The application of system level diagnosis approaches for fault detection and its implementation in Ad-hoc networks

A thesis report submitted in partial
fulfillment of the requirements for the degree of

Bachelor of Technology in Computer Science

By

Amit Vishal Mundu

Roll No.: 10606010

&

Sumeet Gupta

Roll No.:10606021

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela – 769008

The application of system level diagnosis approaches for fault detection and its implementation in Ad-hoc networks

A thesis report submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology in Computer Science

By

Amit Vishal Mundu

Roll No.: 10606010

&

Sumeet Gupta

Roll No.:10606021

Under the guidance of

Dr. Pabitra Mohan Khilar

Asst. Prof

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela – 769008

National Institute of Technology Rourkela
Rourkela-769008, Orissa

Certificate

This is to certify that the work in this Thesis Report entitled "**The application of system level diagnosis approaches for fault detection and its implementation in Ad-hoc networks** " submitted by **Sumeet Gupta**(10606021) and **Amit Vishal Mundu**(10606010) has been carried out under my supervision in partial fulfillment of the requirements for the degree of Bachelor of Technology in Computer Science during session 2009-2010 in the Department of Computer Science and Engineering, National Institute of Technology, Rourkela, and this work has not been submitted elsewhere.

Place: NIT Rourkela
Date :

(**Dr.Pabitra Mohan Khilar**)
Asst. Prof
Department of CSE
NIT Rourkela

Acknowledgements

No thesis is created entirely by an individual, many people have helped to create this thesis and each of their contribution has been valuable. My deepest gratitude goes to my thesis supervisor, Dr. **Pabitra Mohan Khilar**, Asst Prof., Department of CSE, for his guidance, support, motivation and encouragement throughout the period this work was carried out. His readiness for consultation at all times, his educative comments, his concern and assistance even with practical things have been invaluable.

I am grateful to Dr**. Banshidhar Majhi**, *Professor and Head, Dept. of CSE* for his excellent support during my work. I would also like to thank all professors and lecturers, and members of the department of Computer Science and Engineering for their generous help in various ways for the completion of this thesis. And finally thanks to my fellow students for their friendly co-operation.

Amit Vishal Mundu

10606010

Sumeet Gupta

10606021

Abstract

We introduce the application of a distributed system-level fault diagnosis algorithm for detecting and diagnosing faulty processors in Dynamic Positioning System (DPS).In this paper a new approach to the diagnosis problem is presented. We illustrate the procedure of diagnosis verification, which adopts a method of fault injection by setting some faults in the system by programming, and provide the basis idea, the detailed execution steps and the corresponding results. This algorithm is then on two models of ad-hoc networks. Two implementation models are presented in the first one network topology doesn't change during diagnosis and we show that both hard and soft faults can be easily detected based on this, a diagnosis protocol is presented. The evaluation of the protocol indicates that an efficient diagnosis protocol can be designed on our model. In the second model we allow the system topology to change during diagnosis, in this case the ability of diagnosis decreases but it can be rectified too by setting up a fixed area for mobility of nodes.

Contents

List of Figures

Chapter 1 Introduction

1.1 Introduction

The basic idea in incorporating a fault tolerance capability to a distributed system is to provide the system with extra (redundant) resources. A number of investigations have been made to extend traditional notions of "fault-tolerant computing", to deal with the problem of failures, thus affecting the facilities of distributed systems and computer networks. As distinguishing diagnostic responsibility needs the flow of diagnostic information through the network, and the faulty facilities may again themselves participate in this flow and alter, destroy, or generate erroneous diagnostic information in the process, thus delivering the whole diagnostic procedure itself quite complex. The purpose of this study is to simulate a distributed system that is firstly a dynamic positioning system and then an ad-hoc network and carry this fault diagnosis under Arbitrary Network topologies. The distributed system level diagnosis algorithms discussed in this paper give a comprehensive idea about the various issues one must keep track of while going to develop such fault tolerance algorithms. Since "system level diagnosis" is one of the steps in the process of building "distributed fault-tolerant systems", reliability of such a system depends heavily on proper functioning of the diagnosis algorithm [5].

As given in [1] each unit in the system can be used as Master Test Unit to evaluate the other unit that acts as Slave Test Unit, by transmitting interactively. This unit can also act as Slave Test Unit and be evaluated by the other unit that is so-called Master Test Unit correspondingly. But we prescribe that any unit can be tested by one other unit at any time. Master Test Unit will receive information from or send the results of interactive diagnosis to its subset of fault-free Slave Test Units. We assume that at-least one system to be fault-free.

The distributed system has been simulated in JAVA using the Java program to create windows, representing client and server. Afterwards, distributed diagnosis algorithm has been simulated using this environment.

1.2 Related Works

The system level-fault diagnosis was introduced by three eminent professors F.P Preparata , G. Metze and R.T Chien in 1967[11] to diagnose faults in a system which is composed of a number of interconnected units connected by wire. Testing takes place in between a pair of adjacent units. A testing unit sends a test token to tested unit and the tested unit after computing the result returns it to the testing unit. After that, testing unit generates the outcome after comparing the generated the result with expected one. If unit is faulty, outcome is 1, else 0.This concept has been applied to many applications like Dynamic Positioning System [1], Ad-Hoc Networks [2].

System level-fault diagnosis algorithm in applied on Dynamic Positioning System by four professors Hongian Wang, Xinqian Bian, Fuguang Ding, Guiping Han in 2002[1]. They adopted a method which is injecting a fault in the system by programming and then providing the basic details of steps to show how diagnosis is being done.

This algorithm is then applied for fault detection in Ad-Hoc networks by two professors Stefano Chessa and Paolo Santi[2]. They provided two implementations of the model, first one in which the topology of network doesn't changes and in the other, topology changes with time. In both scenarios they showed how hard and soft faults can be diagnosed.

1.3 Thesis Objectives

Our thesis has following objectives:

- apply system level fault diagnosis in Dynamic Positioning System
- apply system level fault diagnosis for randomly distributed nodes
- apply system level fault diagnosis for hypercube interconnected network
- compare the diagnostic latency and message complexity for above two scenarios
- apply system level fault diagnosis for fixed and dynamic ad-hoc networks
- compare the diagnostic latency for wired and wireless networks

1.4 Thesis Organization

Chapter 1 of the thesis gives a brief introduction of our thesis. Then chapter 2 introduces the concept of Dynamic Positioning System. Then chapter 3 introduces the concept of system level fault diagnosis and the details of the simulation steps involved in it. In chapter 4, there is a brief pseudo code for generating the connection set, testing the system and generating the fault vector. Chapter 5 applies system level diagnosis algorithm for random distribution of nodes and results are generated. Chapter 6 applies this algorithm on hypercube interconnected network. Chapter 7 deals with introduction to Ad-Hoc networks. In chapter 8 an algorithm is proposed for finding faulty nodes in ad-hoc networks.

In chapter 9 the algorithm is applied for ad hoc networks with fixed and dynamic topologies. Chapter 10 gives the results for the implemented algorithm and results are compared for wired and wireless networks. Chapter 11 gives the conclusion and future works.

Chapter

2 Dynamic Positioning System

2.1 Dynamic Positioning System

It is a computer controlled system to automatically maintain a vessel's position and heading by using her own propellers and thrusters. Position reference sensors, combined with wind sensors, motion sensors and gyro compasses, provide information to the computer pertaining to the vessel's position and the magnitude and direction of environmental forces affecting its position. The dynamic positioning system of an offshore vessel is a computerized system, which enables the vessel to be automatically controlled. It composes of sensor subsystem, propeller subsystem and control subsystem. Among these subsystems, control subsystem is the kernel equipment.

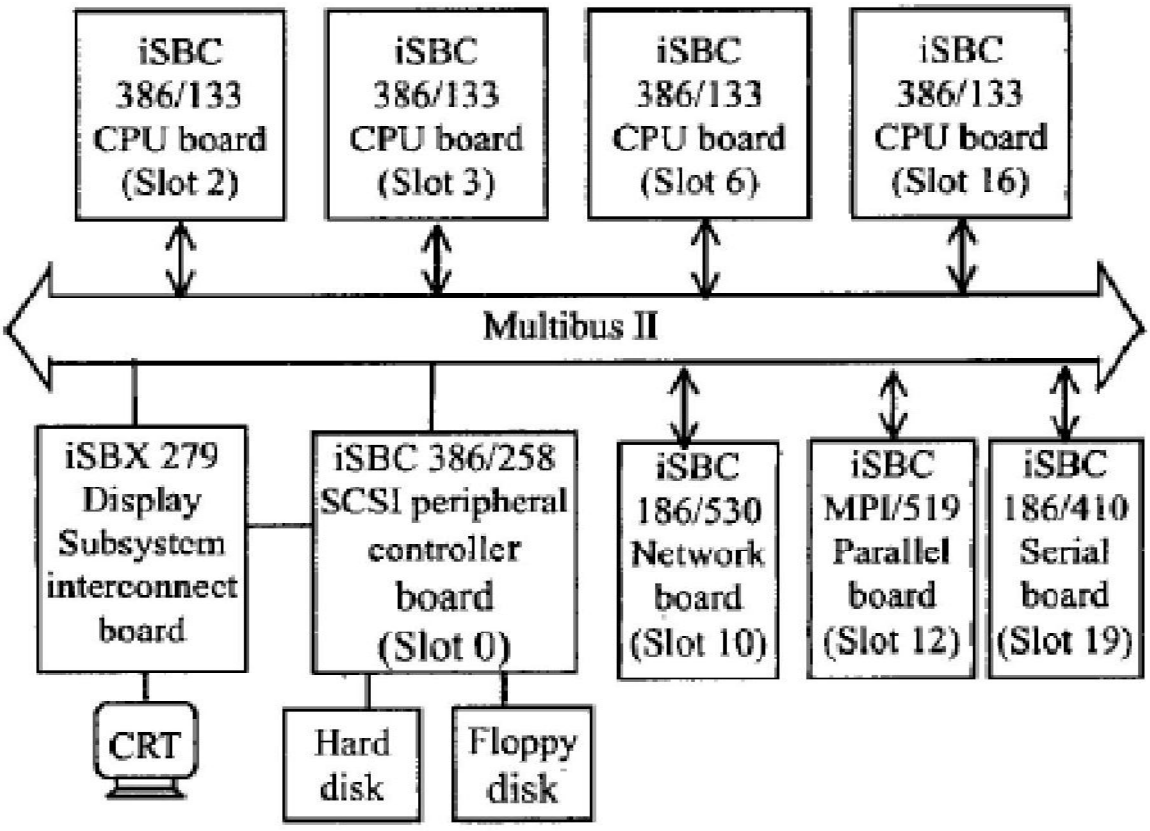

Figure 2.1 Control System in a DPS (Adapted from [1])

Four pieces of ISBC 386/133 boards have the same construction and equal basis function in the system, but completing the different missions and equipped with different interface and application programs. **So** the paper treats the problem of automatic fault diagnosis for DPS with the method of distributed system-level fault diagnosis regarding no centre processor in the system, and each processor separately tests the others independently and a proper diagnosis can be arrived at for some diagnosable fault pattern [1].

Chapter

3 Algorithm For Fault Diagnosis

3.1 Characteristics of a Node:

A node can be either *fault-free* or *faulty*. A fault-free node performs its assigned system computational and communication tasks, and it has a local notion of time. A fault-free node is accepted to know which nodes are its physical neighbors in the network. This information can be assured by the node via external means or internally, through low-level hardware and software methods. In accession, a fault-free node is assumed to be able to initiate a test of a neighboring node and to be able to respond to a test initiated by one of its neighbors. A fault free node, by definition, responds correctly and within a specified time, this period to a test. Finally, a fault-free node is able to request that a neighboring node become its tester when prescribed by the diagnosis algorithm. On the contrary, a faulty node is assumed to be unable to respond to a test, to a request, or to diagnosis information sent to it from a neighboring node. Also, a faulty node is assumed to be unable to format and forward diagnosis information contents and is assumed to be unable to generate inauthentic requests for other nodes to test it. It is assumed that the node that issues a message to a neighboring node receives the corresponding reply (such as a test response or an acknowledgement) within a certain time period if and only if the neighbor is fault-free; otherwise, it times this on the reply. In order to satisfy this requirement, it is necessary that the communication channels between the nodes have a bounded delay. Time, this periods are determined as a function of this delay. Also, it is assumed that message ordering is maintained within the communication channel; equivalently, an appropriate communication protocol can be used by the sender and receiver to ensure that messages are received in the order they are delivered. A fault-free node can become faulty at any time. A faulty node can be repaired and reintegrated into the network at anytime. When a faulty node is repaired, it is assumed that it regains all of the characteristics of a fault-free node, including determining who its physical neighbors are. However, a newly repaired node has no knowledge of the fault-free or faulty status of the other nodes in the system. This paper assumes no communication link faults. A test could fail due to a faulty node or due to a faulty communication link, with this distinction. That means it treats link faults as node faults.

3.2 Levels of Fault Diagnosis:

- The system can't calculate correctly.
- The system can't get data in time from processor.
- The system can't send a token within a certain time
- The system can't receive or response a token within a certain time (As given in [1]).

3.3 Diagnosis Token:

- Request Token: Slave Test Unit sends Request Token to its neighboring units for being test.
- Test Token: Master unit sends test token including test content to Slave Test Unit.
- Answer Token: Answer token is a testing answer to Test Token and is sent to Master Unit by Slave Unit.
- Information Token: It is the result of diagnosis and is sent to fault-free Slave Unit by Master Test Unit.

3.4 Fault Vector

Each system in the DPS has its own fault vector. Let there are 4 systems (MP1, MP2, MP3 and MP4). Then e each system has its own fault vector in uniform format as follows.

Figure 3.1 Fault Vector (Adapted from [1])

Each fault vector includes four bits separately denoted the systems **MPI, MP2, MP3** and **MP4** from left to right. Each bit of fault vector may be marked with the symbol of '0', **'1'** or 'x', correspondingly, '0' is fault-free, **'1'** is faulty, and 'x' is uncertainty. At the beginning period of test, each unit firstly marks the native bit of its fault vector with 'O', and marks the other bits with 'x' [1][3].

3.5 Procedure for finding Fault Vector:

3.5.1 Constructing Test Matrix

At first, a test matrix is made which shows that which system is testing which one. For example, we have 4 systems MP1, MP2, MP3, and MP4. The test matrix can be shown as:

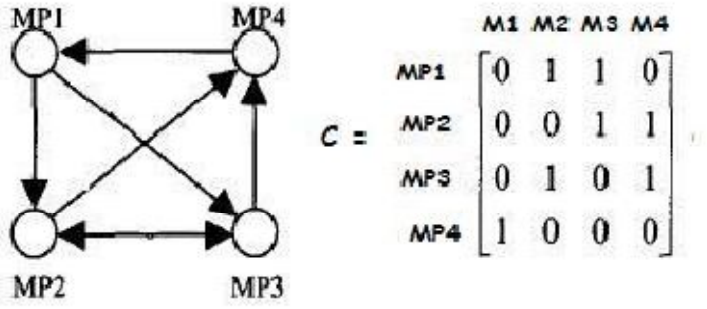

Figure 3.2 Connection Matrix (Adapted from [1])

System at the row tests the corresponding system at the column. The system tests only if the value is 1.[1][3]

3.5. Filling up the Fault Vector

Execution of diagnosis algorithm with the following steps:

1. **MPI** marks the native bit of its fault vector with '0', and marks the other bits with 'x'. The fault vector of **MPI** is shown as follows.

0	X	X	X

Figure 3.3 Fault Vector for MP1 (Adapted from [1])

2. According to the graph-theoretic model, **MP1** tests two units **MP2, MP3** and the test this comes are both 0, and then it puts the test information into its fault vector. At this time the fault vector of **MP1** is changed as follows.

0	0	0	X	(MP1)

Figure 3.4 Changed fault Vector for MP1 (Adapted from [1])

3. Analyzing the fault vector of **MPI,** it's clearly that the units **MP2, MP3** are evaluated by **MP1** and the diagnosis conclusion is that **MP2** and **MP3** are fault-free. **So MP1** receives the Answer Tokens of **MP2** and **MP3** respectively according to the fundamental rule. We can gain

the fault vectors of **MP2** and **MP3** separately on the ground of the parts of interactive diagnosis and self-diagnosis in their Answer Token. Fault vector of **MP2** and fault vector of **MP3** are shown as follows.

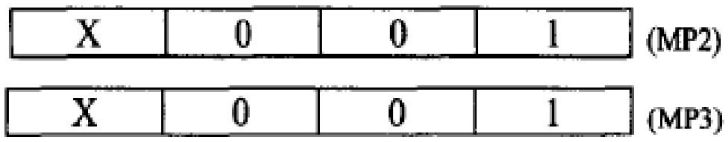

Figure 3.5 Fault Vector for MP2 and MP3 (Adapted from [1])

4. Adding the fault vector of **MP2** and the fault vector of **MP3** to the fault vector of **MP1** and getting the final fault vector of **MPI** as follows.

Figure 3.6 Final Fault Vector for MP1 (Adapted from [1])

5. According to the final fault vector of **MP1,** we can get the conclusion that **MPI, MP2** and **MP3** are all fault-free, but **MP4** is faulty.

3.6 Simulation steps for detecting a faulty system:

The following steps were followed to stimulate our algorithm, in a java code:

- Connection set is build up. (Which system can test which one).
- A fault is inducted in a unit.
- The unit to be tested is selected randomly (say N1).
- N1 sends request token to connected systems.
- Connected Systems send test token to the unit to be tested.
- The answer token is send by tested token (N1).
- Repeating the above steps all the units are checked.
- According to answer token received each unit forms its fault vector.
- Adding the fault vectors of the units we get the final vector for each unit.
- Ultimately all the fault vectors are added to produce the faulty unit.

Chapter

4 Pseudo code

4.1 Creating a connection set:

```
int con[][]=new int[n][n];        // n is total number of nodes
ConnectionSet() {
 int x;
for(int i=0;i<n;i++ )
for(int j=0;j<n;j++)
          {
          x=(int)((Math.random())*2);
          con[i][j]=x;
         }
for(int i=0;i<n;i++ )
for(int j=0;j<n;j++) {
if(con[i][j]==1
con[j][i]=1;
con[i][i]=0;
                        }
for(int i=0;i<n;i++ ) {
for(int j=0;j<n;j++) {
System.out.print(con[i][j]+" ");
        }
System.out.println();
       }
```

4.2 Testing the systems:

```
void Testing() {
 int req,i;
 int get[]=new int[n];
 int cnt=0;
 while(true)
```

```
        {
 req=(int)((Math.random())*n);
 if(rem[req]==0)
           {
rem[req]=1;
req=req+1;
cntset++;
break;
           }
if(cntset==n)
break;
         }
 System.out.println("System "+req+" wants to be tested !!");
System.out.print("System "+req+" is connected to system : ");
for(i=0;i<n;i++) {
if(con[req-1][i]==1) {
get[cnt]=i+1;
cnt++
System.out.print((i+1)+" ");
        }
        }
System.out.print("\nSystem "+req+" sending Request Tokens to Systems : ");
for(i=0;i<cnt;i++)
System.out.print(get[i]+" ");
System.out.print("\nSystem ");
for(i=0;i<cnt;i++)
System.out.print(get[i]+" ");
System.out.print("sending Test Token to System "+req);
System.out.println("\n SYSTEM "+req+" UNDERGOING TESTING !!\n");
try{
Thread.sleep(1000);
       }
catch(Exception e){
e.printStackTrace();
        }
System.out.println("TESTING DONE !!\n");
System.out.print("System "+req+" sending Answer Token to System : ");
for(i=0;i<cnt;i++)
System.out.print(get[i]+" ");
try{
```

```
Thread.sleep(1000);
        }
catch(Exception e){
e.printStackTrace();
        }
System.out.println("\n");
        }
```

4.3 Finding Fault Vector:

```
for(int l=0;l<n;l++)
          {
 if(i==l)
           {
System.out.print(0+"\t");
           }
if(con[i][l]==0 && i!=l)
System.out.print("X\t");
if(con[i][l]==1 && (i+1)!=fault && (l+1)!=fault)
System.out.print(0+"\t");
if(con[i][l]==1 && ((i+1)==fault || (l+1)==fault))
System.out.print(1+"\t");
        }
try{
Thread.sleep(1000);
        }
catch(Exception e){
e.printStackTrace();
        }
System.out.println();
      }
System.out.println("Now... adding the fault vectors of connected fine Systems...We get the Final
FAULT VECTOR !!");
System.out.println("FINAL FAULT VECTORS : \n");
```

```
for(int i=0;i<n;i++)
        {
System.out.print("SYSTEM "+(i+1)+" : ");
try{
Thread.sleep(1000);
         }
catch(Exception e){
e.printStackTrace();
         }
for(int j=0;j<n;j++)
        {
if(i==j) // Complete.....
        {
System.out.print(0+"\t");
continue;
        }
if(con[i][j]==0 && ((i+1)!=fault && (j+1)!=fault))
System.out.print(0+"\t");
if(con[i][j]==1 && ((i+1)!=fault && (j+1)!=fault))
System.out.print(0+"\t");
if(con[i][j]==0 && ((i+1)!=fault && (j+1)==fault))
System.out.print(1+"\t");
if(con[i][j]==1 && ((i+1)!=fault && (j+1)==fault))
System.out.print(1+"\t");
if(con[i][j]==1 && (i+1)==fault)
System.out.print(1+"\t");
if(con[i][j]==0 && (i+1)==fault)
System.out.print(0+"\t");
         }
System.out.println();
         }
```

```
System.out.println("CONCLUSION : ");
System.out.print("According to Systems ");
for(int g=0;g<n;g++)
        {
if((g+1)==fault)
continue;
System.out.print((g+1)+" ");
        }
System.out.println("\n\nSYSTEM "+fault+" IS FAULTY !!! \n");
        }
```

Chapter

5 Implementing in Random Distribution

5.1 GRAPH –Number of Nodes Vs Diagnostic Latency

In this section, we will discuss on two parameters, diagnostic latency and message complexity.

- **Diagnostic Latency:** It is the amount of time required to completely fill fault vectors of all nodes.
- **Message Complexity:** It is the total number of messages exchanged within a system to correctly identify the status (faulty or fault free) of all the nodes.

The above algorithm was implemented and it was found that as the number of nodes increases, diagnostic latency (figure 5.1) and message complexity (figure 5.2) also increases.

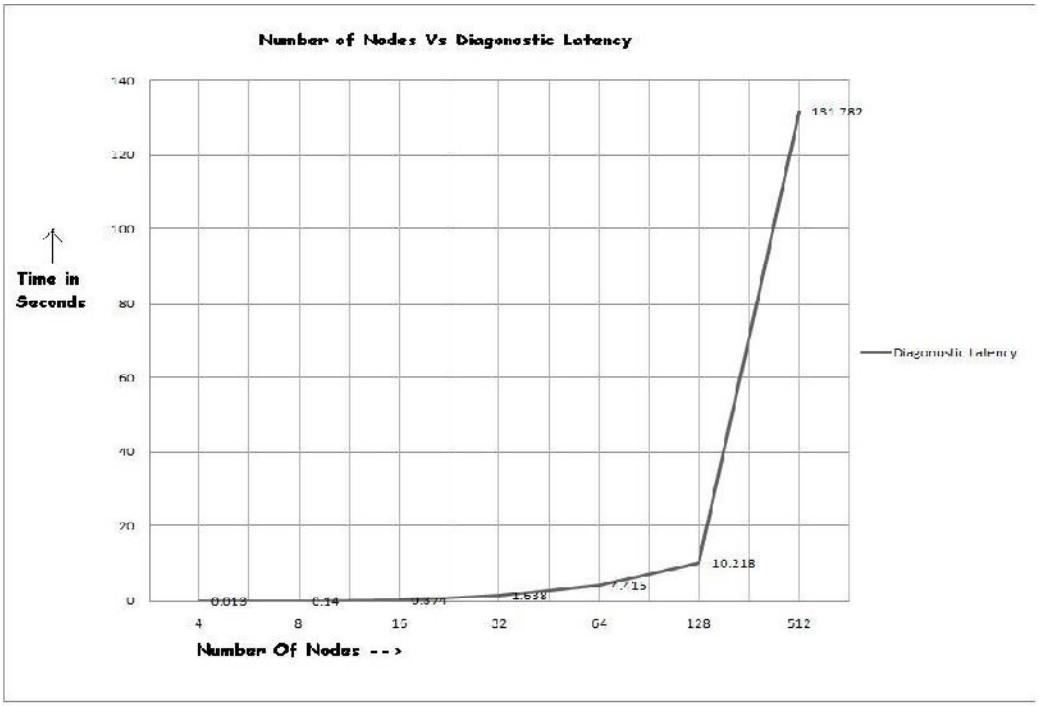

Figure 5.1 Nodes Vs Diagnostic Latency

5.2 GRAPH-Number of Nodes Vs Messages Exchanged

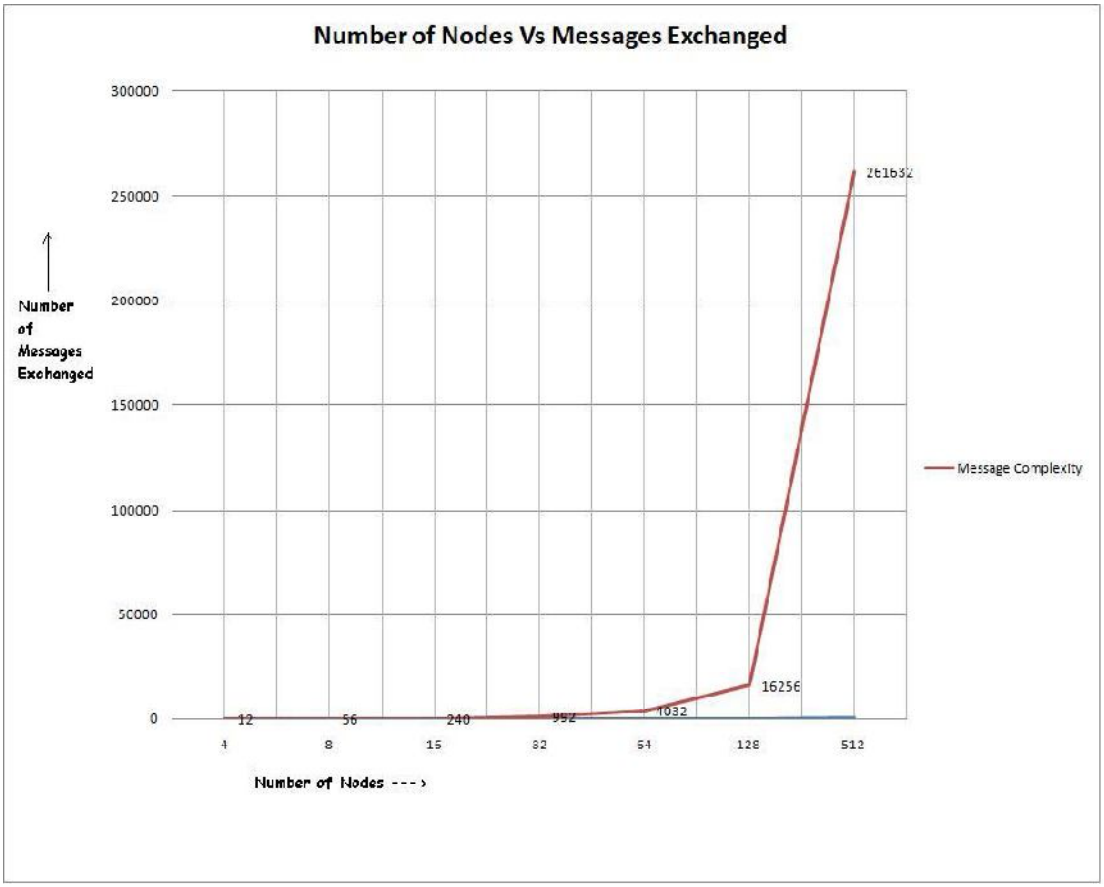

Figure 5.2 Nodes Vs Message Complexity

5.3 Discussion

From the above graph the complexity of the algorithm turns this to be:

In worst case n (n-1).

Here each node has to check the other node and hence n nodes testing the rest n-1 node gives a complexity of n (n-1).

In best case n.

The best case here implies that all the nodes are kept in a straight line and one node is being tested once by a node.

So the main part in the above algorithm turns this to be the arrangement of nodes in a test matrix.

Chapter

6 Hypercube

6.1 Implementation on Hypercube

Hypercube is an n-dimensional structure which is analogous to a square. It consist of N = 2^k nodes arranged in a k dimensional hypercube. 'k' gives the number of edges adjacent to a node like square for k=2, cube for k=3. Processors in a hyper cube type network having 2^k nodes (k>0, integer), each of the nodes being arranged on an apex of a cube and having n sets of links for interconnecting other nodes so as to form an n-dimensional hyper cube type network; a plurality of processors, each processor being connected to each node by input/output links, thereby providing communication paths between processors through the nodes and links; each of the nodes comprising: a device for setting 2^k different connection patterns .

The following figure shows a 3-cube:

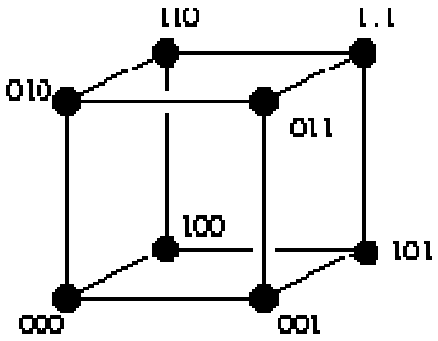

Figure 6.1 A Hypercube of order 3

We can get from any node to any other node in at most log N hops (unless there is contention) so the latency is O (log N). There are N processors, each with $\log_2 N$ interfaces, so the cost is O (N log N). And all the processors can use their links simultaneously, so our aggregate bandwidth is O (N).With continuous increase in network size, a fault has become unavoidable. Hypercube networks are among the earliest proposed network models and still remain as one of the most important and attractive ones. Recent research has also shown that fault tolerant hypercube networks of large size can be used as an effective control topology in supporting large-scale multicast applications in the Internet. Extensive experience has shown that hypercube networks can tolerate a large number of faulty nodes while still remain functioning [9].

6.2 Result

The above system level fault diagnosis algorithm is implemented on nodes connected in hypercube structure. The hypercube considered is 3-cube having 8 nodes. After implementing the algorithm, it is found that the complexity of algorithm is reduced to O (N).

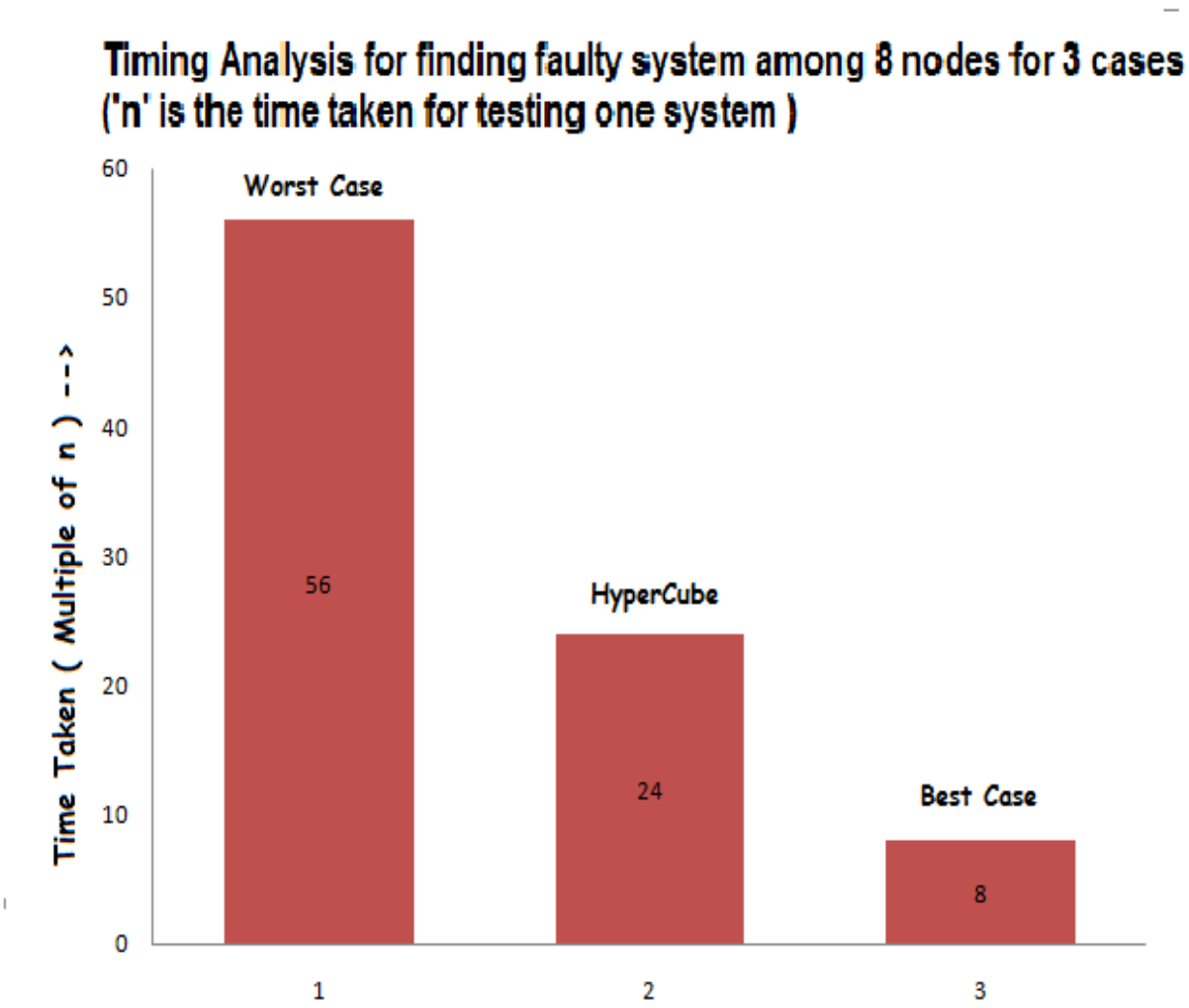

Figure 6.2 Time complexity analysis

The complexity is shown in the following figure. This becomes the average case. The best case becomes when all the nodes except one is fault free and the nodes are being tested in a straight line and provided that the faulty node is the last one. In that case, with minimum message passing, the status of every node is found out.

Chapter 7

Ad-hoc Networks

7.1 Introduction to Ad-hoc Networks

A wireless ad hoc network is a decentralized wireless network. The network is ad-hoc because each node is willing to forward data for other nodes, and so the decision of which nodes forward data is made randomly based on the network connectivity. This is in contrast to wired networks in which routers perform the task of routing. It is also in contrast to managed (infrastructure) wireless networks, in which a special node known as an access point manages communication. The decentralized nature of wireless ad-hoc networks makes them suitable for a variety of applications where central nodes can't be relied on, and may improve the scalability of wireless ad-hoc networks compared to wireless managed networks, though theoretical and practical limits to the overall capacity of such networks have been identified. Minimal configuration and quick deployment make ad-hoc networks suitable for emergency situations like natural disasters or military conflicts. The presence of a dynamic and adaptive routing protocol will enable ad hoc networks to be formed quickly [2].

There are two types of ad-hoc networks [2]

1. Fixed ad-hoc networks
2. Dynamic(mobile) ad-hoc networks or MANET

Fixed ad-hoc networks are those in which topology of the networks does not changes. The nodes which are initially in the transmitting range of one node continues to be in that position. Fault detection is easier in this type of ad-hoc.

Dynamic ad-hoc networks are those in which topology of network frequently changes. They are also called a mobile mesh network. Node mobility causes frequent change n topology [6].

In the given figure, node D was initially within transmitting range of node A. After some time, it migrates out of it [6].

.

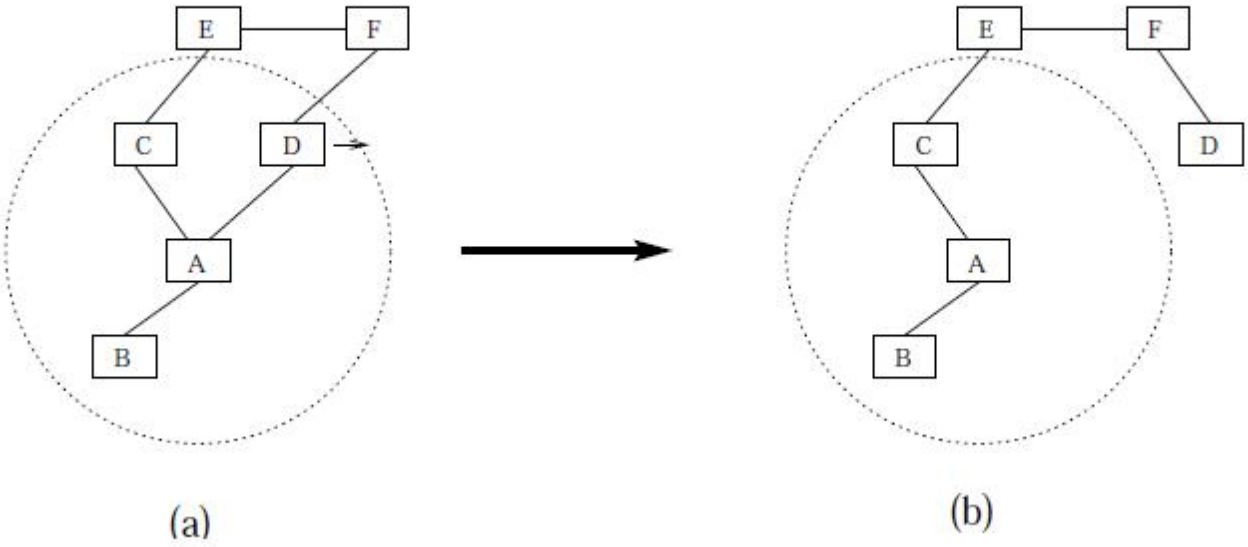

Figure 7.1 Dynamic Ad-hoc Network (Adapted from [7])

7.2 Issues in Deploying Ad-hoc Networks

1. **Unpredictability of environment**: Ad-hoc networks may be deployed in unknown terrains, risky conditions, and even unfriendly environments where fiddling or the actual destruction of a node may be imminent. Depending on the environment, node failures may occur frequently.

2. **Unreliability of wireless medium**: Communication through the wireless medium is unreliable and subject to errors. Also, due to varying environmental conditions such as high levels of electro-magnetic interference (EMI) or merciless weather, the quality of the wireless link may be unpredictable Furthermore, in some applications, nodes may be resource-restrained and thus would not be able to support transport protocols necessary to ensure reliable communication on a lossy link. Thus, link quality may vacillate in an ad-hoc network.

3. **Resource-constrained nodes**: Nodes in a MANET are typically battery powered as well as limited in storage and working capabilities. Moreover, they may be situated in areas where it is not possible to re-charge and thus have limited lifetimes. Because of these restrictions, they must have algorithms which are energy-efficient as well as operating with limited processing and memory resources. The available bandwidth of the wireless medium may also be limited because nodes may not be able to sacrifice the energy consumed by operating at full link speed.

4. **Dynamic topology**: The topology in an ad-hoc network may change constantly due to the mobility of nodes. As nodes move in and this of range of each other, some links break while new links between nodes are created [6].

7.3 Fault Model:

Integrated overlays are often built in a manner that makes failures unlikely to affect the overall system. This is possible because in the wired internet failures are less likely than in a wireless environment and the network is fairly well connected. In the wireless environment the failures will be more drastic, thus exacerbating many of the problems of correlated failure. In an ad-hoc environment there may be one node holding two halves of a network together, and if that node fails the two halves of the network will each see half the nodes as having failed. Moreover there are also many other factors that can cause wireless nodes to fail at a much higher rate. Many devices using wireless interfaces are powered by batteries, and will turn off to conserve batteries. Wireless devices are subject to disconnection due to the peculiarities of signal propagation, a device moving between two rooms may get varying degrees of connectivity. The loss rate of the wireless channel is also much higher than that of a wired Ethernet channel resulting in the need to retransmit more packets.

An integrated overlay designed on top of a treacherous system such as this must be implemented to handle these types of failures graciously and not produce much additional overhead. The system must be bouncy to failure as extreme as large network partitions that may make many nodes unreachable. For successful scalable network services to be built, these types of faults must not greatly degrade the services being provided [4].

So based on these basically we have the following types of faults:

1. **Transmission errors**: The unreliability of the wireless medium and the unpredictability of the environment may lead to transmitted packets being garbled and thus received in error.
2. **Node failures**: Nodes may fail at any time due to different types of hazardous conditions in the environment. They may also drop this of the network either voluntarily or when their energy supply is depleted.

3. **Link failures**: Node failures as well as changing environmental conditions (e.g., increased levels of EMI) may cause links between nodes to break.

4. **Route breakages**: When the network topology changes due to node/link failures and/or node/link additions to the network, routes become this-off date and thus incorrect. Depending upon the network transport protocol, packets forwarded through stale routes may either eventually be dropped or be delayed; packets may take a circuitous route before eventually arriving at the destination node.

5. **Congested nodes or links**: Due to the topology of the network and the nature of the routing protocol, certain nodes or links may become over utilized, i.e., congested. This will lead to either larger delays or packet loss [6].

Over here we basically divide the faults into two broad reasons. In our paper:

The problem of identifying faulty mobiles in ad-hoc networks is considered. Current diagnostic models were designed for wired networks, thus they do not take advantage of the shared nature of communication typical of ad-hoc networks. Here we introduce a comparison based diagnostic model based on the one-to-many communication paradigm. Two implementations of the model are shown. In the first model, we assume that the network topology does not change during diagnosis, and we show that both hard and soft faults can be easily detected. Based on this implementation, a diagnosis protocol is presented. The evaluation of the communication and time complexity of the protocol indicates that efficient diagnosis protocols for ad-hoc networks based on this model can be designed. In the second implementation we allow the system topology to change during diagnosis. We show that the ability of diagnosing faults under this scenario decreases; meaning that mobility significantly reduces the "quality" of the diagnosis returned by a diagnosis protocol. We will deal with two types of fault:

- Hard faults
- Soft faults

Hard faults are when the node doesn't respond at all within a certain time (time out) which is decided on basis of the network such that all nodes can be tested, whereas soft faults are those in which the system response is not valid.

Chapter

8 Algorithm for Ad-hoc Diagnosis

8.1 Participants

The system is composed of n mobile hosts, henceforth called *mobiles*, which communicate via packet radio network [2]. The topology of the system at time *t* can be described as a directed graph $G(t) = (V, L(t))$, called the *communication graph*, where

- V is the set of nodes, denoting mobiles.
- $L(t)$ is the set of *logical links* at time *t*.
- Given any $(u, v) \epsilon V$, edge $(u, v) \epsilon L(t)$ if and only if *v* is in the transmitting range of *u* at time *t*.
- *neighbor set of u at time t*, denoted $N(u, t)$

The following function Ad-Hoc() describes the diagnostic algorithm.

8.2 Function Ad_hoc (*n, t, T*out)

- **for each** u in Graph
- Generate N(u, t)
- To_be_tested_node u = Random(all nodes)
- for each v ϵ N(u, t) , Generate Request token
- flood to all neighbors
- Wait for Test Token from all v
- set the timer to Tout; {set the timeout for hard faults detection}
- All v ϵ N(u, t) generate a Task set and send to u
- Wait for answer token message from u;
- validate answer token and set own Fault_Vector
- select another node to be tested;

The following function Fault_Vector() describes the generation of fault vector.

8.3 Function Fault_Vector (n)

- For each slot i in Fault Vector // i corresponds to slot for node n_i
- If(Test Pass)
- Slot= 1;
- Else
- Slot=0
- Otherwise slot='X'

Chapter

9 Implementation of Ad-hoc Diagnosis

9.1 Conditions

We follow the mentioned process:

The system is considered of n nodes, which communicate via a radio packet network. The topology of system at time t is described as a directed graph G(t)=(V,L(t)),where V is nodes and L is link. We here form a neighbor set N (u, t) such that the neighbors are the ones which are in transmitting range of u at time t.

9.2 Assumptions:

All the nodes have information about their neighboring nodes.

The protocol provides a 1_hop reliable broadcast primitive.

The receiver of a message knows the identity of the sender [2][1].
Each node is connected to atleast one fault-free node.

9.3 Proposed Algorithm:

At time t a node broadcasts a message that it wants to be tested. The token format is same as per our fault diagnosis algorithm. The nodes receiving it forms a neighboring network in which the test has to be carried out. Then randomly 2 nodes (min) are selected from the neighboring nodes to test the node. The result of the test on the basis of neighboring nodes is saved and a fault vector of this network is formed. Similarly other nodes which are not being tested broadcasts a message to be tested and fault vector after all nodes are being tested is formed. This is then used to give the final results showing faulty nodes. The nodes which didn't broadcast or reply to any queries are termed as hard faulted nodes [2][1].

9.4 Complicacy of the algorithm.

In the above algorithm the one major complicacy that lies is the minimum nodes testing a node, The more the no of nodes testing a node more is the complexity but more is the accuracy as well, So, the no of nodes testing a node depends on the quality of a network. The complexity can further be reduced by using a permanent test token for a particular period, in this case the nodes once receiving the token can even test themselves and compare the result to know whether they are faulty.

9.5 Implementation of above algorithm in dynamic ad-hoc network

Since the topology varies with time in general we have N (u, t)! =N (u, t+tout). As a result hard faulted units that migrate out of the testing unit's transmitting range cannot be distinguished from fault free units that migrated out of the range. At time t+tout the units whose test result values are not contained in result token are classified as time-outed units [1][2].

9.6 Algorithm:

In this, we take one assumption that the nodes can't migrate in between the testing. They can migrate only before or after a test has started or finished.

All the steps are similar to that of fixed ad-hoc network, except that:

Consider a set N_S (u) =N (u, t) ∩ N (u, t=tout) and let N^r_S (u) be the set of fault-free or soft faulted units in N_S (u)

Here also we have to consider an assumption that the units which are present in the testing units transmission range at time t and tout were also present at time t'(t'<t+ tout) when the test token was issued because if a unit is in N^r_S(u) does not implies that it received the test token as it could have been out of transmitting range at t' when token was issued and come back in network again later [1][2].

9.7 Complicacy of the algorithm

In above algorithm the units which doesn't reply within tout cannot be classified as hard faulted nodes so the test has to run again and again so that fault vector of all the nodes are collected however, due to time-varying topology of the network, the exact no of such diagnostic is hard to quantify, unless some restriction on the mobility of the units are imposed.

Chapter

10 Results for Ad-hoc

The above algorithm was implemented for ad-hoc networks and following results were found. It was observed that as in wired network, diagnostic latency increases with increase in nodes.

10.1 GRAPH 3 No of Nodes Vs Diagnostic Latency

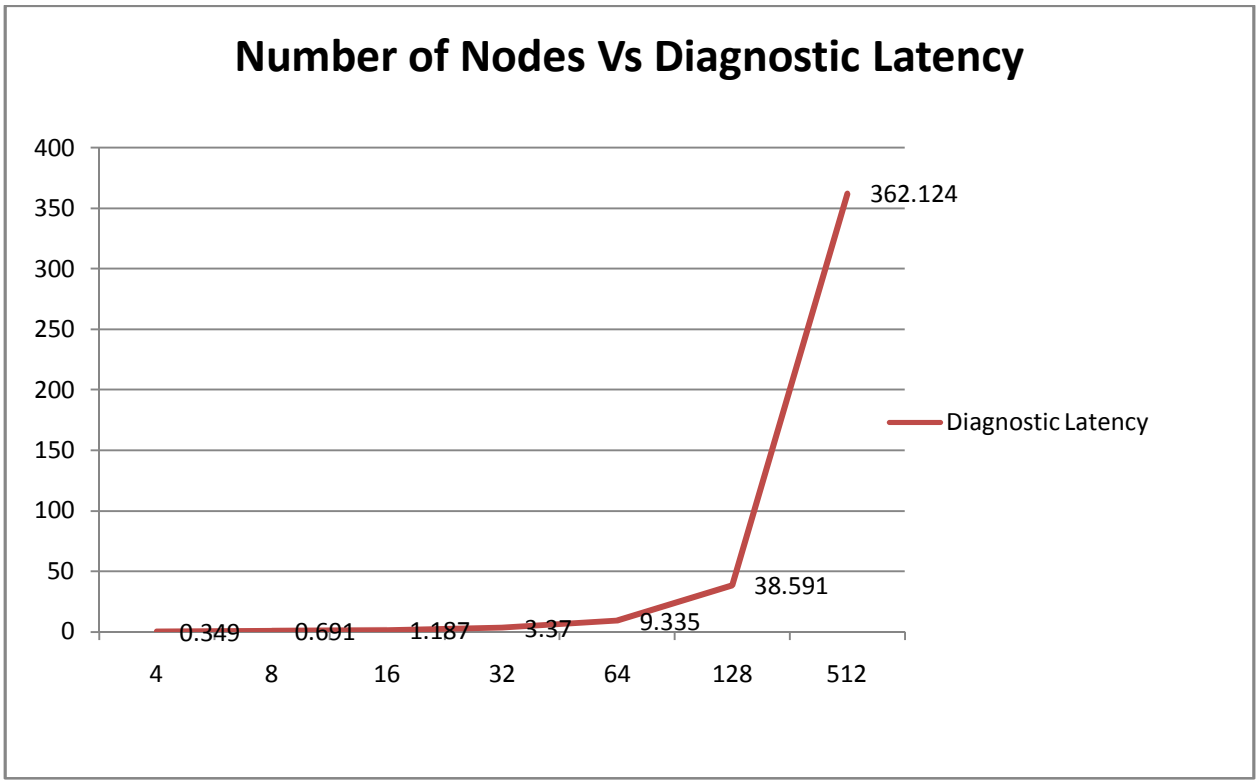

Figure 10.1 Diagnostic Latency for Ad-hoc Network

10.2 GRAPH 4 Comparison of Wireless and Wired networks

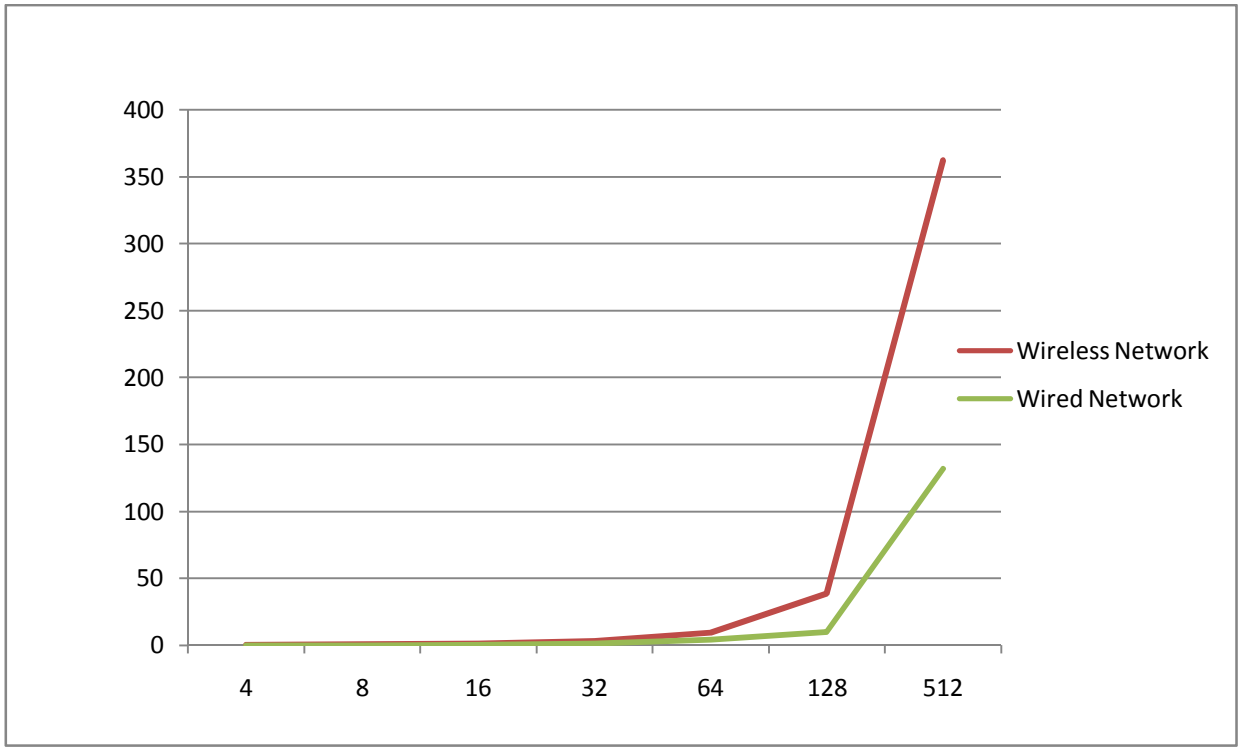

Figure 10.2 Comparison between wired and ad-hoc network in terms of diagnostic latency

A comparison between diagnostic latency in wired and wireless network is made and it is found that diagnostic latency in wireless connection is much more than that in wired networks

Chapter

11 Conclusion

11.1 Conclusion

The distributed system-level diagnosis algorithm for fault detection is implemented on Dynamic Positioning System and ad-hoc networks. For Dynamic Positioning System the complexity can be decreased by making a hypercube network between the communicating nodes. In ad-hoc networks two implementations of the model were done, one with fixed and other dynamic ad-hoc network. Our algorithm works fine with the first case of fixed network and results obtained were satisfactory, whereas the diagnosis is extremely hard in case of dynamic system unless some restrictions on the mobility of the nodes are implemented, further research has to be done in order to deal with this problem.

11.2 Future work

The above analysis can further be extended to find more suitable and better algorithm for fault detection in ad-hoc networks, with much more precision in handling dynamic nodes thus rendering it more useful in case of battle field and other areas where fault in any system is just not accepted.

Chapter

12 References

[1] Hongian Wang, Xinqian Bian, Fuguang Ding, Guiping Han 'The Application of a distributed system-level diagnosis algorithm in dynamic positioning system' Proceedings of the 4' World Congress on Intelligent Control and Automation June 10-24, 2002, Shanghai, P.R.China

[2] Stefano Chessa1 and Paolo Santi2,' Comparison-Based System-Level Fault Diagnosis in Ad-Hoc Networks'1 Dipartimento di Informatica, Univ. of Pisa, Italy2 Dept. of Elec. and Comp. Eng., Georgia Institute of Technology, Atlanta, GA 30332-0280

[3] Ronald Bianchini, Jr.Richard Buskens 'An Adaptive Distributed System-Level Diagnosis Algorithm and Its Implementation' CH2985-0/91/0000/0222 1991 IEEE Carnegie Mellon University Department of Electrical and Computer Engineering Pittsburgh, Pennsylvania 15213(412) 268-7105

[4] Ansley Post abpost@rice.edu 'Towards a scalable ad hoc network infrastructure' Rice University, Houston, TX, USA

[5] Lakshmi Prasad Saikia*, Kattamanchi Hemachandran**, *Senior Lecturer, CSE, Deptt of ETE, NIT Silchar, Assam, India. Email: lp_saikia@yahoo.co.in **Professor,CS, AssamUniversity-Silchar, Assam,India. Email: khchandran@rediffmail.com 'SIMULATION OF SYSTEM LEVEL DIAGNOSIS IN DISTRIBUTED ARBITRARY NETWORK' Journal of Theoretical and Applied Information Technology© 2007 JATIT

[6] Stephen Mueller1, Rose P. Tsang2, and Dipak Ghosal1 1 Department of Computer Science, University of California, Davis, CA 95616 2 Sandia National Laboratories, Livermore, CA 94551 'Multipath Routing in Mobile Ad Hoc Networks:Issues and Challenges'

[7] Lidong Zhou Department of Computer Science Zygmunt J. Haas School of Electrical EngineeringCornell University Ithaca, NY 14853 'Securing Ad Hoc Networks' published in IEEE network, special issue on network security, November/December, 1999.

[8] Anmol Sheth, Carl Hartung and Richard Han Department of Computer Science University of Colorado, Boulder 'A Decentralized Fault Diagnosis System for Wireles Sensor Networks' 0-7803-9466-6/05/ ©2005 IEEE

[9] Jianer Chen Dept. of Computer Science Texas A&M University College Station, TX 77843 chen@cs.tamu.edu Iyad A. Kanj School of CTI DePaul University Chicago, IL 60604 ikanj@cs.depaul.edu Guojun Wang College of Information Eng. Central-South University ChangSha, Hunan 410083, China Gordon434@163.netHypercube Network Fault Tolerance: A Probabilistic Approach_0-7695-1677-7/02 © 2002 IEEE

[10] Anindo Bagchi Bellcore Red Bank, NJ 07701 *S.* L. Hakimi Dept. of Elect. Engr. & Comp. Sci. University of Califomia Davis, CA 95616 An Optimal Algorithm For Distributed System Level Diagnosis CH2985-0/91/0000/0214 1991 IEEE

[11] F. P. Preparata, G. Metze, and R. T. Chien, "On the connection assignment problem of diagnosable systems", IEEE Trans. Comput.,vol. EC-16, pp. 848-854, Dec. 1967.