A Dynamic Distributed Diagnosis Algorithm for an Arbitrary Network Topology with Unreliable Nodes and Links

Pabitra Mohan Khilar¹, S. Mahapatra²
¹Dept. of CSE, NIT, Rourkela – 769 008. ²Dept. of E & ECE, IIT, Kharagpur – 721 302.
Email: ¹khilarpm@yahoo.com, ²sudipta@ece.iitkgp.ernet.in

Abstract

This paper presents a distributed network diagnosis (DND) algorithm for an arbitrary network topology where every node needs to record the status of every other node and links assuming the nodes and links are subjected to crash and value faults in a dynamic fault environment (the node’s or link’s status may change during execution of algorithm). The algorithm operates correctly in each connected component if the network is partitioned due to a set of faulty links or faulty nodes. The worst-case bounds for diagnostic latency is at most O(td) rounds where t is the number of dissemination trees and d is the diameter of the network. The proposed approach uses non-broadcasting method of message dissemination that has similar diagnostic latency with flooding [4] and similar message complexity with Chinese Agent [14] method of message dissemination respectively.

1. Introduction

The fault diagnosis in distributed systems such as mobile adhoc network, wireless sensor network is an important problem where every node keeps track of the status of other nodes or components [4]. The distributed diagnosis provides a cost effective software based fault tolerance solution by reducing the cost overhead and increasing the reliability and throughput of the system as compared to the traditional ways such as using N-Modular Redundancy or N-Version programming. The situations where deploying spares are not feasible, distributed diagnosis techniques are the attractive solutions.

The interconnection network for above distributed systems forms an arbitrary network topology where there is no direct connection between any two nodes. In this paper, we present a distributed diagnosis algorithm that allows every fault free node to obtain the diagnostic map showing the state of the network. As faults in the links may partition the network, the algorithm employs the network connectivity to identify whether a node is reachable or not. The paper is organized as follows. A brief review of previous work is presented in section 2. Then we define system and fault model in section 3. The description of proposed algorithm is presented in section 4. Performance analysis of algorithm and simulation results is discussed in section 5 and 6 respectively followed by the conclusion and future works in section 7.

2. A Brief Review

Most of the existing work on distributed diagnosis algorithms deals with static fault environment assuming a fully connected network topology [1-3]. Few diagnosis algorithms based on general or arbitrary topology network have been available in the literature. The main drawback of these algorithms are as follows: (i) they are basically off-line, (ii) follows periodical diagnosis methods and (iii) assuming links are fault-free, (iii) relying on flooding as method of message dissemination [6-8]. Distributed diagnosis algorithm for MANET can be found in [9], [15-16]. Particularly when arbitrary network topology is considered, communication between distant processors typically requires greater communication overhead and testing time. Significant increase in testing time can have detrimental effects on diagnosis time. The Distributed Network Diagnosis (DND) algorithm introduced in this paper is based on the Non-Broadcast Network Diagnosis, which reduces the number of messages compared to using flooding. The algorithm is structured in three phases: test, dissemination and diagnosis. During the testing phase each link is tested by one of the adjacent nodes at alternating testing intervals. Upon the detection of a new unresponsive link, the tester starts the dissemination phase, in which a distributed breadth-first tree is employed to inform the other connected nodes about the event. The algorithm allows dynamic events, i.e. during the dissemination phase new events may occur and dissemination remains guaranteed.

3. Preliminaries

The system and fault model is similar to the model described by A. Subbiah [4]. We assume a synchronous system, where response time, process execution time, clock drift rate and communication delay is within a bounded known amount of time. The diagnosis algorithm proceeds in rounds unless an event is encountered to support on-line diagnosis. It is assumed that the network
protocol detects and discards the incomplete message. The clock drift rate at system nodes is assumed to be a small value \( \rho \). The network is represented by a graph \( G = (V, E) \) where \( V \) is a set of vertices corresponding to the network nodes, and \( E \) is the set of edges, each representing a communication channel connecting two vertices in \( V \).

Both nodes and links may be subjected to crash or value faults [10-13], [17]. The occurrence of fault is independent and represented by a two state machine i.e., 0 and 1 corresponding to fault free and faulty respectively and can occur at any point of time. Crash fault refers to either switch off or physical damage of nodes whereas value fault refers to the erroneous value produced while processing the binary data packet. The link can be either physically faulty or faulty by transmitting corrupted messages. The tests are executed to detect fault in nodes or links. The time-out mechanism is used to detect a crash fault whereas comparing the estimated and observed diagnostic value by the system nodes is used to detect a value fault. However, a node is detected as faulty due to either the fault in tested node or links. If all the links incident to a node becomes faulty, the network is partitioned. A link may be fault-free, unresponsive, unreachable or responsive with sending arbitrary message to the tester. A node considers a link to be unreachable when the link is not adjacent to any other fault-free node. A message for testing a link can contain the checksum and be transmitted to other nodes. However, the data inconsistency or value faults in nodes and during propagation cannot simply be detected by the conventional CRC codes which are prepared before they are transmitted. Therefore the exchange of diagnostic values in the form of hello and reply message can capture such faults.

Diagnostic model: when DND starts execution on a processor \( P_\alpha \), \( P_\beta \) performs a test on another processor \( P_\gamma \). The testing processor sends a Hello Token to tested processor using link \( P_\alpha \) to \( P_\gamma \). The tested processor \( P_\gamma \) sends the same reply Token Hello to processor \( P_\beta \) using the link \( P_\gamma \) to \( P_\beta \). If this reply message does not arrive at \( P_\beta \) within the predefined timeout, \( P_\gamma \) is considered as faulty. However, if this message arrives but does not match due to faults in processor \( P_\gamma \) or link \( P_\alpha \) to \( P_\gamma \) or link \( P_\gamma \) to \( P_\alpha \), the processor \( P_\gamma \) is considered as faulty by processor \( P_\beta \).

4. Algorithm Description

The algorithm has three phases: testing, dissemination of new event information, and diagnosis through local connectivity computation. Nodes execute tests in order to determine the state of adjacent links or nodes. A node can be detected as faulty either due to fault in the node itself or communication link. If test passes, the node or link are fault-free otherwise considered as faulty. It is assumed that the test coverage is 100% and the tests are executed to detect faulty nodes or links.

Every link is tested every testing interval. There is one tester per link. There are two links between a pair of adjacent processors one in each direction. 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. Each timestamp is initially zero, and is incremented for each new event detected on the respective link. This permits a node to identify redundant messages i.e., it is not the first one about a given set of events. After a new event is discovered, the tester propagates event information to its neighbors employing a parallel dissemination strategy based on a distributed breadth-first tree.

Each diagnostic message, shown in figure 1, carries the tree root, and a set of link event information which contains for each event: (1) the tester identifier, (2) the tested node identifier, and (3) the timestamp for the tested link. Message size is assumed of 14 bytes. 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 that the link is faulty due to unresponsive or unreachable. A node changes its state corresponding to dynamic events fault free and faulty events in the node or link. The timestamp is initially 0 and when a fault event occurs in a link, it is incremented by 1 to record a fault about that link. This event is detected in the next time interval when the tester node tests that link. The algorithm assumes that a node or link does not change its state, until all other nodes learn this event in a particular testing interval.

Each node maintains the same graph representing the complete network topology. As each node also learns about the root of the tree upon receiving a given message, all nodes build the same breadth-first tree when disseminating that message. Considering the breadth-first dissemination tree, after the dissemination is started, the new event information is considered to be pending, until every working node in the system acknowledges the receipt of that information. Acknowledgements are propagated from the leaves to the root. Whenever a node
is a leaf, after receiving the message from its parent it sends back an acknowledgement, called henceforth ack. After receiving acks from all its children, a node sends an ack to its parent in the tree. The dissemination completes when the root receives acknowledgements from all its children. When a node receives acknowledgements from all the nodes corresponding to dissemination information, it changes the pending message to regular message.

4.1 Handling Multiple Events

When a new event occurs and the dissemination of the previous event has not yet completed, nodes must handle the dissemination of multiple events. Let a pending and received message be the message of a pending dissemination and the message received by a node. Both messages may carry new information with respect to each other. A message carries new information when it has information about an event at a link which is not present in the other message, or if it has information about a link that is in other message, however with a greater timestamp. When the received message has new information with respect to the pending message, the node merges the new information in the pending dissemination. If the pending message has new information with respect to the received message, the node starts a new dissemination tree. If that is not the case, then the node simply takes part in the dissemination tree according to the received message. The previous dissemination is abandoned when the new dissemination has the complete information about all events. If the received message does not have new information with respect to the pending message, then the received message is simply discarded. When the received message and the pending message are exactly the same, they necessarily come from different trees, started by different nodes (roots). In this case, the node must take part in both dissemination trees. This situation happens when two or more nodes learn about the same set of events before the dissemination started by one of them reaches others. In this case none of the dissemination trees can be abandoned, because there are no criteria to select one of the messages and discard the others. 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 or response with arbitrary messages, i.e. have an odd timestamp, from the network topology.

4.2. Algorithm Specification

Figure 2 depicts the distributed network diagnosis algorithm executed by any node $i$.

**Step 1: Testing phase:**

1.1 Start & RunForever() While TRUE do {For each neighbor $j$ CheckTestToken($j$) End for wait until Testing Interval expires}

1.2 Module: CheckTestToken($j$)
if Token[$j$] = False; RunTest($j$)
else if TokenTurn[$j$] = TRUE; CheckTestToken($j$);
else TokenTurn[$j$] = FALSE
endif
Token[$j$] = TRUE

1.3 Module: ResponseTest($j$)
receive test request from $j$; send test reply to $j$;
if TokenTurn[$j$] = FALSE
endif Token[$j$] = TRUE;

1.4 Module: RunTest($j$)
if (Status[$i,j$] MOD 2 = 0 && Test-Procedure($i,j$)) Status[$i,j$]++; for all $m$ AckToReceive$ = {}$ && AckToSend$ = {}$ End for

2.1 Module: BFT-Disseminate (Root, Message)
Diss_Id ++; PendingMsg, Diss_Id, Message = Message;
PendingMsg, Diss_Id, Root = Root; PendingMsg, Diss_Id, Ack = 0; AckToSend$ = {}$
Build Breadth-First Tree based on the updated topology;
AckToReceive, Diss_Id = {};
For each node $j$ son of node $i$ in BFT Send(PendingMessage, Diss_Id) to node $j$; AckToReceiveDisssid = AckToReceiveDisssid + $j$ End for

2.2 Module: Receive (RcvdMsg)
if exists PendingMsg$m$ that has newer info compared to RcvdMsg
for all $m$ AckToRecv$ = {}$ && AckToSend$ = {}$ End for
else drop RcvdMsg

Step 2: Dissemination phase:

2.1 Module: BFT-Disseminate (Root, Message)
Diss_Id ++; PendingMsg, Diss_Id, Message = Message;
PendingMsg, Diss_Id, Root = Root; PendingMsg,
Diss_Id, Ack = 0; AckToSend$ = {}$
Build Breadth-First Tree based on the updated topology;
AckToReceive, Diss_Id = {};
For each node $j$ son of node $i$ in BFT Send(PendingMessage, Diss_Id) to node $j$; AckToReceiveDisssid = AckToReceiveDisssid + $j$ End for

2.3 Module: Receive(AckFromNode, AckMsg)
if exists PendingMsg$m$ such that PendingMsg$m$.Root = AckMsg.Root & PendingMsg$m$.Message = AckMsg.Message
AckToRecv$ = {}$
if AckToRecv$ = {}$ && AckToSend$ = {}$
send(AckToSendm, AckMsg) End if; End if

Step 3: Diagnosis Phase: Maintain the status of each node and link using dissemination information

Figure 2. The Distributed Network Diagnosis Algorithm

Initially the data structure used to keep event information is declared. The algorithm considers link events, in which a node detects that a fault-free link has become unresponsive. Thus the tester node identifier, the tested node identifier, and the corresponding timestamp describe an event. If a fault-free node becomes faulty, this is an event and recorded in the status field. A message consists of a list of events preceded by the root of the dissemination tree. An Ack bit is employed to allow acknowledgements to be communicated with the same message structure. No message identifier is required, because the list of events together with the identifier of the tree root is enough to make each message unique.

Each node keeps information about the status of each and every network node and link. Status is initially zero.
A node may receive two types of messages: a dissemination message (RcvdMsg) or an acknowledgement message (AckMsg). After a given dissemination has started, and before it has finished it is considered to be pending. Since a node may take part in more than one concurrent dissemination, each node keeps a list of pending messages PendingMsg. For each pending dissemination a node keeps lists of node identifiers to which acknowledgements are to be sent (AckToSends) and to be received (ACKToReceive). A local Diss_Id is kept for each dissemination that a node initiates or takes part. At each testing interval each link is tested by one of the two nodes it connects. Two adjacent nodes exchange a token each testing interval. TokenTurn is employed by one of the nodes to force a test when the other node is faulty. When a new event is detected, the dissemination phase is started. If there is already a pending message, information about the new event is merged into the pending message to start a new dissemination. Module BFT-Disseminate receives the message containing the root of the breadth-first tree. A new Diss_Id is generated. The tree is build based on the updated topology, i.e. by removing faulty links from the graph. The message is sent to each child of node i in the tree. AckToSends and AckToReceive are updated accordingly. Four different situations may occur after a node receives a dissemination message. (1) In case the pending messages have newer information compared to the received message, and the received message also has newer information compared to the pending messages, then a new dissemination is started, using the current node as root. The message disseminated is a merge of the pending messages and the received message. (2) In case the pending message has newer information compared to the received message, the received message is dropped. (3) In case the pending message does not have new information compared to the received message, but the received message does have new information compared to the pending message, then the node takes its part in the current dissemination. (4) In case neither the pending message nor the received messages have newer information compared to each other, then the node takes its part in the current dissemination. When a node receives a dissemination acknowledgement (ReceiveAck), it updates AckToReceive, removing the identifier of the node from which it has received the acknowledgement. If AckToReceive is empty, then the acknowledgement is sent to the node in AckToSends, which is the father of the current node in the dissemination tree.

5. Analysis of the algorithm

Each working node i keeps graph G, as well as state information about all vertices in V, and all edges in E. Node i may consider a vertex to be either fault-free or unreachable. A link may be either fault-free or faulty. In table timestamp is used to keep for each edge a state counter, which is initially zero, and is incremented each time a fault event is detected at the corresponding link. In order to determine the connected component which contains node i, it removes all unresponsive links from graph G, and employs a graph connectivity algorithm to determine which nodes and links are unreachable. A connected component Gi = (Vi, Ei) is a subgraph of G of a node i where vertex vi belongs to Vi if it is not unreachable to node i, edge ei belongs to Ei if it is not unreachable to node i. After a new event is detected on an adjacent link, a node starts the dissemination of a message on a distributed breadth-first tree. A dissemination round is defined as the time interval in which all nodes at one level of the tree send the message to all their children nodes in the next level.

Theorem 1. Consider a fault event on a link that does not partition a given connected component. Let d be the connected component diameter. Consider that both nodes connected by the link detect the event, and that no other event occurs before the dissemination completes. In at most 3d dissemination rounds the link event information reaches all nodes that belong to the connected component.

Proof. Consider that link (i,j) becomes unresponsive to both node i and node j, which start dissemination tree. As both nodes belong to the same connected component, then there exists at least one node k, which is the first to receive both dissemination messages. The distance from node i to node k is at most d, as well as the distance from node j to node k. So it may take at most 2d dissemination rounds for a node to receive both dissemination messages and start a new dissemination tree with a message that is a merge of the two previous messages. In at most more d dissemination rounds the resulting message reaches all nodes in the connected component. Thus in at most 2d + d = 3d rounds all nodes learn about the event.

Theorem 2. Consider that m link fault events that do not partition a given connected component are detected before the dissemination of any of these events completes. In at most 2m+d dissemination rounds every connected node receives information about all events.

Proof. For each link fault event two nodes determine that the link has become unresponsive and start disseminating trees. Thus for m link fault events 2m dissemination trees are started. As we consider only nodes belong to the same connected component, then in at most d dissemination rounds at least one node, which receives two dissemination messages, starts the dissemination of a message that is a merge of the two received messages. Considering this new dissemination tree, in at most d
dissemination rounds the dissemination message may reach a node that has another pending dissemination. Thus, for all \(2^m\) trees to reach a node that merges their messages it takes at most \(2^m d\) dissemination rounds. Finally, in at most more \(d\) dissemination rounds the resulting message reaches all nodes in the connected component. Thus in at most \(2^m d + d\) rounds all nodes learn about all events.

**Theorem 3.** Consider a link fault event that partitions a given connected component into two connected components. Let \(d_1\) and \(d_2\) be the diameters of the resulting connected components. Consider that information about all previous events on these components have been completely disseminated. Consider also that no other event occurs in these components. In at most \(d_1\) and \(d_2\) dissemination rounds, respectively, every node in each connected component learns about the partition.

**Proof:** Consider that link \((i,j)\) becomes unresponsive to both node \(i\) and node \(j\), which start dissemination trees. Considering that link \((i,j)\) partitions the system into two connected components, node \(i\) is in one of these components and node \(j\) is in the other. As \(d_1\) and \(d_2\) are the diameters of the resulting connected components, the trees built by node \(i\) and node \(j\) take at most \(d_1\) and \(d_2\) dissemination rounds to reach every connected node in the respective component.

**Theorem 4:** Consider a link fault event that partitions a given connected component into two connected components. Consider also that \(t\) disseminations are pending at one of the components, using the system topology as it was before the partition. Let \(d\) be the diameter of that connected component. Consider also that no other event occurs in this component. In at most \(t d + d\) dissemination rounds all nodes in the connected component learn about the partition and all events that were being disseminated before the partition.

**Proof:** Consider that link \((i,j)\) becomes unresponsive to node \(i\) and which starts a dissemination tree. Considering that link \((i,j)\) partitions the system into two connected components, node \(i\) is in one of these components and node \(j\) is in the other. As \(d\) is the diameter of node \(i\)’s component, the tree built by node \(i\) takes at most \(d\) dissemination rounds to reach node \(k\) that has one of the pending disseminations on that component. Node \(k\) learns about the partition and starts the dissemination of a message that is a merge of both messages. Considering this new dissemination tree, in at most \(d\) dissemination rounds the dissemination message may reach a node that has another pending dissemination. Thus, for all \(t\) trees to reach a node that merges their messages it takes at most \(t d\) dissemination rounds. Finally, in at most more \(d\) dissemination rounds the resulting message reaches all nodes in the connected component. Thus in at most \(t d + d\) rounds all nodes learn about all event.

### 6. Simulation results

The C++ is used as the simulation tool. The random graphs of 16, 32, 64, 128 and 256 nodes with connectivity 3 were created. The simulation parameters that have been chosen for evaluating the system are (i) diagnostic latency and (ii) message complexity. Diagnostic latency is the time taken by all the nodes in the network to diagnose an event and message complexity is the total number of messages exchanged to achieve diagnosis. Average diagnostic latency was computed corresponding to all events for 1000 simulation runs. For each simulation run, up to three fault events were scheduled considering both node and link failures. The dynamic nature of the system is modeled using a poison process similar to the simulation model presented in paper [4]. Testing intervals of 30 time units, and dissemination intervals of 1 time unit between nodes were employed.

Transmission, reception and propagation delay of messages were assumed to take 0.1, 0.2 and 1 time unit respectively.

![](image-url)  **Figure 3 Diagnostic Latency vs. n**

![](image-url)  **Figure 4. The Number of messages vs. n**

129
Our future work includes intermittent fault diagnosis for network such as MANET and wireless sensor network. Used for fault diagnosis of the Internet and other types of Chinese Agent. Nevertheless, our approach can also be diagnostic latency of flooding and message complexity of algorithm DND balances the tradeoff between the environment. An important observation is that, the message complexity and thus suitable for static fault environment and similar to Chinese Agent in terms of diagnostic latency and thus suitable for dynamic fault environment and similar to the performance of flooding in terms of latency of O(td) for the algorithm. The algorithm DND is presented. We prove bounds on the worst-case diagnostic and link faults in an arbitrary topology network is A distributed diagnosis algorithm considering node and link faults in an arbitrary topology network is presented. We prove bounds on the worst-case diagnostic latency of O(td) for the algorithm. The algorithm DND is similar to the performance of flooding in terms of diagnostic latency and thus suitable for dynamic fault environment and similar to Chinese Agent in terms of message complexity and thus suitable for static fault environment. An important observation is that, the algorithm DND balances the tradeoff between the diagnostic latency of flooding and message complexity of Chinese Agent. Nevertheless, our approach can also be used for fault diagnosis of the Internet and other types of network such as MANET and wireless sensor network. Our future work includes intermittent fault diagnosis for wireless sensor network.

7. Conclusion

A distributed diagnosis algorithm considering node and link faults in an arbitrary topology network is presented. We prove bounds on the worst-case diagnostic latency of O(td) for the algorithm. The algorithm DND is similar to the performance of flooding in terms of diagnostic latency and thus suitable for dynamic fault environment and similar to Chinese Agent in terms of message complexity and thus suitable for static fault environment. An important observation is that, the algorithm DND balances the tradeoff between the diagnostic latency of flooding and message complexity of Chinese Agent. Nevertheless, our approach can also be used for fault diagnosis of the Internet and other types of network such as MANET and wireless sensor network. Our future work includes intermittent fault diagnosis for wireless sensor network.

References