44

Non-Broadcast Network Fault-Monitoring Based on System-Level Diagnosis

*Elias Procópio Duarte Jr.**
Takashi Nanya †
Tokyo Institute of Technology
Dept. Computer Science
Ookayama 2-12-1 Meguro-Ku
Tokyo 152 Japan
{*elias,nanya*}*@cs.titech.ac.jp*

Glenn Mansfield
Shoichi Noguchi
Sendai Foundation for Applied Information Sciences
Tsutsujigaoka 5-12-55
Miyagino-Ku Sendai 983 Japan
{*glenn,noguchi*}*@tia.ad.jp*

Abstract

Network fault management systems are mission-critical, for they are most needed during periods when part of the network is faulty. Distributed system-level diagnosis offers a practical and theoretically sound solution for fault-tolerant fault monitoring. It guarantees that faults don't impair the fault management process. Recently, results from the application of distributed system-level diagnosis applied for SNMP-based LAN fault management have been reported [1, 2]. In this paper we expand those results by presenting a new algorithm for diagnosis of non-broadcast networks, applied to point-to-point network fault management. In the algorithm, nodes test links periodically, and disseminate link time-out information to all its fault-free neighbors in parallel. Upon receiving link time-out information a node computes which portion of the network has become unreachable. This approach is closer to reality than previous algorithms, for it is impossible to distinguish a faulty node from a node to which all routes are faulty. The diagnosis latency of the algorithm is optimal, as nodes report events in parallel, and latency is proportional to the diameter of the network. The dissemination step includes mechanisms to reduce the number of redundant messages introduced by the parallel strategy. We present a MIB for the algorithm, and a SNMP-based implementation. The evaluation of algorithm's impact on network performance, shows that the amount of bandwidth required is less than 0.1% for popular link capacities. We conclude demonstrating the integration of LAN and WAN fault diagnosis into a unified framework.

Keywords

Distributed System-Level Diagnosis, Network Fault Management, SNMP

* also with Federal University of Paraná, Informatics Dept., C.P. 19081 Curitiba PR, CEP 81531-990, Brasil. The author has a scholarship from the Brazilian research council, CNPq.

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

A. A. Lazar et al. (eds.), *Integrated Network Management V*

1 INTRODUCTION

Fault management is the set of activities required to guarantee network availability, even in the presence of network faults and performance degradation. Fault management must thus be fault tolerant, for network faults should not impair the system that is meant to solve them. Fault management can be broadly subdivided into monitoring and control. Monitoring is the process employed for obtaining information required about the components of a network, in order to make management decisions and subsequently control their behavior. In this paper we present a fault-tolerant approach for non-broadcast network fault-monitoring based on the long standing theory and practice of distributed system-level diagnosis.

Current SNMP-based fault-management systems are based on the manager-agent model, in which a fixed manager station queries a set of agents for management information. This centralized scheme is inherently unreliable, for if the manager becomes faulty, network management stops on the entire network.

Hierarchical schemes are also popular, in which machines that participate in management form a tree, where the root is the main manager, leaves are agents and internal nodes run both the agent server and a monitoring application. An internal node of the tree monitors nodes in the subtree of which it is the root, and reports to its parent in the tree. Hierarchical schemes are also inherently unreliable, for whenever an internal node becomes faulty, monitoring stops on part of the network.

System-level diagnosis offers a theoretically sound and practical framework for fault-tolerant network monitoring: even if any part of the network becomes faulty, fault-free nodes are able to diagnose the system. Recently, results from the application of distributed system-level diagnosis applied for SNMP-based LAN fault management have been reported. In [1] the Adaptive Distributed System-level Diagnosis (ADSD) algorithm was implemented using SNMP. In this algorithm, nodes are assumed to be fully connected, and the testing topology is a ring, such that the number of tests is as low as one per node per testing round. For a network of N nodes, the diagnosis latency of the algorithm is N testing rounds, i.e. proportional to the number of nodes in the network.

In [2] another algorithm for LAN fault-diagnosis, Hi-ADSD (Hierarchical ADSD), was introduced and implemented using SNMP. In this algorithm, the number of tests is the same as in ADSD, but the testing topology is initially a hypercube, and diagnosis is reduced to $\log^2 N$ testing rounds. Those algorithms employ a distributed strategy for fault management, in which a collection of network nodes perform network diagnosis, and the human manager may attach an interface to any of these nodes to receive diagnostic information.

In this paper we expand those results by introducing an algorithm for diagnosis in non-broadcast networks, applied to point-to-point network fault management. In the algorithm, a node tests links periodically, and disseminates link time-out information to all its fault-free neighbors in parallel. Upon receiving link time-out information a node computes which portion of the system has become unreachable. This new approach to diagnosis, based on *link time-out* and *node unreachability* is closer to reality than previous approaches. There are two reasons for this improvement: (1) it is impossible to distinguish a node fault from the fault on all the paths to that node; (2) in previous algorithms, two fault-free nodes in disconnected components keep the old status for each other, which may not correspond to reality.

A node joining the algorithm disseminates information about itself, and collects diagnostic information from its neighbors. The diagnosis latency of the algorithm is optimal, as nodes report events in parallel, and latency is proportional to the diameter of the network. The dissemination step includes mechanism to reduce the number of redundant messages introduced by the parallel strategy. We present a MIB for the algorithm, and a SNMP-based implementation. The evaluation of algorithm's impact on network performance shows that the amount of bandwidth required is less than 0.1% for popular link capacities. We conclude demonstrating the integration of LAN and WAN fault diagnosis into a unified framework.

The rest of the paper is organized as follows. Section 2 reviews system-level diagnosis, including algorithms for LAN fault management. Section 3 reviews algorithms for diagnosis on networks of general topology, and includes the specification of the new algorithm for non-broadcast networks. In section 4 we present a MIB and a SNMP-based implementation of the algorithm. In section 5 we evaluate its impact on network performance. Section 6 concludes the paper, showing the integration of LAN, and WAN fault diagnosis into a unified framework.

2 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* [3]. 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 PMC model assumes the existence of a *central observer* that, based on the syndrome, can diagnose the state of all the units. In this paper, we will use alternatively the word node for unit, and network for system.

Early system-level diagnosis algorithms assumed that all the tests had to be decided in advance. 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* [5].

Early system-level diagnosis algorithms assumed the existence of the previously mentioned central observer. This situation was changed by Kuhl and Reddy [6, 7], who introduced *distributed* system-level diagnosis, in which fault-free nodes reliably receive test results through their neighbors, and each node independently performs consistent diagnosis. Important distributed system-level diagnosis algorithms include [8] and [9]

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 those 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.

The Adaptive Distributed System-level Diagnosis algorithm, *Adaptive DSD*, was introduced by Bianchini and Buskens [10, 11]. Adaptive DSD is at the same time distributed and adaptive. Each fault-free node is required to perform the minimal number of tests per testing interval, i.e., one test, to 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. Later, the Hierarchical Adaptive Distributed System-Level Diagnosis (Hi-ADSD) algorithm [2] was introduced and implemented with SNMP. By using a testing assignment that is initially a hypercube, Hi-ADSD has diagnosis latency of $\log^2 N$ in the worst case.

2.1 Algorithms for Diagnosis in Networks of General Topology

Up to this point, we reviewed system-level diagnosis algorithms for SNMP-based LAN diagnosis, in which the network is assumed to be fully connected, e.g., an Ethernet or a network based on switches. From this point on we review algorithms for diagnosis in non-broadcast networks, which can be applied for point-to-point network fault management. We introduce our new algorithm in the next section.

In [12] Bagchi and Hakimi introduced an algorithm for system-level diagnosis in networks of general topology. Initially each fault-free node knows only about its own state, and of its physical neighbors. Fault-free processors form a tree-based testing graph. Diagnostic messages are sent along the tree. The number of messages required by this algorithm to achieve diagnosis is shown to be optimum. Unfortunately the algorithm is not executed *on-line*, i.e., no processor can become faulty or be repaired during the execution of the algorithm. This characteristic rules out the application of the algorithm for WAN fault diagnosis.

In [13, 14] Bianchini *et.al.* introduced and evaluated through simulation the Adapt algorithm. The Adapt algorithm can be executed on-line: when a given node becomes faulty, a new phase begins in which other nodes reconnect the testing graph. The underlying testing assignment of Adapt is a minimally strongly connected digraph over the physical network. To build the testing graph, Adapt employs a distributed procedure that requires massive amounts of large diagnostic messages to be exchanged among the nodes.

Recently Rangarajan *et.al.* [15] introduced another algorithm for system-level diagnosis for networks of arbitrary topology that can be executed on-line. The algorithm, which we call here *RDZ*, for the author's initials, builds a testing graph that guarantees the optimal number of tests, i.e., each node has one tester. Furthermore it presents the best possible diagnosis latency by using a parallel dissemination strategy. Whenever a node detects an event, it sends diagnostic information to all its neighbors, which in turn send it to all its neighbors, and so on.

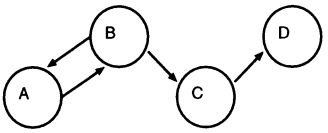

Figure 1 A *jellyfish* fault configuration.

Although the RDZ algorithm presents the best possible diagnosis latency, and the best possible number of testers per node, it does not diagnose link faults and also a node fault configuration which the authors call *jellyfish faulty node configuration*. In this fault configuration, between two connected components there is a set of nodes such that part of

those nodes test each other in a cyclic fashion, and other tests emanate from the cycle. If all nodes in the jellyfish become faulty simultaneously, nodes in the connected components won't diagnose that situation. It should be noted that a jellyfish may involve from one to an arbitrary number of nodes.

Consider figure 1. All nodes form a jellyfish, in which there is a cycle (node A and node B) and tests emanating from the cycle (from node B to node C to node D). If both nodes A and B become faulty, nodes C and D won't be able to diagnose the fault. The same is true if nodes A, B, and C become faulty, i.e., node D doesn't detect the event. The RDZ algorithm cannot be applied for network fault monitoring, for it is unacceptable to have an arbitrarily large portion of the network to become faulty in an undetected fashion.

3 A NEW ALGORITHM FOR DIAGNOSIS ON NON-BROADCAST NETWORKS

In this section, we introduce a new algorithm that diagnoses link time-outs, and node reachability, using the minimum number of tests, i.e. one per link, and also presenting the optimal latency. Before introducing the algorithm, consider figure 2. In fault situation A the node is fault-free, but all links leading to that node are faulty, in fault situation B, the node itself is faulty. From test results it would be impossible for any other node in the system to determine which is the actual situation. Our algorithm is based on this fact: a link may *time-out* to a test, and if all links to a given node have timed-out, then the node is *unreachable*. Thus links may be in one of two states *fault-free, timed-out* and nodes may be *fault-free* or *unreachable*. This approach to fault diagnosis on wide-area networks is closer to reality, for links are usually made up of not only wires but may also involve a number of network devices, hubs and gateways.

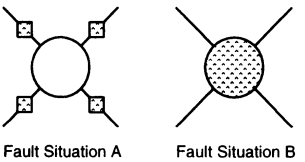

Figure 2 Ambiguous fault configurations.

To keep the number of tests minimum, there is one tester per link. As a link always connects two nodes, and nodes have unique identifiers, the node with the highest identifier tests the link at each testing interval. If the link *times-out*, i.e., the neighbor doesn't reply to the test, and in the past testing interval it did, then there is a new *fault event.* Analogously, if the link has timed-out in the past testing interval, and it does carry a reply this time, then there is a *repair event.*

The algorithm employs a *two-way test.* This guarantees that the *jellyfish* fault configuration is always detected, even keeping the minimum number of tests. When node A is testing the link to node B, node A gets the local time at B, and stores that result on B. In this way, not only node A knows about the state of B, but also node B can monitor the tester activity. If a threshold is decided for the maximum interval between link tests, then a node can time-out the tester whenever the threshold is exceeded. When a node detects

a link time-out or a tester fault, it starts or continues testing the link until it ceases to time out, such that, when the link recovers again, only the node of highest identifier tests the link.

Each node keeps a state counter for each link in the system, which is initially zero, and is incremented at each new event information received for that link. This permits a node to identify redundant messages. After a new event is discovered, each node propagates event information to all neighbors. This parallel dissemination strategy is the same employed by the RDZ algorithm. Besides the nodes identifier, and the status counter, each diagnostic message carry information about which nodes have processed the message. In this way, the number of redundant messages is reduced, and messages do not cycle in the network. After receiving a message, each node appends its own identifier to the list of nodes that has processed the message. Furthermore it appends the identifiers of the neighbors to which the message was already sent. For a full discussion and evaluation of this approach please refer to [15]. It should be clear that, as messages are short, the impact of this strategy on network performance is small. In section 5 we evaluate the percentage of link bandwidth required to run the algorithm.

After a node receives information about a link event, it runs an algorithm (like the breadth-first tree) to compute the system connectivity, thus discovering which portions have become reachable or unreachable.

The data structures of the algorithm are thus:

- A Link table indexed by link identifier, containing a status counter for the link, and the last time the link was tested. The counter is initially zero, and an even value indicates a fault-free state; The last-test-time is updated only on nodes connected to the link and such that the node doesn't test the link, but its neighbor;
- A Link-Events table, containing at each entry the link identifier, the state counter of the link, and a list of nodes that have already processed the message as seen by the sending node.

The algorithm in pseudo-code is:

```
BEGIN
 /* at node i */
 DO FOREVER
  FOR each link i-j, that connects node i to node j
    IF (i > j) OR (node j is faulty)
    THEN test link i-j; /* get local time at node j */
         IF link i-j is fault-free
         THEN set last-time-tested on j;
         IF there is a new event
         THEN add event to new-event table;
    ELSE /* check link tested by neighbors */
    IF last-time-tested > testing interval threshold
    THEN add event to Link-Events table;

  FOR each entry in Link-Events table
       IF entry carries new information
       THEN update link counters;
```

```
            FOR each neighbor k of node i
                 IF k has not received the message
                 THEN set event information to k's new-event table;
  compute node reachability;
  SLEEP(testing interval)
END;
```

3.1 An Example Execution

Consider the example system in figure 3. Initially all links and nodes are fault-free. Each node starts testing links as depicted by arrows, and exchange test information with neighbors. Eventually each node receives information about all links.

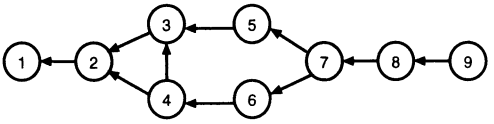

Figure 3 The testing assignment on an example non-broadcast network.

Now consider the first event depicted in figure 4, in which link 3-5 is faulty and times-out. This time-out will be detected by node 5, and immediately disseminated to node 7. This in turn will disseminate to node 8 (and from there to node 9), and node 6. Node 6 disseminates information to node 4, and from there to node 3, node 2 and node 1. Node 2 disseminates the information to node 3. Now, if node 3 timed-out out the tester (link 3-5) before the information arrives from node 2, then node 3 will also disseminate information on the time-out. If a node, say node 2, receives two diagnostic messages about the same event it will only disseminate the first of them, because the second is recognized as old information. Thus, the highest number of messages per event per link is two. After all nodes receive and process diagnostic messages, they run an algorithm to compute system connectivity, and conclude that all nodes are still connected.

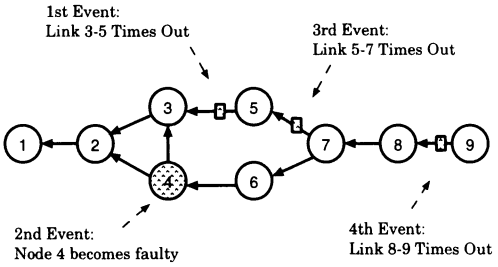

Figure 4 A series of events occur in the network.

In the second event depicted in figure 4, node 4 became faulty. Node 6 detects a time-out on link 6-4, and after the testing threshold expires, node 2 and node 3 detect time-outs on links 4-2 and 4-3 respectively. The system now is divided into two connected components, one consisting of node 1, node 2 and node 3, the other consisting of node 5, node 6, node

7, node 8 and node 9. As on each component a node detects and disseminates the event, diagnostic information will eventually reach every fault-free node in the system.

Now consider the third event, link 7-5 becomes faulty and times-out. The resulting system has 3 connected components, the first consisting of node 1, node 2, and node 3; the second of node 5 alone; and the third of node 6, node 7, node 8, and node 9. In the first component not one node detects the event, because it is already disconnected from the rest of the system. In the second component, node 5 eventually times out on the test of link 7-5 and realizes it is disconnected from the system, i.e., every other node is unreachable. At the third component, node 7 initially detects link 7-5 time-out and the event is disseminated to the other nodes in the component.

If still another link, 9-8, becomes faulty and times-out, node 9 detects the event and recognizes it is disconnected from the system. Node 8 times-out on the testing threshold of link 9-8, and disseminates event information to node 7 and node 6. The other nodes in the network are already in disconnected components.

After these events, when faults are repaired, nodes testing corresponding links will detect the events, and disseminate the information to other nodes in their connected components. Eventually the whole system becomes a unique connected component, and every node receive diagnostic information about all links.

Correctness

Here we give an informal discussion of the correctness of the algorithm. Consider a connected component of the system, made up of fault-free nodes and such that between any pair of those nodes there is a fault-free path. The neighborhood of the component is defined as the set of links that timed-out in the previous testing-round. Clearly, any new event in the component or in its neighborhood is detected by nodes in the component. This is assured by the testing strategy, in which there is a two-way test on each link from any node of the component. Now consider that one event has occurred. If a fault-free node or link has become faulty, then one node in the component will detect the fault, and forward it to other neighbors. As each node forwards new information to all neighbors, information will eventually reach all nodes in the component. If the fault breaks the component in two, then nodes on both components will detect a link time-out, and disseminate the information on their respective components. Now consider a repair event: if a test succeeds on a link that had been timing-out, the two nodes (tested and tester) exchange diagnostic information, and disseminate this information to their neighbors. Thus event information is always disseminated to every fault-free node in the component.

Event counters guarantee that old information is recognized as such. Furthermore, those links that have odd event-counters are timed-out links and those that have even-counters are fault-free links. This is guaranteed because a counter is only incremented when a new event happens, from timed-out to fault-free or opposite. As the counter is initially zero, for a fault-free initial status, and when it times-out it is increased to 1 and so on, an even value will always indicate a fault-free state, and an odd value a faulty state.

4 SNMP-BASED IMPLEMENTATION

In this section we present an implementation of the algorithm for non-broadcast network fault management based on SNMP [17, 18, 19]. Each node running the algorithm keeps

two tables. The first table keeps information about each link in the network: its identifier, the state counter, and the time it was tested. The time field is only used by nodes that test a link to implement the two-way testing strategy. We give below the corresponding ASN.1 table.

```
LinkState OBJECT-TYPE
    SYNTAX  SEQUENCE OF LinkStateEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
        "This is an array that contains link status information."
    ::= { diagnosis 1 }

linkStateEntry OBJECT-TYPE
    SYNTAX  LinkStateEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
            "Each entry of linkState shows if a link is timing-out
             or fault-free, according to the status counter"
    INDEX   { linkID }
    ::= { LinkState 1 }

LinkStateEntry ::=
    SEQUENCE {
        linkID          DisplayString,
        StatusCounter   Counter,
        TestedTime      TimeTicks  }
```

The second table, LinkEvents, is a dynamic table, in which event information is added by the local agent and its neighbors. After each testing interval, all entries in the table are processed. Each entry contains the identifier of the link that suffered the event, the timestamp for that event, and a string containing the identifiers of all the nodes that have already processed the message. The ASN.1 table is given below.

```
LinkEvents OBJECT-TYPE
    SYNTAX  SEQUENCE OF linkEventsEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
        "This is a dynamic table to which information
         about new link events are added."
    ::= { diagnosis 2 }

linkEventsEntry OBJECT-TYPE
    SYNTAX  LinkEventsEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
```

```
            "Each entry of linkEvents carries the link identifier,
             the status counter for the new event, and a sequence
             of identifiers of nodes that have processed the event"
        INDEX   { linkID }
        ::= { LinkEvents 1 }

    LinkEventsEntry ::=
        SEQUENCE {
            LinkID          DisplayString,
            StatusCounter   Counter,
            Path            DisplayString  }
```

In our implementation nodes set neighbors tables, and thus security measures must be taken, specifically assignment of restricted access permission. It should be clear that from the LinkState table that the complete network configuration is available to each node, which can calculate the system connectivity at any time. Works on generating network configuration information automatically have been reported [16], and can be employed to build the LinkState table.

5 IMPACT ON NETWORK PERFORMANCE

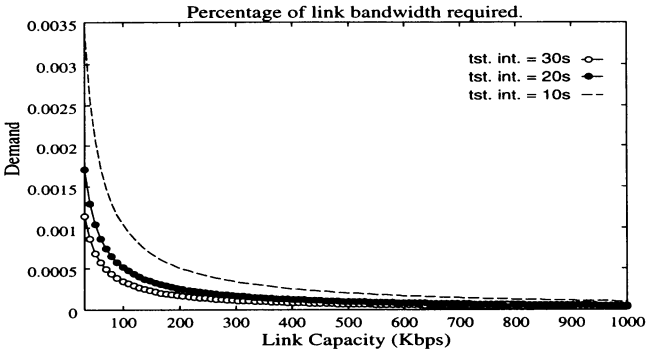

Figure 5 Amount of link bandwidth required to run diagnosis.

At each testing interval, each link carries one message from the tester to the neighbor. Furthermore, for any new event in the network, each link will carry usually one, and at most *two* messages about the event. The reason is that after updating the state counter, a node does not forward any other message that contains known information. The link will carry two messages only if both nodes send information at the same time. Thus, the total number of messages per event required by the algorithm is at most $2 * L$, where L is the number of links.

The graph in figure 5 shows the impact of the algorithm on network performance, by showing the percentage of link bandwidth required by diagnostic messages. The graph shows links of different capacities, and results are shown for different testing intervals, of

10 seconds, 20 seconds, and 30 seconds. We consider a fault rate λ of 0.001. The size of SNMP messages assumed is 128 bytes. Results show the percentage of bandwidth required is always less than 0.1%, on links from 28.8Kbps to 1Mbps.

6 CONCLUSION

In this paper we presented a new distributed algorithm for system-level diagnosis on non-broadcast networks. The purpose of the algorithm is to allow each node to independently detect which portions of the network are faulty or unreachable. We show that in some cases it is impossible to distinguish between the two cases.

A node running the algorithm executes link tests at a testing interval. The algorithm employs the minimum number of tests, i.e., one per link. Of the two nodes connected by a link, the one with highest identifier is the link tester. We assume nodes have local memory, and tests are built in such a way that both ends of a link detect a link time-out in case of link or one node failure.

Upon detecting a new event, diagnostic information is disseminated in parallel, and the algorithm has the minimum diagnosis latency, i.e., proportional to the diameter of the network. Mechanisms are included to reduce the amount of redundant messages. As each message is small, containing information about one event, and any link carries at most two messages, the impact of the algorithm on network performance is small. A MIB and SNMP implementation were presented.

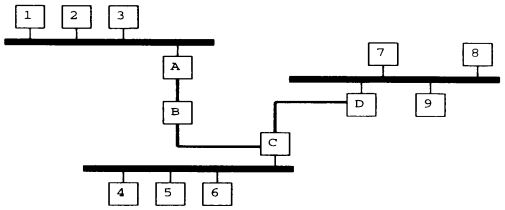

Figure 6 A small internet.

As future work we discuss here an integrated approach for internet fault monitoring. This approach can be achieved by running specific algorithms for diagnosis on broadcast networks (LAN's), like Hi-ADSD, together with the algorithm introduced in this paper. Consider the small internet in figure 6. Nodes with identifiers from 1 to 9 are connected to broadcast networks. Node A, node B, and node D have a link to a broadcast network, and to a non-broadcast network. Node B is takes part only in the non-broadcast network. For the two algorithms to run cooperatively, it is sufficient that nodes only on a broadcast network run an algorithm for diagnosis on the LAN to which it belongs; nodes not on a non-broadcast network run the algorithm for diagnosis on that network; nodes that are on a broadcast network, but also have a link to another network must run both a LAN diagnosis algorithm, and a WAN diagnosis algorithm. This means these nodes execute tests according to the two algorithms, and also carry the necessary data structures to hold information about the entire system. In this way, a truly fault-tolerant network fault management system can be deployed, in which any fault-free node can diagnose the whole system.

REFERENCES

[1] E.P. Duarte Jr., and T. Nanya, "An SNMP-based Implementation of The Adaptive DSD Algorithm for LAN Fault Management," *Proc. IEEE/IFIP NOMS'96*, pp. 530-539, Kyoto, April 1996.

[2] E.P. Duarte Jr., and T. Nanya, "Hierarchical Distributed System-Level Diagnosis Applied for SNMP-based Network Fault Management", *Proc. IEEE* 16^{th} *Symp. Reliable Distributed Systems*, Niagara, September 1996.

[3] 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.

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

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

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

[7] J.G. Kuhl, and S.M. Reddy, "Fault-Diagnosis in Fully Distributed Systems," *Proc.* 11^{th} *Fault Tolerant Computing Symp*, pp. 100-105, 1981.

[8] 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.

[9] R.P. Bianchini, K. Goodwin, and D.S. Nydick, "Practical Application and Implementation of System-Level Diagnosis Theory," *Proc.* 20^{th} *Fault Tolerant Computing Symp*, pp. 332-339, 1990.

[10] R.P. Bianchini, and R. Buskens, "An Adaptive Distributed System-Level Diagnosis Algorithm and Its Implementation," *Proc.* 21^{st} *Fault Tolerant Computing Symp*, pp. 222-229 , 1991.

[11] 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.

[12] A. Bagchi, and S.L. Hakimi, "An Optimal Algorithm for Distributed System-Level Diagnosis," *Proc.* 21^{st} *Fault Tolerant Computing Symp.*, June, 1991.

[13] M. Stahl, R. Buskens, and R. Bianchini, "On-Line Diagnosis on General Topology Networks," *Proc. Workshop Fault-Tolerant Parallel and Distributed Systems*, July 1992.

[14] M. Stahl, R. Buskens, and R. Bianchini, "Simulation of the Adapt On-Line Diagnosis Algorithm for General Topology Networks," *Proc. IEEE* 11^{th} *Symp. Reliable Distributed Systems*, October 1992.

[15] 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.

[16] G.Mansfield, M.Ouchi, K.Jayanthi, Y.Kimura, K.Ohta, Y.Nemoto, "Techniques for automated Network Map Generation using SNMP", Proc. of INFOCOM'96, pp.473-480, March 1996.

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

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

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

BIOGRAPHY

Elias Procópio Duarte, Jr. is a PhD student in Computer Science at Tokyo Institute of Technology, Tokyo, Japan. He is also an Assistant Professor at the Department of Informatics of Federal University of Paraná, Brazil. He has an MSc degree in Telecommunications from the Polytechnical University of Madrid, Spain, 1991. He also has an MSc degree in Computer Science from Federal University of Minas Gerais where he received his BS in Computer Science in 1988. Main research interests include distributed systems and computer networks, their dependability, management, performance evaluation, and algorithms. He is a student member of the IEEE and the ACM.

Glenn Mansfield obtained his Master's degree in 1977 from Indian Institute of Technology, Kharagpore, India in the field of Nuclear and Particle Physics followed by his Masters in Physical Engineering in 1979 from Indian Institute of Science, Bangalore, India. After working with Tata Consultancy Services, Bombay, India as a senior systems analyst for five years, he obtained his Ph.D. specializing in Logic programming, from Tohoku University, Japan. He has worked as a research associate in the computer center of Tohoku University for a period of 3 years and is currently chief scientist at Sendai Foundation for Applied Information Sciences, Japan. His areas of interest include expert systems, logic programming, computer networks and their management, use of the Internet for education. He is a member of the Internet Society, the ACM, the IEEE and the IEEE Communications Society.

Takashi Nanya is a professor in the Research Center for Advanced Science & Technology at the University of Tokyo, and also a professor in the Departmnent of Electrical Engineering at Tokyo Institute of Technology, Tokyo, Japan. His research interests include fault-tolerant computing, computer architecture, design automation and asynchronous computing. He was a visiting research fellow at Oakland University, Michigan, in 1982, and at Stanford University, California, in 1986-87. He received his B.E. and M.E. degrees in mathematical engineering and infomation physics from the University of Tokyo in 1969 and 1971, respectively, and his Dr.Eng. degree in electrical engineering from Tokyo Institute of Technology in 1978. He is a member of the IEEE, ACM, IEICE, and the Information Processing Society of Japan.

Shoichi Noguchi is the director of the Sendai Foundation for Applied Information Sciences, and also a professor at Nihon University, Japan. He received his B.E., M.E. and D.E. degrees in Electrical Communication Engineering from Tohoku University in 1954, 1956 and 1960 respectively. He was a professor at Tohoku University until 1993. He is active in the fields of information science theory, computer network fundamentals, parallel processing, computer network architectures and knowledge engineering fundamentals. He is the president of the Information Processing Society of Japan since 1995.