Simulation of a Distributed Connectivity Algorithm for General Topology Networks

Elias Procópio Duarte Jr.
Andréa Weber

Federal University of Paraná, Dept. Informatics
P.O. Box 19081 Curitiba 81531-990 PR Brazil
e-mail: {elias,andrea}@inf.ufpr.br

Abstract

The Distributed Network Connectivity *algorithm allows every node in a general topology network to determine to which portions of the network it is connected, and which portions are unreachable. The algorithm consists of three phases: test, dissemination, and connectivity computation. During the testing phase each fault-free node tests all its neighbors at each testing interval. Upon the detection of a new fault event, the tester starts the dissemination phase, in which a reliable broadcast algorithm is employed to inform the other connected nodes about the event. At any time, any working node may run the third phase, in which a graph connectivity algorithm shows the complete network connectivity. Extensive simulation results are presented, considering various topologies such as the hypercube and random graphs. Results are compared to those of other algorithms.*

Keywords: Distributed Diagnosis, Network Connectivity, Reliable Broadcast, Distributed Algorithms.

1 Introduction

Organizations and individuals increasingly depend on the correct behavior of network systems. Given the topology, it is important to determine at any instant of time which portions of the network are connected and which portions are unreachable. In this work we present a new algorithm for computing Distributed Network Connectivity (DNC). This algorithm is based on previously published results in System-Level Diagnosis of General Topology Networks [4].

The first model of a diagnosable system is known as the PMC model [2]. This model considers a fully connected system which can be represented by a complete graph. The vertices represent system units, also called nodes in this work, which are capable of executing tests on any other unit. An edge represents a communication channel, also called link, between two units. In this model, a node may assume one of two states: *faulty or fault-free*, and links are always fault-free.

The main assumption of the PMC model is that a fault-free node is reliable in the sense that whenever such a node executes a test on another node, it correctly determines the state of the tested node. Furthermore the tester is also capable of correctly reporting test outcomes. Based on the test reports sent by all nodes, a central monitor performs the diagnosis itself, determining which nodes are faulty and which are fault-free.

Distributed diagnosis eliminates the need for a central observer [6]. The nodes that execute the tests also perform the diagnosis. Adaptive diagnosis is another approach that allows testers to determine which tests should be executed in the next testing interval based on the results of the tests executed in the previous interval [7]. A number of algorithms for fully connected networks are at the same time distributed and adaptive [5].

Local Area Networks (LAN's) are usually modeled as fully connected systems, while Wide Area Networks (WAN's) have general topologies. In a general topology network there is not necessarily a communication channel between every two nodes. Both nodes and links may be either faulty or fault-free at a given instant of time. In this way, a fault may partition the network. Consider a pair of nodes connected by a link. If the tests executed on one of those nodes by the other results in a time-out, it is impossible to determine whether the tested node or the link is faulty. In this way, faults in a general topology network are said to be ambiguous [3].

In [6] Bagchi and Hakimi introduced an algorithm for system-level diagnosis of networks of general topology. The algorithm is executed *off-line*. In [7] 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 algorithm employs a distributed procedure that requires massive amounts of large diagnostic messages.

Rangarajan *et.al.* [8] introduced the RDZ algorithm for system-level diagnosis for networks of arbitrary topology that can be executed on-line. The algorithm 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 the information to all its neighbors, and so on. Although the RDZ algorithm presents the best possible diagnosis latency, and the best possible number of testers per node, it does not diagnose a fault configuration called by the authors *jellyfish faulty node configuration*.

The Distributed Network Connectivity (DNC) algorithm introduced in this paper is based on the Non-Broadcast Network Diagnosis (NBND) algorithm [3, 4, 9]. The algorithm is structured in three phases: test, dissemination and diagnosis. Nodes test adjacent links every testing interval. Upon the detection of a new event, information is disseminated to all nodes through a distributed breadth-first tree, instead of using flooding such as in NBND. Based on that information each fault-free node may compute the network connectivity. The algorithm allows dynamic events, i.e. during the dissemination phase new events may occur and dissemination remains guaranteed. This is done by means of a reliable broadcast algorithm [10] for event dissemination. The proposed strategy is compared with two other approaches: dissemination based on flooding [12] and a sequential strategy based on a distributed Chinese Agent [13]. The strategy employed by DNC is nearly always better than the other two.

The rest of the paper is organized as follows. In section 2 the DNC algorithm is specified. In section 3 simulation results are presented, comparing the cost of the dissemination phase in three algorithms: DNC, Chinese Agent and Flooding. Section 4 concludes the paper.

2 The DNC Algorithm

In this section, we introduce a new distributed connectivity algorithm for general topology networks. The algorithm has three phases: testing, dissemination of new event information, and local connectivity computation. The algorithm considers a synchronous system and crash faults. Both nodes and links may be either faulty or fault-free. However, if there is no reply to a given test over a link, the tester is not able to determine whether it is the tested link or the tested node connected by that link that is faulty. If there is no reply to tests executed over all links to a given node, then the node is *unreachable*. Thus links may be in one of two states *fault-free* or *unresponsive* and nodes may be *fault-free* or *unreachable*.

Every link is tested every testing interval. There is one tester per link. The algorithm thus requires the minimum number of tests for any network topology. The algorithm employs a *token-based testing strategy* [9]. The two nodes connected by a link execute tests over that link at alternating intervals. The tests employed are also said to be *two-way tests*, in the sense that when one node executes a test over a link, not only the tester determines the state of the tested node, but also the tested node determines the state of the tester.

Each node keeps a timestamp which is 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. A redundant message when it is not the firt one about an event. After a new event is discovered, the tester propagates event information to its neighbors. Every node keeps the complete network topology. The parallel dissemination strategy employed is based on a distributed breadth-first tree. Each diagnostic message carries the following information (1) the tester identifier (2) the tested node identifier (3) the timestamp for the tested link. Each node running the algorithm keeps a *link table* indexed by link identifier, containing the timestamp for the link. An even timestamp indicates a fault-free link; an odd timestamp indicates a faulty link.

Whenever a node is a leaf, after receiving the message from its parent it sends back an acknowledgement called hence forth *ack*. After receiving acks from all its children, a node sends an acknowledgement to its parent in the tree. The dissemination completes when the root receives acknowledgements from all its children.

When link a–b becomes faulty, the dissemination proceeds in the following way. Consider that node a is the first to detect the event. Node a starts a dissemination tree, informing the event on link a–b. After this dissemination message reaches node b, this node will determine that link b–a is faulty. Node b gives up the previous dissemination, appending the previous message to the new one, and takes its role in the second dissemination tree. Whenever a node that has a pending dissemination receives a message that contains information about the pending dissemination plus new information, it gives up the first dissemination, and takes its role in the second dissemination.

At any given time a fault-free node running the algorithm may compute the local network connectivity after removing the links that are in the *unresponsive* state, i.e. have an odd timestamp, from the network topology.

2.1 An Example Execution

Consider the example topology in figure 1. This subsection contains a description of the execution of the algorithm, considering a fault event on one link, and subsequent dissemination of new event information.

A fault event on link 1–3 is considered to happen at instant of time t_0. In the next testing interval after the event occurs, consider that node 1 is the tester, and thus detects the fault. Then node 1 starts the dissemination phase in order to communicate the event information to all other reachable nodes. It does so by building a breadth-first traversal tree rooted at itself, yet considering the fault event. The tree is illustrated in figure 2.

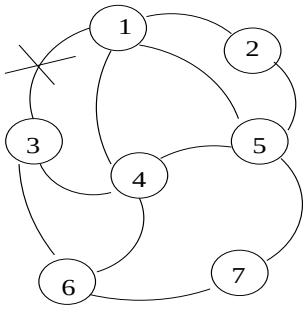

Figure 1: An example topology. Link 1–3 becomes faulty.

The dissemination tree has node 2, node 4 and node 5 at the second level. At the third level, node 4 has two children: node 3 and node 6; node 5 has one child: node 7. Its one time unit for a node to process a dissemination message and for this message to be received by this node's children. So, one time unit after the sending of the dissemination message by node 1 it reaches nodes 2, 4 and 5 simultaneously.

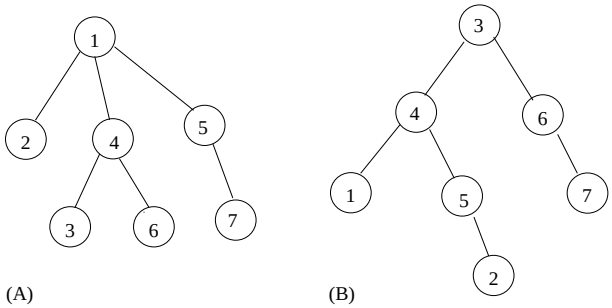

Figure 2: The dissemination trees built by node 1 (A) and node 3 (B).

In this example dissemination, node 2 is a leaf in the dissemination tree, so the message it receives is acknowledged in the next time unit. Nodes 4 and 5 disseminate the information for its children the next time unit, so that when the acknowledgement of node 2 reaches node 1, dissemination messages reach nodes 3, 6 and 7 at the bottom of the tree. Nodes 6 and 7 are ordinary leaves, thus they acknowledge the receipt of the message to nodes 4 and 5, respectively. Node 3, although, once informed about the fault on link 1–3 tests its adjacent links, and realizes that link 3–1 is "also" faulty. Thus, this information must be disseminated. For this to be done, node 3 builds another dissemination tree, as is illustrated in figure 2.

At this point there are two simultaneous dissemination trees. The first dissemination tree was started by node 1, and disseminates information about the fault event on link

1–3, which is called first dissemination message. The first dissemination tree is gradually being abandoned, as the second dissemination tree, which was started by node 3, has complete information about both events (link 1–3 is faulty, link 3–1 is faulty, called the second dissemination message), takes its place.

As the simulation continues, at the same time the acknowledgement messages from nodes 6 and 7 reach nodes 4 and 5 of the first dissemination tree, new information messages are sent by node 3 in the second dissemination tree. Nodes 4 and 6 receive this messages. Node 4 sends the dissemination message to nodes 1 and 5. When node 6 receives the second dissemination message, it has already completed its task in the first dissemination tree. So, it sends the second dissemination message to node 7, its child in the second tree. This happens simultaneously with the sending of the second dissemination message to nodes 1 and 5 by node 4.

At the third level of the second dissemination tree, nodes 1 and 7 are leaves which acknowledge the received messages to its parent nodes. At the same time these acknowledgements move up one level in the tree, node 5 sends the second message to node 2, its child in the second dissemination tree, which sends an acknowledgement after its receipt. At the same time node 5 receives the acknowledgement message from node 2, the root of the tree receives the acknowledgement message from node 6. The next time unit, node 5 acknowledges the message from node 2, and node 4 does the same after receiving the acknowledgement from node 5. After that the root at node 3 receives the last acknowledgement and considers the dissemination as finished. Once begun at the first root at node 1, the whole dissemination is completed in 8 time units.

3 Experimental Results

In this section, experimental results are presented and a comparison is made with results of two other approaches: a parallel algorithm based on flooding [12] and a sequential algorithm based on the Chinese Agent [13]. Various network topologies were considered: initially results were obtained for the example graph shown in figure 1. Next, results were obtained for hypercubes with 16, 64 and 128 nodes; then the algorithm was simulated on a 50-node random graph, the $D_{1,2}$ graph with 9 nodes, and a subset of the topology of the Brazilian National Research Network: RNP (*Rede Nacional de Pesquisa*).

Simulations were built using SMPL *SiMulation Programming Language* [11], a discrete event simulation library. For each simulation experiment, one communication link fault event was scheduled. Testing intervals of 30 time units, and dissemination intervals of 1 time unit between nodes connected by one link were employed.

3.1 An Example Topology

For the 7-node example topology shown in figure 1, results of the algorithm execution of a fault event on link 1–3 such as described in the previous section are shown in table 1.

	Total of Messages	Redundant Messages	Latency
DNC	12	0	9
Flooding	28	16	7
Chinese Agent	7	1	7

Table 1: Results for an example topology.

3.2 Graph $D_{1,2}$

For the $D_{1,2}$ graph topology [8], a fault event was scheduled for link 6–8. Node 6 starts the dissemination building a four-level breadth-first tree rooted at itself. In this way, it takes 2 time units for the dissemination to reach node 8. At that time, node 8 builds another dissemination tree and begins disseminating information about the event on link 6–8 plus the information of the fault event on link 8–6. For this dissemination, another four-level tree is built. Thus, it takes three time units for the information to reach the leaves of that tree, and three time units more for confirmations to reach node 8 at the root. Comparative results with the other mentioned algorithms are shown in table 2.

	Total of Messages	Redundant Messages	Latency
DNC	16	0	9
Flooding	52	36	7
Chinese Agent	13	5	13

Table 2: Results for the 9-node $D_{1,2}$.

3.3 Hypercubes

Results were obtained and are described below for hypercubes with 16, 64, and 128 nodes. For the 16-node hypercube a fault event on link 5–7 was scheduled. Node 5 detects the event and begins the dissemination. A five-level tree is built, with node 7 at the fourth level. Three time units after the beginning, node 7 is informed of the event on link 5–7 and starts the dissemination of the event on link 7–5. This dissemination takes another five-level tree, so that four time units after node 7 initiates the dissemination, the information reaches the bottom of the tree. Four time units after that the last acknowledgement message reaches the top. Results for the other approaches are shown in table 3.

	Total of Messages	Redundant Messages	Latency
DNC (16 nodes)	30	0	12
Flooding (16 nodes)	94	64	9
Chinese Agent (16 nodes)	28	14	28
DNC (64 nodes)	126	0	24
Flooding (64 nodes)	478	352	15
Chinese Agent (64 nodes)	156	93	156
DNC (128 nodes)	254	0	36
Flooding (128 nodes)	990	736	23
Chinese Agent (128 nodes)	348	221	348

Table 3: Results for the 16, 64 and 128 nodes hypercubes.

Considering the 64-node hypercube with a fault event detected on link 5–7 by node 7, the dissemination was performed with 126 messages in 24 time units. Considering the 128-node hypercube topology, with a fault event at link 13–21, the dissemination begins at node 13 and takes 254 messages and 36 time units to complete. Comparative results are shown in table 3.

3.4 A Random Graph

For this simulation a random graph was built with a probability of ten percent that a link exists between any pair of nodes. The number of nodes is equal to 50. The graph, nevertheless, consists of one connected component.

For such a graph, an event on link 30–31 was scheduled and the dissemination began on node 31. This node employed a five-level breadth-first traversal tree, with node 30 at the fourth level. Thus, three time units after the beginning, the dissemination reaches node 30. At this time, node 30 builds another five-level tree and begins disseminating the information of the fault event on link 30–31 together with the received information of the event on link 31–30. Four time units after this dissemination starts, it reaches the leave nodes, and other four time units after that are required for the acknowledgement messages to reach node 30 on the root. Comparative results are shown in table 4.

	Total of Messages	Redundant Messages	Latency
DNC	143	0	14
Flooding	418	320	8
Chinese Agent	109	60	109

Table 4: Results for a random graph.

3.5 A Subset of The Brazilian *RNP*

The *RNP* topology employed for the simulations was that available at http://www.rnp.br in February, 2001. That was a 28-node topology, and a fault event was scheduled for the Sao Paulo–Brasília link.

Once the fault is detected, the Brasilia node starts the dissemination phase employing a four-level breadth-first traversal tree. Two levels below on the tree is the Sao Paulo node, that receives the information two time units after the beginning of the dissemination. Once that happens, the Sao Paulo node starts another dissemination for informing the Sao Paulo–Brasilia fault event and does so by building another four-level breadth-first tree rooted at itself. Three time units after the second dissemination is started it reaches the leaves. Three time units after that, the acknowledgement messages arrive the top of the tree, completing the dissemination. Comparative results are shown in table 5.

	Total of Messages	Redundant Messages	Latency
DNC	54	0	9
Flooding	94	40	6
Chinese Agent	55	27	55

Table 5: Results for a subset of the Brazilian *RNP*.

4 Conclusion

The Distributed Network Connectivity algorithm described in this paper allows any node of a general topology network to compute to which portions of the network it is connected, and which portions are disconnected. Links are tested continually, at a testing interval, disseminating fault event information through a distributed breadth-first tree.

A link fault, and subsequent dissemination was simulated for a number of different topologies: an example network; the $D_{1,2}$ graph; 16, 64, and 128-node hypercubes; a random graph, and a subset of the Brazilian RNP. Results were compared with those of other two algorithms: Flooding and the Chinese Agent.

Flooding always completes the dissemination in the shortest possible interval of time. However it always employs a larger number of messages, with many redundant messages. The number of messages employed by the Chinese Agent is always much lower than the number of messages employed by Flooding. The Chinese Agent presents the lowest

impact on the network as at most one message is being disseminated at a given instant of time. The number of messages employed by DNC is similar to the number employed by the Chinese Agent, while DNC's latency is similar to Flooding's. So DNC has good latency at a low impact on the network. Beyond that, both Flooding and the Chinese Agent employ redundant messages, while DNC does not.

Future work includes developing a new strategy that allows the algorithm to work in the presence of multiple concurrent fault and repair events. A practical tool based on the Internet management protocol, SNMP (Simple Network Management Protocol) is currently being developed. This tool should present a Web-based interface that allows any user to obtain network connectivity information from any network node. Effective testing strategies for WAN links must be developed, possibly using artificial intelligence tools.

References

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

[2] 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.

[3] E.P. Duarte Jr., G. Mansfield, T. Nanya, and S. Noguchi, "Non-Broadcast Network Fault Monitoring Based on System-Level Diagnosis," *Proc. IEEE/IFIP IM'97*, pp.597-609, San Diego, May 1997.

[4] E.P. Duarte Jr., "Um algoritmo para Diagnóstico de Redes de Topologia Arbitrária," *in portuguese*, In Proceedings of the *1st SBC Workshop on and Fault Tolerance*, SBC-WTF'1, pp. 50-55, Porto Alegre, Brazil, 1998.

[5] E. P. Duarte Jr., and T. Nanya, "A Hierarchical Adaptive Distributed System-Level Diagnosis Algorithm," *IEEE Transactions on Computers*, Vol. 47, pp. 34-45, No. 1, Jan 1998.

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

[7] 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.

[8] 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.

[9] J. I. Siqueira, E. Fabris, E. P. Duarte Jr., "A Token Based Testing Strategy for Non-Broadcast Network Diagnosis", *1st IEEE Latin American Test Workshop*, pp. 166-171, Rio de Janeiro, 2000.

[10] P. Jalote, *Fault Tolerance in Distributed Systems*, Prentice Hall, 1994.

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

[12] E.P. Duarte Jr., and G.O. Mattos, "Diagnóstico em Redes de Topologia Arbitrária: Um Algoritmo Baseado em Inundação de Mensagens", *in portuguese*, In Proceedings of the *2nd SBC Workshop on Test and Fault Tolerance*, SBC-WTF'2, pp. 82-87, Curitiba, Brazil, 2000.

[13] E.P. Duarte Jr., and J.M.A.P. Cestari, "O Agente Chines para Diagnstico de Redes de Topologia Arbitrária" *in portuguese*, In Proceedings of the *2nd SBC Workshop on Test and Fault Tolerance*, SBC-WTF'2, pp. 88-93, Curitiba, Brazil, 2000.