CRASH FAULT IDENTIFICATION OF A K-CONNECTED NETWORK IN STATIC FAULT ENVIRONMENT

Rajeeb Kumar Panigrahi(10506061)

Sourav Kumar Giri(10506030)

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela–769 008, Orissa, India

2009

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology

In

Computer Science And Engineering

BY

Rajeeb Kumar Panigrahi(10506061)

Sourav kumar Giri(10506030)

Under the Guidance of Prof. P.M.Khilar

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela

2009

National Institute of Technology Rourkela

CERTIFICATE

This is to certify that the thesis entitled, "**Crash fault identification of k-connected network in static fault environment**" submitted by **Sourav kumar giri(10506030)** and **Rajeeb kumar panigrahi(10506061)** in partial fulfillment of the requirements for the award of Bachelor of Technology Degree in Computer Science And Engineering at the National Institute Of Technology, Rourkela (Deemed University) is an authentic work carried out by them under my supervision and guidance.To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University/Institute for the award of any Degree or Diploma.

Prof. P.M Khilar

Dept. Of Computer Science & Engineering

National Institute of Technology , Rourkela

Date:

ACKNOWLEDGEMENT

We welcome this opportunity to express our heartfelt gratitude and regards to our project guide **Prof. P.M.Khilar** , Department of Computer Science And Engineering, National Institute of Technology, Rourkela for his superb guidance. He always bestowed parental care upon us and evinced keen interest in solving our problems. An erudite teacher, a magnificent person and a strict disciplinarian, we consider ourselves fortunate to have worked under his supervision.

We are highly grateful to **Prof. B.Majhi**, Head of the department, Computer Science And Engineering, NIT Rourkela, for providing necessary facilities during the course of the work.

Rajeeb Kumar panigrahi

Sourav kuamr giri

CONTENTS

ABSTRACT

The problem of distributed diagnosis in the presence of dynamic failures and repairs is considered. To address this problem, the notion of bounded correctness is defined. Bounded correctness is made up of three properties: bounded diagnostic latency, which ensures that information about state changes of nodes in the system reaches working nodes with a bounded delay, bounded start-up time, which guarantees that working nodes determine valid states for every other node in the system within bounded time after their recovery, and accuracy, which ensures that no spurious events are recorded by working nodes. It is shown that, in order to achieve bounded correctness, the rate at which nodes fail and are repaired must be limited. This requirement is quantified by defining a minimum state holding time in the system. Algorithm HeartbeatComplete is presented and it is proven that this algorithm achieves bounded correctness in fully-connected systems while simultaneously minimizing diagnostic latency, start-up time, and state holding time. A diagnosis algorithm for arbitrary topologies, known as Algorithm ForwardHeartbeat, is also presented.ForwardHeartbeat is shown to produce significantly shorter latency and state holding time than prior algorithms, which focusedprimarily on minimizing the number of tests at the expense of latency.

LIST OF SYMBOLS

Symbol	Description
Dsend_init	Send initiation time
Dsend_min	Minimum message delays
Dsend_max	Maximum Message Delays
Dmaxn	Upper bound on communication delay
K	Connectivity of the Network
N	No of Nodes
Treject	Rejection time
Ttimeout	Timeout period
Texist	Time period for which a heartbit exit in the network.
P	Clock drift
C	Small Constant

INTRODUCTION

The problem of distributed diagnosis in the presence of dynamic failures and repairs is considered. To address this problem, the notion of bounded correctness is defined. Bounded correctness is made up of three properties: bounded diagnostic latency, which ensures that information about state changes of nodes in the system reaches working nodes with a bounded delay, bounded start-up time, which guarantees that working nodes determine valid states for every other node in the system within bounded time after their recovery, and accuracy, which ensures that no spurious events are recorded by working nodes. It is shown that, in order to achieve bounded correctness, the rate at which nodes fail and are repaired must be limited. This requirement is quantified by defining a minimum state holding time in the system. Algorithm HeartbeatComplete is presented and it is proven that this algorithm achieves bounded correctness in fully-connected systems while simultaneously minimizing diagnostic latency, start-up time, and state holding time. A diagnosis algorithm for arbitrary topologies, known as Algorithm ForwardHeartbeat, is also presented. ForwardHeartbeat is shown to produce significantly shorter latency and state holding time than prior algorithms, which focused primarily on minimizing the number of tests at the expense of latency.

AN important problem in distributed systems that are subject to component failures is distributed diagnosis problem. In distributed diagnosis, every node must maintain the correct and timely information about the status (working or failed) of all the nodes in the system. In dynamic fault environment, the nodes may change their status during execution of the diagnosis procedure.

The bulk of the work in system diagnosis has assumed a static fault situation, i.e., the statuses of nodes do not change during execution of the diagnosis procedure. Some works have considered the dynamic situations but with assumptions such as existence of centralized diagnosis entity and regular network topology. The relevant algorithm such as Hi-ADSD and its variants have latencies of at least (log22n) rounds. However, these algorithms do not allow both failure and recovery events in dynamic fault environment and assumes a fully connected network. Recently, in the authors have proposed an algorithm in dynamic fault environment but assumes crash fault model. However, the authors have not considered more realistic fault model of value faults in nodes which may frequently occur in runtime due to incorrect computations while executing the distributed workload such as clock monitoring process, load balancing process etc. Here, in this paper, the problem of distributed diagnosis for not completely connected networks (DDNCN) in dynamic fault environment under more realistic fault models such as crash and value fault has been investigated. The algorithm works for any number of nodes can change their state during the execution of the algorithm provided the network remains connected and also there is a limit on how frequently an individual node can change state.

SYSTEM AND FAULT MODEL

We consider not-completely connected networks where intermediate nodes relay messages between some sourcedestination pairs. The number of node failures is limited such that the network remains connected at all times. Diagnosis algorithm can use either unicast or multicast communication.We assume the generic parameters to support this type of communication. We also assume a synchronous system in which the communication delay is bounded. This is an implicit assumption in all prior work on distributed diagnosis. Nodes directly connected by a communication link are called neighbors.

Definition 1. The send initiation time dsend init, is the time between a node initiating a communication and the last bit of the message being injected into the network.

Definition 2. The minimum and maximum message delays, dsend min and dsend mcax, are the minimum and maximum times, respectively, between the last bit of a message being injected into the network and the message being completely delivered at a working neighboring node.

FAULT MODEL

We consider crash and value faults in nodes. Links are assumed to be fault free. The network delivers messages reliably. A faulty node perform it's computation like a nonfaulty node but it may fail to send its value (crash fault) or it may send an erroneous value to another node (value fault). The heartbeat based testing mechanism is followed to detect the faults in nodes. The node whether crash or value faulty will not be used for relaying the heartbeats. The status of a node is modeled by a state machine with two states, failed and working (i.e., 1 or 0). Working nodes execute the normal workload and diagnosis procedure.

Definition 3. The state holding time is the minimum time that a node remains in one state before transitioning to the other state.

Definition 4. We define the connectivity of the network k is the minimum number of nodes, the removal of which can cause the network to become disconnected.

ALGORITHM ASSUMPTION

A restricted case is considered where fewer than k nodes are in the failed state at any given instant of time so that the network remains connected. Each node periodically executes the normal workload and initiates a round of message transmissions to other nodes in order to indicate that it is working. Assume an arbitrary node X initiates a round of heartbeat transmissions at real time t and remains in the working state indefinitely afterward. X will initiate another round of heartbeat transmissions no later than real time t+(l+p)2t, where 21 is the heartbeat period and p is the maximum drift rate of the clock with p << 1.

TIMES AND CLOCK MODELS

We consider the notion of time as we are interested in dynamic failure situations in which failure and recovery timing is critical.

Definition 5: Time that is measured in an assumed Newtonian time frame (which is not directly observable) is referred to as real time.

Definition 6: Time that is directly observable on a node's clock is referred to as clock time. The clock time of node X at real time t is denoted by Tx(t).

Definition 7: While a node is in the working state, it's clock experiences bounded drift. This means that if a node X is in the working state continuously during a real-time interval[t1,t2], then for all real-time intervals [u1,u2] c [t1,t2]

I[Tx(u2)- Tx(ul)]- (u2 - u1) < p(u2 - u1), where p << 1 is the maximum drift of a clock per unit time.

ALGORITHM

Each node X executes one frame of the workload arriving at some value Val(X). Each node broadcasts Val(X) to all working neighboring nodes using a heartbeat; Each working neighboring node rebroadcasts and so on

```
If (node[an intermediate node].status == working)

Relay the heartbeats;
Replaces old Seq_no by Latest Seq_no
```

```
received;

heartbeat delay=

buffered delay + (dsend init +dsend_min + c);

A working node keeps the minimum time the heartbeats stored in it's buffer;

Otherwise,

Will not relay the heartbeats;

Each node X receives incoming messages from every other node Y and compares
with its computed value:

If (Val(X) <> Val(Y) or times-out)
Record node[Y].status= 1;
Removes the heartbeat from it's buffer;
Discard any stale heartbeats coming
through different paths for a period of time
```

```
Treject (heartbeat rejection time);

Otherwise,

Record node[Y].status = 0;
Record the last sequence number;

If (a node is a newly-recovered node)

Seq_no & heartbeat.delay = 0 (during initiation);
Sends a heartbeat message to neighboring nodes;
Collect the buffered heartbeats from it's neighboring node;

//this ensures propagation of heartbeats in a dynamic fault environment;
```

DESCRIPTION OF ALGORITHM

The pseudo-code for the algorithm DDNCN is given above. Each node periodically executes a workload and sends the resultant value using a heartbeat message to a subset of neighboring nodes. The neighboring nodes in turn send heartbeats to their neighbors and so on.

A heartbeat message consists of the following fields:

Node_id: The ID of the node that initiated the heartbeat.

Seq_no: The physical sequence number of the heartbeat.

Val : Value associated with the message.

Delay: The minimum time the heartbeat message was in the network before being received.

The heartbeats are propagated throughout the network. When node X receives a new heartbeat from node Y, node X stores the heartbeat in a buffer replacing any older heartbeats from node Y. If node X times out waiting for node Y's next heartbeat, then node Y's heartbeat is removed from node X's buffer and node Y is diagnosed as faulty by node X. Hence, the presence of node Y's heartbeat in node X's buffer indicates node X believes that node Y is working. If a neighbor of node X recovers, then node X sends the newly recovered neighboring node all heartbeats stored in its buffer. This ensures propagation of heartbeats in a dynamic fault environment.

When a node Y initiates a new heartbeat, it initializes the delay field to the minimum delay that will be encountered before the heartbeat messages could reach its neighboring nodes and then sends them out. Also, at the same time, node Y stores this new heartbeat in its local buffer with the delay field set to zero. Nodes keep track of the amount of time each heartbeat is stored in their buffers. For a heartbeat stored locally in the originating node, the delay field will always be 0. When a node retransmits or relays a heartbeat, it adds to the delay field the length of time the heartbeat was stored in its buffer and the minimum time it takes to traverse the next hop to reach the neighboring node before sending it out. Thus, a node keeps track of the minimum length of time the heartbeats stored in its buffer have existed in the network.

ANALYSIS OF ALGORITHM

First we compute the maximum time between a node initiating a heartbeat and a heartbeat with the same or a higher sequence number from the same node being received by all other nodes that are working since heartbeat initiation, denoted by Dmaxn. This is derived by taking a particular sequence of events considering the worst case network scenario. Dmaxn is the upper bound on the communication delay and is d(k-1)(n-I)dsend init + (n+k-2)(d send-init + d send-max + c) - (k-1)C, if the failed state holding time, SHTf is at least Dmaxn Where, d is the maximum number of neighbors of any node, k is connectivity of the network, n is number of nodes, £ = is a small positive constant and is the smallest time such that a heartbeat send in dsend-init - , time is an invalid heartbeat and c is the comparison time between value associated with the heartbeat message and the node's own computed value.

Due to the existence of multiple paths between nodes, it is possible that a node times out thereby detecting a fault event and then receives a stale heartbeat. Arrival of this stale heartbeat does not indicate a recovery event. Hence, when nodes time out, they discard any heartbeats they may receive from the node just diagnosed to be faulty for a predetermined amount of time called the heartbeat rejection time (Treject). As a result, for a genuine recovery event to be detected, the failed state holding time should be made sufficiently large to guarantee that no new heartbeat message arrives during the rejection time. In order to compute failed state holding time, it is necessary to find the maximum possible time a heartbeat can exist in the network.

Using Dmaxn we derive time out period (Ttime-out), maximum possible time a heartbeat can exist in the network (Texist), heartbeat rejection time (Treject) and failed state holding time (SHTf) as follows:

Lemma 1. Upon receipt of a new heartbeat, the period of time a node that is continuously in the working state waits for the next new heartbeat before detecting a fault event, called the timeout period Ttimeout is (1+2p)n + (l+p)(Dmaxn - heartbeat.delay) where heartbeat.delay is the value in the delay field of the last heartbeat received.

Lemma 2. The maximum possible time a heartbeat can exist in the network, denoted by Texist, is (1+2p)n + (l+p) Dmaxn + n(dsend-max - dsend-min + c). A heartbeat is said to exist in the network if either the heartbeat is propagating in the network or the heartbeat is stored in some node's buffer.

Lemma 3. The heartbeat rejection time, denoted by Treject, is the period of time a node X discards heartbeats from a node Y after diagnosing node Y to be faulty. The heartbeat rejection time (Treject) = 2pm + 2pDmaxn + n(l+p)(dsend-maxdsend-min + c).

Lemma 4. The failed state holding time, denoted by SHTf, is the minimum time a node remains in the failed state before transitioning to the working state. The failed state holding time (SHTf) is Texist + (l+p)Treject - dsend-init.

The algorithm DDNCN maintains a data structure containing fields such as node id, heartbeat delay, the status and the value of all nodes and propagates the heartbeats by flooding to all nodes via intermediate nodes. The maximum time between a node initiating a message and a message with the same or a higher sequence number from the same node being received by all nodes that are working since message initiation, denoted by Dmaxn is d (k-1)(n-1) d send_init + (n +k–2)(dsend_init + dsend_max + c) – (k-1) ε, where ε is anarbitrary small positive constant, if the failed state holding time, SHTf, is at least Dmaxn and c is the comparison time between the value associated with heartbeat and the node's own computed value. Dmaxn is the upper bound on the communication delay and is the worst-case latency in detecting recovery event. When nodes time out, they discard the arrival of any stale heartbeats they may receive from the node just diagnosed to be faulty for a predetermined amount of time called the heartbeat rejection time (Treject). As a result, for a genuine recovery event to be detected, the failed state holding time should be made sufficiently large to guarantee that no new heartbeat message arrives during the rejection time and depends on maximum possible time a heartbeat can exist in the network. Using Dmaxn we derive time out period (Ttime-out), maximum possible time a heartbeat can exist in the network (Texist), heartbeat rejection time (Treject) and failed state holding time (SHTf). Upon receipt of a new heartbeat, the period of time a node that is continuously in the working state waits for the next new heartbeat before detecting a fault event, called the timeout period Ttimeout is (1+2ρ)π + (1+ ρ)(Dmaxn – heartbeat.delay) where heartbeat. delay is the value in the delay field of the last heartbeat received. Using Ttimeout we derive Texist, Treject and SHTf. It can be seen that Texist is the worst-case latency in detecting failure event. The algorithm DDNCN achieves bounded correctness with a diagnostic latency of Texist – dsend_init.

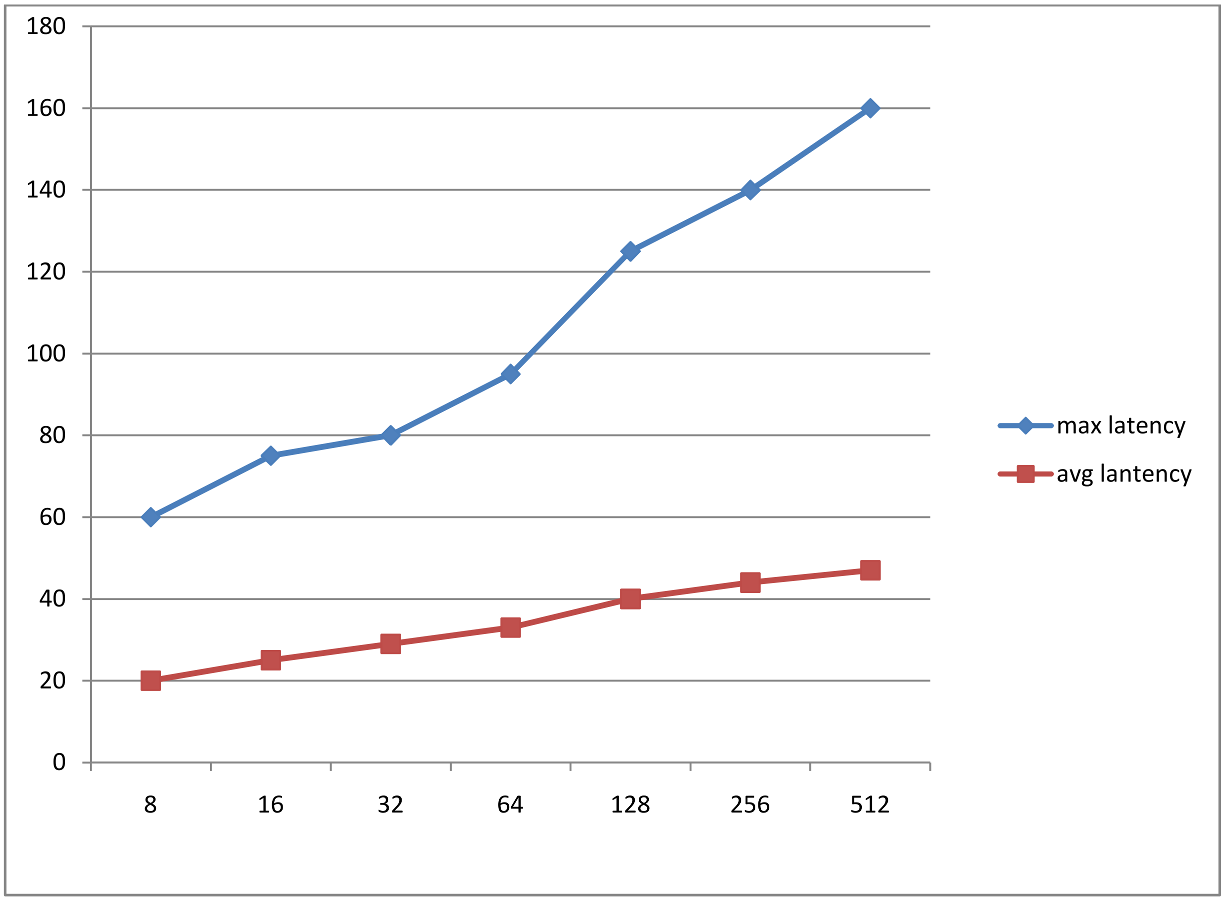
SIMULATION RESULT

Algorithm was simulated on randomly generated networks. dsend init, dsend min, dsend mcax, c, Poisson mean, p ,k and z were kept fixed at 0.002, 0.008, 0.08, 0.05, 1, 0.001,3 and 60 seconds, respectively, for all simulations. Simulations were performed on networks of sizes 8, 16, 32, 64, 128, and 256. Simulations were done using discrete event simulation techniques. The dynamic nature of the system was modeled using a Poisson process. When an event occurs on a node, the time at which the next event occurs on the same node is the state holding time for the current state plus an additional time as given by the Poisson process. If a failure event is not possible to occur because the number of failed nodes is greater than (k-1), then the failure event is rescheduled to a later time again according to a Poisson process. In all simulations, the minimum state holding times for both failed and working states were set to the failed state holding time.

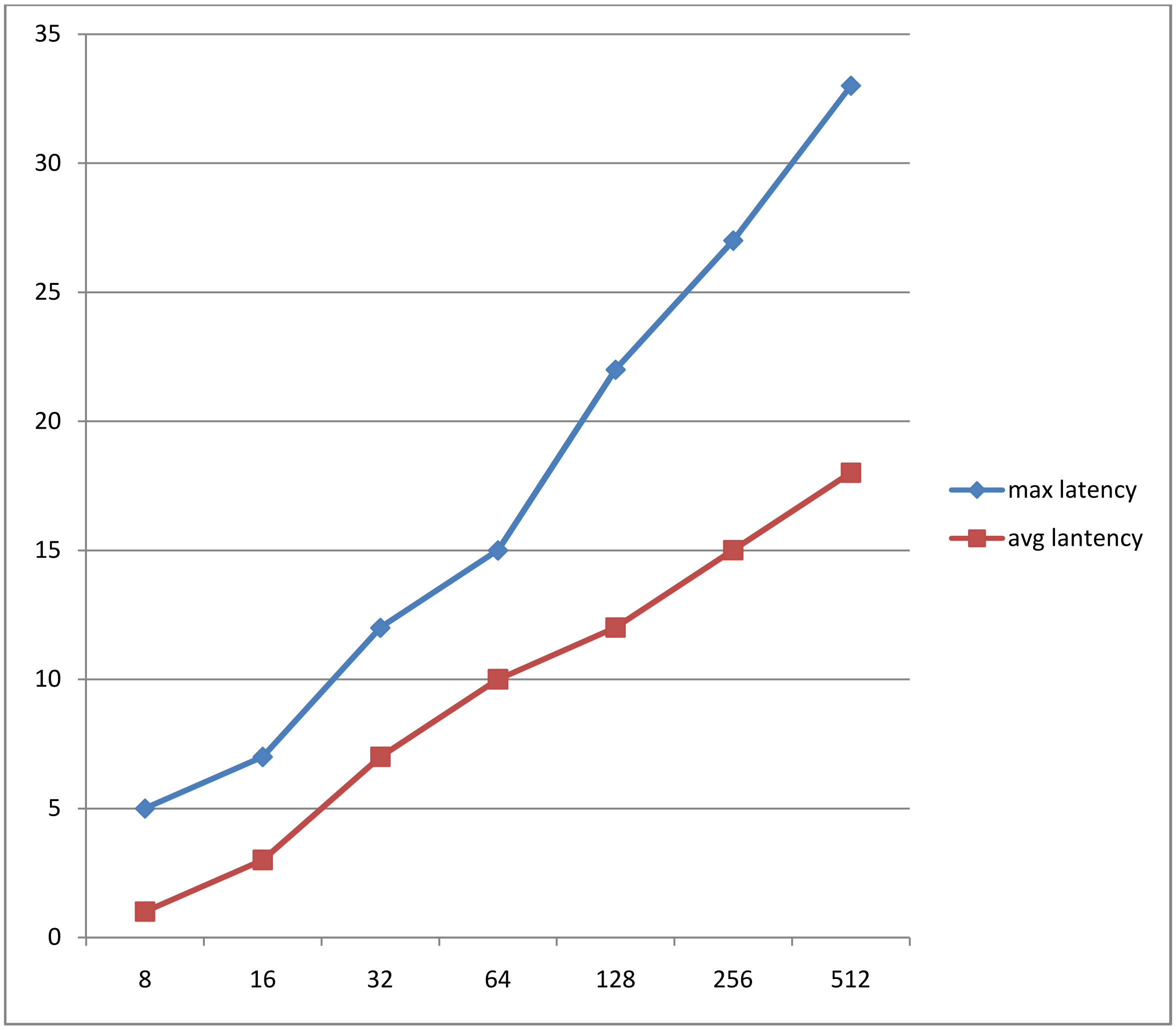
Graphs were randomly generated for a given n and k. Since every node must have k neighbors, links were randomly introduced such that every node has at least k neighbors. If the connectivity of the network is less than k, then n links are randomly introduced into the network. This process is continued until the resulting connectivity of the network is at least k. All graphs generated this way had a connectivity exactly equal to k. Simulations were carried for different values of k. To investigate the suitability of the networks, five different networks were generated for every value of (n, k).

LATENCY IN DETECTING FAILURE/RECOVERY EVENT

Figure 1 and 2 shows the latency in detecting failure and recovery events for algorithm DDNCN. The latency in detecting a failure event is variable due to the difference in the actual propagation time of the last heartbeatand the delay tracked by the intermediate nodes. Hence, the more links that are traversed by the last heartbeat that is sent by a node before it could fail, the more varied the latency will be. The average latency in detecting a failure event for DDNCN is lower. This is because the latency is lower for value fault than crash fault.



NO OF NODES VS LATENCY IN DETECTING FAILURE EVENT

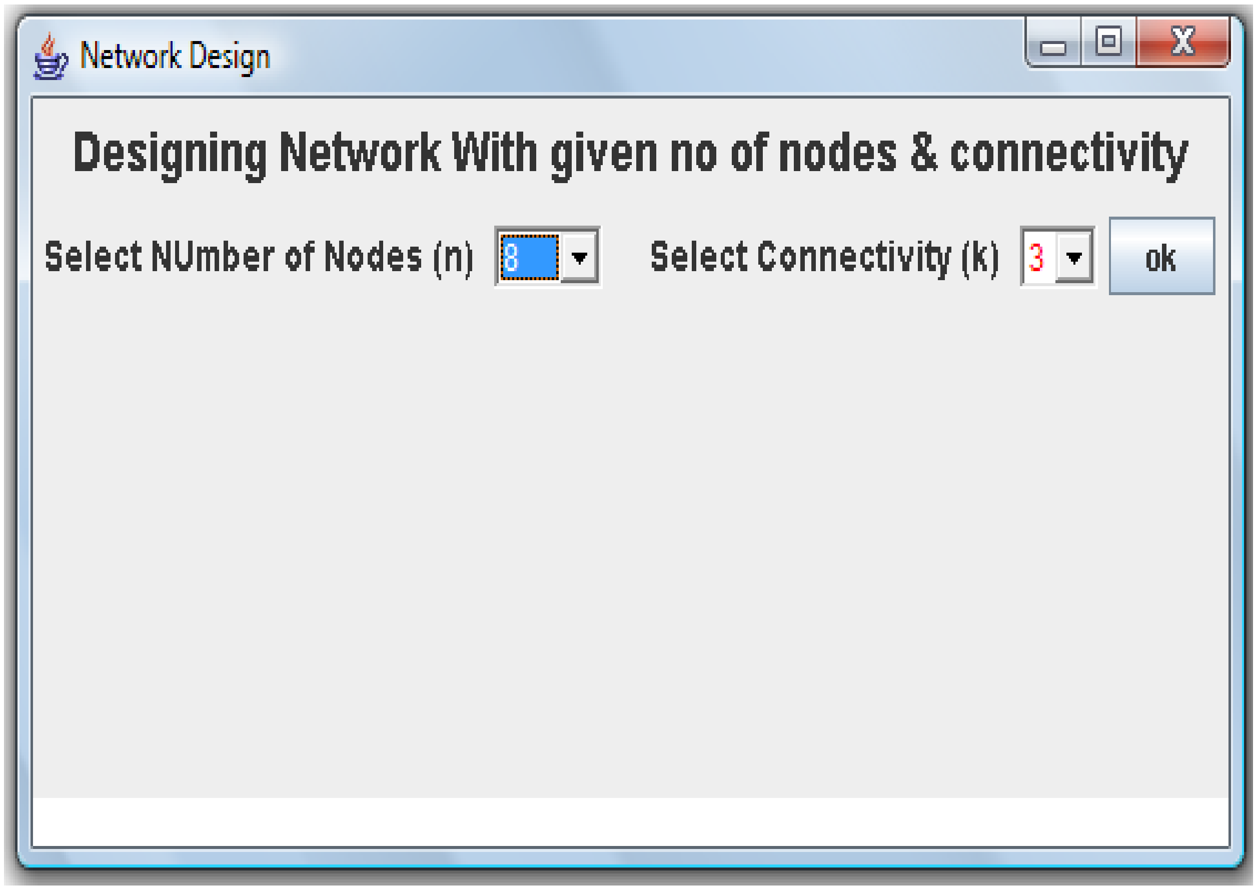


NO OF NODES VS LATENCY IN RECOVERING EVENT

ALGORITHM FOR RANDOM NETWORK GENERATION

1. Generate two random numbers(provided both are not same) for a predefined no of times ,where 1st random number act as Source Node and the 2nd random no act as Destination Node. Mark the edge value between source and destination node as 1(i.e. connection is there).

2. Check the Connectivity of the network designed in step-1. If connectivity satisfied it is a k-connected network and exit. If connectivity does not satisfied continue step1 for a predefined no of times.

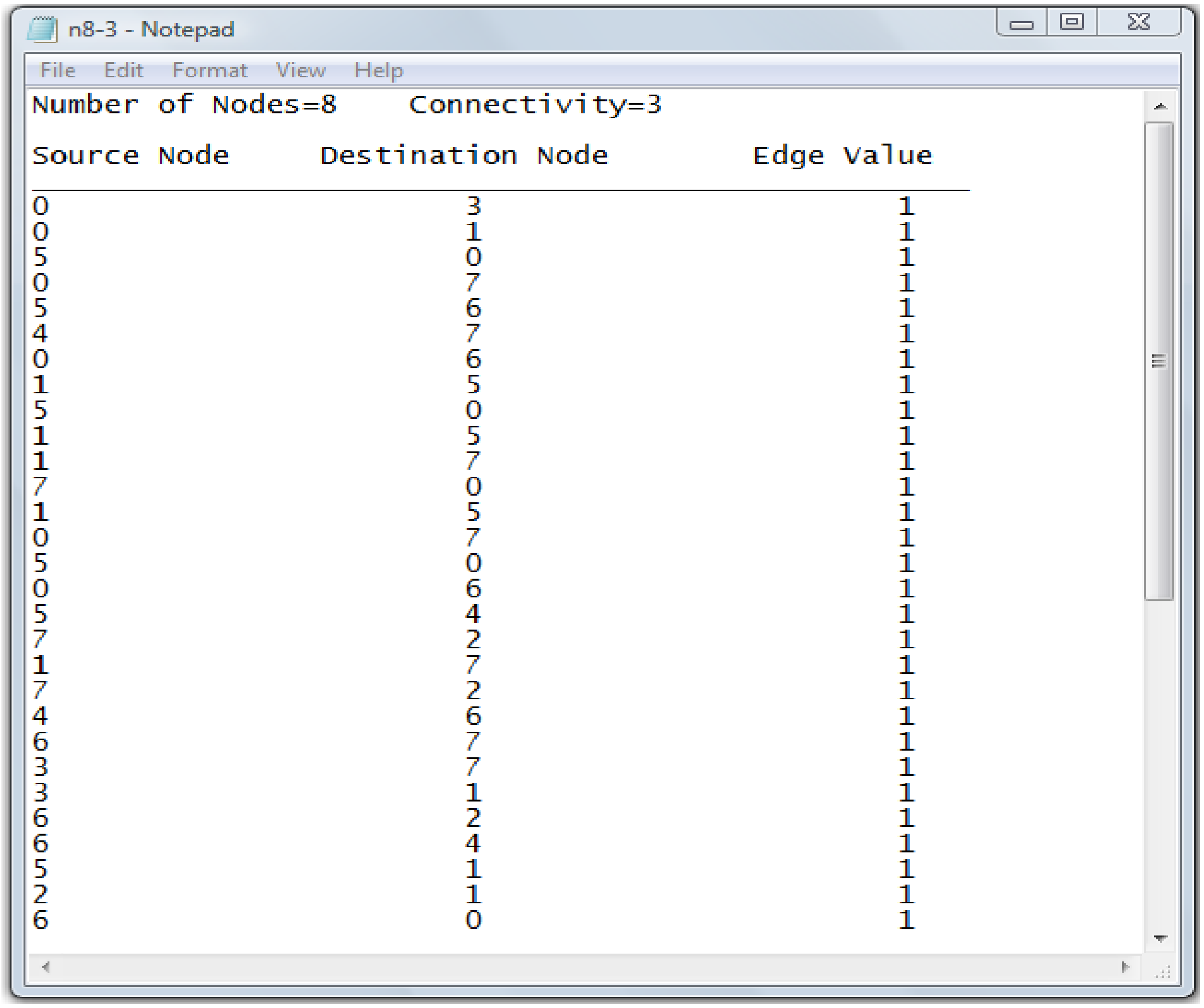

Network Design

Designing Network With given no of nodes & connectivity

Select NUmber of Nodes (n) 8

Select Connectivity (k) 3

NETWORK WITH N=8 AND K=3



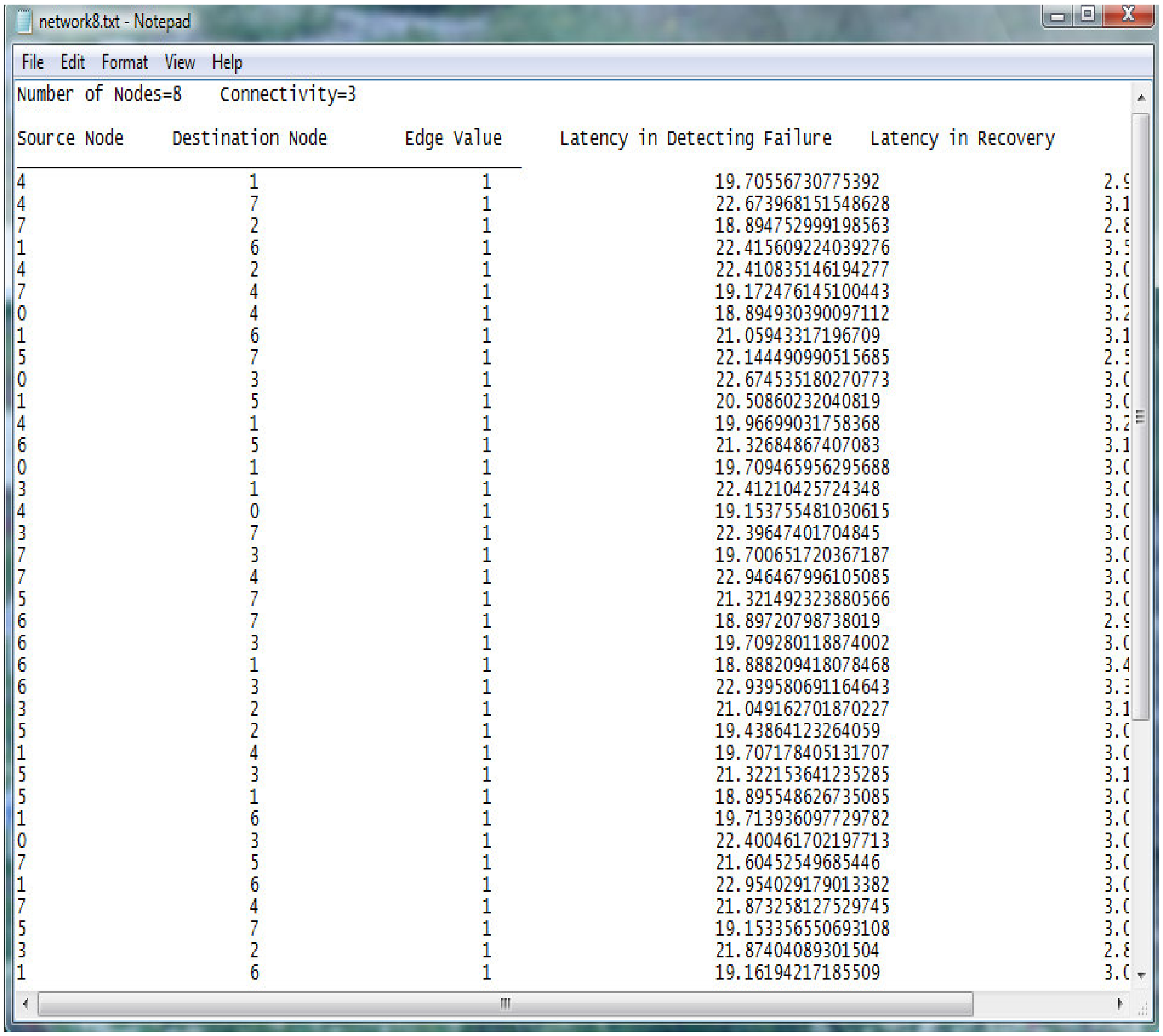
n8-3 - Notepad

File Edit Format View Help

```
Number of Nodes=8     Connectivity=3

Source Node      Destination Node          Edge Value
_____________________________________________________
0                        3                        1
0                        1                        1
5                        0                        1
0                        7                        1
5                        6                        1
4                        7                        1
0                        6                        1
1                        5                        1
5                        0                        1
1                        5                        1
1                        7                        1
7                        0                        1
1                        5                        1
0                        7                        1
5                        0                        1
0                        6                        1
5                        4                        1
7                        2                        1
1                        7                        1
7                        2                        1
4                        6                        1
6                        7                        1
3                        7                        1
3                        1                        1
6                        2                        1
6                        4                        1
5                        1                        1
2                        1                        1
6                        0                        1
```

LATENCY IN DETECTING FAILURE AND IN RECOVERY



network8.txt - Notepad

File Edit Format View Help

```
Number of Nodes=8     Connectivity=3

Source Node     Destination Node       Edge Value     Latency in Detecting Failure     Latency in Recovery
_______________________________________________________
4                    1                    1                   19.70556730775392                     2.9
4                    7                    1                   22.673968151548628                    3.1
7                    2                    1                   18.894752999198563                    2.8
1                    6                    1                   22.415609224039276                    3.5
4                    2                    1                   22.410835146194277                    3.0
7                    4                    1                   19.172476145100443                    3.0
0                    4                    1                   18.894930390097112                    3.2
1                    6                    1                   21.05943317196709                     3.1
5                    7                    1                   22.144490990515685                    2.5
0                    3                    1                   22.674535180270773                    3.0
1                    5                    1                   20.50860232040819                     3.0
4                    1                    1                   19.96699031758368                     3.2
6                    5                    1                   21.32684867407083                     3.1
0                    1                    1                   19.709465956295688                    3.0
3                    1                    1                   22.41210425724348                     3.0
4                    0                    1                   19.153755481030615                    3.0
3                    7                    1                   22.39647401704845                     3.0
7                    3                    1                   19.700651720367187                    3.0
7                    4                    1                   22.946467996105085                    3.0
5                    7                    1                   21.321492323880566                    3.0
6                    7                    1                   18.89720798738019                     2.9
6                    3                    1                   19.709280118874002                    3.0
6                    1                    1                   18.888209418078468                    3.4
6                    3                    1                   22.939580691164643                    3.3
3                    2                    1                   21.049162701870227                    3.1
5                    2                    1                   19.43864123264059                     3.0
1                    4                    1                   19.707178405131707                    3.0
5                    3                    1                   21.322153641235285                    3.1
5                    1                    1                   18.895548626735085                    3.0
1                    6                    1                   19.713936097729782                    3.0
0                    3                    1                   22.400461702197713                    3.0
7                    5                    1                   21.60452549685446                     3.0
1                    6                    1                   22.954029179013382                    3.0
7                    4                    1                   21.873258127529745                    3.0
5                    7                    1                   19.153356550693108                    3.0
3                    2                    1                   21.87404089301504                     2.8
1                    6                    1                   19.16194217185509                     3.0
```

CONCLUSION AND FUTURE WORK

The algorithm propagates status information as quickly and through as many redundant paths as possible to allow it to effectively handle dynamic situations. All nodes can change state at the same time or a cascade of status changes can occur, while maintaining a notion of correctness at all times.The algorithm DDNCN has been able to handle these behaviors effectively in not-completely-connected networkswith diagnostic latency of O(1).

The work presented here can be extended in many possible ways mainly based on the variant workload types under consideration, the types of distributed diagnosis algorithm whether on-line or off-line monitoring and with respect to different fault models. As a first step to use the distributed diagnosis algorithm particularly for performance and monitoring of actuators, we develop a generalized framework for on-line diagnosis for safety-critical applications such as fly-by-wire, drive-by-wire, steer-by-wire, neuclear reactor,etc.

REFERENCES

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

[2] E.P. Duarte Jr., and T. Nanya, "A Hierarchical Adaptive Distributed System-Level Diagnosis Algorithm", IEEE Trans on Comp, Vol.47, pp. 34-45, Jan. 1998.

[3] E.P. Duarte Jr., A. Brawerman, and L.C.P. Albini, "An Algorithm for Distributed Hierarchical Diagnosis ofDynamic Fault and Repair Events", Proc. IEEE ICPADS'OO, 2000.

[4] Arun Subbiah, and D.M.Blough, "Distributed Diagnosis in Dynamic Fault Environments", IEEE Trans on PDS, Vol 15, No. 5, May 2004.

[5] S. Rangarajan, A.T. Dahbura, and E. Ziegler, "A Distributed System-Level Diagnosis Algorithm for Arbitrary Network Topologies, "IEEE Trans. Computers, vol.44,pp.312-334, Feb. 1995.

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

[7] M.Hiltunen, "Membership and System Diagnosis, " Proc. 14th Symp.Reliable Distributed Systems, pp.208-217, 1995.

[8] P. M. Khilar, S.Mahapatra " Distributed Diagnosis in Dynamic Fault Environments for Arbitrary Network Topologies " IEEE INDICON 2005 Conference, Chennai, India, 11-13, Dec 2005, pp. 56-59