

**Leader Election Algorithms for Three Dimensions  
Torus Networks with the Presence of One or Two  
Links Failure**

خوارزميات انتخاب القائد

للشبكات المحدبة ثلاثية الأبعاد في حالة وجود عطل في وصلة أو وصلتين

**Presented by:**

**Abdelkhaleq Al Hammouri**

**Supervised by**

**Prof. Ala'a Al Hamami**

**Dissertation Submitted in partial fulfillment of the  
requirements for the degree of Doctor of Philosophy  
in Computer Science**

**College of Computer Sciences and Informatics**

**Amman Arab University**

**September 2012**

## Authorization of Dissemination

I, Abdel Kheleq a. Hammouri , authorize Amman Arab University to reproduce this dissertation in whole or part for purpose of research


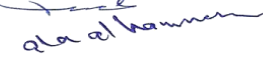


Name : Abdel kheleq a. Al hammouri

Signature: 

Date:2/9/2012

## Dissertation Committee

This dissertation with the title "Leader Election Algorithm For Three Dimension Torus Network In Presence Of One or Two LinkS failure" was defended and approved on 2/9/2012.

<u>Committee Member's</u>	<u>Title</u>	<u>Signature</u>
Prof. Dr. ghassan kana'an	Chair	
Prof. Dr. alaa al hammami	Member and supervisor	
Ass. Prof. Dr. basel al kasasbeh	Member	
Ass. Prof. Dr Husain al bahadly	Member	

## **Dedication**

*To my parents, my wife who kept with me day and night to complete this work and to everybody who contributed to enrich the science of human, even in one character*

## Acknowledgment

After praises and thanks to ALLAH, I would like to thank my dean and supervisor prof. Dr. Ala'a Al-Hamami, who encouraged and helped me to complete this work.

Finally, I would like to thank all lecturers, administration, and staff of Amman Arab University for their help and support.

## Table of Contents

Dedication .....	IV
Acknowledgment.....	V
Table of Contents.....	VI
List of Tables.....	IX
List of Figures .....	X
Acronyms and Abbreviations.....	XI
Abstract.....	XII
Arabic summary .....	XIV
Chapter one Introduction.....	1
1.1 Overview .....	1
1.2 About leader election algorithms. ....	1
1.3 Importance of parallel computing. ....	4
1.4 How to choose the network. ....	7
1.6 Statement of the Problem.....	10
1.7 Goals of this Dissertation.....	10
1.8 Problem Definition.....	11
1.9 Contributions.....	11
1.10 Research Limitations.....	12
1.11 Definitions of Terms: .....	12
1.12 Organization of the dissertation.....	13
Chapter-2 Approaches to Interconnection Networks and Topologies .....	14
2.1 Overview.....	14
2.2.1.1 Direct Networks Types. ....	16
2. Star Connected Networks. ....	16
3. Ring Networking.....	17
4. Bus (line) Topology .....	18
5. Hypercube Networks.....	19
6. Butterfly Networks .....	19
7. Tree Topology.....	20
8. Mesh Networks .....	21

9. Torus (Wraparound) Networks. ....	22
10. Shuffle-Exchange Network. ....	22
2.2.1.2 Direct Networks Evaluation. ....	23
2.2.2 Indirect Networks. ....	24
2.2.3. Shared-Medium Networks. ....	26
2.2.4. Hybrid Networks. ....	27
Chapter 3 Literature Review. ....	32
3.1 Introduction. ....	32
3.2 Related Works. ....	32
Chapter 4 leader election algorithms. ....	49
4.1 Introduction. ....	49
4.2 Model Description and Assumptions. ....	49
4.3 Leader Election Algorithms in 3D torus Networks. ....	52
4.3.1 Leader Election in 3D Torus Networks with one Link Failure .....	52
4.3.2 Leader election In 3D torus networks with Two links failure .....	57
Chapter 5 algorithms performance evaluation. ....	67
5.1 Introduction. ....	67
5.2 Leader Election in 3D Torus Networks with One Link Failure Analyses. ....	67
5.2.1 Number of Messages: ....	67
5.3 Leader Election in 3D Torus Networks with Two Links Failure Analyses. ....	73
5.3.1 Number of Messages: ....	74
5.3.2 Time Steps. ....	77
5.4. Simulation. ....	80
5.4.1 Programming Language Used. ....	80
5.4.2 Algorithm Simulation. ....	80
Chapter 6 Conclusion and Future Work. ....	94
6.1 Introduction. ....	94
6.2 Results. ....	94
6.2.1 Results of the First Proposed Algorithm. ....	94
6.2.2 Results of the Second Proposed Algorithm. ....	95

6.3 Future Works.....	96
References.....	97
Appendices .....	103



## List of Tables

Table	Title	Page
1	Experiments of Computer Simulation	51
2	Link Failure Solution By Detours	73

## List of Figures

Figure	Title	Page
1	Direct Network router based Examples	6
2	Complete Connection Network	21
2	Star Connection Network	22
4	Ring Network	23
5	Bus Topology	24
6	0D to 4D Hypercube topology	25
7	Butterfly topology	26
8	8 Tree Network	27
9	2D Mesh (4 * 4)	28
10	2D Torus (6 * 6)	28
11	shuffle-exchange network	29
12	Bus- Based Network	32
13	Cross-Bar Switch Network	33
14	Omega interconnection networks	34
15	Shared-Medium Networks	35
16	Deterministic Routing	39
17	directions in 3d torus	65
18	(3 X 3X 3) Torus Networks	66
19	link failure in phase 2	69
20	Phase 3 election steps and link failure solution	71
21	Phase 4 election steps and link failure solution	72
22	Phase 5 Broadcast leader message through all axes	73
23	two link failure phase two	76
24	Phase three two link failure	77
25	Phase four election	78
26	26 phase four election two link failure +y,+x	80
27	with two link failure +x, +x	81
28	State diagram for nodes	84
29	Flow chart	85

## Acronyms and Abbreviations

abbreviation	acronym
3DT	Three Dimension Torus
ACK	Acknowledgment
CSMA/CD	Carrier Sense Multiple Access With Collision Detection
CSMA/CA	Carrier Sense Multiple Access With Collision Avoidance
DS	Distributed System
DSPT	Distributed Space Partitioning Tree
ID	Identification Number Of The Node
LAN	Local Area Network
LEA	Leader Election Algorithm
MAU	Media Access Unit
MANET	Mobile ad hoc networks
O(n)	Mechanical function used to evaluate the algorithm in time unit
RN	Radio Networks
WAN	Wide Area Net Work
Wid	Weighted Identification Number

## Abstract

Leader election is a very important algorithm in wired and wireless networks. It is used to solve the single point failure in distributed systems when one process which called leader is responsible to coordinate and manage the whole network. The leader election algorithms solve the instability problem in the network, which is caused by leader failure. The algorithm aims to find a new leader identified by some characteristics from all other nodes. When the algorithm is terminated, the network is returned to a stable state with one node detected as a leader, All the other nodes will be informed to be aware of this leader. The algorithm starts by one or more processes who detect that the leader process is failed for any reason like battery age, CPU speed, memory...etc . It terminates when all nodes know who the new leader is.

**The research work reported here is concerned with building and designing two leader election algorithms, to contribute in solving leader crash problem in three dimensional torus networks. The algorithms solve the problem in presence of one/two link failure.**

Many issues are considered when designing these algorithms, For instance, Collision avoidance mechanisms which may decrease the message and time complexity, and messages synchronization which may help in concurrent execution overall the network.

Algorithm performance is evaluated by calculating the number of messages and time steps overall the algorithms. In a network of N nodes connected by a three dimensional torus network, the performance is evaluated in simple case, when leader failure is detected by one node and In the worst case, when leader

failure is detected by (N-1) nodes. For all cases the number of messages is  $O(N)$  in  $O(\sqrt{N})$  steps. This result is valid even in the presence of one or two links failure. Simulation model is used to proof the result of mathematical analysis.

## Arabic summary

خوارزميات انتخاب القائد

للشبكات المحدبة ثلاثية الأبعاد في حالة وجود عطل في وصلة أو وصلتين

إعداد

عبد الخالق الحموري

إشراف الاستاذ الدكتور

علاء الحمامي

تعتبر خوارزمية انتخاب القائد من الخوارزميات المهمة في الشبكات السلكية واللاسلكية في انظمة الحوسبة الموزعة مركزية التحكم لمختلف التراكيب الشبكية، حيث يكون أحد معالجات الشبكة مركزا للتحكم ويسمى القائد بينما تكون باقي المعالجات تابعة له. يستخدم هذا النوع من الخوارزميات لحل مشكلة تعطل قائد الشبكة وتبدأ هذه المشكلة عندما يتم اكتشاف تعطل القائد من قبل معالج آخر أو أكثر، فتدخل كافة معالجات الشبكة في انتخابات لاختيار أحدها قائدا جديدا للشبكة، وتنتهي عملية الانتخاب بمعرفة جميع المعالجات بالقائد الجديد.

تبحث هذه الأطروحة في خوارزميات انتخاب القائد في الشبكات المحدبة ثلاثية الأبعاد في حالة وجود عطل في وصلة أو وصلتين، وقد تم اقتراح خوارزميتين لانتخاب القائد، الاولى تقدم فكرة جديدة بكفاءة عالية لعملية الانتخاب في هذا النوع من الشبكات في حالة وجود عطل متقطع او دائم في وصلة واحده. بينما تبتكر الثانية حلا لمشكلة تعطل القائد في هذه الشبكات في حالة وجود عطل متقطع او دائم في وصلتين من الخطوط الواصلة بين المعالجات في اي موقع في الشبكة. وتقدم الأطروحة اثباتا رياضيا يعتمد على حساب عدد الرسائل اللازمة لإتمام عمل الخوارزمية وعدد الخطوات الزمنية بالإضافة لبرنامج simulation يوضح الخوارزمية ويعزز الاثبات الرياضي.

علما بأن الدراسة تأخذ بعين الاعتبار القضايا الأساسية التي ترافق هذا النوع من الخوارزميات، مثل التزامن (Synchronization) والتزام (Contention).

إذا افترضنا أن  $N$  عدد المعالجات في الشبكة فإن كل من الخوارزميتين المقترحتين تستخدم في جميع الحالات  $O(N)$  رسالة ضمن  $O(\sqrt{N})$  خطوة زمنية.

# Chapter one

## Introduction

### 1.1 Overview

In centralized control networks, one of the most fundamental problems is the single point failure, which is also called leader failure or leader crash. This problem converts the network to unstable state in which all processes cant reach the leader process . So all processes are in the same state, that is, they all are candidates and can become new leader. The common solution for this problem is Leader Election Algorithms (LEAs) which is a distributed algorithm, LEA return the network to normal state where one process as leader and all nodes aware of this leader (Antoniou and Srimani, 1996 ; Castillo et al. ,2007).

In distributed computing, leader election is the process of designating a single process as the organizer of some task distributed among several processes. Before task begins, all network nodes are unaware which node will serve as the "leader" or coordinator of the task. After a leader election algorithm has been run, however, each node throughout the network recognizes a particular, unique node as the task leader, The network nodes communicate among themselves in order to decide which of them will get into the "leader" state. For that, they need some method in order to break the symmetry among them. For example, if each node has unique identities, then the nodes can compare their identities, and decide that the node with the highest identity is the leader (Kariwala, 2011).

### 1.2 About leader election algorithms.

LEA is a program distributed over all processes. It starts when the leader failure is detected by one process at a simple case or by

all processes at the worst case. It terminates when all processes are aware of the new elected leader (Singh, 1996; Tanenbaum, 2002).

LEAs are widely used in centralized systems to solve a single point failure problem. For example, in client-server, the LEAs are used when the server fails and the system needs to transfer the leadership to another station. The LEAs are also used in a token ring. When the node that has the token fails, the system should select a new node to have the token (Abu-Amara and Lokre, 1996). Leader election is an important primitive for distributed computing, useful as a subroutine for any application that requires the selection of a unique processor among multiple candidate processors. Applications that need a leader range from the primary-backup approach to replication based fault-tolerance to group communication systems, and from videoconferencing to multi-player games (Ingram et al., 2009).

In this research, the LEAs from a parallel perspective will be studied. In parallel computing, many processors cooperate together to solve the same software problem. A bottleneck in parallel computing systems is the communication between processors. The merging of computers and communications has had a profound influence on the way computer systems are organized. The old model of a single computer serving all of the organization's computational needs has been replaced by one in which a large number of separate but interconnected computers do the job. These systems are called Interconnection networks (Tanenbaum, 2003).

Interconnection networks can be classified into four major categories: shared-medium networks, direct-networks (router-based), indirect networks (switch-based) and hybrid networks.



In shared-medium networks, processors are connected by a common transmission medium such as a bus. All processors share the medium which does not generate a message.

To send a message to a destination, a source broadcasts the message on the medium and the destination picks up the message. Because processors may send messages to the medium simultaneously, the resolution of network access conflicts is needed. The nature of the shared medium also limits the bandwidth of the network and the number of end systems in the network. Examples of the shared medium networks include the Ethernet. The protocol for the medium access control used in the Ethernet is known as CSMA/CD (Carrier Sense Multiple Access with Collision Detection).

In point-to-point networks, end systems are connected by point-to-point communication links. The networks can be further classified into two categories: direct networks and indirect networks.

In direct networks, point-to-point links directly connect each end system to some other end systems. In indirect networks, end systems are connected via one or more switches and switches are connected via point-to-point links.

Some networks may have more complicated structures such as hierarchical structures or hyper graph topologies. Such networks are classified as hybrid networks. On the other hand direct networks consist of a set of nodes and a set of point-to-point links. Each node is directly connected to a small subset of nodes by links. Each node performs both routing and computing. A direct network is usually modeled as a graph, with vertices and edges. Figure 1 shows direct networks router based examples.

## Direct Networks (Router-based)

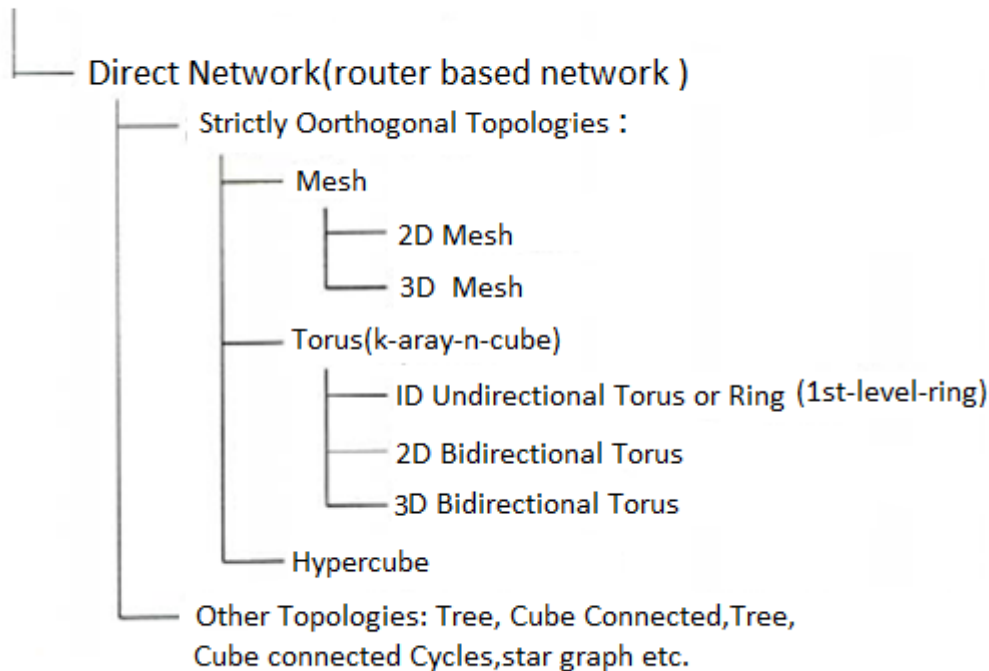


Figure 1: Direct Network router based Examples

Therefore, the performance of interconnection networks is a critical issue in parallel computing. This has been a major driving force for the research of inter-connection networks. The study of interconnection networks in parallel computing system includes the performance and cost issues (Qianping ,2011).

### 1.3 Importance of parallel computing.

Parallel computers are divided into two main types: multiprocessors, where all processors use the same memory, and multi-computers, where each computer has local memory and processor.

In distributed system each node must operate accurately to cooperate with other node, there are some mechanism needed such as file server, time server and central lock coordinator in

the distributed system in generally, these servers are called leaders. Algorithms which select a leader are called the leader election algorithms (Mirakhorli et al. ,2007).

There is considerable confusion in the literature between a computer network and a distributed system. The key distinction is that in a distributed system, a collection of independent computers appears to its users as a single coherent system. Usually, it has a single model or paradigm that it presents to the users. Often a layer of software on top of the operating system, called middleware, is responsible for implementing this model. A well-known example of a distributed system is the World Wide Web, in which everything looks like a document (Web page).In a computer network; this coherence, model, and software are absent(Tanenbaum ,2003).

The machines look and act in a coherent way. If the machines have different hardware and different operating systems, that is fully visible to the users. If a user wants to run a program on a remote machine, he/she has to log onto that machine and run it there. In effect, a distributed system is a software system built on top of a network. The software gives it a high degree of cohesiveness and transparency. Thus, the distinction between a network and a distributed system lies with the software (especially the operating system), rather than with the hardware. Nevertheless, there is considerable overlap between the two subjects. For example, both distributed systems and computer networks need to move files around. The difference lies in who invokes the movement, the system or the user (Tanenbaum ,2003).

In multi-computers, there are many network topologies such as torus, hypercube, meshes, ring, bus, star,...etc.

These topologies may be either hardware processors or software processes embedded over the other parallel hardware topology. Many parallel computers using direction interconnection network have been designed and commercialized in last decade. Mesh, torus and hypercube have been the most popular topologies (Camara et al. ,2009).

The efficiency of large-scale multi-computers is critically dependent on the performance of its interconnection network, which is greatly influenced by the topology, switching method, routing algorithm as well as the traffic pattern exhibited by parallel applications. Low-dimensional versions of k-ary n-cubes, also known as tori, have been popular in the latest generation of multi-computers due to their desirable properties, such as: ease of implementing Processor Scheduling and Allocation for 3D Torus Multicomputer Systems(Choo and youn, 2000). Although the dynamic communication performance of a program on a multicomputer depends on the actual times taken for data transfer, a smaller average distance and diameter of an interconnection network yields a smaller communication latency of that network( Al Faisal and rahman, 2009).

An important challenge added to this work is the type of routing that will be used to fast the interconnection in the network. Routing is the process of selecting paths in a network along which to send network traffic. Routing performed for many kinds of networks, including the telephone network (Circuit switching), electronic data networks (such as the Internet), and transportation networks .

The Internet is comprised of a mesh of routers interconnected by links. Communication among nodes on the Internet

(routers and end-hosts) takes place using the Internet Protocol, commonly known as IP. IP datagrams (packets) travel over links from one router to the next on their way towards their final destination. Each router performs a forwarding decision on incoming packets to determine the packet's next-hop router (Gupta, 2000).

There are a number of properties that we desire for all lookup and classification algorithms of routing:

- High speed.
- Low storage requirements.
- Flexibility in implementation.
- Ability to handle large real-life routing tables and classifiers.
- Low preprocessing time.
- Low update time.
- Scalability in the number of header fields (for classification algorithms only).
- Flexibility in specification (for classification algorithms only). In multiprocessor system, the topology of interconnection network is critical (Gupta, 2000).

#### **1.4 How to choose the network.**

To achieve this work store and forward routing will be used , so that the parallel processing system can efficiently perform various application algorithms in engineering and scientific. An interconnection network is usually represented by an undirected graph, where the node set represents the processors, and the edge set represents the communication link.

It can be said the diameter of a network is the maximum inter-node distance, Le., the maximum number of links that must be traversed to send a message to any node along the shortest path.

As a definition, the distance between adjacent nodes is unity. Diameter is the maximum distance among all distinct pairs of nodes along the shortest path. The diameter is commonly used to describe and compare the static network performance of the network's topology. Networks with small diameters are preferable. The smaller the diameter of a network, the shorter the time to send a message from one node to the node that is farthest away from it( Al Faisal and Rahman, 2009)

Inter-node distance, message traffic density, and fault tolerance are dependent on the diameter and the node degree. The product (diameter x node degree) is a good criterion for measuring the relationship between cost and performance of a multiprocessor system. An interconnection network with a large diameter has low message passing bandwidth, and a network with a high node degree is very expensive. In addition, A network should be easily scalable; there should be no changes in the basic node configuration as we increase the number of nodes. The cost of different networks is plotted and it is shown that the cost of STTN( Al Faisal and rahman, 2009) .

In a topology of interconnection network, degree of a node, diameter and cost are three important measures that aid in evaluating different network structures. The degree of a node is denoted as the number of input/output links per node. The diameter of the network is the maximum value of all shortest paths between any two nodes. In designing a topology, there is always a trade-off between degree, which relates to the hardware cost, and diameter, which relates to the message transmission time. ( Al Faisal and Rahman, 2009) .

### 1.5 torus network.

In order to reduce these three metrics and optimize other characteristics, many topologies are proposed. Among those, mesh and torus are the most popular topologies. However, they have an important drawback that they are not edge-symmetric such that the diameter is far from the optimum value. The twisted torus topologies proposed in recover edge-symmetry and, consequently, improve the performance up to 74%. The Gaussian networks proposed in model many toroidal networks including the twisted torus ( Al Faisal and Rahman, 2009) . Symmetry of torus networks topology lead to more balance utilization of communication links, under random traffic , than in mesh topology (david ,1995). Torus network has better dynamic communication performance than a mesh.

The efficiency of large-scale multicomputer is critically dependent on the performance of its interconnection network, which is greatly influenced by the topology, switching method, routing algorithm as well as the traffic pattern exhibited by parallel applications. Low-dimensional versions of k-ary n-cubes, also known as torus, have been popular in the latest generation of multicomputer due to their desirable properties, such as ease of implementation, recursive structures, and ability to exploit communication locality to reduce message latency (Min, et al. 2003).

This work proposes two algorithms to solve leader election in the 3D torus networks. The 3D torus has many preferable properties such as scalability, routing, similarity and regularity. The torus network  $T(n, p)$  with  $n$  columns and  $p$  rows is certainly one of the most popular regular networks. Indeed, this is due to the fact that we have to place

a circuit in a torus and minimize the distance of any point in the torus( Barth, and Berthomé ,2004). These properties make the 3D torus more flexible and more preferable topology.

There is not LEAs to solve the leader failure problem with the existence of link failure for the 3D torus networks. This work will present a new solution for leader election problem in the 3D torus with the presence of links failure. So, the proposed solution will take in consideration, the existence of links failure along with leader failure problem.

### **1.6 Statement of the Problem.**

Leader election algorithms currently don't solve the problem of leader failure for the three dimensional torus networks with presence of links failure. The purpose of this study is to design two new algorithms to solve this problem in the three dimensional torus with the presence of links failure. The first algorithm will solve the problem of link failure with presence of one link failure; the second algorithm will solve the same problem with presence of two link failure. The solutions will solve this problem with less time steps and less number of messages.

### **1.7 Goals of this Dissertation.**

In this dissertation two new algorithms will be proposed, to solve the leader problem. The problem of leader failure in the three dimension torus network will be solved by new algorithms that have the following objectives:

Solve the problem with the presence of link failure.

Propose two new efficient algorithms with high speed .

Less number of message and time steps .

The length of message will be shorter.



The dissertation will use asynchronous communication in sending messages during communication.

### **1.8 Problem Definition.**

Parallel and distributed systems are widely used to increase the speed of computations. In these systems, many computers cooperate with each other to solve the computation problem. The control of these systems may be centralized by one node called leader. On the other hand, the control may be distributed among nodes. In centralized control, fault tolerance of leader crash is a very important issue in these systems. This problem can be solved by electing a new node to substitute the leader failure. The election process is the same program distributed over all nodes. It starts when one or more nodes discover that the leader has failed. It terminates when the remaining processors know who the new Leader is. This dissertation will present new algorithms to elect a new leader in three dimensional torus networks with the following properties:

Link(s) failure will be tolerated.

Less contention and light load over the network.

The number of time steps and the number of messages overall the algorithm will be minimized.

Asynchronous and synchronous communications to transfer messages will be used.

Mathematical analyses will be provided for each algorithm and simulation models for the two algorithms will be made.

### **1.9 Contributions**

The contributions of this research are to present two new algorithms.

The first one solves the leader failure problem in three dimensional torus networks despite the existence of one link failure. The second algorithm solves the problem despite the presence of more than one link failures.

These two algorithms will solve the problem of Leader selection in torus network.

### **1.10 Research Limitations**

The proposed algorithms will not be applied on real systems in this dissertation. To validate the results, the study will present a mathematical analysis for the two proposed algorithms and simulation.

### **1.11 Definitions of Terms:**

**Distributed system:** It is a collection of autonomous processes that communicate with each other, either synchronously or asynchronously.

**Leader failure:** the node loses the leader proprieties but is still able to receive and pass messages.

**Leader election:** A process of choosing a new leader to substitute the failure one in a centralized control networks.

**Link failure:** A broken link i.e. no messages can pass through this link.

**Candidate:** A state of a node when it is aware of the failure and participates in the election process.

**Normal:** A state of a node when it knows the leader node and the network is stable.

**Node-ID:** Each node has a unique ID that is used to participate in the election process. This ID reflects the priority of the node over other nodes in relation to the new leader.

## **1.12 Organization of the dissertation.**

Chapter One: **overview about leader election algorithm problem, problem statement, goals of the dissertation, contributions and dissertation organization.**

Chapter Two: **parallel and distributed information related to dissertation, distributed systems, multicomputer, multiprocessors, interconnection network, design issues, Performance and complexity and finally the description and properties of the torus model.**

Chapter Three: **literature review for the previous works.**

Chapter Four: **present the two proposed algorithms.**

**Chapter Five: analyses of the two proposed algorithms. Mathematical proves and simulation that verify the algorithms validity.**

Chapter Six: **conclusions and future works.**

## Chapter-2

### Approaches to Interconnection Networks and Topologies

#### 2.1 Overview.

Leader election algorithm (LEA) is a fundamental problem in distributed computing as well as in parallel computers. The main function of this algorithm is to solve, the popular problem, single point failure. It has been studied in various computation models. In the leader election problem there are  $N$  processes or nodes in the network. Each node has a unique Identity (ID) which represents the priority or the weight to be the next leader. Initially all nodes are passive and unaware of the identity of any other nodes. An arbitrary subset of nodes, called the candidates, wake up spontaneously and starts the election protocol. On the termination of the election protocol, exactly one node is announced as the new leader where all other nodes are aware of this leader (Singh, 1992). Leader election problem was also defined as follow: at any point in time there exists at most one leader for the network, and when there is no leader at any time then within at most  $K$  time units a new leader must be elected (Fertzer et al, 2000). Researchers used different methods to validate their algorithms. One method is to use mathematical models to compute the number of messages and the number of time steps and another way is to use simulation program to validate the results. In this dissertation we considered the problem of electing a leader in 3D torus network. We use both mathematical analyses and simulation program to validate our results.

## Interconnection Networks

Interconnection networks offer an attractive solution to this communication crisis and are becoming pervasive in digital systems. A well-designed interconnection network makes efficient use of scarce communication resources — providing high bandwidth, low-latency communication between clients with a minimum of cost and energy (Culler et al,1999).

The communication process between distributed and parallel computers, types : multicomputer and multiprocessors, connects the processors with the shared memory or with each other. This interconnection uses many types of networks topology which can be classified into two types: dynamic and static networks. Static network uses point to point communication lines in its processors interactions. Dynamic networks are constructed by using switching elements and communication lines (Kumar, 2005). Interconnection networks in parallel and distributed networks have four types:

Direct networks.

Indirect networks.

Shared-Medium Networks.

Hybrid networks

Direct Networks:

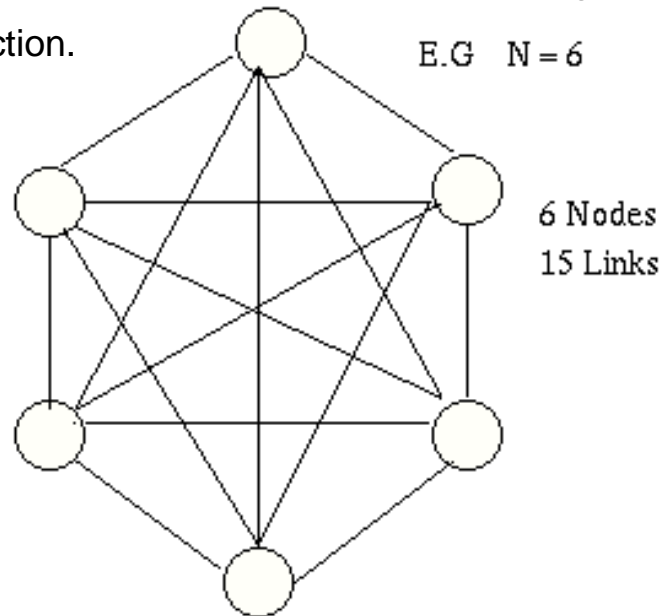
A network topology describes the arrangement of systems on a computer network. It defines how the computers, or nodes, within the network are arranged and connected to each other.

Direct networks use channels to connect network processors. These Processors communicate with each other and exchange the data using message passing through the channels. Message passing is achieved by a set responsible to direct the messages from source to destination, called router (Culler et al, 1999).

### 2.2.1.1 Direct Networks Types.

#### 1. Fully connected or all-to-all (Complete) Network

This is the most powerful interconnection network (topology): each node is directly connected to all other nodes. Figure-2 shows this type of connection.



Each node (neighbors) giving a total of  $N(N-1)/2$  connections for the network. Even though this is the best network to have the high number of connections per node means this network can only be implemented for small values of N. Therefore some form of limited interconnection network must be used.

#### 2. Star Connected Networks.

In local area networks with a star topology, each network host is connected to a central hub with a point-to-point connection. The network does not necessarily have to resemble a star to be classified as a star network, but all of the nodes on the network must be connected to one central device. All traffic that traverses the network passes through the central hub. The hub acts as a signal repeater.

The star topology is considered the easiest topology to design and implement. An advantage of the star topology is the simplicity of adding additional nodes. The primary disadvantage of the star topology is that the hub represents a single point of failure .

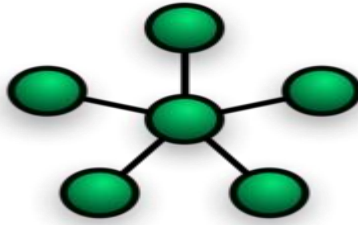


Figure 3 : Star Connection Network

### 3. Ring Networking

A ring network is Local Area Network (LAN) in which the nodes (workstations or other devices) are connected in a closed loop configuration. Adjacent pairs of nodes are directly connected. Other pairs of nodes are indirectly connected, the data passing through one or more intermediate nodes. Each node is shown as a sphere, and connections are shown as straight lines. The connections can consist of wired or wireless links. A break in the cable of a ring network may result in degraded data speed between pairs of workstations for which the data

path is increased as a result of the break. If two breaks occur and they are not both in the same section of cable, some workstations will be cut off from some of the others. When system reliability is a critical concern, a bus network or star network may prove superior to a ring network. If redundancy is required, the mesh network topology may be preferable. In ring topology each processor connects to two processors to right and left. If the first and last

processors connect to one processor the topology becomes linear array or bus Figure-4 shows the ring Network graph (Culler et al, 1999; Kumar et al,2003).

#### **4. Bus (line) Topology**

Nodes are connected to a main (bus) cable. If data is being sent between nodes then other nodes cannot transmit. If too many nodes are connected then the transfer of data slows dramatically as the nodes have to wait longer for the bus to be clear. Bus topology has many advantages such as: the simplest and cheapest to install and extend and well suited for temporary networks with not many nodes and flexible topology as nodes can be attached or detached without disturbing the rest of the network. Also failure of one node does not affect the rest of the simpler than a ring topology to troubleshoot if there is a cable failure because sections can be isolated and tested independently. On the other hand bus topology has disadvantages such as : If the bus cable fails then the whole network will fail and performance of the network slows down rapidly with more nodes or heavy network traffic, also the bus cable has a limited length and must be terminated properly at both ends to prevent reflected signals and slower than a ring network as data cannot be transmitted while the bus is in use by other nodes. Figure-5 shows the bus topology.



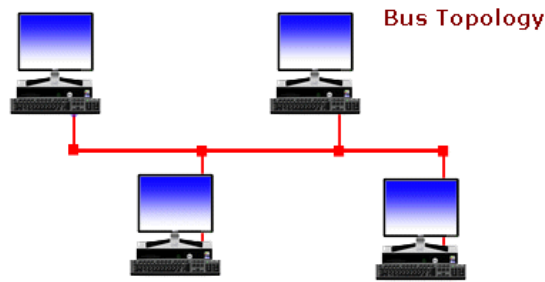


Figure 5 : Bus Topology

## 5. Hypercube Networks

This topology has  $N$  processors which equal  $2^d$  ( $d$  is the hypercube dimension). Each processor connects to  $d$  neighbors; hypercube is symmetry graph which means that all nodes are the same position properties. It has the shortest diameter and many other advantages but the complexity in number of links as the size increase is the main disadvantage of this topology. Figure 6 shows different dimensions of hypercube topology.

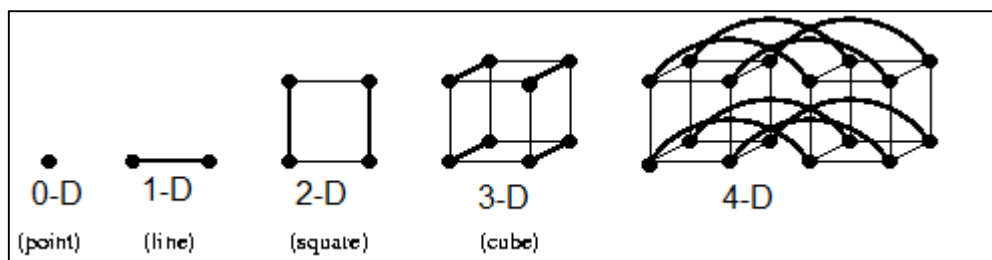


Figure 6: 0D to 4D Hypercube topology

## 6. Butterfly Networks

A scheme that connects the units of a multiprocessing system and needs  $n$  stages to connect  $2n$  processors; at each stage a switch is thrown, depending on a particular bit in the addresses of the processors being connected. Figure 7 shows the Butterfly

topology.

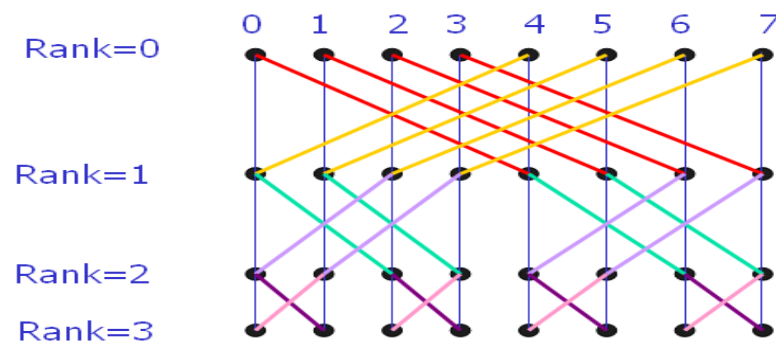


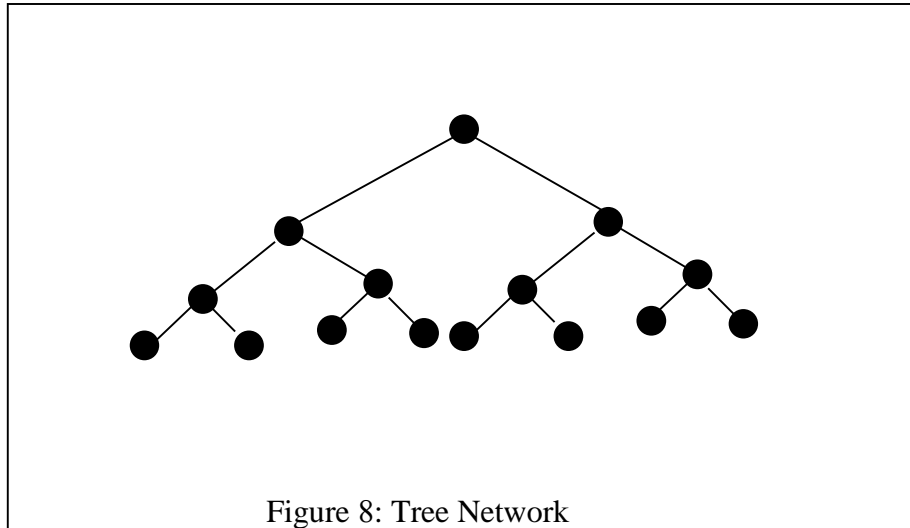
Figure 7: Butterfly topology

## 7. Tree Topology

Tree Topology is a

combination of the bus and the Star Topology. In tree network there is only one path between any two processors. The tree-like structure allows you to have many servers on the network and you can branch out the network in many ways.

A Tree Structure suits best when the network is widely spread and vastly divided into many branches. Like any other topologies, The tree Topology has its advantages and disadvantages. A Tree Network (Figure 8) may not suit small networks and it may be a waste of cable to use it for small networks. Tree Topology has some limitations and the configuration should suit those limitations.



### 8. Mesh Networks

In mesh topology nodes are arranged into q-dimensional lattice. Communication is allowed only between neighboring nodes; hence interior nodes communicate with  $2q$  other processors. The diameter of a q-dimensional mesh with  $k * k$  nodes is  $q(k-1)$ . The bisection width of q-dimensional mesh with  $k * k$  nodes is  $k * q-1$ . The max edges per node is  $2q$  and The max edge length is constant for two & three dimensional mesh. Figure 9 shows 2D Mesh.

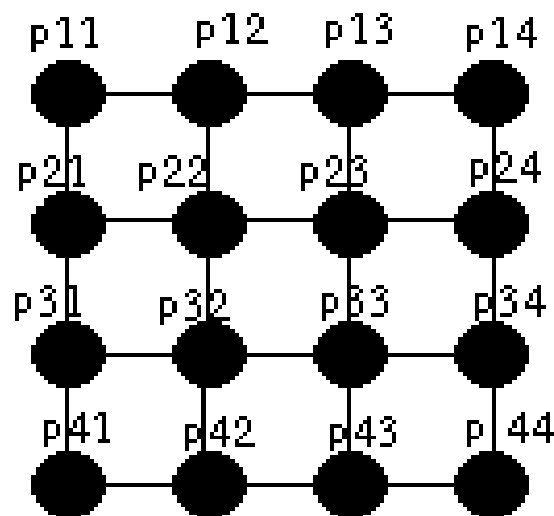


Figure 9: 2D Mesh (4 \* 4)

## 9. Torus (Wraparound) Networks.

Torus network doesn't differ from meshes except in the connection between the first and the last nodes in each dimension. This connection makes all nodes connect to the same number of neighbors Figure 10 shows two dimensional torus (Culler et al, 1999;Kumar et al,2003; William and Winnipeg,2001).

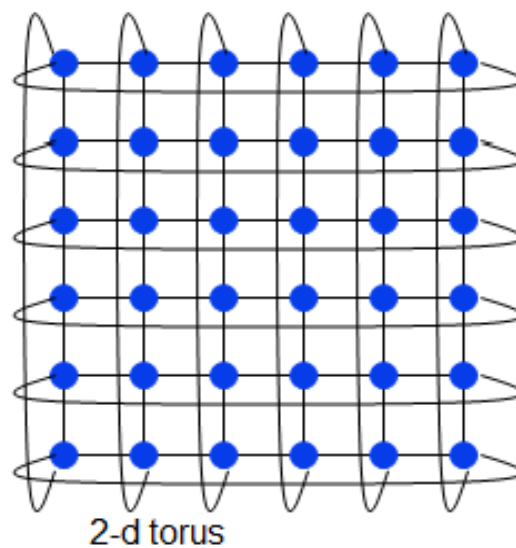


Figure 10: 2D Torus (6 \* 6)

## 10. Shuffle-Exchange Network.

In the static shuffle-exchange, the curved arcs take the same role as the shuffles between the switch boxes in the Omega, and the straight arcs take the same role as the switch boxes. This gives us, once again, an  $O(\log n)$  latency and an  $O(n)$  aggregate bandwidth; the bisection bandwidth is always 4 (so it's  $O(1)$ ) and the nodes always have 3 links, so the cost is  $O(n)$  as shown in Figure-11.

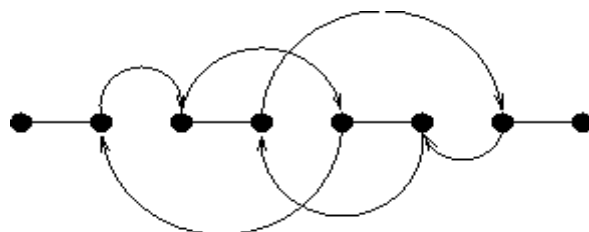


Figure 11: shuffle-exchange network

### 2.2.1.2 Direct Networks Evaluation.

The following properties determine the utilization and performance of the direct networks:

**1. Diameters:** the diameter of a network is the maximum distance between any two processing nodes in the network. The distance between two processing nodes is defined as the shortest path between them (Kumar et al,2003). The shortest diameter is desirable because the distance determines the communication time. The diameter in the complete networks is one link. In star network the diameter is two and in ring it is  $\lfloor P/2 \rfloor$  links when P is the number of processing units. The diameter in complete binary tree is  $2(\text{Log}((P+1)/2))$ , in mesh without wraparound it is  $(P^{1/2}-1)$ . The diameter in wraparound meshes (torus) network is  $2 \lfloor P^{1/2}/2 \rfloor$ . For the hypercube the diameter is  $\text{Log } P$  (Culler et al, 1999; Kumar et al,2003).

**2. Connectivity.** The connectivity of a network is a measure of the multiplicity of paths between any two nodes. One measure of connectivity is the minimum number of arcs that have to be removed to break down the network into two parts. This is called the arc connectivity of the network. It is two for rings and 2-d meshes without wraparound and d for d-dimensional hypercube. The arc connectivity is one in star network, tree and linear arrays (Culler et al, 1999;Kumar et al,2003).

**3. Cost.** The cost can be determined by a number of communication links in the network. For instance linear and tree networks required P-1 links, torus needs Width\*Length links. Hypercube needs  $(P \text{Log}(p))/2$  (Culler et al, 1999;Kumar et al,2003).

**4. Symmetry.** The network is symmetry when it is the same when looking at it from different sides (Schneider, 1993; Ostrovsky et al,1994)).

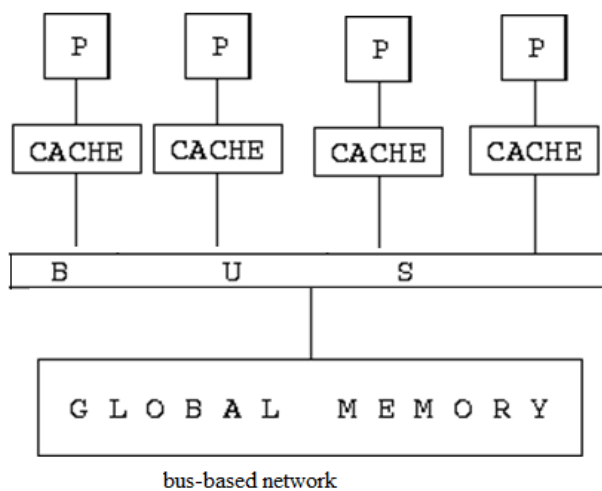
### 2.2.2 Indirect Networks.

There are many types of indirect networks that use shared memory:

#### Bus-Based Networks.

A bus-based interconnection network here used to implement a shared-memory parallel computer. Each processor (P) is connected to the bus, which in turn is connected to the global memory. A cache associated with each processor stores recently accessed memory values, in an effort to reduce bus traffic as shown in Figure 12.

Buses are commonly used in shared-memory parallel computers to communicate read and write requests to a shared global memory. In principle, the use of a global memory in a shared-memory computer simplifies parallel programming by making locality a nonissue., most shared-memory parallel computers introduce caches in an attempt to reduce bus traffic; hence, locality continues to be important.



#### Cross-Bar Switch Network

Figure 12 :Bus- Based Network

In a network, a cross-bar switch is a device that is capable of channeling data between any two devices that are attached to it up to its maximum number of ports. The paths set up between devices can be fixed for some duration or changed when desired and each device-to-device path (going through the switch) is usually fixed for some period.

Cross-bar topology can be contrasted with bus topology, an arrangement in which there is only one path that all devices share. Traditionally, computers have been connected to storage devices with a large bus. A major advantage of cross-bar switching is that, as the traffic between any two devices increases, it does not affect traffic between other devices. In addition to offering more flexibility, a cross-bar switch environment offers greater scalability than a bus environment.

In an IBM mainframe environment, the ESCON director is an example of a cross-bar switch(Rouse ,2007) as in Figure 13.

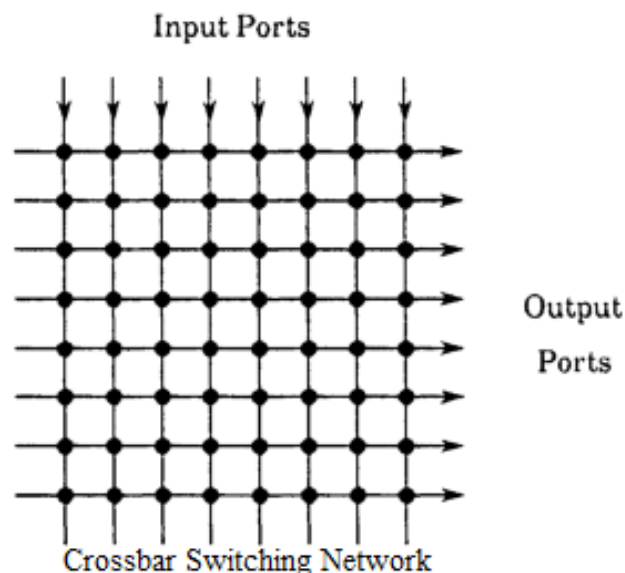
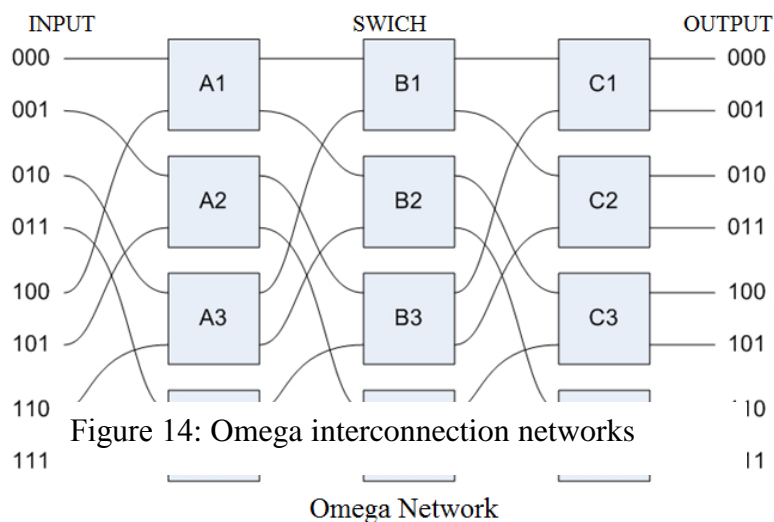


Figure 13: Cross-Bar Switch Network

### 3. Multistage Interconnection Networks.

Bus networks are scalable in terms of cost and not scalable in terms of performance. Crossbar interconnection is scalable in terms of performance and not scalable in terms of cost. An intermediate class of network connection called multistage interconnection network lies in between. This network consists of P processing nodes and b memory banks. Figure 14 shows commonly used Omega interconnection networks



#### 2.2.3. Shared-Medium Networks.

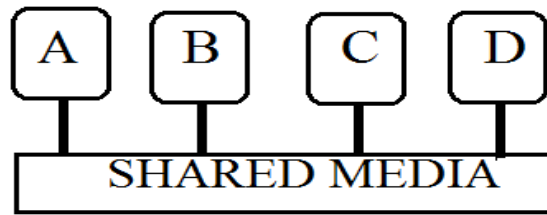
Some network topologies share a common medium with multiple nodes. At any one time, there may be a number of devices attempting to send and receive data using the network media.

When two or more nodes are sending data at the same time, data may be unusable due to collision. There are rules that govern how these devices share the media to solve the collision problems. There are two basic media access control methods for shared medium:

CSMA/Collision Detection

CSMA/Collision Avoidance





Dynamic network topologies change, as nodes are added and removed. Traffic patterns change cyclically. Overall network load changes. Figure 15 :Shared-Medium Networks

Dynamic network topologies change, as nodes are added and removed.

Traffic patterns change cyclically.

Overall network load changes.

#### 2.2.4. Hybrid Networks.

Hybrid networks use a combination of any two or more topologies in such a way that the resulting network does not exhibit one of the standard topologies (e.g., bus, star, ring, etc.). For example, a tree network connected to a tree network is still a tree network topology. A hybrid topology is always produced when two different basic network topologies are connected. Two common examples for Hybrid network are: "star ring network" and "star bus network". A Star ring network consists of two or more star topologies connected using a [Media Access Unit multi station access unit] (MAU) as a centralized hub. A Star Bus network consists of two or more star topologies connected using a bus trunk (the bus trunk serves as a network's backbone).

While grid and torus networks have found popularity in [high-performance computing] applications, some systems have used [genetic algorithms] to design custom networks that have the fewest possible hops in between different nodes. Some of the resulting layouts are nearly incomprehensible, although they function quite well.

A Snowflake topology is really a "Star of Stars" network, so it exhibits characteristics of a hybrid network topology but is not composed of two different basic network topologies being connected together.

**Definition: Hybrid topology is a combination of Bus, Star and ring topology.**

### **Routing Mechanism for Direct Networks.**

The Internet is comprised of a mesh of routers interconnected by links. Communication among nodes on the Internet (routers and end-hosts) takes place using the Internet Protocol, commonly known as IP.

IP datagram's (packets) travel over links from one router to the next on their way towards their final destination. Each router performs a forwarding decision on incoming packets to determine the packet's next-hop router

Routing mechanism determines the path for messages from source to destination. It affects network performance and speed. It uses source and destination as inputs. Some routing techniques adaptive, depends on the network state.

#### **2.3.1 Routing Mechanism Classifications.**

Routing mechanism classifications depend on many factors such as the distance from source to destination, the position of routing decision and the number of addresses the message contains.

**Distance from Source to Destination Factor.** The routing mechanisms classify into two types: Minimal routing and non minimal routing. The first one selects the shortest path from source to destination. In the second the shortest path is not important in selecting the routing. Minimal routing causes contentions in some parts in the network, while the non minimal routing may use long distance to avoid this problem (Duato et al, 1997).

### **Routing Decision Location Factor.**

Depending on this factor routing mechanisms are classified into four types:

#### **a. Centralized Routing.**

A main controller updates all nodes routing tables.

Fault tolerant.

Suitable for small networks.

**b. Distributed Routing.** The responsibility of the routing process is distributed among more than one router.

Route computation shared among nodes by exchanging.

Widely used.

**c. Source Routing.** The router of the source node is responsible for routing the messages.

**d. Hybrid Routing.** The routing process is between Source routing and distributed routing (Quinn, 1994).

#### **- Number of Addresses Factor.**

According to this factor, the routing mechanism is divided into two types:

**Unicast Routing.** Message carries one address on this mechanism. It uses point-to point communication between processors. Despite, the simplicity of this type, time to complete the communication process is the main disadvantage (Afek and Gafni ,1991).

**Multi destination Routing.** Message carries more than one address on this mechanism. It uses one-to-many communication between processors (Foster, 1995).

There are two types of routing mechanism, deterministic routing as shown in figure 16 and Adaptive routing. Current Deterministic Routing. Deterministic (static) routing : fixed path Minimal and deadlock free(Pimentel , 1999).

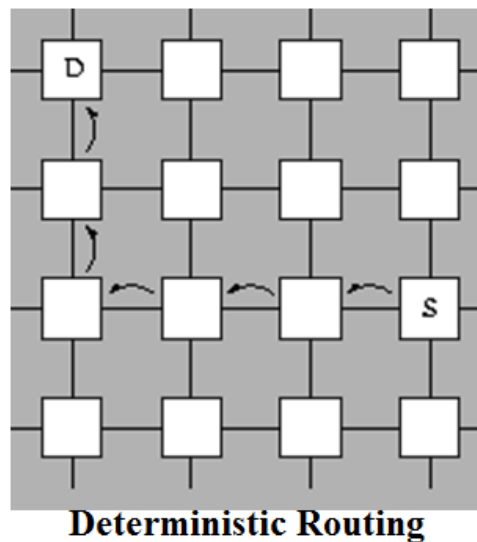


Figure 16: Deterministic Routing

### **Adaptive Routing.**

Adaptive routing exploits alternative paths, Less prone to contention and more fault-tolerant, Potential deadlocks.

Reassembling of messages (out-of-order arrival of packets), Cannot be source-based routing, Minimal or non-minimal , Partially adaptive vs fully adaptive. This type uses the information regarding the current state of the network to determine the path of the message( Pimentel , 1999).

### **2.4 Research Assumptions.**

This Research assumes the following:

Routers should work all the time even with fault node because the fault is in leader properties.

All the communication links are bidirectional.

Leader node could fail due to different reasons which lead to lose the leader property. Other nodes can detect this failure, when the time out exceed without acknowledgement. Nodes detect this failure start the election algorithm.

To solve leader failure problem, each node calculates a weight that defines its relative importance. Then compares it with the weight of other nodes that it has received and propagate the maximum weight. This weight is represented in this dissertation by ID variable. Each node has a distinguished ID. The election algorithm depends on this ID.

The fault node shared in the election with ID = 0, so it can not win the election.

One intermittent link failure is recoverable.

Leader failure may be detected by a subset of nodes (concurrent failure). This case becomes complicated when the failure is detected by N-1 nodes (worst case).

Each node has the following variables:

ID: A unique value for the election process.

Position: The label indicates its position.

Leader ID and Leader position.

Phase and step.

State: leader or normal or candidate.

Flags to synchronize the election algorithm.

## Chapter 3

### Literature Review

#### 3.1 Introduction

In distributed systems the major problem that faces the researchers and scientists is the single point of failure and the relevant Leader Election Algorithms (LEAs) . Previous studies in LEAs vary based on the following:

Topology type (e.g., tree, complete graph, meshes, torus, and hypercube).

Communication Mechanism used (synchronous vs. asynchronous).

Transmission Media (wired vs. wireless or radio).

Some of the previous work dealt with the link failure.

Most previous researches are based on mathematical proof to verify their algorithms. They use minimum number of messages and time steps to complete their algorithms. This chapter presents the previous work in election algorithms and focus to the most relevant researches.

#### 3.2 Related Works

Zhenyu xu. and srimani p. 2006.

They proposed a new self-stabilizing anonymous LEA in a tree graph. They showed the correctness of the protocol and also showed that the protocol terminates in  $O(n^4)$  time starting from any arbitrary initial state. The protocol can elect either a leaf node or a non leaf node (starting from the same initial state) depending on the behavior of the daemon. This algorithm runs on arbitrary anonymous tree graph, starting in the tree  $T$  and  $N(i)$  to be the set of immediate neighbors of a node  $i$  in the tree. A node  $i$  is called leaf node iff  $|N(i)|=1$ . A node  $i$  is called internal node if  $|N(i)| > 1$ . When

the algorithm terminates, one and only one node is elected to be the leader. All other nodes will have a local pointer that pointing to a neighboring node which is on the unique path to the leader node in the tree.

Vasudevan S. et al. (2005) .

They presented a LEA that is highly adaptive to arbitrary (possibly concurrent) topological changes and is therefore well-suited for use in mobile ad hoc networks(MANETs). The algorithm is based on finding an extremis and uses diffusing computations for this purpose. They showed that, using a linear-time temporal logic, that the algorithm is weakly. Self-stabilizing and terminating. Then simulate the algorithm in a mobile and ad hoc (MANETs) setting. Through the simulation study, it elaborate on several important issues that can significantly impact performance of such a protocol for mobile ad hoc networks such as choice of signaling, broadcast nature of wireless medium etc. the simulation study shows that the algorithm is quite effective in that each node has a leader approximately 97-99% of the time in a variety of operating previous algorithms are designed to perform random node election and cannot be modified to perform extreme ending. It therefore proposes an election algorithm to perform extreme-ending in a highly dynamic and asynchronous environment such as found in a mobile, ad hoc network. Unlike existing work on leader election, an important contribution of this paper is that it presents a very systematic and methodical evaluation of our leader election algorithm based on simulations in a mobile, wireless setting. The simulation study provides useful insights relating to the design of our algorithm and the signaling used. As we will see, the paper exploits these insights to develop a very efficient leader election algorithm.

Villadanjós s. et al, (2006).

They analyze the algorithms for leader election in complete networks using asynchronous communication channels. They presented a novel algorithm that reduces the information necessary to select a leader compared with other leader election algorithms for complete networks. They showed that, the algorithm works without sense of direction means, in general, that given a graph  $G \equiv (V, E)$ , being  $V$  the nodes of the graph,  $E$  its set of edges between nodes of  $V$  and  $E(n)$  the edges of node  $n$  in  $G$ ; There are different alternatives for sense of direction functions, which derive in different sense of direction definitions. And, it does not require to know the number of nodes in the system. The proposal of this paper requires  $O(n)$  messages and  $O(n)$  time, where  $n$  is the number of nodes in the system, to elect a leader.

Zhang (2009).

They presented, a low-level asynchronous algorithm with the collision avoidance mechanism, and solved the problem. They designed experiments in application layer. The leader-elected time was from those experiments and the result was compared with synchronous algorithm. This algorithm can be used in fire fighting, construction supervision and rescue work in which a hierarchy ad hoc network is required to dynamically be established. By electing a new leader in case of previous leader loss, the algorithm can enhance the robustness of whole network.

The proposed algorithm elects leader node in a hierarchy ad hoc network. So the whole network must be established before the algorithm designs the steps for setup as follows:

Each node detects nearby connectable nodes, and notes them as neighbor nodes.



Root node sends the SETUP message to its neighbors each node has a value called SETUP\_VISITED to mark whether received SETUP messages.

After received the message, node compared its own Cluster ID. If equaled, continue the following operation, else abandon the message.

Because the node still can be the leader of bottom level cluster, after replying the ACK messages, we repeat step (b) until there is no unvisited node in the network.

Ramanathan, et al. (2000) .

They present an efficient randomized algorithm for leader election in large-scale distributed systems. They proposed that algorithm is optimal in message complexity ( $O(n)$  for a set of  $n$  total processes), has round complexity logarithmic in the number of processes in the system, and provides high probabilistic guarantees on the election of a unique leader. The algorithm relies on balls and bins abstraction and works in two phases. The main novelty of the work is in the first phase where the number of contending processes is reduced in a controlled manner.

The main contributions of their work can be summarized as follows:

A randomized leader election algorithm that is optimal in the number of messages in the election phase  $O(n)$ , has round complexity logarithmic in the number of processes in the system  $O(\log n)$ , and elects a unique leader .

An approach in which the lack of global information is intelligently leveraged to prune the number of processes participating in the leader election algorithm.

An asynchronous version of the first phase of the algorithm and a partially synchronous version of the second phase so that the algorithm can be effectively Realized for general distributed applications.

A rigorous analysis to prove the correctness and the complexity of the algorithm. This paper presents the design of an efficient randomized algorithm for leader election in large-scale distributed systems. The algorithm guarantees correctness with high probability and has optimal message complexity  $O(n)$ . To our knowledge, this is the first result providing high probabilistic guarantees with optimal message complexity for a general topology. It proposes variants of the algorithm for synchronous as well as asynchronous environments. It gives an analysis for the correctness of the algorithm and bounds on the number of messages and the number of rounds. Acknowledgments.

Zargarnata j. (2007).

He Presented a new method based on electing a leader and assistant. If the leader crash, the assistant takes, care of the leader's responsibilities. The result revealed that often, after a leader crash, leader assistant elect as a leader and continues to work. This is important when the scale of network increases the important of this paper is the novel solution to leader election. And here the method uses an assistant for a leader and when there is a leader crash, the assistant coordinate all nodes immediately. This method causes that always the leader and assistant represent best performance to other nodes. The method of this paper is, when the number of nodes increases the performance increases. This method has the following characteristics: While the existing

algorithm elect new leader, after leader's crash, the proposed method uses assistant immediately. While the existing algorithms have developed with the specific network topology, the proposed method can apply to any network topologies and while the pre-election method elects the provisional leader that doesn't have enough efficiency, the proposed method elects the assistant that is more efficient for all nodes except the leader.

Ando et al. (2008).

They presented two new randomized leader election protocols in  $n$ -station RN with no knowledge of  $n$ . The expected  $O(\log(n))$  time complexity of both Algorithms in the paper achieves a quasi-optimality (up to a constant factor) with each station keeping awake for  $O(\log(n))$  time slots in both algorithms. The contribution of this paper is to propose a class of energy efficient and quasi-optimal leader election protocols. In this paper, It present two new randomized leader election protocols in  $n$ -station RN with no knowledge of  $n$ . The expected  $O(\log(n))$  time complexity of both algorithms achieves a quasi-optimality (up to a constant factor), with each station keeping awake for  $O(\log(n))$  time slots in both algorithms. Finally, It gives a lower bound  $\Omega(\log n)$  on the space complexity, that is, it shows that it is impossible to construct a leader election algorithm if only  $\log n$  bits are available for a processor. Its important to know that, For asynchronous, anonymous network in which processors communicate with each other by message passing, we showed an upper bound  $O(n \log d)$  and a lower bound  $\Omega(\log n)$  on the space complexity. These bounds are for the leader election algorithm that solves arbitrary  $n$ .

The space complexity  $\Omega(\log n)$  arises because the algorithm needs to be applicable for any  $n$ . We showed how to construct an algorithm that uses only one bit local memory when  $n$  is fixed.

Lavault, et al. (2003).

They designed and analyzed two distributed leader election protocols in radio networks (RNs) where the number  $n$  of radio stations is unknown. Where the first algorithm runs under the assumption of Limited collision direction, while the second assumes the other one assumed that no collision detection is available. By "limited collision detection, which means that exactly one station sends (broadcast a message. then all stations (including the transmission that are listening at this moment receive the sent message. By contrast, the second no-collision detection algorithm assumes that a station cannot simultaneously send and listened signals. Moreover both protocols allow the stations to keep asleep as long as possible, thus minimizing their awake time slots (such algorithms are called energy efficient). Both randomized protocols in RN are shown to elect a leader in  $O(\log \log (n))$  time slot.

Mirakhorli, et al. (2007).

They proposed algorithm in which each node does not need knowing any structure such as ring algorithm; also, it does not need knowing higher ID nodes. It is based on Introducing several nodes as candidate nodes that reduces the number of required messages. The complexity of proposed algorithm is smaller than bully algorithm and competes with ring algorithm. Also, in ring algorithm, all of the nodes know a logical ring structure between nodes that is drawback of ring algorithms then it simulated bully and

ring and our algorithm with computer and the results is shown in Table 1. The crash probability of each node is 0.01 and the number of candidate nodes is 5.

Table 1: Experiments of computer simulation

N	Bully	Ring	paper Algorithm
100	289	200	101
1000	18,204	2,000	1,000
10,000	1,705,560	20,000	10,000
100,000	169,499,620	200,000 1	100,002

Experiments show that thier algorithm needs small messages rather than two other algorithms. When N is large, the bully algorithm needs very large number of required messages. The ring algorithm is better than Bully algorithm but its number of messages is two times of our algorithm approximately.

Shirali, et al. (2008).

They provided a review on LEAs and designing issues. Now depending on your network condition, you can design the best architecture. For example now you know that in the small networks, bully algorithms are the best one. However if you have a ring network, you choose the ring algorithm and if your network is vast and has a lots of nodes, considering number of collisions and depending on the conditions like position of nodes, the distance between them and duple Xing of the links you can use one of the above algorithms or a combination of them. Below Sun Algorithm Idea:

Number of messages =  $(m+n) + m + (m+n) \Rightarrow O(m+n)$

Latency =  $4(m) + 2(n/m) \Rightarrow O(m + (n/m))$

Kutyłowski<sup>1</sup>, et al. (2003).

They present an approach that yields a randomized LEAs for a single-hop no-CD radio network. The algorithm has time complexity  $O(\log^3 M)$  and energy cost  $O(\log M)$ . This is worse than the best algorithms constructed so far ( $O(\log M)$  time and  $O(\log M)$  energy cost), but succeeds in presence of an adversary with energy cost  $\epsilon(\log M)$  with probability  $2^{-\epsilon(\log M)}$ . (The  $O(\log M)$  energy cost algorithm can be attacked by an adversary with energy cost  $O(1)$ ). The algorithm offers some additional features – it yields a group of  $\epsilon(\log M)$  active stations which know each other. This can turn out to be useful for replacing a leader that gets faulty or exhausts its energy resources. Certainly, the solution presented is only a small step towards a secure network self organization.

Svensson, and Thomas, (2005).

This paper introduces a new implementation of a leader election algorithm used in the generic leader behavior known as generate leader. The first open source release of the generic leader contains a few errors. The new implementation is based on a different algorithm, which has been adopted to fulfill the existing requirements. The testing techniques used to identify the errors in the first implementation have also been used to check the implementation it proposes here. The paper even extended the amount of testing and used an additional new testing technique to increase our confidence in the implementation of this very tricky algorithm. The new implementation passed all tests successfully. Then the paper describes the algorithm and discusses the testing

techniques used during the implementation. In this case they are interested in a solution that is fault-tolerant with respect to failing and restarting processes and failing and restarting nodes. We assume Erlang nodes to have reliable communication without lost messages (basically the TCP/IP setting in which all nodes can directly communicate with all other nodes in a reliable way). In the open source Erlang community presented an implementation of a leader election algorithm. This implementation is based on an article written by Singh but contains numerous adaptations to the Erlang setting. The implementation originates from the work at Ericsson with the AXD 301 telecommunication switch, but has been rewritten and turned into the OTP behavior `gen_leader`. From a user point of view, the generic leader behaves like a generic server with callback functions like `call` and `cast`. The intended use is that of having one generic leader per node and clients access only the generic leader on their node. The generic leaders communicate with each other and forward all requests to the chosen leader. In some rare circumstances, two leaders can be elected at the same time. In addition, there is a possibility that the election of a new leader stands in a deadlock. The system may run for years without showing any failure, but there is always the potential danger that one day the circumstances are exactly such that those faults occur. After failing to repair the implementation they proceeded to make a new implementation based on another algorithm. The new implementation is based on the article 'Leader Election in Distributed Systems with Crash Failures' by Stoller. Compared with Singh, Stoller takes a slightly different approach to the leader election problem, which seems to fit better into the Erlang setting.

However, they still had to modify the algorithm, since it was designed for a completely different situation. they took care to supply the same interface for this new implementation as defined for the original, incorrect, implementation. However, due to the differences in the implemented algorithms the interface functions that return all alive nodes and the one returning all dead nodes, could not be provided. Apart from that the behavior of the new implementation should be, when viewed from the outside, the same as the behavior of the old implementation. Except for the failures, they have tested the implementation thoroughly, using both the test method with abstract traces that revealed the errors in the original implementation.

Bagchi, and Das, 2005.

This paper proposes a round-2 randomized algorithm to elect a leader and co-leader of the server group without assuming any particular network topology. The algorithm is implemented and the network-paging latency values of wireless network are measured experimentally. Results indicate that in most cases the algorithm successfully terminates in first round. The network-paging latency values indicate that MDVM system is realizable using 3G/4G wireless communication systems. In addition, the overall message complexity of the algorithm is  $O(|Na|)$ , where  $Na$  is the size of the server-group. network-paging latency values are experimentally evaluated for LAN, Wireless VPN and 2.5G GPRS systems connecting a mobile client and a server in SG, where SG is a component of the MDVM system architecture. The proposed algorithm assigns the node IDs to the MDVM servers online and does not rely on the static pre-assigned node IDs. The algorithm is implemented in the distributed system setup.



The experimental results illustrate that the algorithm terminates in the first round of election phase in majority cases by electing a unique leader and co-leader of the SG. The round ratio values decrease with the increase in the number of MDVM servers. In addition, the number of randomly chosen servers entering in the second round of election phase is substantially reduced as compared to the initial set of servers. The experimental results illustrate that the filter ratio values decrease significantly with the increase in the size of SG. The proposed algorithm is realizable, free from any assumption regarding static node IDs, network topology and buffer space limitations. The experimental values of network-paging latencies between the mobile-clients and servers of the SG indicate that MDVM system comprised of SG is realizable using 3G/4G mobile communication technologies. The overall message complexity of the algorithm is  $O(|Na|)$  where,  $|Na|$  is the total number of MDVM servers in a SG.

Derhab and Badache, (2008).

They proposed a self-stabilizing leader election algorithm that can tolerate multiple concurrent topological changes. By introducing the time-interval-based computation concept, the algorithm ensures that a network partition can within a finite time converge to a legitimate state even if topological changes occur during the convergence time. Our simulation results show that their algorithm can ensure that each node has a leader over 99 percent of the time. We also give an upper bound on the frequency at which network components merge to guarantee the convergence. The algorithm can converge to a legitimate state even in the presence of

topological changes during convergence time. By defining concurrent and disjoint computations and their corresponding intervals, an older reference level encompasses any new one belonging to its equivalence class. In the same way, an older DAG propagation encompasses new ones. Simulation results show that the proposed self stabilizing leader election algorithm experiences very optimal results in terms of the fraction of stabilization time, convergence time, and message overhead compared to Malpani's algorithm. It provided a novel observation about self-stabilization by defining the unstable period by the time period that begins when a component C that contains a node i enters an illegitimate state and terminates when this particular component is again in a legitimate state. By determining the frequency at which merging can occur, it has shown that the unstable period is bounded.

Castillo, et al. (2007).

This paper proposed A modified algorithm Compared with The original algorithm, introduced by Villadangos which, had the aim of reducing the number of exchanged messages in order to select a leader. However, the original  $O(n)$  algorithm fails to choose a leader on several occasions. A modified algorithm, which eliminates the problems that cause the wrong behavior, is proposed here. It is formally proved that the new algorithm verifies the correctness criteria that consist of selecting a unique leader in every case. Additionally, the algorithm maintains the  $O(n)$  complexity in both messages and time, where n is the number of nodes in the system. Conclusions of this paper are to a modified leader election algorithm to choose a leader in a complete network. Then illustrated through various examples how the algorithm presented by Villadangos failed to work correctly in some situations.

Ingram, et al (2009).

They presented algorithm for electing a leader in an asynchronous network with dynamically changing communication topology. The algorithm ensures that, no matter what pattern of topology changes occur, if topology changes cease, then eventually every connected component contains a unique leader. The algorithm combines ideas from the Temporally Ordered Routing Algorithm (TORA) for mobile ad hoc networks with a wave algorithm all within the framework of a height-based mechanism for reversing the logical direction of communication links. It is proved that in certain well-behaved situations, a new leader is not elected unnecessarily. We assume a system consisting of a set  $P$  of computing nodes and a set  $L$  of bidirectional communication links between nodes.  $L$  consists of one link for each unordered pair of nodes, i.e., every possible link is represented. The nodes are assumed to be completely reliable. The links between nodes go up and down, due to the movement of the nodes. While a link is up, the communication across it is in first-in-first-out order and is reliable but asynchronous. They model the whole system as a set of (infinite) state machines that interact through shared events. Each node and each link is modeled as a separate state machine. The shared events are Link Up/Down notifications and receipt of messages, all of which are controlled and initiated by the link and responded to by the node. The sending of a message is also a shared event, but it is controlled and initiated by the node and responded to by the link.

Alvarado-Magana and Fernandez-Zepeda, 2007.

This paper presents an algorithm to modify the algorithm of Xu and Srimani (Self-Stabilizing Anonymous Leader Election in a Tree) that finds a leader in a tree graph. The worst case execution time for this algorithm is  $O(N^4)$ , where  $N$  is the number of nodes in the tree.

Svante Janson, Christian Lavault, Guy Louchard, (2008)

They presented a probabilistic analysis of an algorithm under some conditions on the probability distributions  $P(n, k)$ , including stochastic monotonicity and the assumption that roughly a fixed proportion  $\alpha$  start with a set of  $n$  players. With some probability  $P(n, k)$ , suggest to kill  $n-k$  players; the other ones stay alive, and we repeat with them. What is the distribution of the number  $X_n$  of phases (or rounds) before getting only one player including stochastic monotonicity and the assumption that roughly a fixed proportion  $\alpha$  of the players survive in each round. It proves a kind of convergence in distribution for  $X_n - \log_{1/\alpha} n$ ; as in many other similar problems there are oscillations and no true limit distribution, but suitable subsequences converge, and there is an absolutely continuous random variable  $Z$  such that, The paper generalizes the convergence theorem by considering a general leader election algorithm in the paper. Note that the mean number of needed messages is asymptotically  $2n \log_3(n)$ , as we use  $2n$  messages per round.

Ali, et al, (2009)

The proposed algorithm that relies on collecting and re-distributing information amongst local nodes in order to find the leader. It is based on the assumption that if this process is repeated sufficiently then the algorithm will converge towards a unique leader. It is shown that the proposed mechanism outperforms existing algorithms in terms of time complexity and response to node mobility.

The algorithm was simulated for Bluetooth ad hoc networks, which, by default, rely on a master/slave architecture, however, the cooperative approach could be adapted to any network that exhibits the master/slave configuration such as clustered ad hoc networks or Zig Bee-based sensor networks. This paper presented a novel leader election algorithm for master/slave mobile ad hoc networks and showed that it scales well with network size and node mobility. The algorithm exploits knowledge of the underlying network, which is obtained through a random tour, in order to estimate the number of rounds a master needs to repeat the election procedure for a leader to be elected. When compared to Tree-based algorithms, it was shown that the cooperative approach performs better in terms of convergence time as well as the response to node mobility, which makes it more attractive for ad hoc settings that exhibit spontaneous changes in topology. Furthermore, the cooperative election approach may be applied to other types of ad hoc networks without any major modifications. This will be investigated in the future.

Heutelbeck & Hemmje, (2009).

This paper uses a peer-to-peer data structure, the so-called distributed space partitioning tree (DSPT). A DSPT is a general use peer-to-peer data structure, similar to distributed hash tables (DHTs), that allows publishing, updating of, and searching for dynamic sets. In this paper we present an efficient distributed leader election algorithm that can be used in DSPTs to eliminate redundant network traffic DSPTs are relevant for many interesting application in mobile computing,. this paper described the algorithm IIS to solve the redundant reply problem in DSPTs. As the algorithm makes no assumptions on the way the search space is partitioned by the

DSPT, we expect that it will be possible to apply the algorithm in new future DSPT implementations, going beyond the DSPT realization by the authors of this paper. The algorithm presented here solves the redundant reply problem without introducing new communication costs to the DSPT. We also conducted some experiments that indicate that the heuristic used in IIS significantly improves the running time of the algorithm for real-world location data.

## **Chapter 4**

### **leader election algorithms**

#### **4.1 Introduction**

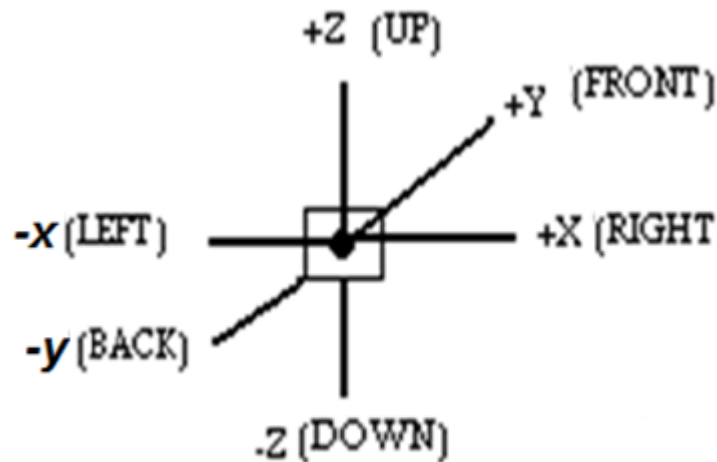
The LEA is a program distributed over all nodes. As discussed in chapter one and two, It starts when the leader failure is detected by one process at a simple case or by all processes at the worst case. It terminates when all processes are aware of the new elected leader (Singh G., (1996). The function of LEA is to move the system from an initial state, where all nodes are in the same computation state, to a new state in which one of these nodes is a leader or a coordinator (Tanenbaum, A.,2002). The current LEAs for the 3D torus networks did not solve the leader failure problem with the existence of link failure. This dissertation presents a new solution for leader election problem in the 3D torus with the presence of links failure to enhance previous algorithms. So, the proposed solutions will take in consideration, the existence of links failure along with leader failure problem.

This research aims to present two new algorithms. The first one solves the leader failure problem in three dimensional torus networks despite existence of one link failure, while the second one solves the problem despite the presence of two links failure. Before algorithms discussion, this chapter starts with model description in the next section.

#### **4.2 Model Description and Assumptions.**

3D torus network is not different from the 3D mesh except in the connections between the first and the last nodes in each dimension. These connections make all nodes connected with six neighbors

(X, Y, Z) which are (right, left, front, back, up and down). Figure -17 shows the directions in the proposed topology and Figure -18 shows a three dimensional torus network with (3 X 3 X 3) dimensions.



X, Y, Z Directions in 3D torus networks

Figure 17: directions in 3D torus

**This research assumes the following:**

- 1- The nodes regularly form an X, Y, Z three Dimensional torus network as shown in Figures 17 and 18.
- 2- The network contains N nodes numbered from 0 to N-1.
- 3- The position of nodes designated by (X,Y,Z), from (0,0,0) to (X-1,Y-1, Z-1).
- 4- Communication is with only one node at a time. Multicast is not implemented.



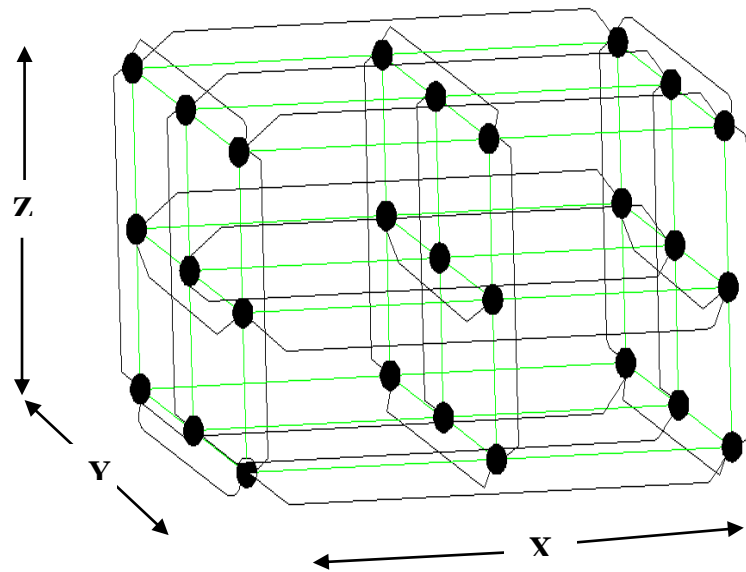


Figure 18: (3 X 3X 3) Torus Networks

5- A node can send or receive simultaneously to and from the same or different nodes.

6- The network uses XYZ -routing: a message is routed within X-axis to then Y-axis then Z-axis that contains the destination node.

7- Leader failure may occur at any time. This failure may be discovered by one node in a simple case, or concurrently by more than one node which reaches at the worst case N-1 nodes.

8- Each node is connected by six links (right, left, front, back, up, down) as shown in Figure-17.

All links are bidirectional.

10 - Leader failure means that node loses the ability of control, but it can send and receive messages.

11- The existence of one link failure is probable in the first algorithm (link failure means that the messages can not pass through the link in both directions). In the second algorithm two links failure are probable.

12- Each node has distinguished  $W_{id}$  used in the election algorithm.

### 4.3 Leader Election Algorithms in 3D torus Networks.

In this section the proposed algorithms will be presented. Before describing the first algorithm, the definition of the following variables will assist to understand the algorithm:

$W_{id}$ : For a given process on processor node  $i$  there is set of attributes such as storage capacity, CPU speed, battery age, ram speed ,..., let us use  $A=\{A_1,A_2,\dots,A_k\}$ , where  $k$  is the number of attributes ,and  $W_{id}$  is the weight value which will be computed from these attributes. To every node in the network, we will use this value to compare  $W_{id}$  for every node with others to elect the leader to be the one with the highest value .

Node State: during the execution of the algorithm, the node will be in one of the following states:

Leader: one node must have this state in a stable network.

Candidate: there is a failure and the election process is in progress inside this node.

Normal: the network is normal and no leader failure is detected by this node.

#### 4.3.1 Leader Election in 3D Torus Networks with one Link Failure

This algorithm is composed of five phases as follows.

Phase One: The algorithm starts by a node that detects leader failures with  $n$  locations. This node changes its state to candidate. It sends (failure messages) through  $X$  axes (right , left) and through  $Y$  axes front , back to inform all neighbors in the same 2D torus about the failure. Each node which receives failure message makes the following:

Node state changes to candidate.

Passes the failure message to the opposite direction through the opposite links depending on the direction from which it has received the message.

Start phase two: selects its  $W_{id}$  as greater  $W_{id}$  , and send election message through links ( $z$  axis up direction) to the node with  $n$  by  $n$ . The election message is composed of (message type, Phase, Step, Greater  $W_{id}$ , and Position of the Greater  $W_{id}$  , Message initiator). If the state is candidate, the received message is ignored.

**Note:** to avoid the probability of link failure in phase one, the proposed algorithm sends failure messages in two directions ( left and right) in X axis and (front and back) in Y axis. Phase one guarantees that all nodes in the 2D level which have the node(s), that detects the failure, are informed about the leader failure. Each node from this level starts phase 2 by sending an election message in the column Z, as it follows in phase 2.

Phase two.

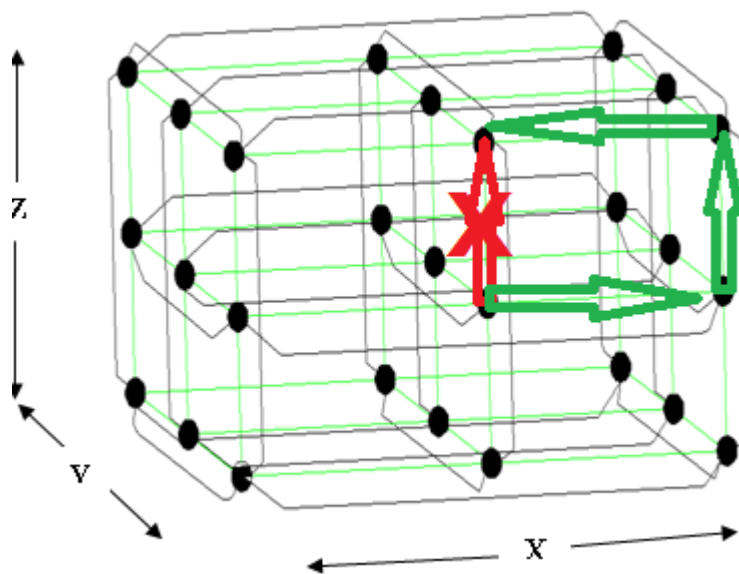


Figure 19: link failure in phase 2

### Candidate

nodes, that have been informed about leader failure in phase one, send election messages through Z -axis (Up). Any node which receives the election message compares its  $W_{id}$  with the receiving  $W_{id}$ , and continue with the greater  $W_{id}$  . When the election message reaches the node that it starts the process (complete ring), it sends the result of election to the first node in the column. The proposed algorithm puts the results of phase two in the first node of each column with  $(Z=0)$  i.e.  $(X,Y,0)$ . Phase two faces two problems, concurrent phase two initialization in the same ring and link failure. To deal with the first problem: any candidate node receives an election message and, its coordinator position is greater than the initiator of the message position, it will ignore the message because it has started the phase so there is no need to repeat the process. If there

is no link failure, the result for the column is found in the node that completes the ring. This node sends the result to the node labeled  $(X, Y, 0)$ .

To solve the second problem, the node that sends an election message, waits for acknowledgment. If the node doesn't receive this message after time out, it detects that there is a link failure. The role of the node that detects the link failure is, to send a link-failure message through the link to its right  $(+x)$ , then the node that receives the link-failure message from left  $(-x)$  forwards it through up  $(+z)$  then left  $(-x)$  and then bypass the link failure. Figure -19 and table 2 shows the solution of link failure in phase two.

Phase Three:

Nodes that finish phase two with  $(y)$  position equal 0, start phase three by sending election messages through  $Y$  axes forward  $+y$ . Any node that receives the election message, compares its  $W_{id}$  with the received  $W_{id}$ , and continues with the greater  $W_{id}$ . If a sender doesn't receive acknowledgment message after time out, it assumes that there is a link failure. To solve this problem, it sends a link-failure message through link Right  $(+x)$  (forward  $+y$ ) then link (left  $-x$  axis (backward) and forwards it left to bypass the link failure. The proposed algorithm puts the results of phase three in the row with  $(X,0,0)$  coordinates. Figure 20 shows phase three steps and link failure solution by detour.

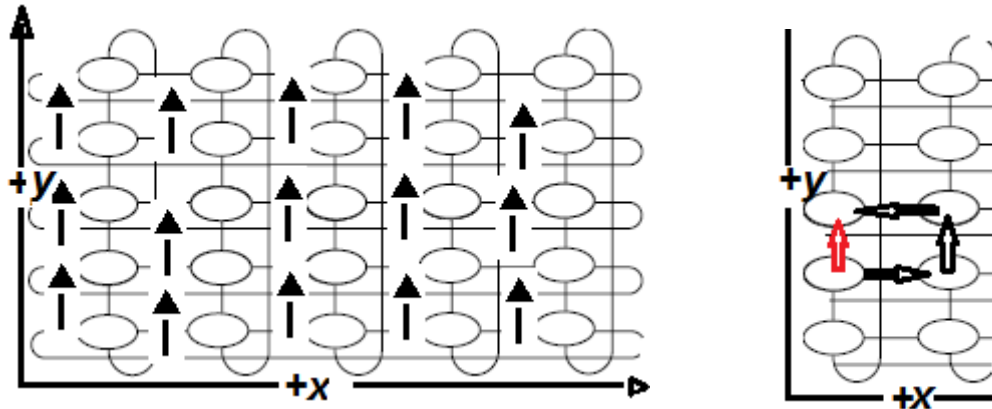


Figure 20: Phase 3 election steps and link failure solution

Phase Four:

After phase three the new leader  $W_{id}$  is available in the row labeled  $(X,0,0)$ . When node  $(0,0,0)$  finishes phase three, it starts phase four by sending an election message through links X-axis(right). Any node that receives a phase four election message from left direction, sends an acknowledgment message. It compares  $W_{ids}$  and sends a phase four election message to the right of  $X$ . If the node doesn't receive the acknowledgment message after time out, it detects that there is a link failure. To solve this problem in this phase, it sends a link-failure message through link y-axis forward  $+y$  then link  $-x$  then  $-y$  to bypass the link failure. Phase four is terminated when the phase four election message is received by node  $(0,0,0)$ . This node starts phase five by broadcasting the result to all nodes. Figure 21 shows phase four election steps and link failure solution

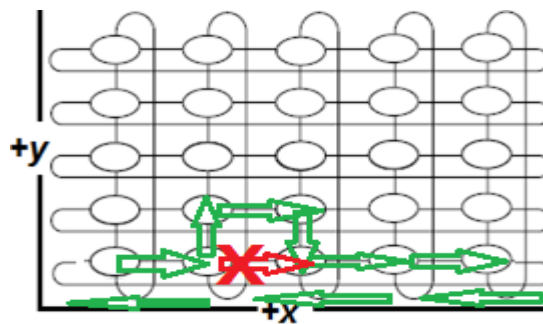


Figure 21: phase four election steps and link failure solution

Phase Five:

At the end of phase four, only one node is aware of the new leader information. This node broadcasts the result as follows:

**X-axis broadcast:** node (0, 0, 0) sends a leader message in two directions through links -X and +X (left, right) to inform all nodes in the x-axis of the new leader information.

**Y-axis broadcast:** node (0, 0, 0) and nodes on X-axis nodes send the leader message to nodes in front and back to inform the Y axis of the new leader information.

**Z-axis broadcast:** all nodes in the 2D (X, Y, 0) inform by sending the leader message through links Up and down to inform Z axis of the new leader information. Each node that receives the Z axis broadcast changes its state to normal and changes its contents information according to the leader message ( Any node aware of the new leader in phase five ignores any new message about election algorithm).

Table 2: Link failure solution by detours

In phase

Z axes

Det #	Link failure Direction	Detour Routing
1	+ Z	+X(RIGHT),+Z(UP),-X(LEFT)
2	+y	+X(RIGHT),+y( FORWARD),-X(LEFT)
3	+x	+Y(FORWORD), +X (RIGHT),-Y(BACKWORD)
4	- Z	+X(RIGHT),-Z(UP),-X(LEFT)
5	-y	+X(RIGHT),-y( FORWARD),-X(LEFT)
6	-x	+Y(FORWORD), -X (RIGHT),-Y(BACKWORD)

send the leader message in two directions, to tolerate the probability of one link failure as shown in Figure 22. The broadcasting of leader message is done through X axes then Y axes then Z axes.

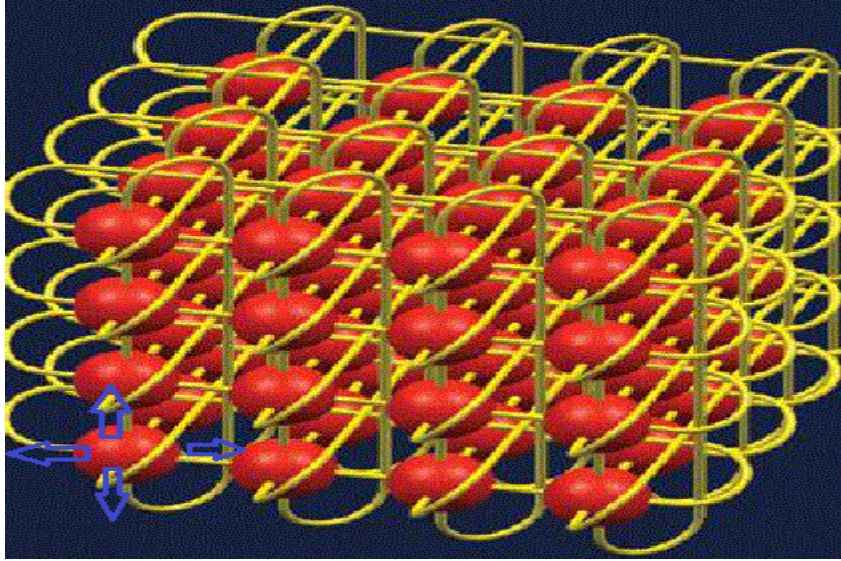


Figure 22 :Phase five Broadcast leader message through all axes

#### 4.3.2 Leader election In 3D torus networks with Two links failure

Second algorithm deals with the presence of two links failure. The main idea to solve the problem is by using more detours in all directions to pass the messages over the link failure. Thinking of more links failure will be researched to reach best solutions in this dissertation. The second algorithm is designed to solve the probability of two links failure. It is also composed of five phases as in the first one as follow:

Phase one: The algorithm starts by a node(s) that detects leader failures in any locations. This node changes its state to candidate. It sends (failure messages) through X axes right (+X) and through Y axes forward (+Y), to inform all neighbors in the same 2D torus about the leader failure.

Each node which receives leader failure message makes the following:  
Node state changes to candidate.

Passes the failure message to the opposite direction through the opposite links depending on the direction from which it has received the message.

Start phase two: selects its  $W_{id}$  as greater  $W_{id}$ , and send election message through links (z axis up direction).



The election message is composed of (message type, Phase, Step, Greater  $W_{id}$ , and Position of the Greater  $W_{id}$ , Message initiator). If the state is candidate, the received message is ignored.

Note: to solve the probability of links failure in all phases in this algorithm sender wait for acknowledgement message from receiver, then after time out it uses the detour way to bypass the message to the target node. This way is applied even if the second link failure in the detour itself. Detour routing depends on the direction of the missed message as it shown in table 2. Phase one guarantee that all nodes in the 2D level which have the node(s), that detects the failure, are informed about the leader failure. Each node from this level starts phase 2 by sending an election message in the column Z, as it follows in phase 2.

Phase two: Nodes in candidate state continue election process by sending election message to the neighbor up on the +Z axes. If the message is received successfully, The receiver sends acknowledgment messages to the sender and continues to send leader election messages up to the next neighbor. Any node which receives the election message compares its  $W_{id}$  with the receiving  $W_{id}$ , and continues with the greater  $W_{id}$ . When the election message reaches the node that it starts the process, it sends the result of election to the first node in the column. Eventually this phase puts the results of phase two in the first node of each column with ( $Z=0$ ) or ( $x, y, 0$ ). To solve the probability of two links failure in phase two, as in phase one, this algorithm uses detours way, to by pass the message to the target node. This way is applied even if the second link failure in the detour itself. Detour routing depends on the direction of the missed message as it shown in table 2. The links failure in the second phase is explained as follow:

1- If the node that detected link failure in the link is ( up )( +z ), it sends link failure message using detour number 1 from table 2 which uses the following path +X(RIGHT),+Z(UP),-X(LEFT). If the node detects a link failure for the second time in link +X it sends link failure message using detour 3 +Y(FORWARD), +X (RIGHT),-Y(BACKWORD).

By this way the algorithm continues until the message reaches its target  
Figure-23.



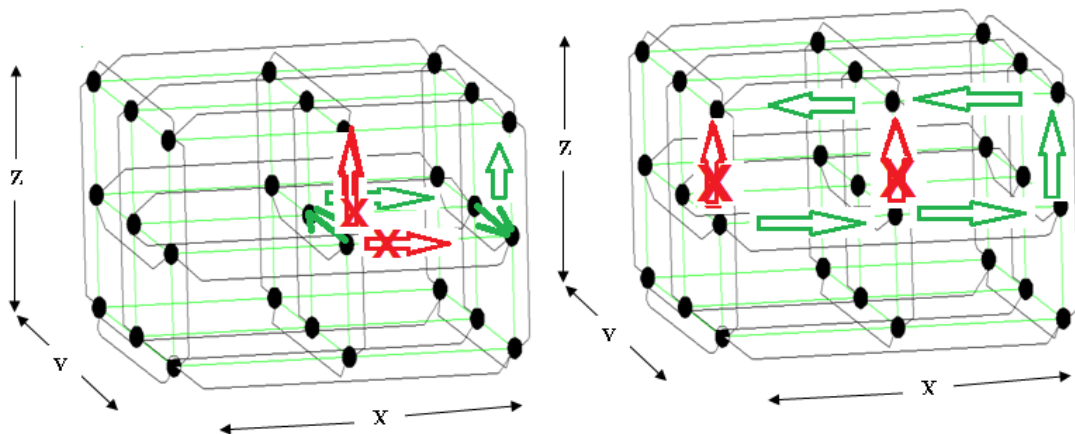


Figure 23: two link failure phase two

Phase-3: Nodes in the 2D torus  $Z = 0$  and  $Y = 0$ , or  $(x, 0, 0)$  start the election in the Y axes, to obtain the result in one row,  $Y = 0, Z=0$  row, which is  $(x, 0, 0)$ . If the message is received successfully, the receiver will send acknowledgment message to the sender and continues to send leader election messages to the next neighbor in the direction of  $+Y$ . This process continues until the message return to the initiator candidate node. The role of these nodes is to wait for phase 4 except node  $(0, 0, 0)$  it starts phase four.

To solve the probability of one or two Links failure in  $+Y$ , there will be an alternative path  $+X$  (RIGHT),  $+Y$  (FORWARD),  $-X$  (LEFT). the node that detect a link failure will send a link failure message to the node that is in the direction  $+X$ , if the node from the right is  $+X$ , it sends acknowledgment to the node that send link failure message to continue in the alternative path  $(+X, +Y, -X)$  in the direction of  $+Y$  if the node doesn't receive the acknowledgment message after time out, it detects that there is a link failure in the direction of  $+Y$ . To solve this problem in this phase, it sends a link-failure message through link to the right on the X-axis, and continue the new alternative path  $(+X, +Y, -X)$  to inform the node in the  $+Y$  direction as in the figure 24 .

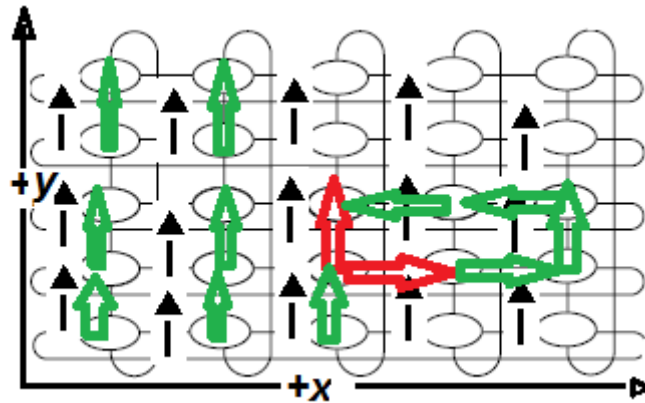


Figure 24: phase two link failure

If the second Link failure is in the direction  $-X$ , there will be alternate path  $(+X, +Y, -X)$  the node that detect a link failure will send a link failure message to the node that is in the direction in  $+X$ , if the node is from the right  $+X$ , send acknowledgment to the node that send link failure the message will continue in the alternate path  $(+X, +Y, -X)$  in the direction of  $+Y$ , then continue to the direction  $-X$  if the node doesn't receive the acknowledgment message after time out, it detects that there is a link failure. In the direction of  $-X$  To solve this problem in this phase, it sends a link-failure message through detour number 6 as in table 2  $+Y$ (FORWORD),  $-X$  (RIGHT),  $-Y$ (BACKWORD), and continue the new alternate path to inform the node in the  $+Y$  direction as in Figure 25

Phase four: Node  $(0, 0, 0)$  start phase 4 by sending election message to its neighbor in  $X$ -axis, to make the election in one row to obtain the result in one node  $X=0, Y=0, Z=0$ .

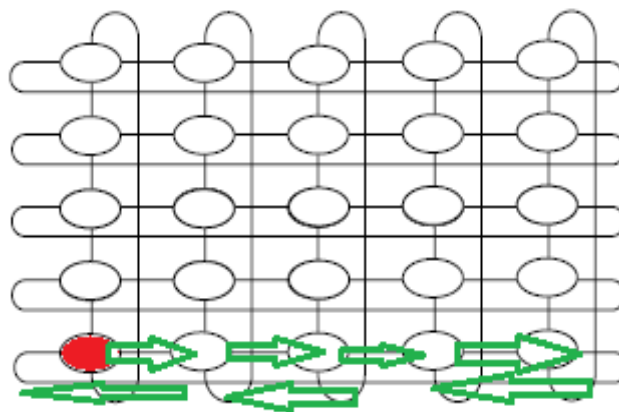


Figure 25 phase four election

**Note:** to avoid the probability of links failure in phase four the algorithm uses detours way as in table 2 for any link failure even if the second link failure is in the detour itself.

This phase has the following states of two links failure:

**State 1- Links failure in directions +X and +Y**

If the sender sends an election message to the right and doesn't receive acknowledgment after time out it detects that there is a link failure in the direction of +X . this node use the detour number 3 from table 2 +Y(FORWORD), +X(RIGHT),-Y(BACKWORD) to bypass the message to the target. At the first step in the detour, if the sender doesn't receive acknowledgment after time out it detects the exist of another link failure in the direction +Y. to solve the problem algorithm use detour number 2 from table 2 +X(RIGHT),+y( FORWARD),-X(LEFT) to bypass the second link failure continue to the target node for the message. Figure 26 explains this case.

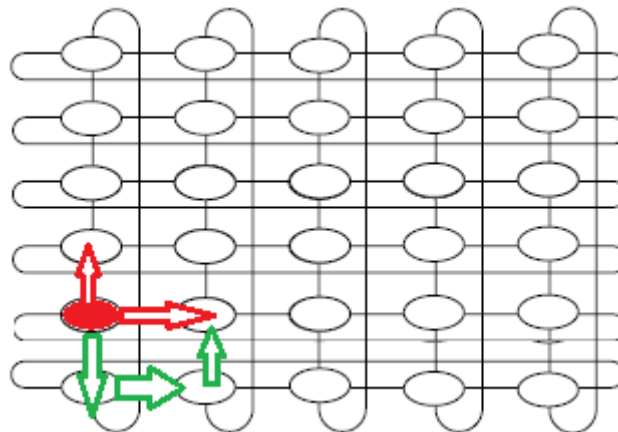


Figure 26: phase four election two link failure +y, +x

**State 2- Links failure in directions +X and +X**

If the sender send an election message to the right and doesn't receive acknowledgment after timeout it detect that there is a link failure in the direction of

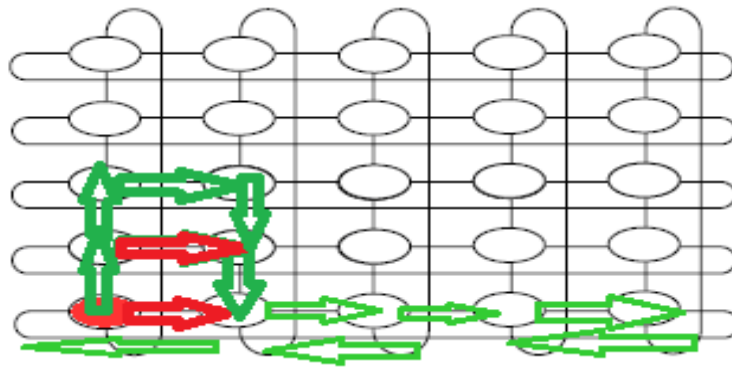


Figure 27: with two link failure +x, +x

**+X. This node uses the detour number 3 from table 2 +Y (FORWORD), +X (RIGHT),-Y (BACKWORD) to bypass the message to the target. At the first step in the detour, the message is successful to reach the node but in the second step in detour. the sender doesn't receive acknowledgment after time out it detects the existence another link failure in the direction +X. to solve the problem algorithm uses detour number 3 from table 2 +Y(FORWORD), +X(RIGHT),-Y(BACKWORD) to bypass the second link failure then continue to the target node for the message. Figure 27 explains this case.**

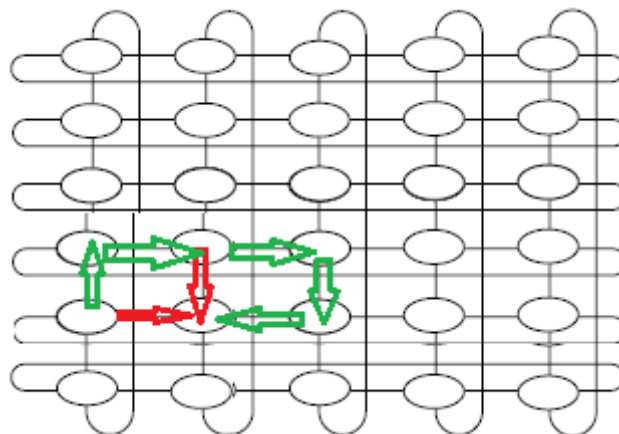


Figure 28: two Links failure in directions +X and -Y

**State 3- Links failure in directions +X and -Y**

**If the sender sends an election message to the right and doesn't receive acknowledgment after timeout it detects that there is a link failure in the**

direction of +X. This node use the detour number 3 from table 2 +Y (FORWORD), +X (RIGHT),-Y (BACKWORD) to bypass the message to the target. At the first and second steps in the detour, the message is successful to reach the node. but in the third step the sender doesn't receive acknowledgment after time is out it detects the existence of another link failure in the direction -Y. To solve the problem, algorithm use detour number 5 from table 2 +X(RIGHT),-y( FORWORD),-X(LEFT) to bypass the second link failure then continues to the target node for the message.

Phase Five:

At the end of phase four, only one node is aware of the new leader information node (0, 0, 0). This node broadcasts leader message in three steps : the first step is to send the leader message in two directions (+,-) X to inform all nodes in the X axis of the new leader information. In the second step each node finish the first step send the leader message in Y axes in two direction (+,- )Y to inform the firs 2D torus about the new leader. Each node in this 2D torus send leader message in Z axes to complete the leader message broadcast. Any node aware of the new leader in phase five ignores any new message about election algorithm. To deal with two links failure, our algorithm use detour way as in table 2 to bypass the links failure as discussed in phase 4.

#### 4.4 State diagram and flow chart.

The algorithm starts when normal state node detects the leader failure Figure 30. The first reaction is to change the state to candidate. Another way for the node in normal state, to become candidate is when inform about the leader failure by any message. The transition from leader state to failure state occurs when the leader node crashes, and loses leader properties. If the failed process returns, the state is changed to normal. Also the candidate state transition has two outcomes: change to leader if the leader ID = Local ID or change to normal if the leader ID is greater than Local ID.

To explain the idea Figure 29 shows that the finite state machine is composed of four states and transitions,(normal, candidate ,failed ,leader ).

Initially a process does not know the ID of the leader, and, consequently it can not decide whether it becomes a leader or not. Once the identity of the leader is known, there are two possible outcomes: the process should become (the new) Leader or not. From the above, we conclude that a process may be in one of the following possible states: Candidate, the start state of the FSM when the node detects the leader failure or receives any message mention the leader failure. In candidate state, the node does not yet know whether it will become a leader or not. The second state is Leader when it actually is a leader. A node is in a normal state, when the network has no failure and the node is not a leader. It is in a failed state when the leader node crashes and loses, the leader properties.

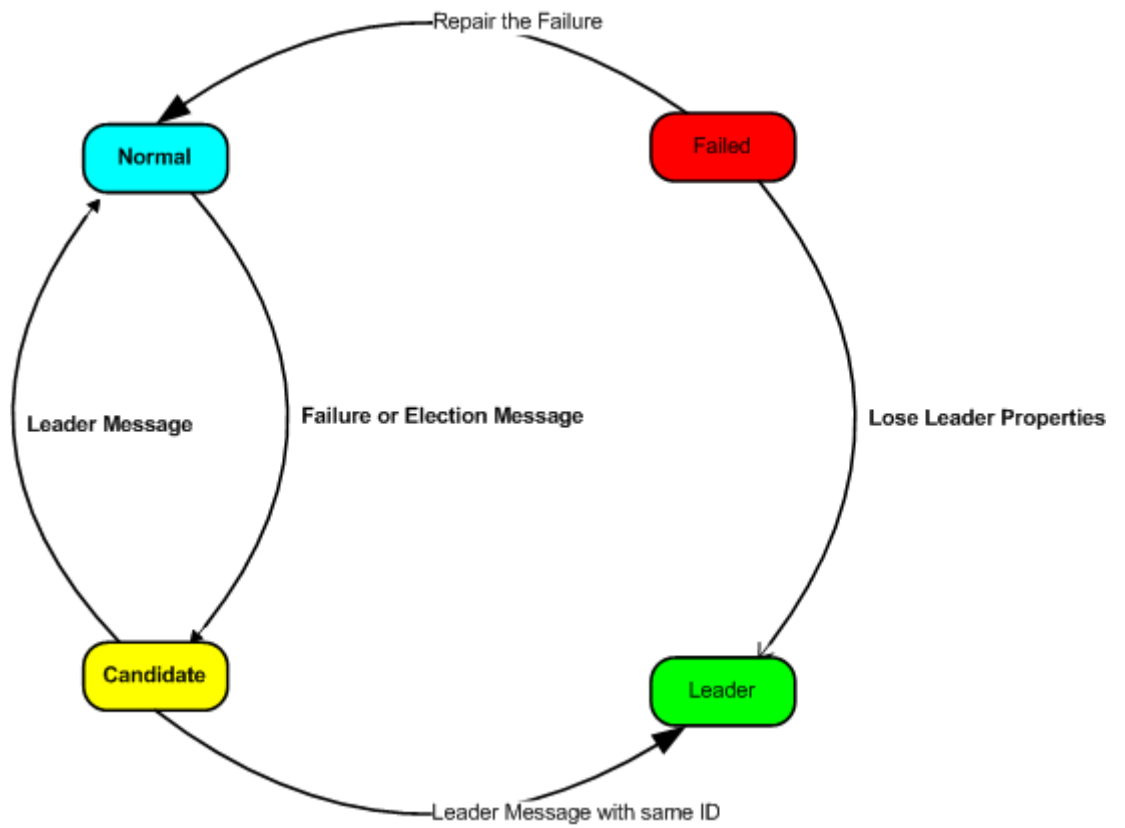
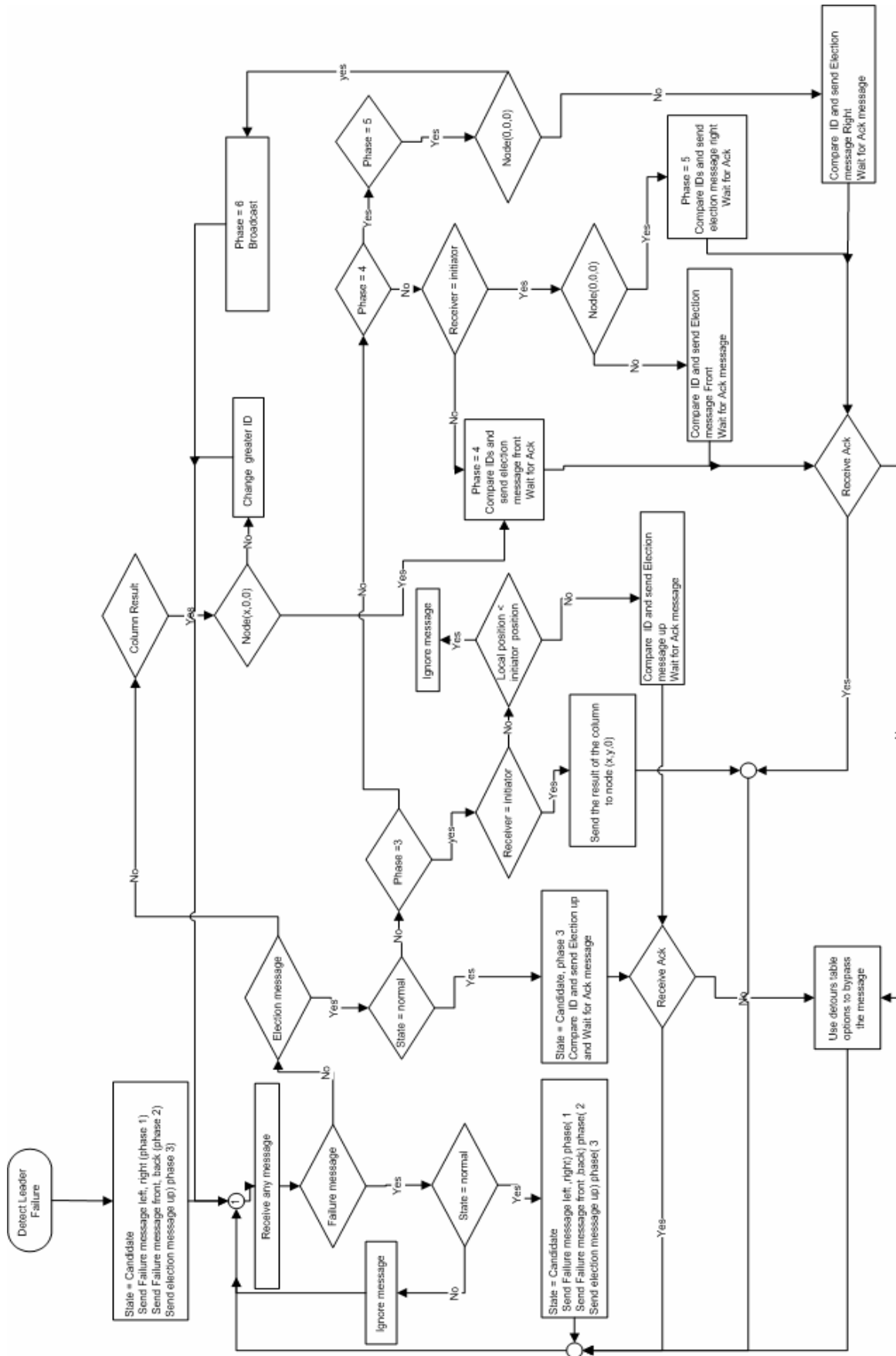


Figure 29 : State diagram for node in LEA

# Flowchart of the solution of the problem





## Chapter 5

### algorithms performance evaluation

#### 5.1 Introduction

This research uses mathematical analyses to evaluate the performance of the proposed algorithm. The evaluation is carried out by computing the number of messages and time steps for each algorithm. The analyses process is carried out for two cases. The first case is the simple case, when the failure is detected by one node. While the second case, is when the leader failure is detected by subset of nodes which can reach all nodes in the worst case. In section 5.2, the first proposed algorithm is analyzed. Section 5.3 presents performance evaluation for the second algorithm. Section 5.4 presents simulation proof for the proposed algorithm.

#### 5.2 Leader Election in 3D Torus Networks with One Link Failure Analyses.

In this section the first algorithm is evaluated and its performance is computed mathematically. Section 5.2.1 analyzes performance from the number of messages perspective. On the other hand section 5.2.2 shows the analyses from time steps perspective. In both analyses simple case the leader failure is detected by one node to the worst case when the leader failure is detected by a subset of nodes reached to  $N-1$  nodes are considered.

##### 5.2.1 Number of Messages:

**Theorem(1)** : assume that we have  $N$  number of nodes in three dimensional torus networks. Then, leader election algorithm with the existence of one link failure needs  $O(N)$  messages to complete.

**Proof:** Number of messages is computed for each phase. Then, add the results to get the total number overall the algorithm, proof is for simple case and worst case, as follows:

## 1. Simple Case:

**Phase One:** Each node receives one message except the last two nodes, they receive one more message for each node, when they send the last message before receiving from inverse direction. After inform messages are sent from nodes in line (x,0,0) through Y axis to inform nodes in (x,y,0) So, the number of messages needed to complete phase one is in the following two formula raw message (rawmsg) and 2d message(2dmsg):

$$\text{Rawmsg} = X + 2$$

(1)

$$2\text{dmsg} = X(Y+2)$$

(2)

**Phase Two:** XY candidate nodes start the election by sending election messages through links labeled 2. In each step from one to Z the algorithm needs XY messages. Same number of messages is required for acknowledgments. XY messages are needed to inform the first 2D torus about columns-result. This formulated as in the following formula 3d message(3dmsg):

$$\left[ \sum_{i=0}^{Z-1} 2(XY) \right] + 2(XY) = 2XYZ + 2XY \quad (3)$$

**Phase three:** Nodes (X, y, 0) needs Y election messages through link 4 to start the phase, and waits for acknowledgement. Eventually, the result reaches to nodes in row labeled (0,y,0) after this number of messages:

$$\sum_{i=0}^{X-1} 2Y = 2XY \quad (4)$$

**Phase Four:** Node (0, 0, Z) starts leader election in Z axes, each node, along Z axes, sends election message and receives acknowledgement. The result reaches to node (0, 0, 0) after this number of messages:

$$\sum_{i=0}^{Z-1} 2 = 2Z$$

(5)

**Phase five:** Node (0, 0, 0) starts row broadcast by sending leader messages through links 1 and 4. As shown in phase 1 and 2, X+2 and X(Y+2) messages are needed for row and depth and XY(Z+2) for columns. Number of messages needed to inform the leader message to all nodes is:

$$(X+2) + X(Y+2) + XY(Z+2) = XYZ + 3XY + 3X + 2 \quad (6)$$

To cover the link failure in phases ( three, four, five), the algorithm needs three messages. So, the total number of messages overall the algorithm is computed by add messages in the equations ( 1 to 6) as in Equation 7:

$$\begin{aligned} \text{Total message} &= X + 2 + X(Y+2) + 2(XYZ) + 2XY + 2Z + XYZ + 3XY \\ &+ 3X + 2 + 3 = \\ &= 3XYZ + 6XY + 6X + 2Z + 7 \end{aligned} \quad (7)$$

When  $X=Y=Z = \sqrt[3]{N}$  then  $XYZ = N$ , so the total messages by using N is expressed in Formula 8:

$$\text{Total message} = 3N + 6\sqrt[3]{N^2} + 8\sqrt[3]{N} + 7 = O(N) \text{ messages} \quad (8)$$

### **Worst Case**

**Phase one:** all nodes detect the leader failure simultaneously. To start the algorithm each node sends two messages in links 1 and 4

and sends two messages in links 3 and 6. Phase one is finished after two steps because all nodes state transform to candidate. The number of messages all phase one message (ph1msg) is equal:

$$\text{Pha1msg} = 2XYZ + 2XYZ \text{ messages} \quad (9)$$

**Phase Two:** All nodes start phase three simultaneously by sending election messages through link 2. All nodes also send acknowledgement messages. There for step1 needs 2XYZ messages. Algorithm needs 2XY for each step from 2 to Z. To send the result to the first 2D algorithm needs 2XY. The number of messages needed in this phase is in formula 11:

$$2XYZ + \sum_{I=2}^Z 2XY + 2XY = 4XYZ + 2XY \quad (10)$$

**Phases (3, 4 and 5)** are the same as in the simple case so, , the total number of messages overall the algorithm in the worst case is computed by add messages in formulas (9,10, 4, 5, 6) besides 9 messages to cover the link failure as in formula 12:

$$2XYZ + 2XYZ + 4XYZ + 2XY + 2Z + XYZ + 3XY + 3X + 2 + 9 = 9XYZ + 5XY + 3X + 2Z + 11 \quad (11)$$

When  $X = Y = Z = \sqrt[3]{N}$  the previous equation equals:

$$9N + 5\sqrt[3]{N^2} + 5\sqrt[3]{N} + 11 = O(N) \text{ messages} \quad (12)$$

The results in 8 and 12 proof theorem (1)

### 5.2.2 Time Steps.

**Theorem (2):** Assume that we have N number of nodes in three dimensional torus networks. Then, leader election algorithm with the existence of one link failure needs  $O\sqrt[3]{N}$  time steps to complete.

**Proof:** Number of time steps is computed for each phase. Then add these numbers to get the total number of time steps overall the algorithm. We apply the computations at the simple case and then at the worst case as follow:

## 1. Simple Case.

### Phase One.

Step 1: One node detects leader failure and sends Leader-failure message to the right and left neighbors.

Steps 2 to  $X/2 + 1$ : In each step, nodes that received leader-failure messages forward the message through the inverse link, and sends acknowledgement message. Number of time steps is equal to:  $X/2+1$  steps. The same way but through  $Y$  rows it needs  $Y/2 + 1$  steps, so phase one needs:

$$2d_{\text{messag}} = (X/2+1) + (Y/2 + 1) \quad (13)$$

**Phase Two:** In step one all candidate nodes send election messages to the upper neighbors through links labeled 2 (Up).

Step 2 to step  $Z$ : nodes receive the election messages make the comparison and pass election messages up with the greater ID. After  $Z - 1$  steps the result of the column leader is found in phase three initiator node. These nodes need another step to send column results to nodes with coordinators  $(x,y,0)$ . So the algorithm needs **( $Z+1$ )** steps to complete phase 2 as in Equation 6:

$$1+Z-1+1 = Z+1 \quad (14)$$

**Phase three:** Nodes with coordinators  $(x, Y, 0)$  start election process in step one by sending the greater ID through link 6 (back).

This process continues as follow:

Step 2 to step  $Y$ : Any node that receives the election message, makes the comparison and sends election message with greater ID to the back neighbor. Phase four is terminated when nodes  $(x, 0, 0)$  receive the election message from link 3 (front). Because the election will be in the rows of  $X$ , and the result of every row will be in the  $y$  axes so this phase needs:

$$\text{Colymsg} = Y \text{ steps} \quad (15)$$

**Phase Four:** Node (X, 0, 0) starts election process in step one by sending the greater ID through link 4 (left). This process continues as follow:

Step 2 to step X: Any node that receives the election message, makes the comparison and sends election message with greater ID to the left neighbor. Phase four is terminated when node (0, 0, 0) receives the election message from link 1 (right). This process needs X steps.

To tolerate the probability of the presence of one link failure in phase 3, 4 and 5 the algorithm needs 3 steps as explained in the algorithm description. So the total steps for this phase:

$$\text{Rawxmsg} = X + 3 \text{ steps} \quad (16)$$

**Phase five:** Since node(0,0) has the new leader information and it sends this information in two directions (left and right), the row broadcasting is terminated after X/2 steps and extra step may occur if X is odd. Same idea for depth broadcasting (Y/2+1) steps and column broadcasting (Z/2+1) steps. So, the total number for this phase is:

$$\text{Totalmsg} = X/2 + 1 + Y/2 + 1 + Z/2 + 1 \text{ steps} \quad (17)$$

The total time steps overall the algorithm in simple case is the summation of time steps in (13 to 17) is as in Equation 19:

$$X/2 + 1 + Y/2 + 1 + Z/2 + 1 + Y + X + 3 + X/2 + 1 + Y/2 + 1 + Z/2 + 1 = 2X + 2Y + (3 * Z) / 2 + 9 \text{ Time steps} \quad (18)$$

When  $X = Y = Z = \sqrt[3]{N}$ , the number of time steps can be expressed as in Equation 21:

$$1\frac{1}{2} \sqrt[3]{N} + 7 = O(\sqrt[3]{N}) \quad (19)$$

## 2. Worst case

In the worst case when all nodes detect the leader failure simultaneously, the time steps will be as follow.

**Phase one:** all nodes start the algorithm by sending leader-failure message. All nodes states become candidate after one time step. After one step is terminated and all nodes start phase 2. Therefore, phase one needs two time steps to complete.

**Phase Two:** in step one, all nodes start phase two. In step two, one node in each column continues the election, while all other nodes in the same column stop the process. So, number of time steps in this phase is equal Z, and need one step for column result message. Thus the total for phases 1,2 and 3 is:

$$\text{Totalmsg} = \mathbf{Z+3} \text{ steps} \quad (20)$$

Phases (3, 4 and 5 ) are the same as in the simple case. The total time steps overall the algorithm in worst case is the summation equations (20, 17, 18, and 19) as follow:

$$1 + 1 + Z + 1 + Y + X + 3 + X/2 + 1 + Y/2 + 1 + Z/2 + 1 = \\ \mathbf{3/2X + 3/2Y + 3/2Z + 9} \text{ Time steps} \quad (21)$$

When  $X = Y = Z = \sqrt[3]{N}$ , the number of time steps can be expressed as in Equation 15:

$$\frac{9}{2} \sqrt[3]{N} + 9 = O\sqrt[3]{N} \text{ steps} \quad (22)$$

The results in 21 and 24 proof theorem (2)

### 5.3 Leader Election in 3D Torus Networks with Two Links Failure Analyses.

In this section the second proposed algorithm is evaluated and its performance is computed mathematically as for the first one in the

previous section. Section 5.3.1 analyzes performance in from the number of messages perspective. On the other hand section 5.3.2 shows the analyses from time steps perspective. In both analyses simple case to the worst case when the leader failure is detected by a subset of nodes reached to N-1 nodes are considered.

### 5.3.1 Number of Messages:

**Theorem (3):** assume that we have N number of nodes in three dimensional torus networks. Then, leader election algorithm with the existence of two links failure needs O (N) messages to complete.

**Proof:** Number of messages is computed for each phase. Then, add the results to get the total number overall the algorithm, proof is for simple case and worst case, as follow:

#### 1. Simple Case.

**Phase One:** Each node receives one message except the last two nodes they receive one more message for each node, when they send the last message before receiving from inverse direction  $X + 2$ . After that inform messages are sent from nodes in line  $(x,0,0)$  through Y axis to inform nodes in  $(x,y,0)$ . So the number of messages needed is  $X *(Y+2)$  to recover two links failure the algorithm needs 2 detours or 12 messages ( messages and acknowledgements) and these messages added one time overall the algorithm so it added at the end of this case , therefore the number of messages needed to complete phase one is in the following two formula (Phase1msg):

$$\text{Phase1msg} = (X + 2) + (X (Y + 2)) \quad (23)$$

**Phase Two:**  $X*Y$  candidate nodes start the election by sending election messages through links labeled 2. In each step from one to Z the algorithm needs  $X*Y$  messages. Same number of messages



is required for acknowledgments.  $X*Y$  messages are needed to inform the first 2D torus about columns-result. To recover two links failure algorithm needs 6 more messages but this number is added in phase 1 and we proposed the number of links failure is 2 overall the algorithm. The number of messages needed to complete phase 2 is formulated as in the following formula:

$$\left[ \sum_{i=0}^{Z-1} 2(X * Y) \right] + 2(X * Y) = 2XYZ + 2XY \quad (24)$$

**Phase three:** Nodes  $(X, y, 0)$  needs  $Y$  election messages through link 4 to start the phase, and waits for acknowledgement. Eventually, the result reaches to nodes in row labeled  $(0,y,0)$  after this number of messages:

$$\sum_{i=0}^{X-1} 2Y = 2XY . \quad (25)$$

**Phase Four:** Node  $(0, 0, Z)$  starts leader election in  $Z$  axes, each node, along  $Z$  axes, sends election message and receives acknowledgement. The result reaches to node  $(0, 0, 0)$  after this number of messages:

$$\sum_{i=0}^{Z-1} 2 = 2Z \quad (26)$$

**Phase five:** Node  $(0, 0, 0)$  starts row broadcast by sending leader messages through links 1 and 4. As shown in phase 1 and 2,  $X+2$  and  $X*(Y+2)$  messages are needed for row and depth and  $XY*(Z+2)$  for columns. Number of messages needed to inform the leader message to all nodes is:

$$(X+2) + X*(Y+2) + XY*(Z+2) = XYZ + 3XY + 3X + 2 \quad (27)$$

To cover the links failure in phases (one, two, three, four, five), the algorithm needs six messages. So, the total number of messages

overall the algorithm is computed by add messages in the equations ( 23 to 27) as in Equation 7:

$$\mathbf{X + 2 + X *(Y+2) + 2(XYZ) + 2 XY + 2 Z + XYZ + 3XY + 3X + 2 + 12 = 3XYZ + 6XY + 6X + 2Z + 19} \quad (28)$$

When  $X=Y =Z = \sqrt[3]{N}$  then  $XYZ =N$  , so the total messages by using N is expressed in Formula 8:

$$\mathbf{3N + 6\sqrt[3]{N^2} + 8\sqrt[3]{N} + 19 = O(N)} \text{ messages} \quad (29)$$

### **Worst Case.**

**Phase one:** all nodes detect the leader failure simultaneously. To start the algorithm each node sends two messages in links 1 and 4 and sends two messages in links 3 and 6. Phase one is finished after two steps because all nodes state transform to candidate. The number of messages (phsmg) is equal:

$$\mathbf{Phs1msg= 2XYZ + 2XYZ} \text{ messages} \quad (30)$$

**Phase Two:** All nodes start phase three simultaneously by sending election messages through link 2. All nodes also send acknowledgement messages. There for step1 needs 2XYZ messages. Algorithm needs 2XY for each step from 2 to Z. To send the result to the first2D algorithm needs 2XY.The number of message needed in this phase is in formula 30:

$$2XYZ + \sum_{I=2}^Z 2XY + 2XY = \mathbf{4XYZ + 2XY} \quad (31)$$

**Phases (3, 4 and 5)** are the same as in the simple case so, the total number of messages overall the algorithm in the worst case is computed by adding messages in formulas (30, 31, 25, 26, 27) besides 12 messages to cover the links failure as in formula 12:

$$2XYZ + 2XYZ + 4XYZ + 2XY + 2Z + XYZ + 3XY + 3X + 2 + 12 = \mathbf{9XYZ + 5XY + 3X + 2Z + 14} \quad (32)$$

When  $X= Y = Z =\sqrt[3]{N}$  the previous equation equals:

$$9N + 5\sqrt[3]{N^2} + 5\sqrt[3]{N} + 14 = O(N) \text{ messages} \quad (33)$$

The results in 29 and 33 proof theorem (3)

### 5.3.2 Time Steps.

**Theorem (4):** Assume that we have N number of nodes in three dimensional torus networks. Then, leader election algorithm with the existence of two links failure needs  $O\sqrt[3]{N}$  time steps to complete.

**Proof:** Number of time steps is computed for each phase. Then add these numbers to get the total number of time steps overall the algorithm. We apply the computations at the simple case and then at the worst case as follow:

#### 1. Simple Case.

##### Phase One

Step 1: One node detects leader failure and sends Leader-failure message to the right and left neighbors.

Steps 2 to  $X/2 + 1$ : In each step, nodes that received leader-failure messages forward the message through the inverse link, and sends acknowledgement message. Number of time steps is equal to:  $X/2+1$  step. The same way but through Y rows it needs  $Y/2 + 1$  steps, to deal with two links failure the algorithm needs 6 more messages for all phases in this case which will be added to the total after phase 5. So phase one needs (p1timst):

$$p1timst = (X/2+1) + (Y/2 + 1) \quad (34)$$

**Phase Two:** In step one all candidate nodes send election messages to the upper neighbors through links labeled 2 (Up).

Step 2 to step Z: nodes receive the election messages make the comparison and pass election messages up with the greater ID. After Z -1 steps the result of the column leader is found in phase three initiator node.

These nodes need another step to send column results to nodes with coordinators (x,y,0) . So the algorithm needs **(Z+1)** steps to complete phase 2 as in Equation 6:

$$1+Z-1+1 = Z+1 \quad (35)$$

**Phase three:** Nodes with coordinators (x, Y, 0) start election process in step one by sending the greater ID through link 6 (back). This process continues as follow:

Step 2 to step Y: Any node receive the election message, makes the comparison and sends election message with greater ID to the back neighbor. Phase four is terminated when nodes (x, 0, 0) receive the election message from link 3 (front). This phase needs (p3timst):

$$p3timst = Y \text{ steps} \quad (36)$$

**Phase Four:** Node (X, 0, 0) starts election process in step one by sending the greater ID through link 4 (left). This process continues as follow:

Step 2 to step X: Any node receives the election message, makes the comparison and sends election message with greater ID to the left neighbor. Phase four is terminated when node (0, 0, 0) receive the election message from link 1 (right). This process needs **X** steps.

To tolerate the probability of the presence of one link failure in phase 3, 4 and 5 the algorithm needs **3** steps as explained in the algorithm description. So the total steps for this phase (totp4timst):

$$totp4timst = X+ 3 \text{ steps} \quad (37)$$

**Phase five:** Since node(0,0) has the new leader information and it sends this information in two directions (left and right), the row broadcasting is terminated after X/2 steps and extra step may occur if X is odd. Same idea for depth broadcasting (Y/2+1) steps and

column broadcasting ( $Z/2+1$ ) steps. So, the total number for this phase is:

$$X/2+1+ Y/2+1+ Z/2+1 \text{ steps} \quad (38)$$

The total time steps overall the algorithm in simple case is the summation of time steps in (34 to 38) **besides 6 time steps for two links failure** as in Equation 39:

$$X/2 + 1+Y/2+1+ Z+1+Y +X+3 +X/2+1+ Y/2 + 1 + Z/2 + 1 + 6= \\ 2X+2Y+ (3 *Z) /2 +15 \text{ Time steps} \\ (39)$$

When  $X= Y = Z = \sqrt[3]{N}$  , the number of time steps can be expressed as in Equation 21:

$$1 \frac{1}{2} \sqrt[3]{N} + 13 = O(\sqrt[3]{N}) \quad (40)$$

## 2. Worst case

In the worst case, when all nodes detect the leader failure simultaneously, the time steps will be as follow.

**Phase one:** all nodes start the algorithm by sending leader-failure message. All nodes state become candidate after one time step. After one step is terminated and all nodes start phase 2. Therefore, phase one needs **two** time steps to complete.

**Phase Two:** in step one, all nodes start phase two. In step two, one node in each column continues the election, while all other nodes in the same column stop the process. So, number of time steps in this phase is equal Z, and need one step for column result message. Thus the total for phases 1 and 2 ( totalp1p2) is:  
totalp1p2=**Z+3** steps (41)

Phases (3, 4 and 5 ) are the same as in the simple case.

The total time steps overall the algorithm in worst case is the summation these equations besides 6 time steps to bypass two links failure as follow:

$$1 + 1 + Z + 1 + Y + X + 3 + X/2 + 1 + Y/2 + 1 + Z/2 + 1 + 6 =$$

$$\frac{3}{2}X + \frac{3}{2}Y + \frac{3}{2}Z + 15 \text{ Time steps}$$

(42)

When  $X = Y = Z = \sqrt[3]{N}$ , the number of time steps can be expressed as in Equation 15:

$$\frac{9}{2} \sqrt[3]{N} + 15 = O \sqrt[3]{N} \text{ steps} \quad (43)$$

The results in 43 and 40 proof theorem (4)

#### 5.4. Simulation

To validate the results of the proposed first algorithm, a simulation is designed. This section explains how to execute the simulation. One example and analyses will be done.

##### 5.4.1 Programming Language Used.

Simulation is programmed in Visual basic programming language. The VB6 is an interpreted language using event programming. It has many objects, tools, and activeX that give the programmer the ability to use a user friendly interface.

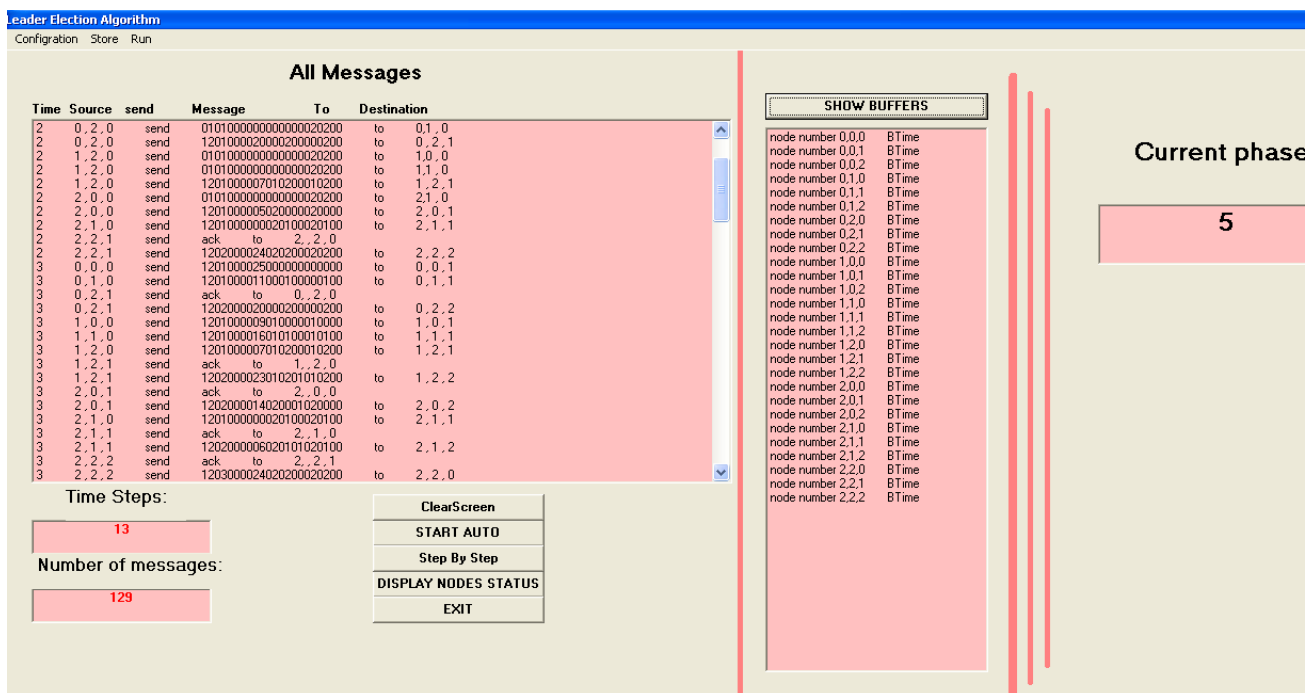
##### 5.4.2 Algorithm Simulation

The torus network is represented in dynamic three dimensional array. Each element in the array represents one node in the network. A node is designed as an object with several properties and methods (Appendix A shows the simulation code). Each node has a buffer to receive messages. The buffer is queue data structure and can store up to six messages. A node is connected to its

neighbors by six links. Distributed processing is applied in the simulation, so all nodes can send and receive simultaneously. The simulation uses coded messages to design the election protocol.

### 5.4.2.1 How to Use This Simulator?

When the simulator starts, a user needs to set up the network configuration. This can be done by selecting configuration from the first menu. Users use the first menu to set up the configuration for the current attempt as shown in Figure-30. It is a must to enter number X such that X represent the number of nodes in X axes and the number Y when Y is the number of nodes in Y axes and number Z for Z axes. These numbers can be changed any time from the setting menu using Configuration sub menu.



The configuration screen as in Figure-31 enables the user to enter the following information for simulation main screen.

Network size: using X , Y and Z dimension.

Link failure: by selecting one node coordinates and the direction for the failed link.

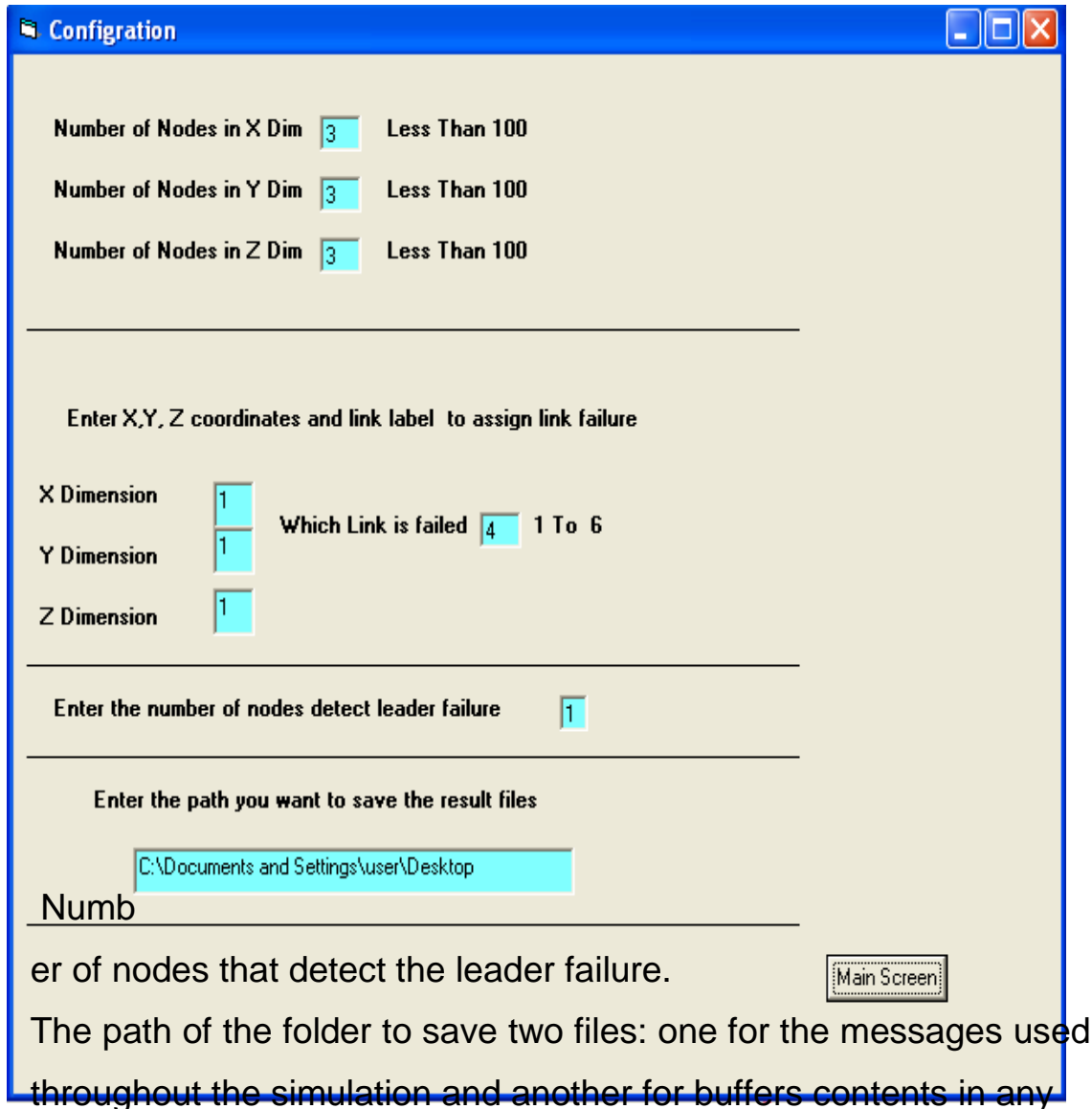


Figure 31: Configurations Screen

Selected step.

The main screen is shown in Figure-30. It shows the simulation environment which can be described as follows:

1. Menus Bar: which contains the following menus:

Setting menu: To establish the configuration for the simulation. Two choices in this menu:

Set configuration: reset or establish the nodes configuration. This is done automatically through the default options, and user can change this default setting.

Exit: To end the simulation.



Store View Menu: To store the messages and node status through the execution of the algorithm.

Run Menu: To start and stop the simulation.

List box: shows for all messages (time, source, message and destination).

Buffers list: shows the contents of all buffers in 3D torus nodes. This list is invisible by default, user can show it when press on buffer command.

Number of messages: shows the number of messages during the simulation and total number of messages as the execution is finished.

Time Steps: counter to compute the current time step which the algorithm reaches and the total time steps when the execution is finished.

Command Buttons: The following commands are found on the main screen:

Start Auto: This Command completes the election algorithm by executing all steps and gives the final results.

Step by Step: This command runs the simulation step by step. Each step shows what happens in one time step.

Display nodes status : When pressed on this command, the status for all nodes is shown in a new screen.

Clear Screen: To clear the main screen .

Exit: To end the simulation and go out from the program.

#### 5.4.2.2. Example.

In this sub section, an example to show the execution of the election algorithm in (3 X 3 X 3) torus network. When the program

starts, a user assigns network configuration as in figure-31. Results and messages as in figure 32. to explain the steps and messages, log file is displayed in Table 2.

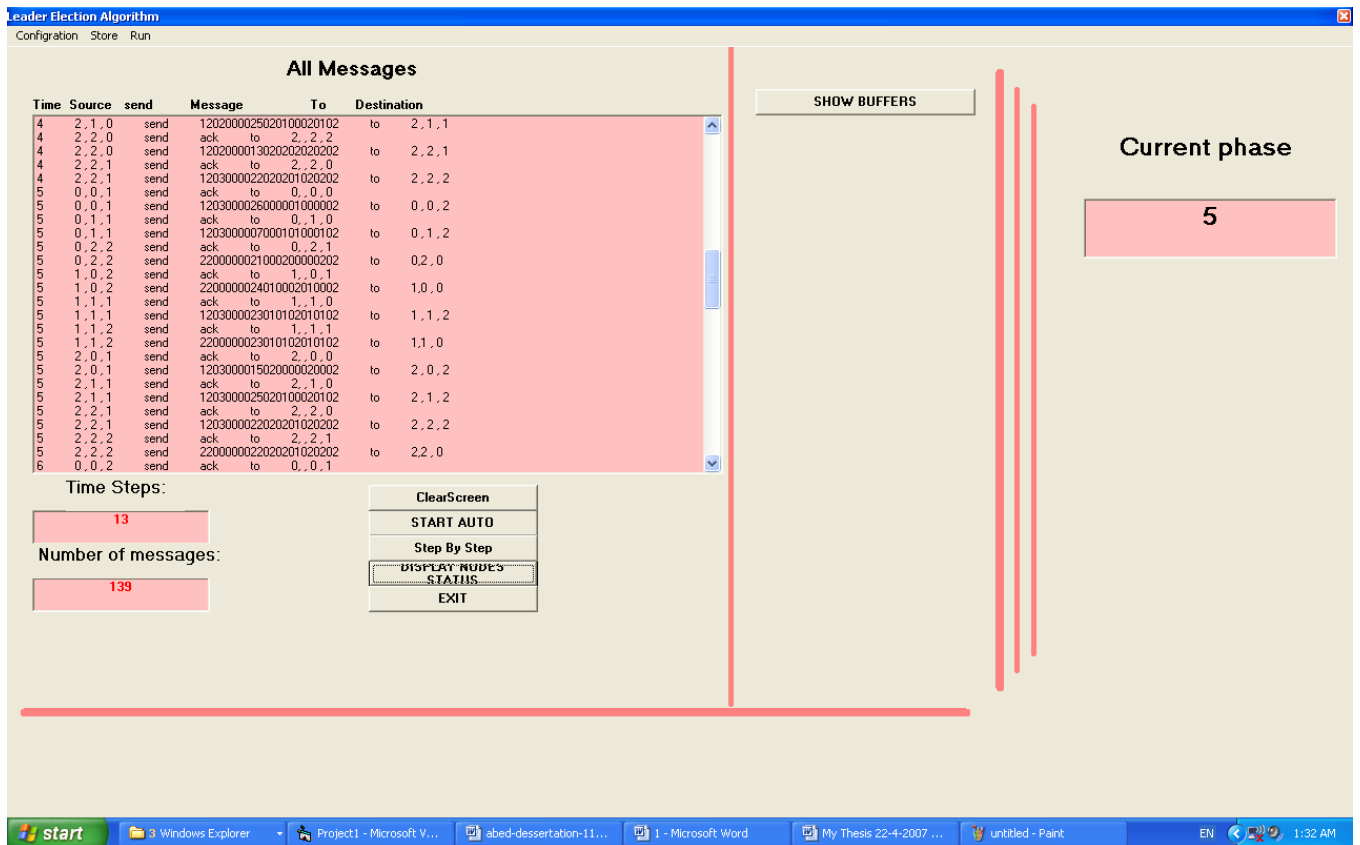


Figure 32: Example 1 execution

Table 1: Messages sent in the example 1 with sources and destinations and steps

Time	Source	send	Message	to	Destination
1 , 2"	1 , 2 , 2	send	010100000000000010202	to	2,2
"1 , 2"	1 , 2 , 2	send	010100000000000010202	to	0,2
"1 , 2"	1 , 2 , 2	send	010100000000000010202	to	1,0

"1 , 2"	1 , 2 , 2	send	010100000000000010202	to	1,1
"1 , 2 , 0"	1 , 2 , 2	send	1201000014010202010202	to	1 , 2 , 0"
"2 , 2"	0 , 2 , 2	send	010100000000000010202	to	2,2 , 2"
"2 , 2"	0 , 2 , 2	send	010100000000000010202	to	0,0 , 2"
"2 , 2"	0 , 2 , 2	send	010100000000000010202	to	0,1 , 2"
"2 , 2 , 0"	0 , 2 , 2	send	1201000002000202000202	to	0 , 2 , 0"
"2 , 2"	1 , 0 , 2	send	010100000000000010202	to	2,0 , 2"
"2 , 2"	1 , 0 , 2	send	010100000000000010202	to	1,1 , 2"
"2 0 , 0"	1 , 0 , 2	send	1201000024010002010002	to	1 , 0 , 0"
"2 , 2"	1 , 1 , 2	send	010100000000000010202	to	2,1 , 2"
"2 1 , 0"	1 , 1 , 2	send	1201000023010102010102	to	1 , 1 , 0"
"2 1 , 2 , 1"	1 , 2 , 0	send	ack to 1 , , 2 , 2"		
"2 2 , 2 , 0"	1 , 2 , 0	send	1202000014010202010202	to	1 , 2 , 1"
"2 0 , 0 , 0"	2 , 2 , 2	send	1201000013020202020202	to	2 , 2 , 0"
"3 0 , 0 , 0"	0 , 0 , 2	send	1201000016000002000002	to	0 , 0 , 0"
"3 1 , 0"	0 , 1 , 2	send	1201000001000102000102	to	0 , 1 , 0"

"3	0, 2, 0	send	ack	to	0, , 2, 2"
"3	0, 2, 0	send	1202000021000200000202	to	0, 2, 1"
"3	1, 0, 0	send	ack	to	1, , 0, 2"
"3	1, 0, 0	send	1202000024010002010002	to	1, 0, 1"
"3	1, 1, 0	send	ack	to	1, , 1, 2"
"3	1, 1, 0	send	1202000023010102010102	to	1, 1, 1"
"3	1, 1, 2	send	1201000023010102010102	to	1, 1, 0"
"3	1, 2, 1	send	ack	to	1, , 2, 0"
"3	1, 2, 1	send	1203000019010201010202	to	1, 2, 2"
"3	2, 0, 2	send	1201000012020002020002	to	2, 0, 0"
"3	2, 1, 2	send	1201000009020102020102	to	2, 1, 0"
"3	2, 2, 0	send	ack	to	2, , 2, 2"
"3	2, 2, 0	send	1202000013020202020202	to	2, 2, 1"
"3	2, 2, 2	send	1201000013020202020202	to	2, 2, 0"
"4	0, 0, 0	send	ack	to	0, , 0, 2"
"4	0, 0, 0	send	1202000016000002000002	to	0, 0, 1"
"4	0, 1, 0	send	ack	to	0, , 1, 2"
"4	0, 1, 0	send	1202000005000100000102	to	0, 1, 1"
"4	0, 2, 1	send	ack	to	0, , 2, 0"
"4	0, 2, 1	send	1203000021000200000202	to	0, 2, 2"

"4	1, 0, 1	send	ack	to	1, , 0, 0"
"4	1, 0, 1	send	1203000024010002010002	to	1, 0, 2"
"4	1, 1, 0	send	ack	to	1, , 1, 2"
"4	1, 1, 0	send	1202000023010102010102	to	1, 1, 1"
"4	1, 1, 1	send	ack	to	1, , 1, 0"
"4	1, 1, 1	send	1203000023010102010102	to	1, 1, 2"
"4	1, 2, 2	send	ack	to	1, , 2, 1"
"4	1, 2, 2	send	2200000019010201010202	to	1,2, 0"
"4	2, 0, 0	send	ack	to	2, , 0, 2"
"4	2, 0, 0	send	1202000015020000020002	to	2, 0, 1"
"4	2, 1, 0	send	ack	to	2, , 1, 2"
"4	2, 1, 0	send	1202000025020100020102	to	2, 1, 1"
"4	2, 2, 0	send	ack	to	2, , 2, 2"
"4	2, 2, 0	send	1202000013020202020202	to	2, 2, 1"
"4	2, 2, 1	send	ack	to	2, , 2, 0"
"4	2, 2, 1	send	1203000022020201020202	to	2, 2, 2"
"5	0, 0, 1	send	ack	to	0, , 0, 0"
"5	0, 0, 1	send	1203000026000001000002	to	0, 0, 2"
"5	0, 1, 1	send	ack	to	0, , 1, 0"
"5	0, 1, 1	send	1203000007000101000102	to	0, 1, 2"
"5	0, 2, 2	send	ack	to	0, , 2, 1"

"5 , 0"	0 , 2 , 2	send	2200000021000200000202	to	0,2
"5	1 , 0 , 2	send	ack	to	1 , , 0 , 1"
"5 , 0"	1 , 0 , 2	send	2200000024010002010002	to	1,0
"5	1 , 1 , 1	send	ack	to	1 , , 1 , 0"
"5 1 , 2"	1 , 1 , 1	send	1203000023010102010102	to	1 ,
"5	1 , 1 , 2	send	ack	to	1 , , 1 , 1"
"5 , 0"	1 , 1 , 2	send	2200000023010102010102	to	1,1
"5	2 , 0 , 1	send	ack	to	2 , , 0 , 0"
"5 0 , 2"	2 , 0 , 1	send	1203000015020000020002	to	2 ,
"5	2 , 1 , 1	send	ack	to	2 , , 1 , 0"
"5 1 , 2"	2 , 1 , 1	send	1203000025020100020102	to	2 ,
"5	2 , 2 , 1	send	ack	to	2 , , 2 , 0"
"5 2 , 2"	2 , 2 , 1	send	1203000022020201020202	to	2 ,
"5	2 , 2 , 2	send	ack	to	2 , , 2 , 1"
"5 , 0"	2 , 2 , 2	send	2200000022020201020202	to	2,2
"6	0 , 0 , 2	send	ack	to	0 , , 0 , 1"
"6 , 0"	0 , 0 , 2	send	2200000026000001000002	to	0,0
"6	0 , 1 , 2	send	ack	to	0 , , 1 , 1"
"6 , 0"	0 , 1 , 2	send	2200000007000101000102	to	0,1
"6 , 0"	0 , 2 , 0	send	3201000021000200000202	to	1,2

"6	1 , 1 , 2	send	ack	to	1 , , 1 , 1"	
"6	1 , 1 , 2	send	2200000023010102010102	to	1,1	, 0"
"6	2 , 0 , 2	send	ack	to	2 , , 0 , 1"	
"6	2 , 0 , 2	send	2200000015020000020002	to	2,0	, 0"
"6	2 , 1 , 2	send	ack	to	2 , , 1 , 1"	
"6	2 , 1 , 2	send	2200000025020100020102	to	2,1	, 0"
"6	2 , 2 , 2	send	ack	to	2 , , 2 , 1"	
"6	2 , 2 , 2	send	2200000022020201020202	to	2,2	, 0"
"7	0 , 0 , 0	send	3201000026000001000002	to	1,0	, 0"
"7	0 , 1 , 0	send	3201000007000101000102	to	1,1	, 0"
"7	1 , 2 , 0	send	ack	to	0,2 , 0"	
"7	1 , 2 , 0	send	3201000021000200000202	to	2,2	, 0"
"8	1 , 0 , 0	send	ack	to	0,0 , 0"	
"8	1 , 0 , 0	send	3201000026000001000002	to	2,0	, 0"
"8	1 , 1 , 0	send	ack	to	0,1 , 0"	
"8	1 , 1 , 0	send	3201000023010102000102	to	2,1	, 0"
"8	2 , 2 , 0	send	ack	to	1,2 , 0"	
"8	2 , 2 , 0	send	3201000022020201000202	to	0,2	, 0"
"9	0 , 2 , 0	send	ack	to	2,2 , 0"	
"9	2 , 0 , 0	send	ack	to	1,0 , 0"	
"9	2 , 0 , 0	send	3201000026000001000002	to	0,0	, 0"

"9	2, 1, 0	send	ack	to	1, 1, 0"
"9	2, 1, 0	send	3201000025020100000102	to	0, 1, 0"
"10	0, 0, 0	send	ack	to	2, 0, 0"
"10	0, 0, 0	send	4400000026000001000002	to	0, 1, 0"
"10	0, 0, 0	send	ack	to	0, 2, 0"
"10	0, 0, 0	send	5500000026000001000002	to	1, 0, 0"
"10	0, 0, 0	send	5500000026000001000002	to	2, 0, 0"
"10	0, 0, 0	send	5500000026000001000002	to	1, 0, 0"
"10	0, 0, 0	send	5500000026000001000002	to	2, 0, 0"
"10	0, 0, 0	send	5500000026000001000002	to	0, 1, 0"
"10	0, 0, 0	send	5500000026000001000002	to	0, 2, 0"
"10	0, 0, 0	send	5500000026000001000002	to	0, 0, 1"
"10	0, 0, 0	send	5500000026000001000002	to	0, 0, 2"
"10	0, 1, 0	send	ack	to	2, 1, 0"
"11	0, 0, 1	send	5500000026000001000002	to	1, 0, 1"
"11	0, 0, 1	send	5500000026000001000002	to	2, 0, 1"
"11	0, 0, 1	send	5500000026000001000002	to	0, 1, 1"



"11 , 1"	0 , 0 , 1	send	5500000026000001000002	to	0,2
"11 , 2"	0 , 0 , 2	send	5500000026000001000002	to	1,0
"11 , 2"	0 , 0 , 2	send	5500000026000001000002	to	2,0
"11 , 2"	0 , 0 , 2	send	5500000026000001000002	to	0,1
"11 , 2"	0 , 0 , 2	send	5500000026000001000002	to	0,2
"11	0 , 1 , 0	send	ack	to	0,0 , 0"
"11 , 0"	0 , 1 , 0	send	4400000026000001000002	to	0,2
"11 , 0"	0 , 1 , 0	send	5500000026000001000002	to	1,1
"11 , 0"	0 , 1 , 0	send	5500000026000001000002	to	2,1
"11 , 0"	0 , 2 , 0	send	5500000026000001000002	to	1,2
"11 , 0"	0 , 2 , 0	send	5500000026000001000002	to	2,2
"12 , 1"	0 , 1 , 1	send	5500000026000001000002	to	1,1
"12 , 1"	0 , 1 , 1	send	5500000026000001000002	to	2,1
"12 , 2"	0 , 1 , 2	send	5500000026000001000002	to	1,1
"12 , 2"	0 , 1 , 2	send	5500000026000001000002	to	2,1
"12	0 , 2 , 0	send	ack	to	0,1 , 0"

"12 , 1"	0 , 2 , 1	send	5500000026000001000002	to	1,2
"12 , 1"	0 , 2 , 1	send	5500000026000001000002	to	2,2
"12 , 2 , 0"	0 , 2 , 1	send	5500000026000001000002	to	0,
"12 , 2"	0 , 2 , 2	send	5500000026000001000002	to	1,2
"12 , 2"	0 , 2 , 2	send	5500000026000001000002	to	2,2

nodes status														
normal	leader ID	leader_pos	L_phase	L_step	my pos	my id	C_ID	C_pos	L_F1L_F2L_F3L_F4L_F5L_F6					
True	26	0 0 1	0	0	0 0 0	6	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	0 0 1	26	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	0 0 2	16	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	0 1 0	5	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	0 1 1	7	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	0 1 2	1	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	0 2 0	21	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	0 2 1	20	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	0 2 2	2	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	1 0 0	3	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	1 0 1	17	0	0 0 0	False	True	False	False	False	False
True	26	0 0 1	0	0	1 0 2	24	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	1 1 0	18	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	1 1 1	8	0	0 0 0	False	False	False	True	False	False
True	26	0 0 1	0	0	1 1 2	23	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	1 2 0	0	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	1 2 1	19	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	1 2 2	14	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	2 0 0	15	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	2 0 1	10	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	2 0 2	12	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	2 1 0	25	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	2 1 1	4	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	2 1 2	9	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	2 2 0	11	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	2 2 1	22	0	0 0 0	False	False	False	False	False	False
True	26	0 0 1	0	0	2 2 2	13	0	0 0 0	False	False	False	False	False	False

Main Screen

Figure 33 : Nodes Status After finish the algorithm

### 5.3.2.3 Simulation Survey:

In this section, we explain different scenario for the simulation. For each scenario we

**5.4.2.3** In this sub section we explain different scenarios fore the simulation for each scenario we use different inputs to the simulation. The scenarios uses 3D torus with size equal (3 X 3 X 3) nodes with different node numbers that detect the leader failure:

**State 1:** Table-2 summarizes the inputs and the results when the simulation was executed using torus(3 X 3 X 3). The changes will be in number of node that detects the leader failure and their positions.

The first column lists trial number. Column two contains number of nodes that detect leader failure, column three lists these nodes, and column 4 and 5 contain simulation results ( number of messages and time steps). Three trails were recorded as in Table (2).

Table 2: Simulation inputs and results for different trials applied on torus (3X3)

trial	Nodes detect failure		Messages	time steps
	Number	nodes		
1	1	(2,1,0)	75	14
2	2	(0,1,2),(1,2,1)	81	16
3	3	(0,0,2),(1,2,0),(2,1,1),(1,0,1),(2,0,2)	83	15

Table-2 shows that the number of messages increases as the number of nodes that detect leader failure increases. The number of time steps vibrates through small changes in the number of these nodes and it is almost fixed in the large numbers.

## **Chapter 6**

### **Conclusion and Future Work**

#### **6.1 Introduction.**

This work presents two distributed solutions to the leader failure problem in 3D torus networks. The first solution is a distributed algorithm to elect a new leader in 3D torus networks with minimum number of messages and time steps with the presence of one link failure. The second solution solve more complicated problem when the leader failure is in a network with two links failure. This chapter concludes the dissertation and presents the results for each algorithm. The last section presents future work for researchers in leader election algorithms.

#### **6.2 Results.**

In this work, complexity analysis was used to evaluate performance for the proposed algorithm. The number of messages and time steps are the main factors for performance evaluation. The total number of messages and the total time steps were founded for the proposed algorithm. These totals were translated to big O notation to express the complexity.

##### **6.2.1 Results of the First Proposed Algorithm.**

The first algorithm consists of five phases. Phase one is initiated when one or more nodes detects leader failure. This node(s) informs other nodes in the same row about leader failure to change its state to candidate; nodes in candidate state inform 2D torus about the failure. In phase two, nodes aware of leader failure start election process throughout their columns. In phase three, another election is applied on the 2D torus to obtain the new leader information in one row.

In phase four leader election is applied to this row to get new leader information in one node. Last phase, broadcasts one to all is applied to disseminate the new leader information to all nodes. Proposed algorithm considered the probability of link failure in all phases.

Algorithm performance was evaluated by calculating the number of messages and time steps overall the algorithm. In a network of N nodes connected by a three dimensional torus network (X,Y,Z), the performance is evaluated in simple case, when leader failure is detected by one node and In the worst case, when leader failure is detected by (N-1) nodes. For all cases The number of messages is  $O(N)$  in  $O(\sqrt[3]{N})$  steps.

### 6.2.2 Results of the Second Proposed Algorithm.

The main idea in the second algorithm is how to complete the leader election algorithm despite the existing of two links failure. Second algorithm almost has the same structure as in the first one; it has five phase and same steps in each phase. But the algorithm use detours way to bypass the links failure. This way uses the table-3 which bypass the link failure even if it is in detour itself.

Table 2: Link failure solution by detours

Second algorithm performance was also evaluated by calculating

Det #	Link failure Direction	Detour Routing
1	+ Z	+X(RIGHT),+Z(UP),-X(LEFT)
2	+y	+X(RIGHT),+y( FORWARD),-X(LEFT)
3	+x	+Y(FORWARD), +X (RIGHT),-Y(BACKWORD)
4	- Z	+X(RIGHT),-Z(UP),-X(LEFT)
5	-y	+X(RIGHT),-y( FORWARD),-X(LEFT)
6	-x	+Y(FORWARD), -X (RIGHT),-Y(BACKWORD)

the number of messages and time steps overall the algorithm. The result is the same in complexity as in algorithm one despite the

differences in intermediate steps and messages. For all cases the number of messages is  $O(N)$  in  $O(\sqrt{N})$  steps.

The algorithm is applied by Simulation model which confirm most of mathematical results. Simulation program prove the increasing in messages as the number of nodes that detect leader failure increase.

### **6.3 Future Works.**

Several possible extensions to improve the work proposed here:

- Design an algorithm to solve the leader failure in meshes networks with the presence of link failure.
- Design an algorithm to solve leader failure in hyper mesh Topology, when the ID is not distinguished
- Apply election algorithm meshes and other topologies with wireless communications environment

## References

- Abu-Amara, H., & Lokre, J. (1994). **Election in Asynchronous Complete Networks with Intermittent Link Failures**, IEEE Transactions on Computers, Vol. 34 No. 7, pp. 778-788.
- Afek, Y, & Gafni, E. (1991) **An efficient Technique For End –to-End Communication**. Proceedings of the 11th ACM Symposium on Principles of Distributed Computing.
- Akbar B., Mohammed. E., & Mahdi. E. (2006). **Improvement of Election Algorithm in Distributed Systems Base on Ring Algorithm**, Symposium Proceedings Volume II Computer Science & Electronics Engineering, European University of Lefke, North Cyprus, PP. 497-501,.
- Al Faisal, F., Rahman M., (2009), **Symmetric Tori connected Torus Network**, ,IEEE, Computers and Information Technology, 12th International Conference on
- Ali, R., Lor, S.; Benouaer, R., Rio, M.,(2009), **Cooperative Leader Election Algorithm for Master/Slave Mobile Ad Hoc Networks**, Wireless Days, 2009 2nd IFIP IEEE pp. 1 – 5.
- Alvarado-Magana, J.P.; Fernandez-Zepeda, J.A.,(2007) **Average Execution Time Analysis of a Self-stabilizing Leader Election Algorithm**,parallel and Distributed Processing Symposium, 2007. International , IEEE International, pp.1 – 7
- Andot, E.; Ono, H.; Sadakane, K.; Yamashita, M.(2008)**The Space Complexity of the Leader Election in Anonymous Networks**, IEEE International Symposium on pp.1 – 8.
- Antonoiu, G. and Srimani, K.,( 1996) ," **A Self-Stabilizing Leader Election Algorithm for Tree Graphs**", Journal of Parallel and Distributed ,System vol. 34, Issue 2 pp.227 – 232

Bagchi, S.; Das, P.(2005) **Round-2 Randomized Leader Election algorithm and Latency for MDVM System**, Proceedings of the International Conference on Next Generation Web Services Practices.pages

Barth, D. and Berthomé P.(2004) **(Periodic Gossiping in Commuted Networks**, Theory Comput. Systems , Springer-Verlag ,New York, September 2004, Volume 37, Issue 5, pp. 559-584

Brassard, G., Bratley P. (1996). **"Fundamental of Algorithmic"** , Prentice-Hall Publishers ,USA.

Cámara, J.M.; Moretó, M.; Vallejo, E.; Beivide, R.; Miguel-Alonso, J.; Martínez, C.; Navaridas, J.(2010), **mixed-radix Twisted torus interconnection networks**,ieee,journal, Volume : 21 , Issue: 12 , pp.1765 – 1778.

Castillo M., Fariña F., Córdoba A., and Villadango J. s.(2007) **A modified O(n) leader election algorithm for complete networks**. 15th Euromicro Conference on Parallel, Distributed and Network-Based Processing), Naples, Italy. Conference Proceedings, pp.189-199,ieee conference.

Choo H., Yoo S., Youn H. , ( 2000).**Processor Scheduling and Allocation for 3D Torus Multicomputer Systems**, ,ieee Journals & Magazines,volume 11 issu 5, pp. 475-484.

Conference Publications, pp.174-179.

Coulouris, G. ,Dollimore, J. And kindberg, T.,(2005). **"Distributed Systems Design and Concept**, Addison- Wesley Publisher, USA.

Culler, E. Singh, P. and Gupta, A. (1999) **Parallel Computer Architecture A Hardware/Software Approach**,Amsterdam, Morgan Kaufmann Publisher, Inc,.



Dallys W., And Towels, B.,(2010) **Principles and Practices of Interconnection Networks**, Amsterdam ,Morgan Kaufmann.

David f .robenson, Philip k., mckinley ,(1995)**optimal multicasting communication in whormhole -routing torus network**,ieee ,vol, 6 no 10 October PP. 1029-1042.

Derhab, A.; Badache, N.,(2008),**A Self-Stabilizing Leader Election Algorithm inHighly Dynamic Ad Hoc Mobile Networks** ,ieee transactions on parallel and distributed systems,iee vol. 19, NO. 7, PP. 926 – 939.

Duato, J. Yalamanchili, S. and Ni, L. , (1997) **Interconnection Networks an Engineering Approach**, California,IEEE Computer Society press,

Fredrickson, N. and Lynch , N., (1987). "**Election a Leader in Asynchronous Ring**". Journal of the ACM, Vol.34, PP 98-115.

Gu Q.,(2000),**Interconnection Networks**, available. On line <http://www.cs.sfu.ca/CourseCentral/765/qgu/Notes/note3.pdf>.

Gupta P. ,**Algorithms for routing lookups and packet classiffication**, A dissert submitted to the department of computer and science of Stanford university for the degree of doctor pholosophy december 2000.

Heutelbeck, D.; Hemmje, M.,(2006) **Distributed Leader Election in P2P Systems for Dynamic Sets**, Proceedings of the 7th International Conference on Mobile Data Management ,IEEE, PP. 2-29.

Ingram, R.; Shields, P.; Walter, J.E.; Welch, J.L, (2009),**An Asynchronous Leader Election Algorithm for Dynamic Networks** ,IEEE journal, p 1- 12 Conference Publications.

Janson S., Lavault C. Louchard., G., (2008), **Convergence of Some Leader Election algorithms**, Discrete Mathematics and Theoretical Computer Science journal ,\_vol. 10 PP. 171-196

Kumar, V. , Grama, A. , Gupta, A. and Karypis G., (2003).**Introduction to Parallel Computing**, The Benjamin Community Publishing Company, Inc, Redwood City, California.

Kutyłowski<sup>1</sup> M; and Rutkowski<sup>1</sup> W., (2003)**Adversary Immune Leader Election in Ad Hoc Radio Networks**,springers, Vol. 2832, PP. 397-408.

Lavault, C.; Marckert, J. F.; Ravelomanana, V.,(2003) **Quasi-Optimal Leader Election Algorithms in Radio Networks with Log-logarithmic Awake Time Slots**, IEEE, PP. 1113 - 1119 vol.2.

Min G.,ould-khaoua M.,**Communication Delay in Wormhole-Switched Tori Networks under Bursty Workload**, The Journal of Supercomputing, 26, PP. 77–94, 2003, Kluwer Academic Publishers. Manufactured in the Netherlands.

Mirakhorli, M.; Sharifloo, A.A.; Abbaspour, M, (2007).**A Novel Method for Leader Election Algorithm** ,Computer and Information Technology, 2007. 7th IEEE International Conference on, PP.: 452 – 456.

Mirakhorli, M.; Sharifloo, A.A.; Abbaspour, M,(2007) **A Novel Method for Leader Election Algorithm** , 7th IEEE International Conference on PP. 452 – 456.

Molina G. H, (1982).**Elections in A Distributed Computing systems**, IEEE Transactions on Computers, Vol. 31 Jan 1982, pp. 48-59.

Quinn J., Parallel Computing Theory and Practice, New York McGraw-HILL, 1994.

Ramanathan M., Krishna ., Ferrier R., . Jagannathan S ., Gram A., Szpankowski W.,(2007 **randomize leader election**, Springer-JOURNAL, Volume 19, Numbers 5-6 (2007),P 403-418.

Rouse M.(2006), **mesh-network** available online: <http://searchnetworking.techtarget.com/definition/> .

Rouse.M.,(2007),network topologies ,available online : [searchnetworking.techtarget.com/definition/cross-bar-switch](http://searchnetworking.techtarget.com/definition/cross-bar-switch).

Shirali, M.; Toroghi, A.H.; Vojdani, M.,(2008) **Leader election algorithms: History and novel schemes**,IEEE Volume: 1 , PP. 1001 – 1006.

Singh G, (1996)," **Leader Election in the Presence of Link Failures**, IEEE Transactions on Parallel and Distributed Systems, VOL 7, No 3, pp 76-89.

Singh G, (1996), **Leader Election in the Presence of Link Failures**, IEEE Transactions on Parallel and Distributed Systems, VOL 7, No 3, pp 76-89.

Svensson H., and Arts T., (2005), **A New Leader Election Implementation**, Erlang'05 September 25, 2005, Tallinn, Estonia. PP. 35 - 39

Tanenbaum A.,(2003),**Computer Networks**, 4th Ed New jersey, Prentice Hall USA.

Tanenbaum, A., (2002). **Distributed Systems**, Prentice-Hall International, Inc, New Jersey.

Tanenbaum, A.,(1995). **Distributed Operating Systems**, Prentice-Hall International, Inc, New Jersey,1995.

Vasudevan S. ; Kurose,J. ,And Towsley , D.(2005). **Design and Analysis of a Leader Election Algorithm for Mobile Ad Hoc Networks**, Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE,International Conference on .

Villadangos, J. Cordoba A., Farina, F., Prieto, M. ,(2005), **Efficient leader election in complete networks**, Proceedings of the 13th Euromicro Conference on Parallel, IEEE, Page(s): 136 – 143.

Xu C., Wang J., Jun Huang **Petersen-Twisted-Torus Networks for Multiprocessor Systems** , (2010). Journal of Convergence Information Technology Volume 5, Number 9. November.

Zargarnataj, M., (2007) **New Election Algorithm based on Assistant in Distributed Systems**, IEEE journal page(s): 324 – 331.

Zhang G., Chen J., Zhang Y., Liu C,(2009) **Research of Asynchronous Leader Election**, . 5th International Conference on,, Page(s): 1 – 4.

Zhenyu xu .and srimani p., (2006). "**Self-stabilizing Anonymous Leader Election in a Tree**", International Journal of Foundations of computer science vol.17, No. 2 pp 323-335.

# Appendices

## Appendix A:

### Simulation source code

#### 1. Class Definition

Option Explicit

```
Public my_id As Integer
Public x_pos As Integer
Public y_pos As Integer
Public z_pos As Integer
Public leader_x_pos As Integer
Public leader_y_pos As Integer
Public leader_z_pos As Integer
Public leader_id As Integer
Public candidate_x_pos As Integer
Public candidate_y_pos As Integer
Public candidate_z_pos As Integer
Public candidate_id As Integer
Public normal As Boolean
Public local_phase As Integer
Public local_step As Integer

Public phase2done As Boolean
Public end_after_phase As Boolean
Public link_failure1 As Boolean
Public link_failure2 As Boolean
Public link_failure3 As Boolean
Public link_failure4 As Boolean
Public link_failure5 As Boolean
Public link_failure6 As Boolean
```

```

Public wait As Boolean
Public wait_time As Integer
Public wait_message As String
Public stored_phase3_message As String
Public stored_phase4_message As String
Private Type rec
    message As String * 22
    time As Integer
End Type

Private buffer(6) As rec
Const buffer_size = 7
Private buffer_head As Integer
Private buffer_tail As Integer

Public Function add(message As String, timer As Integer)

    Form1.Lnumberofmessage.Caption = number_of_message
    If Not isfull Then
        buffer(buffer_tail).message = message
        buffer(buffer_tail).time = timer
        buffer_tail = (buffer_tail + 1) Mod buffer_size
    End If
End Function

Public Sub delete()
    If Not isempty() Then buffer_head = (buffer_head + 1) Mod buffer_size
End Sub

Public Function isempty() As Boolean
    If buffer_head = buffer_tail Then
        isempty = True
    Else

```

```

        isempty = False
    End If
End Function
Public Function isfull() As Boolean
    If (buffer_tail + 1) Mod (buffer_size) = buffer_head Then
        isfull = True
    Else
        isfull = False
    End If
End Function
Public Function read_buffer() As String
    If Not isempty Then

        read_buffer = buffer(buffer_head).message
        delete
    End If
End Function
Public Function read_btime() As Integer
    If Not isempty Then
        read_btime = buffer(buffer_head).time
    End If
End Function

Public Sub show_buffer()
    Dim i%, bh%, bt As Integer
    bh = buffer_head
    bt = buffer_tail
    Do Until bh = bt
        Form1.List2.AddItem (buffer(bh).message & " " & read_btime())
        bh = (bh + 1) Mod buffer_size
    Loop
End Sub

```

## 2. Module

```
Public N&, x&, y&, z&
```

```
Public d As Integer, source_Y_pos$, source_X_pos$, source_Z_pos$
```

```
Public step As Boolean, message_type$
```

```
Public path$, filename As String
```

```
Public leader_x_pos%, leader_y_pos%, leader_z_pos%, time_step As Integer,
```

```
number_of_message As Long, ppp As Boolean, phase3inprogress As Boolean,
```

```
phase4inprogress As Boolean
```

```
Public node() As New vertices
```

```
Public Sub send(message$, ByVal tt%, dir%, ByVal a%, ByVal b%, ByVal c%)
```

```
'this sub send the message from the source node to the destination node buffer
```

```
Dim msg
```

```
number_of_message = number_of_message + 1
```

```
Select Case dir
```

```
Case 1
```

```
If Not node(a, b, c).link_failure1 Then ' send message through link labeled 1
```

```
Call node((a + 1) Mod x, b, c).add(message, tt)
```

```
msg = tt & Space(10) & node(a, b, c).x_pos & " , " & node(a, b, c).y_pos & "  
, " & node(a, b, c).z_pos & Space(10) & "send" & Space(10) & message &  
Space(10) & "to" & Space(10) & (node(a, b, c).x_pos + 1) Mod x & ", " & node(a,  
b, c).y_pos & " , " & node(a, b, c).z_pos
```

```
Form1.List1.AddItem (msg) ' show the message in the listbox
```

```
End If
```

```
Case 3
```

```
If Not node(a, b, c).link_failure3 Then ' send message through link labeled  
3
```

```
Call node((a - 1 + x) Mod x, b, c).add(message, tt)
```

```
msg = tt & Space(10) & node(a, b, c).x_pos & " , " & node(a, b, c).y_pos & "  
, " & node(a, b, c).z_pos & Space(10) & "send" & Space(10) & message &  
Space(10) & "to" & Space(10) & (node(a, b, c).x_pos - 1 + x) Mod x & ", " &  
node(a, b, c).y_pos & " , " & node(a, b, c).z_pos
```



```

                Form1.List1.AddItem (msg)
    End If
    Case 2
        If Not node(a, b, c).link_failure2 Then 'sned message through link labeled
2
            Call node(a, (b + 1) Mod y, c).add(message, tt)
            msg = tt & Space(10) & node(a, b, c).x_pos & " , " & node(a, b, c).y_pos & "
, " & node(a, b, c).z_pos & Space(10) & "send" & Space(10) & message &
Space(10) & "to" & Space(10) & (node(a, b, c).x_pos) & "," & (node(a, b,
c).y_pos + 1) Mod y & " , " & node(a, b, c).z_pos
                Form1.List1.AddItem (msg)
    End If
    Case 4
        If Not node(a, b, c).link_failure4 Then 'sned message through link labeled
4
            Call node(a, (b - 1 + y) Mod y, c).add(message, tt)
            msg = tt & Space(10) & node(a, b, c).x_pos & " , " & node(a, b, c).y_pos & "
, " & node(a, b, c).z_pos & Space(10) & "send" & Space(10) & message &
Space(10) & "to" & Space(10) & node(a, b, c).x_pos & "," & (node(a, b, c).y_pos
- 1 + y) Mod y & " , " & node(a, b, c).z_pos
                Form1.List1.AddItem (msg)
    End If
    Case 5
        If Not node(a, b, c).link_failure5 Then 'sned message through link labeled
5
            Call node(a, b, (c + 1) Mod z).add(message, tt)
            msg = tt & Space(10) & node(a, b, c).x_pos & " , " & node(a, b, c).y_pos & "
, " & node(a, b, c).z_pos & Space(10) & "send" & Space(10) & message &
Space(10) & "to" & Space(10) & (node(a, b, c).x_pos) & " , " & node(a, b,
c).y_pos & " , " & (node(a, b, c).z_pos + 1) Mod z
                Form1.List1.AddItem (msg)
    End If
    Case 6

```

If Not node(a, b, c).link\_failure6 Then 'sned message through link labeled  
6

Call node(a, b, (c - 1 + z) Mod z).add(message, tt)

msg = tt & Space(10) & node(a, b, c).x\_pos & " , " & node(a, b, c).y\_pos & "  
, " & node(a, b, c).z\_pos & Space(10) & "send" & Space(10) & message &  
Space(10) & "to" & Space(10) & node(a, b, c).x\_pos & "," & " , " & node(a, b,  
c).y\_pos & " , " & (node(a, b, c).z\_pos - 1 + z) Mod z

Form1.List1.AddItem (msg)

End If

Case 7 'sned message to first 2 dimintional torus

Call node(a, b, 0).add(message, tt)

msg = tt & Space(10) & node(a, b, c).x\_pos & " , " & node(a, b, c).y\_pos & "  
, " & node(a, b, c).z\_pos & Space(10) & "send" & Space(10) & message &  
Space(10) & "to" & Space(10) & node(a, b, c).x\_pos & "," & node(a, b, c).y\_pos  
& " , " & 0

Form1.List1.AddItem (msg)

End Select

End Sub

### 3. Main Form Code

Option Explicit

```
Private Sub cmdcls_Click()
```

```
    Cls
```

```
    time_step = 0
```

```
    configuration
```

```
End Sub
```

```
Private Sub CMDExit_Click()
```

```
End
```

```
End Sub
```

```
Private Sub CMDstep_Click()
```

```
    If time_step = 0 Then
```

```
        Dim ff As Integer
```

```
        ff = Val(configfrm.numtxt.Text)    'InputBox("enter the number of process that  
detect failure")
```

```
        If ff > N Then ff = 1
```

```
        leader_failure (ff)
```

```
        GoTo 60
```

```
    End If
```

```
        Timer1.Enabled = True
```

```
        step = True
```

```
60:
```

```
End Sub
```

```
Private Sub Nodes_status_Click()
```

```
Form3.Cls
```

```
Form3.Show
```

```
Dim i%, j%, k%
```

```
Form3.Print
```

```
Form3.Print Tab(5); " normal"; Tab(13); "leader ID"; Tab(23); "leader_pos";  
Tab(35); "I_phase" _
```

```
    ; Tab(45); "I_step"; Tab(55); "my pos"; Tab(65); "my id"; Tab(75); "C_ID";
```

```
        Tab(85); "C_pos"; Tab(100);
```

```
        "L_F1L_F2L_F3L_F4L_F5L_F6"
```

```

Form3.Print Space(10); String(105, "_")
For i = 0 To x - 1
  For j = 0 To y - 1
    For k = 0 To z - 1

      Form3.Print Tab(5); node(i, j, k).normal;
      Form3.Print Tab(13); node(i, j, k).leader_id;
      Form3.Print Tab(23); node(i, j, k).leader_x_pos;
      Form3.Print Tab(27); node(i, j, k).leader_y_pos;
      Form3.Print Tab(30); node(i, j, k).leader_z_pos;
      Form3.Print Tab(35); node(i, j, k).local_phase;
      Form3.Print Tab(45); node(i, j, k).local_step;
      Form3.Print Tab(55); node(i, j, k).x_pos;
      Form3.Print Tab(58); node(i, j, k).y_pos;
      Form3.Print Tab(61); node(i, j, k).z_pos;
      Form3.Print Tab(65); node(i, j, k).my_id;
      Form3.Print Tab(75); node(i, j, k).candidate_id;
      Form3.Print Tab(85); node(i, j, k).candidate_x_pos;
      Form3.Print Tab(88); node(i, j, k).candidate_y_pos;
      Form3.Print Tab(91); node(i, j, k).candidate_z_pos;
      Form3.Print Tab(100); node(i, j, k).link_failure1;
      Form3.Print Tab(105); node(i, j, k).link_failure2;
      Form3.Print Tab(110); node(i, j, k).link_failure3;
      Form3.Print Tab(115); node(i, j, k).link_failure4;
      Form3.Print Tab(120); node(i, j, k).link_failure5;
      Form3.Print Tab(125); node(i, j, k).link_failure6
    'Form3.Print Tab(105); node(i).phase2done;

  ' Form3.Print Tab(115); node(i).end_after_phase
    Next k
  Next j
Next i
End Sub

```

```

Private Sub CStart_Click()
    Dim ff As Integer
    30 ff = Val(configfrm.numtxt.Text) 'InputBox("enter the number of process
that detect failure")
    If ff > N Then MsgBox ("rong number tryagain"): GoTo 30

    leader_failure (ff)
    Timer1.Enabled = Not Timer1.Enabled
    step = False
End Sub

```

```

Private Sub Cbuffer_Click()
    Dim i%, j%, k%, message$, msg$
    filename = path & "\hh"
    Open filename For Output As #1
    List2.Clear

    List2.Visible = Not List2.Visible

    For i = 0 To x - 1
        For j = 0 To y - 1
            For k = 0 To z - 1

                msg = "node number " & node(i, j, k).x_pos & "," & node(i, j, k).y_pos & "," &
node(i, j, k).z_pos & " " & "BTime"
                List2.AddItem (msg)

                node(i, j, k).show_buffer
            Next k
        Next j
    Next i
    For j = 0 To List2.ListCount - 1
        Write #1, List2.List(j)
    
```

```

    Next j
    Close #1
End Sub
Private Sub Exit_Click(Index As Integer)
End
End Sub
Private Sub Form_Load()
path = configfrm.Pathtxt.Text
configuration
Me.Move (Screen.Width / 2) - (Me.Width / 2), (Screen.Height / 2) - (Me.Height
/ 2)

End Sub
Private Sub LFAILURE_Click(Index As Integer)
    Dim ff As Integer
    ff = Val(configfrm.numtxt.Text)
    If ff > N Then ff = 1

    leader_failure (ff)
End Sub

Private Sub sendmessages_Click(Index As Integer)
Dim i As Integer
filename = path & "\tt"
Open filename For Output As #1 ' open the file to save all messages
For i = 0 To List1.ListCount

Write #1, List1.List(i) ' save the message in the file
Next i
Close #1
End Sub

```

```
Private Sub setconfiguration_Click(Index As Integer)
    configfrm.Show
```

```
End Sub
```

```
Public Sub configuration()
    time_step = 0
    Randomize
```

```
List1.Clear
ppp = False
phase3inprogress = False
phase4inprogress = False
```

```
step = False
time_step = 0
Lnumberofmessage.Caption = ""
ltimesteps.Caption = ""
```

```
Dim aa%, Link_failure, X1, Y1, Z1, cv%, array1() As Integer, i%, j%, k%, bb%,
r%, flag As Boolean, b As Boolean
```

```
Label4.Caption = 1
```

```
x = Val(configfrm.NODESX.Text)
y = Val(configfrm.NODESY.Text)
z = Val(configfrm.NODESZ.Text)
    N = x * y * z
    ReDim node(x, y, z)
    ReDim array1(N)
    For i = 0 To N - 1
        array1(i) = 0
    Next i
```

```

For i = 0 To N - 1
  j = -1
  r = Int(((N) * Rnd()) + 1)
  Do Until j = i
    If r = array1(j + 1) Then
      r = Int(((N) * Rnd()) + 1)
      j = -1
    Else
      j = j + 1
    End If

    Loop
    array1(i) = r
  Next i
  cv = 0
For i = 0 To x - 1
  For j = 0 To y - 1
  For k = 0 To z - 1

  Do Until node(i, j, k).isempty
    node(i, j, k).delete
  Loop
  Next k
  Next j
  Next i
For i = 0 To x - 1
  For j = 0 To y - 1
  For k = 0 To z - 1
    node(i, j, k).my_id = array1(cv)
    cv = cv + 1
  Next k
  Next j
  Next i

```



```

For i = 0 To x - 1
For j = 0 To y - 1
For k = 0 To z - 1
If node(i, j, k).my_id = N Then
    leader_x_pos = i
    leader_y_pos = j
    leader_z_pos = k
End If
Next k
Next j
Next i

X1 = Val(configfrm.failureX.Text)
If (X1 > x) Or (X1 < 0) Then X1 = x

Y1 = Val(configfrm.failureY.Text)

If (Y1 > y) Or (Y1 < 0) Then Y1 = y
Z1 = Val(configfrm.failureZ.Text)
If (Z1 > z) Or (Z1 < 0) Then Z1 = z
Link_failure = Val(configfrm.Link.Text)
If Link_failure > 6 Or Link_failure < 1 Then Link_failure = 6

Select Case Link_failure
Case 1
    node(X1, Y1, Z1).link_failure1 = True
    node((X1 + 1) Mod x, Y1, Z1).link_failure3 = True
Case 2
    node(X1, Y1, Z1).link_failure2 = True
    node(X1, (Y1 + 1) Mod y, Z1).link_failure4 = True

```

Case 3

```
node(X1, Y1, Z1).link_failure3 = True  
node((X1 - 1 + x) Mod x, Y1, Z1).link_failure1 = True
```

Case 4

```
node(X1, Y1, Z1).link_failure4 = True  
node(X1, (Y1 - 1 + y) Mod y, Z1).link_failure2 = True
```

Case 5

```
node(X1, Y1, Z1).link_failure5 = True  
node(X1, Y1, (Z1 + 1) Mod z).link_failure6 = True
```

Case 6

```
node(X1, Y1, Z1).link_failure6 = True  
node(X1, Y1, (Z1 - 1 + z) Mod z).link_failure5 = True
```

End Select

ppp = False

For i = 0 To x - 1

For j = 0 To y - 1

For k = 0 To z - 1

bb = i

```
node(i, j, k).normal = True  
node(i, j, k).leader_id = N  
node(i, j, k).leader_x_pos = leader_x_pos  
node(i, j, k).leader_y_pos = leader_y_pos  
node(i, j, k).leader_z_pos = leader_z_pos  
node(i, j, k).local_phase = 0  
node(i, j, k).local_step = 0  
node(i, j, k).x_pos = i  
node(i, j, k).y_pos = j  
node(i, j, k).z_pos = k  
node(i, j, k).candidate_id = 0  
node(i, j, k).candidate_x_pos = 0  
node(i, j, k).candidate_y_pos = 0
```

```

node(i, j, k).candidate_z_pos = 0
node(i, j, k).wait = False
node(i, j, k).wait_time = 0
node(i, j, k).wait_message = ""
node(i, j, k).stored_phase3_message = ""
node(i, j, k).stored_phase4_message = ""
Next k
Next j

Next i
number_of_message = 0

End Sub

Private Sub start_Click(Index As Integer)
Timer1.Enabled = Not Timer1.Enabled
End Sub

Private Sub stop_Click()
Timer1.Enabled = Not Timer1.Enabled
End Sub

Private Sub Timer1_Timer()

Dim i%, j%, k%, message$, mssg$, source_Y_pos$, source_X_pos$, dist%,
phase3step%, phase4step%, msg As String, mm As String

message = ""
mssg = ""

time_step = time_step + 1
ltimesteps.Caption = time_step
For i = 0 To x - 1

```

```

For j = 0 To y - 1
  For k = 0 To z - 1
    If Not node(i, j, k).isempty Then
      Do Until node(i, j, k).isempty Or node(i, j, k).read_btime = time_step
        message = node(i, j, k).read_buffer

        If Left(message, 3) = "ack" Then
          node(i, j, k).wait = False
          node(i, j, k).wait_time = 0
          node(i, j, k).wait_message = ""
          GoTo 20
        End If

        If Mid(message, 2, 1) > Val(Label4.Caption) Then Label4.Caption =
Mid(message, 2, 1)

        If Val(Mid(message, 1, 1)) < 5 Then
          node(i, j, k).normal = False
        End If
        If Val(Mid(message, 1, 1)) = 5 Then
          node(i, j, k).normal = True
        End If
' =====
'error number 1 in phase 2 the detour is 3,5,1
If Mid(message, 1, 1) = "a" Then
  mssg = "b" & Mid(message, 2, 22)
  Call send(mssg, time_step, 5, i, j, k)
End If
If Mid(message, 1, 1) = "b" Then
  mssg = "c" & Mid(message, 2, 22)
  Call send(mssg, time_step, 1, i, j, k)
End If
If Mid(message, 1, 1) = "c" Then
  mssg = "1" & Mid(message, 2, 22)

```

```

    Call phase2(mssg, i, j, k, 0)
End If

'error number 2 in phase 3 the detour is 5,1,6
If Mid(message, 1, 1) = "d" Then
    mssg = "e" & Mid(message, 2, 22)
    Call send(mssg, time_step, 1, i, j, k)

End If
If Mid(message, 1, 1) = "e" Then
    mssg = "f" & Mid(message, 2, 22)
    Call send(mssg, time_step, 6, i, j, k)
End If
If Mid(message, 1, 1) = "f" Then
    mssg = "3" & Mid(message, 2, 22)
    Call phase3(mssg, i, j, k, 0)
End If

'error number 3 in phase 4 the detour is 1,2,3
If Mid(message, 1, 1) = "g" Then
    mssg = "h" & Mid(message, 2, 22)
    Call send(mssg, time_step, 2, i, j, k)

End If
If Mid(message, 1, 1) = "h" Then
    mssg = "i" & Mid(message, 2, 22)
    Call send(mssg, time_step, 3, i, j, k)
End If
If Mid(message, 1, 1) = "i" Then
    mssg = "4" & Mid(message, 2, 22)
    Call phase4(mssg, i, j, k, 0)
End If

```

```

If Val(Mid(message, 1, 1)) = 0 Then

    Call phase1(message, i, j, k)
End If

'-----

If Val(Mid(message, 1, 1)) = 1 Then
    Call phase2(message, i, j, k, 1)
End If

'-----

If Mid(message, 1, 1) = "2" Then

    node(i, j, k).candidate_id = Mid(message, 5, 6)
    node(i, j, k).candidate_x_pos = Mid(message, 11, 2)
    node(i, j, k).candidate_y_pos = Mid(message, 13, 2)
    node(i, j, k).candidate_z_pos = Mid(message, 15, 2)
    node(i, j, k).local_phase = 3

    If node(i, j, k).x_pos = 0 Then
        message = "3" & Mid(message, 2, 22)
        Mid(message, 3, 2) = "01"
        Call send(message, time_step, 1, i, j, k)
        node(i, j, k).wait = True
        node(i, j, k).wait_time = time_step
        node(i, j, k).wait_message = "d" & Mid(message, 2, 22)
    End If

    If Len(node(i, j, k).stored_phase3_message) >= 7 Then
        If Val(Mid(node(i, j, k).stored_phase3_message, 5, 6)) >
Val(Mid(message, 5, 6)) Then
            message = node(i, j, k).stored_phase3_message
        End If
    End If

```

```

        message = "3" & Mid(message, 2, 22)

        Mid(message, 3, 2) = Val(Mid(node(i, j,
k).stored_phase3_message, 3, 2)) + 1

        Call send(message, time_step, 1, i, j, k)

    End If
    GoTo 20
End If
'-----

If Val(Mid(message, 1, 1)) = 3 Then
    Call phase3(message, i, j, k, 1)
End If
'-----

If Mid(message, 1, 1) = "4" Then
    Call phase4(message, i, j, k, 1)
End If
'-----

If Mid(message, 1, 1) = "5" Then
    Call phase5(message, time_step, i, j, k)
End If
'-----

20:
    message = ""
    mssg = ""

    Loop
End If

```

```

If node(i, j, k).wait And time_step - node(i, j, k).wait_time = 2 Then
    mssg = node(i, j, k).wait_message

    If node(i, j, k).local_phase = 2 Then Call send(mssg, time_step, 3, i, j, k)
    If node(i, j, k).local_phase = 3 Then Call send(mssg, time_step, 5, i, j, k)
    If node(i, j, k).local_phase = 4 Then Call send(mssg, time_step, 1, i, j, k)
    node(i, j, k).wait = False
    node(i, j, k).wait_time = 0
    node(i, j, k).wait_message = ""
End If
'Dim ttt%
'And time_step < (((9 / 2) * x) + 9 + rand)
    If Not node(i, j, k).normal Then ppp = True
Next k
Next j
Next i
40:
If step = True Then Timer1.Enabled = False
If Not ppp Then Timer1.Enabled = False ': MsgBox (" finished")

ppp = False
End Sub

Private Sub leader_failure(f As Integer)
    Randomize
    Static flag2 As Boolean
    If flag2 Then configuration
    flag2 = True
    Dim i%, k%, a%, b%, c%, r%, message$, dist%, msg As String
    time_step = 1
    node(leader_x_pos, leader_y_pos, leader_z_pos).my_id = 0
    filename = path & "\tt"

```



For i = 1 To f

70: a = (Int(N \* Rnd())) Mod x

b = (Int(N \* Rnd())) Mod y

k = (Int(N \* Rnd())) Mod z

If node(a, b, k).my\_id = node(a, b, c).leader\_id Then GoTo 70

If node(a, b, k).normal = True Then

node(a, b, k).normal = False

node(a, b, k).local\_phase = 1

node(a, b, k).local\_step = 1

node(a, b, k).leader\_id = -1

node(a, b, k).leader\_x\_pos = -1

node(a, b, k).leader\_y\_pos = -1

node(a, b, k).leader\_z\_pos = -1

node(a, b, k).candidate\_id = node(a, b, k).my\_id

node(a, b, k).candidate\_x\_pos = node(a, b, k).x\_pos

node(a, b, k).candidate\_y\_pos = node(a, b, k).y\_pos

node(a, b, k).candidate\_z\_pos = node(a, b, k).z\_pos

message = Format(0, "0") & Format(node(a, b, k).local\_phase, "0") &  
Format(node(a, b, k).local\_step, "00") & Format(0, "000000") & Format(0, "00")  
& Format(0, "00") & Format(0, "00") & Format(node(a, b, k).x\_pos, "00") &  
Format(node(a, b, k).y\_pos, "00") & Format(node(a, b, k).z\_pos, "00")

Call send(message, time\_step, 1, a, b, k)

Call send(message, time\_step, 3, a, b, k)

Call send(message, time\_step, 2, a, b, k)

Call send(message, time\_step, 4, a, b, k)

node(a, b, k).local\_phase = 2

node(a, b, k).local\_step = 1

message\_type = 1

source\_X\_pos = node(a, b, k).x\_pos

```

source_Y_pos = node(a, b, k).y_pos
source_Z_pos = node(a, b, k).z_pos
message = Format(message_type, "0") & Format(node(a, b,
k).local_phase, "0") & Format(node(a, b, k).local_step, "00") & Format(node(a,
b, k).candidate_id, "000000") & Format(node(a, b, k).x_pos, "00") &
Format(node(a, b, k).y_pos, "00") & Format(node(a, b, k).z_pos, "00") &
Format(source_X_pos, "00") & Format(source_Y_pos, "00") &
Format(source_Z_pos, "00")

```

```

Call send(message, time_step, 5, a, b, k)

```

```

node(a, b, k).wait = True

```

```

node(a, b, k).wait_time = time_step

```

```

node(a, b, k).wait_message = "a" & Mid(message, 2, 22)

```

```

Else

```

```

i = i - 1

```

```

End If

```

```

Next i

```

```

ltimesteps.Caption = time_step

```

```

End Sub

```

```

Private Sub phase1(message$, i%, j%, k%)

```

```

Dim mssg As String

```

```

If node((i + 1) Mod x, j, k).normal = True Then

```

```

    Call send(message, time_step, 1, i, j, k)

```

```

    node((i + 1) Mod x, j, k).normal = False

```

```

End If

```

```

If node((i - 1 + x) Mod x, j, k).normal = True Then

```

```

    Call send(message, time_step, 3, i, j, k)

```

```

    node((i - 1 + x) Mod x, j, k).normal = False

```

```

End If

```

```

If node(i, (j + 1) Mod y, k).normal = True Then

```

```

    Call send(message, time_step, 2, i, j, k)

```

```

    node(i, (j + 1) Mod y, k).normal = False

```

```

End If
If node(i, (j - 1 + y) Mod y, k).normal = True Then
    Call send(message, time_step, 4, i, j, k)
    node(i, (j - 1 + y) Mod y, k).normal = False
End If

node(i, j, k).local_phase = 2
node(i, j, k).local_step = 1
node(i, j, k).leader_id = -1
node(i, j, k).leader_x_pos = -1
node(i, j, k).leader_y_pos = -1
node(i, j, k).candidate_id = node(i, j, k).my_id
node(i, j, k).candidate_x_pos = node(i, j, k).x_pos
node(i, j, k).candidate_y_pos = node(i, j, k).y_pos
node(i, j, k).candidate_z_pos = node(i, j, k).z_pos
message_type = 1      ' election message
source_X_pos = node(i, j, k).x_pos
source_Y_pos = node(i, j, k).y_pos
source_Z_pos = node(i, j, k).z_pos

mssg = Format(message_type, "0") & Format(node(i, j, k).local_phase,
"0") & Format(node(i, j, k).local_step, "00") & Format(node(i, j, k).candidate_id,
"000000") _
    & Format(node(i, j, k).candidate_x_pos, "00") & Format(node(i, j,
k).candidate_y_pos, "00") & Format(node(i, j, k).candidate_z_pos, "00") &
Format(source_X_pos, "00") & Format(source_Y_pos, "00") &
Format(source_Z_pos, "00")

Call send(mssg, time_step, 5, i, j, k)
node(i, j, k).wait = True
node(i, j, k).wait_time = time_step
node(i, j, k).wait_message = "a" & Mid(mssg, 2, 22)

End Sub

```

```

Private Sub phase2(message$, i%, j%, k%, a%)
Dim mssg$

    If node(i, j, k).local_phase = 2 And Val(node(i, j, k).z_pos) >
Val(Mid(message, 21, 2)) Then
        If a = 1 Then Call send("ack", time_step, 6, i, j, k): GoTo 30

    End If

    node(i, j, k).local_phase = 2

    If node(i, j, k).my_id > Val(Mid(message, 5, 6)) Then
        Mid(message, 5, 6) = Format(node(i, j, k).my_id, "000000")
        Mid(message, 11, 2) = Format(node(i, j, k).x_pos, "00")
        Mid(message, 13, 2) = Format(node(i, j, k).y_pos, "00")
        Mid(message, 15, 2) = Format(node(i, j, k).z_pos, "00")
    End If

    node(i, j, k).leader_id = -1
    node(i, j, k).leader_x_pos = -1
    node(i, j, k).leader_y_pos = -1
    node(i, j, k).leader_z_pos = -1

    If a = 1 Then Call send("ack", time_step, 6, i, j, k)
        Mid(message, 3, 2) = Format(Val(Mid(message, 3, 2) + 1), "00")
        If node(i, j, k).z_pos <> Val(Mid(message, 21, 2)) Then
            Call send(message, time_step, 5, i, j, k)
            node(i, j, k).wait = True
            node(i, j, k).wait_time = time_step
            node(i, j, k).wait_message = "a" & Mid(message, 2, 22)
        Else
            Mid(message, 3, 2) = Format(0, "00")
            mssg = "2" & Mid(message, 2, 22)
            Call send(mssg, time_step, 7, i, j, k)
            If k <> 0 Then
                node(i, j, k).candidate_id = 0
            End If
        End If
    End If
End Sub

```

```

node(i, j, k).candidate_x_pos = 0
node(i, j, k).candidate_y_pos = 0
node(i, j, k).candidate_z_pos = 0
End If
node(i, j, k).wait = False
node(i, j, k).wait_time = 0
node(i, j, k).wait_message = ""
End If

```

30:

End Sub

Private Sub phase3(message\$, i%, j%, k, a%)

```

    If a = 1 Then Call send("ack", time_step, 3, i, j, k)
        If node(i, j, k).local_phase = 3 Then
            If node(i, j, k).candidate_id >= Val(Mid(message, 5, 6)) Then
                Mid(message, 5, 6) = Format(node(i, j, k).candidate_id,
"000000")
                Mid(message, 11, 2) = Format(node(i, j,
k).candidate_x_pos, "00")
                Mid(message, 13, 2) = Format(node(i, j,
k).candidate_y_pos, "00")
                Mid(message, 15, 2) = Format(node(i, j,
k).candidate_z_pos, "00")
            Else
                node(i, j, k).candidate_id = Mid(message, 5, 6)
                node(i, j, k).candidate_x_pos = Mid(message, 11, 2)
                node(i, j, k).candidate_y_pos = Mid(message, 13, 2)
                node(i, j, k).candidate_z_pos = Mid(message, 15, 2)
            End If
            Else
                node(i, j, k).stored_phase3_message = message
            End If

```

```

If node(i, j, k).x_pos <> 0 Then
  If node(i, j, k).local_phase = 3 Then
    If node(i, j, k).candidate_id >= Val(Mid(message, 5, 6)) Then
      Mid(message, 5, 6) = Format(node(i, j, k).candidate_id,
"000000")
      Mid(message, 11, 2) = Format(node(i, j,
k).candidate_x_pos, "00")
      Mid(message, 13, 2) = Format(node(i, j,
k).candidate_y_pos, "00")
      Mid(message, 15, 2) = Format(node(i, j,
k).candidate_z_pos, "00")
    Else
      node(i, j, k).candidate_id = Mid(message, 5, 6)
      node(i, j, k).candidate_x_pos = Mid(message, 11, 2)
      node(i, j, k).candidate_y_pos = Mid(message, 13, 2)
      node(i, j, k).candidate_z_pos = Mid(message, 15, 2)
    End If
    Call send(message, time_step, 1, i, j, k)
    node(i, j, k).wait = True
    node(i, j, k).wait_time = time_step
    node(i, j, k).wait_message = "d" & Mid(message, 2, 22)
  Else
    node(i, j, k).stored_phase3_message = message
  End If
End If
If node(i, j, k).x_pos = 0 And node(i, j, k).y_pos = 0 And node(i, j,
k).z_pos = 0 And node(i, j, k).local_phase = 3 Then

  node(i, j, k).local_phase = 4
  Mid(message, 1, 1) = Format(4, "0")
  Mid(message, 2, 1) = Format(4, "0")
  Mid(message, 3, 2) = Format(0, "00")
  Call send(message, time_step, 2, i, j, k)
  node(i, j, k).wait = True

```

```

node(i, j, k).wait_time = time_step
node(i, j, k).wait_message = "g" & Mid(message, 2, 22)
End If

If node(i, j, k).x_pos = 0 And node(i, j, k).y_pos <> 0 And node(i, j,
k).z_pos = 0 Then
node(i, j, k).local_phase = 4
If Len(node(i, j, k).stored_phase4_message) >= 7 Then
If Val(Mid(node(i, j, k).stored_phase4_message, 5, 6)) >
Val(Mid(message, 5, 6)) Then
message = node(i, j, k).stored_phase3_message
End If
message = "4" & Mid(message, 2, 22)

Mid(message, 3, 2) = Val(Mid(node(i, j,
k).stored_phase3_message, 3, 2)) + 1

Call send(message, time_step, 2, i, j, k)

End If

End If

```

End Sub

Private Sub phase4(message\$, i%, j%, k, a%)

```

If a = 1 Then Call send("ack", time_step, 4, i, j, k)
If node(i, j, k).y_pos <> 0 And node(i, j, k).x_pos = 0 And node(i, j,
k).z_pos = 0 Then
If node(i, j, k).local_phase = 4 Then

```

```

If node(i, j, k).candidate_id > Val(Mid(message, 5, 6)) Then
    Mid(message, 5, 6) = Format(node(i, j, k).candidate_id, "000000")
    Mid(message, 11, 2) = Format(node(i, j, k).candidate_x_pos, "00")
    Mid(message, 13, 2) = Format(node(i, j, k).candidate_y_pos, "00")
    Mid(message, 15, 2) = Format(node(i, j, k).candidate_z_pos, "00")
    Mid(message, 3, 2) = Format(Mid(message, 3, 2) + 1, "00")
End If
Call send(message, time_step, 2, i, j, k)
node(i, j, k).wait = True
node(i, j, k).wait_time = time_step
node(i, j, k).wait_message = "g" & Mid(message, 2, 22)
Else
    node(i, j, k).stored_phase4_message = message
End If
End If

```

```

If node(i, j, k).x_pos = 0 And node(i, j, k).y_pos = 0 And node(i, j, k).z_pos
= 0 Then

```

```

    node(i, j, k).local_phase = 5
    node(i, j, k).leader_id = Mid(message, 5, 6)
    node(i, j, k).leader_x_pos = Mid(message, 11, 2)
    node(i, j, k).leader_y_pos = Mid(message, 13, 2)
    node(i, j, k).leader_z_pos = Mid(message, 15, 2)
    Mid(message, 1, 1) = Format(5, "0")
    Mid(message, 2, 1) = Format(5, "0")
    Mid(message, 3, 2) = Format(0, "00")
    ' MsgBox (message)
    Call send(message, time_step, 1, i, j, k)
    Call send(message, time_step, 3, i, j, k)
    node(i, j, k).normal = True
    node(i, j, k).leader_id = Val(Mid(message, 5, 6))
    node(i, j, k).leader_x_pos = Val(Mid(message, 11, 2))
    node(i, j, k).leader_y_pos = Val(Mid(message, 13,

```

2))



```

node(i, j, k).leader_z_pos = Val(Mid(message, 15, 2))
node(i, j, k).local_phase = 0
node(i, j, k).local_step = 0
node(i, j, k).candidate_id = 0
node(i, j, k).candidate_x_pos = 0
node(i, j, k).candidate_y_pos = 0
node(i, j, k).candidate_z_pos = 0
node(i, j, k).wait = False
node(i, j, k).wait_time = 0
node(i, j, k).wait_message = ""
node(i, j, k).stored_phase3_message = ""
node(i, j, k).stored_phase4_message = ""

```

End If

End Sub

Private Sub phase5(message\$, time\_step, i%, j%, k%)

Dim mssg As String

```

node(i, j, k).normal = True
' MsgBox ("inside phase 5 " & message)
node(i, j, k).leader_id = Mid(message, 5, 6)
node(i, j, k).leader_x_pos = Mid(message, 11, 2)
node(i, j, k).leader_y_pos = Mid(message, 13, 2)
node(i, j, k).leader_z_pos = Mid(message, 15, 2)

```

If node((i + 1) Mod x, j, k).normal = False Then

Call send(message, time\_step, 1, i, j, k)

node((i + 1) Mod x, j, k).normal = True

End If

If node((i - 1 + x) Mod x, j, k).normal = False Then

Call send(message, time\_step, 3, i, j, k)

```
node((i - 1 + x) Mod x, j, k).normal = True
End If
```

```
If node(i, (j + 1) Mod y, k).normal = False Then
  Call send(message, time_step, 2, i, j, k)
  node(i, (j + 1) Mod y, k).normal = True
End If
```

```
If node(i, (j - 1 + y) Mod y, k).normal = False Then
  Call send(message, time_step, 4, i, j, k)
  node(i, (j - 1 + y) Mod y, k).normal = True
End If
```

```
If node(i, j, (k + 1) Mod z).normal = False Then
  Call send(message, time_step, 5, i, j, k)
End If
```

```
If node(i, j, (k - 1 + z) Mod z).normal = False Then
  Call send(message, time_step, 6, i, j, k)
  node(i, j, (k - 1 + z) Mod z).normal = True
End If
```

```
node(i, j, k).local_phase = 0
node(i, j, k).local_step = 0
node(i, j, k).candidate_id = 0
node(i, j, k).candidate_x_pos = 0
node(i, j, k).candidate_y_pos = 0
node(i, j, k).candidate_z_pos = 0
node(i, j, k).wait = False
node(i, j, k).wait_time = 0
node(i, j, k).wait_message = ""
node(i, j, k).stored_phase3_message = ""
node(i, j, k).stored_phase4_message = ""
```

End Sub

