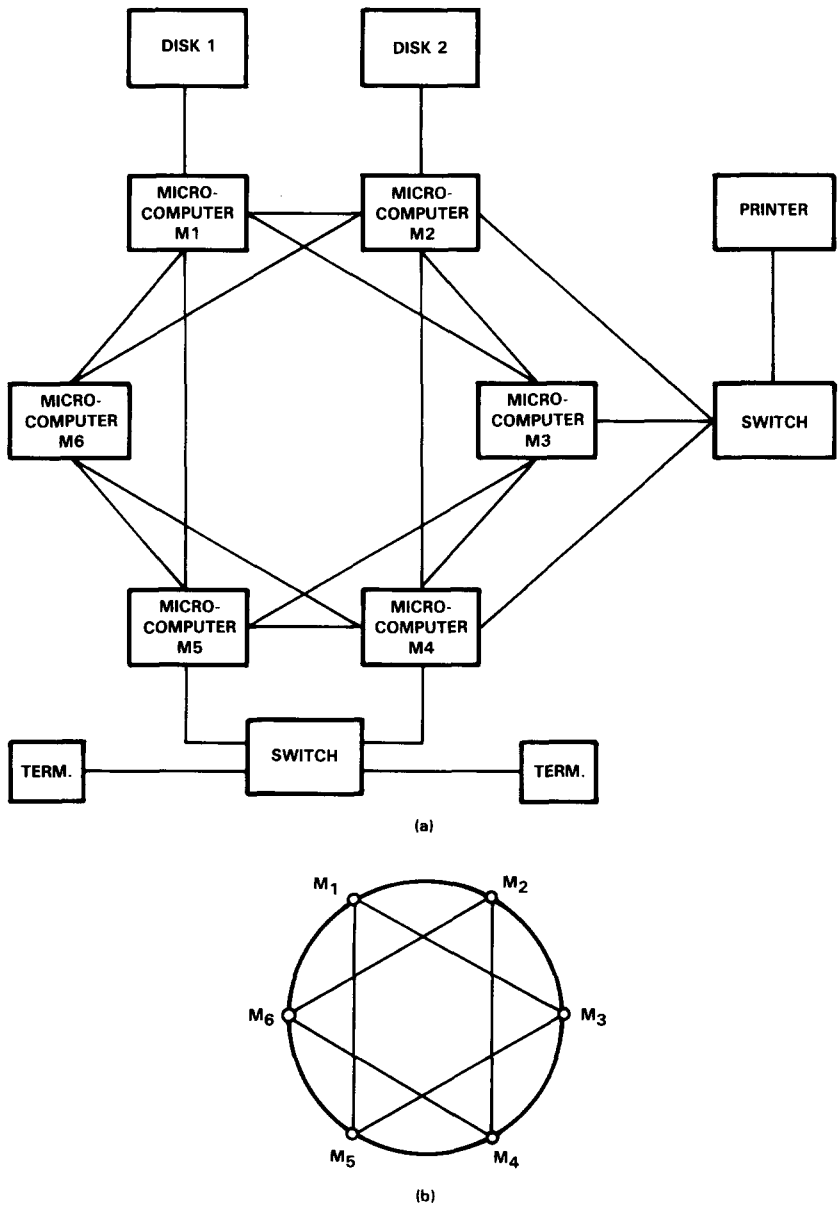



Fig. 9. The Basic Fault-Tolerant Computer (BFS): (a) the 6-computer BFS; (b) its graph representation.

Tolerant System (BFS), which is being developed in West Germany [12]. A suitable facility graph model of BFS is the cycle power graph $C_6^2$, as illustrated in Fig. 9(b). A node $x_i$ in $C_6^2$ corresponding to a microcomputer $M_i$ in BFS. According to Theorem 5, is a 5-step 1-FR realization of a 5-node cyclic system $C_5$ with respect to $R_3$.

Theorem 3 may be extended to systems whose basic graph is the cycle power graph $C_n^m$, rather than the simple cycle $C_n$. Note that the $C^{mn}$ topology is common in computer networks, since nodes tend to be connected directly to nearby nodes.

**Corollary 1.** *The fault-tolerant system* $C_{n+k}^{m+k}[C_n^m]$ *is* $(n-1)k$ *step* $k$-FR *with respect to the recovery strategy* $R_3$.

## 5. Conclusions

A new methodology for characterizing recovery in fault-tolerant distributed-processing loop or cyclic networks has been presented. A graph model representing the basic nonredundant system as well as fault-tolerant versions was proposed, which allows recovery and reconfiguration problems to be described in precise graph-theoretical terms. Several recovery strategies for centralized and distributed recovery were described and their performance was analyzed with respect to fault-tolerant loop networks. An extension of the model to networks that may be represented by the power graph of a cycle was also noted. Applications of this methodology to some other network structures such as trees are discussed in [15].

## References


[1] T. Anderson and P.A. Lee, *Fault Tolerance: Principles and Practice* (Prentice-Hall, London, 1981).

[2] P.H. Enslow, Jr., Multiprocessor Organization—A survey, *Computing Surveys 9* (1977) 103–129.

[3] F. Harary, *Graph Theory* (Addison-Wesley, Reading, MA, 1969).

[4] J.P. Hayes, A Graph Model for Fault-Tolerant Computing Systems, *IEEE Transactions on Computers 25* (1976) 875–884.

[5] J.P. Hayes and R. Yanney, Fault Recovery in Multiprocessor Networks, *Proc. 8th Symposium Fault-Tolerant Computing*, Toulouse, France (1987) 123–128.

[6] K.Y. Liu, Distributed Loop Computer Networks, in: M.C. Yovits et al., ed., *Advances in Computers 17* (Academic Press, New York, 1978) 163–221.

[7] P.M. Merlin, A Study of the Recoverability of Computing Systems, Dissertation, University of California, Irvine, 1974.

[8] R.C. Read and D.G. Corneil, The Graph Isomorphism Disease, *Journal of Graph Theory 1* (1977) 339–363.

[9] F. Saheban and A.D. Friedman, A Survey and Methodology of Reconfigurable Multi-Module Systems, *Proc. COMPSAC 78, Computer Software and Applications Conference*, Chicago, IL (1978) 790–796.

[10] R.R. Schell, Dynamic Reconfiguration in a Modular Computer System, Dissertation, M.I.T., 1971.

[11] D.C. Schmidt and L.E. Druffel, A Fast Backtracking Algorithm to Test Directed Graphs for Isomorphism Using Distance Matrices, *Journal of the ACM 21* (1976) 433–445.

[12] E.J. Schmitter and P. Baues, The Basic Fault-Tolerant System, *IEEE Micro 4* (1984) 66–74.

[13] R. Troy, Dynamic Reconfiguration: An Algorithm and its Efficiency Evaluation, *Proc. 7th Symposium Fault-Tolerant Computing*, Los Angeles (1977) 44–49.

[14] J.J. Wolf et al., Design of a Distributed Fault-Tolerant Network, *Proc. 9th Symposium Fault-Tolerant Computing*, Madison, WI (1979) 17–24.

[15] R.M. Yanney, Fault Recovery in Multiprocessor Networks, Dissertation, University of Southern California, 1982.

[16] R.M. Yanney and J.P. Hayes, Distributed Recovery in Fault-Tolerant Multiprocessor Networks, *IEEE Transactions on Computers 35* (1986) 871–879.