Hierarchical Adaptive Distributed System-Level Diagnosis Applied for SNMP-based Network Fault Management

Elias Procópio Duarte Jr. *† **Takashi Nanya** ‡
Tokyo Institute of Technology
Ookayama 2-12-1 Meguro-ku Tokyo 152 Japan
E-mail: {elias,nanya}@cs.titech.ac.jp

Abstract

*Fault Management is a key functional area of Network Management Systems, but currently deployed applications often implement rudimentary diagnosis mechanisms. This paper presents a new hierarchical adaptive distributed system-level diagnosis (*Hi-ADSD*) algorithm and its implementation based on SNMP (Simple Network Management Protocol). Hi-ADSD is a fully distributed algorithm that has diagnosis latency of at most $(log_2 N)^2$ testing rounds for a network of N nodes. Nodes are mapped into progressively larger logical clusters, so that each node executes tests in a hierarchical fashion. The algorithm assumes no link faults, a fully-connected network and imposes no bounds on the number of faults. Both the worst-case diagnosis latency and correctness of the algorithm are formally proved. Experimental results are given through simulation of the algorithm for large networks. The algorithm was implemented on a small network using SNMP. We present details of the implementation, including device fault management, the role of the Network Management Station (NMS), and the Diagnosis MIB (Management Information Base).*

1. Introduction

As computer networks have grown into complex, enterprise-wide systems, management of operations and associated risks has become a critical task. The goal of Network Management Systems is to monitor, interpret and control network operations, optimizing costs and reducing risks.

In the manager-agent paradigm, a Network Management System consists of a Network Management Station (NMS), also called monitor or manager, that queries a set of agents for information describing the state of links, devices, protocol entities, and nodes. Agents collect operational data (e.g. performance parameters) and detect exceptional events (e.g. error rates exceeding thresholds). This information is kept in the Management Information Base (MIB). Agents may issue alarms to inform the NMS about an exception. The NMS and the agents communicate through a network management protocol. Applications based on the Simple Network Management Protocol (SNMP) [1, 2, 3] are currently widely available.

Current network management systems often implement rudimentary fault diagnosis mechanisms. Consider a Local Area Network (LAN). The traditional approach to monitoring [6, 7] is to have a few managers, usually only one, organized in a tree, each of them responsible for querying a set of agents, and reporting to monitors in higher levels of the tree, as shown in figure 1. In these trees, agents ("Ag") are the leaves, and intermediate nodes are monitors ("mon") that implement both an agent process (i.e., SNMP server) and a manager process (i.e., SNMP client). This approach presents two drawbacks: (1) if monitors become faulty or unreachable, diagnosis stops on an entire portion of the network; (2) all monitors are required to test a large number of network nodes.

The field of distributed system-level diagnosis has flourished for years. Not only theoretical, but also practical implementations have been presented. In [15, 16] Bianchini and Buskens introduced the Adaptive Distributed System-level Diagnosis (Adaptive-DSD) algorithm, and also its implementation in an Ethernet environment. Adaptive-DSD has diagnosis latency of N testing rounds, for a network of N nodes, requiring that each node be tested by one node per testing round.

In this paper we present a new Hierarchical Adaptive Distributed System-level Diagnosis (*Hi-ADSD*) algorithm and its implementation integrated to a Network Manage-

*The author has a scholarship from the Brazilian research council, CNPq.

†also with the Departmento de Informática, Universidade Federal do Paraná, C.P. 19081 Curitiba PR, CEP 81531-990, Brasil

‡also with the Research Center for Advanced Science & Technology University of Tokyo, 4-6-1 Komaba, Meguro-ku, Tokyo 153, Japan

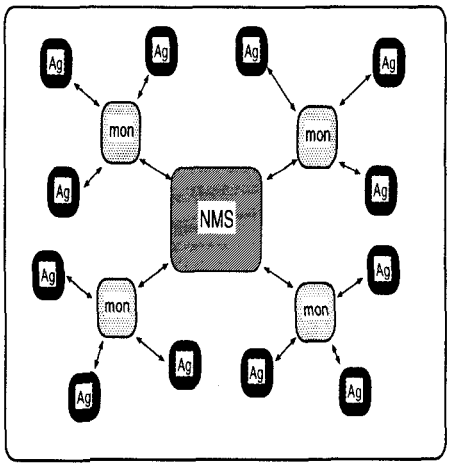

Figure 1. A common approach to network management monitoring.

ment System based on SNMP (Simple Network Management Protocol). Hi-ADSD is a fully distributed algorithm that has diagnosis latency of at most $log^2 N$ testing rounds for a network of N nodes. Nodes are grouped in progressively larger logical clusters, so that each node executes tests in a hierarchical fashion. The algorithm assumes no link faults, a fully-connected network and imposes no bounds on the number of faults. All logarithms used in this paper are base 2.

The rest of the paper is organized as follows. Section 2 reviews distributed system-level diagnosis. In section 3 the Hierarchical Adaptive Distributed System-level (*Hi-ADSD*) algorithm is specified and its correctness is formally proved. Section 4 shows experimental results of diagnosis on large networks obtained through simulation. Section 5 presents details of the implementation of Hi-ADSD integrated to an SNMP-based Network Management System, including device fault management, the role of the Network Management Station (NMS), and the Diagnosis MIB. This is followed by conclusions in section 6.

2. Adaptive Distributed System-Level Diagnosis

Consider a system consisting of N units, which can be faulty or fault-free. The goal of system-level diagnosis is to determine the state of those units. For almost 30 years researchers have worked on this problem, and the first model of diagnosable systems was introduced by Preparata, Metze, and Chien, the *PMC Model* [8]. In the PMC model units are assigned a subset of the other units to test, and fault-free units are able to accurately assess the state of the units they test. The set of all tests makes up a testing graph, i.e., a directed graph in which vertices represent the system's units and an edge from vertex i to vertex j corresponds to a test performed by unit i on unit j.

The collection of all test results is called the *syndrome* of the system. The problem of diagnosis is to obtain the state of the system from a given syndrome. The PMC model assumes the existence of a *central observer* that, based on the syndrome, can diagnose the state of all the units. For a given testing assignment the diagnosability of a system may be limited by the number of faulty units, and determining this number is called the *diagnosability problem*. Preparata *et al.* showed that for a system to be t-diagnosable, it is necessary that $N \geq 2t + 1$, and each unit is tested by at least t other units. Later, Hakimi and Amin [9] proved that if no two units test each other this conditions are sufficient for t-diagnosability.

Early system-level diagnosis algorithms assumed that all the tests had to be decided in advance. The tests were then executed, and from the obtained results, it was determined which units were faulty. Those algorithms focused on finding properties of the testing graph which would allow the observer to identify the faulty units from the tests corresponding to the testing graph's edges.

An alternative approach, which requires fewer tests, is to assume that each unit is capable of testing any other, and to issue the tests adaptively, i.e., the choice of the next tests depends on the results of previous tests, and not on a fixed pattern. Hakimi and Nakajima called this approach *adaptive* [10]. Early adaptive system-level diagnosis results assumed the existence of the previously mentioned central observer. Furthermore, a bound on the number of faulty nodes was imposed for the system to achieve correct diagnosis.

Adaptive system-level diagnosis algorithms proceed in testing rounds, i.e., the period of time in which each unit has executed the tests it was assigned. To evaluate adaptive algorithms two measures are normally used: the total number of tests required per testing round and the diagnosis latency, or delay, i.e., the number of testing rounds required to determine the state of the units.

Previously, Kuhl and Reddy [11, 12], introduced *distributed* system-level diagnosis, in which fault-free nodes reliably receive test results through their neighbors, and each node independently performs consistent diagnosis. They proposed the SELF distributed system-level diagnosis algorithm, that although fully distributed, is non-adaptive, i.e., each unit has a fixed testing assignment, and the number of faulty units in the system cannot exceed t. We will use alternatively the word node for unit, and network for system.

Later, Hosseini, Kuhl and Reddy, [13] extended the SELF algorithm, introducing the NEW-SELF algorithm, which also has a fixed inter-node test assignment, but is executed on-line, permitting faulty nodes to reenter the network

after being repaired. NEW-SELF ensures the accuracy of test-results by restricting the forwarding of testing results to fault-free nodes. For correct diagnosis, NEW-SELF requires that every fault-free node receive all test results from all other fault-free nodes. To reduce the amount of network resources required for diagnosis, the EVENT-SELF algorithm was proposed by Bianchini *et.al.*[14] This algorithm uses event-driven techniques to improve both the diagnosis latency and the impact of the algorithm on network performance.

The Adaptive Distributed System-level Diagnosis algorithm, *Adaptive-DSD*, was introduced by Bianchini and Buskens [15, 16]. Adaptive-DSD is at the same time distributed and adaptive. Each node must be tested only one time per testing interval. All fault-free nodes achieve consistent diagnosis in at most N testing rounds. There is no limit on the number of faulty nodes for fault-free nodes to diagnose the system.

Adaptive-DSD is executed at each node of the system at predefined *testing intervals*. Each time the algorithm is executed on a fault-free node, it performs tests on other nodes until it finds another fault-free node, or it runs out of nodes to test. A *testing round* is defined as the period of time in which all nodes of the system have executed Adaptive-DSD at least once. After one testing round, if there are at least two fault-free units, the testing graph has the format of a ring, as shown in figure 2. In the example shown in figure 2, node 1, node 4, and node 5 are faulty, and the rest are fault-free. Node 0 tests node 1 and finds it faulty; so it goes on and tests node 2, which is fault-free, and then stops testing. Node 2 then tests node 3 as fault-free, and so on.

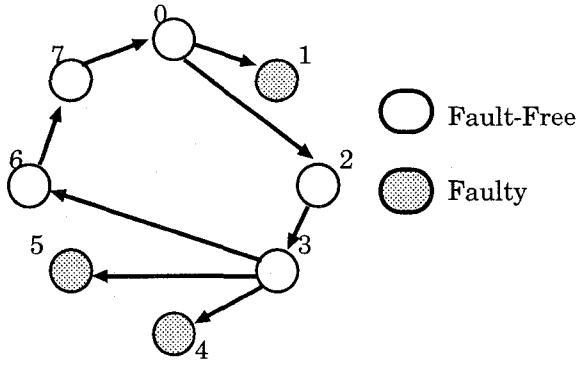

Figure 2. Example of test assignment in Adaptive-DSD.

Each node i that executes the algorithm has an array called TESTED-UP$_i$, that contains N entries, indexed by the node identifier. The entry TESTED-UP$_i[k] = j$ means that the node i has received diagnostic information from a fault-free node specifying that node k has tested j to be fault-free. An entry TESTED-UP$_i[j]$ is "arbitrary" if node j is faulty.

When node i finds node j to be fault-free, it saves this information in TESTED-UP$_i[i]$. In the next testing round, this test data of i is taken by its first fault-free predecessor, and so on, until all nodes get the information. In this way, the diagnostic information in the TESTED-UP array is forwarded to nodes in the reverse direction of the testing network. Using the information in TESTED-UP$_i$ a node i has to diagnose the state of all nodes in system, for this task another algorithm, called *Diagnose* is employed.

Adaptive-DSD has a diagnosis latency of N testing rounds. It is desirable to reduce this latency. In the original papers, Bianchini and Buskens [15, 16], use event-driven mechanisms to reduce the latency, like employing multicast or broadcast just after a new situation is identified. In certain systems it may be unnecessary or even impossible to introduce these extra event-driven mechanisms. In the next section, we introduce a new Hierarchical Adaptive Distributed System-level Diagnosis (Hi-ADSD) algorithm. Hi-ADSD is hierarchical in the sense that it employs a divide-and-conquer testing strategy [20]. Hi-ADSD is the first hierarchical diagnosis algorithm that is at the same time adaptive and distributed. The algorithm has diagnosis latency of log^2N rounds in the worst case, without employing extra event-driven mechanisms, and requiring less diagnostic information than Adaptive-DSD.

The results discussed here assume a fully connected network, no link faults and the PMC fault model. Besides the PMC fault model, many other fault models have been proposed. For example, a survey of probabilistic diagnosis is presented in [19]. Diagnosis of link faults were treated in [17]. Diagnosis on networks of general topology has received a great deal of attention recently, e.g. [18].

3. Hierarchical System-Level Diagnosis

In this section the Hierarchical Adaptive Distributed System-Level Diagnosis *(Hi-ADSD)* algorithm is presented, its correctness is formally proved, and it is compared to the Adaptive-DSD algorithm. Hi-ADSD maps nodes to clusters, which are sets of nodes, and employs a divide-and-conquer testing strategy to permit nodes to independently achieve consistent diagnosis in at most log^2N testing rounds.

Before the algorithm is specified, it is important to recall the concepts of *test* and *testing round*, to avoid confusions. These concepts are the same used by Bianchini and Buskens for Adaptive-DSD in [15, 16]. At specified time intervals, for example 30 seconds, each fault-free node in the system executes *tests* on other nodes of the system, until each node finds another node in the system that is fault-free, or tests all

other nodes as faulty. For instance, if the first node tested is fault-free, the tester stops testing. 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. The *diagnosis latency* of Hi-ADSD is once more the same as used for Adaptive-DSD, being defined as the number of *testing rounds* required for all fault-free nodes in the system to achieve diagnosis.

3.1 Algorithm Specification

Consider a system S consisting of a set of nodes n_i, each of which is assigned a unique natural identifier $i = 0, 1, ..., N-1$. In this paper we alternatively refer to node n_i as *node* i. The system is assumed to be fully connected, i.e., there is a communication link between any two nodes (n_i, n_j). Each node n_i is assumed to be in one of two states, *faulty* or *fault-free*. A combination of the state of all nodes constitutes the system's fault situation. Nodes perform tests on other nodes in a testing interval, and fault-free nodes report test results reliably.

In Hi-ADSD, nodes are grouped into *clusters* for the purpose of testing. Clusters are sets of nodes. The number of nodes in a cluster, its size, is always a power of two. Initially, N is assumed to be a power of 2, and the system itself is a cluster of N nodes.

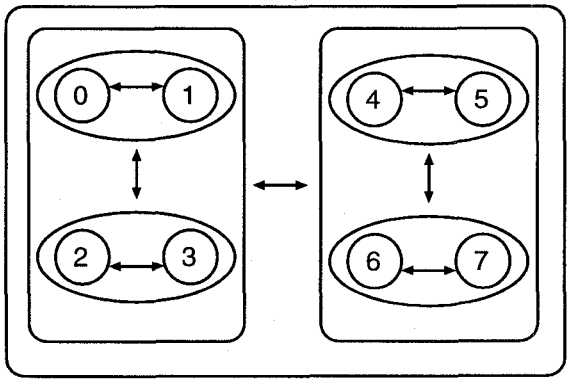

Figure 3. A hierarchical approach to test clusters.

A cluster of n nodes $n_j, ..., n_{j+n-1}$, where $j \text{ MOD } n = 0$, and n is a power of two, is recursively defined as either a node, in case $n = 1$; or the union of two clusters, one containing nodes $n_j, ..., n_{j+n/2-1}$ and the other containing nodes $n_{j+n/2}, ..., n_{j+n-1}$. Figure 3 shows a system with eight nodes organized in clusters.

In the first testing interval, each node performs tests on nodes of a cluster that has one node, in the second testing interval, on nodes of a cluster that has two nodes, in the third testing interval, on nodes of a cluster that has four nodes, and so on, until the cluster of $2^{\log N - 1}$ nodes is tested. After that, the cluster of size 1 is tested again, and the process is repeated.

The lists of ordered nodes tested by node i in a cluster of size 2^{s-1}, in a given testing interval, are denoted by $c_{i,s}$. The following expression completely characterizes list $c_{i,s}$, for all $i = 0, 1, ..., N-1$, and $s = 1, 2, ..., logN$. In the expression, a DIV b is the quotient of the integer division of a by b, and a MOD b is the remainder of the same integer division.

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

Where:

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

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

When node i performs a test on nodes of $c_{i,s}$, it performs tests sequentially, until it finds a fault-free node, or all other nodes are faulty. Supposing a fault-free node is found, from this fault-free node, node i copies diagnostic information of all nodes in $c_{i,s}$. For the system in figure 3, for all i and s, $c_{i,s}$ is listed in table 1.

If all nodes in $c_{i,s}$ are faulty, node i goes on to test $c_{i,s+1}$ in the same testing interval. Again, if all nodes in $c_{i,s+1}$ are faulty, node i goes on to test $c_{i,s+2}$ and so on, until it finds a fault-free node. For example, figure 4 shows the testing hierarchy for 8 nodes, from the viewpoint of node 0. When node 0 tests a cluster of size 2^2, it first tests node 4. If node 4 is fault-free, node 0 copies diagnostic information regarding nodes 4,5,6 and 7. If node 4 is faulty, node 0 tests node 5, and so on.

Hi-ADSD uses a tree to store information about the tests in all clusters. To effectively diagnose the state of all nodes, it is sufficient to list all nodes in the tree. Figure 5 shows the tree for node 0, for the case that all nodes are fault-free.

The algorithm is given in figure 6.

It is important to observe that the system is asynchronous, i.e., at any time, different nodes in the system may be testing clusters of different sizes. In other words, a node running Hi-ADSD does not know which tests are being performed by other nodes at any time. Even if nodes could be initially synchronized, after some of them become faulty and recover, the system would lose the initial synchronization. If there are at least two fault-free nodes in the system, in a testing round of Hi-ADSD, each node has

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

Table 1. $c_{i,s}$, for the system in figure 3.

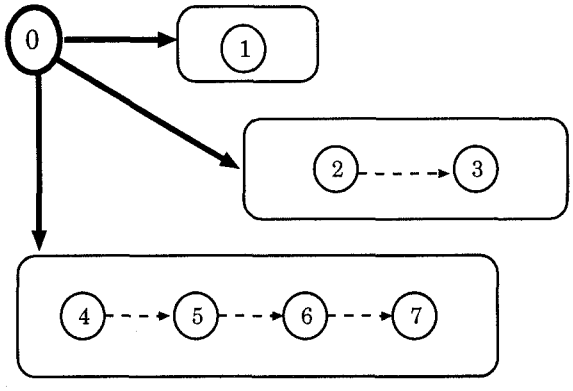

Figure 4. Each node adaptively tests all clusters.

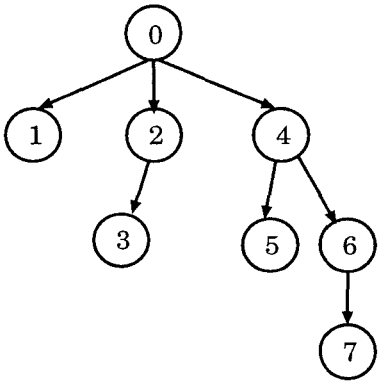

Figure 5. A tree keeps all testing information.

tested at least one other fault-free node in c_{i,s_t}, but the other nodes don't know which s_t. This fact has major consequences on the performance of the algorithm, as will be seen in the next subsection.

It is assumed that a node cannot fail and recover from that failure during the time between two tests by another node. In Hi-ADSD this time may be of up to $logN$ testing rounds, in the worst case. This assumption can be enforced by, for example, recording and storing fault events, or by reducing the testing interval between consecutive tests [15].

In Hi-ADSD, like in Adaptive-DSD, whenever a faulty node becomes fault-free, it doesn't have complete diagnostic information for log^2N testing rounds. The algorithm can be easily modified to incorporate this transient behaviour: if a tested node has been fault-free for less than log^2N testing rounds, then the tester should test another fault-free node. However, during the algorithm initialization, every node has been fault-free for less than log^2N testing rounds. To guarantee the correct initialization, nodes that are fault-free for less than log^2N testing rounds must also test and obtain diagnostic information from other nodes that have been fault-free for less than log^2N testing rounds.

3.2 Correctness Proof

We now proceed to prove the correctness and the worst case of the diagnosis latency of the algorithm. For this proof we assume a system fault situation that doesn't change for an enough amount of time. The correctness proof of Adaptive-DSD also carried this assumption.

We begin by defining the *Tested-Fault-Free* graph, $T(S)$. In $T(S)$ for each node i and each $c_{i,s}$, there is an edge directed from node i to the last node that node i tested as fault-free in that $c_{i,s}$. If, in the most recent testing interval in which node i tested $c_{i,s}$, all nodes in $c_{i,s}$ were tested as faulty, then $T(S)$ doesn't contain an edge from node i to any node in that $c_{i,s}$.

For example, consider the system in figure 3. The Tested-Fault-Free graph of that system contains a directed edge from any node i to the last node that i tested as fault-free in $c_{i,1}$, another edge to the last node that i tested as fault-free in $c_{i,2}$, and another edge to the last node that i tested as fault-free in $c_{i,3}$.

Definition 1 *The Tested-Fault-Free graph $T(S)$ is a directed graph whose nodes are the nodes of S. For each node i, and for each cluster $c_{i,s}$, there is an edge (i,t), directed from i to $t \in c_{i,s}$ if i has tested t as fault-free in the most recent testing interval in which it tested $c_{i,s}$.*

Lemma 1 *For any node i, any given s, and at any given instant of time t_i, it takes at most $logN$ testing rounds for node i to test $c_{i,s}$.*

Proof: *This follows from the definition of the algorithm, i.e., at a given testing interval node i tests a cluster, and*

```
Algorithm Hi-ADSD;
{ at node i }
{ please refer to the text for c(i,s) }
{ j indexes the nodes of a given c(i,s) }
REPEAT
  FOR s := 1 TO logN DO
    REPEAT
     node_to_test := next in c(i,s);
     IF "node_to_test is fault-free"
     THEN "update cluster diagnostic information"
    UNTIL ("node_to_test is fault-free") OR
        ("all nodes in c(i,s) are faulty");
    IF "all nodes in c(i,s) are faulty"
    THEN "erase cluster diagnostic information";
  END FOR;
FOREVER
```

Figure 6. The Hi-ADSD algorithm.

looks for a fault-free node in that cluster. In one testing round, by definition, each fault-free node tests at least another fault-free node, if there is one. There may be at most logN clusters for node i to test. In logN consecutive intervals, at each interval a different cluster is tested. Thus, if node i executes exactly one successful test per testing round, it will take logN testing rounds for it to test all clusters. Therefore, in the worst possible case, for t_i immediately after a given cluster is tested, it will take up to logN testing rounds for that cluster to be tested again. □

Theorem 1 *The shortest path between any two fault-free nodes in $T(S)$ contains at most logN edges.*

Proof: *We will conduct an induction on t, for a system of 2^t nodes.*

First, consider a system of 2^1 nodes; each node tests the other, thus the shortest paths in $T(S)$ contain one edge.

Next, assume that for a system of 2^t nodes, a shortest path between any two nodes in $T(S)$ contains at most t edges. Then, by definition, in the system of 2^{t+1} nodes there are two clusters of 2^t nodes. Consider a subgraph of $T(S)$ that contains only the nodes in one of these clusters. By definition, this subgraph is isomorphic to the Tested-Fault-Free graph of a system of 2^t nodes. So, by the assumption above, the shortest path between any two nodes in this subgraph has at most t edges. Consider any two nodes, i and j. If i and j are in the same cluster of 2^t nodes, the shortest path between them in T(S) has at most t edges. Now, consider the case in which i and j are in different clusters of 2^t nodes. Without loss of generality let's consider the shortest path from i to j. Node i tests one node in the cluster in which j is contained, call this node p. In T(S), the shortest distance from i to p contains one edge, and the shortest distance from p to j contains at most t edges. Thus the shortest distance from i to j contains at most t + 1 edges. □

As an example, consider a system of size 2^2; this system has size four, and each node tests two other nodes, and gets information about the fourth node indirectly, through the tested nodes. This makes up a path of length two. Now consider a system of size 2^3, there are two clusters of size 2^2, and each node in one cluster tests one node in the other, thus, in T(S), there is an edge from each node in one cluster to the other. Therefore, the paths from a node in one cluster to the nodes in the other have lengths of the paths within the cluster which are at most of length 2, plus 1, for the edge linking the two clusters. Thus, in a system of size 2^3, the shortest path has length at most 3. For example, look to node 5 and node 2 on figure 7. For node 5 to get information about node 2, node 5 tests node 1, which tests node 3 which tests node 2. In this system of 8 nodes, the maximum path has size log 8.

Now let's consider each test in this worst case shortest path. How many testing rounds does it take to execute one test, in the worst case? Consider figure 7 again. If node 3 has tested node 2 just before it became faulty, then only after three testing rounds node 3 will discover that node 2 is faulty. Then, in the worst case, if node 1 tests node 3 just before node 3 tests node 2, it will take other three testing rounds for node 1 to discover that node 2 is faulty. If we are very unlucky and node 5 tested node 1 just before node 1 tested node 3, then it will take other three testing rounds for node 5 to discover that node 2 is faulty. In other words, there are three tests in the maximum path, and each one takes three testing rounds to be executed in the worst case, thus, in total, it may take up to nine testing rounds to execute all three tests.

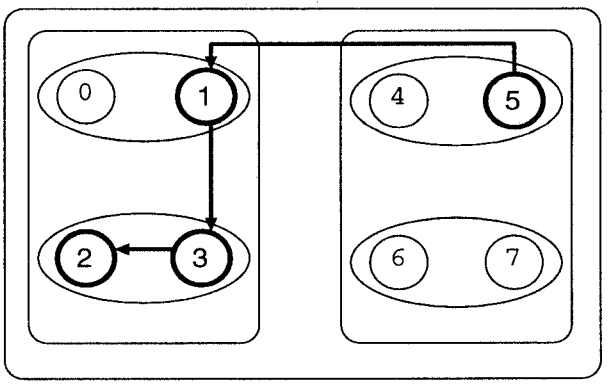

Figure 7. The shortest path from node 5 to node 2 has $\log 8 = 3$ edges.

Theorem 2 *Consider the system fault situation at a given time. After at most log^2N testing rounds, each node that has remained fault-free for that period correctly determines that fault situation.*

Proof: *It was proved in theorem 1 that the shortest path between any two nodes in $T(S)$ has at most $logN$ edges. But, from lemma 1, each of the tests corresponding to an edge in $T(S)$ can take up to $logN$ testing rounds to be executed in the worst case. In other words, there are up $logN$ different tests to execute, and each may take up to $logN$ testing rounds to be executed. So, in total, they may take at most $logN$*$logN$ testing rounds to be executed. Thus, it may take up to log^2N testing rounds for a fault-free node to obtain diagnostic information about an event in S.* □

These results also hold for a dynamic fault situation, in which multiple nodes become faulty and are repaired after a given event occurs. When a node becomes faulty, the lengths of the paths in T(S) that included that node are reduced of at least one unit, for a tester of this node will execute at least one more test. Furthermore, if a faulty node becomes fault-free, during the period of transient behavior described above, the tester of that node also executes at least one more test, until it gets to a node with complete diagnostic information. In this way nodes that are fault-free for log^2N testing rounds detect the fault situation of all nodes that haven't changed state for log^2N testing rounds, even if other nodes in the network have become faulty and/ or been repaired during that period.

We believe that, in average, nodes running Hi-ADSD achieve diagnosis in less than log^2N testing rounds, and our simulation results confirm this fact. As if nodes are roughly synchronized they will run the algorithm in $O(\log N)$ testing rounds, if extra synchronization mechanisms are introduced better bounds can be guaranteed.

It should be clear that in Hi-ADSD, like in Adaptive-DSD, there is no limit in the number of faulty nodes for fault-free nodes to perform consistent diagnosis. In the worst case, when $N-1$ nodes are faulty, the number of tests required still N, as shown by Bianchini and Buskens for Adaptive-DSD. For example, if $N-1$ nodes are faulty, the fault-free node must test all other nodes to diagnose the system.

It is not necessary that the number of nodes, N, be a perfect power of 2. In this case, the algorithm works as if for a system of $2^{\lceil \log N \rceil}$ nodes, $2^{\lceil \log N \rceil} - N$ nodes were faulty.

3.3 Comparison of Adaptive-DSD and Hi-ADSD

To compare Hi-ADSD and Adaptive-DSD we begin comparing the number of testing rounds required by both algorithms. We then compare the number of tests required, and conclude with the amount of diagnostic information that must be exchanged by nodes in the system until the fault situation is diagnosed.

The first difference between the two algorithms is their worst case diagnosis latencies, in terms of testing rounds.

N	Hi-ADSD	Adaptive-DSD
4	4	4
8	9	8
16	16	16
32	25	32
64	36	64
128	49	128
256	64	256
512	81	512
1024	100	1024

Table 2. Examples of diagnosis latency.

While Adaptive-DSD's diagnosis latency is N testing rounds, Hi-ADSD's is log^2N.

Table 2 lists the diagnosis latency in terms of testing rounds for both algorithms, for networks having from 4 to 1024 nodes. The figures in this table should be clearly understood. They show the number of testing rounds that are needed for all nodes in the system to diagnose one change in the fault situation. For example, if all nodes are fault-free, and one node becomes faulty, that diagnostic information will take N testing rounds, being transferred sequentially through N nodes until all nodes diagnose the situation. In Hi-ADSD, the diagnostic information will be transferred through a tree of depth $\log N$ and to reach all nodes it takes at most log^2N testing rounds. For networks of 4 and 16 nodes, the algorithms present the same worst case latency. In one case, for a network of 8 nodes, Adaptive DSD presents better latency then Hi-ADSD, but this changes quickly as the number of nodes grows.

To compare the number of tests required by both algorithms we show the number of tests required in one testing round and the number of tests required to achieve complete diagnosis. Both algorithms employ *approximately* the same number of tests per testing round, i.e. each fault-free node executes tests until it finds another fault-free node. If there are t faulty nodes in the system, Adaptive-DSD needs N tests per testing round, while Hi-ADSD needs $N+\epsilon$ tests. ϵ corresponds to the unlikely situation in which two or more nodes have the same entry point to a given cluster (which is bounded by $\log N$), they test this cluster at the same time, and this entry point is faulty. Furthermore, as the number of testing rounds required by Adaptive-DSD is N, and by Hi-ADSD is log^2N, the total number of tests required for diagnosis in Adaptive-DSD is $O(N^2)$ and for Hi-ADSD is $O(N \log^2 N)$.

Now consider the total number of diagnostic messages required by the algorithms. Adaptive-DSD requires a total of N^2 messages for all nodes to achieve diagnosis, while Hi-ADSD requires $N\ log^2N$ messages in the worst case. Extra mechanisms like timestamps [16] can be employed to

avoid transferring diagnostic messages unless strictly necessary. Using these mechanisms, for each event, only information regarding that event must be transfered.

If extra mechanisms are not employed to avoid transfering more than information regarding new events, then there is also a major difference in the size of diagnostic messages in Adaptive-DSD and Hi-ADSD. Nodes running Adaptive-DSD get messages with diagnostic information concerning all nodes in all testing intervals, in contrast, Hi-ADSD's diagnostic messages only contain information about the nodes in each cluster being tested. Let's call the information about one node a *diagnostic unit*. Consider $logN$ consecutive testing intervals, during this period, a node running Adaptive-DSD requires $NlogN$ diagnostic units, while a node running Hi-ADSD requires only $2^0+2^1+...+2^{\log N-1} = N-1$ units during the same period.

4. Simulation of Hi-ADSD

In this section we present experimental results of diagnosis on large networks using Hi-ADSD, obtained through simulation. The simulation was conducted using the discrete-event simulation language *SMPL* [21]. Nodes were modeled as SMPL facilities, and each node was identified by a SMPL token number. Three kinds of events were defined: (1) test, (2) fault, and (3) repair. Tests were scheduled for each node at each 30 seconds, considering time exponentially distributed. We modeled the fault as the facility being reserved, and the repair as the facility being released. During each test, the status of the facilities were checked, and if the node is fault-free diagnosis information regarding the cluster is copied to the testing node. If the tested node is faulty, the testing nodes proceeds testing as in the algorithm.

We conducted several experiments with networks of different sizes. In this paper we present results of two experiments: on a network of 512 nodes, a failure occurs at time 100, and is repaired at time 1100. On the second experiment, on a network of 64 nodes, after a node becomes faulty, a second node also becomes faulty, and after that they are sequentially repaired. These four events were scheduled for times 100, 1000, 2100 and 3000, respectively.

Results are shown in the graphs in figures 8 and 9. These graphs show representative simulation outcomes from the set of experiments executed. Both graphs have the number of testing rounds as the x-axis and the number of nodes that diagnosed the event as the y-axis. Figure 8 shows that after node 2 becomes faulty, it takes 9 testing rounds for the other 63 nodes to diagnose the event. For the other events in this graph, it takes 8 testing rounds for all fault-free nodes to achieve diagnosis.

Figure 9 shows that after node 2 becomes faulty, it takes 15 testing rounds for all 511 fault-free nodes to diagnose

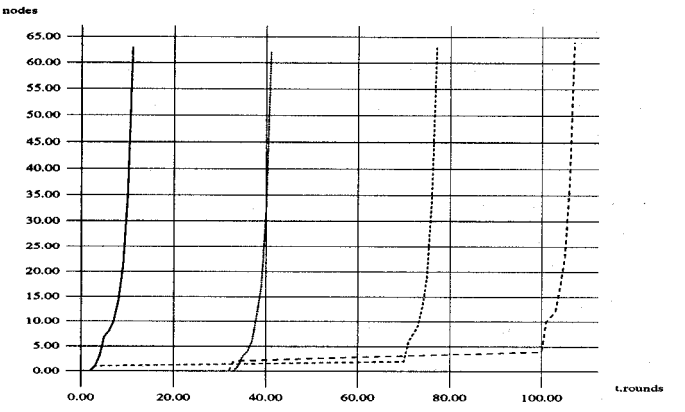

Figure 8. Simulation of Hi-ADSD for a 64-node network.

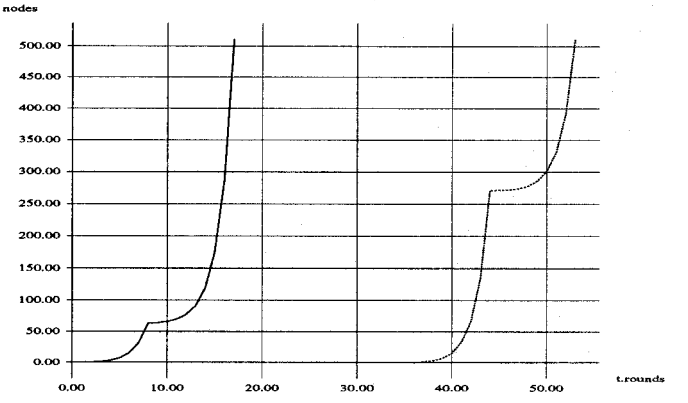

Figure 9. Simulation of Hi-ADSD for a 512-node network.

the fault event. After node 2 is repaired it takes 17 testing rounds for all nodes to diagnose the event. To compare with Adaptive-DSD, without extra event-driven mechanisms, we point out that Adaptive-DSD takes 511 testing rounds for all fault-free nodes to diagnose any event in this network.

5. An SNMP-based Fault Management Approach

In this section we present the application of hierarchical distributed system-level diagnosis results to SNMP-based fault management. We take into account that the primary goal of SNMP-based fault management is to permit a central NMS to determine the state of all nodes in the network, in a reliable and efficient way. By reliable, we mean that if any node fails in the network, the diagnosis process con-

tinues, even if the faulty-node is the current NMS itself. By efficient, we mean that the diagnosis is accomplished within a small delay, and the overhead imposed by diagnosis messages requires a reasonable percentage of network bandwidth.

One of the goals of network management systems is to provide network state information to the human manager at the NMS. The concept of a central observation point is not contradictory with the previously presented distributed approach: the NMS can be seen now as a *management interface*, and not as the single monitor. This approach gives a number of advantages to the human manager, as she/he has a choice of workstations from which to control the network. Furthermore, there are obvious advantages in terms of the reliability of the network monitoring system itself, as fault-free nodes achieve correct diagnosis for any number of faulty nodes.

It has been shown that Hi-ADSD has a diagnosis latency of at most $log^2 N$ testing rounds. To further reduce this latency at the NMS, a feasible solution is to employ SNMP traps, i.e., an agent reports any new state information as soon as it is discovered. This combination of distributed monitoring and traps gives the system high resilience over errors, while keeping delays conveniently short. The NMS receives all changes in state information as soon as they are discovered. Using a simple configuration mechanism, all stations are informed of the NMS identity. Furthermore, even if the NMS is changed (or becomes faulty) soon after receiving and acknowledging the trap, by the time another node assumes the role of NMS, the information is delivered to this new NMS through the testing network.

5.1 Network Device Fault Management

To permit Hi-ADSD to monitor the state of network devices, each unit is classified into a *testing* node or a *tested-only* node. *Testing* nodes are usually workstations, which are not only subject to tests, but are also capable of testing. In contrast, *tested-only* nodes are only tested, and don't perform any testing on other elements. A number of managed devices, like printers, modems, terminals, among others are *tested-only*. Furthermore, to improve the diagnosis delay, some workstations may be *tested-only*. In addition to participating in the logical testing network, each *testing* node has some associated *tested-only* nodes, that are tested at each testing interval. Whenever a *testing* node finds another *testing* node to be faulty it must test all *tested-only* nodes associated with that faulty *testing* node.

5.2 Diagnosis MIB

Hi-ADSD was implemented to demonstrate the operational potential of the SNMP-based distributed diagnosis. The implementation was done on a small system of four nodes. The CMU SNMP public-domain packet [4] was used as a base to implement the diagnosis agent, in which we coded the *Diagnosis MIB* variables. From the Sony News-OS SNMP application [5] we used client programs to access and update MIB variables.

The main portion of the *Diagnosis MIB* is the *Diag-Tree* which is implemented as an SNMP table. The ASN.1 coding is shown bellow:

```
DiagTree OBJECT-TYPE
 SYNTAX  SEQUENCE OF diagTreeEntry
 ACCESS  not-accessible
 STATUS  mandatory
 DESCRIPTION
  "This is the tree in which each testing node
   keeps network diagnostic info."
::= { diagnosis 1 }

diagTreeEntry OBJECT-TYPE
 SYNTAX  DiagTreeEntry
 ACCESS  not-accessible
 STATUS  mandatory
 DESCRIPTION
  "Each entry of DiagTree identifies
   which node the testing node recognized
   as up in the last testing round."
 INDEX   { testingID }
::= { DiagTree 1 }

TestedUPEntry ::=
 SEQUENCE {
  testingID    INTEGER,
  testingAD    IpAddress,
  testingUP    INTEGER,
  tested-only1 IpAddress,
  to-status1   INTEGER,
     ...
  tested-only5 IpAddress,
  to-status5   INTEGER      }
```

testingID is the identifier of each node, it varies from 1 to N. *testingAD* is the ip-address of the each node. *testingUP* contains 1 if the node is known to be fault-free, and 0 otherwise. Five slots were reserved for the associated *tested-only* nodes, for each *testing* node. *to-status-i* contains 1 if the *tested-only* node whose address is *tested-only-i* is fault-free, and 0 otherwise.

6. Conclusions

In this paper we have presented an efficient approach to LAN fault management based on distributed system-level diagnosis.

The Hierarchical Distributed System-level Diagnosis algorithm, Hi-ADSD, was presented. Hi-ADSD maps nodes

to clusters, and uses a divide-and-conquer testing strategy to achieve diagnosis in at most log^2N testing rounds. In this way Hi-ADSD improves the diagnosis latency of previous algorithms, while requiring that each node perform one test per testing interval.

Hi-ADSD was implemented integrated to an SNMP-based network management system of a small network. Issues regarding the actual deployment of the algorithm were discussed, like the role of the NMS, the Diagnosis MIB, and device fault management. We presented experimental results of diagnosis on large networks using simulation.

As SNMP applications are currently widely deployed, this implementation is an important step in the direction of improving the dependability of the Internet itself.

Acknowledgements

We wish to thank the many colleagues at Tokyo Institute of Technology that have helped in this project, especially: Yoichiro Ueno and Pavlin I. Radoslavov for their expert help in the SNMP implementation; Dr. Patrick Trane for fruitful discussions on the correctness proof; and Andreas Savva for his valuable comments on all phases of the project. Special acknowledgements are due to Fátima L.P. Duarte, for her expert help in the SMPL simulation, and to Prof. Doug Blough, of U.California Irvine, for his insightful comments on the algorithm.

References

[1] M. Rose, and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets," *RFC 1155*, 1990.

[2] J.D. Case, M.S. Fedor, M.L. Schoffstall, and J.R. Davin, "A Simple Network Management Protocol," *RFC 1157*, 1990.

[3] K. McCloghtie and M.T. Rose, "Management Information Base for Network Management of TCP/IP-based Internets," *RFC 1213*, 1991.

[4] M.T. Rose, *The Simple Book - An Introduction to Internet Management,* 2^{nd} ed., Prentice-Hall, Englewood Cliffs, NJ, 1994.

[5] News OS Release 4.2.1, *Administrator's Guide Vol.1*, "Network Management", Sony, Tokyo, 1991. *(In Japanese)*

[6] L. Steinberg, "Techniques for Managing Asynchronously Generated Alerts," *RFC 1224*, 1991.

[7] J. Herman, "Distributed Network Management," *Data Communications Magazine*, June, 1992.

[8] F. Preparata, G. Metze, and R.T. Chien, "On The Connection Assignment Problem of Diagnosable Systems," *IEEE Transactions on Electronic Computers*, Vol. 16, pp. 848-854, 1968.

[9] S.L. Hakimi, and A.T. Amin, "Characterization of Connection Assignments of Diagnosable Systems," *IEEE Transactions on Computers*, Vol. 23, pp. 86-88, 1974.

[10] S.L. Hakimi, and K. Nakajima, "On Adaptive System Diagnosis" *IEEE Transactions on Computers*, Vol. 33, pp. 234-240, 1984.

[11] J.G. Kuhl, and S.M. Reddy, "Distributed Fault-Tolerance for Large Multiprocessor Systems," *Proc. 7th Annual Symp. Computer Architecture*, pp. 23-30, 1980.

[12] J.G. Kuhl, and S.M. Reddy, "Fault-Diagnosis in Fully Distributed Systems," *Proc. FTCS-11*, pp. 100-105, 1981.

[13] S.H. Hosseini, J.G. Kuhl, and S.M. Reddy, "A Diagnosis Algorithm for Distributed Computing Systems with Failure and Repair," *IEEE Transactions on Computers*, Vol. 33, pp. 223-233, 1984.

[14] R.P. Bianchini, K. Goodwin, and D.S. Nydick, "Practical Application and Implementation of System-Level Diagnosis Theory," *Proc. FTCS-20*, pp. 332-339, 1990.

[15] R.P. Bianchini, and R. Buskens, "An Adaptive Distributed System-Level Diagnosis Algorithm and Its Implementation," *Proc. FTCS-21*, pp. 222-229 , 1991.

[16] R.P. Bianchini, and R. Buskens, "Implementation of On-Line Distributed System-Level Diagnosis Theory," IEEE Transactions on Computers, Vol. 41, pp. 616-626, 1992.

[17] C.-L. Yang, and G.M. Masson, "Hybrid Fault-Diagnosability with Unreliable Communication Links," *Proc. FTCS-16*, pp. 226-231, 1986.

[18] S.Rangarajan, A.T. Dahbura, and E.A. Ziegler, "A Distributed System-Level Diagnosis Algorithm for Arbitrary Network Topologies," *IEEE Transactions on Computers*, Vol.44, pp. 312-333, 1995.

[19] G. Masson, D. Blough, and G. Sullivan, "System Diagnosis," in *Fault-Tolerant Computer System Design,* ed. D.K. Pradhan, Prentice-Hall, 1996.

[20] E.P. Duarte Jr., and T. Nanya, "Multi-Cluster Adaptive Distributed System-Level Diagnosis Algorithms", *IEICE Technical Report FTS 95-73*, 1995.

[21] M.H. MacDougall, *Simulating Computer Systems: Techniques and Tools,* The MIT Press, Cambridge, MA, 1987.