

**Design and Evaluation of a Distributed Diagnosis  
Algorithm for Arbitrary Network Topologies in  
Dynamic Fault Environments**

A Thesis  
Presented to  
The Academic Faculty

by

**Arun Subbiah**

In Partial Fulfillment  
of the Requirements for the Degree of  
Master of Science in Electrical and Computer Engineering

Georgia Institute of Technology  
November 2001

**Design and Evaluation of a Distributed Diagnosis  
Algorithm for Arbitrary Network Topologies in  
Dynamic Fault Environments**

Approved:

---

Dr. Douglas M. Blough, Chairman

---

Dr. Linda Wills

---

Dr. David Schimmel

Date Approved \_\_\_\_\_

# Dedication

This thesis is dedicated to my parents and my sister

# Acknowledgements

I would first like to thank my advisor Dr. Douglas Blough. Without his help, guidance and support, this thesis would not have been possible. I am also grateful to my parents and my sister for their support and encouragement.

# Contents

<b>Dedication</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>List of Figures</b>	<b>vii</b>
<b>Summary</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 System Model and Bounded Correctness</b>	<b>4</b>
2.1 Communication Model . . . . .	4
2.2 Fault Model . . . . .	5
2.3 Time and Clock Models . . . . .	7
2.4 Algorithm Assumptions . . . . .	7
2.5 Bounded Correctness . . . . .	9
<b>3 Related Work</b>	<b>12</b>
<b>4 Algorithm ForwardHeartbeat</b>	<b>17</b>
4.1 Description of the algorithm . . . . .	17
4.2 Analysis of Algorithm ForwardHeartbeat . . . . .	22
4.2.1 Diagnosis of non-neighboring nodes . . . . .	22
4.2.2 Diagnosis of neighboring nodes . . . . .	37
4.2.3 Overall Results . . . . .	41

<b>5</b>	<b>Simulation Results</b>	<b>48</b>
5.1	Hypercube Networks . . . . .	48
5.2	Random Networks . . . . .	59
5.2.1	Effect of the Poisson Mean . . . . .	61
5.2.2	Effect of Heartbeat period ( $\pi$ ) . . . . .	67
5.2.3	Effect of connectivity $k$ . . . . .	69
<b>6</b>	<b>Conclusion</b>	<b>75</b>

# List of Figures

1	Pseudocode for Algorithm ForwardHeartbeat (Recovering Node Portion) . . . . .	22
2	Pseudocode for Algorithm ForwardHeartbeat (Received Message Portion) . . . . .	23
3	Pseudocode for Algorithm ForwardHeartbeat (Timer Handler Portion) . . . . .	24
4	Network for proof of lemma 4 . . . . .	28
5	Network for proof of lemma 5 . . . . .	31
6	Theoretical and measured latency in detecting failure events for Poisson mean 200 seconds for hypercube networks . . . . .	49
7	Theoretical and measured latency in detecting failure events for Poisson mean 1 second for hypercube networks . . . . .	50
8	Theoretical and measured latency in detecting recovery events for Poisson mean 200 seconds for hypercube networks . . . . .	51
9	Theoretical and measured latency in detecting recovery events for Poisson mean 1 second for hypercube networks . . . . .	52
10	Measured worst case and average latency in detecting failure events for Poisson mean 200 seconds for hypercube networks . . . . .	54
11	Measured worst case and average latency in detecting failure events for Poisson mean 1 second for hypercube networks . . . . .	55
12	Measured worst case and average latency in detecting recovery events for Poisson mean 200 seconds for hypercube networks . . . . .	56

13	Measured worst case and average latency in detecting recovery events for Poisson mean 1 second for hypercube networks	57
14	Message cost of Algorithm ForwardHeartbeat for hypercube networks . . . . .	58
15	Average latency in detecting failure events for some random networks . . . . .	60
16	Average latency in detecting recovery events for some random networks . . . . .	61
17	Latency in detecting failure events vs $n$ for different values of the Poisson mean for random networks . . . . .	62
18	Latency in detecting recovery events vs $n$ for different values of the Poisson mean for random networks . . . . .	63
19	Message cost vs $n$ for different values of the Poisson mean for random networks . . . . .	64
20	Number of events per heartbeat period vs $n$ for different values of $k$ and Poisson mean for random networks . . . . .	65
21	Latency in detecting failure events vs $n$ for different values of $\pi$ for random networks . . . . .	66
22	Latency in detecting recovery events vs $n$ for different values of $\pi$ for random networks . . . . .	67
23	Message cost vs $n$ for different values of $\pi$ for random networks	68
24	Latency in detecting failure events vs $n$ for different values of connectivity for random networks . . . . .	69
25	Theoretical latency in detecting recovery events vs $n$ for different values of connectivity for random networks . . . . .	70
26	Measured latency in detecting recovery events vs $n$ for different values of connectivity for random networks . . . . .	71

27	Message cost vs $n$ for different values of connectivity for random networks . . . . .	73
28	Message cost per link vs $n$ for different values of connectivity for random networks . . . . .	74

# Summary

This thesis considers the problem of achieving distributed diagnosis in arbitrary-topology networks in dynamic fault environments. It is assumed that only nodes are subject to crash faults and the communication links are fault free. In dynamic fault environments, no restriction is placed on the number of nodes that can fail or recover at the same time or when an event occurs. However, we consider a restricted case where it is assumed that at all times, there exists a path between every pair of nodes in the network. In addition, it has been shown that the frequency with which events can occur on a single node must be limited to achieve correct diagnosis. This is characterized by the state holding time.

Existing works perform poorly in the above conditions. An algorithm called “ForwardHeartbeat” is proposed that achieves correct diagnosis under the above stated conditions. Algorithm ForwardHeartbeat works by having nodes perform independent diagnosis of the system. A working node indicates that it is fault free by sending out messages called heartbeats every  $\pi$  time units. These heartbeats are flooded throughout the network by the other nodes facilitating each node to perform its own independent diagnosis of other nodes in the network. Arrival of a new heartbeat after a node has been diagnosed to be in the failed state indicates a recovery event. Failure to receive a heartbeat once the node has been diagnosed to be in the working state indicates a failure event. In addition, issues arising due to loss of messages and messages arriving in duplicate and out of order are handled.

Since we consider a dynamic fault environment, we use a set of correctness properties that define the concept of Bounded Correctness. Details of these properties are

presented in this thesis. It has been formally proven that Algorithm ForwardHeartbeat satisfies the properties of Bounded Correctness.

The algorithm has been simulated on hypercube networks and randomly generated networks. The parameters varied were the connectivity of the network, the dynamic nature of the system and the period with which nodes send heartbeats. It has been found that the message cost (number of messages in a given period of time) increases when the heartbeat period is decreased. Both the theoretical and measured latencies have been found to be a little above the heartbeat period for typical values of network parameters. The heartbeat period is the parameter that needs to be set when implementing the algorithm. The heartbeat period should be set such that the theoretical latency of the algorithm is the maximum latency allowed by the application. If the resulting message cost exceeds the bandwidth allocated for the algorithm, then the network bandwidth must be increased, or the required latency cannot be achieved with this algorithm.

# Chapter 1

## Introduction

An important problem in distributed systems that are subject to component failures is the distributed diagnosis problem. In distributed diagnosis, each working node must maintain correct information about the status (working or failed) of each component in the system. We consider the problem of achieving diagnosis in a dynamic fault environment in networks with arbitrary topologies. Previous work has almost exclusively dealt with the static fault situation wherein statuses of nodes remain fixed for as long as it takes an algorithm to completely diagnose the system. A few works have attempted to consider dynamic events in completely connected networks, but none exist for arbitrary topology networks. By dynamic fault environment we mean that events (node failures and repairs) can take place at any time provided the network remains connected at all times and the frequency with which a single node can suffer events is limited as required by the distributed diagnosis algorithm.

In this thesis, we present a distributed diagnosis algorithm called Algorithm ForwardHeartbeat for achieving distributed diagnosis for networks with arbitrary topologies in a dynamic fault environment. The network is assumed to consist of nodes that are subject only to crash faults interconnected by fault free communication links. It is assumed that the number of nodes that can be in the failed state at any given time cannot exceed the connectivity of the network so that it is guaranteed that the network is connected at all times. Algorithm ForwardHeartbeat satisfies the properties of bounded correctness which are: bounded diagnostic latency, bounded startup

and accuracy. For bounded diagnostic latency, all working nodes that were in the working state since an event occurred must learn about that event (node failure or repair) within a bounded time  $L$ , known as the diagnostic latency of the algorithm. For bounded startup, nodes that recover must determine a state for every other node within time  $S$  of entering the working state that was actually held by other nodes after this node recovered. Finally, accuracy ensures that no spurious events are recorded by any working node. The concept of bounded correctness is required in dynamic fault situations. It is formally proven that algorithm ForwardHeartbeat achieves bounded correctness.

Algorithm FloodHeartbeat works by having nodes send heartbeats periodically which are flooded throughout the network. Each working node performs independent diagnosis of other nodes in the network. It is crucial in these assumptions that heartbeat messages are not lost due to the dynamic nature of the system. This problem is overcome by having nodes buffer heartbeat messages. In addition, due to the nature of the network, it is possible for messages to arrive out of order and with variable delays. Upon receiving a heartbeat, a node starts a timer waiting for the next heartbeat for the same node. If a new heartbeat arrives before the timer expires, then upon receipt of the new heartbeat a new timer is started and the old timer is killed. Receipt of a heartbeat indicates that the node that sent it is in the working state. If the timer expires and no new heartbeat is received, then a timeout is said to have occurred and a failure event is detected.

Since messages can be duplicated and arrive with different delays, it is possible that after a node is declared to be in the failed state, a duplicate, stale heartbeat message is received which should not be taken as a sign of a recovery event. Hence, nodes reject heartbeats after a timeout has occurred for a certain period of time. It follows that the state holding time should be large enough to account for this.

The performance of an algorithm is characterized by the theoretically expected,

maximum observed and average latencies in detecting failure and recovery events, the message costs and the state holding times. Algorithm ForwardHeartbeat was simulated on hypercube networks and networks with arbitrary topologies. The dependence of the latency and the message cost on varying sizes of the network, connectivity and the dynamic nature of the system was studied and analyzed. Some unexpected results include the observation that the theoretical value for the latency in detecting recovery events is a linear function of the number of nodes in the network while the experimentally observed values hardly varied with the number of nodes in the network. Theoretical results show that the latency in detecting a recovery event will be higher when the connectivity of the network is higher, while the experimentally observed results demonstrate the exact opposite. The diagnostic latency and state holding time of algorithm ForwardHeartbeat are shown to be approximately one heartbeat period (testing round). Hence, in terms of latency and the state holding time, Algorithm ForwardHeartbeat performs significantly better than existing algorithms which have latencies and state holding times of about  $n - 1$  testing rounds.

# Chapter 2

## System Model and Bounded Correctness

In this chapter, we describe the system model assumed in this thesis and the definitions of bounded correctness.

### 2.1 Communication Model

Diagnosis algorithms can use either unicast or multicast communication. We do not assume one or the other type. Rather, we assume the following generic parameters that could apply to either type of communication. However, we do assume a *synchronous* system, i.e. one in which the communication delay is bounded. This is an implicit assumption in all prior work on distributed diagnosis.

**Definition 1** *The send initiation time,  $\Delta_{\text{send\_init}}$ , is the time between a node initiating a communication and the last bit of the message being injected into the network. This includes message set-up time on the node, any delay in accessing the communication medium, and the time to inject all of the message bits into the network. To simplify analysis, it is assumed that  $\Delta_{\text{send\_init}}$  is a constant.*

**Definition 2** *The minimum and maximum message delays,  $\Delta_{\text{send\_min}}$  and  $\Delta_{\text{send\_max}}$ , 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.*

Any message processing time on a receiving node is assumed to be included in the message delay. It is important to separate out the message initiation time on a sender in order to account for failures that occur while a node is in the middle of transmitting a message. We assume that messages are encoded in such a way, e.g. using checksums, to enable incomplete messages to be detected and discarded. Hence, failures that occur on a sending node in the middle of a message transmission (prior to the last bit of the message, including the checksum, being injected into the network) appear as omissions at receiving nodes.

In networks with arbitrary topologies, intermediate nodes are required to relay messages between some source-destination pairs. Hence, the number of node failures is limited such that the network remains connected at all instants of time. We also assume that not-completely-connected networks use a store and forward mechanism to relay messages throughout the network.

## 2.2 Fault Model

We consider crash faults in nodes. Fault timing is arbitrary, i.e. a node can crash at any time. The crash fault assumption differs from traditional work on system-level diagnosis for which the fault model is not specified. However, the classical assumption in the field that tests are perfect implies some class of easily-detectable faults. The crash fault assumption is necessitated by our use of a heartbeat-based algorithm for diagnosis, which have been more commonly used in group membership algorithms. In [12], it is shown how heartbeat-based algorithms can be transformed into test-based algorithms and vice versa. Using this transformation, algorithm ForwardHeartbeat could be easily converted to ones that use explicit testing and the crash fault assumption could then be loosened.

Since we are interested in dynamic failures and repairs, nodes can alternate between working correctly and being crashed in our model. Hence, the status of a node is modeled by a state machine with two states, *failed* and *working*. Nodes that are in the working state execute faithfully the diagnosis procedure. Nodes that are in the failed state do not send messages nor do they perform any computation.

**Definition 3** *The working state holding time, denoted by  $SHT_w$ , is the minimum time a node remains in the working state before transitioning to the failed state.*

**Definition 4** *The failed state holding time, denoted by  $SHT_f$ , is the minimum time a node remains in the failed state before transitioning to the working state.*

Ideally, no restrictions would be placed on the length of time a node is in one state before moving to the other, because this constrains the model and could prevent its application in certain systems or environments. However, it has been found that it is necessary to have a non-zero state holding time in order to be able to achieve diagnosis under a dynamic failure and repair model. Hence, an important goal is to try to achieve the minimum possible state holding times.

We assume that faults are restricted to nodes, i.e. the network delivers messages reliably. Thus, if a sending node injects an entire message into the network, the message will be delivered correctly to each neighboring node that is in the working state from the time the first bit of the message arrives until the message is completely received.

We consider a restricted fault model where the number of nodes that can be in the failed state at any instant of time is limited to at most  $(k - 1)$ , where  $k$  is the connectivity of the network as defined below.

**Definition 5** *The connectivity of the network, denoted by  $k$ , is the minimum number of nodes that need to be removed so that there exists no path between some pair of nodes in the network.*

The maximum number of neighbors a node has in the network is denoted by  $d$ .

By restricting the number of nodes that can be in the failed state at any instant of time to at most  $(k - 1)$ , we ensure that the network remains connected at all times, though this restriction is not a necessary condition for the network to be connected.

## 2.3 Time and Clock Models

Since we are interested in dynamic failure situations in which failure and recovery timing is critical, it is imperative that the notion of time be well defined.

**Definition 6** *Time that is measured in an assumed Newtonian time frame (which is not directly observable) is referred to as real time.*

**Definition 7** *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  $T_x(t)$ .*

**Definition 8** *While a node is in the working state, its clock experiences bounded drift. This means that if a node  $x$  is in the working state continuously during a real-time interval  $[t_1, t_2]$ , then for all real-time intervals  $[u_1, u_2] \subseteq [t_1, t_2]$*

$$|[T_x(u_2) - T_x(u_1)] - (u_2 - u_1)| \leq \rho(u_2 - u_1),$$

where  $\rho \ll 1$  is the maximum drift rate of a clock.

## 2.4 Algorithm Assumptions

We assume algorithms work by use of heartbeat messages, i.e. that each node periodically initiates a round of message transmissions to other nodes in order to indicate that the node is working (not crashed).

**Definition 9** *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  $t + (1 + \rho)\pi$ , where  $\pi$  is the heartbeat period.*

Our assumption of heartbeat-based algorithms differs from the standard testing assumption adopted in distributed diagnosis where the test is initiated by the tester. However, the effect is the same in that, if node  $x$  periodically sends a heartbeat to node  $y$  when  $x$  is working, then  $y$  periodically evaluates the state of (tests)  $x$ . In fact, heartbeats are more efficient because only one message is required instead of two in the standard testing model (stimulus and response). In addition, as long as node  $x$  remains in the failed state, no messages are sent in a heartbeat implementation and yet node  $y$  maintains the correct status for  $x$ . In the standard testing model, while  $x$  remains in the failed state, node  $y$  would continue to periodically test  $x$  and would, therefore, generate additional messages. Of course, heartbeats are only a valid method of implementing a test when faults are restricted to crash type, as is assumed in this thesis.

We must also consider the behavior of the algorithm after recovery, i.e. after moving from the failed state to the working state. In this situation, a heartbeat-based algorithm could initiate a round of heartbeats immediately after entering the working state or it could wait before doing so. If the algorithm waits, however, it should not wait more than  $\pi$  in local clock time in order to maintain the heartbeat period. This leads to the following definition.

**Definition 10** *The recovery wait time for an algorithm, denoted by  $W \leq \pi$ , is the local clock time for which the algorithm waits after entering the working state before initiating a round of heartbeat transmissions.*

In [8], we see that when  $W$  is approximately  $\pi/2$ , the state holding times and the maximum time taken to detect events is minimum for an algorithm in completely connected networks.

## 2.5 Bounded Correctness

In a system that dynamically experiences failures and repairs and has non-zero communication delay, the view that any node has of the system at any time is, inevitably, out of date. Each working node should have timely information about the status of every other node, either working or failed, in the system. A transition between the states working and failed on a node should, therefore, trigger the status of that node to be changed on every working node with a minimum delay. Any transition between the two states on a node is referred to as an *event*. The goal is for working nodes to learn about every event in the system as quickly as possible, to have their views of other nodes be out of date by only a bounded amount, and to not detect any spurious events.

In [8], this goal is represented by three properties which are collectively refer to as *bounded correctness*. Specification of one of these properties requires the following definition:

**Definition 11** *A state held by a working node  $x$  for another node  $y$  at time  $t$  is said to be  $T$ -valid if node  $y$  was in the indicated state at some point during the interval  $[t - T, t]$ .*

### Property 1 Bounded Diagnostic Latency

*Consider any event in the system that occurs at an arbitrary real time  $t$ . Any node that is in the working state continuously during the interval  $[t, t + L]$  learns about the event and records the new state of the node that experienced the event by time  $t + L$ ,*

where  $L$  is an algorithm-dependent bounded time referred to as the diagnostic latency of the algorithm.

### **Property 2 Bounded Start-up**

*Consider the recovery of an arbitrary node  $x$  at real time  $t$ . If  $x$  remains in the working state continuously during the interval  $[t, t + S]$ , then at time  $t + S$ ,  $x$  holds  $L$ -valid states for every other node in the system, where  $S \geq L$  is an algorithm-dependent bounded time referred to as the start-up time of the algorithm.*

### **Property 3 Accuracy**

*Consider an arbitrary working node  $x$  after its start-up time. Every state transition (working to failed or failed to working) recorded by  $x$  for an arbitrary node  $y$  corresponds to an actual event that occurred on  $y$  and no single event on  $y$  causes multiple state transitions to be recorded on  $x$ .*

Taken together, these properties ensure that after a node recovers (or starts up for the first time), it will determine valid state information about every other node in the system within bounded time and from that time on it will maintain a faithful record of events that occur on all nodes. Bounded Diagnostic Latency ensures that no events are missed, while Accuracy guarantees that no spurious events are recorded.

These three properties together imply something that could be referred to as bounded staleness. Bounded staleness means that if a working node holds a view of the state of another node at some time after its start-up period, then that view is out of date by at most  $L$ . Bounded Diagnostic Latency is not sufficient by itself to ensure bounded staleness, because it deals only with event detection. Bounded staleness requires a node that recovers from a fault and returns to the working state to determine the states of all other nodes within bounded time even if no events occur in the system during that time. Accuracy is required to ensure this notion of bounded

staleness as well. We do not state bounded staleness as a separate property because it is already implied by the three properties that constitute Bounded Correctness.

## Chapter 3

### Related Work

Distributed Diagnosis algorithms were first introduced by Kuhl and Reddy [14, 16, 17] in which each fault free node in the system reliably receives test results through its neighbors to perform consistent diagnosis. These algorithms assumed arbitrary network topologies and no event occurred during the execution of the algorithm (static fault model).

The bulk of the work in system diagnosis has assumed a static fault situation [1, 2, 3, 4, 6, 9, 10, 18, 19, 20, 22], i.e., the statuses of nodes do not change during the execution of the diagnosis procedure. Some diagnosis algorithms, e.g. [5, 19, 21] allow dynamic failures and repairs to occur but are only guaranteed to be correct when system status has become stable. One of the diagnosis algorithms in [7] assumes that nodes can fail dynamically but cannot be repaired during execution of the diagnosis procedure. This approach is suitable in systems where repairs are accomplished in an offline fashion after all fault free nodes in the system have diagnosed a faulty node to be faulty. The diagnosis model of [15] considers dynamic failures but the approach is centralized and is, therefore, not comparable to the distributed diagnosis approach.

Another aspect of previous work on distributed diagnosis is that it has focussed almost exclusively on minimising the number of tests performed. Most of this work considers only a static fault situation, while some discuss briefly dynamic events but prove correctness only when the situation has stabilized. Prior algorithms that minimize the number of tests construct sparse testing graphs and propagate information

backwards, i.e., in the reverse direction of tests. The ideal testing property these algorithms strive is to have each node tested by exactly one other node in each testing round. In dynamic situations, this yields poor results.

In [11], Duarte, Brawerman and Albini introduced Hi-ADSD with timestamps algorithm for a system in a dynamic fault environment. The authors assume a fully connected network and that events do not occur on a node too frequently, failing which incorrect diagnosis could result.

In [8], Blough gave a formal definition of correctness for distributed diagnosis algorithms in dynamic fault environments called Bounded Correctness. The notion of bounded correctness provided, for the first time, a continuous time correctness property that can be applied to diagnosis algorithms. The notion of correctness in prior work was inherently targeted at a fixed snapshot of a system's state. Hence, the bounded correctness property provided higher confidence in the stream of diagnostic results that are produced when failures and repairs are dynamic. Blough also proved that in order to achieve correct diagnosis a node must stay in a state for a certain minimum time called the state holding time. Thus, [8] defines new quantities (latency, staleness and the state holding time) that serve as metrics for diagnosis algorithms in a dynamic fault environment besides the traditional metric of message costs.

[8] also gives the distributed diagnosis algorithm called HeartbeatComplete for fully connected networks that achieves the minimum possible diagnostic latency, staleness and state holding time.

One of the earliest works on distributed diagnosis for arbitrary network topologies was by Bagchi and Hakimi [1]. The idea was to form a tree in the most efficient way and distribute diagnosis information through that tree. The authors assumed that faults are permanent and nodes initiate the tree formation algorithm spontaneously and at arbitrary times. Hence there is no theoretical upper bound on the diagnostic latency for this algorithm.

In [24, 25], Stahl, Buskens and Bianchini introduced the Adapt algorithm for distributed diagnosis in arbitrary topology networks. The Adapt algorithm does not assume a permanent fault model and performs periodic testing of nodes. The Adapt algorithm starts with an initial testing assignment and any event introduces minimal change to the testing assignment existing before the event occurred and at the same time keeping the new testing assignment graph minimally strongly connected. Diagnosis information is then transmitted through the new testing assignment graph. The Adapt algorithm can be proved to satisfy bounded correctness in a dynamic fault environment with high state holding time and high diagnostic latency.

In [21], Rangarajan, Dahbura and Ziegler introduced a distributed diagnosis algorithm in which any node in the network is tested by only one other node in the system. When an event is detected, the information is flooded throughout the network during which time if a node finds that it is unable to propagate the information to a neighboring node because the neighboring node is in the failed state, it floods the whole network with the information that the neighboring node has failed. It can be shown that this algorithm too satisfies the properties of bounded correctness with a high diagnostic latency and state holding time.

Both the Adapt algorithm and the above algorithm are event driven algorithms where information is propagated throughout the network only when an event is detected.

[8] was the first work to derive the latency and the state holding time for an algorithm that performs distributed diagnosis in a dynamic fault environment for a fully connected network. No previous work has been done on distributed diagnosis algorithms characterized by the diagnostic latency and the state holding time for arbitrary network topologies in a dynamic fault environment.

Algorithms mentioned in [1, 24, 21] were analyzed assuming a dynamic fault

model. The algorithm given in [1] assumes that faults are permanent. Nodes spontaneously and at arbitrary times initiate a tree formation algorithm and at the end of the tree formation procedure a single tree is generated. Fault free nodes exchange diagnostic information through the tree. The primary goal of this paper is to arrive at a single tree in the quickest possible way and at the same time reducing the number of messages to be transmitted for diagnosis to be complete. To make the algorithm work in a dynamic fault environment, we modified the algorithm so that nodes periodically initiate tree formation algorithms and not spontaneously and at arbitrary times. The algorithm satisfies the properties of bounded correctness with a latency of approximately  $(n - 1)\pi$  (assuming message transmission delays are negligible compared to  $\pi$  provided the state holding time is approximately  $(n - 1)\pi$  where  $n$  is the total number of nodes in the network and  $\pi$  is the period with which nodes initiate the tree formation algorithm).

The Adapt algorithm in [24] assumes an initial testing assignment. When an event is detected, minimal changes are made to the initial testing assignment to arrive at the new testing assignment and at the same time keeping the new testing assignment graph minimally strongly connected. The performance of the Adapt algorithm in a dynamic fault environment depends to a great extent on the initial testing assignment. Since the Adapt algorithm tries to minimize message costs and testing overheads the number of nodes testing any node in the network is kept to a minimum. This has a high probability of yielding a chain of nodes where each node in the chain is tested only by the next node in the chain. The diagnostic latency and the state holding time will be at least  $(M - 1)\pi$  in this case where  $M$  is the number of nodes in the chain and  $\pi$  is the period with which a node tests other nodes as per the testing assignment. In the worst case, if the system graph is Hamiltonian the testing assignment could reduce to a ring structure with all nodes in the network present in the ring in which case the latency and the state holding time will reduce to approximately  $(n - 1)\pi$ ,

where  $n$  is the total number of nodes in the network.

The algorithm given in [21] minimizes message costs and testing overhead by requiring that each node is tested by only one other node periodically. This algorithm favors a ring structure for the testing assignment. The longer the ring, the larger will be the diagnostic latency and the state holding time. In the worst case, if all  $n$  nodes in the network were part of the ring, then the diagnostic latency and the state holding time will reduce to approximately  $(n - 1)\pi$ , where  $\pi$  is the period with which a node tests another node.

If the number of nodes in the network is 60 and the period with which a node sends heartbeats is 30 seconds (as assumed in the simulation studies of the ADSD algorithm [5]), the latency and the state holding time turns out to be 30 minutes. Hence, the existing algorithms are not suitable for a system in a dynamic fault environment because of the high diagnostic latency and the high state holding time required to satisfy bounded correctness by these algorithms. Algorithm ForwardHeartbeat is presented in this thesis which achieves bounded correctness with a latency and state holding time of approximately  $\pi$  time units, where  $\pi$  is the period with which a node initiates heartbeats. The price paid to achieve this is the increase in message cost.

# Chapter 4

## Algorithm ForwardHeartbeat

In this chapter, a distributed diagnosis algorithm, known as Algorithm ForwardHeartbeat, that achieves bounded correctness when applied to networks with arbitrary topologies is presented.

### 4.1 Description of the algorithm

Since the network is not fully connected, messages are propagated throughout the network by having nodes relay them. In this thesis, we consider a restricted fault model where the number of nodes that can be in the failed state at any instant of time is limited to at most  $(k - 1)$ , where  $k$  is the connectivity of the network as defined in Chapter 2. By restricting the number of nodes that can be in the failed state at any instant of time to at most  $(k - 1)$ , we ensure that the network remains connected at all times, though this restriction is not a necessary condition for the network to be connected.

Having the network connected at all times, however, does not guarantee that a message reaches every working node in the network. Consider the following example. A node  $X$  has  $a > k$  neighbors, with some  $b < k$  neighboring nodes in the failed state and all remaining neighbors in the working state. At some time instant  $t$ , node  $X$  issues a message (or relays a message) to all its neighbors. The  $(a - b)$  working nodes receive it and just before they can start processing it, the  $b$  failed neighbors

recover and the  $(a - b)$  working neighbors fail. The  $b$  nodes that recovered did not receive node  $X$ 's message and the  $(a - b)$  nodes that received it failed to relay the message. The original message sent by node  $X$  is lost although the network might have remained connected at all times.

In algorithm ForwardHeartbeat, this problem is overcome by having nodes buffer messages that they send or relay. The last heartbeat message received from a node  $Y$  is kept buffered by node  $X$  until node  $X$  holds a status of *working* for node  $Y$ . Arrival of a new heartbeat message from node  $Y$  replaces any existing buffered heartbeat message from node  $Y$ .

In Algorithm ForwardHeartbeat, working nodes perform their own independent diagnoses of the system. Working nodes periodically send heartbeats which are propagated throughout the network. In our terminology, a heartbeat consists of several heartbeat messages that may be propagating through different parts of the network at a given instant of time. Thus, a single heartbeat, which should be understood at the conceptual level, is physically represented in the network as several heartbeat messages. A node initiates a heartbeat by sending heartbeat messages on all its links. These heartbeat messages, when received by working neighboring nodes, are in turn sent on all their links and this process is repeated resulting in the heartbeat being propagated throughout the network.

Failure to receive a new heartbeat from a node within a certain period of time (called the timeout period) since the last heartbeat was received results in the node being declared faulty. Arrival of a new heartbeat from a node after it has been declared faulty indicates that the node has recovered.

The diagnosis performed by a node is different for its neighboring nodes and its non-neighboring nodes. A node, say  $X$ , discards any heartbeat it may receive regarding a neighboring node if the heartbeat did not arrive directly from the neighboring node. Hence, for the diagnosis of neighboring nodes, relaying of messages is

not involved, thereby simplifying the algorithm. Issues that arise due to relaying of heartbeat messages arise when diagnosis is performed on non-neighboring nodes. Relaying of messages leads to the following problems.

1. *Loss of heartbeat messages:* As pointed out earlier, this is overcome by having nodes buffer and retransmit heartbeat messages.
2. *Out of order delivery of heartbeat messages:* This means that a heartbeat reaches a node before a previously issued heartbeat reaches that same node. In case of Algorithm ForwardHeartbeat, this does not cause any problems because what is of significance here is that some new heartbeat arrived before the node could timeout waiting for the next heartbeat.
3. *Spurious heartbeats:* 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 which arrived through another path with a larger propagation delay. Since this heartbeat is a stale heartbeat and should not be treated as an indication of a recovery event, it must be discarded. Hence, when nodes time out, they discard any heartbeats they may receive from the node they diagnosed to be faulty for a predetermined amount of time called the heartbeat rejection time defined below. 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.

**Definition 12** *The Heartbeat Rejection Time, denoted by  $T_{\text{reject}}$ , is the period of time after a node(say  $Y$ ) has been declared to be in the failed state during which time the node that declared node  $Y$  to be in the failed state rejects any heartbeat message that it may receive regarding node  $Y$ .*

The above discussions are related to issues that determine the correctness of Algorithm ForwardHeartbeat. Another issue that arises due to the relaying of heartbeat messages is that heartbeats could take a long time to propagate to other nodes in the network. Since the failure of a node is detected by timing out while waiting for a new heartbeat since the last heartbeat was received, having nodes timeout after a fixed timeout period regardless of the delay with which the last heartbeat was received could lead to a high latency in detecting a failure event. The large delay in the propagation of a heartbeat message is due to the dependence on a relaying mechanism for the propagation of heartbeats coupled with the dynamic nature of the system. Hence, this is a restriction forced by the system model. However, a large delay in the propagation of a heartbeat message implies that the heartbeat message was kept buffered at some intermediate node's buffer and/or the heartbeat message propagated through a number of intermediate nodes before arriving at the destination node. Hence, the timeout value associated with a heartbeat message as the heartbeat message passes through successive nodes should decrease. Hence, an intermediate node, while relaying a heartbeat message, also appends information stating for how much time the heartbeat was kept buffered locally and the minimum time it takes to relay the heartbeat to the next node. This quantity keeps increasing as the heartbeat message passes through a series of nodes.

In other words, when a node receives a heartbeat message, it estimates how far back in time the heartbeat was initiated, thereby helping the node decide for how much more time it needs to wait for the next new heartbeat before it should timeout. This functionality is implemented using the *delay* field present in a heartbeat message as detailed below.

Taking into account the above factors, the algorithm is designed the following way. A heartbeat message consists of the following fields:

Node\_id: This field contains the ID of the node that initiated the heartbeat.

`Seq_no`: This field contains the sequence number of the heartbeat.

`delay`: This field contains the minimum delay the heartbeat message experienced before it could reach the intermediate/destination node.

The `Seq_no` field serves to distinguish heartbeats. Due to the dynamic and non-fully connected nature of the network, a node may get multiple heartbeat messages generated from the same heartbeat. Such messages, after the first, are duplicate heartbeat messages and should be eliminated from the network. Upon receipt of a heartbeat message, nodes store the sequence number of the heartbeat message and will discard heartbeat messages that have a sequence number that is logically less than or equal to the highest sequence number already received from a particular node.

When a node recovers, the first (logical) heartbeat sequence number used is 0. Successive heartbeats from a node have increasing sequence numbers, and when the maximum sequence number (denoted by *MAX\_SEQ\_NUM*) that can be stored in this field is exceeded, the sequence number is wrapped around to 0. However, the logical sequence number is a monotonically increasing quantity. Unless otherwise mentioned, throughout this paper sequence numbers denote logical sequence numbers.

The last heartbeat received from a node is kept buffered until the next new heartbeat from the same node is received, or until a timeout occurs and the node is declared to be in the failed state, whichever occurs earlier. In the former case, the new heartbeat replaces the old heartbeat in the buffer.

Each node maintains an array of  $n$  entries, where  $n$  is the total number of nodes in the network. The  $i^{th}$  entry in the array contains:

`Status`: This is the status of node  $i$  in the network. Possible values are failed, working and unknown.

`Last_seq_no`: This is the physical heartbeat sequence number of the largest logical heartbeat sequence number received from node  $i$ .

*Upon entering the working state, node  $i$  executes:*

```
Node[ $i$ ].Status  $\leftarrow$  working;  
Node[ $i$ ].Last_seq_no  $\leftarrow$  -1;  
SetSendHeartbeatTimer( $W$ );  
SetStartupTimer( $(1 + \rho)(W + T_{\text{exist}})$ );  
for  $j = 0$  to  $n - 1$ ,  $j \neq i$  do  
    Node[ $j$ ].Status  $\leftarrow$  unknown;  
    Node[ $j$ ].Last_seq_no  $\leftarrow$  -1;  
endfor;
```

**Figure 1: Pseudocode for Algorithm ForwardHeartbeat (Recovering Node Portion)**

When a node, say  $X$ , recovers, diagnosis of another node  $Y$  in the network follows from the time a heartbeat is received from node  $Y$ . If no heartbeat is received from node  $Y$  until the startup time, then node  $Y$  is diagnosed to be in the failed state. The pseudocode for algorithm ForwardHeartbeat is shown in Figures 1–3.  $T_{\text{exist}}$  and  $T_{\text{timeout}}$  are defined in the next section.

## 4.2 Analysis of Algorithm ForwardHeartbeat

As pointed out earlier, we distinguish between diagnosis of neighboring nodes and diagnosis of non-neighboring nodes because the dynamic nature of the system does not affect diagnosis of neighboring nodes. Nodes accept heartbeats about their neighbors only when received from the respective neighbors directly.

### 4.2.1 Diagnosis of non-neighboring nodes

We first define some terms so that the results can be stated in a more concise form.

*Upon receiving a heartbeat message from a neighboring node, node  $i$  executes:*

```

 $k \leftarrow \text{ReceivedHeartbeat.Node\_id};$ 
if ((ReceivedHeartbeat.Seq\_no > Node[ $k$ ].Last\_seq\_no) and (not)( $\exists T_{\text{reject}} \text{Timer}_k$ ))
then
  if (node  $k$  is not a neighbor of node  $i$ ) then
    SetReceiveHeartbeatTimer $_k(T_{\text{timeout}})$ ;
    Node[ $k$ ].Status  $\leftarrow$  working;
    Node[ $k$ ].Last\_seq\_no  $\leftarrow$  ReceivedHeartbeat.Seq\_no;
    Buffer[ $k$ ]  $\leftarrow$  ReceivedHeartbeat;
    Add ( $\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}}$ ) to ReceivedHeartbeat.delay;
    Send ReceivedHeartbeat on all links except the link on which it arrived;
  else
    if (heartbeat received on link connecting node  $k$ ) then
      SetReceiveHeartbeatTimer $_k((1 + \rho)((1 + \rho)\pi + \Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}))$ ;
      Node[ $k$ ].Last\_seq\_no  $\leftarrow$  ReceivedHeartbeat.Seq\_no;
      Buffer[ $k$ ]  $\leftarrow$  ReceivedHeartbeat;
      if (Node[ $k$ ].Status  $\neq$  working) then
        Add  $(1 - \rho)$ (amount of time heartbeat was kept buffered) and
        ( $\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}}$ )
        to the delay field in all heartbeats stored in buffer and
        send them to node  $k$ ;
      endif;
      Node[ $k$ ].Status  $\leftarrow$  working;
      Add ( $\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}}$ ) to ReceivedHeartbeat.delay;
      Send ReceivedHeartbeat on all links except the link on which it arrived;
    endif;
  endif;
endif;

```

**Figure 2: Pseudocode for Algorithm ForwardHeartbeat (Received Message Portion)**

*Handlers execute when the corresponding timers expire:*

```

Procedure SendHeartbeatTimerHandler
    Heartbeat.delay = ( $\Delta_{\text{send\_init}}$  +  $\Delta_{\text{send\_min}}$ );
    Send Heartbeat on all links;
    Buffer[ $i$ ]  $\leftarrow$  Heartbeat;
    SetSendHeartbeatTimer( $\pi$ );

Procedure ReceiveHeartbeatTimerHandler
    Set  $T_{\text{reject}} \text{Timer}_k((1 - \rho)T_{\text{reject}})$ ;
    Node[ $k$ ].Status  $\leftarrow$  failed;
    Node[ $k$ ].Last_seq_no  $\leftarrow$  -1;
    Buffer[ $k$ ]  $\leftarrow$  null;

Procedure StartupTimerHandler
    for  $j = 0$  to  $n - 1$ ,  $j \neq i$ , do
        if (Node[ $j$ ].Status = unknown) then
            Node[ $k$ ].Status  $\leftarrow$  failed;
            Node[ $k$ ].Last_seq_no  $\leftarrow$  -1;
            Buffer[ $k$ ]  $\leftarrow$  null;
        endif;
    endfor;

```

Figure 3: **Pseudocode for Algorithm ForwardHeartbeat (Timer Handler Portion)**

**Definition 13** *The maximum number of neighbors of any node in the network is denoted by  $d$ .*

**Definition 14** *The maximum time between a node initiating a heartbeat with a logical sequence number greater than 0 and a heartbeat with the same or a higher logical sequence number being received by all non-neighboring nodes that were working from heartbeat initiation until heartbeat receipt is denoted by  $D_{\text{maxn}}$ .*

**Definition 15** *The maximum time between a node initiating a heartbeat with a logical sequence number equal to 0 and a heartbeat with the same or a higher logical sequence number being received by all non-neighboring nodes that were working from heartbeat*

initiation until heartbeat receipt is denoted by  $D_{\max 0}$ .

**Definition 16** *The timeout period, denoted by  $T_{\text{timeout}}$ , is the amount of time a node waits for the next heartbeat after receiving an earlier heartbeat before detecting a fault event.*

For concise presentation in the following analysis, we define three quantities -  $q$ ,  $m$  and  $r$ .  $q$  is defined such that if a node  $X$  receives a heartbeat with sequence number  $a$ , then the next heartbeat it receives from node  $X$  could have a sequence number of  $a + q$ .  $m$  and  $r$  are quantities such that  $[(r - 1)[d(n - 1)\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}] + (m + 1)(\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}) + 2(\Delta_{\text{send\_init}} - \epsilon) - q\pi]$  is greater than or equal to zero and as small as possible. The actual meaning of  $m$  and  $r$  will become evident in the proof of lemma 6.

**Lemma 1** *The minimum time between a node initiating a heartbeat and a heartbeat with the same or a higher sequence number being received at some non-neighboring node, denoted by  $D_{\min}$ , is  $2(\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}})$ .*

**Proof:** Consider two non-neighboring nodes  $X$  and  $Y$ . Minimum delay in the propagation of a heartbeat from node  $X$  to node  $Y$  will occur when nodes  $X$  and  $Y$  are separated by only one node, with the intermediate node relaying the heartbeat as soon as it receives it. In addition, the heartbeat encounters minimum delays when traversing the communication links. Hence,  $D_{\min}$  is  $2(\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}})$ . ■

**Lemma 2** *The timeout period,  $T_{\text{timeout}}$ , is  $(1+2\rho)\pi+(1+\rho)(D_{\max n}-\text{Heartbeat.delay})$ , where  $\text{Heartbeat.delay}$  is the value stored in the delay field of the last heartbeat received.*

**Proof:** Suppose an arbitrary node  $X$  initiates a heartbeat at real time  $t$ . That heartbeat will arrive at node  $Y$  at real time  $t + \text{Heartbeat.delay}$  at the earliest. Node

$X$  initiates its next heartbeat at real time  $t + (1 + \rho)\pi$ , which will reach node  $Y$  at real time  $t + (1 + \rho)\pi + D_{\max n}$  at the latest. Node  $Y$  should not have timed out in this interval. Hence, the timeout period  $T_{\text{timeout}}$  is  $(1 + \rho)[(1 + \rho)\pi + D_{\max n} - \text{Heartbeat.delay}]$ . Since  $\rho$  is assumed to be very small, this reduces to  $(1 + 2\rho)\pi + (1 + \rho)(D_{\max n} - \text{Heartbeat.delay})$ . ■

**Lemma 3** *The minimum diagnostic latency, staleness and the state holding times for Algorithm ForwardHeartbeat are achieved when  $W = 0$ .*

**Proof:** Consider an arbitrary node  $X$  in the network that is in the failed state for a long time. Since the connectivity of the network is  $k$ , we assume that node  $X$  has only  $k$  neighboring nodes. With the number of failed nodes in the system limited to  $k - 1$ , and with node  $X$  already in the failed state, there can be a maximum of  $k - 2$  neighboring nodes in the failed state. At real time  $t$ , node  $X$  recovers and at the same instant one of the two neighboring nodes that is in the working state fails. Node  $X$  waits for a time  $W$  on its clock before sending out its first heartbeat. Lets say the only other neighboring node that is in the working state fails as soon as it receives the first heartbeat from node  $X$  and at that same instant one of the neighboring nodes that is in the failed state for a long time recovers. This node waits for a time  $W$  on its clock before sending out its first heartbeat. Node  $X$ , upon receiving this heartbeat, sends out its heartbeat again. The working neighboring node cannot fail as it recovered after node  $X$  recovered, hence it propagates this heartbeat to its neighboring nodes. The working node's neighbors could be node  $X$ 's neighboring nodes that are in the failed state, and one other node that is not a neighbor of node  $X$ . This is required because a network with a connectivity of  $k$  has  $k$  disjoint paths between any two pair of nodes.

The working node that receives the heartbeat from the neighboring node of node  $X$  fails as soon as it receives the heartbeat and one of the other neighboring nodes

of node  $X$  comes up and the same process continues. From this analysis, we can see that the working state holding time,  $SHT_w$ , is atleast  $(k - 1)W$ .

The above analysis also shows that the time taken to propagate the first heartbeat throughout the network is also atleast  $(k - 1)W$ . Doing a similar analysis for a node that initiates a heartbeat after being in the working state for more than  $\pi$  (the starting condition being  $(k - 1)$  nodes of the  $k$  neighboring nodes are in the failed state for a long time) shows that  $D_{\max n}$  is also atleast  $(k - 1)W$ .

The worst case considered for deriving the failed state holding time is similar to the case considered for completely connected networks. Assume, at real time  $t$ , node  $X$  sends a heartbeat which reaches some other node  $Y$  in the network at real time  $t + D$ . Node  $X$  fails at real time  $t + T_{\text{ON}}$ , where  $T_{\text{ON}}$  is the minimum length of time node  $X$  remains in the working state after initiating a heartbeat such that  $D - T_{\text{ON}}$  is maximized. Node  $X$  remains in the failed state for  $SHT_f$  and recovers at real time  $t + T_{\text{ON}} + SHT_f$ . Node  $X$  now waits for time  $W$  on its clock before initiating its first heartbeat which reaches node  $Y$  with a delay of  $D_{\min}$ . We require node  $Y$  to timeout, declare node  $X$  to be in the failed state and the  $T_{\text{reject}}$  timer to expire before node  $Y$  receives the first heartbeat from node  $X$  after node  $X$  recovered. This translates to the following inequality.

$$t + T_{\text{ON}} + SHT_f + (1 - \rho)W + D_{\min} > t + D + (1 + \rho)T_{\text{timeout}} + (1 + \rho)T_{\text{reject}}$$

Since  $T_{\text{timeout}}$  has a  $D_{\max n}$  term in it which is atleast  $(k - 1)W$ ,  $SHT_f$  is greater than  $(k - 2)W$ .

Hence, unlike the case of completely connected networks, in Algorithm Forward-Heartbeat, reducing  $W$  minimizes the state holding times and the latency. Hence,  $W$  is set to zero so that the minimum state holding times and diagnostic latency can be achieved. ■

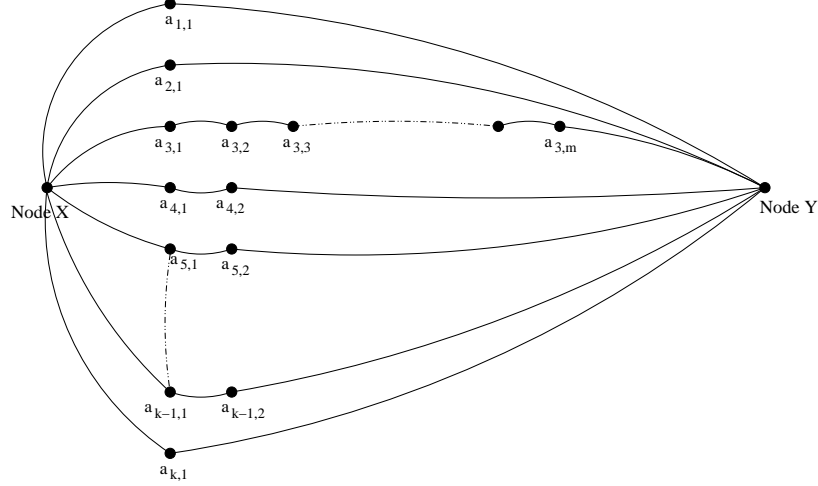


Figure 4: Network for proof of lemma 4

**Lemma 4** *The working state holding time,  $SHT_w$ , is  $(d(k-3)(n-1) + 2n + k - 6)\Delta_{\text{send\_init}} + (n+k-6)\Delta_{\text{send\_max}} - (k-2)\epsilon$  and  $D_{\text{max}0}$  is  $(d(k-2)(n-1) + n+k-4)\Delta_{\text{send\_init}} + (n+k-4)\Delta_{\text{send\_max}} - (k-2)\epsilon$ , where  $\epsilon$  is an arbitrarily small quantity, if the failed state holding time is atleast  $D_{\text{max}0}$ .*

**Proof:** Consider two arbitrary nodes  $X$  and  $Y$  in the network. Since the network has a connectivity of  $k$ , there exists  $k$  disjoint paths between nodes  $X$  and  $Y$ . In the worst case, all  $n$  nodes in the network are present in the  $k$  disjoint paths. Consider the network shown in Figure 4. To begin with, we assume that there are no links connecting any two paths. Let  $a_{i,j}$  denote the  $j^{\text{th}}$  node present in the  $i^{\text{th}}$  row of the network.

The worst case is constructed in the following way. Initially, nodes  $X$ ,  $a_{3,1}$ ,  $a_{4,1}$ , ...,  $a_{k,1}$  are in the failed state for time greater than  $SHT_f$  and all other nodes are

in the working state for time greater than  $SHT_w$ . At real time  $t$ , node  $X$  recovers and node  $a_{1,1}$  fails. The first heartbeat sent by node  $X$  is received only by node  $a_{2,1}$ . Node  $a_{2,1}$  fails just before it could pass on the heartbeat to node  $Y$  and at the same instant node  $a_{3,1}$  recovers.

$$\text{Time elapsed} = \Delta_{\text{send\_init}} + \Delta_{\text{send\_max}} + \Delta_{\text{send\_init}} - \epsilon$$

where  $\epsilon$  is the smallest time such that a heartbeat sent in  $\Delta_{\text{send\_init}} - \epsilon$  time is an invalid heartbeat.

Node  $a_{3,1}$  recovers, sends a heartbeat to all its neighbors and ultimately forwards node  $X$ 's heartbeat to node  $a_{3,2}$ . Node  $a_{3,1}$  cannot fail because it recovered after node  $X$  recovered, and the amount of time node  $X$  is required to be in the working state is the working state holding time. The heartbeat is propagated till node  $a_{3,m}$ , where  $m$  is the maximum number of nodes possible in the  $3^{\text{rd}}$  row. Node  $a_{3,m}$  fails just before it could forward the heartbeat to node  $Y$  and at the same instant of time node  $a_{4,1}$  recovers.

$$\text{Time Elapsed} = d(n-1)\Delta_{\text{send\_init}} + (m+1)(\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}) - \epsilon$$

When a node recovers, it takes a maximum of  $\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}$  time for the first heartbeat to reach its neighboring nodes. The node that just recovered starts receiving buffered heartbeat messages from its neighbors after an additional time of  $\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}$ . Each buffered heartbeat that is received is assumed to be propagated onto the remaining links. Hence, the node that just recovered takes an additional  $d(n-1)\Delta_{\text{send\_init}}$  before it could start processing the heartbeat message we are interested in.

Node  $a_{4,1}$  recovers, passes on node  $X$ 's heartbeat to node  $a_{4,2}$  and node  $a_{4,2}$  fails just before it could pass on the heartbeat to node  $Y$ . (Node  $a_{4,2}$  is directly connected to node  $Y$ )

$$\text{Time Elapsed} = d(n-1)\Delta_{\text{send\_init}} + 3(\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}) - \epsilon$$

The same situation as described for row 4 happens in rows 5 to  $k-1$ .

$$\text{Time Elapsed} = (k-5)[d(n-1)\Delta_{\text{send\_init}} + 3(\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}) - \epsilon$$

There is only one node in row  $k$ . Node  $a_{k,1}$  recovers and just passes on node  $X$ 's heartbeat to node  $Y$ .

$$\text{Time Elapsed} = d(n-1)\Delta_{\text{send\_init}} + 2\Delta_{\text{send\_init}} + 3\Delta_{\text{send\_max}}$$

From Figure 1, we can see that assuming all  $n$  nodes in the network are present in the diagram,  $m = n - 2k + 3$ . Hence, the total time elapsed since time  $t$  at which node  $X$  first initiated a heartbeat is

$$D_{\text{max}0} = (d(k-2)(n-1) + n + k - 4)\Delta_{\text{send\_init}} + (n + k - 4)\Delta_{\text{send\_max}} - (k-2)\epsilon \quad (1)$$

and the working state holding time

$$SHT_w = (d(k-3)(n-1) + 2n + k - 6)\Delta_{\text{send\_init}} + (n + k - 6)\Delta_{\text{send\_max}} - (k-2)\epsilon \quad (2)$$

It is easy to see that the latency in detecting a recovery event is  $D_{\text{max}0}$ . ■

**Lemma 5** *The maximum time taken by a heartbeat that has a logical sequence number greater than zero to reach every node that is in the working state since the time the heartbeat was initiated, denoted by  $D_{\text{max}n}$ , is  $d(k-1)(n-1)\Delta_{\text{send\_init}} + (n+k-2)(\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}) - (k-1)\epsilon$  if the failed state holding time is atleast  $D_{\text{max}n}$ .*

**Proof:** Consider an arbitrary node  $X$  that is in the working state for time greater than  $\pi$  (so that we can now consider heartbeats that have a logical sequence number

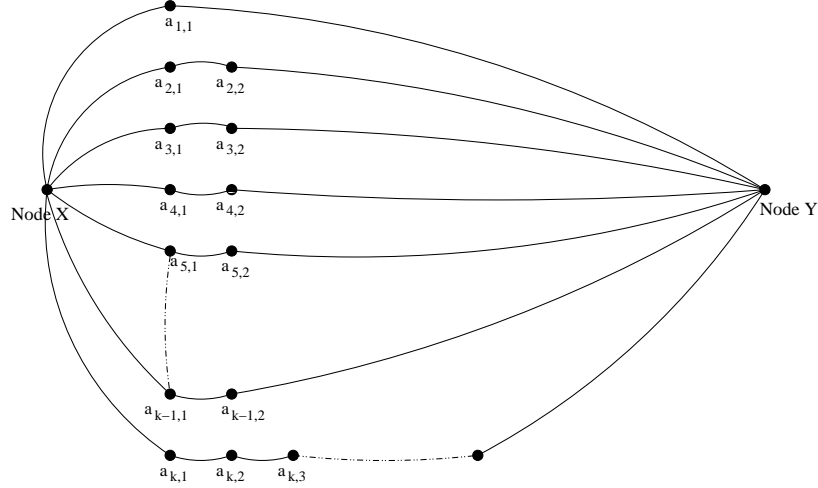


Figure 5: **Network for proof of lemma 5**

greater than zero). At real time  $t$ , node  $X$  initiates a heartbeat. Since the failed state holding time,  $SHT_f$ , is atleast  $D_{\max n}$ , any node that fails at or after real time  $t$  cannot recover until the heartbeat initiated by node  $X$  at time  $t$  reaches all nodes in the system that were in the working state since time  $t$ . Further, since the number of nodes that can be in the failed state is limited to  $k - 1$ , the worst case for calculating  $D_{\max n}$  is arrived at by having only  $k$  neighbors for node  $X$ .

We construct the worst case in the following way. At real time  $t$ ,  $(k - 1)$  of the  $k$  neighboring nodes of node  $X$  are in the failed state for time greater than  $SHT_f$ . Node  $X$  initiates a heartbeat at time  $t$ , which is received by the neighboring node that as in the working state. This working neighboring node fails just before it could forward the complete heartbeat to its neighboring nodes, and at the same instant of time one of the neighboring nodes that is in the failed state recovers. This node sends out a heartbeat indicating its recovery which prompts node  $X$  to forward its

buffered heartbeats to it. As seen in the proof of lemma 4, this neighboring node cannot fail before it could forward the heartbeat it received. Hence, the node that just recovered forwards the heartbeat to its neighbors. In the worst case, this node could have the  $(k - 1)$  failed nodes as its neighbors, and one other working node which is not a neighbor of node  $X$ . This one other working node should not be a neighbor of node  $X$  because a graph with a connectivity of  $k$  has  $k$  disjoint paths between any two pair of nodes.

The working node, which is not a neighboring node of node  $X$ , fails just before it could forward it to its neighbors, and at the same instant of time one of the failed nodes that is a neighbor of node  $X$  recovers. This same process repeats until the  $k$ th neighbor of node  $X$  recovers. No more nodes can fail because all nodes that are in the failed state at this point in time cannot recover until  $D_{\max n}$  time has elapsed since they failed and the number of nodes that can be in the failed state at any given instant of time is limited to  $k - 1$ .

We assume that the heartbeat that propagates through the  $k$ th neighboring node of node  $X$  passes through all the remaining working nodes in the system. This is possible if the remaining working nodes form a linear chain between them, with each working node having its remaining  $k - 1$  neighbors as the nodes that are currently in the failed state. A sketch of this network is shown in Figure 5.

Hence,  $D_{\max n}$  reduces to the following expression.

$$\begin{aligned} & \Delta_{\text{send\_init}} + \Delta_{\text{send\_max}} + \Delta_{\text{send\_init}} - \epsilon + (k - 2)[d(n - 1)\Delta_{\text{send\_init}} + 3(\Delta_{\text{send\_init}} + \\ & \Delta_{\text{send\_max}}) - \epsilon] + \\ & d(n - 1)\Delta_{\text{send\_init}} + 2\Delta_{\text{send\_init}} + 3\Delta_{\text{send\_max}} + (n - 2k)(\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}) \end{aligned}$$

Simplifying the above expression for  $D_{\max n}$  gives the expression stated in the lemma. ■

**Lemma 6** *The maximum possible time a heartbeat can exist in the network, denoted by  $T_{\text{exist}}$ , is  $(q + 1 + 3\rho)\pi + (1 + 2\rho)D_{\text{maxn}} + (n - m + r)(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}})$ .*

**Proof:** Let node  $X$  issue a heartbeat at time  $t$ . This heartbeat will be either replaced by a newer heartbeat or will cease to exist in the network if node  $X$  fails and other nodes timeout. The worst case is when node  $X$  fails sometime after sending a heartbeat so that the heartbeat issued is removed by having nodes timeout waiting for the next heartbeat from node  $X$ . Let this heartbeat reach node  $Y$  with a delay  $D$ . Hence, node  $Y$  will timeout waiting for the next heartbeat from node  $X$  at time  $t_1 = t + D + (1 + 3\rho)\pi + (1 + 2\rho)(D_{\text{maxn}} - \text{heartbeat.delay})$  at the latest. Consider another node  $Z$  in the network such that one of the paths connecting nodes  $Y$  and  $Z$  has  $k - 1$  nodes in it, with all these  $k - 1$  nodes in the failed state. Let the  $k - 1$  nodes be denoted by  $a_1, a_2, \dots, a_{k-1}$ . At time  $t_1 - k(\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}})$ , node  $Y$  receives a heartbeat from node  $a_1$  indicating the recovery of node  $a_1$ . Node  $Y$  forwards the buffered heartbeat to node  $a_1$ , adding  $(\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}})$  to the *heartbeat.delay* field. Node  $a_1$ , forward the heartbeat to node  $a_2$  because it just received a heartbeat from node  $a_2$  indicating node  $a_2$ 's recovery. Assuming the same situation repeats itself for the remaining  $k - 2$  failed nodes, node  $Z$  will receive a buffered heartbeat regarding node  $X$  at time  $t + T_{\text{exist}} = t + D + (1 + 3\rho)\pi + (1 + 2\rho)(D_{\text{maxn}} - \text{heartbeat.delay}) + k(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}})$ .

$D - \text{heartbeat.delay}$  will be maximum under the following circumstances. Assume node  $X$  has only  $k$  neighboring nodes. Consider the  $k$  disjoint paths connecting node  $X$  and node  $Y$ . At real time  $t$ , all but one of the  $k$  neighboring nodes of node  $X$  is in the working state. The path between node  $X$  and node  $Y$  that has the working neighboring node of node  $X$  has  $n - k - 1$  nodes in it. Node  $X$  initiates a heartbeat at real time  $t$ , which is propagated right away through the  $n - k - 1$  nodes to node  $Y$ , with the communication delay across a single link always being  $\Delta_{\text{send\_max}}$ . Hence,  $\max(D - \text{heartbeat.delay})$  reduces to  $(n - k)(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}})$ .

The above value of  $\max(D - \text{heartbeat.delay})$  is valid if  $\pi$  is large. If  $\pi$  is sufficiently small, then if node  $X$  issues a heartbeat with a sequence number  $n$ , then considering a situation similar to the one considered for calculating  $D_{\max n}$ , node  $Y$  may end up getting a heartbeat with a sequence number  $n + q$ . The largest value of  $q$  is  $\left\lfloor \frac{D_{\max n} - \Delta_{\text{send\_init}} - \Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}}{(1 - \rho)\pi} \right\rfloor$ . Considering a sequence of events similar to the one considered for calculating  $D_{\max n}$ , let  $r$  denote the number of rows that do not get a heartbeat with sequence number  $n + q$  and let  $m$  nodes be contained in the  $r$  rows. The reason this sequence of events is adopted is because the *heartbeat.delay* field is initialized to zero by node  $X$  for every new heartbeat it initiates for the first time.

Then  $D$  is given by

$$q\pi + (n - m + r - k)(\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}})$$

The situation described earlier corresponds to  $r = 0$ , implying  $\pi$  is sufficiently large such that  $q = 0$ . The above expression holds when  $r > 0$ .

Hence, the value of  $\max(D - \text{heartbeat.delay})$  is

$$q\pi + (n - m - k + r)(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}})$$

Substituting this value in the derived expression for  $T_{\text{exist}}$  gives the result stated in the lemma. ■

**Lemma 7** *The heartbeat rejection time,  $T_{\text{reject}}$ , is  $(q + (q + 2)\rho)\pi + 2\rho D_{\max n} + (1 + \rho)(n - m + r)(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}})$ .*

**Proof:**

Let node  $X$  initiate a heartbeat at real time  $t$ . Node  $Y$  receives the heartbeat with delay  $D$ . Hence, node  $Y$  will timeout waiting for the next heartbeat from node  $X$  at time  $t + D + (1 + \rho)\pi + D_{\max n} - \text{heartbeat.delay}$  and will reject any heartbeat it may

receive regarding node  $X$  until time  $t + D + \pi + D_{\max n} - \text{heartbeat.delay} + (1 - \rho)T_{\text{reject}}$  at the earliest. Node  $Y$  could get a stale heartbeat at time  $t + T_{\text{exist}}$ . We require that node  $Y$  still rejects heartbeats regarding node  $X$  at this point in time. This translates to the following inequality:

$$t + \min(D - \text{heartbeat.delay}) + (1 + \rho)\pi + D_{\max n} + (1 - \rho)T_{\text{reject}} > t + T_{\text{exist}}$$

$\min(D - \text{heartbeat.delay}) = 0$ . Simplifying the above inequality gives the result stated in the lemma. ■

**Lemma 8** *The failed state holding time,  $SHT_f$ , is  $(q+1+(2q+5)\rho)\pi+(1+4\rho)D_{\max n} - D_{\min} - \Delta_{\text{send\_init}} + (2n - m + r + 2\rho(n - m + r))(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}})$ .*

**Proof:** The failed state holding time should be large enough so that the first heartbeat sent after a node recovers reaches other nodes after they have timed out, declared the node to be faulty and the  $T_{\text{reject}}$  timer has expired. In the worst case, let node  $X$  send a heartbeat at time  $t$  and fail at time  $t + T_{\text{ON}}$ . This heartbeat reaches some other node  $Y$  with a delay  $D$  and the delay field in the received heartbeat has a value  $\text{heartbeat.delay}$ . Let node  $Y$  timeout and declare node  $X$  to be faulty at time  $t + D + (1 + 3\rho)\pi + (1 + 2\rho)(D_{\max n} - \text{heartbeat.delay})$  at the latest, with node  $Y$  being in the working state since the time node  $X$  initiated the heartbeat. Consider another node  $Z$  that recovers at a time when all nodes except node  $Y$  have timed out. Let node  $Z$  and node  $Y$  be connected by a path containing  $k - 1$  nodes, with a situation similar to the one considered in the proof of lemma 6. Then node  $Z$  will experience the  $\max(D - \text{heartbeat.delay} - T_{\text{ON}})$  given by  $N(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}) - \Delta_{\text{send\_init}}$ . Node  $Z$  will not accept any further heartbeats regarding node  $X$  for time  $(1 + \rho)T_{\text{reject}}$  after it times out.

Hence, the first heartbeat sent by node  $X$  at time  $t + T_{\text{ON}} + SHT_f$  must reach node  $Z$  after this point in time. This can be expressed as the following inequality.

$$\begin{aligned}
t + T_{\text{ON}} + SHT_f + D_{\text{min}} &> t + D + (1 + 3\rho)\pi \\
&+ (1 + 2\rho)(D_{\text{maxn}} - \text{heartbeat.delay}) + (1 + \rho)T_{\text{reject}}
\end{aligned}$$

which when simplified can be expressed as

$$\begin{aligned}
SHT_f &> (1 + 3\rho)\pi + (1 + \rho)T_{\text{reject}} + (1 + 2\rho)D_{\text{maxn}} - D_{\text{min}} \\
&+ \max(D - (1 + 2\rho)\text{heartbeat.delay} - T_{\text{ON}})
\end{aligned}$$

From the analysis of  $\max(D - \text{heartbeat.delay})$ , it can be seen that the delay factor excluded when  $\pi$  is sufficiently small is now taken care of by  $T_{\text{ON}}$ . Hence, the situation that will yield the maximum value of  $(D - (1 + 2\rho)\text{heartbeat.delay} - T_{\text{ON}})$  is when  $k - 1$  of the  $k$  neighbors of node  $X$  are in the failed state and the last heartbeat initiated by node  $X$  before it could fail is propagated right away through the longest path in the network. With  $n$  nodes in the network and one (failed) node in each of the remaining  $k - 1$  paths, the longest path connecting node  $X$  and node  $Y$  will have  $n - k - 1$  nodes. Hence,  $\max(D - (1 + 2\rho)\text{heartbeat.delay} - T_{\text{ON}})$  will be  $(n - k)(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}) - \Delta_{\text{send\_init}}$  assuming the node that received the heartbeat is in the working state since the time the heartbeat was first initiated by node  $X$ . Now consider another node  $Z$  that has a path to node  $Y$  having  $k - 1$  nodes in it. If node  $Z$  recovers  $k(\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}})$  time before node  $Y$  could timeout, and the heartbeat propagates through the  $k - 1$  nodes to  $Z$ , then node  $Z$  will receive the heartbeat with  $\max(D - (1 + 2\rho)\text{heartbeat.delay} - T_{\text{ON}})$  equal to  $n(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}) - \Delta_{\text{send\_init}}$ . Substituting this value in the above expression and simplifying gives the result stated in the lemma. Note that with  $\max(D - (1 + 2\rho)\text{heartbeat.delay} - T_{\text{ON}})$  a positive quantity and  $T_{\text{reject}}$  greater than  $D_{\text{min}}$ , the expression for the failed state holding time derived above is greater than  $D_{\text{maxn}}$  and  $D_{\text{max0}}$  which was assumed in the earlier lemmas. ■

**Corollary 1** *At any instant of time, if some working nodes maintain a view of working for an arbitrary node  $X$ , they are referring to the same instance of the working state of node  $X$ . The instance referred to is the current state of node  $X$  if node  $X$  is currently in the working state, or, if node  $X$  is currently in the failed state, it refers to the working state that was held by node  $X$  just prior to the current failed state.*

**Proof:** From theorem 8, we have that the failed state holding time is set so large that when the first heartbeat sent by a node after it recovers reaches other nodes, all working nodes in the system do not hold a status of *working* for the node that recovered. Hence, if a node  $Y$  holds a view of *working* for node  $X$ , then either node  $X$  is currently in the *working* state, or node  $X$  is currently in the *failed* state and node  $Y$  is yet to timeout and detect the failure event. ■

## 4.2.2 Diagnosis of neighboring nodes

**Lemma 9** *Assume an arbitrary node  $X$  is in the working state continuously for sufficiently long to send two consecutive heartbeats to a neighboring node  $Y$ . The maximum time between the two heartbeats arriving at  $Y$ ,  $\Delta_{\text{heartbeat}}$ , is  $(1 + \rho)\pi + \Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}$ .*

**Proof:** Suppose  $X$  initiates the first heartbeat at time  $t$ . That heartbeat will arrive at  $Y$  at time  $t + \Delta_{\text{send\_init}} + \Delta_{\text{send\_min}}$  at the earliest. By Definition 9,  $X$  will initiate its next heartbeat at time  $t + (1 + \rho)\pi$  at the latest. This heartbeat will arrive at  $Y$  at time  $t + (1 + \rho)\pi + \Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}$  at the latest. Subtracting the earliest and latest arrival times on  $y$  yields the maximum heartbeat interarrival time stated in the lemma. ■

Lemma 9 can now be used to prove Lemmas 10 and 11, which provide the desired performance limits for any algorithm.

**Lemma 10** *The diagnostic latency and staleness of any algorithm that achieves bounded correctness are both at least*

$$(1 + 3\rho)\pi + 2(1 + \rho)\Delta_{\text{send\_max}} - (1 + 2\rho)\Delta_{\text{send\_min}}$$

**Proof:** The worst case latency for the detection of node  $X$ 's failure by a neighboring node  $Y$  occurs if  $X$  fails immediately after initiating a heartbeat to  $Y$ . If  $t$  represents the heartbeat initiation time,  $X$  will fail at time  $t + \Delta_{\text{send\_init}}$  and the heartbeat will be received by  $Y$  at time  $t + \Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}$  at the latest. Factoring in clock drift,  $Y$  must then wait  $(1 + \rho)\Delta_{\text{heartbeat}}$  time on its local clock without receiving a heartbeat before concluding that  $X$  has failed. Thus, in real time, if we ignore negligible  $\rho^2$  terms it could be as late as

$$\begin{aligned} t + \Delta_{\text{send\_init}} + \Delta_{\text{send\_max}} + (1 + 2\rho)\Delta_{\text{heartbeat}} = \\ t + \Delta_{\text{send\_init}} + (1 + 3\rho)\pi + 2(1 + \rho)\Delta_{\text{send\_max}} - (1 + 2\rho)\Delta_{\text{send\_min}} \end{aligned}$$

before  $Y$  can conclude that  $X$  has failed. Subtracting the failure time from the latest detection time yields a maximum failure detection latency of  $(1 + 3\rho)\pi + 2(1 + \rho)\Delta_{\text{send\_max}} - (1 + 2\rho)\Delta_{\text{send\_min}}$ .

We now analyze the latency in detecting a node's recovery. We assume that nodes initiate heartbeat transmission immediately after making a transition from the failed state to the working state because this produces the shortest possible latency for recovery detection. The maximum delay before a working node receives a heartbeat from a recovered node is, therefore,  $\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}$ . Since  $\Delta_{\text{send\_init}} < \pi$ , this is shorter than the failure detection latency derived above. Hence, the worst-case latency is equal to the maximum failure detection latency of  $(1 + 3\rho)\pi + 2(1 + \rho)\Delta_{\text{send\_max}} -$

$$(1 + 2\rho)\Delta_{\text{send\_min}}.$$

For startup time, we also need to consider the amount of time it takes after a node  $X$  returns to the working state before it determines an initial status for each other node in the system. Clearly, if node  $X$  receives a heartbeat from a neighboring node  $Y$  shortly after recovering, it will mark node  $Y$  as working within the diagnostic latency bound. So, the question is how long must node  $X$  wait without receiving a heartbeat from a node  $Y$  before concluding that node  $Y$  is failed? The worst case occurs if a heartbeat arrived from node  $Y$  just prior to node  $X$ 's recovery. In this case,  $X$  must wait for clock time  $(1 + \rho)\Delta_{\text{heartbeat}}$  before it is safe to conclude that node  $Y$  failed. In real time, this could take as long as  $(1 + \rho)(1 + \rho)\Delta_{\text{heartbeat}}$  which is equal to  $(1 + 3\rho)\pi + (1 + 2\rho)(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}})$ . Since this is less than the minimum diagnostic latency and the startup time cannot be less than latency, the startup time is equal to the minimum latency derived above. ■

Lemma 10 states that the minimum diagnostic latency and minimum staleness achievable by any algorithm are slightly greater than  $\Delta_{\text{heartbeat}}$ . Since the arrival times of two consecutive heartbeats from a working node can be separated by  $\Delta_{\text{heartbeat}}$ , the receiving node can not distinguish a failed node's behavior from a good node's behavior unless more than this time passes without a heartbeat being received. This fact is central to the proof of Lemma 10.

**Lemma 11** *The working state holding time and failed state holding time for algorithm ForwardHeartbeat to achieve bounded correctness when diagnosing neighboring nodes are  $\Delta_{\text{send\_init}}$  and  $(1 + 3\rho)\pi + 2(1 + \rho)(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}) - \Delta_{\text{send\_init}}$  respectively.*

**Proof:** From the section on diagnosis of non-neighboring nodes, we have that  $W = 0$ .

If  $\text{SHT}_w < \Delta_{\text{send\_init}}$ , then it is possible for a node  $X$  to transition from the failed state to the working state and back to the failed state prior to successfully initiating a single heartbeat. In this situation, there is no way for any other node to detect the event of node  $X$ 's recovery and the bounded diagnostic latency property is violated. Hence,

$$\text{SHT}_w = \Delta_{\text{send\_init}} \quad (3)$$

This is the minimum time a node must remain in the working state after making a transition into that state.

Now, consider the minimum time a node must remain in the failed state in order for the transition into that state to be detected. Such a transition can not be detected if the node returns to the working state and sends a new round of heartbeats as early as if it had never left the working state. The worst case is if a node fails at time  $t + \Delta_{\text{send\_init}}$  immediately after successfully initiating a heartbeat to another node  $Y$  and returns to the working state in time  $\text{SHT}_f$ . In this situation, the first heartbeat arrives at a neighboring node  $Y$  at time  $t + \Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}$  at the latest. The second heartbeat will arrive at  $Y$  at time  $t + \Delta_{\text{send\_init}} + \text{SHT}_f + \Delta_{\text{send\_init}} + \Delta_{\text{send\_min}}$  at the earliest. If the difference between these arrival times is no greater than  $(1 + 2\rho)\Delta_{\text{heartbeat}}$ , then node  $Y$  can not distinguish this situation from one in which node  $X$  remained in the working state for the entire interval. This yields

$$\text{SHT}_f > (1 + 3\rho)\pi + 2(1 + \rho)(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}) - \Delta_{\text{send\_init}} \quad (4)$$

■

### 4.2.3 Overall Results

To ensure correct diagnosis for all nodes in the network, the larger of the state holding times derived for the diagnosis of non-neighboring nodes and for the diagnosis of neighboring nodes must be taken. Theorem 1 gives these values and states that ForwardHeartbeat is bounded correct with specified latency and start-up time given that the state holding times are maintained. Since the state holding times specified in the theorem is larger than the state holding times required for diagnosis of neighboring nodes, the following theorem holds for diagnosis of all nodes in the network.

**Theorem 1** *With a working state holding time of*

$$(d(k-3)(n-1) + 2n + k - 6)\Delta_{\text{send\_init}} + (n + k - 6)\Delta_{\text{send\_max}} - (k-2)\epsilon$$

*and a failed state holding time of*

$$(q+1 + (2q+5)\rho)\pi + (1+4\rho)D_{\text{maxn}} - D_{\text{min}} - \Delta_{\text{send\_init}} + (2n - m + r + 2\rho(n - m + r))(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}})$$

*Algorithm ForwardHeartbeat achieves bounded correctness with a diagnostic latency( $L$ ) of  $(1+3\rho)\pi + (1+2\rho)D_{\text{maxn}} + n(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}) - \Delta_{\text{send\_init}}$  and a startup time of  $(1+2\rho)T_{\text{exist}}$ .*

**Proof:**

PART 1: BOUNDED DIAGNOSTIC LATENCY

*Case 1a: Fault Event*

We need to prove that, if node  $X$  fails at time  $t$ , then another node  $Y$  diagnoses the fault event by time  $t + L$ .

Assume node  $X$  initiated its last heartbeat at time  $t - t'$  before it could fail. This heartbeat reaches node  $Y$  at time  $t - t' + D$  with the delay recorded in the

heartbeat message being denoted by *heartbeat.delay*. Hence node *Y* will timeout at time  $t - t' + D + (1 + 3\rho)\pi + (1 + 2\rho)(D_{\max n} - \textit{heartbeat.delay})$  at the latest. We require that the heartbeat node *X* sends after it recovers from the fault event that occurred at time  $t$  reaches node *Y* after node *Y* has timed out waiting for the next heartbeat from node *X*. This translates to proving the following inequality.

$$t + SHT_f + D_{\min} > t - t' + D + (1 + 3\rho)\pi + (1 + 2\rho)(D_{\max n} - \textit{heartbeat.delay})$$

which reduces to proving that

$$SHT_f > (1 + 3\rho)\pi + (1 + 2\rho)D_{\max n} - D_{\min} + \max(D - (1 + 2\rho)\textit{heartbeat.delay} - t')$$

Substituting  $n(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}) - \Delta_{\text{send\_init}}$  for  $\max(D - (1 + 2\rho)\textit{heartbeat.delay} - t')$  shows that the inequality is true. Hence, a failure event is always detected.

#### *Case 1b: Recovery Event*

From case 1a, we have that a fault event is always detected. Hence, a subsequent recovery event will be detected if the working state holding time is large enough so that all nodes in the working state since the recovery event occurred receive the heartbeat sent after the node recovered. From lemma 4, we have that the working state holding time,  $SHT_w$ , satisfies this condition.

Next, we require that node *Y* does not reject any heartbeat it receives from node *X* after node *X* has recovered. From the proof of lemma 7, we have that it is precisely this situation that is avoided in deriving the value of  $T_{\text{reject}}$ .

Hence, a recovery event is always detected with a latency  $D_{\max 0}$  which is less than the latency in detecting a failure event.

## PART 2: BOUNDED STARTUP

Consider an arbitrary node  $X$  recovering at time  $t$ . By time  $t + S$ , it diagnoses every node in the system. We need to prove that the state diagnosed at time  $t + S$  for an arbitrary node  $Y$  is  $L$ -valid. For algorithm ForwardHeartbeat, we have  $S$  equal to  $(1 + 2\rho)T_{\text{exist}}$ .

### *Case 2a: Status of working held at time $t + S$*

Any heartbeat initiated by node  $Y$  before time  $t$  will no longer exist at time  $t + T_{\text{exist}}$ . The situation that will yield the highest staleness value is when node  $X$  is about to timeout and thereby detect a fault event at time  $t$ , with node  $Y$  failing at some time before  $t + S$ . Hence, if a status of *working* is held for node  $Y$  at time  $t + S$ , then the heartbeat must have been initiated at time  $t' = t + S - (1 + 3\rho)\pi - D - (1 + 2\rho)D_{\text{maxn}} + (1 + 2\rho)\text{heartbeat.delay}$ , just before node  $X$  could timeout. Assume node  $Y$  failed at time  $t' + T_{\text{ON}}$ . Then the state detected at time  $t + S$  by node  $X$  for node  $Y$  is  $t + S - (t' + T_{\text{ON}})$  valid. This expression reduces to  $(1 + 3\rho)\pi + (1 + 2\rho)D_{\text{maxn}} + D - (1 + 2\rho)\text{heartbeat.delay} - T_{\text{ON}}$ . From the proof of lemma , we have the maximum of this quantity as  $(1 + 3\rho)\pi + (1 + 2\rho)D_{\text{maxn}} + n(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}) - \Delta_{\text{send\_init}}$  which is equal to the latency  $L$ . Hence, if a status of *working* is held at time  $t + S$ , then that view is  $L$  valid.

We have assumed here that the accuracy condition is not violated for diagnosis performed by node  $X$  before time  $t + S$ . More specifically, the above proof assumes a spurious recovery event was not detected before time  $t + S$ . When node  $X$  times out and diagnoses a fault event on node  $Y$ , it rejects any heartbeat from node  $Y$  that it may receive for a time  $T_{\text{reject}}$  after detecting the fault event. We require that  $T_{\text{reject}}$  is large enough so that node  $X$  cannot receive a stale heartbeat and thereby detect a spurious recovery event. Assuming the last heartbeat initiated by node  $Y$  at time  $t'$  was received by node  $X$  with a delay  $D$ , we require the following inequality to be

satisfied so that a spurious recovery event is not detected.

$$t' + (1 + \rho)\pi + D_{\max n} + \min(D - \text{heartbeat.delay}) > t' + T_{\text{exist}}$$

Substituting the derived values for the various quantities involved shows that the above inequality is satisfied. Hence, a spurious recovery event is never detected.

*Case 2b: Status of failed held at time  $t + S$*

If node  $Y$  were to be in the working state during the entire interval  $[t + S - \pi - D_{\max n}, t + S - D_{\min}]$ , then node  $Y$  would have definitely issued a heartbeat which would have reached node  $X$  such that node  $X$  still has not timed out waiting for the next heartbeat from node  $Y$  at time  $t + S$ . This is because if, say, node  $Y$  issued a heartbeat at time  $t''$  say, then the earliest time node  $X$  will timeout after receiving this heartbeat is  $t'' + \pi + D_{\max n} + \min(D - \text{heartbeat.delay})$ . With  $\min(D - \text{heartbeat.delay})$  equal to zero, the earliest time node  $X$  would timeout is  $t'' + \pi + D_{\max n}$ . Since node  $X$  holds a status of failed at time  $t + S$ , node  $Y$  must have been in the failed state some time during the interval  $[t + S - \pi - D_{\max n}, t + S - D_{\min}]$ . Since  $\pi + D_{\max n} < L$ , the state detected is  $L$  valid.

**PART 3: ACCURACY**

We first prove that no events are detected more than once. Consider a false detection of a recovery event on node  $Y$  by node  $X$  at real time  $t$ . This could occur if the  $T_{\text{reject}}$  value is small so that node  $X$  accepts stale heartbeats issued by node  $Y$  before it could fail as an indication of a recovery event on node  $Y$ . To prevent this, we require that node  $X$  rejects stale heartbeats from node  $Y$  as long as there is a possibility of receiving a stale heartbeat exists. If node  $Y$  initiated a heartbeat at real time  $t$  and then failed, then from lemma 6 we have that such a heartbeat exists in the

network for a maximum of  $T_{\text{exist}}$  time. The earliest time node  $X$  detects the failure event would be  $t + D + \pi + D_{\text{maxn}} - \text{heartbeat.delay} = t + \pi + D_{\text{maxn}}$ . Node  $X$  rejects heartbeats from node  $Y$  for a period of  $T_{\text{reject}}$  after this. Hence we require  $t + \pi + D_{\text{maxn}} + T_{\text{reject}} > t + T_{\text{exist}}$ . Substituting the values for the quantities involved shows that the inequality is satisfied. Hence, it is not possible for a recovery event that is detected to not correspond to an actual recovery event.

Next, consider the false detection of a failure event. The timeout value derived in lemma 2 is set such that if a node is continuously in the working state, then successive heartbeats reach other working nodes before they could timeout. Hence, it is not possible for a fault event that is detected to not correspond to an actual fault event.

Next, we prove that an event recorded for node  $Y$  by node  $X$  corresponds to an actual event that occurred on node  $Y$ . As in the proof of Algorithm Bounded-CorrectComplete, we show that for any state transition recorded for node  $Y$  by a working node  $X$ , if the original state is valid, then the transition preserves accuracy and results in a valid final state. From part 2 of this proof we have that the initial states held by a working node for every other node is valid within its startup time after recovery, thereby preserving accuracy.

Node  $X$  detects a recovery event on node  $Y$  by receiving a heartbeat from node  $Y$  while holding a status of *failed* for node  $Y$ . Let node  $X$  receive such a heartbeat from node  $Y$  at time  $t$ . Since node  $X$  held a status of *failed* for node  $Y$  at time  $t$ , node  $Y$  must have been in that state at some time  $t - \max(L, S)$  or later. From lemma 7, we have that node  $X$  rejects any stale heartbeat it may receive after node  $X$  detected the fault event on node  $Y$ . From lemma 8, we have that the failed state holding time is so large such that the first heartbeat sent after a node recovers reaches other nodes after they have gone beyond the  $T_{\text{reject}}$  phase and accept heartbeats from the node under consideration. Hence, the heartbeat received by node  $X$  at time  $t$  must

have been sent by node  $Y$  at some time in the interval  $[t - D_{\max0}, t - D_{\min}]$ . It must be noted here that node  $X$  is in the working state when node  $Y$  recovers, because  $S > L$ , and we are proving accuracy of the diagnosis performed by node  $X$  after it has performed diagnosis for time  $S$ . Since  $D_{\max0} < \max(L, S)$ , node  $Y$  must have indeed experienced a recovery event sometime in the interval  $[t - \max(L, S), t - D_{\min}]$ . Hence, the recovery event detected corresponds to an actual recovery event.

Node  $X$  detects a failure event on node  $Y$  by timing out waiting for the next heartbeat from node  $Y$ . From lemma 2, we have the timeout value set such that node  $X$  does not timeout waiting for a heartbeat from node  $Y$  if node  $Y$  is in the working state continuously. Since node  $X$  held a status of *working* for node  $Y$  at time  $t$ , node  $Y$  must have been in the working state at time  $t - \max(L, S)$  or later. Since node  $X$  timed out waiting for a new heartbeat from node  $Y$  at time  $t$ , node  $Y$  must have failed sending a heartbeat at its scheduled interval of  $\pi$  time units. Hence node  $Y$  must have failed some time after  $t - \max(L, S)$ . Hence, the failure event detected corresponds to an actual fault event that occurred on node  $Y$ .

■

The analysis considered thus far assumed that the heartbeat sequence number is a monotonically increasing quantity, or, in other words, the heartbeat sequence number assumed was a logical sequence number. In practice, due to the finite length of the field used to store the heartbeat sequence number, the maximum number that can be stored in the field (denoted by  $MAX\_SEQ\_NUM$ ) will be reached and wrap around must occur. This poses a problem because the algorithm distinguishes stale heartbeats by their (logical) sequence numbers. The following lemma defines a lower bound for  $MAX\_SEQ\_NUM$ .

**Lemma 12**  $MAX\_SEQ\_NUM$ , the maximum physical heartbeat sequence number, beyond which the sequence number wraps around to zero, is  $2p$ , where  $p$  is  $\left\lfloor \frac{T_{\text{exist}} - D_{\min}}{(1-\rho)\pi} \right\rfloor$  for theorem 1 to hold.

**Proof:** Assume node  $X$  sends a heartbeat with sequence number  $n$  at real time  $t$ . This heartbeat could be received by another node  $Y$  at time  $t + T_{\text{exist}}$ . Node  $Y$  will timeout waiting for the next heartbeat from node  $X$  at time  $t + T_{\text{exist}}$ . A heartbeat sent by node  $X$  after time  $t$  could reach node  $Y$  just before it could timeout with the minimum delay of  $D_{\text{min}}$ . The largest sequence number of this heartbeat will be  $\left\lfloor \frac{T_{\text{exist}} - D_{\text{min}}}{(1-\rho)\pi} \right\rfloor = p(\text{say})$ . This means that if node  $Y$  receives a heartbeat with sequence number  $MAX\_SEQ\_NUM$ , then the next heartbeat it could get before it could timeout can have a maximum physical sequence number of  $p - 1$ . Hence, a node that gets a heartbeat with a sequence number  $a$  should consider a heartbeat with sequence number  $a - p$  as a stale heartbeat, and expect to receive a heartbeat with a sequence number between  $a + 1$  and  $a + p$  subsequently. Hence,  $MAX\_SEQ\_NUM$  is  $(p - 1) + 1 + p$  which reduces to the expression stated in the lemma. ■

# Chapter 5

## Simulation Results

Algorithm ForwardHeartbeat was simulated on hypercube networks and randomly generated networks. The network parameters,  $\Delta_{\text{send\_init}}$ ,  $\Delta_{\text{send\_min}}$  and  $\Delta_{\text{send\_max}}$ , for all simulations were fixed at 0.002, 0.008 and 0.08 seconds, respectively. Two values for the heartbeat period, denoted by  $\pi$ , were used - 4 seconds and 60 seconds. Simulations were performed on networks with the number of nodes equal to 32, 64, 128, and 256. Simulations were done using discrete event simulation techniques. The dynamic nature of the system was modeled using a Poisson distribution. Two values for the Poisson mean were used, 1 second and 200 seconds. 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 state currently held by the node plus some additional time as given by the Poisson distribution. If a failure event is not possible to occur because the number of nodes in the failed state at any instant of time cannot be greater than  $k - 1$ , then the failure event is rescheduled to a later time again using the Poisson distribution.

### 5.1 Hypercube Networks

In this section, we set the state holding times to be the theoretically derived state holding times.

Figures 6 and 7 show the theoretical worst-case latency compared to the observed

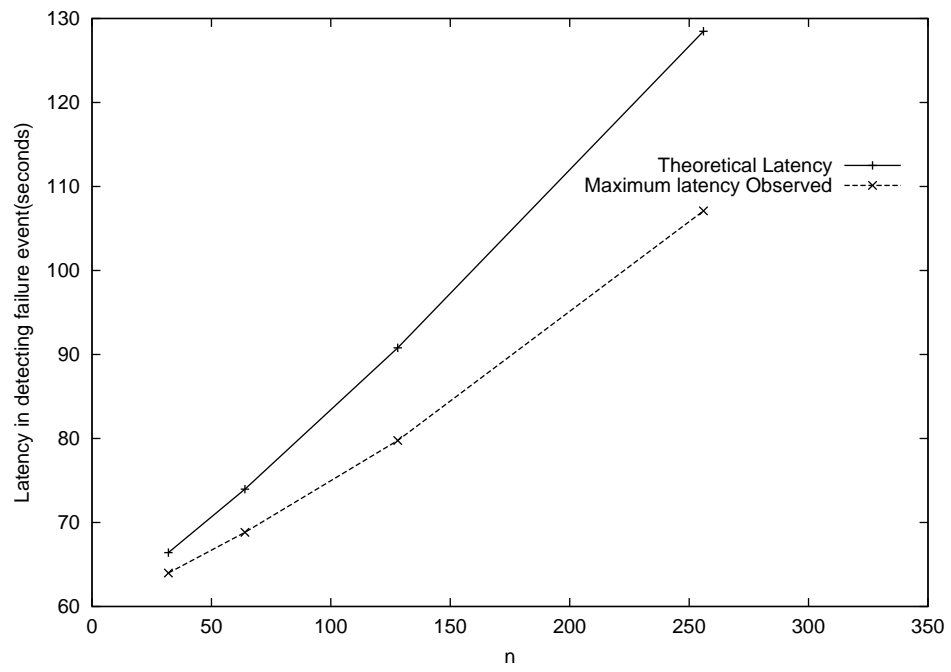


Figure 6: Theoretical and measured latency in detecting failure events for Poisson mean 200 seconds for hypercube networks

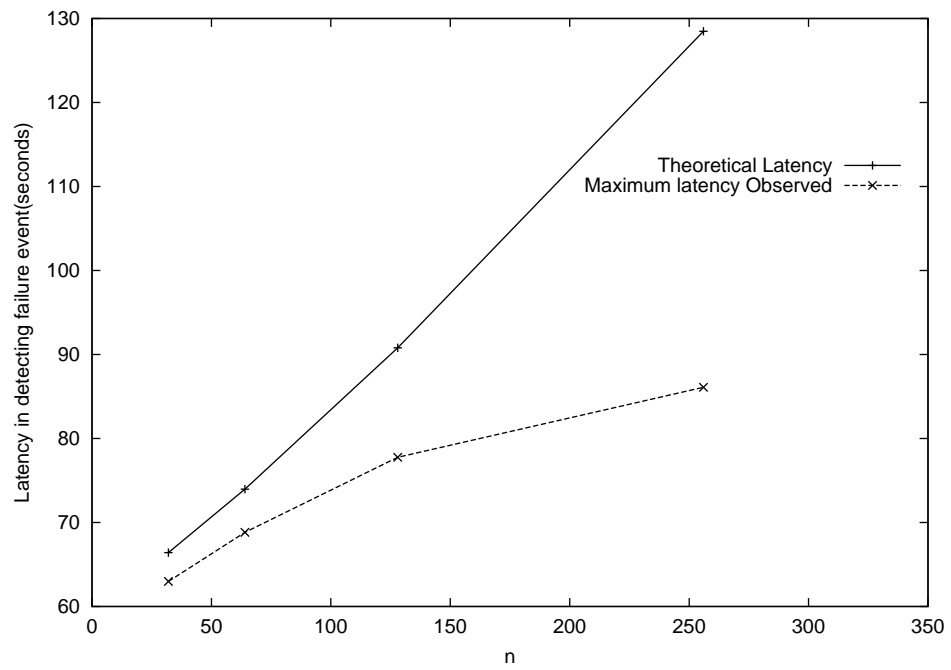


Figure 7: Theoretical and measured latency in detecting failure events for Poisson mean 1 second for hypercube networks

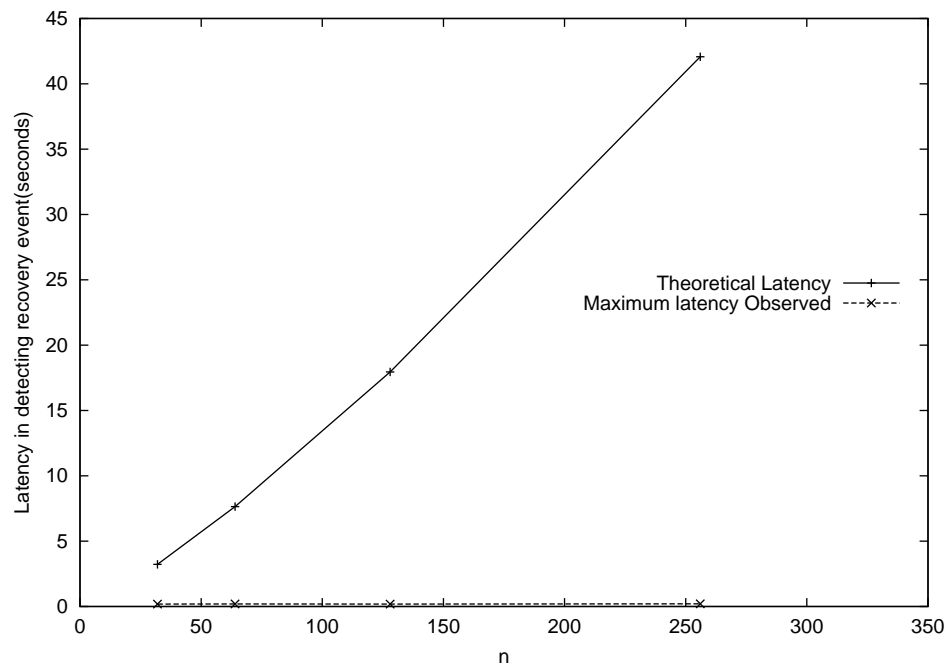


Figure 8: **Theoretical and measured latency in detecting recovery events for Poisson mean 200 seconds for hypercube networks**

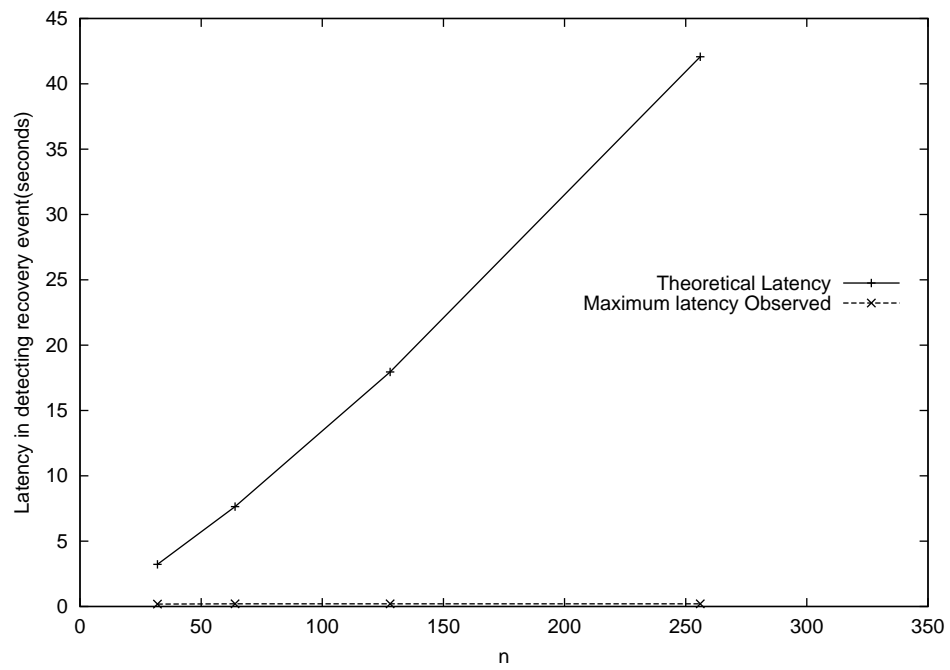


Figure 9: Theoretical and measured latency in detecting recovery events for Poisson mean 1 second for hypercube networks

worst-case latency for failure events for Poisson mean of values 200 seconds and 1 second. Figures 8 and 9 show the theoretical worst-case latency compared to the observed worst-case latency for recovery events for Poisson mean of values 200 seconds and 1 second.

Both the theoretical latency as stated in Theorem 1 and the observed maximum latency are plotted. As expected, the latency in detecting a failure event increases with  $n$ . Since the sequence of events considered in calculating the latency theoretically is very unlikely to happen and becomes even more unlikely with larger  $n$ , the difference between the experimentally observed latency and the theoretical latency increases with  $n$ . In the case of recovery latency, the observed recovery latency is more or less the same with increasing values of  $n$ . This is because it is related to the diameter of the network, which increases quite slowly with  $n$ , while the theoretically derived latency in detecting a recovery event is directly proportional to  $n$ . Again, the worst-case events that lead to the theoretical worst case are quite unlikely to occur.

Figures 10, 11 and 12, 13 show the relationship between the observed maximum and average latencies in detecting failure and recovery events, respectively.

In figure 10, we see that the difference between the average latency in detecting a failure event and the maximum latency is almost the same for increasing  $n$ . For failure events, this difference is roughly  $\pi/2$ , which is simply the difference between a node failing at the worst-case time (immediately after sending a heartbeat) and the average-case time (halfway between sending a heartbeat and the next heartbeat initiation time). In figure 11, we observe that in a highly dynamic case(Poisson mean = 1 second), the average latency in detecting a failure event approaches the measured maximum latency for increasing values of  $n$ . This is because in a more dynamic case(as against Poisson mean 200 seconds(Figure 10)), heartbeats undergo greater delays during propagation.

For recovery events, the average latency is only about half of the worst case.

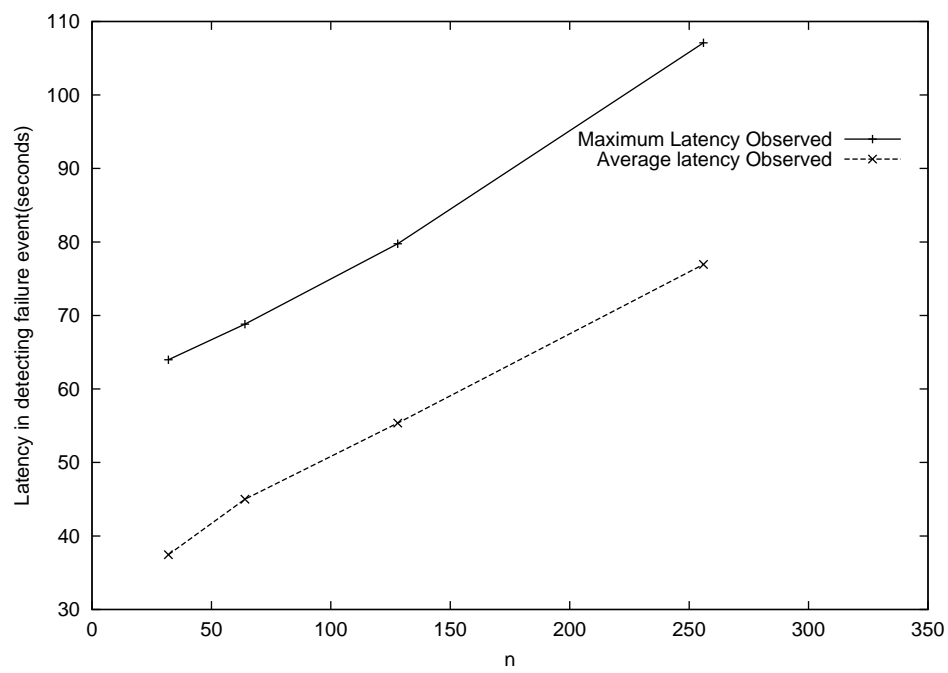


Figure 10: Measured worst case and average latency in detecting failure events for Poisson mean 200 seconds for hypercube networks

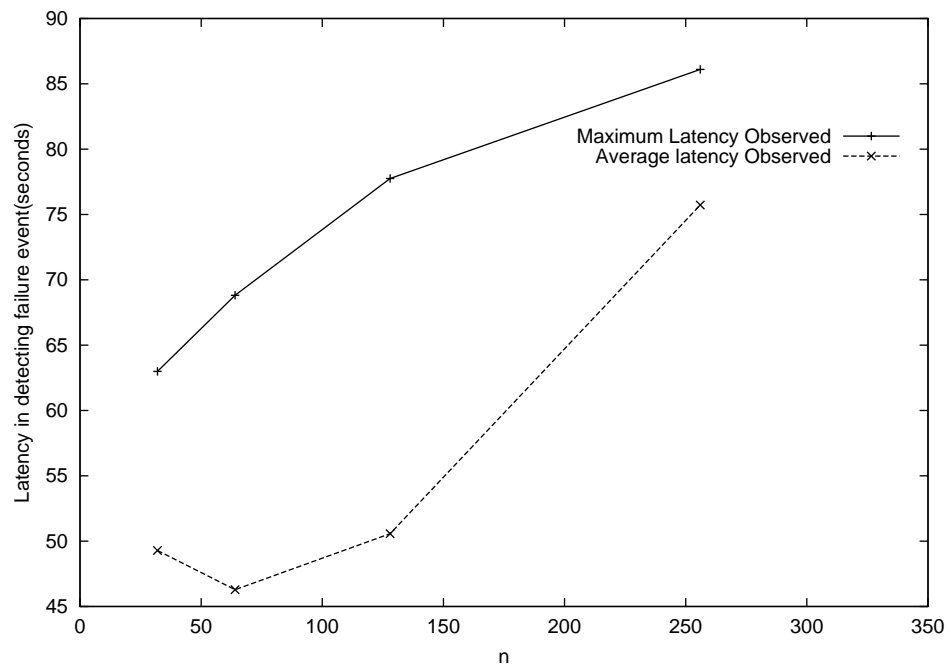


Figure 11: Measured worst case and average latency in detecting failure events for Poisson mean 1 second for hypercube networks

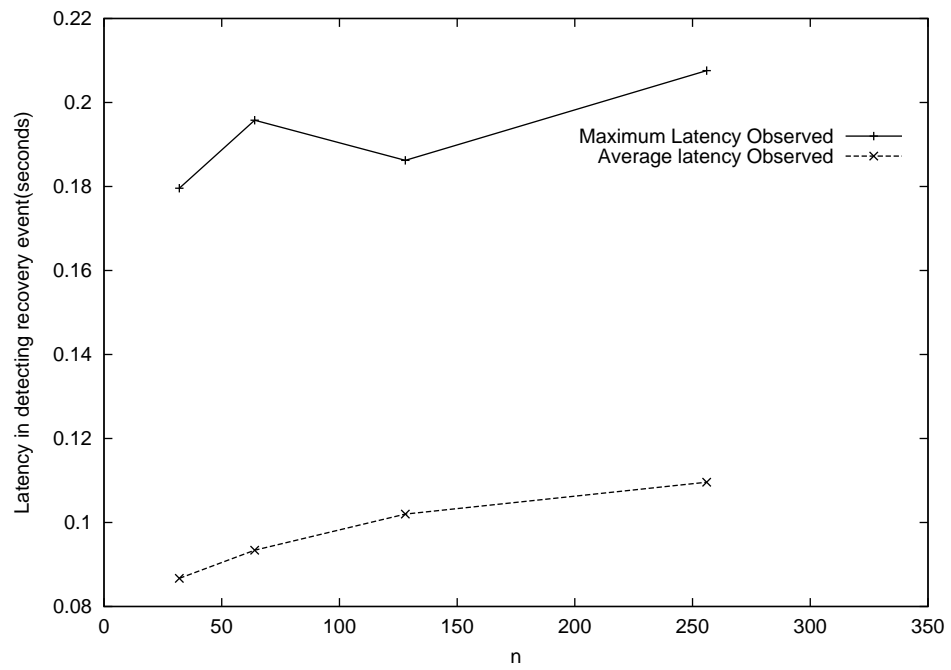


Figure 12: Measured worst case and average latency in detecting recovery events for Poisson mean 200 seconds for hypercube networks

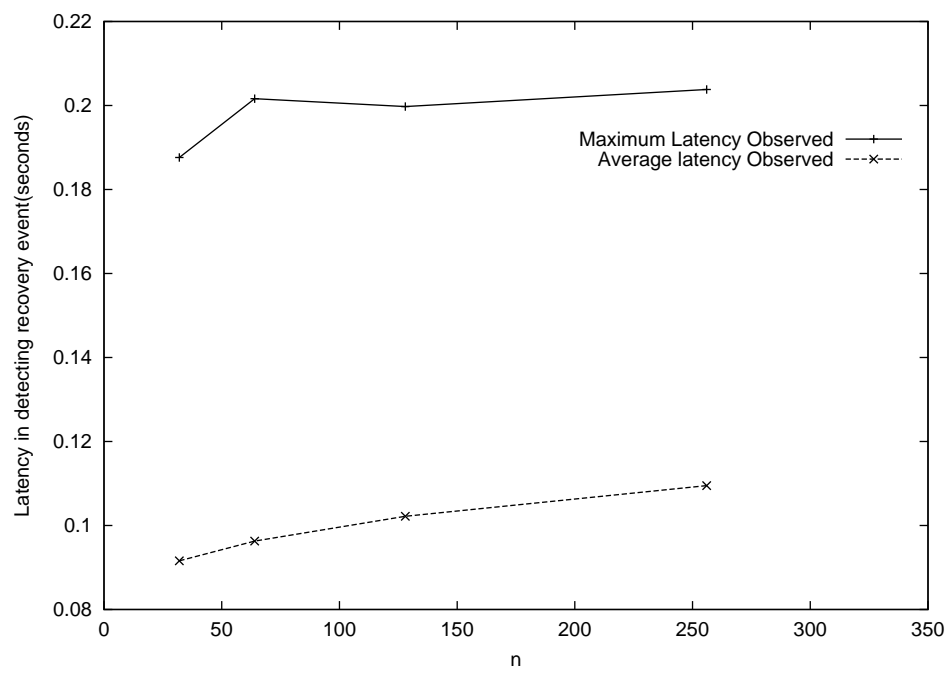


Figure 13: Measured worst case and average latency in detecting recovery events for Poisson mean 1 second for hypercube networks

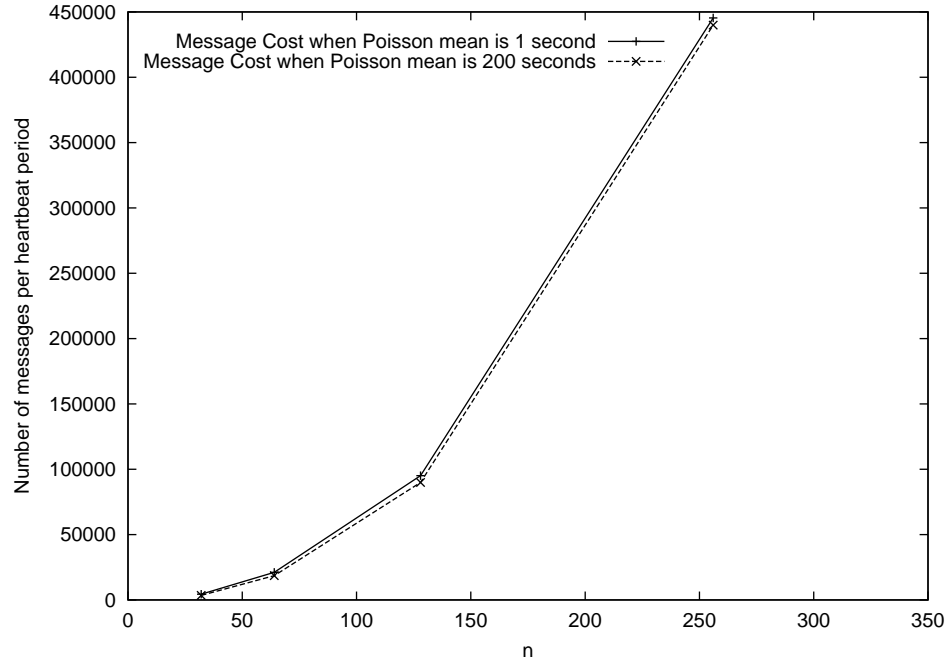


Figure 14: **Message cost of Algorithm ForwardHeartbeat for hypercube networks**

For the dominant part of the overall latency (failure detection) for large values of  $n$ , average latency is about one heartbeat period and worst-case latency is less than 2 heartbeat periods. Recall that existing algorithms have latencies of about  $n - 1$  testing rounds (equivalent to heartbeat periods) when faced with dynamic failures and repairs.

Figure 14 shows the variation of message cost with  $n$ , where the cost is stated as the number of messages per heartbeat period. As expected, the message cost increases at a rate of approximately  $n^2 \log_2 n$ . Each link in the hypercube carries about one message per working node per heartbeat period in the algorithm. Although this does not scale well to very large values of  $n$ , the cost is moderate for the values of  $n$  simulated. It should also be pointed out that all messages in the algorithm are of length proportional to  $\log_2 n$ , which is quite short for all reasonable values of  $n$ .

## 5.2 Random Networks

Besides simulating the algorithm on specific network topologies such as hypercubes, algorithm ForwardHeartbeat was also simulated on randomly generated networks. Graphs are randomly generated for a given value of  $n$ , the total number of nodes in the network, and  $k$ , the connectivity of the network. Since every node must have at least  $k$  neighbors, we first ensure that this is achieved by randomly introducing links such that every node has  $k$  neighbors. The connectivity of the network is found out by running the Ford-Fulkerson algorithm [23] on the network. If the connectivity of the network is less than  $k$ , then  $n$  links are randomly introduced into the network. The choice of  $n$  here is totally arbitrary. This process is continued until the resulting connectivity of the network is at least  $k$ . None of the graphs generated this way had a connectivity greater than the one desired.

In the simulations carried out on hypercube networks, it was found that having the working state holding time equal to the theoretically derived one did not matter because the occurrence of events was controlled by the failed state holding time and the restriction that the number of nodes in the failed state at any instant of time is limited to  $k - 1$ . Since we are trying to model a real system here, the working state holding time for the simulations on random networks was kept equal to the failed state holding time.

Simulations were carried out for two different values of the Poisson mean (1 second and 200 seconds), two different values of the heartbeat period  $\pi$  (4 seconds and 60 seconds) and two different values of the connectivity  $k$  (3 and  $\log_2 n$ ). For every value of  $(n, k)$ , five different networks were generated. Figure 15 shows the average latencies in detecting failure events for the different networks generated when  $k = 3$ , Poisson mean = 200 seconds and  $\pi = 60$  seconds. We find that the average latency in detecting failure events lies within the confidence interval of one of the networks. The

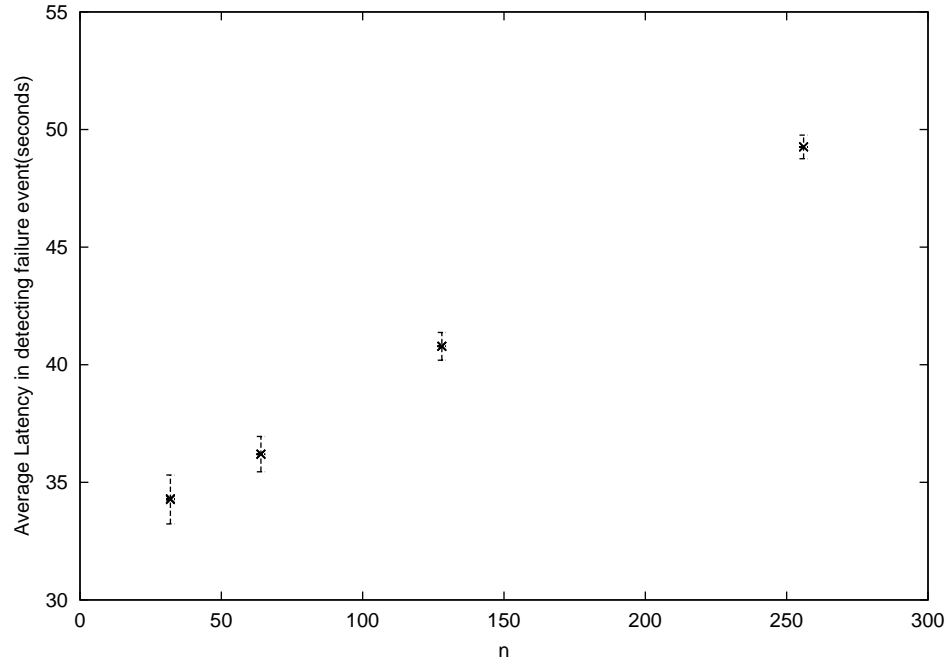


Figure 15: **Average latency in detecting failure events for some random networks**

confidence interval used here is arrived at using the Student  $t$  distribution [13] for a probability of 95%. In other words, the average latency in detecting failure events are found to be very close to each other for the different networks. We find a similar case happening in the case of average latencies in detecting recovery events(Figure 16). In the results presented in figure 16,  $k$  was assumed to be  $\log_2 n$ , Poisson mean = 200 seconds and  $\pi = 60$  seconds.

The same networks yield similar results when the other values of Poisson mean and  $\pi$  were used. Other networks generated for different values of  $k$  show similar characteristics. Hence, in the following discussions, we take the mean of the desired quantity yielded by the random networks generated for a given  $n$  and  $k$  and consider it to be representative of all other randomly generated networks.

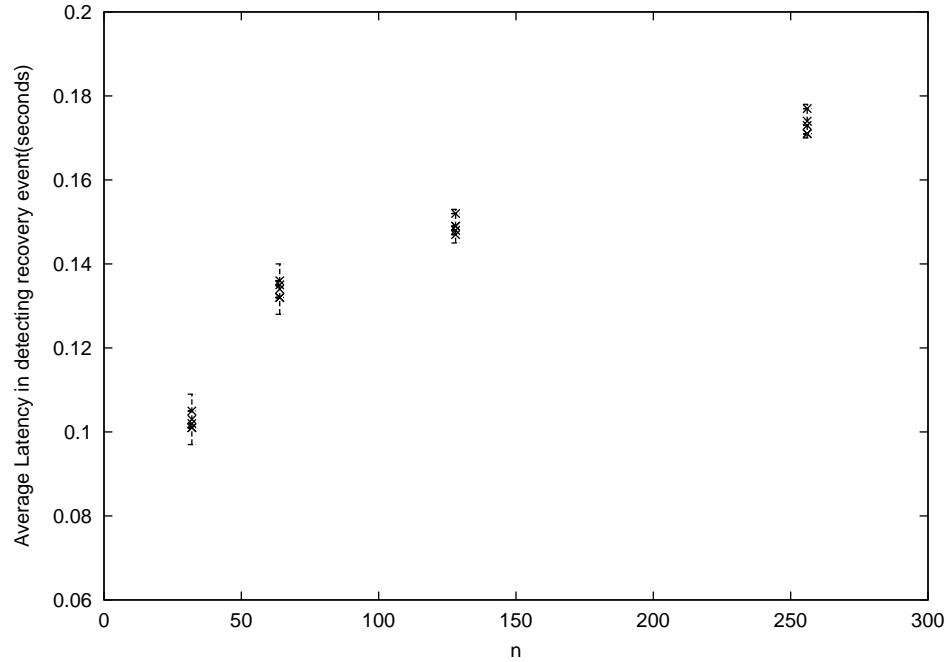


Figure 16: **Average latency in detecting recovery events for some random networks**

### 5.2.1 Effect of the Poisson Mean

We consider randomly generated networks for various  $n$  with  $k = 3$ . The heartbeat period,  $\pi$ , is kept constant at 60 seconds. Figure 17 shows the variations in the latency involved in detecting failure events with increasing  $n$ . The Poisson mean affects the dynamic nature of the system as mentioned earlier. The experiment involved two values for the Poisson mean - 1 second and 200 seconds. The theoretically derived expression for the latency is independent of the Poisson mean. Hence only a single curve for both values of the Poisson mean is shown in the figure.

We find that for most values of  $n$ , a more dynamic fault model yields a noticeably larger latency in detecting a failure event. As seen in the theoretical analysis of the algorithm, the latency involved in detecting a failure event is variable due to the difference in the actual propagation time of the last heartbeat and the delay kept

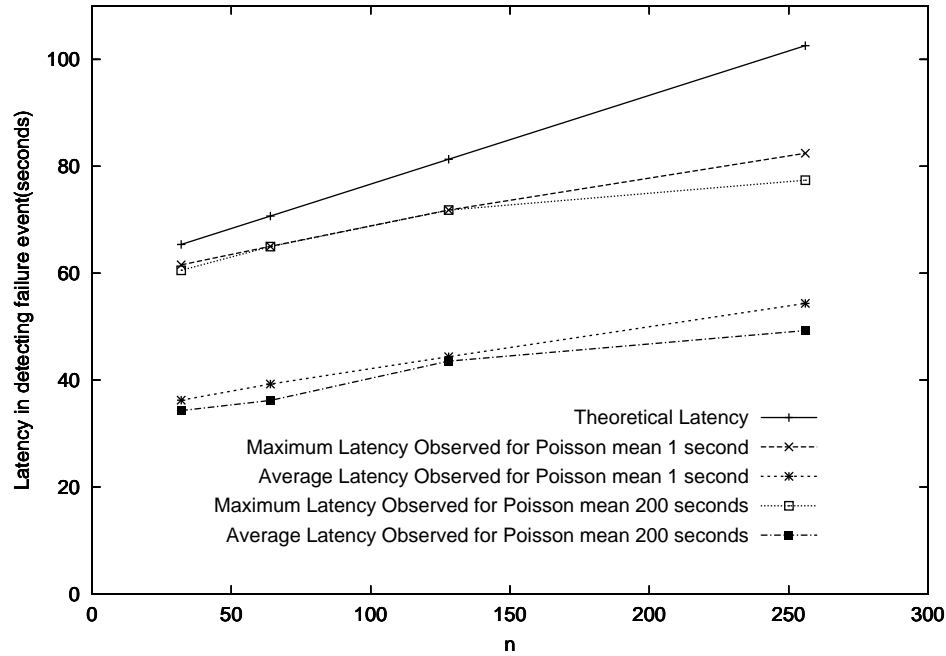


Figure 17: Latency in detecting failure events vs  $n$  for different values of the Poisson mean for random networks

track of by the intermediate nodes. Hence, the more links the last heartbeat that is sent by a node before it could fail traverses, the more varied the latency will be. In a more dynamic fault model, it can be expected that a heartbeat takes a longer time to propagate than in networks that have a relatively less dynamic fault model. Hence, the more dynamic fault model (Poisson mean = 1 second) yields a larger latency.

We also observe that the maximum observed latencies do not seem to catch up with the theoretical latency for increasing  $n$ . This is because the theoretically derived maximum latency assumes a particular localized sequence of events which is very unlikely to occur, which is all the more unlikely when  $n$  gets larger. The average latency in detecting a failure event is again higher in a more dynamic fault model due to the reasons given above.

Figure 18 shows the variations in the latency involved in detecting recovery events. The theoretical latency is again independent of the dynamic nature of the system (Poisson

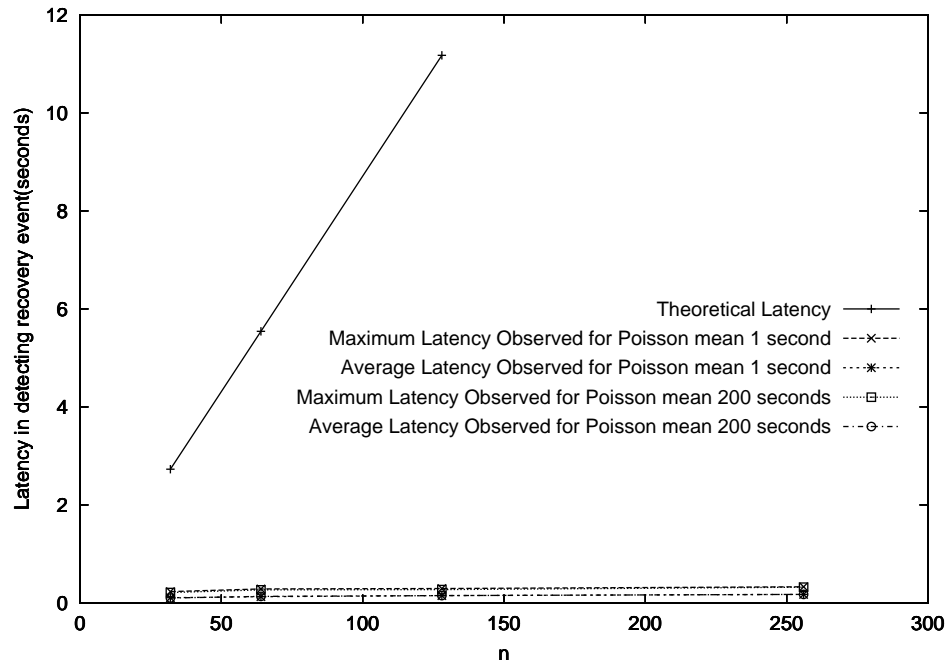


Figure 18: **Latency in detecting recovery events vs  $n$  for different values of the Poisson mean for random networks**

mean). Since we artificially construct a possible sequence of events that gives rise to a maximum latency in detecting a recovery event, the theoretical latency appears to linearly increase with  $n$ , while the maximum observed latency in detecting recovery events shows a relatively marginal increase with increasing  $n$ . This is due to the fact that, in practice, a heartbeat is flooded throughout the network through multiple paths. The consequent delay does not change appreciably with increasing  $n$ . The latency in detecting a recovery event is the maximum time the heartbeat sent after the recovery event takes to reach a node that is in the working state since the event occurred. The Poisson mean seems to hardly affect the measured worst case and average latencies in detecting recovery events. However dynamic the system may be, the sequence of events considered during the theoretical analysis of the latency involved in detecting recovery events is very unlikely to happen. The theoretical maximum latency for  $n = 256$  is not shown in the graph because including it would scale down

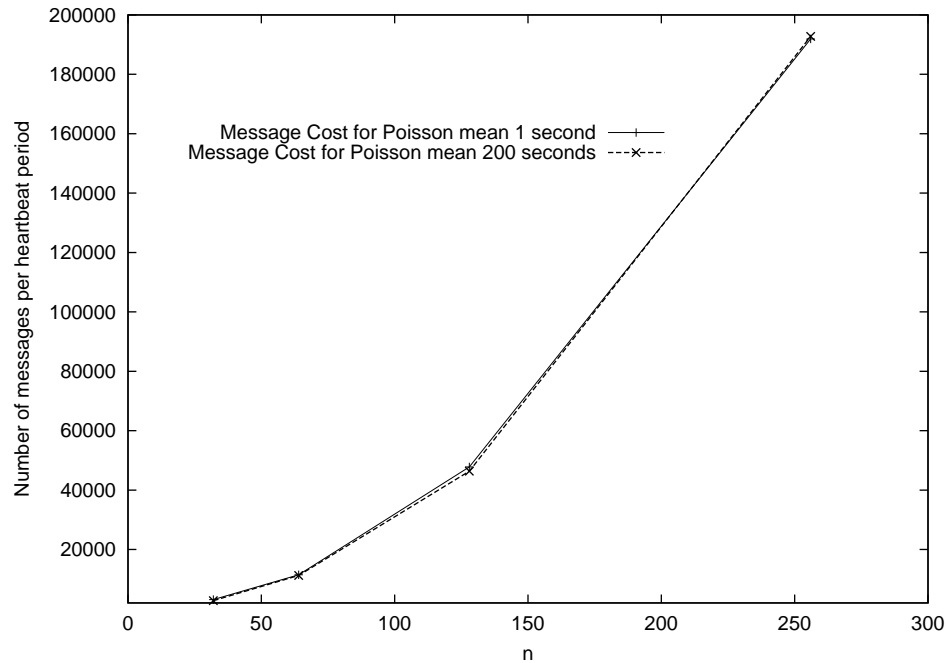


Figure 19: **Message cost vs  $n$  for different values of the Poisson mean for random networks**

the y-axis such that the measured maximum and average latency curves get blurred out.

Figure 19 shows the effect of the Poisson mean on the message cost. The message cost is measured as the number of messages sent in one heartbeat send period, that is,  $\pi$ . When a node recovers, neighboring nodes that are working forward all buffered heartbeat messages to it. Hence, there will be a higher message cost when the system is more dynamic. However, the message costs involved while performing the routine issuing and relaying of heartbeat messages is so high that the dynamic nature seems to hardly have any effect on the system.

Summarizing, a more dynamic fault model leads to increased latencies in detecting failure events and a very small increase in message costs. The dynamic nature of the system seems to have negligible impact on the latency in detecting recovery events.

Figure 20 shows if the Poisson mean alone determined the dynamic nature of the

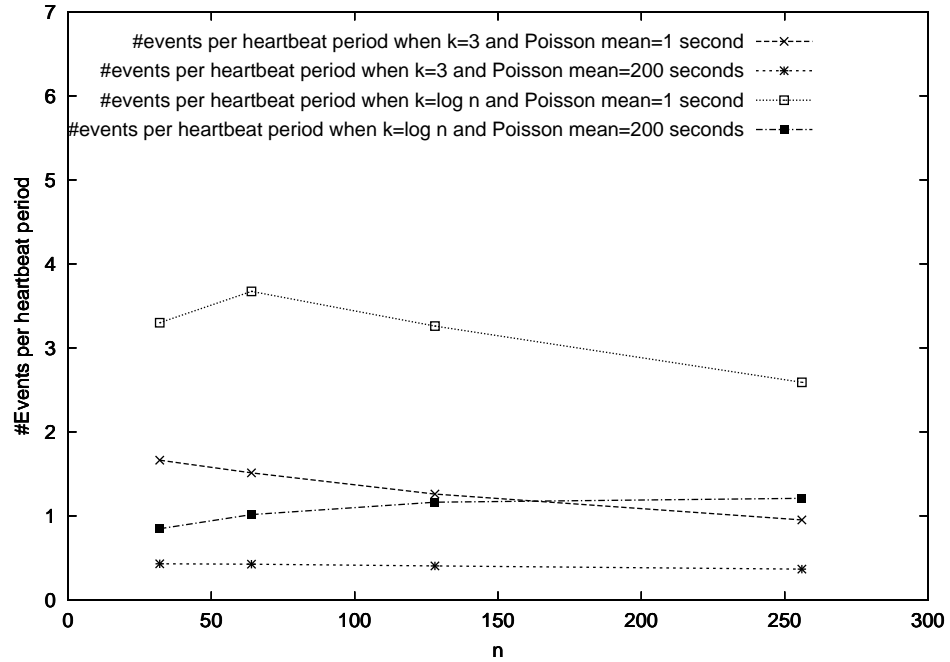


Figure 20: **Number of events per heartbeat period vs  $n$  for different values of  $k$  and Poisson mean for random networks**

system. The other limiting factor is the restriction that the number of nodes in the failed state at any time cannot exceed  $k - 1$ . Hence, for a given network, lowering the Poisson mean below some particular value will hardly have any effect on the dynamic nature of the system. Figure 20 shows the average number of events(failure and recovery) that occurred for the different values of connectivity and Poisson mean considered in the simulations. Barring the case when  $k = \log_2 n$  and Poisson mean = 200 seconds, we observe that the average number of events per heartbeat period decreases with increasing  $n$ . This is because the state holding times increase with increasing  $n$ , thus cutting down the number of events that can occur in a given interval of time. For the same value of the Poisson mean and  $n$ , more number of events per heartbeat period takes place when  $k$  is larger as the number of nodes that can be in the failed state at any given time is now greater. When  $k = \log_2 n$  and Poisson mean = 200 seconds, the restriction that the number of nodes that can be

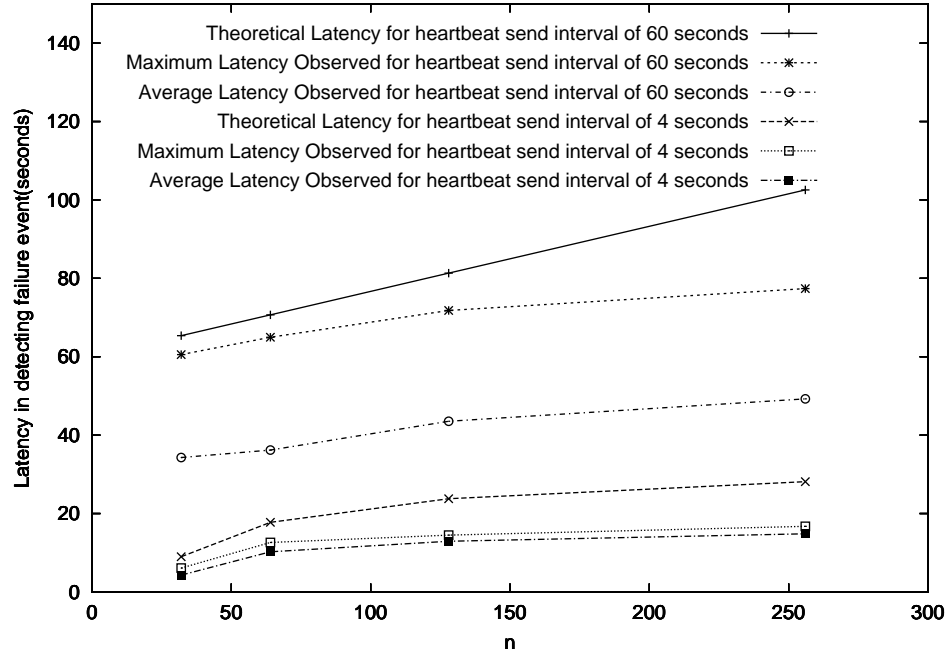


Figure 21: Latency in detecting failure events vs  $n$  for different values of  $\pi$  for random networks

in the failed state at any given time cannot exceed  $k - 1$  is not a limiting factor because the Poisson mean is so large; hence, more number of events per heartbeat period occurs when more number of nodes exist in the network. Also notice that  $k$  increases with  $n$  too. In this case ( $k = \log_2 n$  and Poisson mean=200 seconds), the number of nodes in the failed state at any given time is less than  $k - 1$ . The same reason explains the increase in the number of events per heartbeat period when  $n$  is increased from 32 to 64 with  $k = \log_2 n$  and the Poisson mean = 1 second. Hence, the restriction that the number of nodes that can be in the failed state at any given instant of time cannot be greater than  $k - 1$  does not limit the dynamic nature of the system only when  $k = \log_2 n$  and Poisson mean = 200 seconds.

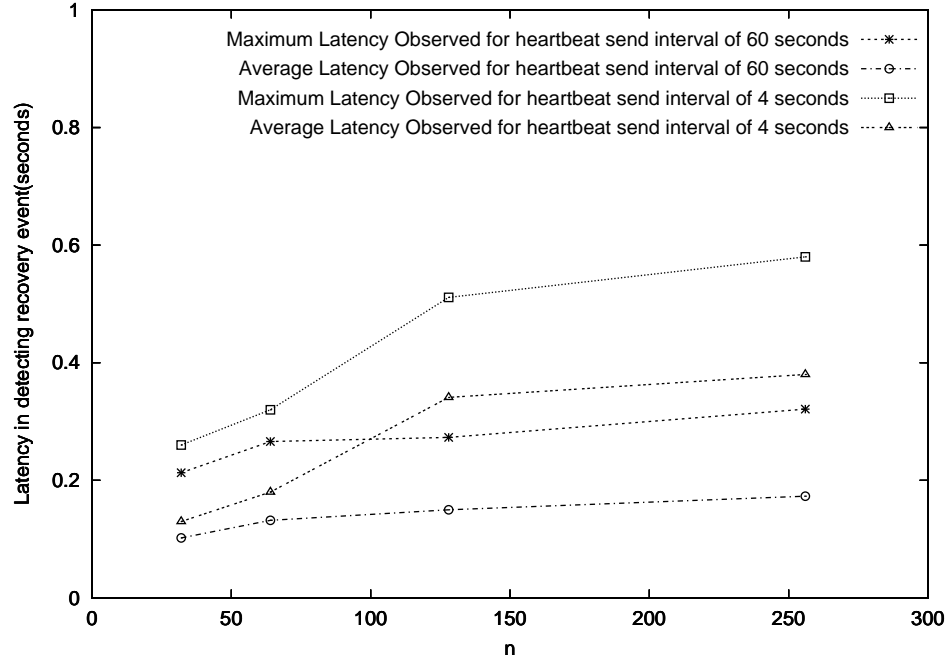


Figure 22: Latency in detecting recovery events vs  $n$  for different values of  $\pi$  for random networks

### 5.2.2 Effect of Heartbeat period ( $\pi$ )

Figure 21 shows the variation of the latency involved in detecting failure events with  $n$  for the heartbeat period being 4 seconds and 60 seconds, with the Poisson mean kept fixed at 200 seconds and the connectivity kept fixed at 3. The general characteristics of the graph for a given  $\pi$  has already been explained in the above section. When  $\pi = 4$ , a working node sends out heartbeats more frequently thereby helping other nodes detect events quickly (the timeout period is reduced). Maximum latency arises when a node fails soon after it sends a heartbeat. Average latency broadly considers the case when a node fails half way during the heartbeat period. Hence, the difference between the measured maximum latency in detecting a failure event and the average latency in detecting fault events should differ by approximately  $\pi/2$ , and can be noticed in the figure.

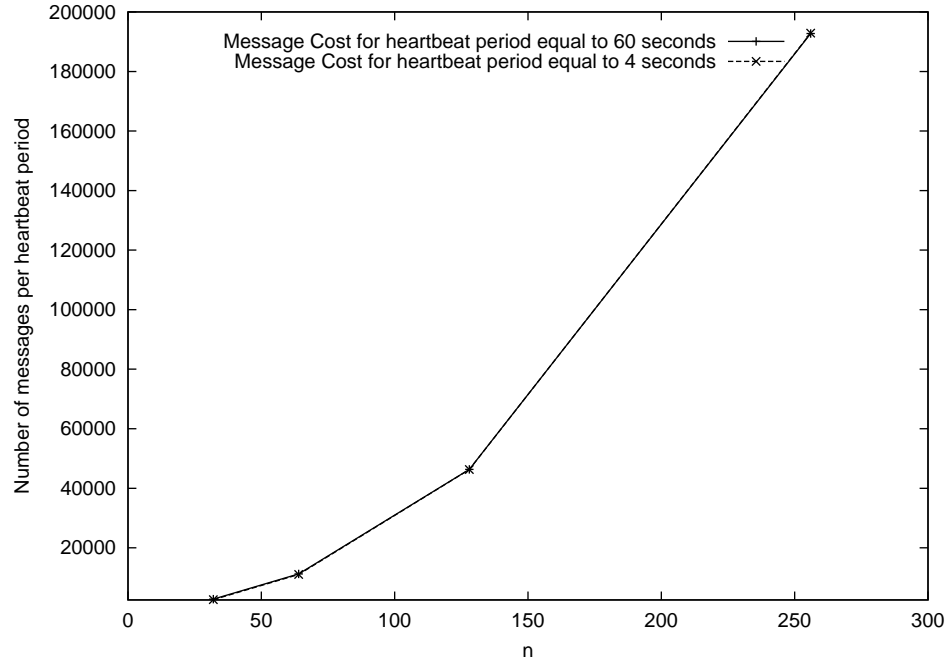


Figure 23: **Message cost vs  $n$  for different values of  $\pi$  for random networks**

Figure 22 shows the effect of varying the heartbeat period on the latency involved in detecting recovery events. Since the theoretical value of the recovery latency is independent of  $\pi$  and has already been shown in figure 18, it is not shown in figure 22. The measured maximum and average latencies in detecting recovery events depend on how much time a heartbeat takes to propagate to all nodes in the network. This is a quantity that will greatly be affected with the general amount of traffic in the network. When  $\pi = 4$  seconds, there is more traffic in the network compared to the case when  $\pi = 60$  seconds. Hence, the latency in detecting recovery events is higher for the lower value of  $\pi$ .

Figure 23 shows the effect of the heartbeat period  $\pi$  on the message cost. Since the message cost is the number of messages transmitted per heartbeat period, there is hardly any difference in the message cost. If the message cost is defined as the number of messages issued during a fixed interval of time, then the message cost

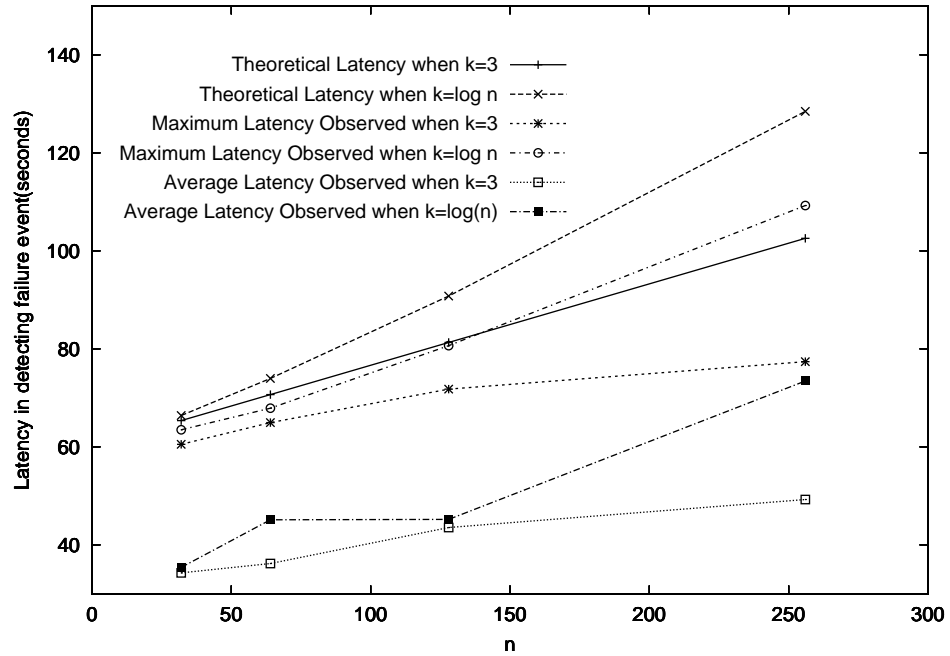


Figure 24: Latency in detecting failure events vs  $n$  for different values of connectivity for random networks

when  $\pi = 4$  seconds will definitely be higher than the case where  $\pi = 60$  seconds as more heartbeat messages are issued and relayed.

### 5.2.3 Effect of connectivity $k$

Simulations were carried out with the heartbeat period kept fixed at 60 seconds and the Poisson mean kept fixed at 200 seconds. Higher connectivity implies more number of paths through which heartbeat messages can propagate. This results in quicker propagation of heartbeat messages. In figure 24, we analyze the impact of varying the connectivity on the latency involved in detecting a failure event. It seems counter intuitive that the latency is larger when the connectivity is larger. To detect a fault event, a node needs to timeout. The timeout period is dependent on the theoretically derived value of  $D_{\max n}$  which increases appreciably with a small increase in  $k$ . Hence,

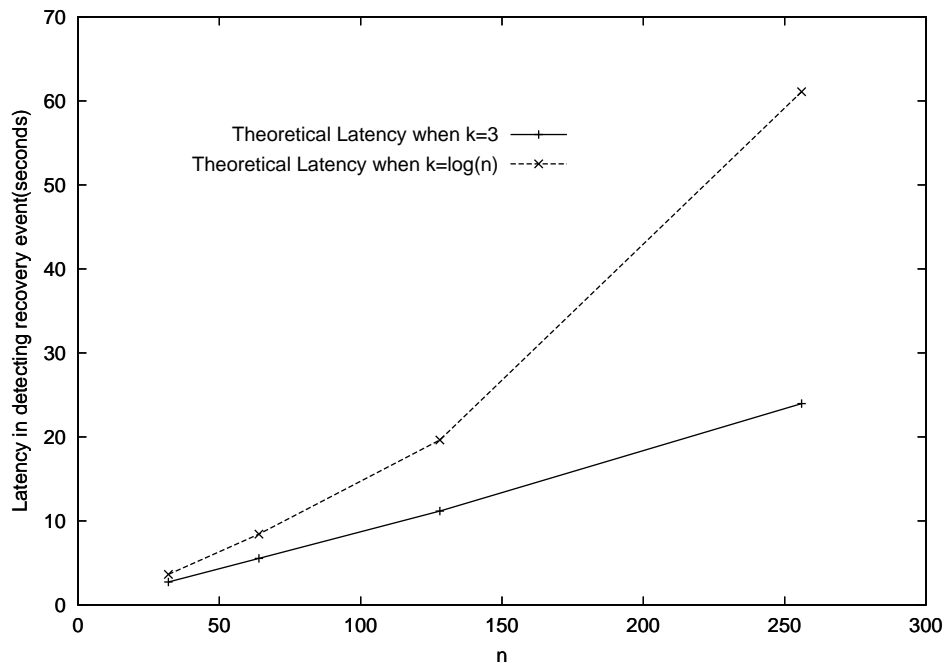


Figure 25: **Theoretical latency in detecting recovery events vs  $n$  for different values of connectivity for random networks**

this more than offsets the quicker propagation of the heartbeat. Hence, latencies in detecting failure events are higher with a larger connectivity  $k$ .

It may be noticed that the difference between the theoretical latency and the measured maximum latency for increasing  $n$  is higher when  $k$  is kept constant. This is because the theoretical value of  $D_{\max n}$ , which affects the timeout period as pointed out earlier, is proportional to  $k$ . As  $n$  increases,  $k = \log_2 n$  increases thereby approaching the theoretical latency closer than the case where  $k = 3$ .

Figure 25 shows the variation of the theoretically derived maximum latency in detecting recovery events for the two different values of the connectivity for increasing  $n$ . Since the latency has a linear dependence on  $k$ , the theoretically derived latency is higher for a larger  $k$ .

Figure 26 shows the variation of the measured maximum and measured latencies

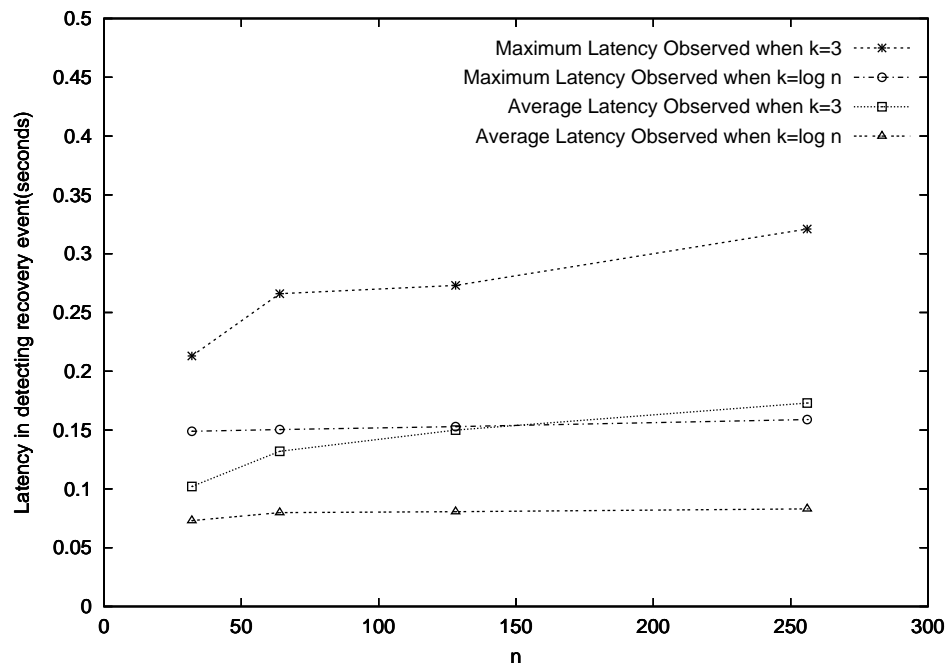


Figure 26: Measured latency in detecting recovery events vs  $n$  for different values of connectivity for random networks

in detecting recovery events for different connectivities. For the different values of  $k$  when all other parameters are kept constant, the latency is higher when  $k$  is lower, contradicting the theoretically expected result. This is because with a higher  $k$  there exists more links thereby aiding quicker propagation of heartbeats. The theoretically derived latency considers a very specific sequence of events that yield latencies that increase with  $k$ . In practice, the occurrence of this particular sequence of events is very unlikely. It is also observed that both the measured and maximum latencies in detecting recovery events increase more for increasing  $n$  when  $k$  is constant as compared to the case where  $k$  increases with  $n$ . This is because, with  $k$  constant and increasing  $n$ , a heartbeat needs to propagate through a greater number of nodes. When  $k$  becomes larger with  $n$ , there are more links thereby facilitating quicker propagation of heartbeats and the lengths of paths are not increased significantly as compared to the case when  $k$  is kept constant.

Figure 27 shows the variation of the message cost with  $n$  for the two different values of connectivity considered. Message cost is defined as the number of messages sent per heartbeat period( $\pi$ ). As expected, with larger values of connectivity, the message cost increases substantially.

Figure 28 shows the variation of the message cost(number of messages per heartbeat period) per link vs  $n$  for the two different values of the connectivity when the Poisson mean is equal to 200 seconds and  $\pi = 60$  seconds. As expected, the message cost increases with increasing  $n$ . A higher value of  $k$  indicates that a node now has more number of links connected to it. Hence, when a node initiates a heartbeat, more messages are sent by the node. Also, when a node recovers, neighboring nodes detect the recovery event and send buffered heartbeat messages to the node that recovered. With a higher value of  $k$ , there will be a greater number of neighboring nodes that will be sending buffered heartbeat messages when a node recovers. In addition, the node that recovered in turn forwards these heartbeats on the remaining links. Hence,

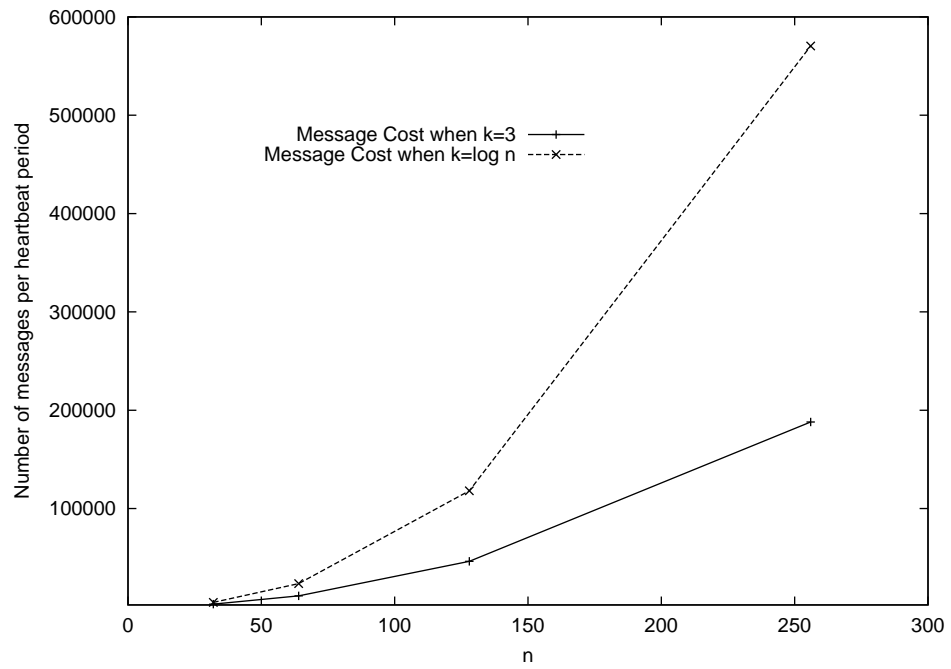


Figure 27: **Message cost vs  $n$  for different values of connectivity for random networks**

the message cost increases when  $k$  is higher.

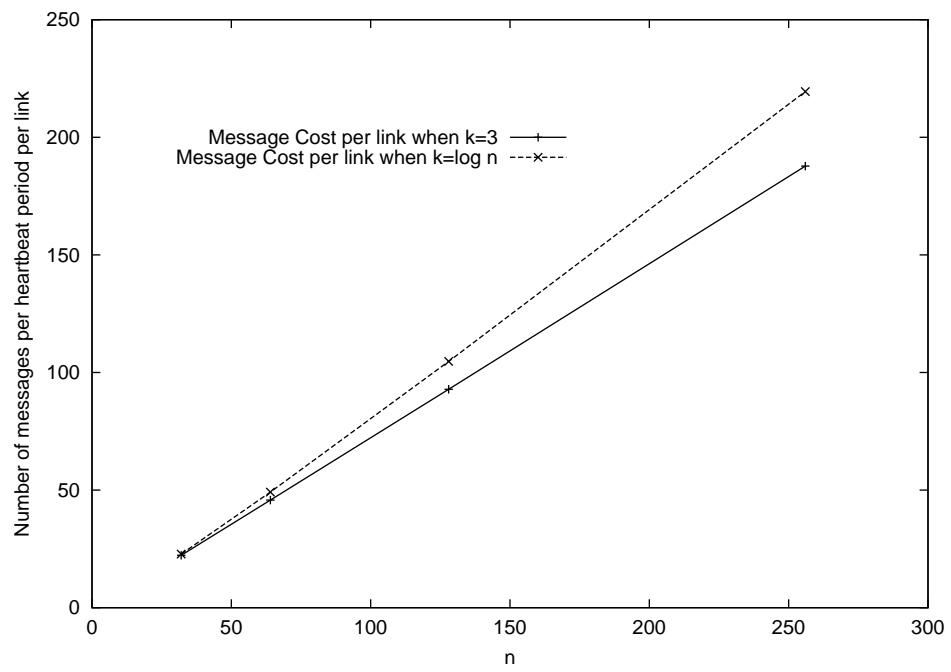


Figure 28: Message cost per link vs  $n$  for different values of connectivity for random networks

## Chapter 6

### Conclusion

The design of a distributed diagnosis algorithm is non-trivial due to the complications of dynamic failures and repairs that can cause the set of paths connecting any two given nodes to change continuously with time. The diagnostic latency and state holding time of algorithm ForwardHeartbeat were both shown to be approximately one heartbeat period (testing round). Hence, in terms of latency, Algorithm ForwardHeartbeat performs significantly better than existing algorithms which have been shown to have latencies and state holding times of about  $n - 1$  testing rounds.

Existing algorithms, particularly for not-completely-connected networks, have been designed to minimize the number of tests performed and as a result have a very difficult time handling dynamic failures and repairs. Algorithm ForwardHeartbeat is designed to propagate status information as quickly and through as many redundant paths as possible to allow it to effectively handle dynamic situations. The message cost of ForwardHeartbeat is roughly  $n$  messages per link per testing round (heartbeat period). For very large values of  $n$ , this cost can be quite high but for moderate values of  $n$  and typical testing round durations, the overhead is quite modest. Nevertheless, an interesting open question is whether algorithms with lower message cost can be developed that either have the same latency and state holding time as ForwardHeartbeat or allow trade-offs between message cost and the latency and the state holding time.

An example of such an algorithm would be algorithm "EventDisseminate", where

nodes perform independent diagnosis of neighboring nodes and when an event is detected, that information is flooded throughout the network. Thus, a node performs its own independent diagnosis of its neighboring nodes and depends on other nodes to perform diagnosis of non-neighboring nodes. Heartbeat messages are not propagated. Hence, the message cost in this case is significantly improved, though the impact on the latency and the state holding time is yet to be investigated.

The simulation results show some interesting results. It has been found that, while the theoretical latency in detecting recovery events increases linearly with  $n$ , the number of nodes in the network, in practice the effect of increasing  $n$  does not have a big impact on the observed latency in detecting recovery events. The theoretical results show that for networks with greater connectivity the latency in detecting recovery events is larger, while the contrary happens in practice. The particular sequence of events considered in the theoretical analysis of Algorithm ForwardHeartbeat could still occur thereby verifying the theoretical results but the probability of the occurrence of a particular sequence of events is very unlikely.

The latency in detecting failure events conform to theoretical predictions for different values of the connectivity because the timeout period is predetermined theoretically and this quantity is proportional to the connectivity of the network.

This work assumed that the number of nodes in the failed state at any given instant of time is always less than the connectivity of the network. Future work could focus on a system model where there is no restriction on the number of nodes that can be in the failed state at any given instant of time even though the network remains connected at all times. A further step would be to come up with new correctness definitions when the network can get disconnected due to no restrictions being placed on the number of nodes that can be in the failed state at any given time.

# Bibliography

- [1] A. Bagchi and S. L. Hakimi, "An Optimal Algorithm for Distributed System Level Diagnosis," *Digest of the 21st Int. Symp. Fault Tolerant Computing*, pp. 214–221, 1991.
- [2] M. Barborak, M. Malek, and A.T. Dahbura, "The Consensus Problem in Fault-Tolerant Computing," *ACM Computing Surveys*, Vol. 25, pp. 171–220, June 1993.
- [3] R. Bianchini, K. Goodwin, D. Nydick, "Practical Application and Implementation of Distributed System-Level Diagnosis Theory," *IEEE Conference*, pp 1990.
- [4] R. Bianchini, R. Buskens, "An Adaptive Distributed System-Level Diagnosis Algorithm and its Implementation," *Fault Tolerance Computing Symposium*, pp. 222–229, 1991.
- [5] R. Bianchini and R. Buskens, "Implementation of On-Line Distributed System-Level Diagnosis Theory," *IEEE Transactions on Computers*, Vol. 41, pp. 616–626, May 1992.
- [6] R. Bianchini, M. Stahl and R. Buskens, "The Adapt2 On-Line Diagnosis Algorithm for General Topology Networks," *Globecom*, pp. 610–614, December 1992.
- [7] D. Blough and H. Brown, "The Broadcast Comparison Model for On-Line Fault Diagnosis in Multicomputer Systems: Theory and Implementation," *IEEE Transactions on Computers*, Vol. 48, pp. 470–493, May 1999.
- [8] D. Blough and A. Subbiah, "Distributed Diagnosis in Dynamic Fault Environments," submitted to the *International Conference on Distributed Computing Systems*, 2002.
- [9] R. Buskens, R. Bianchini, "Distributed On-Line Diagnosis in the presence of Arbitrary Faults," *Twenty third International Symposium on Fault Tolerant Computing*, pp 470–479, June 1993.
- [10] E.P. Duarte, Jr. and T. Nanya, "A Hierarchical Adaptive Distributed System-Level Diagnosis Algorithm," *IEEE Transactions on Computers*, Vol. 47, pp. 34–45, January 1998.

- [11] E. P. Duarte Jr., A. Brawerman, and L. C. P. Albin, "An Algorithm for Distributed Hierarchical Diagnosis of Dynamic Fault and Repair Events," *Proc. of the 7th International Conference on Parallel and Distributed Systems*, pp. 299–306, 2000.
- [12] M. Hiltunen, "Membership and System Diagnosis," *Proceedings of the 14th Symposium on Reliable Distributed Systems*, pp. 208–217, 1995.
- [13] P. G. Hoel, S. C. Port, and C. J. Stone, "Introduction to Statistical Theory," *Houghton Mifflin Company*, 1971.
- [14] S. Hosseini, J. Kuhl, and S. Reddy, "A Diagnosis Algorithm for Distributed Computing Systems with Dynamic Failure and Repair," *IEEE Transactions on Computers*, Vol. C-33, pp. 223–233, March 1984.
- [15] W. Hurwood, "Ongoing Fault Diagnosis," *Proceedings of the 15th Symposium on Reliable Distributed Systems*, pp. 108–117, 1996.
- [16] J. G. Kuhl and S. M. Reddy, "Distributed Fault Tolerance for large Multiprocessor Systems," *Proc. 7<sup>th</sup> Annual Symp. COmputer Architecture*, pp. 23–30, May 1980.
- [17] J. G. Kuhl and S. M. Reddy, "Fault Tolerance in fully distributed Systems," *Proc. 11<sup>th</sup> Intl. Symp. Fault Tolerant Computing*, pp. 100–105, June 1981.
- [18] P. Maestrini and P. Santi, "Self Diagnosis of Processor Arrays using a Comparison Model," *Proceedings of the 14th Symposium on Reliable Distributed Systems*, pp. 218–228, 1995.
- [19] A. Pelc, "Undirected Graph Models for System-Level Fault Diagnosis," *IEEE Transactions on Computers*, Vol. 40, pp. 1271–1276, Nov. 1991.
- [20] S. Rangarajan and D. Fussell, "Diagnosing Arbitrarily Connected Parallel Computers with High Probability," *IEEE Transactions on Computers*, Vol. 41, pp. 606–615, May 1992.
- [21] S. Rangarajan, A.T. Dahbura, and E. Ziegler, "A Distributed System-Level Diagnosis Algorithm for Arbitrary Network Topologies," *IEEE Transactions on Computers*, Vol. 44, pp. 312–334, February 1995.
- [22] A. Sengupta and A.T. Dahbura, "On Self-Diagnosable Multiprocessor Systems: Diagnosis by the Comparison Approach," *IEEE Transactions on Computers*, Vol. 41, pp. 1386–1396, Nov. 1992.

- [23] Shimon Even, "Graph Algorithms," *Computer Science Press*, 1979.
- [24] M. Stahl, R. Buskens and R. Bianchini, "On-Line Diagnosis in General Topology Networks," *Proc. Workshop Fault Tolerant Parallel and Distributed Systems*, pp. 114–121, 1992.
- [25] M. Stahl, R. Buskens, R. Bianchini, "Simulation of the Adapt On-Line Diagnosis Algorithm for General Topology Networks," *Symposium on Reliable Distributed Systems*, pp 180–187, 1992.