Multi-Level Adaptive Distributed Diagnosis for Fault Detection in a Network of Processors

Ming Shan Su and K. Thulasiraman
(Based on Chapter II of Su's Ph.D. Thesis)

In this chapter we present and discuss a multilevel adaptive distributed diagnosis algorithm for fully connected networks. This is a generalization of Bianchini and Buskens' ADSD algorithm [1].We first present in sections 1 and 2 the essential features of the diagnosis algorithms of Bianchini and Buskens, and Duarte and Nanya [2]. Several concepts and ideas used in these algorithms are also relevant to the design of our new algorithm to be discussed later in the paper. We present the new multilevel diagnosis algorithm and several aspects of this algorithm in section 3 In section 4, we present simulation results comparing these three algorithms.

A preliminary version of the work in this chapter has been reported in [3].

2.1 The ADSD Algorithm

The adaptive-distributed system-level diagnosis algorithm (to be called ADSD) proposed by Bianchini and Buskens [1] assumes the existence of a logical fully connected network and does not permit link failures. It is distributed, adaptive on testing set, and imposes no limit on the number of faulty nodes. It is assumed that there are no links failures. The PMC fault model [4] is used. Also, during the testing process a node cannot fail and recover from that failure during the time between two tests by another node.

The ADSD algorithm is the first practical application of system level diagnosis theory and has been implemented to run on an Ethernet network of over 200 workstations at the Carnegie Mellon University.

Before we discuss the specification of the algorithm, we have to clarify the concepts of "*test*" and "*testing round*" and "*diagnosis latency*" used in distributed system-level diagnosis literature.

A "*test*" could be just simply a node *i* sending a message to node *j* to ask for some information. If the response is proper and on-time, then node *i* evaluates node *j* as fault-free. Otherwise, node *j* is faulty.

The concept of testing round plays a very important role in expressing the diagnosis latency (or capturing the time complexity) of a distributed diagnosis algorithm.

A "*testing round*" is defined as the period of time in which every fault-free node in the system has tested another node as fault-free, and has obtained diagnostic information from that node, or has tested all other nodes as faulty [1]. In other words, the duration of a "*testing round*" includes the time taken by a node *i* to find a fault-free node *j* or evaluate all the nodes as faulty. For example, assume a node *i* at time t_1 starts its sixth test execution, and finds nodes *i*+1, *i*+2 as faulty and *i*+3 as fault-free at time t_4. At this time, node *i* stops testing. On the other hand, node *i*+3 at time t_2 starts its sixth test execution and finds node *i*+4 as fault-free at time $t_{3,}$ and then stops testing. Although the times and the number of tests for node *i* and node *i*+3 to find a fault-free node are different, we still say that nodes *i* and *i*+3 performed their tests in the same testing round, that is, sixth testing round.

The "*Diagnosis latency*" is defined as the time from the detection of a fault event to the time when all the fault-free nodes correctly diagnose the event. In the following our interest is in diagnosis latency after the last fault event has occurred.

Algorithm Specification:

In the ADSD algorithm, a node *i* uses an array called TESTED_UP$_i$ to update the testing results and to respond to the request of its tester. An example of the data structure for node 2 in an eight node system is shown in

TESTED_UP$_2$[0] = 2
TESTED_UP$_2$[1] = *x*
TESTED_UP$_2$[2] = 5
TESTED_UP$_2$[3] = *x*
TESTED_UP$_2$[4] = *x*
TESTED_UP$_2$[5] = 6
TESTED_UP$_2$[6] = 7
TESTED_UP$_2$[7] = 0

Fig.2.1 The TESTED_UP information stored in node 2

Figure 1.

The TESTED_UP$_i$ array contains *N* entries, and the array indices and the values of the entries are node identifiers. For example, entry TESTED_UP$_i$[*u*] = *v*, means that node *i* has received a diagnostic message from a neighbor node (which it has tested as fault-free) indicating node *u* has tested node *v* and found node *v* fault-free. Also an entry of TESTED_UP$_i$[*i*] = *u* means that node *i* itself has tested node *u* as fault-free. If the value of an entry is "*x*", it means that the entry is arbitrary. Figure 1 shows the values kept at node 2 for an eight node network with nodes 1, 3, and 4 faulty.

A special property of this array is that in one testing round after the last fault event has occurred a "*fault-free ring*" will be formed if we start from a fault-free node *i* and connect the fault-free paths from node *i* to other fault-free nodes. Using the above as an example, if we start from node 2, then the fault-free tests are 2 to 5, 5 to 6, 6 to 7, 7 to 0, and 0 to 2. By viewing these fault-free tests as paths and connecting them together, we will have a "*fault-free ring*" (e.g., $2\rightarrow5\rightarrow6\rightarrow7\rightarrow0\rightarrow2$). This property plays an important role in the proof of correctness of the ADSD algorithm as well as the algorithm we shall propose in the following section.

During the process of diagnosis, each node, in a testing round, executes the ADSD algorithm to completion and resumes the testing after a predefined interval. Each node tries to find a fault-free node and uses the information from that fault-free node to update its local diagnosis information in the TESTED_UP array. In at most N testing rounds after the last fault event, each fault-free node will have consistent diagnosis of the fault status of the nodes in the network, thereby resulting in a diagnosis latency of $O(N)$ testing rounds.

Before the execution of the algorithm, all the nodes are ordered sequentially in a list, as $(n_0, n_1, ..., n_{N-1})$. Thus, a node i will test nodes i+1, i+2, ..., etc., sequentially until a fault-free node is found, and then acquires the diagnosis information from that node. Since all the additions are modulo N, we will find that once we connect the testing paths of all the fault-free nodes, the testing paths will form a ring. Therefore, in each testing round, a node i will perform a test in a "*forward*" manner from node i to node i+1 as in Figure 2.2(a), but will get the diagnosis information in a "*backward*" manner from node i+1 to node i as in Figure 2.2(b). It takes one testing round for the diagnostic information, TESTED_UP$_{i+1}$, to be propagated between fault-free nodes i+1 and i. Likewise, it will take two testing rounds for fault-free node i-1 to get TESTED_UP$_{i+1}$ from node i. At the end of at most N testing rounds, all the fault-free nodes will have the same fault status information of all the nodes in the network. Based on the information in TESTED_UP, an algorithm called "*Diagnose*" is used to determine all the fault-free nodes in the network.

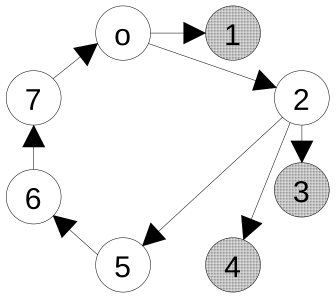


Fig. 2.2(a) An adaptive testing topology for an eight node network with nodes 1, 3, 4 faulty

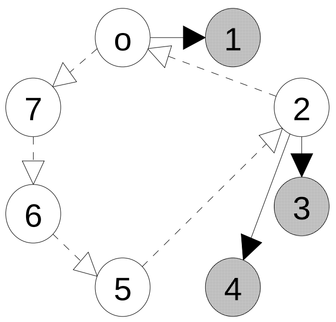


Fig.2.2(b) Information propagation along fault-free nodes

Summarizing, an informal description of the ADSD algorithm and the *Diagnose* algorithm are given Figures 2.3 and 2.4 respectively.

```
/* Diagnose algorithm*/
/* The following is executed at each $n_x$, $0 \le x < N$ when $n_x$
   desires diagnosis of the systems. */
1.  For $i = 0$ to $N - 1$
1.1.      STATE$_x[i]$ = faulty;
2.  node_pointer = $x$;
3.  repeat {
3.1.      STATE$_x$[node_pointer] = fault-free;
3.2.      node_pointer = TESTED_UP$_x$[node_pointer]
3.3. } until (node_pointer == $x$);
```

Fig. 2.4 Diagnose algorithm

ADSD algorithm (informal)

- List the nodes in sequential order, as $(n_0, n_1, \dots, n_{N-1})$.

Testing round for node n_x

1. Node n_x identifies the next sequential fault-free node in the list
   - sequentially **testing** consecutive nodes $n_{x+1 \bmod n}$, $n_{x+2 \bmod n}$, …, etc.,
   - until a fault-free node is found.
2. Diagnostic information received from the tested fault-free node is utilized to **update** local information.

- Repeat steps 1 and 2 in subsequent testing rounds.

Fig. 2.3 ADSD algorithm

2.2 Hierarchical Adaptive Distributed System-Level Diagnosis

The hierarchical adaptive distributed system-level diagnosis (Hi-ADSD) algorithm of Duarte and Nanya [2 uses a divide-and-conquer testing strategy to reduce the diagnosis latency. For testing, Hi-ADSD divides nodes into clusters of various sizes (from small to large), and then collects the diagnosis information in each cluster to accomplish the diagnosis of the system.

Hi-ADSD has a diagnosis latency of at most $O(log^2N)$ testing rounds where N is the number of nodes in the network. It is both adaptive and hierarchical on the testing nodes, distributed on execution, and imposes no limit on the number of faulty nodes. It is also claimed that it has less communication overhead in terms of packet size. Additionally, it has been integrated with the simple network management protocol (SNMP) and applied in a 37-node Ethernet network platform.

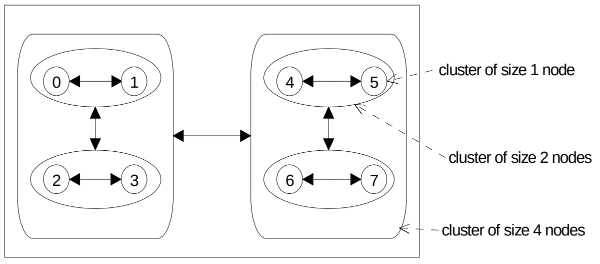
Algorithm Specification:

Consider a network (system) S of N nodes with each node in one of two states, faulty or fault-free. The system is assumed to be a logically complete network. It is assumed that there are no links failures. Again the PMC fault model [4]is used. Also, during the testing process a node cannot fail and recover from that failure during the time between two tests by another node, and that the time could be as long as *logN* testing rounds in the worst case.

Firstly, the Hi-ADSD algorithm at each node divides the remaining nodes into various sizes of clusters for testing, and the size of the cluster tested will vary at different testing rounds. In the following discussion, the network size N and all the cluster sizes are assumed to be a power of 2. In general, a cluster of m nodes will contain nodes u_i, ..., u_{i+m-1} with i MOD m = 0, and m is a power of 2. If m = 1, then the cluster has only one node. If m = 2, then the cluster is the union of two smaller clusters, one containing nodes, u_i, ..., $u_{i+m/2-1}$ and the other containing nodes $u_{i+m/2}$, ..., u_{i+m-1}. As an example, an eight node network with clusters of different sizes is illustrated in Figure 2.5. Duarte and Nanya have developed a formula to identify the clusters with respect to each node. This helps reduce the complexity of the algorithm.

Next, during the execution of the algorithm, in any testing round, each node starts by testing a cluster of size one, then a cluster of size two, and so on, ..., up to a cluster of size of 2^{logN-1} or $N/2$ nodes, and then repeats this testing process. Accordingly, after the continued execution of the Hi-ADSD in at most log^2N testing rounds after the last fault event, each fault-free node will have the same fault status information of the system.

Basically, in one testing round, a node i will sequentially test the nodes in cluster $C_{i,s}$. Function $C_{i,s}$ is used to generate the list of nodes in a cluster as shown below.



A hierarchical approach to test clusters in Hi-ADSD algorithm

Fig. 2.5 An eight node network with clusters of different

$C_{i,s} = \{n_t \mid t = (i \text{ MOD } 2^s + 2^{s-1} + j) \text{ MOD } 2^{s-1+a} + (i \text{ DIV } 2^s) * 2^s + b * 2^{s-1}; j = 0, 1, ..., 2^{s-1}-1\}$,

where

$$a = \begin{cases} 1 & \text{if } i \text{ MOD } 2^S < 2^{S-1} \\ 0 & \text{otherwise.} \end{cases}$$

$$b = \begin{cases} 1 & \text{if } a = 1 \text{ AND } (i \text{ MOD } 2^s + 2^{s-1} + j) \text{ MOD } 2^{s-1+a} + (i \text{ DIV } 2^s) * 2^s < i \\ 0 & \text{otherwise.} \end{cases}$$

For example, the table generated by function $C_{i,s}$ which contains the lists of nodes in various clusters to test for an eight nodes network is shown in Table 2.1.

During execution of the Hi-ADSD, if node i finds a fault-free node j in the cluster, then node i will stop testing and copy the diagnosis information regarding the nodes in $C_{i,s}$ from node j. If node i can not find a fault-free node in $C_{i,s}$, then node i will continue to test the next cluster $C_{i,s+1}$. This testing process at cluster s+1 will stop when node i finds a fault-free node in some cluster or all the nodes in all the clusters are tested as faulty. In the next testing round, node i will start testing the cluster next to the one where it stopped in the previous testing round. However, regardless of the number of clusters that node i has to test in order to find a fault-free node or find the rest of the nodes as faulty, we define the time interval node i spends in doing this as belonging to one testing round.

Table 2.1

Table generated by function $C_{i,s}$ which contains the lists of nodes in various clusters to test for an eight nodes network is shown below.

s	$c_{0,s}$	$c_{1,s}$	$c_{2,s}$	$c_{3,s}$	$c_{4,s}$	$c_{5,s}$	$c_{6,s}$	$c_{7,s}$
1	1	0	3	2	5	4	7	6
2	2, 3	3, 2	0, 1	1, 0	6, 7	7, 6	4, 5	5, 4
3	4, 5, 6, 7	5, 6, 7, 4	6, 7, 4, 5	7, 4, 5, 6	0, 1, 2,3	1, 2, 3, 0	2, 3, 0, 1	3, 0, 1, 2

Summarizing, an informal presentation of the Hi-ADSD algorithm is given Figure 2.6 along with an illustration in Figure 2.7.

```
Hi-ADSD algorithm (informal)
•  Divide nodes into $logN$ clusters of various sizes, (from
   small to large), with cluster index $s = 0, 1, …, logN-1$.
•  Cluster $s$ will contain an ordered list of size $2^s$ nodes.
   Initially, set $s = 0$.

            Each testing round for node $n_x$

1.    Node $n_x$ identifies a fault-free node in cluster $s$
        •  sequentially testing list nodes in cluster $s$
        •  until a fault-free node is found or all the nodes
           in cluster $s$ are faulty.
2.    If a tested fault-free node is found, then
        •  copy diagnostic information of all nodes in
           cluster $s$ from this fault-free node.
        •  Set $s = (s + 1)$ modulo $logN$
      Else goto Step 1 with $s = (s + 1)$ modulo $logN$
```

Fig. 2.6 Hi-ADSD algorithm

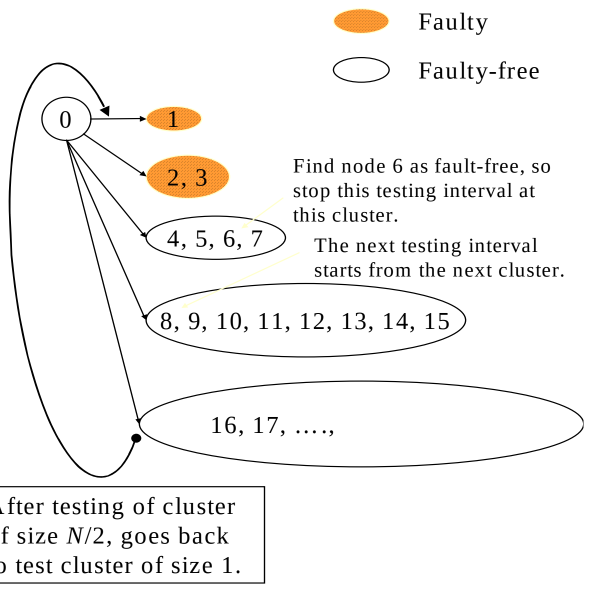

Faulty
Faulty-free
0
1
2, 3
4, 5, 6, 7
8, 9, 10, 11, 12, 13, 14, 15
16, 17, ….,
Find node 6 as fault-free, so stop this testing interval at this cluster.
The next testing interval starts from the next cluster.
After testing of cluster of size N/2, goes back to test cluster of size 1.

Fig. 2.7 Illustration of Hi-ADSD algorithm

We conclude this section by pointing out that the ADSD algorithm can be viewed as a special case of the Hi-ADSD algorithm.

In the Hi-ADSD algorithm each node i views the remaining nodes as partitioned into $log\ N$-1 clusters of varying sizes. In the first testing round it starts testing a cluster of size one, then a cluster of size 2, and so on until it finds a cluster with a fault free node. In the subsequent testing round, node i starts testing with the cluster next to the one where it stopped in the previous testing round.

On the other hand, node i in the ADSD algorithm views the remaining nodes as one single cluster of N-1 nodes: i+1, i+2,…, i-1. So, in every testing round node i tests its only cluster starting each time testing node i+1.

Thus the ADSD algorithm can be viewed as a 1-level hierarchical algorithm whereas the Hi-ADSD algorithm is a $log\ N$-1 level hierarchical algorithm.

2.3 ML-ADSD: Multi-Level Adaptive Distributed System-Level Diagnosis

In this section we propose the design of a multi-level adaptive distributed diagnosis algorithm for fully connected networks, which generalizes the ADSD algorithm of Bianchini and Buskens. This algorithm, to be called the **ML-ADSD algorithm**, has been motivated by the need to reduce the diagnosis latency of the ADSD algorithm as well as the message transmission overhead. The main features of the ML-ADSD algorithm are: multi-level divide-and-conquer partition strategy; adaptive on the next testing assignment; distributed on execution; no upper bound on the number of faulty nodes; autonomous leader election; easy control on synchronization; and less message transmission overhead.

For the sake of simplicity in explanation, the following discussion of the ML-ADSD algorithm assumes a two-level scheme. The details of the algorithm for more than two levels will be presented in the following section.

2.3.1 Level-1 Clusters

Again, we assume a logically complete network G of N nodes, n_0, n_1, ..., n_{N-1}. The nodes are first partitioned into p clusters of equal size. Thus each cluster has N/p nodes. For reasons which will become clear later we shall call these clusters as **level-1 clusters**. To make the discussion and the algorithm simpler to present, we assume that both N and p are powers of two. Also it is assumed that each node is able to correctly test and determine the state of other nodes based on the PMC fault model. Link failures are not permitted. An example of a Bus/Ethernet Network of eight nodes and its logical fully connected network is shown in Figures 2.8(a) and (b). Figures 2.9(a) and (b) also shows the re-mapping of old node ids to new node

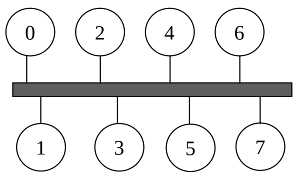


Fig. 2.8(a) An eight node Ethernet network

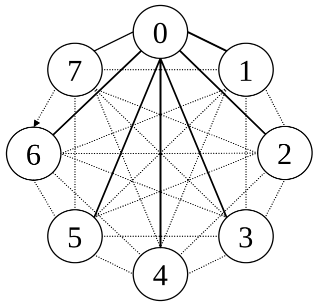


Fig. 2.8(b) A logical fully connected network

ids and the partition of a network of eight nodes into four

k	0	1	2	3	4	5	6	7
i	0	1	0	1	0	1	0	1
j	0	0	1	1	2	2	3	3

n_k = old node id
n_k => $n_{i,j}$, i: new node id,
j: cluster id
$i = k$ MOD (N/p),
$j = k$ DIV (N/p)
e.g., n_2 => $n_{0,1}$, n_5 => $n_{1,2}$ where N = 8, p = 4.

Fig. 2.9(a) Node IDs re-mapping table

clusters.

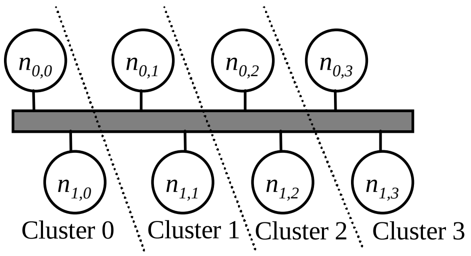


Fig. 2.9(b) Re-map and partition of the nodes into four clusters

2.3.2 Informal Description of the Two-Level Diagnosis Algorithm

Consider Figure 2.10 which gives a pictorial view of our two-level algorithm. In this tree description of the algorithm we have the p original clusters at level 1. Whenever possible, we associate a leader node with each cluster. The fault free node with the smallest id in a cluster is selected as the **leader** of that cluster. If a node is not a leader node, then it is called a **regular node**. Note that if all the nodes in a level-1 cluster are faulty, then this cluster has no leader. At level 2 there is a single **level-2 cluster** consisting of the leaders from the different level-1 clusters. Thus the size of the level-2 cluster is at most p nodes.

Initially, all nodes are regular nodes. Each node begins the diagnosis process by testing, in its first testing round, only the nodes in its cluster. This testing round is called a **level-1 testing round**. The actions taken during this testing round are almost the same as in the testing round of the ADSD algorithm. Specifically, during a level-1 testing round each fault free node identifies a unique fault free node, if it exists, and updates the local diagnosis information. Moreover, in at most N/p testing rounds after

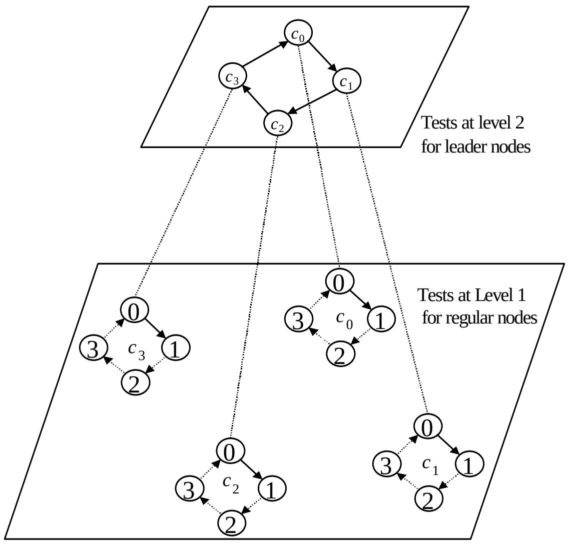

c_0
c_1
c_2
c_3
Tests at level 2 for leader nodes
Tests at Level 1 for regular nodes
0
1
2
3

Fig. 2.10 A two-level scheme

the occurrence of the last fault event these fault free nodes will form a cycle and a node will be able to detect all the nodes in this cycle. A subroutine "***Cluster-leader***" (in Figure 2.12) to be discussed later is used to detect this cycle. If a node finds itself to have the smallest id among the ids of all the nodes in this cycle, it elects itself as leader of its cluster. After completing the testing of nodes in a testing round, the node changes its status to "leader", if necessary, and ends the testing round.

On the other hand, a leader node executes a **level-2 testing round** to be defined in the following and also updates its status to "regular", if necessary.

Note that in a level-1 testing round a node $n_{i,j}$ will try to find a fault-free node sequentially in cluster j and uses the diagnosis information from that fault-free node to update its local diagnosis information. On the other hand, in a level-2 testing round a leader node $n_{i,j}$ will try to find a fault-free node starting from $n_{0,j+1}$ in cluster j+1 (modulo p). The leader will then use the diagnosis information from that fault-free leader node in cluster j+1 to update the status of the nodes in clusters other than its own. If all the nodes in cluster j+1 are found faulty, the leader node in cluster j will continue to look for a fault-free node starting from $n_{0,j+2}$ in cluster j+2, ..., and so on, if necessary. As in the case of level-1 testing rounds, in each level-2 testing round each leader identifies a unique leader from another cluster and these leaders will form a cycle. But a leader will be able to detect the nodes in this cycle only after performing at most p level-2 testing rounds. Also, during a level-2 testing round, in some cases, a leader node may be required to do a level-1 testing if it continues to be the fault free node with the smallest id in its cluster and take appropriate actions, if necessary.

We next present the details of the data structures and algorithms used in implementing the above diagnosis scheme.

2.3.3 Data Structures

In the ML-ADSD algorithm, a node $n_{i,j}$ uses an array called TESTED_UP$_{i,j}$ to update the testing results. TESTED_UP$_{i,j}$ is a two-dimensional array of size (N/p x p) where N is the number of nodes of the network and p is the number of clusters defined by the user. Also the row index i represents the re-mapped node id and the column index j represents the cluster id. Additionally, the values in the array are the re-mapped node identifiers. In addition, entry TESTED_UP$_{i,j}$[u][k] = v at node $n_{i,j}$ means that node i in cluster j has received a diagnosis message from a neighbor node (which it has tested as fault-free) indicating node u in cluster k has tested node v in cluster k and found node v as fault-free Also, an entry TESTED_UP$_{i,j}$[i][j] = u means that node i itself has tested node u and found node u as fault-free in cluster j. If the value of an entry is "x", it means that the entry is arbitrary. Figure 2.11 shows that the values kept at node $n_{0,1}$ for an eight node network of four clusters with nodes $n_{1,1}$, and $n_{0,2}$ (where the old node ids are n_3 and n_4) faulty.

TESTED_UP$_{0,1}$	[0][0] = 1
TESTED_UP$_{0,1}$	[1][0] = 0
TESTED_UP$_{0,1}$	[0][1] = 0
TESTED_UP$_{0,1}$	[1][1] = x
TESTED_UP$_{0,1}$	[0][2] = x
TESTED_UP$_{0,1}$	[1][2] = 1
TESTED_UP$_{0,1}$	[0][3] = 1
TESTED_UP$_{0,1}$	[1][3] = 0

Fig. 2.11Values kept at node $n_{0,1}$

2.3.4 Cluster Leader Election

The subroutine "*Cluster_leader*" in Figure 2.12 identifies the smallest node id along a fault-free ring (cycle) in a cluster as the leader in that cluster. Recall that one of the special properties of the one-dimensional TESTED_UP$_i$ array used in the ADSD algorithm is that a fault-free ring will be formed in at most N testing rounds after the last fault event if there exists at least one fault-free node in the network. A fault-free ring can be formed at a node i by connecting all the fault-free tests in the TESTED_UP$_i$ array starting from the value in TESTED_UP$_i$[i]. To find out a leader, a node i just goes along the members of the fault-free ring and compares its own node id with other members' node ids. If node i has the smallest node id among the members in the ring, then node i is the leader in the fault-free ring. Otherwise, node i is not the leader. If a node is not in a ring, then it is not a leader node.

Similarly, we will apply the same fault-free ring concept in the two-dimensional TESTED_UP$_{i,j}$ array in this section by viewing each cluster's values in TESTED_UP$_{i,j}$ as one smaller one-dimensional TESTED_UP$_i$. To find out the cluster leadership in cluster j, a node $n_{i,j}$ can start from the node id value in TESTED_UP$_{i,j}$[i][j] and go along the fault-free ring to compare the rest of node ids in TESTED_UP$_{i,j}$. The pseudo code for the routine "*Cluster_leader*" is shown in Figure 2.12.

```
/* Cluster_leader election subroutine at node $n_{i,j}$ */

1.  MARK $n_{i,j}$ = True;        //Mark node $n_{i,j}$ visited
2.  next_node = TESTED_UP$_{i,j}$[i][j];        //get the next tested node id

3.  While (next_node ≠ 'x') do
4.      If (has_marked(next_node) == True)
5.          If (next_node == i)
6.              stop, i is the leader;    //cycle back to myself
7.          Else            // cycle detected but i is not in the cycle
8.                  stop, i is not the leader;
9.      Else
10.         If (next_node < i)
11.             stop, i is not the leader;   //someone's id smaller than mine
12.         Else
13.           MARK next_node = True;
14.           next_node = TESTED_UP$_{i,j}$[next_node][j];
15. End_While

16. Stop;  // i is not leader or the fault-free ring has not formed yet
```

Fig. 2.12 Cluster leader election subroutine

2.3.5 Cycle Detection at Level 2

To detect the cycle at level 2 we use a data structure similar to the TESTED_UP array. With each node $n_{i,j}$, we associate a one-dimensional leader array LEADER$_{i,j}$.

The LEADER$_{i,j}$ array contains p entries. The array indices represent the cluster ids and the values of the entries are node identifiers. For example, entry LEADER$_{i,j}[u]$ = $n_{x,y}$, means that node $n_{i,j}$ has received an information from a neighbor node (which it has tested as fault-free) indicating a node in cluster *u* has tested node $n_{x,y}$, in cluster *y* and found node it fault-free. Also an entry of LEADER$_{i,j}[j]$ = $n_{x,y}$ means that the leader node itself in cluster *j* has tested node $n_{x,y}$, in cluster *y* as fault-free. If the value of an entry is "*x*", it means that the entry is arbitrary.

As in the case of cluster leader election, the LEADER$_{i,j}$ array can be used to determine if node $n_{i,j,}$ is in a cycle at level 2. The subroutine for cycle detection is given in Figure 2.13. Once a node detects that it is in a cycle its status is changed to “regular”.

If a faulty node at level 2 recovers it may detect a fault free cycle at level 2, but may not find itself in the cycle. In this case also we change the status of the node to “regular”. Certain actions need to be taken if the node does not detect a cycle. See the algorithm description given below.

Now we are ready to present our two-level adaptive distributed diagnosis algorithm.

```
/* Cycle Detection subroutine at node $n_{i,j}$*/

1.  MARK j = True;   //Mark cluster id j Visited
2.  $n_{a,b}$ = LEADER$_{i,j}$[j];     //get the next tested
    node id

3.  While $n_{a,b} \neq$ ‘x’ do
4.     If ( has_marked(b) == True )
          // Has cluster b been visited?
5.           If ( a == i )     //cycle back to myself
6.                 Stop, $n_{i,j}$ is in a cycle;
7.           Else
8.                 Stop, $n_{i,j}$ detects a cycle but
                   it is not in the cycle;
9.     Else
10.          MARK b = True;        // Mark cluster
              id b visited
11.          $n_{a,b}$ = LEADER$_{i,j}$[b];
12. End_While

13. Stop;  // $n_{i,j}$ does no detect a cycle
```

Fig. 2.13 Cycle detection subroutine

2.3.6 Two-Level Algorithm Description

2-Level ADSD Algorithm

Initially, all the nodes are regular nodes.

During a testing round each node *v* performs the following:

CASE 1: If *v* is a regular node, then execute the “Level-1 Testing Algorithm” of Figure 2.14, which includes leader election.

- Change the status of *v* to “leader”, if it becomes a leader.
- End testing round.

CASE 2: If *v* is a leader, then execute the “Level-2 Testing Algorithm” of Figure 2.15.

- Execute the Cycle detection algorithm of Figure 2.13.
- If *v* detects a cycle at level 2, then
  - o change the status of *v* to "regular".
  - o End testing round.
- If *v* does not detect a cycle, then
  - o if *v* is not the fault free node with smallest id in its cluster at level-1, then change the status of *v* to “regular”.
  - o Otherwise, execute the “Level-1 Testing Algorithm.”
  - o End testing round.

```
/* Regular node, Level-1 Testing Algorithm at node
   $n_{x,a}$ */

1.  y = x            //assign my node id
2.  repeat {
3.       y = (y + 1) mod N/p
4.       request $n_{y,a}$ to forward TESTED_UP$_{y,a}$ to $n_{x,a}$
5.  } until ($n_{x,a}$ tests $n_{y,a}$ as “fault-free”)

6.  for node = 0 to (N/p – 1)              // update
    local cluster diagnosis information
7.       TESTED_UP$_{x,a}$[node][a]                  =
    TESTED_UP$_{y,a}$[node][a]

8.  TESTED_UP$_{x,a}$[x][a] = y                 // x itself
    tests y as fault free


/* Level-1 Cluter Leader Election or Update */
9.  If ( Cluter_leader( ) == “leader”)       // check
    the leadership
10.      status(x) = leader
11. Else// update diagnosis information regarding
    other clusters
12.      for cluster = 0 to (p – 1)
13.         for node = 0 to (N/p – 1)
14.              if (cluster $\neq$ a)
15.                   TESTED_UP$_{x,a}$[node][cluster]      =
    TESTED_UP$_{y,a}$[node][cluster]

16. Stop.           // end of Level-1 testing round
```

Fig. 2.14 Level-1 testing algorithm

The *Diagnose* algorithm is very similar to that used in the ADSD algorithm. As in the diagnosis algorithm in Figure 2.4, the STATE information is completed at each node $n_{i,j}$ by using the node identifiers in TESTED_UP$_{i,j}$ and LEADER$_{i,j}$ arrays.

/* Leader node, Level-2 Testing Algorithm at node $n_{x,a}$*/

```
1.   j = a;   /* assign my cluster id*/
2.   repeat {
3.        j = (j + 1) mod p;
4.        u = N/p - 1;  // always start to test node id 0 in a new cluster
5.        repeat {
6.             u = (u + 1) mod N/p;
7.             request n_{u,j} to forward TESTED_UP_{u,j} and LEADER_{u,j} to n_{x,a};
8.        } until (n_{x,a} tests n_{u,j} as "fault-free") or ("all the nodes in cluster j are faulty");
9.   } until (n_{x,a} tests n_{u,j} as "fault-free")

10.  for cluster = 0 to (p – 1)      // update info. regarding other clusters
11.       for node = 0 to (N/p – 1)
12.            if (cluster ≠ a)
13.                 TESTED_UP_{x,a}[node][cluster]= TESTED_UP_{u,j}[node][cluster]

14.  Stop.       // end of Level-2 testing round
```

Fig. 2.15 Level-2 testing algorithm

2.3.7 Proof of Correctness and Diagnosis Latency

Note that our ML-ADSD algorithm is a generalization of the ADSD algorithm of Bianchini and Buskens [BB92]. We can view the ADSD algorithm as a level-1 algorithm. Since all the nodes are at level 1 in the ADSD algorithm, in at most N testing rounds after the last fault event every fault free node will have correct and consistent fault status information of all the nodes in the network. But in the case of the multilevel algorithm some of the faulty nodes could be at level 2 and may recover to become fault free nodes while being at level 2. Until they return to level 1, the information they contain is not reliable and may not be correct. Also, they may not perform level-1 testing, thereby preventing the election of the leaders at level 1. So, in proving the correctness of the two-level algorithm we need to ensure that a faulty node after recovery does not prevent the election of leader at level 1 and also eventually returns to level 1. The actions taken by the ML-ADSD algorithm achieve this.

Consider again the two-level scheme show in Figure 2.10. Without loss of generality, we assume that initially all nodes are fault free and are regular nodes. Our proof of correctness involves establishing that after executing a certain number of testing rounds after the occurrence of the last fault event, all the nodes will have the correct view of the fault status of all the nodes in the system.

Note that a testing round for a node includes the time taken by the node to perform the actions specified by the 2-level ADSD algorithm.

The algorithm may be viewed as consisting of three phases.

Phase 1: In this phase all the nodes identify their respective leaders. The nodes in each level-1 cluster acquire correct view of the fault status of all the nodes in that cluster. In order to elect the leader each node must identify, using the data in the TESTED_UP array, the cycle of fault free processors at level 1. To do so, each node must execute certain number of level-1 testing rounds after the last fault event.

We need to consider several cases.

Case 1: No faults occur.

In this case, all nodes are regular nodes and as in the ADSD algorithm, in at most ***N/p* level-1 testing rounds** after the last fault event, all nodes in each level-1 cluster will get consistent and correct fault status information of all nodes in that cluster, and the leader of each cluster will be selected.

In the following, cluster refers to a level-1 cluster.

Case 2: A faulty node can recover only when it is at level 1.

Without loss of generality, let us assume that the last fault event is the recovery of a faulty node, say node *k*, in cluster 1. Let C_1, the cycle of fault free processors in cluster 1 just before the last fault event be as in Figure 2.16(a). Let *i* be the fault free node with the smallest id in C_1. In the worst case node *k* may also be in a cycle C_2 of faulty processors before its recovery. Cycle C_2 is shown in Figure 2.16(b). Since node *k* recovers in the last faulty event, the cycle of fault free processors after the recovery of node *k* will be as in Figure 2.16(c). This cycle needs to be identified by all the fault free nodes in cluster 1.

- Let us first assume that node *k* is less than *i*.

In the first testing round after the recovery of node *k*, node *i* will execute a level-1 testing (Case 1 in the 2-level ADSD algorithm) and identify node i_1 as the next node in the cycle of fault free processors. Node *i* being the leader of cluster 1 before the recovery, the TESTED_UP$_i$ will indicate that node *i* is a leader and so it will change its status to "leader". This is because, in the TESTED_UP$_i$

array of node *i*, node *k* will not be identified as a fault free processor until after a certain number of testing rounds. Other nodes including node *k* also will execute one level-1 testing in this first round, but will not identify themselves as leaders.

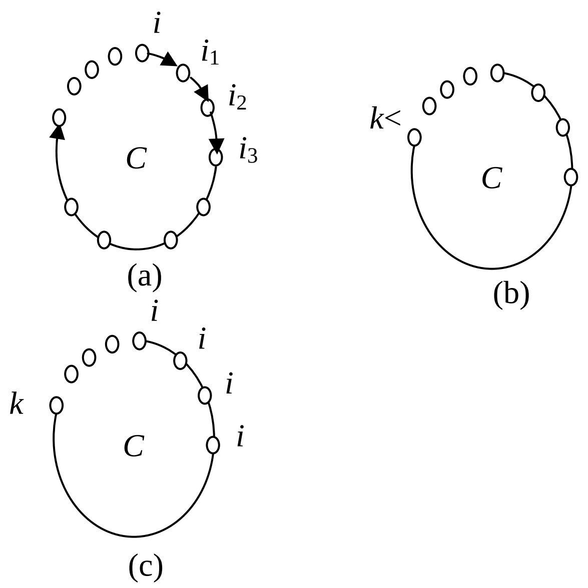


Fig. 2.16 Illustration of proof of correctness

In the second testing round, being a leader node, node *i* will execute a level-2 testing (Case 2 in the 2-level ADSD algorithm). Node *i* will not find itself to be the fault free node with the smallest id in cluster. So, it will set its status to "regular". This completes the actions taken by node *i* in the second testing round after the recovery of node *k.* Other nodes in cluster 1 will execute one level-1 testing and remain as regular nodes. These nodes also will identify the next two nodes in the fault free cycle $C_{3.}$ In particular, node i_1 will identify i_2 and i_3 as the next two nodes in C_3.

In the third testing round, node *i* will execute one level-1 testing and identify itself as leader. After testing i_1 fault free and updating its TESTED_UP array, it will also identify nodes i_1, i_2, and node i_3 as the next three nodes in the fault free cycle C_3.

Thus in an even testing round node *i* will execute one level-2 testing and in an odd testing round it will execute one level-1 testing. This will continue until node *i* identifies node *k* as the smallest fault free node in C_3 and node *i* recognizes that it is no longer the leader of its cluster. On the other hand, all other nodes will execute only level-1 testings.

Thus, if node *k* is *x*th one after node *i* in the fault free cycle $C_{3,}$ then these *x* nodes will be identified by node *i* in *x* or *x*+1 testing rounds after the recovery of node *k*, and the cycle C_3 of fault free processors will be identified by all the nodes in at most ***N/p* or *N/p*+1 testing rounds** after the last fault event. Also, at the end of these testing rounds, node *k* will be identified as the leader of cluster 1.

The above reasoning is applicable even if more than one node recovers in the last fault event.

- Next consider the situation when *k* is greater than *i*.

In this case, in the first testing round after the last event, node *i* will execute a level-1 testing round and may find itself as leader. In the second and subsequent testing rounds, being a leader and the fault free node with the smallest id, node *i* will execute a level-1 testing as well as a level-2 testing and will remain at level 2. On the other hand, all other nodes in cluster 1 will execute only level-1 testings. Thus, in at most ***N/p* testing rounds** after the last event, node *i* and all other nodes in cluster 1 will identify node *i* as the leader of cluster 1.

Case 3: Faulty nodes at level 2 may recover

Let the last fault event be the recovery of node *k* at level 2.

- Let us first consider the situation when node *k* is not the fault free node with the smallest id in cluster 1 after the last fault event.

In the first testing round after its recovery, node *k* will execute a level-2 testing round and will return to "regular" node status. In the second testing round, it will execute one level-1 testing and may find itself as a leader because, possibly, it was in a cycle of faulty nodes before the last fault event. In the third testing round it will execute one level-2 testing and again return to "regular" node status. In the fourth testing round it will execute one level-1 testing and may again find itself to be a leader. This sequence of alternation between "regular" and "leader" status will continue until at most ***N/p* testing rounds** are executed. At the end of these testing rounds, node *k* will identify the cycle $C_{3.}$ This is also true of all other nodes in the cycle $C_{3.}$

- Let us next consider the situation when node *k* is the fault free node with the smallest id in cluster 1 after it recovers in the last fault event.

In this case, in each testing round after the last fault event, the node *k* will execute both a level-1 testing and a level-2 testing and will remain at level 2. All other nodes will execute level-1 testings in these testing rounds. So, in at most ***N/p* testing rounds** after the last event, every node in cluster 1 will identify node *k* as the leader.

Summarizing, in **at most *N/p* or *N/p* +1** testing rounds after the last fault event all nodes in all clusters will identify the leader nodes and acquire correct view of the status of all the nodes in their respective clusters.

Phase 2: During this phase, in **at most *p* testing rounds** the fault free leaders of the clusters identify the ring of leader nodes at level 2 using the LEADER array, update their TESTED_UP arrays with the status information of all the nodes in clusters other than their own.

Phase 3: During this phase consisting of **at most *N/p* Level-1 testing rounds**, the information at the fault-free leader nodes will be propagated to the remaining nodes in their respective clusters.

Thus at the end of at most **2(*N/p*) + *p* + 1** testing rounds after the last fault event all the fault free nodes in the network will have consistent and correct status information of all the nodes in the network. Thus we have the following theorem.

We now draw attention to certain issues we encountered while developing the two-level algorithm.

Suppose no faults occur (Case 1 in Phase 1 discussed above). The following simplified version of the 2-level ADSD algorithm will be adequate for all nodes to acquire a correct view of the status of all the nodes in the network.

Simplified 2-Level ADSD Algorithm

Initially, all the nodes are regular nodes.

During a testing round each node *v* performs the following.

CASE 1: If *v* is a regular node, then execute the "Level-1 testing algorithm" of Figure 2.14, which includes leader election.

- Change the status of *v* to "leader", if it becomes a leader.
- End testing round.

CASE 2: If *v* is a leader, then execute the "Level-2 testing algorithm" of Figure 2.15.

- Execute the "Cycle detection algorithm" of Figure 2.13.
- If *v* detects a cycle at level 2, then change the status of *v* to "regular".
- End testing round.

Consider Case 2 in Phase 1. Assume *k* is less than *i* (as in Figure 2.17). In this situation, if the above simplified algorithm is used, then in the first testing round node *i* will identify itself as leader of cluster 1 and move to level 2. In the second and subsequent testing rounds it will execute only level-2 testing. It will also identify the leader node *q* of cluster 2 and update its LEADER array appropriately. Since it does not execute any level-1 testings, the nodes in cluster 1 will never be able to identify their leader node *k*. But the node *t* will become a leader of cluster 4 at some point and will identify node *k* as the leader of cluster 1 and update its LEADER accordingly. Since node *k* has not been able to identify itself as leader of cluster 1, none of the leader nodes will identify the cycle of leaders at level 2. Thus, a deadlock situation results (as in Figure 2.17) and the nodes execute their respective actions without ever being able to acquire the correct view of the status of the nodes in the network.

Similar deadlock situations would occur in case 3 too. The additional actions taken in Case 2 of the 2-level ADSD algorithm ensure that such deadlock situations do not occur.

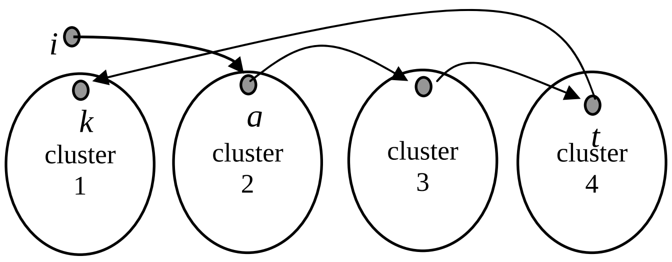


Fig. 2.17 Illustration of deadlock

2.3.8 ML-ADSD Algorithm for the General Case

We now present the details of the Multi-level Adaptive Distributed Diagnosis Scheme (ML-ADSD) for the general case of level greater than 2. A pictorial tree description of this algorithm is shown in Figure 2.18(a). To simplify the discussion and without loss of generality, we assume that each cluster at level 1 (the original *p* clusters) has at least one fault free node. We denote the last level by *M*.

Algorithm Specification

- **Level-*i* Clusters and Leaders**

At level 1 of the tree representation are the *p* original clusters, to be called the **level-1 clusters**. At level 2 there are *p*/2 clusters, to be called **level–2 clusters**. Each level-2 cluster consists of at most 2 nodes which are leaders of two level-1 clusters. In general, at level *i* there are $p/2^{i-1}$ clusters, each containing at most two nodes which are leaders of two level-(*i*-1) clusters. In view of our assumption that each cluster has at least one fault free node, all clusters at levels greater than one contain exactly two nodes. Note that, as before, in each cluster the node with the smallest id will be called the leader of that cluster. The leader will be called the left node and the other node will be called the right node of that cluster. Also, if $M < \log p + 1$, then at level *M* there will be one cluster of $p/2^{M-2}$ nodes, one for each cluster at level $M - 1$.

Each cluster at level $i < M$ may be viewed as the root of the subtree with 2^{i-1} level-1 clusters as leaves. Specifically, consider the *j*th cluster (counted from left) in level *i*. This cluster may be viewed as representing the level-1 clusters numbered from $j2^{i-1}$ to $(j+1)2^{i-1}$-1. Each one of the two nodes in this cluster will then represent

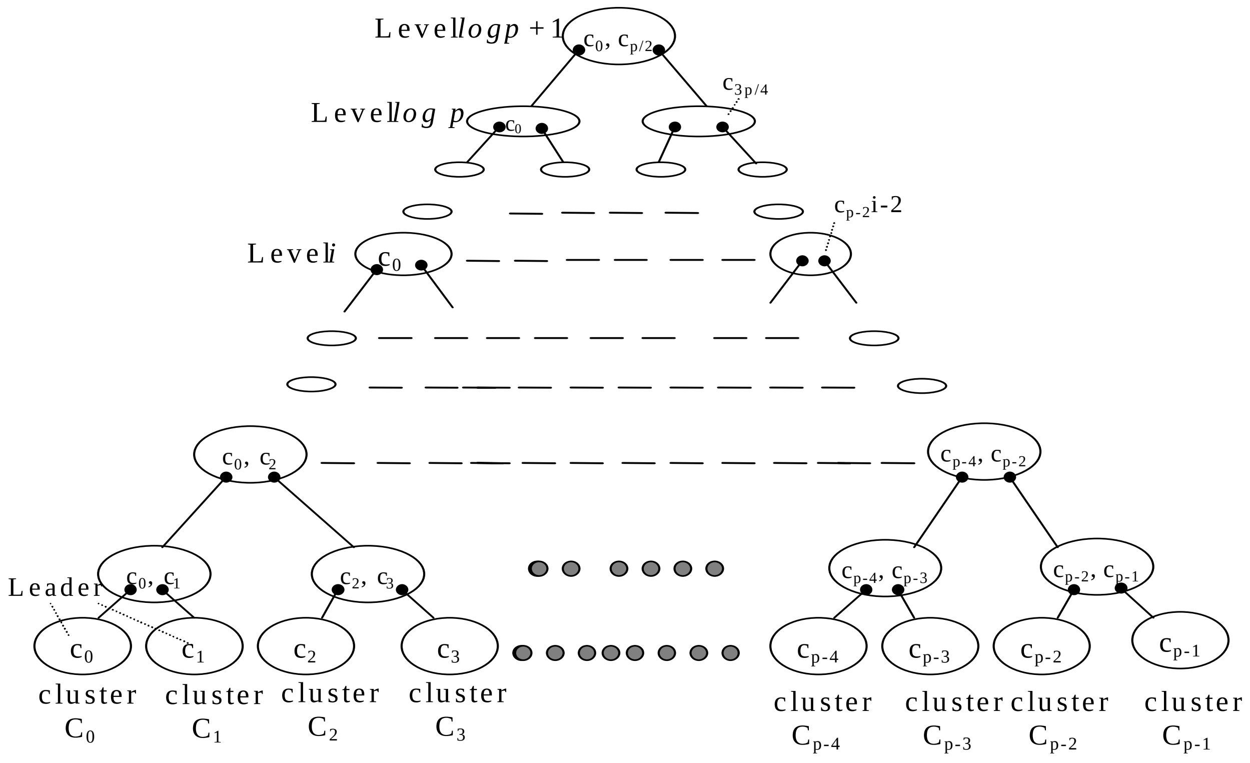


Fig. 2.18(a).Tree representation of ML-ADSD algorithm

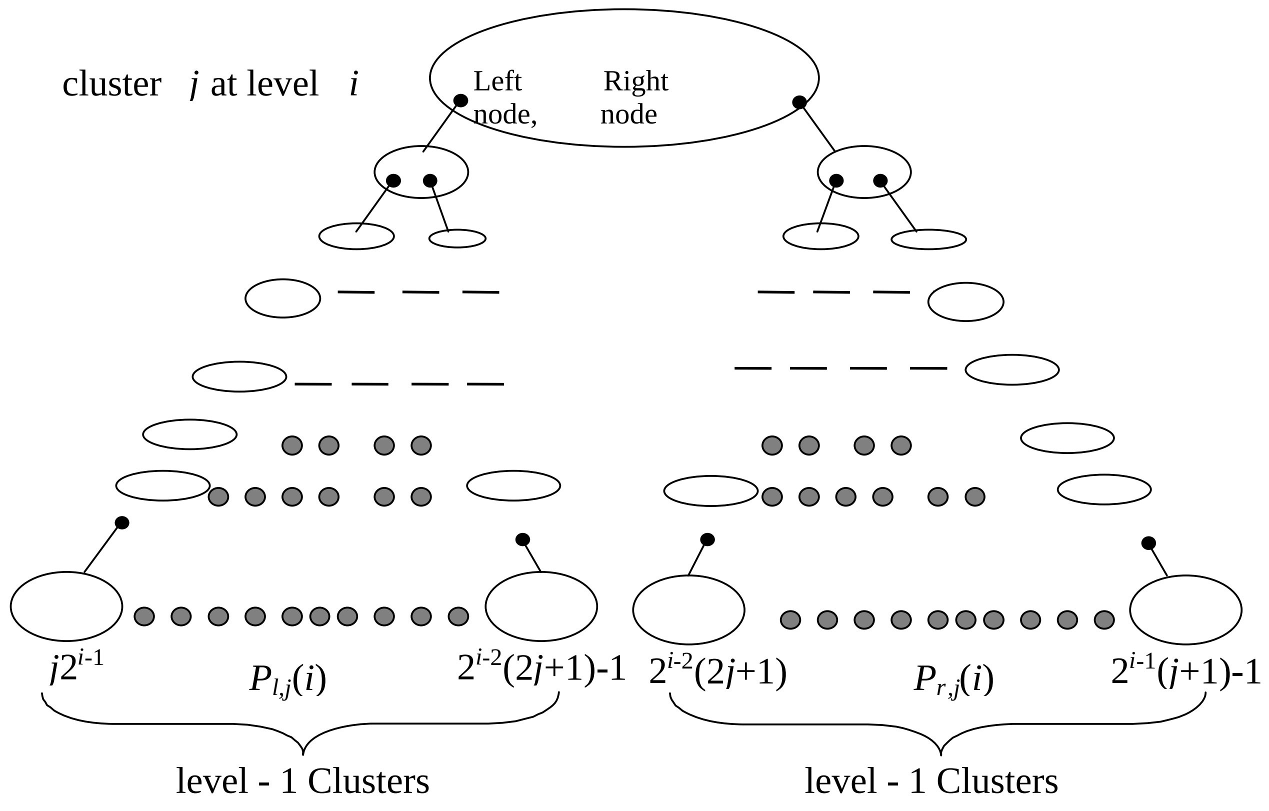


Fig. 2.18(b) Subtree representation at the node of cluster *j* at level *i*

half of this group of the level-1 clusters. That is, the left node (the node with the smallest id) in this cluster will represent the level-1 clusters numbered from $j2^{i-1}$ to $2^{i-2}(2j+1)-1$, and the right node will represent the clusters numbered from $2^{i-2}(2j+1)$ to $(j+1)2^{i-1}-1$. We shall refer by $P_{l,j}(i)$ the set of level-1 clusters represented by the left node in cluster j at level i, and by $P_{r,j}(i)$ the set of clusters represented by the right node in jth cluster at level i. See Figure 2.18(b) for a pictorial explanation of these definitions.

If $M < \log p + 1$, then the level-1 clusters represented by the jth node at level M are those at the leaves of the subtree rooted at this node and will be denoted by $P_j(M)$.

Level-*i* Testing Rounds

If $M = \log p + 1$, then nodes at level i perform level-i testing as follows. The node in the cluster j at level i representing $P_{l,j}(i)$ will test the nodes in the level-1 clusters in the set $P_{r,j}(i)$ until it finds the smallest fault free node, if such a node exists, and updates in its TESTED_UP array the status of the nodes in these clusters. The other node in the cluster j at this level representing $P_{r,j}(i)$ will test the nodes in the level-1 clusters in the set $P_{l,j}(i)$ until it finds the smallest fault free node , if it exists, and updates in its TESTED_UP array the status of nodes in these clusters.

If $M < \log p +1$, then nodes at level M perform the level-M testing as follows. The j^{th} node in this cluster will test the nodes not in $P_j(M)$ until it finds the smallest fault free node in a cluster greater than its cluster (modulo p) and updates the TESTED_UP array with the status of nodes in other clusters. This is similar to the level-2 testing round definition in section 2.3.2.

With these definitions we are now ready to present the ML-ADSD algorithm for the general case of level greater than two.

- **ML-ADSD Algorithm**

Initially all nodes are regular nodes and are at level 1.

During each testing round each node v executes the following.

Let k be the current level of node v.

CASE 1: If $k == 1$, then do the following.

- Execute the level-1 testing algorithm of Figure 2.19
- Execute the leader election algorithm of Figure 2.12
- If node v is a leader, then set status(v) = *leader*
- If node v is not a leader then execute the algorithm in Figure 2.20 and update the TESTED_UP array with the status of nodes in other clusters.

CASE 2: If $k > 1$, then do the following.

- Execute the level-k testing algorithm of Figure 2.21.

CASE 2.1 If v detects a cycle (using the LEADER array) then do the following.

▪ If k is not the last level then do:

(a) If v is not a leader then change the level of v to 1 and End testing round.

(b) If v is a leader at level k, then change the level of v to k+1 and End testing round.

▪ If k is the last level, then change the level of v to 1 and End testing round.

CASE 2.2 If v does not detect a cycle at level k, then test

▪ If v is still the fault free node with the smallest id in the group of level-1 clusters at the leaves of the subtree rooted at v.

- If not, change the level of v to 1.
- Otherwise, execute the level-1 testing algorithm (see Figure 2.19) and End testing round.

```
/* Multi-level Level-1 testing algorithm at node $n_{x,a}$*/
1.  $y = x$            //assign my node id
2.  repeat {
3.      $y = (y + 1) \bmod N/p$
4.      request $n_{y,a}$ to forward TESTED_UP$_{y,a}$ to $n_{x,a}$
5.  } until ($n_{x,a}$ tests $n_{y,a}$ as “fault-free”)

6.  for $node = 0$ to $(N/p - 1)$              // update
    local cluster info.
7.      TESTED_UP$_{x,a}$[$node$][$a$] =
    TESTED_UP$_{y,a}$[$node$][$a$]

8.  TESTED_UP$_{x,a}$[$x$][$a$] = $y$                // $x$ itself
    tests $y$ as fault free

9.  End of level-1 testing
```

Fig. 2.19 Multi-level level-1 testing algorithm

```
// Update information regarding other clusters at
node $n_{x.a}$
1.  for $cluster = 0$ to $(p - 1)$
2.          for $node = 0$ to $(N/p - 1)$
3.              if ($cluster \neq a$)
4.                  TESTED_UP$_{x,a}$[$node$][$cluster$] =
    TESTED_UP$_{y,a}$[$node$][$cluster$]
```

Fig. 2.20 Information update for non-leaders

2.3.9 Proof of Correctness and Diagnosis Latency of the ML-ADSD Algorithm

As in our discussion in the previous section on the

```
/* Multi-level level k testing algorithm */
1. If $M = log\ p +1$, then nodes at level $i$ perform level-$i$ testing rounds as follows.
    • The node in the cluster $j$ at level $i$ representing $P_{l,j}(i)$ will test the nodes in the level-1
      clusters in the set $P_{r,j}(i)$ until it finds the smallest fault free node, if such a node exists,
      and updates in its TESTED_UP array the status of nodes in these clusters. The other
      node in the cluster $j$ at this level representing $P_{r,j}(i)$ will test the nodes in the level-1
      clusters in the set $P_{l,j}(i)$ until it finds the smallest fault free node , if it exists, and
      updates in its TESTED_UP array the status of nodes in these clusters.
2. If $M < log\ p +1$, then nodes at level $M$ perform the level-$M$ testing rounds as follows.
    • The $jth$ node in this cluster will test the nodes not in $P_j(M)$ until it finds the smallest
      fault free node in a cluster greater than its cluster (modulo $p$) and updates the
      TESTED_UP array with the status of nodes in other clusters.
```

Fig. 2.21 Multi-level level-*k* testing algorithm

proof of correctness of the 2-level algorithm, we view the ML-ADSD algorithm as consisting of three phases. We show that after executing certain number of testing rounds after the last fault event, all nodes acquire correct view of the status of all the nodes in the network.

First, we shall assume that the number of levels $M = log\ p + 1$.

Case 1: No faults occur

Phase 1: In at most ***N/p* level-1 testing rounds** after the last fault event, all nodes in each level-1 cluster will get consistent and correct fault status information of all nodes in that cluster, and the leader of each cluster will be selected.

Phase 2: The leaders (left nodes of the two clusters at level *log p* of the first cluster (*0*th cluster) and the *N/*2-1th cluster) will reach the last level in 2(*log p*-1) testing rounds, and then perform two testing rounds. At the end of these **2*logp* testing rounds**, these two nodes will have correct fault status information of all the nodes in the network.

- While the left nodes of the two clusters at level *log p* move to the last level, the right nodes of these clusters move to level 1. After performing one testing round at level 1, these right nodes will move to level *log p* after 2(*log p*-2) testing rounds and perform two additional testing rounds at that level to collect information from the left nodes in their respective clusters the correct fault status information of all the nodes in the network. In all, they perform **2(*log p*-1) + 1 testing rounds** to collect correct fault status information of all the nodes in the network.
- Continuing as above, in general, the right nodes of the clusters at level i perform $2(i-1) + 1$ testing rounds to collect the correct fault status information of all the nodes in the network. Note that $2 \le i \le log\ p$.

Phase 3: Finally, the nodes in all level-1 clusters will collect from their respective leaders the correct fault status information of all the nodes in the network in at most ***N/p* level-1 testing rounds**

Combining all these testing rounds, all the nodes will have correct fault status information of all the nodes in the network in at most

$N/p + [\ 2\ log\ p + 2\ (log\ p - 1) + 2\ (log\ p - 2) +\dots\dots\dots+ 2\ ] + log\ p - 1 + N/p$

$= 2\ N/p + (log\,p + 1)\ (log\ p) + log\ p - 1$

$= 2\ N/p + (log\,p + 2)\ log\ p - 1$

testing rounds.

Case 2:Only Faulty nodes at level 1 recover

As in Case 2 in the discussion in the previous section, all nodes will identify their cluster leaders in Phase 1 in at most *N/p* or *N/p* + 1 testing rounds. The other phases will proceed as in Case 1 above. So the diagnosis latency in this case is one more than that for the first case.

Case 3: Faulty nodes at levels greater than one may recover

In this case, an additional at most 2 *log p* testing rounds will be required for all the fault free nodes to return to level 1. So, in this case the diagnosis latency is **2 *N/p* + (*log p* +4) *log p* testing rounds**.

Summarizing the above, we have the following:

Theorem 2: If $M = \log p + 1$, the diagnosis latency of the M-Level ML-ADSD algorithm is at most **2 *N/p* + (*log p* + 4) *log p* testing rounds**.

Proceeding as above, we can determine the diagnosis latency for the M-level ML-ADSD algorithm as in Theorem 3.

Theorem 3: The diagnosis latency of the M-level algorithm (M > 2) is at most
2 *N/p* + (M-2)(M+4) + *p*2 $^{(-M+3)}$ + 1 testing rounds.

We wish to note that other implementations of our multilevel scheme are possible. For instance, at each level we can combine more than 2 clusters to form clusters for the next higher level. Diagnosis latency calculation and proof of correctness in these cases will proceed as above with some appropriate minor changes.

2.4 Simulation and Discussion

In this section, we present the results of our simulation of the ADSD [1], Hi-ADSD [2] and our ML-ADSD algorithms (for two-level and three-level schemes) for networks of various sizes. The algorithms were simulated using the discrete-event simulation language *SSS* . The fail-stop model was used. All of the network nodes are modeled as independent processes and each node is assigned a unique node identifier.

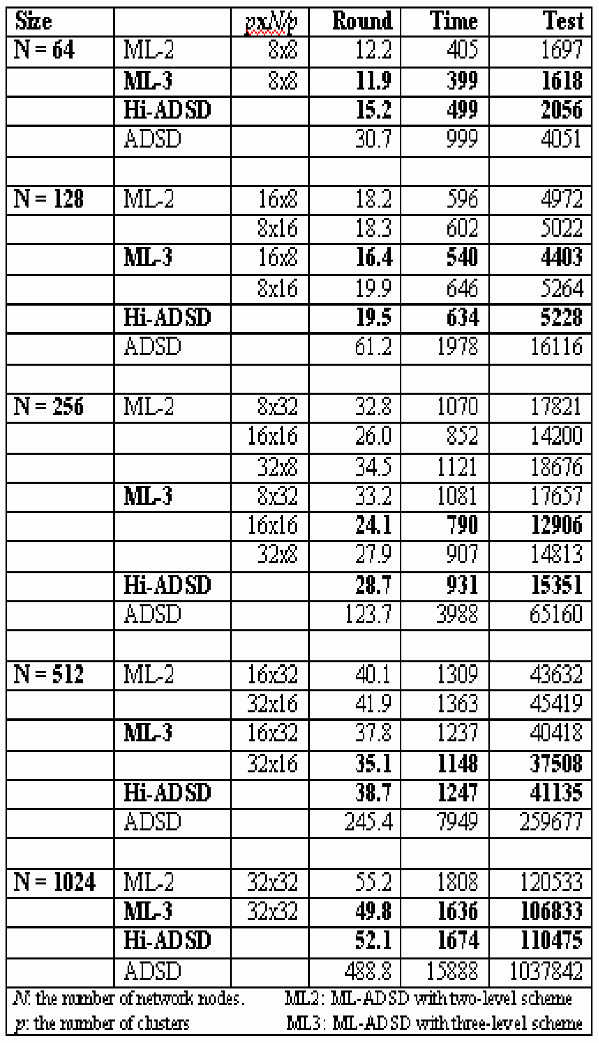
Three types of events are defined: test, fault occurrence and recovery.

Also, tests are scheduled for each node at each 30 ± σ time units as in [2], where σ is a random number in the range of 0 and 3. This is the time interval between two consecutive testing rounds at a node.

The fault event is modeled as the process being in *faulty* state and the recovery as the process being in *recovery* state. During each test, the status of the node is checked and, if the node is fault-free, the whole diagnosis information stored in the tested fault-free node is copied to the testing node. If the tested node is faulty, the testing nodes proceed testing as in the algorithm.

Experiments are conducted for all three algorithms on networks of different sizes. In each simulation, we first made five percent of the nodes fail. We then allowed 60% of these failed nodes to recover. In all, there were $0.08*N$ event fault events. These events occurred about $x * (30 \pm \sigma)$ time units apart, where x is a random number between 0 and 5. There are 30 + σ time units between the last fault event and the first recovery event. The average diagnosis latency in terms of testing rounds and also in testing time units as well the total number of test messages exchanged from the last event and until all the fault free nodes have the same correct diagnosis information were collected. The average was over 50 simulation runs. The results are presented in Table 2.2 and Figure 2.22.

Table 2.2 Simulation results for network sizes from 64 to 1024 nodes



Size		pxN/p	**Round**	**Time**	**Test**
N = 64	ML-2	8x8	12.2	405	1697
	ML-3	8x8	**11.9**	**399**	**1618**
	Hi-ADSD		**15.2**	**499**	**2056**
	ADSD		30.7	999	4051
N = 128	ML-2	16x8	18.2	596	4972
		8x16	18.3	602	5022
	ML-3	16x8	**16.4**	**540**	**4403**
		8x16	19.9	646	5264
	Hi-ADSD		**19.5**	**634**	**5228**
	ADSD		61.2	1978	16116
N = 256	ML-2	8x32	32.8	1070	17821
		16x16	26.0	852	14200
		32x8	34.5	1121	18676
	ML-3	8x32	33.2	1081	17657
		16x16	**24.1**	**790**	**12906**
		32x8	27.9	907	14813
	Hi-ADSD		**28.7**	**931**	**15351**
	ADSD		123.7	3988	65160
N = 512	ML-2	16x32	40.1	1309	43632
		32x16	41.9	1363	45419
	ML-3	16x32	37.8	1237	40418
		32x16	**35.1**	**1148**	**37508**
	Hi-ADSD		**38.7**	**1247**	**41135**
	ADSD		245.4	7949	259677
N = 1024	ML-2	32x32	55.2	1808	120533
	ML-3	32x32	**49.8**	**1636**	**106833**
	Hi-ADSD		**52.1**	**1674**	**110475**
	ADSD		488.8	15888	1037842

N: the number of network nodes. ML2: ML-ADSD with two-level scheme
p: the number of clusters ML3: ML-ADSD with three-level scheme

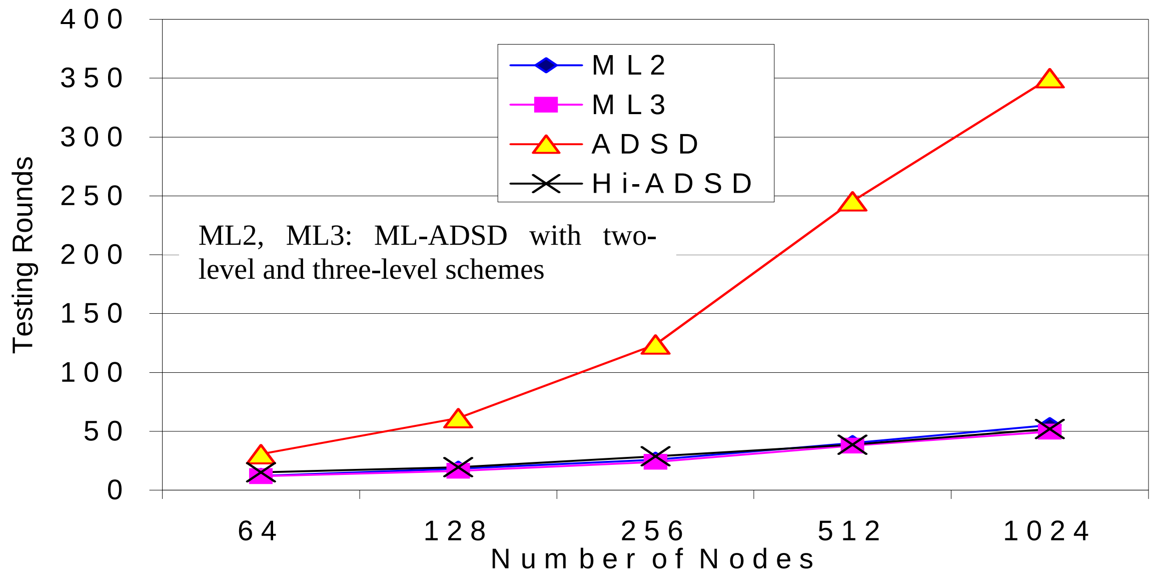


Fig. 2.22 (*a*) Comparison of diagnosis latencies in terms of testing rounds

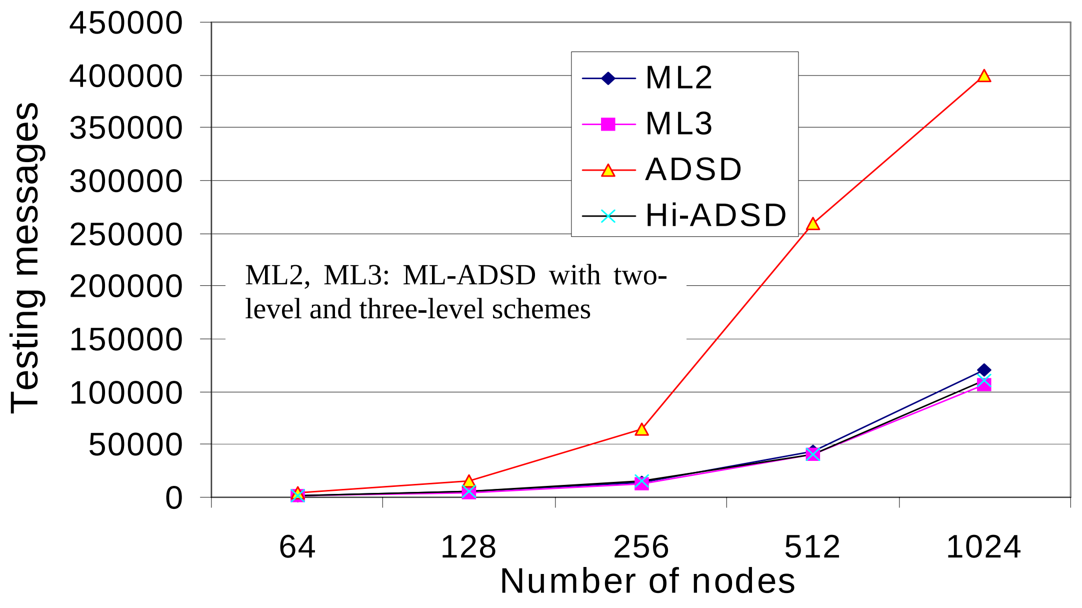


Fig. 2.22(*c*) Comparison of the total numbers of testing messages

Recall that the ADSD algorithm has a diagnosis latency of $O(N)$ and the Hi-ADSD algorithm has a diagnosis latency of $O(log^2N)$. As we can see from Table 2.2 and Figure 2.22 and as expected, in all respects, the performance of the ML-ADSD algorithm is much better than that of the ADSD algorithm. In all cases, the number of tests (messages) used by the ML-ADSD algorithm is smaller than the number for the Hi-ADSD algorithm. In all cases, the time required by the ML-ADSD algorithm is better than or the same as for the Hi-ADSD algorithm. Note that the performance of the ML-ADSD algorithm can be improved by an appropriate choice of the number of clusters and the number of levels. We would also like to point out that the ML-ADSD algorithm is scalable in the sense that only some minor modifications will be required to adapt the algorithm to networks of varying sizes. This property is not shared by the Hi-ADSD algorithm.

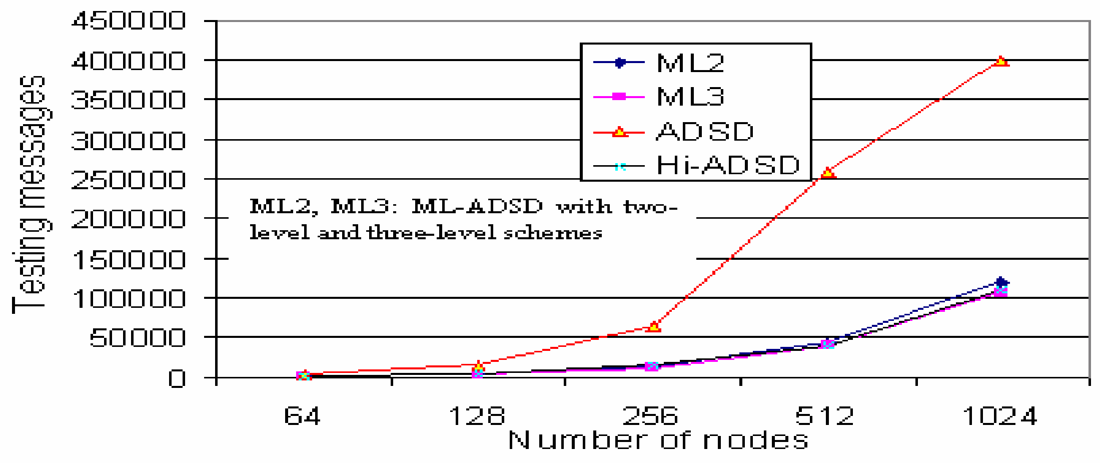


Fig. 22(c) Comparison of the total numbers of testing messages

2.4 References

[1] R. Bianchini, R. Buskens, “Implementation of on-Line distributed system level diagnosis theory”, *IEEE Transactions on Computers*, Vol. 41, No. 5, May 1992, pp. 616-626.

[2] Elias Procopio Duarte Jr., Takashi Nanya, “A hierarchical adaptive distributed system level diagnosis algorithm”, *IEEE Transactions on Computers*, Vol. 47, No. 1, Jan. 1998, pp. 34-45.

[3] Ming-Shan Su, K.Thulasiraman and Anindya Das, “ Multilevel adaptive distributed fault location in a network of processors “ *Proceedings of the Allerton Conference on Communication, Control and Computing*, October 2001.

[4] F. P. Preparata, G. Metze , and R. T.Chien, “ On the connection assignment problem of diagnosable systems” *IEEE Transactions on Electronic Computers*, EC-16, pp.848-854, 1967.