New Leader Election Algorithms in Hypercube Networks

Prepared by

Mohammed N. S. Alrefai

Supervisor

Prof. Naim Ajlouni

A dissertation submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Computer Science.

Graduate College of Computer Studies

Amman Arab University for Graduate Studies

December, 2006

Authorization of Dissemination

b

Authorization of Dissemination

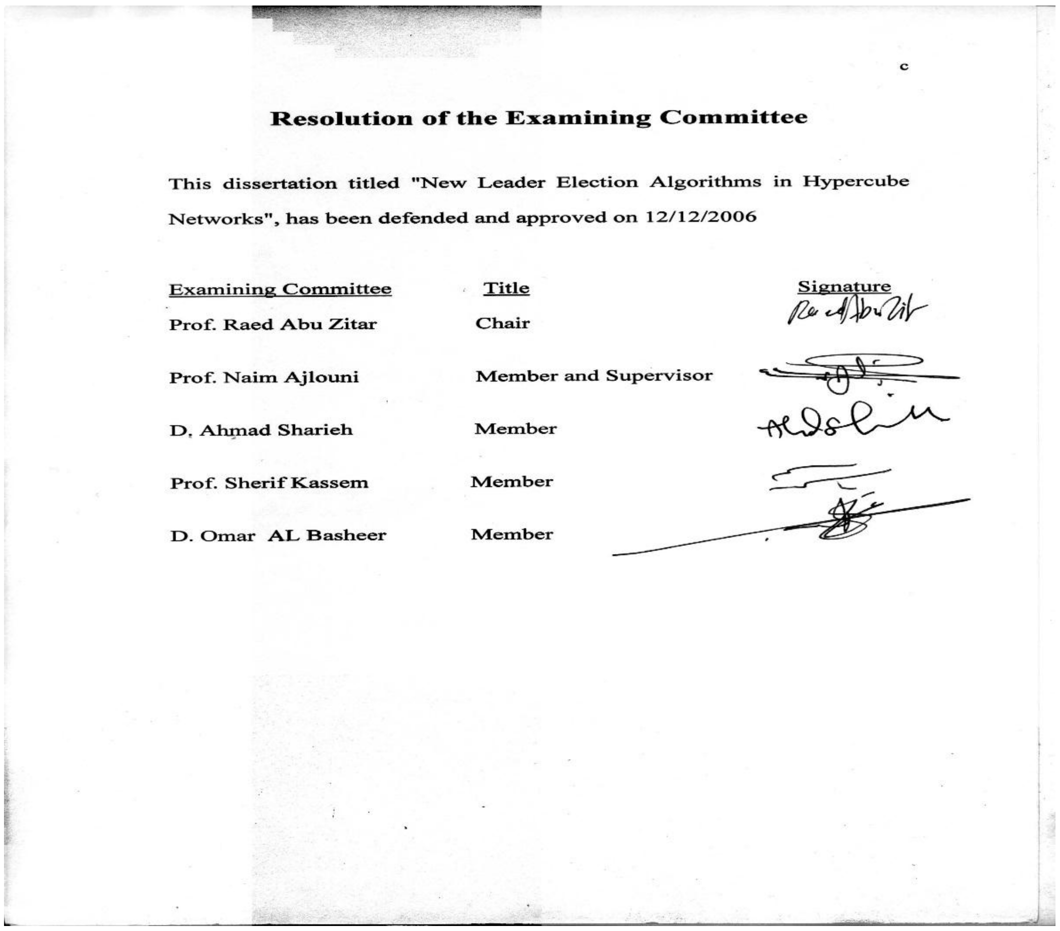
I, the undersigned "Mohammed Nayef AlRefai" authorize hereby Amman Arab University for Graduate Studies to provide copies of this Dissertation to libraries, institutions, agencies, and any other parties upon their request.

Name: Mohammed Nayef Al-Refai

Signature: [illegible]

Date: 15/1/2007

Resolution of the Examining Committee



c

Resolution of the Examining Committee

This dissertation titled "New Leader Election Algorithms in Hypercube Networks", has been defended and approved on 12/12/2006

Examining Committee	Title	Signature
Prof. Raed Abu Zitar	Chair	[illegible]
Prof. Naim Ajlouni	Member and Supervisor	[illegible]
D. Ahmad Sharieh	Member	[illegible]
Prof. Sherif Kassem	Member	[illegible]
D. Omar AL Basheer	Member	[illegible]

Acknowledgment

"All praises and thanks to ALLAH"

I would like to thank my supervisor Dr. Naim Ajlouni who encouraged and helped me very much to produce this work. I also thank all my colleagues and relatives for providing me support throughout and encouraging me always to give my best.

My special thanks to my wife who kept close to me all the time, and my parents for their praying and providing supports.

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

Dedication

I dedicate this work to

My Parents, Wife, Son, Brothers, Sisters, and Friends

Table of Contents

List of Tables

List of Figures

List of Appendices

New Leader Election Algorithms in Hypercube Networks

Prepared by
Mohammed N. S. Alrefai

Supervisor
Prof. Dr. Naim Ajlouni

Abstract

Leader Election is a fundamental problem in centralized control of distributed systems. The election process starts when one or more processors discover that the leader has failed, and it terminates when the remaining processors are aware who the new leader is.

This Dissertation presents two new algorithms in distributed systems to solve this problem in Hypercube networks. The first algorithm presents a new solution to leader failure problem with least number of messages and time steps. The second algorithm provides a new solution to solve leader failure problem with the presence of one link failure.

Distributed leader election algorithms performance is evaluated in this dissertation by mathematical proof and simulation program for the first algorithm was made. Contention and synchronization issues are considered in both algorithms.

In a network of N nodes connected by a hypercube network, the first algorithm uses O(N) messages to elect a new leader in O(log(N)) time steps when the leader failure is detected by one processor in the simple case. In the worst case, when the failure is detected by more than one processor reached to N-1, the first algorithm uses O(N Log(N)) messages to elect a new leader in O(log(N)) time steps.

For the second algorithm, it uses O(N) messages to elect a new leader in O(log(N)) time steps in the simple case and O(N Log (N)) messages in O(log(N)) time steps in the worst case.

خوارزميات جديدة لانتخاب القائد في الشبكات عالية التكعيب

إعداد

محمد نايف الرفاعي

اشراف الاستاذ الدكتور

نعيم العجلوني

الملخص

Arabic summary

تعتبر مشـكلة انتخاب القائد من المشـاكل المهمة في انظمة الحوسـبة الموزعة المركزية التحكم لمختلف التراكيب الشـبكية، حيث يكون أحد معالجات الشـبكة مركزا للتحكم ويسـمى القـائد بينما تكون باقي المعالجات تابعة له. وتبدأ هذه المشـكلة عندما يتم اكتشـاف تعطل القائد من قبل معالج آخر أو أكثر، فتدخل كافة معالجات الشـبكة في انتخـابـات لاختيـار أحدها قائدا جديدا للشـبكة، وتنتهي عملية الانتخاب بمعرفة جميع المعالجات بالقائد الجديد.

تبحث هذه الاطروحة في خوارزميات انتخاب القائد في الشـبكات عالية التكعيب، وقـد تم اقتراح خوارزميتين لانتخـاب القـائـد، الاولى تقـدم فكرة جـديـدة بكفـاءة عـاليـة لعملية الانتخاب في الشـبكات عالية التكعيب. بينما تبتكر الثانية حلا لمشـكلة تعطل القائد مع وجود عطل في احد خطوط الاتصال البيني.

وتقدم الاطروحة اثبات رياضي يعتمد على حسـاب عدد الرسـائل اللازمة لإتمام عمل الخوارزميـة وعـدد الخطوات الزمنيـة. كما تجري عمليـة محاكاة تقوم بتنفيذ الخوارزمية الاولى على اي شكل من اشكال الشبكات عالية التكعيب.

علما بأن الدراسـة تأخذ بعين الاعتبار القضـايا الأسـاسـية التي ترافق هذا النوع من الخوارزميات، مثل التزامن (Synchronization) والتزاحم (Contention).

إذا افترضـنا أن N عدد المعالجات في الشـبكة فإن كل من الخوارزميتين المقترحتين تسـتخدم في أفضـل الحالات (N)**O** رسـالة ضـمن ((O(Log(N خطوة زمنية. أما في أسـوأ الحالات فانها تسـتخدم (O(N Log N رسـالة للحصـول على قائد جديد للشـبكة، وتحتاج ((O(Log (N خطوة زمنية لإتمام ذلك.

CHAPTER ONE
INTRODUCTION

1.1. Overview

One of the most fundamental problems in distributed systems is the leader failure. This problem can be solved by Leader Election Algorithms (LEAs) (Shrira and Goldreich, 1987). These algorithms move the system from an initial state, where all the nodes are in the same computation state, into a new state where only one node is distinguished computationally called leader (Dolev et al, 1997).

Distributed systems are used to increase the computational speed of problem solving. These systems use a number of computers which cooperate with each other to execute some task (Tanenbaum, 2002). The control of distributed algorithms requires one node to act as a controller (leader). If the leader crashes or fails for any reason a new leader should be elected. The LEA's solves this problem by substituting the failed leader by a new leader.

The election process is a program distributed over all nodes, it starts when the leader failure discovered by one node at simple case, or by more than one, until all nodes except the failed one in the worst case. It terminates when all nodes know who's the new leader (Flocchini and Mans,1996).

The LEAs are widely used in centralized systems to solve single point failure problem. For example, in client-server, LEAs are used when the server fails and the system needs to transfer the leadership to another station (Larrea et al, 2000). The LEAs are also used in token ring. When the node has the token fails, the system should select a new node to have the token.

In distributed and parallel systems, there are many network topologies like hypercube, meshes, ring, bus,…etc. These topologies may be either hardware processors, or software processes embeded over the other hardware topology. The hypercube model has many properties that make it one of the most important topologies in distributed and parallel systems, the short diametar, nodes addressing, optimal routing path from initiater to the destination and its symmetry and regularity are some of these properties (Castorino and Ciccarella, 1999;Kumar et al, 2003; Flocchini and Mans,1996; Gerard ,1993; Singh,1997).

This dissertation will focus on the hypercube nodes topology where one node works as leader and the others as defeated. We propose two algorithms for leader election to solve leader failure. The first one solve the leader failure in hypercube with less time steps and less number of messages. The second algorithm solve a more complicated problem when the system has one link failure as well as a leader failure. Both algorithms will be executed without user intervention.

This research is based on mathematical method to find the number of messages and the number of time steps . It will use simulation to validate the results for the first algorithm.

1.2.Problem Statement

The current leader election algorithms for the hypercube suffer from inefficient performance caused by the high number of messages and time steps (Dobrev and Ruzicka, 1997; Flocchini and Mans,1996; Gerard ,1993; Castorino and Ciccarella, 1999). Also current algorithms do not solve the problem of link failure during the leader election algorithm process. This research is aimed at presenting two new algorithms; the first deals with the leader failure problem in hypercube with minimum time steps and minimum number of messages, which will enhance the algorithm performance. The second algorithm presents a solution for the leader election with the presence of one link failure.

1.3 .Goals of this Dissertation

This dissertation proposed two algorithms, which solve the leader failure problem in hypercube. The new algorithms will have the following objectives:

1- More efficient than the previous algorithms.

2- Less contention and light load over the network.

3- Require least time steps and number of messages.

4- Better synchronization between different steps during the execution of the algorithm.

5- Use asynchronous communication in transfer of messages.

6- Shorter length of messages.

7- Solve the problem with the presence of link failure.

1.4.Dissertation Contributions

The contributions of this dissertation are summarized in presenting two new algorithms for leader election in hypercube. The first one solves the problem of leader crash by using less number of messages and time steps. The second algorithm considers the probability of one link failure occurring during the execution of the leader election process in hypercube.

1.5.Dissertation Structure

This dissertation is organized as follows:

- **Chapter One:** presents introduction to dissertation subject, election algorithms, problem statement, goals of the dissertation and dissertation contributions.
- **Chapter Two:** provides some parallel and distributed information related to dissertation, distributed systems, multicomputers, multiprocessors, interconnection network, design issues, Performance and complexity and finally the description and properties of the hypercube model.
- **Chapter Three:** presents the previous work in election algorithms problem.
- **Chapter Four**: contains the two election algorithms that represent dissertation work.
- **Chapter Five:** deals with the analyses of the research algorithms. It presents mathematical proves and the necessary simulation that verifies the first algorithm validity.

- **Chapter Six:** presents conclusions, results discussion, and future work.
- **Appendices:** three appendices are added**:** the first appendix contains simulation code, and the second contains the messages used during one step in the simulation. The third appendix presents flowcharts for the proposed algorithms.

CHAPTER TWO

INTRODUCTION TO PARALLEL AND DISTRIBUTED SYSTEMS

2.1. Introduction.

Computer systems from 1945 until about 1985 were large and very expensive. Even minicomputers normally cost tens of thousands of dollars and took up a very huge space. Starting in the 1980s two advantages in computer and communications technology began to change the situation. The first was the development of powerful microprocessors. Which had a very high computing power, but for a fraction of the price. The second development was the invention of high speed computer networks. The Local Area Networks or LANs allow many machines within an area to be connected to each other so a small amount of information can be transferred between machines in millisecond (Tanenbaum, 1995).

The result of these technologies is that it is easy to put together computer systems composed of large number of CPUs connected by high speed networks. They are usually called distributed systems, in contrast to the previous centralized systems which consisted of single processor systems (Tanenbaum, 2002). This creates a new parallel computer technology. In parallel computing, many processors cooperate with each other to solve a huge computational problem.

The architecture of the distributed and parallel computers consists of multiple CPUs. Parallel computers can be organized in many different ways, especially in terms of how they are interconnected and how they communicate. This chapter includes the distributed and parallel systems.

2.2 Distributed Systems and Parallel Computers

Distributed system is defined as a collection of independent computers that appear to the users of the system as a single computer. This definition deals with hardware: the machines are autonomous and from a software perspective the user think of the system as a single computer (Tanenbaum, 1995).

A parallel computer is defined as a set of processors that are able to work cooperatively to solve a computational problem (Foster, 1995).

2.2.1Advantages of Distributed Systems over Centralized Systems.

- **Economics**: The most cost effective solution to get a faster computer is to harness a large number of cheap CPUs together in a system. This means that microprocessors offer a better price/performance than mainframe.
- **Speed**: A collection of microprocessors not only can give better performance than mainframe, but may give an absolute performance that no mainframe can achieve at any price.
- **Inherent Distribution**: Some applications built as distributed systems with separated machines.
- **Reliability**: If one processor or machine crashes, the system as a whole can still survive.
- **Scalability:** Computing power can be added in small increments.

2.2.2 Disadvantages of Distributed Systems.

Despite the many advantages and strength mentioned, distributed systems still have some disadvantages and weakness relative to single processor systems:

- **Networking**: The distributed systems connected to each other by networks so they suffer from the networks problems. It could lose messages, intermittent and cut connections … etc. Once the system comes to depend on the network, its message loss or other problems can negate most off the advantages the distributed system was built to achieve, this problem is not found in the single processor systems.
- **Security**: The shared data that is described as an advantage of distributed system may turn out to be a two-edged sword. If people can share data, security is often a problem. Some users may enter to unauthorized data and penetrate the security rules.
- **Programming**: This problem occurred first time with distributed systems because it was different from normal programming. But now I think the problem is narrated and there are many experts on distributed programming (Tanenbaum, 1995).

2.2.3.Hardware Concepts:

In distributed systems there are several ways in which the hardware can be organized. The structure depends on terms of how they are interconnected and how they communicate (Tanenbaum, 1995).

The most popular taxonomy of multiple CPU computer systems that have been proposed is Flynn's classification. Flynn selected two characteristics that he considered essential: the number of instructions stream and the number of data streams. Four classifications he proposed are (Kumar et al, 2003):

Single Instruction Stream and Single Data stream (SISD) : All traditional uniprocessor (having one CPU) computers fall in this category , from personal computers to mainframes.

- **Single Instruction Stream and Multiple Data Stream (SIMD):** Refers to an array of processors with one instruction unit that fetch an instruction and then commands many data units to carry it out in parallel, each with its own data.
- **Multiple Instruction Stream and Single Data Stream (MISD):** Pipeline computer is one of the famous examples of this type.
- **Multiple Instruction Stream and Multiple Data Stream (MIMD):** A group of independent computers, each with its own program counter, program and data. All distributed systems are MIMD (Kumar et al,2003).

MIMD Computers are divided into two groups: Multiprocessors, with shared memory, and Multicomputers, with independent memory for each. Figure-1 shows the MIMD divisions (Tanenbaum, 2002).

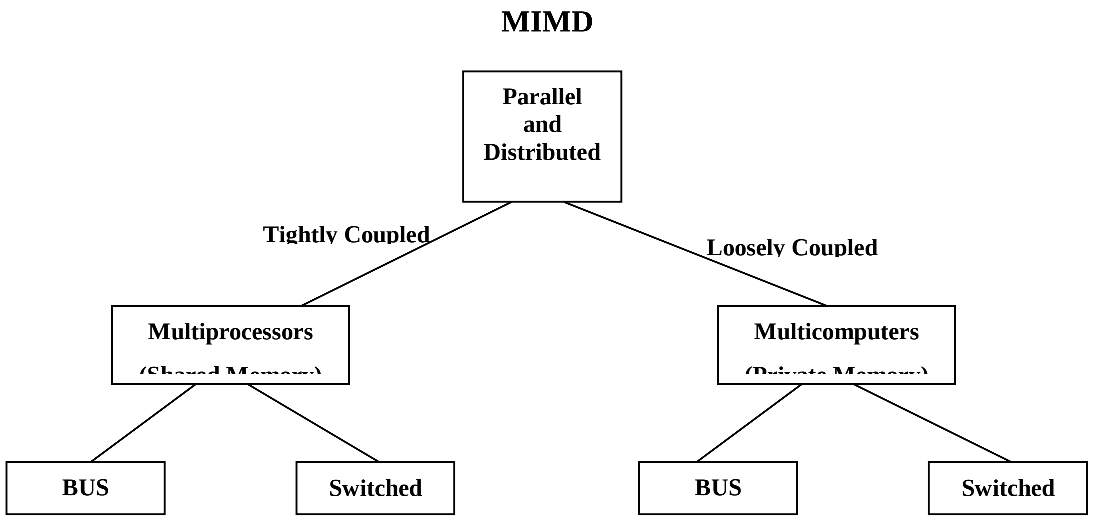


Figure-1 Parallel and distributed systems

Another dimension of taxonomy is that in some systems, the machines are tightly coupled and in others they are loosely coupled. In tightly coupled systems, the data rate is high and message delay from one computer to another is short. In contrast loosely coupled systems, the message delay is large and the data rate is low.

Multiprocessors tend to be more tightly coupled than multicomputers. Parallel systems are used as tightly coupled systems and distributed systems are used as loosely coupled systems. Figure-2 and Figure-3 show Multicomputers and Multiprocessors designs (Tanenbaum, 1995).

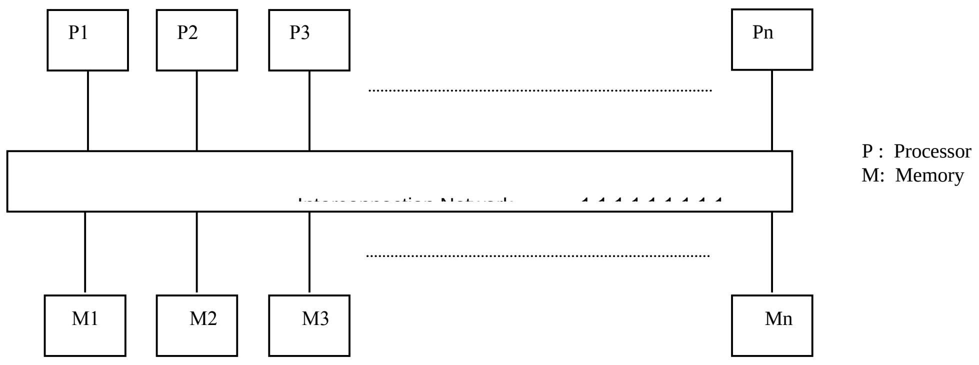


Figure-2 Multiprocessors design

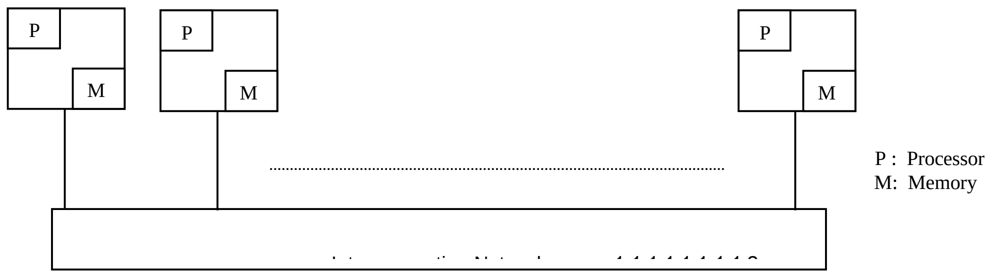


Figure-3 Multicomputers Design

2.2.4.Software Concepts.

The software in distributed systems is very important as the hardware. It presents system image to users, and it controls the way of thinking and the behavior of the system. This section presents the various types of operating systems software goes with which type of hardware (Tanenbaum, 1995).

2.2.4.1 Network Operating System.

Network operating system is loosely coupled software over loosely coupled hardware. Most organizations use this type of operating system. A workstations connected by LAN are a typical example for this type. One approach of network operating system is to provide global file system accessible from all workstations. This approach called Clint-Server systems (Tanenbaum, 2002, Kumar et al, 2003).

2.2.4.2 True Distributed Systems.

In network operating systems the system consists of many computers. Each can run its own operating system and whatever its user wants. There is no coordination at all, except for the rules that client-server traffic obey the system's protocol.

True distributed operating systems is found in tightly coupled software over loosely coupled hardware. The user does not feel of distinct machine. The machines act like a virtual uniprocessor. The essential idea in distributed systems that users shouldn't have to be aware of multiple CPUs in the system (Tanenbaum, 2002).

2.2.4.3 Multiprocessor Timesharing Systems.

In multiprocessors more than one CPU use the same shared memory. This type of systems is tightly coupled software on tightly coupled hardware. The characteristics of this class of systems is the existence of a single run queue: A list of unlocked processes that are ready to run. This run queue is a data structure kept in the shared memory (Tanenbaum, 2002)..

An area in which multiprocessor differs from network or distributed systems is in organization of the file system. The operating system contains a file system, including a single block cache. When any process executes a system call, a trap is made to the operating system which carried it out, using monitor or semaphore to lock out other CPUs while critical sections are being executed. On the whole the file system is different from single processor system. In fact, on some multiprocessors one of the CPUs is dedicated to running the operating system; while the other ones run user programs. So the operating system machine is often a bottleneck (Tanenbaum, 2002).

2.2.5 Design Issues.

The designers of the distributed operating systems must deal with some issues that are related to hardware and software points of view.

2.2.5.1 Transparency.

To achieve transparency the designers should make the users feel that the distributed system is similar to a single machine. Transparency can be achieved at two levels. The first level, is the easiest, in this case it is aimed to hide the distribution from the users. The second level is hard, in this case it is aimed to hide distribution from programmer. The concept of transparency can be applied to several aspects of a distributed system as follow(Tanenbaum, 2002):

- Location transparency: The users cannot tell where the resources are located.
- Migration transparency: Resources can move at will without changing their names.
- Replication transparency: the users cannot tell how many copies exist.
- Concurrency transparency: multiple users can share resources automatically.

- Parallelism transparency: Activities can happen in parallel without users knowing.

2.2.5.2 Flexibility.

The ability to add, delete and modify services to the system. The distributed systems use two types of kernels: monolithic kernel and micro kernel. The second kernel gives more services than the first but the first kernel is faster and small. Micro kernel provides four minimal services:

- An interprocess communication mechanism.
- Some memory management.
- A small amount of low-level process management and scheduling.
- Low level input/output.

The monolithic kernel provides the file system, directory system, full process management and much of the system handling (Tanenbaum, 2002).

2.2.5.3 Reliability.

Distributed systems are more reliable than single-processor systems. The idea is that if a machine goes down, one of the other machines takes over the job. Availability refers to the fraction of time that the system is usable. An availability can be enhanced by a design that does not require the simultaneously functioning of a substantial number of critical components. Redundancy is another tool for improving availability.

Highly reliable system must be highly available, but that is not enough. Data must not be lost in any way. So files must be stored redundantly on multiple servers, all copies must be kept consistent security is another factor of reliability. Files and other resources must be protected from unauthorized usage (Tanenbaum, 2002). Fault tolerance is also another issue of reliability. This dissertation presents a solution for this problem using leader election algorithm

2.2.5.4 Performance.

Performance is the most important issue in distributed system. Building a distributed system with all the above issues without achieving better performance than single processor does not mean any improvement. Response time, throughput (number of jobs per hour), system utilization and network capacity are performance metrics.

Communication between processors, which is essential in distributed system and absent in single processor, is typically quite slow. The best way to gain performance is to have many activities running in parallel on different processors and minimize the number of message sending over the network (Tanenbaum, 2002).

2.2.5.5 Scalability.

This issue takes into considered the system growth. The distributed system must be capable of expanding to some limit that causes improvement in operations (Tanenbaum, 2002).

2.3.Interconnection Networks

The communication process between distributed computers, in its two types: multicomputers 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 (Duato et al , 1997). Interconnection networks in parallel and distributed networks have four types:

- Direct networks.
- Indirect networks.
- Shared-Medium Networks.
- Hybrid networks

2.3.1 Direct Networks:

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).

2.3.1.1 Direct Networks Types.

2.3.1.1.1 Completely Connected Networks.

Each processor in this type has a direct line to each one of the other processors. The advantage of this type is that each processor can send a message to any processor by one step. The disadvantage is the high number of links when the processors increase

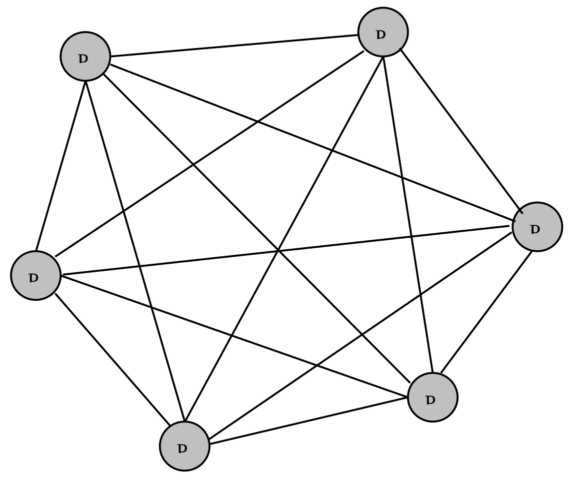


Figure -4 Complete Networks

2.3.1.1.2 Star Connected Networks.

One processor in the center of the network is connected to all the others. Contention and single point failure are the disadvantages of this topoLogy Figure-5 show the star network (Culler et al, 1999; Kumar et al, 2003).

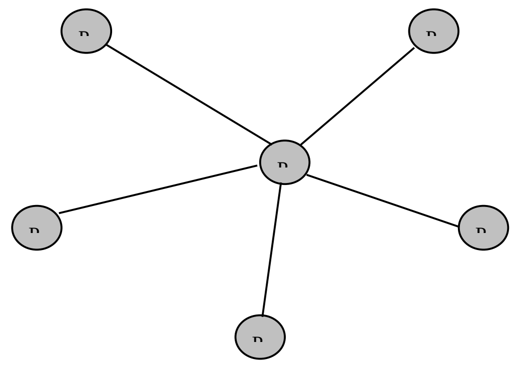


Figure-5 Star Networks

2.3.1.1.3Linear and Ring Arrays.

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 (Culler et al, 1999;Kumar et al,2003).

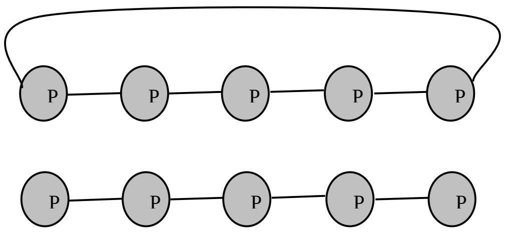


Figure-6 Ring and Linear Arrays

2.3.1.1.4Tree Networks.

In tree network there is only one path between any two processors. Linear and star networks are considered as types of tree networks. The message is sent by source node to higher levels until it reaches the root node of sub tree that contains source and destination nodes. Root node sends the message down toward the destination.

Tree network suffers from contention, especially in the higher levels, when the left side needs to interconnect with the right side. This problem can be minimized by increasing the number of links between nodes to make what is called a Fat tree network. Figure-7 show the tree network (Culler et al, 1999;Kumar et al,2003).

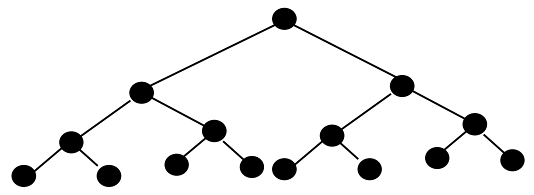


Figure-7 Tree Network

2.3.1.1.5 Mesh networks.

This topology is one of the main networks used in parallel computers. Two and three dimensional meshes are shown in figure-8.

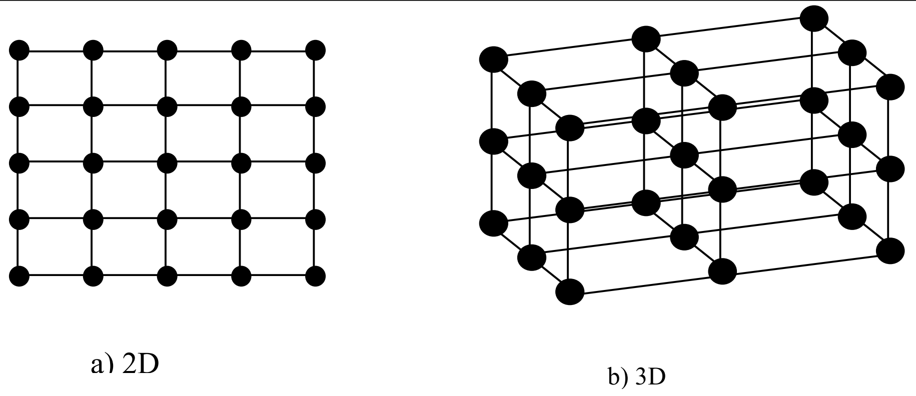


Figure-8 2D and 3D Meshes

Meshes have many desirable properties like scalability, modularity and simple construction (Culler et al, 1999; Kumar et al, 2003).

2.3.1.1.6. 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 make all nodes connect to the same number of neighbors Figure-9 shows two dimensional torus (Culler et al, 1999;Kumar et al,2003)..

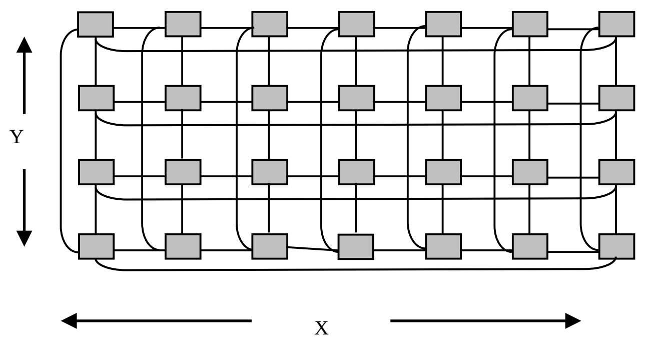


Figure-9 2D Torus Network

2.3.1.1.7 Hypercube Networks.

This topology has N processors which equal 2^d (d is the hypercube dimension). Each processor connects to d neighbors, Figure-10 shows different hypercube network (Culler et al, 1999;Kumar et al,2003).

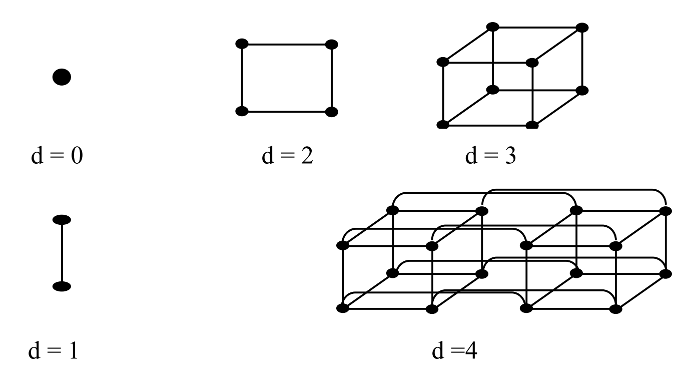


Figure-10 Hypercube Networks

2.3.1.1.8 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(Log((P+1)/2)), in mesh without wraparound it is ($P^{1/2}$-1). The diameter in wraparound meshes is $2\lfloor P^{1/2}/2 \rfloor$. For the hypercube the diameter is 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 (PLog(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)).

2.3.2. Indirect Networks.

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

1. **Bus-Based Networks**. The processors are connected to the memory by general path called Bus. Bus-based is simple in design and structured. Figure-11 shows bus-based network. The disadvantages of this type are the heavy load on bus or contention and scalability. When the number of processors increases this problem appears clearly so cache memory was added to each processor to decrease the need for main memory (Culler et al, 1999;Kumar et al,2003).
2. **Crossbar Switching Networks.** In this type a number of processors P are connected to a number of memory banks M. Figure-12 explains this connection (Culler et al, 1999;Kumar et al,2003).

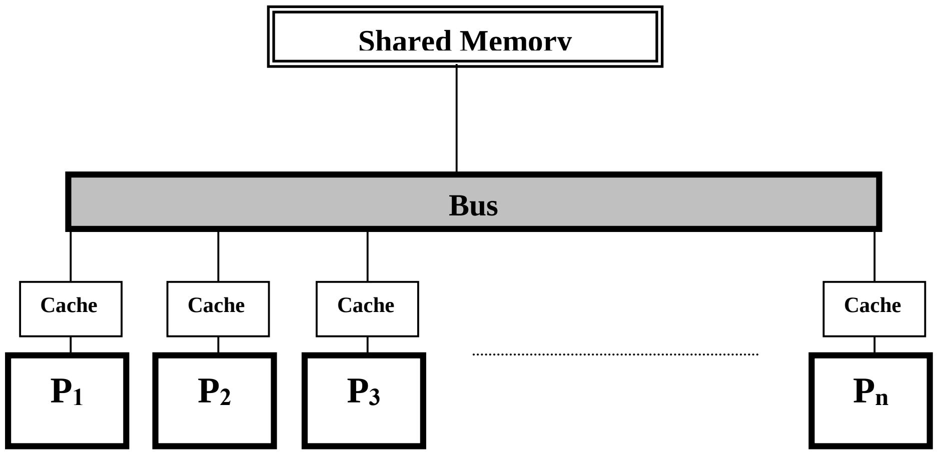


Figure-11 Bus-Based Network

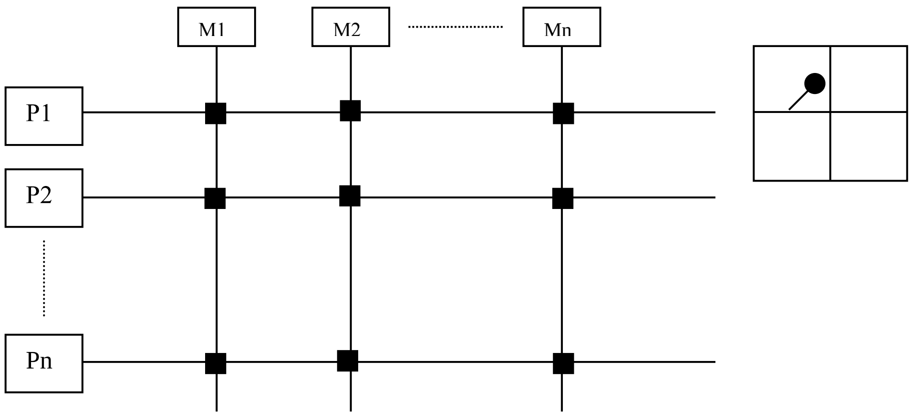


Figure-12 Crossbar Switching Network

3. Multistage Interconnection Networks.

Bus networks is scalable in terms of cost and unscalable in terms of performance. Crossbar interconnection is scalable in terms of performance and unscalable 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-13 shows commonly used interconnection network (Omega network). An omega network has (PLog P)/2 switching nodes (Culler et al, 1999;Kumar et al,2003).

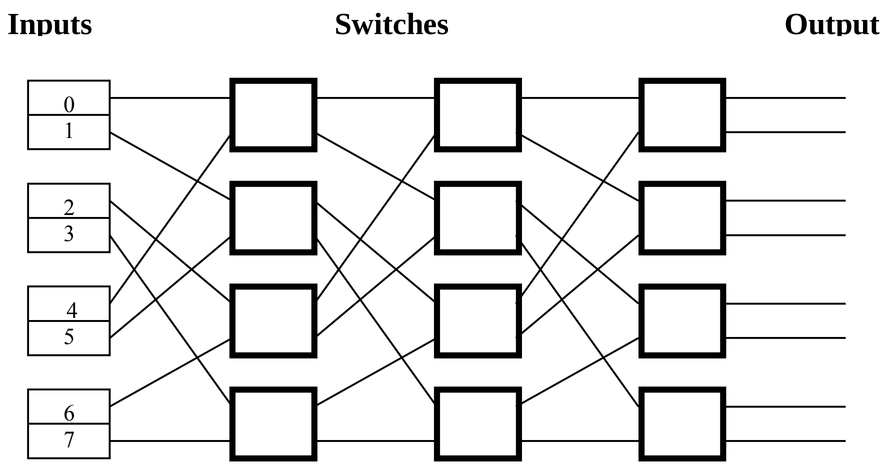


Figure-13 Omega Interconnection Network

2.3.3. Shared-Medium Networks.

This type depends on shared medium made from fiber optic or metal. This type doesn't allow more than one machine to use the network at the same time. It has the ability of Carrier Sense Bus and Collision Detection. Ethernet is an example of this type(Duato et al ,1997).

2.3.4. Hybrid Networks.

Hybrid network is a combination of two types or more to increase the speed of sending messages (Duato et al,1997).

2.4. Routing Mechanism for Direct Networks.

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. This section explains the classification for routing mechanisms and routing types.

2.4.1. 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:

  Centralized Routing. The routing is controlled by one router called centralized controller. This type suffers dead lock and single point failure problems.

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

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

Hybrid Routing. The routing process is between Source routing and distributed routing (Duato et al, 1997).

- **Number of Addresses Factor.**

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

1. **Unicast Routing.** Message carries one address on this mechanism. It uses point-to point communication between processors. Although, the simplicity of this type, time to complete the communication process is the main disadvantage (Panda et al, 1994; Afek and Gafni ,1991).
2. **Multidistination Routing.** Message carries more than one address on this mechanism. It uses one-to-many communication between processors (Panda et al, 1994).

2.4. 2.outing Types.

There are two types of routing mechanism: Deterministic routing and Adaptive routing.

- **Deterministic Routing.** A unique path for the messages is determined, based on source and destination. It doesn't use any information regarding the state of the network (Culler et al , 1999). Congestion is the main disadvantage of this type.

- **Adaptive Routing.** This type uses the information regarding the current state of the network to determine the path of the message (Larrea et al, 1999). Adaptive routing detects the congestion in the network and routes around it.

2.4.3 Problems in Routing Mechanisms.

Distributed algorithms need to exchange and transfer messages. Message faces some problems during the route to its destination.

- **Deadlock Problem**. This occurs when the message stops, waiting for an action that can't happen (Culler et al, 1999). This means that the message reaches a locked path. Deadlock occurs when there is a closed cycle in the network. This problem is solved when the cycle is opened (Duato et al, 1997). Designers can avoid this problem by synchronizing the steps in the overall algorithm.
- **Live Lock Problem.** This occurs when a message can't reach the destination because the channel in between is busy (Duato et al , 1997).
- **Contention Problem**. This occurs when there is more than one message attempting to use the same channel. Only one can succeed and remaining have to wait (Culler et al, 1999).

2.5..Algorithms Design and Analysis

An algorithm is a sequence of nonambiguous instructions for solving a problem in a finite amount of time (Levitin, 2003). An input to an algorithm specifies an instance of the problem the algorithm solves. Algorithms can be specified in a natural language or a pseudo code; they can also be implemented as computer program (Rechard and Kumarss, 2004).

2.5.1.Complexity and Order Analysis

To analyze the performance of algorithms we use order analyses and Complexity of functions:

There are tree types of functions as follow (Kumar et al, 2003; Rechard and Kumarss, 2004; Levitin, 2003):

1. Exponential function: A function *f* from reals to reals is called an Exponential function in *x* if it can be expressed in the form

   $f(x) = a^x$ for x, a ∈Ж (the set of real numbers) and a>1.such as 2^x,1, 5^{x+2}

2. Polynomial functions: A function *f* from reals to reals is called a polynomial function of degree b in x if it can be expressed in the form $f(x) = x^b$ for x,b ∈Ж and b > 0. A linear function is a polynomial function of degree one and a quadratic function is a polynomial function of degree two. Examples of polynomial functions are 4 , 3x , $3.5x^{5.3}$.

3. Logarithmic functions: A function *f* from reals to reals that can be expressed in the form $f(x) = \text{Log}_b x$ for b ∈Ж and b > 1 is logarithmic in x.

Most functions in this dissertation can be expressed as sums of two or more functions. A function f is said to dominate g if f(x)grows at a faster rate than g(x). Thus, function f dominates function g if and only if f(x)/g(x) is monotonically increasing function in x. An exponential function dominate a polynomial function and a polynomial function dominates a logarithmic function.

2.5.2. Order Analysis of Functions.

In the analyses of algorithms, it is often impossible to derive exact expressions for parameters such as run time, speedup, and efficiency. In any cases an approximation of the exact expression is adequate (Power, 1999). The approximation may indeed be more illustrative of the behavior of the function because it focuses on the critical factors influencing the parameters. This approximation can be expressed by the order analyses of the expression.

The O Notation: O notation is defined as follows: given a function g(x) , f(x) = O g(x) if and only if for any constant c>0 , their exists an x_0>=0 such that f(x) <= cg(x) for all x >= x_0 (Neapolitan and Naimipour, 2004; Levitin, 2003; Kumar et al, 2003).

The Ω Notation: The O notation sets an upper bound on the rate of growth of function . The Ω notation is the converse of the O notation.

Given a function g(x), f(x) = Ω g(x) if and only if for any constant c>0 , their exists an x_0>=0 such that f(x) >= cg(x) for all x >= x_0 (Levitin, 2003;Kumar et al, 2003; Rechard and Kumarss, 2004).

The θ Notation: θ notation is defined as follows: given a function g(x) , f(x) = **θ**(g(x)) if and only if for any constant c_1,c_2 >0 , their exists an x_0>=0 such that:c_1g(x) <= f(x) <= c_2g(x) for all x >= x_0 (Levitin, 2003;Kumar et al, 2003; Rechard and Kumarss, 2004).

2.5.3. Worst-Case, Best-Case, and Average-Case Efficiencies.

Worst case efficiency analysis provides very important information about an algorithm's efficiency by bounding its running time from above. The worst case analysis guarantees that for any instant of size N, The running time (in our algorithms time steps and the number of messages) will not exceed its running time on the worst-case. The best case efficiency of an algorithm is its efficiency for the best case input of size N. The analysis of the **best case** efficiency is not as important as that of the worst case efficiency. But it is not completely useless, either. It should be clear that neither the worst case analysis nor its best case counterpart yields the necessary information about an algorithm's behavior on a random input. This is the information that the **average case** efficiency seeks to provide. We must make some assumptions about possible inputs, to analyze the algorithm's average case efficiency.

2.6 Model Description and Properties

2.6.1 Introduction

The computational model is hypercube network. That means interconnection topology of the network is a hypercube graph with N = 2^d nodes (d is the dimensions of the hypercube). This section explains the model description, properties and design assumptions for this research.

2.6.2 Model Description

A d-dimensional hypercube network is represented by N-nodes labeled by d binary bits from (0 to (2^d -1)) (d equivalent to Log(N)). These nodes are connected by (N Log(N))/2 bidirectional links. A zero dimension hypercube has one node, one dimension has two nodes and two dimensions has four nodes …etc.

The difference between any two-neighbor nodes is only in one bit in the labels. The distance between any two nodes equal the Hamming distance between their canonical labels. The diameter of the hypercube equal (Log(N)). The shortest path between any two nodes is less than or equal (Log(N)). This path can be founded by using Exclusive Or (EXOR) operation between the source label and destination label. The Hypercube graphs have an elegant recursive structure. To construct a labeled (d+1) dimensional hypercube, take two d-dimensional hypercube and extend all labels in the first one with a 0, and all labels in the second with 1. For each process in the first one adds an edge (of direction d) to connect it with the associated node in the second hypercube.

Each node in the hypercube is connected to its neighbors by Log(N) links. These links can be labeled to add a useful configuration to the model. Each link connects two nodes. Link label is obtained from the nodes labels. The order of the different bit in the two nodes is the link label. This order is from 1 to Log(N). Figure-14 shows the 3-dimension hypercube with the labels for its nodes.

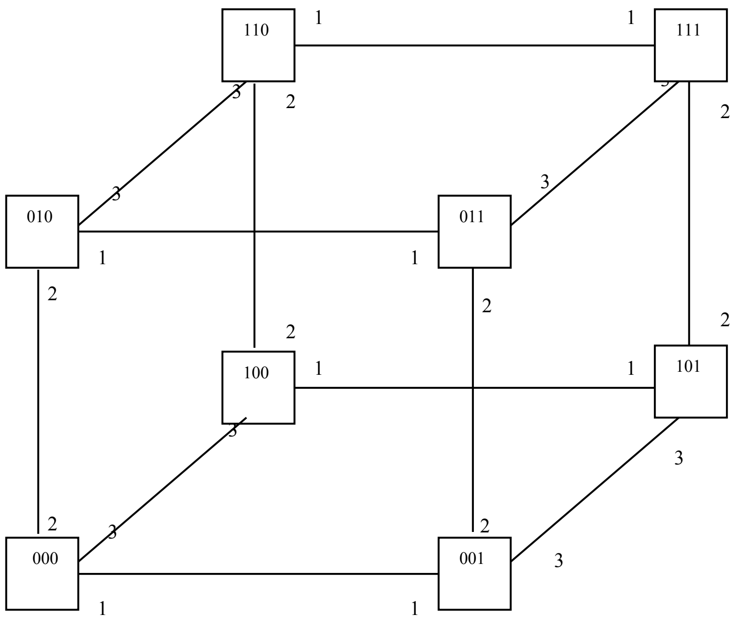


Fig-14 3 Dimension Hypercube

2.6.3 Model Properties.

Hypercube has some advantages that make it one of the preferable topologies. Diameter of the hypercube is Log(N). Node labels in binary code help in routing techniques. The number of links is (N Log(N))/2. Hypercube is an attractive Structure for parallel processing due to its symmetry and regularity (Castorino and Ciccarella, 1999). In fact it has been shown to be a very versatile and robust architecture capable of executing several efficient parallel algorithms; besides this topology is a suitable architecture for design of both tightly coupled parallel systems and distributed systems (Ciccarella and Patricelli, 1994). On the other hand the increase in number of links with the increase in number of nodes is the main disadvantage in hypercube.

2.6.4 Research Assumptions.

This Research assumes the following:

1. All the communication links are bidirectional**.**
2. 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.
3. Routers should work all the time even with fault node because the fault is in leader properties.
4. To solve leader failure problem, each node calculate 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.
5. Each node has a distinguished ID. The election algorithm depends on this ID.
6. The fault node shared in the election with ID = 0, so it can not win the election.

7. One intermittent link failure is recoverable.
8. The worst case for the algorithm is when the leader failure is detected by more than one node (concurrent failure). This case becomes complicated when the failure is detected by N-1 nodes.
9. 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.

2.7 Conclusion.

Chapter three presented an introduction to distributed systems and parallel computers. Distributed systems were described, its hardware and software were explained. The chapter also presented advantages and disadvantages of distributed systems against single processor computer. Interconnection networks and routing mechanisms were explained. The chapter finished by the description of hypercube model and proposed algorithms assumptions.

CHAPTER THREE
LITERATURE REVIEW

3.1 Introduction.

The leader election problem has been studied by a number of researchers. In all these studies the researchers presented different methods to deal with the leader election problem. As it will be shown, each researcher only deals with one issue of the problem and presents a solution to this problem.

In distributed systems the major problem that faces the researchers and scientist is the leader failure and the relevant leader election algorithm. Election algorithms will vary based on (Fokkink .W and Jun Pang, 2004; Junguk L. and Geneva G., 1996; Signh,1997; abu-Amara,1988):

- Communication mechanism used (synchronous vs. asynchronous), (wired vs. wireless).
- Node names (unique identity vs. anonymous).
- Network topology (ring, tree, complete graph, meshes, torus and hypercube).
- The nature of the algorithms (dynamic vs. Static).
- Some of the previous work dealt with the link failure.

Most of the previous researches were based on mathematical proof to verify their algorithms. They used the big O notation to obtain the complexity of the number of messages and time steps, which represent the domain factors of the algorithm complexity (Fokkink .W and Jun Pang, 2004; Junguk L. and Geneva G., 1996; Signh,1997; Valeric et al,2001; Russell,1999). Other researchers used simulation to validate their algorithms (Devillers et al, 2004; Vos,2000) .

Leader election solution was first thought of at the end of the seventies, it was started by the ring and complete networks (Leeuwen and Tan, 1987). In the nineties meshes, hypercube and tree were studied. To date all these topologies and wireless networks are still being studied.

This chapter will list the most relevant researches and compare them to work being proposed.

3.2 Related Works

- Molina-G. , **Elections in a Distributed Computing Systems,** 1982:This study presented an algorithm to solve the link failure for complete network. When a node notice that the leader is no longer responding to requests, it initiates an election . A node P, Holds an election as follows (Molina,1982):

    1- P sends an election message to every node with the higher number.

    2- If no one responds, P wins the election and becomes the leader.

    3- If one of the higher numbers answers, it takes over P's job.

    4- At any time the old leader recovers it takes over the leadership so this algorithm is called Bully. (Garcia-Molina, H ,1982)

Bully algorithm is considered a basic algorithm for election problem. It is listed in most operating systems and distributed systems books as an example of leader election algorithms.

- Fredrickson and Lynch, **Election of a Leader in Asynchronous Ring**, 1987. This study assumes the nodes are physically ordered when the nodes represent processor and logically ordered when it represent processes, so that each node knows who its successor is. The election message is built when any node notices that the leader is not functioning. The node sends messages containing its number to the successor. If successor is down, the sender will skip over it and go on to the next number along the ring. During each step the sender adds its own number to the list in the message. Eventually, the message gets back to the node that started it all. That node recognizes this event when it receives an incoming message containing its own node number. At that point, the node sends a leader message to inform all the nodes about the new leader.

- Gerard, **Linear Election for Oriented Hypercube, 1993:** This study proposes an election algorithm for oriented hypercube, where each edge is assumed to be labeled with its dimension in the hypercube. When N represents the size of the cube, the algorithm exchanges (O(N)) messages and uses (O(Log^{2}N)) time steps to solve the problem in the simple case, when one node detects the leader failure. In more complicated cases when the failure is detected by subset of the nodes, the time complexity is linear, and the algorithm terminates in (O(N)) time steps.

How this algorithm works:

This Gerard algorithm heavily relies on the recursive structure of hypercube graph. A hypercube of dimension (d+1) of two copies of the hypercube of dimension d, where each pair of corresponding nodes is connected with an edge of direction d. To elect a leader in this hypercube the algorithm first recursively elects a leader in both of constituting hypercubes of dimension d, and then proceeds to level one of the two leader nodes. To avoid confusion between leadership at different levels of recursion, a node is called a d- leader if it won the election in a d-dimensional hypercube. The base case of the algorithm, election in hypercube of dimension 0, is easy; the network consist of exactly one node, which becomes a 0-leader immediately. To complete the election, the algorithm passes in two phases the tournament and matching phases:

1- **Tournament**: After the election of d-leader in the two d-dimensional sub-hypercube, the nodes must cooperate to elect one of these to become the (d+1)- leader. A tournament between two nodes that can communicate directly is easily organized. Each node send the other one a message containing its name. The nodes with greater ID becomes the leader and the other becomes not-leader. But the difficulty is when the nodes don’t know how to reach the leader in the other side, or even their own. As the first step, node P that becomes d-leader sends a tournament message (**tour**, p, d) through its edge d. It is the responsibility of the receiving node called entry node to forward this message to its d-leader in d steps.

2- Match- Making Technique

To make a match between the d-leader and the entry node , the d-leader announces its leadership to all nodes in (d/2) dimensional face, referred to as the leaders row. The entry node broadcast the tournament message through a (d/2) dimensional face called its column. There is one node, called match node, that receives both messages.

The election algorithm in this research needs O(N) time steps. This time is greater than the time steps in the proposed algorithm in this dissertation which is equal O(Log(N)).

- Abu-Amara and Loker, **Election in Asynchronous Complete Networks with Intermittent Link Failures**, 1994: This study considers the problem of a fault tolerant leader election in asynchronous complete (fully connected) distributed networks. It assumes that the processors are reliable, but some of communication channels may fail intermittently before or during the execution of the algorithm. Channel failures are undetectable due to asynchronous nature. When N represents the number of processors in the network,

- and F represents the maximum number of faulty channels on each processor, where (F <= (N-1)/2). This algorithm uses at most ($O(N^2+NF^2)$) messages to elect a unique leader.

This work results are not affected or related to proposed algorithms, but it presents a solution of leader failure problem with the presence of link failure in complete networks. The second proposed algorithm in this dissertation solves the same problem in hypercube network.

- Antonoiu and Srimani, **A Self-Stabilizing Leader Election Algorithm for Tree Graphs** 1996. This work proposed a self-stabilizing algorithm for leader election in a tree graph. It doesn't assume the nodes are assigned unique identification numbers. Each node maintains an ordered list of its neighbors and the predecessor pointer to point to one of its neighbors or null.When the algorithm terminates (in finite time), there is a unique node with a level value that is strictly greater than the levels of all other nodes; this is the elected leader node and each of the rest of the nodes has a unique way to reach that leader. The nodes in the tree are treated uniformly in the sense that each node executes a single uniform rule. Each node has only a partial view of the global state; it knows its own state and the states of its neighbors. Starting from any illegitimate state, the algorithm can elect an arbitrary internal node to be the new leader; but no leaf node will ever be selected as the leader of the tree (a leaf node in a tree is a node with

- exactly one neighbor). The work shows that it may not be possible to design a self stabilizing protocol that can elect a leaf node to be the leader.

The work didn't present analyses for the number of messages and time steps, it presented just the steps and description of the algorithm.

- Flocchini and Mans, **Optimal Election in Labeled Hypercube, 1996:** In this work the election problem in hypercube networks was studied, by using two models with sense of direction, the dimensional and the distance models. The proposed algorithm needs ($\theta(\text{Log}^3 N)$) time steps using ($\Omega(N)$) messages(Flocchini and Mans,1996).

Algorithm Description: The algorithm Proceeds in phases of tournament. The idea is to halve the number of computing processors on each phase such that on overall LogN phases; or steps. Initially all nodes are duelist (0). The algorithm terminates after log(N) steps with a single Duelist (duelist(Log(N)) and all the other nodes are second. In every phase of the algorithm, each successful Duelist goes for the next duel by challenging its respective Duelist. The algorithm is based on repeated sequences of a duel (namely a combination of two cubes into a large one) and, hence takes Log(N) steps (one per dimension).

At each step K, each duelist has to challenge (sends an attack) and to be challenged (receives an attack) by its respective duelist in the rank of the tournament (the duelist node in its cube image according to the dimension

K). The opposite duelist handshake: the duelist with the greater identity value wins the duel of step K and becomes duelist of level K+1. The duelist with the smaller identity value loses and becomes a Second of level K and keeps the path of its winner. Neither acknowledgement nor surrender messages are required. The task of a duelist is to fight a duel, whereas the task of a second is to relay an incoming attack to its duelist.

When an attack From duelist(S) reached a Second(K) , (with K< S), the second forwards it to the nodes that will be known as duelist(K+1). If this node has been defeated and became a second, it forwards it to its respective duelist(K+2). The process is repeated until the duelist(S) is found.

- Singh G., **Leader Election in the Presence of Link Failures**, 1996: This research proposes a protocol for leader election tolerant to intermittent link failure in the complete graph network. It assumes that up to (N/2 – 1) links incident on each node may fail. It assumes that up to (N^2/4 – N/2) links overall the system may fail. In this case nodes represent the processors and edges represent bi-directional communication channels between the processors. In this problem, it is assumed that initially all nodes are passive, an arbitrary subset of nodes, called the candidate nodes, wake up spontaneously and start the protocol. On the termination of the protocol exactly one node must announce itself the leader. The protocol depends on the fact that for any pair (i,j) of nodes, there exists a node k such that both i and j have no faulty link to k .

- The protocol composed of iterations. Iterations composed of phases. When the iterations reach (Log(N) + 2) the node is the leader. The message complexity of the protocol is (O (N^2)).

This work results related to proposed algorithms, but it presents a solution of leader failure problem with the presence of link failure. The second proposed algorithm in this dissertation solves the same problem in hypercube network.

- Dolev S., Israeli A. and Moran S,(1997)., **Uniform Dynamic Self-Stabilizing Leader Election**.

  This research studies uniform dynamic self stabilizing protocols for leader election under read/write atomicity. Protocols use randomizes to break symmetry. The leader election protocol stabilizes in (O(Δ DLog(N))) time when the number of processors is unknown and (O(Δ D)),otherwise. Here Δ denotes the maximum degree of nodes, D denotes the diameter of the graph and N denotes the number of processors in the graph.

- Stefan D. and Peter R., **Linear Broadcasting and (N (Log (Log(N)))) Election in Unoriented Hypercube,** 1997**.** This work provides two algorithms for broadcasting and leader election in unoriented hypercube. When N represent the number of vertices in

- the hypercube, (O(N)) broadcasting and traversing algorithms are introduced. The second algorithm design an O(N(Log(Log(N))) messages Leader election algorithm in unoriented hypercube.

This research ignored the effects of time steps and focused on the number of messages to get the best results. In our dissertation the proposed algorithms attempts to compromise between time steps and number of messages were done to get better time steps than the other algorithms in hypercube.

- Castorino A. and Ciccarella G.,(1999), **Optimal-Election Algorithms for Hypercube**. This research studies leader election algorithm for hypercube networks. It has presented two algorithms based on the hypothesis that the election process may be started by one processor, so the cope with a typical situation that may arise in several distributed systems. In this research the time complexity is OLogN and the number of messages is O(N) messages.

  The first algorithm in this paper, which is called, Round trip algorithm is divided into two phases: in the first one the nodes cooperate to elect a leader (election phase) till one of them obtains the result of the election process. In the second phase (Broadcasting phase) the node that possesses the result broadcast all the other nodes. In the second algorithm which is called Parallel Election Algorithm it composed of two phases as in the first algorithm; but the difference is that the election process is started by two nodes in different hypercube.

The difference between our proposed algorithm and these algorithms besides the complexity is that this paper doesn't take the worst case when the leader failure is detect by all nodes.

- Yamshita M. and Kammeda T., **Leader Election Problem on Networks in which Processor Identity Numbers are not Distinct,** 1999.

  This research solve leader election problem in a network in which processors identity are not distinct. It uses a simple network graph which has neither self loops nor parallel edges. Performance evaluation depends on the number of messages and message length. When the message length is (O Log(N)) the number of messages is bounded by (O(N | E |)) , E is number of edges. This work doesn't relate to proposed algorithms because the proposed algorithms solve the problem when processor Identity numbers are distinct

- Navneet M., Jennifer L., Welch, Nitin V., **Leader Election Algorithms for Mobile Ad Hoc Networks,** 2001: This research presents two new leader election algorithms for mobile ad-hoc networks. The algorithms ensure that eventually each connected component of the topology graph has exactly one leader. The algorithms are based on routing algorithms called TORA.

- Sudarshan V., Decleene B., Immerman N. , Kurose J.,Towsley D., **Leader Election Algorithms for Wireless Ad Hoc Networks,** 2003. This study proposes two cheat-proof election algorithms: Secure extreme Finding Algorithm (SEFA), and Secure Preference-based

- Leader Election Algorithm (SPLEA). Both algorithms assume asynchronous distributed systems in which the various rounds of election proceed in a lock-step fashion. SEFA assumes that all elector-nodes share a single common evaluation function that returns the same value at any elector-node when applied to a given candidate-node. When elector-nodes can have different preferences for a candidate-node, the scenario becomes more complicated. SPLEA deals with this case. Here, individual utility functions at each elector-node determine an elector-node's preference for a given candidate-node.

- Fokkink and Pang, **Simplifying Itia-Rodeh Leader Election for Anonymous Ring**, 2004. this work presents two leader election algorithms for anonymous unidirectional rings with FIFO channels, based on finite-state. They satisfy the property with probability 1, eventually exactly one leader is elected. The generation of state space and verification were performed on a1.4 GHz AMD Althlon Processor with 512 MB memory. This research solved the election problem in ring which is not affected or compared with dissertation algorithms.
- Jean-Franqois Marckert (2005), **Quasi-Optimal Leader Election Algoriths in Radio Network with Log-Logarithmic Awake Time Slots**.

  This work presents two leader election protocols for radio networks, in both telephone and walkie-talkie models, where the number of radio is unknown. Those randomized protocols are shown to elect a leader in (O(Log(N))) expected time, and to be energy-efficient due to a small total awake time of the radio stations.

3.3 Conclusion.

This chapter presented different types of leader election algorithms. Differences were in date of publishing or in the subject of the work. The Chapter gave a short description of related works, and compared it with proposed algorithms. Other comparisons will be done through dissertation pages.

CHAPTER FOUR
NEW LEADER ELECTION ALGORITHMS IN HYPERCUBE NETWORKS

4.1 Introduction

This chapter presents description for the two algorithms. The first algorithm, presents a new idea for leader election algorithms in hypercube with minimum time steps and number of messages. This algorithm is composed of three phases; the description for all three phases is presented in section 4.2. These phases and all steps within each phase are explained using a Pseudo code for the algorithm. The second algorithm, presents a new idea for leader election in hypercube with the presence of one link failure. The second algorithm is explained in section 4.3.

4.2 Proposed Leader Election Algorithm in Hypercube.

The leader election algorithms solve the problem of leader failure. As described in Chapter two this problem studied in different ways and several algorithms were designed with different complexity. This algorithm presents a solution for leader election in hypercube with minimum time steps and number of messages.

4.2.1 Algorithm Description

The proposed algorithm consists of three phases. Each phase has a number of time steps and messages. Phase one is initiated when one or more nodes detects the leader failure, this initiates an election process. This phase reduces the numbers of participated nodes in the election process to N/2 nodes who are aware of the election. The second phase uses the reduction all-to-one communication operation to have the result in one node that posses the address ($X_1 0_2 0_3 \ldots 0_d$) (X means 1 or 0 , d represents the hypercube diameter). When X = 0, the half of the hypercube with the most left bit in the nodes labels = 0 will continue the algorithm until the leader is known by ($0_1 0_2 0_3 \ldots 0_d$) and the other half halt the processing, and vice versa. Finally, in the third phase node address ($X_1 0_2 0_3 \ldots 0_d$) broadcasts the leader message to all nodes in the hypercube using broadcast one-to-all communication operation. During each step, in phase one and two, the received ID is compared with the local ID. The greater ID is passed to the next step. The orders of steps for the phases are as follows:

Phase One: This phase starts when one or more nodes detect a leader failure.

Each node detect the failure initiate the election:

Step1: send an election message to the node that differs in label in the first bit from the right through link 1. Election message is composed of (phase, step, winner ID, and winner position).

Step2: the sender and receiver in the previous step send an election messages to the two associated nodes that differ in the second bit from the right through link 2.

Step3: the senders and receivers from the previous step send an election messages to the nodes that differ in the third right bit through link 3. The next steps are the same until step Log(N) is reached.

In Step Log(N): N/2 nodes (the senders and receivers from the Log(N) –1 step) send an election messages to the nodes that differ in the Log(N) right bit through link Log(N).

During the execution of phase one, if the receiver is aware of the failure and is in progress with its own initiated election step, it will complete the greater step and terminate the smaller one. Each node receives the election message, compares its own ID with the received ID then completes the next step with the greater ID. Phase one ends after Log(N) steps, reducing the participant nodes to N/2. The leader ID and its position for the whole hypercube are within (Log(N) –1) dimension hypercube.

Phase Two: The second phase uses the reduction all-to-one communication operation to guide the result towards the node that have the address $X_1 0 \ldots 0_d$. This is achieved as follows:

Step1: nodes with the second left bit = 1 ($X_1 1 X_3 \ldots X_d$) send an election message to nodes with the second left bit = 0 ($X_1 0 X_3 \ldots X_d$) through link Log (N) – Step number.

Step2: the receivers in the previous step and third bits in its labels = 1 ($X_1 X_2 1 \ldots X_d$) send an election messages to the nodes that differ in this bit ($X_1 X_2 0 \ldots X_d$) through link Log (N) – Step number. The next steps are the same until step Log(N) -1 is reached.

Step (Log(N) – 1): the receiver in the previous step with Log(N) bit in its label equal 1 ($X_1 0_2 0_3 \ldots 1_d$) , sends an election message to the node that differ in the right most bit ($X_1 0_2 0_3 \ldots 0_d$) through Log (N) - step.

After the end of phase 2 the last node ($X_1 0_2 0_3 \ldots 0_d$) is the node that has all the information to deduce the leader ID.

Phase3: In this phase the node ($X_1 0_2 0_3 \ldots 0_d$) broadcasts a message containing the result of the election using one-to-all broadcast operation, the broadcasted message (leader message) contains new leader ID.

4.2.2 Examples

- Example One:

In this example leader election algorithm is applied using three dimensional Hypercube with eight nodes. Example Assumes that, node with label 101 detects leader failure; the algorithm will be executed as follows:

Step One: Node 101 changes step to 1, and sends an election message to node 100 through link label = 1 (Solid arrow in the figure 15).

Step Two: Nodes 101, 100 change step to 2 and select the greater ID. Then they send election message to nodes differ in the second bit from right (110,111) through link 2 (Dash arrows in the figure 15).

Step Three: Nodes (100,101,110,111) compare its ID with the received ID and select the greater, then change Step to 3 and send election messages to nodes (000,001,010,011) respectively (Dots arrows in figure 15). After step 3 which is equal Log N phase two is started in half of the hypercube.

3 time steps and 7 messages are needed for phase 1.

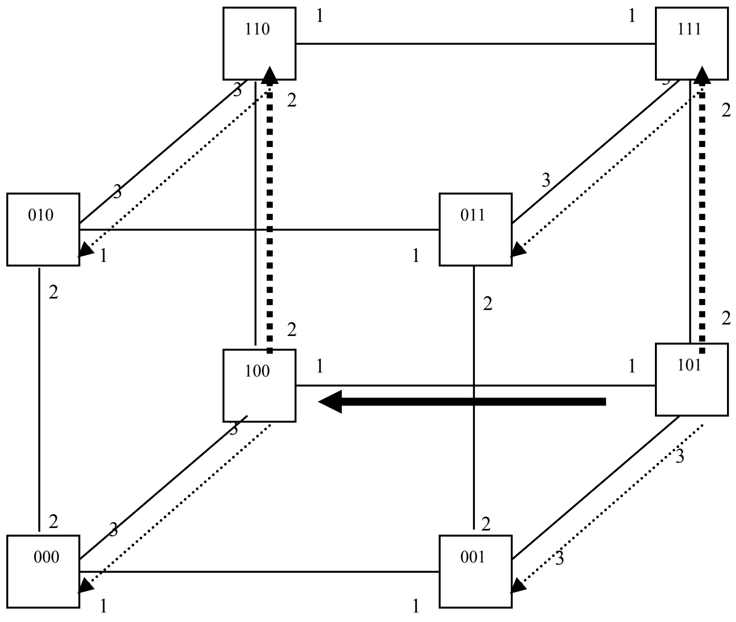


Figure 15 Phase One messages in Example One

Phase Two:

Step One: Nodes (010, 011) compare received ID with local ID and select the greater. Then they send election message to nodes (000,001). (Solid arrows in figure-16).

Step Two: Node 001 send winner ID to node 000. after step two which is step (Log (N)-1) the leader ID and position is Known by node 000. (Dashes arrow in figure-16).

1 time steps and 3 messages are needed to complete phase 2 in this example.

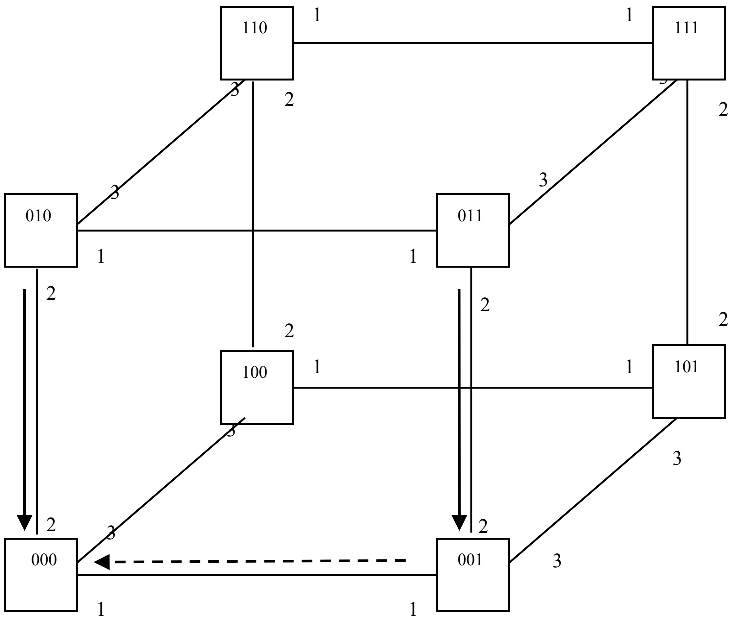


Figure-16 Phase Two messages

Phase Thre

Step 1: Node 000 send Leader message to node 001.(Solid arrow in Figure 17)

Step 2: Nodes 000,001 send leader messages to nodes 010, 011.(Dash arrows in Figure 17)

Step 3: Nodes 000,001,010,011 send leader message to nodes 100,101,110,111. after Phase three network return stable with new leader. (Dot arrows in Figure 17)

3 time steps and 7 messages to complete phase 3.

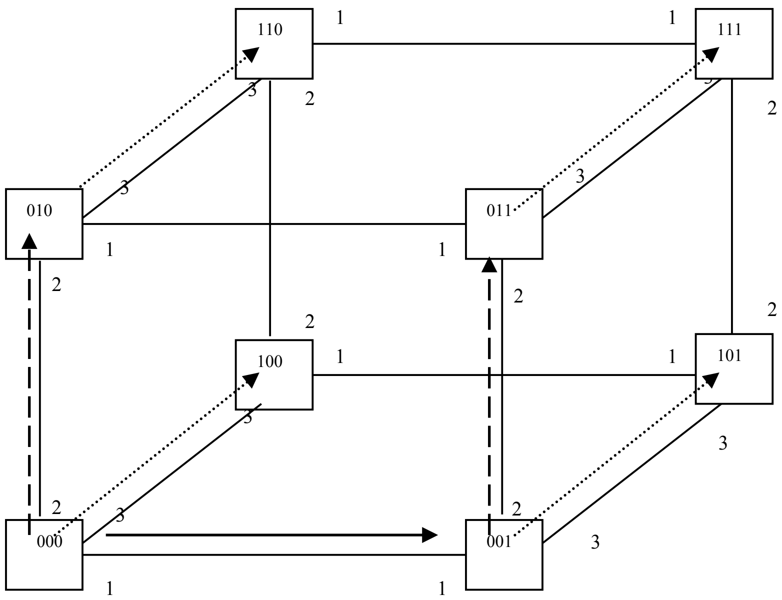


Figure-17 Phase Three messages in Example 1

Total number of time steps is equal:

3+2+3 = 8 time steps

Total number messages are equal:

7 +3 +7 = 17 messages

- Example Two:

This example assumes that nodes with labels 101,001, and 010 detect leader failure, the algorithm will executed as follows:

Step One: Nodes that detect the failure, change step to 1. After that it sends election message to nodes differ in the first bit from right through link label = 1, as follows:

101 sends election message to node 100

001 sends election message to node 000

010 sends election message to node 011

These messages are shown in solid lines in Figure 18.

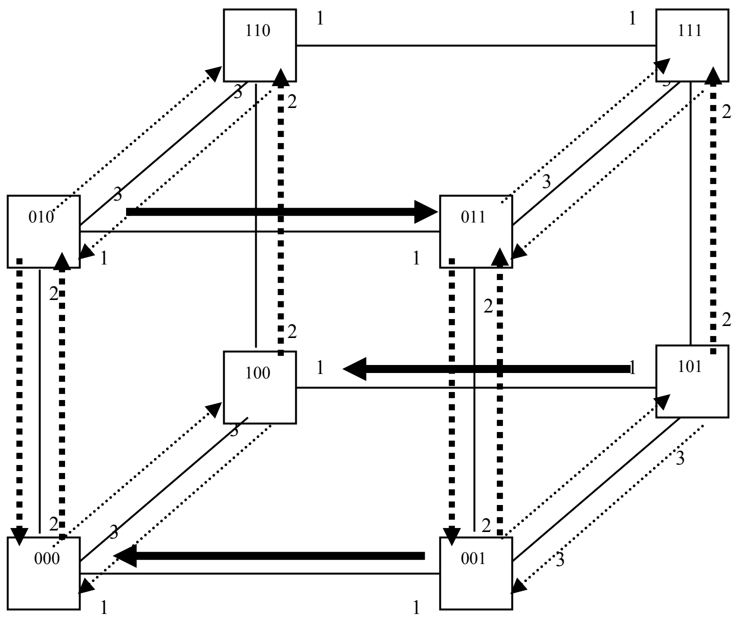


Figure 18: Phase One messages in example two

Step Two: Nodes 000,001,010,011,100and 101 change step to 2 and select the greater ID. Then they send election message to nodes differ in the second bit from right (010,011,000,001,110,111) respectively through link 2 (Dash lines in Figure 18).

Step Three: Nodes ,that Know about election, compare its ID with the received ID and select the greater, then change Step to 3 and send election messages to nodes differ in the third bit from right (Dot lines in figure 18). After step 3 which is equal Log N, phase two is started by the first half of the hypercube, while the second half stop the process. 3 time steps and 25 messages to complete phase 1.

Phase Two and phase three same as in example one:

Total number of time steps is equal:

3+2+3 = 8 time steps

Total number messages are equal:

25 +3 +7 = 35 messages

4.2.3 Abstract Algorithm

This section presents the pseudo code for the election algorithm. A number of assumption and variables have to be assigned, these are as follows:

Each node has the following variables:

a. **Local ID**: the node ID that participate in the election process.
b. **Local Pos**: The node Position.
c. **Curr_Step**: Last step in the election process.
d. **Ph1_finish_flag**: if true it indicates that the Phase 1 was finished.

The algorithm uses two types of messages

- **Election**: contains Phase (1 OR 2), step, ID(the winner ID), Pos (The winner position).
- **Leader**: contains the new Leader (ID and Position).

The nodes are in one of two states:

- **Normal**: when the node is unaware of any failure and the network is normal.
- **Candidate**: when the node is aware of the failure and the node is participating in the election process.
- **Leader**: one node must have this state in stable network.

1. Case state = normal

```
  Upon detecting failure
      {
         State = Candidate
           Phase = 1
Step = 1
                ID = Local_ID
               Pos = Local_Pos
           Curr_Step = Step
        Send Election(Phase, Step ,ID, Pos)  on Link 1.
      }

  Upon receiving election message on link r
  if (Phase == 2)
    Store the message and wait until the state becomes candidate and Phase 1
   finish.
  else
           {
                  State = Candidate
                    if (ID < Local ID )
                      {
                        ID = Local ID
                        Pos = Local Pos
                      }

                    if (r < Log(N))
                      {
                       Step = Step+1
                           r = r+1
                          Curr_Step = Step
                       Send Election(Phase, Step ,ID, Pos)  on Link  r .
                      }
                    if (r = Log(N))
                     {
                       Ph1_finish_flag = true
                       if (node label = (XX…1X))
                        {
                          Phase = 2
                            Step = 1
                    r = Log(N) – Step
                  Curr_Step = Step
                Send Election(Phase, Step ,ID, Pos)  on Link  r .
                 }
            }
       }
```

- ***Case state = Candidate***

```
Upon Receiving Election message
 If (Phase = =1 )
 {
      If  (Curr_Step > Step)
           Ignore the message
      If  (Curr_Step == Step) and (the r bit in the node label = =1)
           Ignore the message

      If   (Step >  Curr_Step) OR ((Curr_Step == Step)  and (the r bit in
                  the node label ==0)) and (r < Log(N))
        {
          Step = Step+1
           r = r+1
          Curr_Step = Step
         Send Election(Phase, Step ,ID, Pos)  on Link  r .
        }

      if (Step >  Curr_Step) OR ((Curr_Step == Step)  and (r ==
           Log(N))) and  (node_ label == (XX…1X))
         {
          Phase = 2
            Step = 1
            r = Log(N) – Step
           Curr_Step = Step
          Send Election (Phase, Step, ID, Pos) on Link r.
         }

 }
 if (Phase == 2)  and (Ph1_finish_flag = True )
 {
     if ( Step< LogN –1 )
          {
            Step = Step +1
           r = Log(N) – Step
           Send Election(Phase, Step ,ID, Pos)  on Link  r
          }
     if ( Step == LogN –1 )
        BROADCAST LEADER(ID,Pos)
 }
```

Appendix three shows a flowchart for the first algorithm

4.3 Proposed Leader Election Algorithm in Hypercube with the Presence of One Link Failure:

This section describes the Leader Election Algorithm in Hypercube with the Presence of One Link Failure in plain text and in Pseudo code.

4.3.1 Algorithm Description

This algorithm consists of three phases. Each phase has several steps and messages. Phase one is initiated when one or more nodes detect a leader failure, it initiate the election process. This phase reduces the count of participated nodes in the election process to N/2 nodes aware of the election. The second phase uses the reduction all-to-one communication operation "with additional steps to deal with link failure" to have the result in one node that have the address ($X_1 0_2 0_3 \ldots 0_d$) (X means 1 or 0). Finally in the third phase ($X_1 0_2 0_3 \ldots 0_d$) node broadcasts leader message to all nodes in the hypercube using broadcast one-to-all communication operation, "with additional steps to deal with link failure". During each step in phase one and two, the received ID is compared with the local ID. The greater ID is passed to the next step. The steps for phases are as follows:

Phase One: This phase starts when one or more nodes detect the leader failure. Each node detect the failure change its state to candidate and initiate the election algorithm:

Step1: Send an election message to the node that differs in the right most bit through link 1. Election message composed of (phase, step, winner ID, winner position). The election message change the state of receiving node from normal state to candidate state.

Step2: The sender and receiver in the previous step (nodes in the candidate state) send an election messages to the two associated nodes that differ in the second right most bit through link 2.

Step 3 to LogN : The senders from the previous step send an election messages to the nodes that differ in the r right bit through the link Step. The receiver nodes send an authentication message (election message) through link Step-2 to insure that the election message reached the node, due to link failure. After sending the authentication message the node wait for acknowledgement or time out to send election through link r.

The nodes that receive the authentication message may have also received the election message in advance, so the authentication message is ignored (not acknowledged) after receiving the greater ID. In the second case ,when the authentication message is received by a node in normal state, To deal with the probability of link failure the node send election message through link (Step-1). In both cases, the nodes that receive the authentication message send an acknowledgement to the sender.

The nodes that receive the authentication message send an election messages through Step + 1 link to continue the algorithm. The next steps are the same until step Log(N) is reached.

Step Log(N) : N/2 nodes (the senders and receivers from the Log(N) –1 step) send an election messages to the nodes that differ in the Log(N) bit order from right through Log(N) link. End phase one.

After this phase leader ID is within N/2 nodes. These nodes will continue to phase 2.

During the execution of phase one, if the receiver is in progress in the algorithm, it will complete the greater step and terminate the smaller one. Each node receives the election message; it compares its own ID with the received ID then completes the next step with the greater ID. Phase one ends after Log(N) steps, reducing the participant nodes to N/2. The leader ID, and its position for the whole hypercube will be within (Log(N) –1) dimensional hypercube. If the two sub hypercube reach step Log(N) simultaneously 'in the worst case' the sub cube with the most left bit = 0 complete the algorithm. And the other sub terminate.

Phase Two: The second phase uses the reduction all-to-one communication operation. Our algorithm adds some alterations deal with the link failure.

The operation applied to (N/2) nodes hypercube that resulted from phase one, will have the result in the node that have address $X_1 0_2 \ldots 0_d$. this is achieved as follows:

Step1 To Step (Log (N) –3) : nodes that reached phase two with bit order from left (step + 1) and (step + 2) have the value =1 ($X_1 1 1 X_4 \ldots X_d$) exchange the greater ID with the nodes with the same properties except that (step + 2) bit order from the left = 0 through the link (Log(N) – step – 1). The participants in the exchange need an acknowledgement or it will wait until time out to progress. After the exchange, these nodes send election message to nodes with the second left bit = 0, ($X_1 0 X_3 \ldots X_d$) through link (Log(N) – step). These steps include the result inside a two dimensional hypercube even with the probability of one link failure.

Step (Log(N) -2): the receivers in the previous step and with the two right bits in the nodes labels (10, 01) send an election messages to node with the two right bits(11).

Step (Log(N)-1): After comparison and receiving the greater ID node with the two right bits (11) sends the winner ID and position to nodes with the two right bits (10, 01).

Step Log(N): In last step nodes with the two right bits (10, 01) send an election message to the node ($X_1 0_2 0_3 \ldots 0_d$).

After the end of phase 2 the last node ($X_1 0_2 0_3 \ldots 0_d$) is the only node that knows the leader ID.

Phase3: In this phase, the node ($X_1 0_2 0_3 \ldots 0_d$) broadcasts the result to all nodes considering the link failure. Leader message contains new leader ID and position. The broadcast will be as follows:

Step 1 : node ($X_1 0_2 0_3 \ldots 0_d$) send leader message through link (step) to node ($X_1 0_2 0_3 \ldots 1_d$).

Step 2 : after increment the step value, the two nodes in phase 1 send the message through link (step).

Steps 3 to Log(N): Each node that receives the leader message sends it through the link step. Due to the probability of the presence of link failure in this phase each node receives the leader message sends another message through link (step – 2) to check if the message reaches the other side. Node that receives the extra message and is already aware of the broadcast ignores the extra message. If the node does not know about the message it sends the extra message through link (step -1) .The last message isn't needed after step (Log (N)).

4.3.2 Abstract Algorithm

This section presents Pseudo code for the election algorithm. Each node has the following properties:

a. Local ID: the node ID that participate in the election process.

b. Local Pos: The node Position.

c. Curr_Step: Last step in the election process.

The algorithm uses five types of messages.

- **Election**: contains Phase (1 OR 2), step, ID (the winner ID) ,Pos (The winner position).
- **Leader**: contains the new Leader (ID and Position).
- **Authentication:** contains Phase (1 OR 2), step, ID (the winner ID) , Pos (The winner position).
- **Acknowledgement**: contains (OK) and the max ID it knows.
- **Check:** node receives the authentication message sends a check message to avoid the effect of link failure.

The nodes are in one of three states:

- **Normal**: when the node is not aware of any failure and the network is normal.
- **Candidate**: when the node is aware of the failure, and participating in the election process.
- **Leader**: one node must has this state in stable network.

```
1.  Case state =  normal
   Upon detecting failure
         {
           State = Candidate
               Phase = 1          Step = 1
                  ID = Local_ID
                 Pos = Local_Pos
             Curr_Step = Step
          Send Election(Phase, Step ,ID, Pos)  on Link 1.
         }
Upon receivingd election message on link r
   if (Phase == 2)
     Store the message and wait until the state becomes candidate
    and Phase 1 finish.
   else
              {
                State = Candidate
                        if (ID < Local ID )
                            {
                              ID = Local ID
                              Pos = Local Pos
                            }
                   Send authentication message on link step-1
                                                    //authentication message
                                                    contain  same information
                                                    such as election
                                                    message.
              }
   Upon receiving authentication message
                    {
                       If step < Log(N)
                       {
                     Compare ID and send acknowledgment on link step
             -1
                      Send check message on link step // to overcome
             link failure
                       }
                    If step =  Log(N)
                      {
                     Compare ID and choose the winner
                    Phase = 2
                   Step = 1
```

```
        State = candidate
    Send exchange message to the node that differs in the third
       most left in label
            And wait until receiving the opposite message or
       timeout.

        }

        Upon receiving check message
         Compare ID and send election message on link step
       + 1
           if (r < Log(N))
               Step = Step+1
                 r = r+1
                Curr_Step = Step
               Send Election(Phase, Step ,ID, Pos)  on Link  r
       .
             }
           if (r = Log(N))
            {
              Ph1_finish_flag = true
               if (node label = (XX…1X))
                 {
                  Phase = 2
                   Step = 1
                     r = Log(N) – Step
                   Curr_Step = Step
                 Send Election(Phase, Step ,ID, Pos)   on Link
       r .
Up on receiving leader message or check message
  Ignore the message   /// because these messages just effect
   candidate state
```

- ***case state = Candidate***

```
Upon Receiving Election message on link r
        If (Phase = 1 )
         {
          If (Step < Log(N))
If  (Curr_Step > Step) or((Curr_Step == Step) and (the r bit in the node label = =1)
                        Ignore the message

               Else   if step >=2
                     {
                      compare id and send authentication message
  on link step -1
                     wait for acknowledgement
                    }
        }
  If step = Log(N)
       {
      phase = 2
     step = 1
     Send exchange message on link Log(N) – step – 1
 }
   if ( Phase = 2 )
     {
       if  ( step <  Log(N) – 3)
         {
          compare the ID's
          ++step
          send exchange message on link Log(N) – Step – 1
       if (step = Log(N) -3 )
        {
        Compare  the ID's
        ++ step
               if  the second and third left bit = 10
                   send election message on link 1
              if  the second and third left bit = 01
                   send election message on link 2
         }
       if (step = Log(N) -2 )
        {
        Compare  the ID's
        ++ step
                   send election message on link 1 and 2
          }
       if (step = Log(N) -1 )
```

```
                    ++step
                      if  the second and third left bit = 10
                       send election message on link 2
                       if  the second and third left bit = 01
                      send election message on link 1
                   }
         if (step = Log(N) )
            {
              compare ID's
               Send LEADER MESSAGE  on link 1
             }
    }    //  end election  message

Up on receiving authentication message on link r
                    {
                      Compare ID and send acknowledgment message on
    link r.
                       ++Step
                     Send election message on link r+ 1
                   }
    Upon receiving Acknowledgment message or time out
                   {
                  Compare ID
                  ++ Step
                   Send election on link step
            }
    Up on receiving Leader message on link r
    {
     Change Leader ID to the new leader.
     Change node state to normal.
       If ( r< Log(N) )
            send leader message on link r+1
             if r > = 2  send Check message on link  r-1
      }
    Up on receiving Check message on link r
       Send leader message on link r+1

END OF THE ALGORITHM
```

Appendix three shows a flowchart for the second algorithm

4.4 Conclusion.

Chapter four described the proposed algorithms in two ways: Plain text and pseudo code. Each algorithm is composed of three phases. Each phase is composed of several steps. Synchronizations between phases and steps were controlled by using Boolean flags. Flags can hold the algorithm in any node to wait until the phase or step is reached.

The proposed algorithms deal with the most common problem in centralized distributed systems. The new leader election algorithm in hypercube networks proposed is a new algorithm that differs from all previous algorithms in its design and efficiency. It focuses on minimizing the time steps and the number of messages. The leader election algorithm in hypercube with the presence of one link failure is the only solution that deals with the presence of a link failure in hypercube networks.

Next chapter will present analyses and performance evaluation for the proposed algorithms.

CHAPTER FIVE
PERFORMANCE EVALUATION AND SIMULATION

5.1 Introduction

The algorithm is analyzed by computing the number of messages and time steps. The analyses process is carried out for the two cases. The first 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. Analyses are presented in this chapter to compute the messages and time steps for each algorithm. In Section 5.2.1 the new leader election algorithm in hypercube is analyzed. In Section 5.2.2 the leader election algorithm in hypercube in the presence of one link failure is analyzed. Section 5.3 presents the simulation and proof for the first algorithm by simulation.

5.2 Analyses.

The Proposed algorithms are evaluated and their performance is computed mathematically in this section.

5.2.1 New Leader Election Algorithm in Hypercube Networks

This algorithm performance is evaluated in simple case when the leader failure is detected by one node and the worst case when the leader failure is detected by N-1 nodes.

5.2.1.1 Simple Case.

Theorem 1: In the simple case, the number of messages needed to complete the algorithm is at most O(N) messages.

Proof:

To find the number of messages, a complete computation is carried out for each phase, and then the total number of computation for the overall algorithm is calculated.

Phase One:

During this phase, each node receives one message except the initiator. So the number of messages is equal to (N-1). Another way to compute the messages during phase one is as follows:

Step 1: Needs one message from the initiator to the node that differ in the right bit

Step 2: Needs two messages from the participated nodes in step1 to nodes differ in bit order 2 from right, and so on until the step Log (N) is reached. This is shown in Equation (1).

$$2^0 + 2^1 + 2^2 + ... + 2^{Log(N-1)} = \sum_{i=0}^{Log(N)-1} 2^i = N - 1 \qquad (1)$$

Phase Two: Each node sends one message during the second phase (reduction phase) except the last node, so the number of messages is equal to (N/2 – 1).

In step1, (N/4) election messages are sent, in step2 (N/8) is sent until the last step which needs N/N message. This is as in Equation (2).

$$\frac{N}{4}+\frac{N}{8}+\frac{N}{16}+......+\frac{N}{N}=\sum_{i=0}^{Log(N)-2} 2^i=(\frac{N}{2}-1) \qquad (2)$$

Phase Three: Broadcast needs N-1 messages, since each node receives one leader message except the initiator. This is shown in Equation (3).

$$1+2+4+8+...+\frac{N}{2}=\sum_{i=0}^{LogN-1} 2^i=N-1 \qquad (3)$$

The total number of messages is given by Equation (4).

Total:

$$(N-1)+(\frac{N}{2}-1)+(N-1)=\frac{5N}{2}-3=O(N) \text{ Messages} \qquad (4)$$

<u>Theorem 2</u>: The election algorithm in the hypercube becomes stable with a new leader by O (Log (N)) time steps.

Proof:

The same procedure as above is followed to find the total time steps.

Phase One: Reducing the nodes containing the leader ID to one half of the nodes, this will need Log (N) time steps.

Step 1: node detects the failure sends the election message to node that differ in the first bit from the right.

Step 2: nodes knowing about the election send the election messages to nodes that differ in the second bit from the right.

Steps 3 to Log (N): nodes knowing about the election send the election messages to nodes that differ in the (Step number) bit order from the right.

Phase Two: this phase resumes the election process with (d-1) dimensional hypercube with all its nodes aware of the election process, and the nodes in this phase are in position of the election result from the first phase. The reduction algorithm is used to guide the result of the election in this phase to one node in position of all the information regarding the new leader ID, this will require (Log (N) –1) Steps. These steps are described in algorithm description in Chapter Three**.**

Phase Three: Broadcasting (One-To-All), the leader message in hypercube needs Log (N) time steps as follows:

Step 1: node that has the new leader information sends this information to the node differs in label address in the most left bit.

Step 2: the sender and receiver in the previous step send the leader information to the nodes differ in the bit order 2 from left.

Steps 3 to Log(N): receivers and senders in the previous steps send the leader message to nodes differ in the bit order (Step number).

The total time steps for all phases is in Equation (5).

Total :

$$Log(N) + Log(N) - 1 + Log(N) = 3Log(N) - 1 = O(Log(N))\ \ Timesteps \qquad (5)$$

5.2.1.2 Worst Case:

Theorem 3: The number of messages needed to complete the algorithm, in the worst case, is at most O(N Log(N)) messages.
Proof:
To find the number of messages, it is necessary to compute the number of messages for each phase then calculates the total number of messages for the algorithm.

Phase One: Each node sends one message during each step in the first phase. The number of nodes is equal to (N-1) and the number of steps is equal to Log (N). This phase needs:

(N-1) * (Log(N)) = N Log(N) – Log(N) messages (6)

Phase Two: each node sends one message during the second phase (reduction phase)

except the last node, so the number of messages is equal to (N/2 – 1).

In step1 (N/4) election messages are sent. In step2, (N/8) is sent until the last step which needs N/N message. This is as in Equation (7).

$$\frac{N}{4}+\frac{N}{8}+\frac{N}{16}+......+\frac{N}{N}=\sum_{i=0}^{Log(N)-2}2^i=(\frac{N}{2}-1) \qquad (7)$$

Phase Three: Broadcast needs N-1 messages, since each node receives one leader message except the initiator. This is shown in Equation (8).

$$1+2+4+8+...+\frac{N}{2}=\sum_{i=0}^{Log(N)-1}2^i=N-1 \qquad (8)$$

Phase Two and Phase Three are same as in simple case**.** The total for the election algorithm in worst case is in Equation (9).

Total :

$$(N-1)(Log(N))+(\frac{N}{2}-1)+(N-1)=O(NLog(N))\ \ Messages \qquad (9)$$

Theorem 4: In any case the election algorithm in the hypercube becomes stable with a new leader by O (Log(N)) steps.

Proof:

Phase One: Reducing the nodes containing the leader ID to one half of the nodes. This will need Log (N) time steps.

Step 1: nodes detect the failure sends the election message to node that differ in the first bit from the right.

Step 2: nodes aware of the election send the election messages to nodes that differ in the second bit from the right.

Steps 3 to Log (N): nodes aware of the election send the election messages to nodes that differ in the (Step number) bit order from the right.

Phase Two: this phase resumes the election process with d-1 dimensional hypercube with all its nodes aware of the election process, and the nodes in this phase are in position of the election result from the first phase. The reduction algorithm is used to guide the result of the election in this phase to one node in position of all the information regarding the new leader ID, this will require (Log (N) –1) Steps these steps are described in algorithm description in chapter three**.**

Phase Three: Broadcasting (One-To-All), the leader message in hypercube needs Log (N) time steps as follows:

Step 1: node that has the new leader information sends this information to the node differs in label address in the most left bit.

Step 2: the sender and receiver in the previous step send the leader information to the nodes differ in the bit order 2 from left.

Steps 3 to Log(N): receivers and senders in the previous steps send the leader message to nodes differ in the bit order (Step number).

The total time steps for all phases is in Equation (10).

Total :

$$Log(N) + Log(N) - 1 + Log(N) = 3Log(N) - 1 = O(Log(N))\ \ Timesteps \qquad (10)$$

From the previous proof, we note that the time steps are the same for any case in proposed algorithm.

Analysis for Message Size:

Algorithm use two types of messages: Election Message and Leader Message.

* Election message composed of:

Message type needs	1 bit
Phase	2 bits
Step	Log(Log(N)) bits
Winner ID	Log N bits

Winner Position	Log N bits

The election message length is:

Total : 2Log N + Log(Log (N)) + 3 = O Log N Bits

* Leader Message Composed of:

Message type needs	1 bit
Phase	2 bits
Step	Log(Log(N)) bits
Leader ID	Log N bits
Leader Position	Log N bits

Total : 2Log N + Log(Log (N)) + 3 = O Log N Bits

5.2.1.3 Contention Free.

We can prove that the algorithm is a contention free if the nodes do not send more than one message through the same link at the same time.

Theorem 5: *The algorithm does not allow any two messages through the same link at the same time*.

Proof:

In any time step, each link is used by one message only as follows:

Phase one: in this phase no link is used more than one time. Node use one link to send election message to its neighbor and do not send this message again through the same link.

Phase two: in all to one reduction there is no contention because participant link used to pass one message only.

Phase three: broadcasting is the same as reduction all to one each step uses different links and no more than one message in the same link in the same step.

Therefore no contention will take place within the algorithm phases.

5.2.1.4 Comparison with Previous Algorithms.

Four election algorithms in hypercube were reviewed. The first was by (Gerard, 1993), the second was by (Flocchini and Mans, 1996), the third algorithm was by (Dobrev and Ruzecka,1997) and the fourth was by **(Castorino and Ciccarella**,1999) . The proposed algorithm in this dissertation is better in time steps using O(Log(N)) steps over the whole algorithm**.** Gerard algorithm uses **O(Log^{2}N)** and **Flocchini and Mans** algorithm uses **Θ(Log^{3}N). Dobrev and Ruzecka** algorithm was proposed in 1997 . This algorithm focused on number of messages and ignored the time steps to get (N (LogLog(N))) message complexity. **Castorino and Ciccarella** in 1999 use Log(N) Steps, when only one node detect the failure and they didn't take the worst case of the algorithm. The message size is better in Gerard algorithm where all the others are the same. Table-1 views the differences between proposed algorithm and the previous works:

Table-1 Comparison between proposed and previous algorithms

Algorithm	Number of Messages	Time Steps	Message Size /bits	Year
Proposed algorithm	**O(N)**	**O(Log(N))**	**2Log N + Log(Log (N)) + 3**	**2006**
Flocchini and Mans	**Ω(N)**	**$\Theta(\mathrm{Log}^3 N)$**	**3LogN +Log(Log(N))**	**1996**
Gerard	**O(N)**	**$O(\mathrm{Log}^2 N)$**	**O(Log(Log(N))**	**1993**
Dobrev and Ruzecka	**O(NLogLog(N))**	**Not computed**	**Not Computed**	**1997**
Castorino and Ciccarella	**O(N)**	**O(Log(N))**	**3LogN**	**1999**

5.2.2. Leader Election Algorithm in Hypercube Network with the Presence of One Link Failure

We analyze the algorithm by computing the number of messages and time steps. This analyses takes two ways. The first one is simple case when the failure is detected by one node. The second way is, when the leader failure is detected by a subset of nodes reaching in the worst case to (N-1) at the same time.

5.2.2.1 Simple Case:

Simple case is when the algorithm is started by one node that aware of leader failure.

Theorem 6: The number of messages needed to complete the algorithm is at most O(N) messages.

Proof:

To find the number of messages, we compute it for each phase, then calculate the total over all algorithm.

Phase One:

During this phase the algorithm uses three types of messages: election messages where each node receives one message except the first node. The number of election messages is equal (N-1). This is as shown in Equation (11).

$$1+2+4+\ldots+\frac{N}{2} = \sum_{i=0}^{LogN-1} 2^{i} = N-1 \qquad (11)$$

The second type is the authentication message which starts from step 2, where each node sends one authentication message until the end of this phase. The total number of messages from this type is (N-2) messages. This is as shown in Equation (12).

$$2+4+\ldots+\frac{N}{2} = \sum_{i=1}^{LogN-1} 2^{i} = N-2 \qquad (12)$$

The third type is the acknowledgement message. Each node when it receives the authentication message sends an acknowledge message except in the last step. The number of acknowledgement messages is (N/2 –2). This is as shown in Equation (13).

$$2+4+\ldots+\frac{N}{4} = \sum_{i=0}^{LogN-2} 2^{i} = \frac{N}{2}-2 \qquad (13)$$

If there is a link failure we need one message to inform the previous node about the election. So the total messages for phase one is in Equation (14).

$$(N-1)+(N-2)+(\frac{N}{2}-2)+1 = 5\frac{N}{2}-4 \qquad (14)$$

Phase Two: As explained in the description of phase 2 the nodes start the reduction node to conclude the result in one node. Half of the participant nodes in phase 2 send two messages for each. One message to avoid the probability of link failure and the second to go on the reduction process. This process is valid until the result becomes inside 4 nodes. Then the algorithm needs 6 messages to obtain the leader ID in one node. This shown in Equation (15).

$$\frac{N}{4}+\frac{N}{4}+\frac{N}{8}+\frac{N}{8}+\cdots+4+4+6=2\sum_{i=2}^{Log(N)-2} 2^i +6$$

$$=2(\frac{N}{2}-4)+6= N-2 \text{ Messages} \qquad (15)$$

Phase Three: Broadcast needs N-1 messages, since each node receives one leader message except the initiator. This shown in Equation (16).

$$1+2+4+8+\cdots+\frac{N}{2}=\sum_{i=0}^{Log(N)-1}2^i = N-1 \ \ Messages \qquad (16)$$

To cover the probability of link failure we need (N-2) Messages as in Equation (17).

$$2+4+8+\ldots+\frac{N}{2}= \sum_{i=1}^{Log(n)-1}2^i = N-2 \ \ Messages \qquad (17)$$

If there is a link failure we need another message so the number of messages for phase three is as in Equation (18).

$$(N-1)+(N-2)+1=2N-1 \ \ \text{Messages} \qquad (18)$$

Total : From equations 14,15,18 the total number of messages over all the algorithm is :

$$(5\frac{N}{2}-4)+(N-2)+2(N-1)=(11\frac{N}{2})-10= \quad O(N) \ \ \text{Messages} \qquad (19)$$

Theorem 7: The election algorithm in the hypercube becomes stable with a new leader by O(Log(N)) steps .

Proof:

To find the total time steps we compute the time steps for each phase and then add the results to find it overall the algorithm.

Phase One: Reducing the nodes containing the leader ID to the half nodes of the model needs Log(N) time steps.

Step 1: node detect the failure sends the election message to the node differ in the first right bit needs one time step, no need for the authentication and the acknowledgement.

Step 2 to Log(N) : each step needs 3 time steps, first for election message, second for the authentication message and third for the acknowledgement.

The last step doesn't need the acknowledgment. The total as shown in Equation (20).

$$1 + 3(\mathrm{Log(N)}1) - 1 = 3\,\mathrm{Log(N)}3 \quad \mathrm{Steps} \qquad (20)$$

Phase Two: this phase continues the election process with (d-1) dimensional hypercube with all its nodes aware of the election and have the result from the first phase. The first stage of this phase needs three time steps: two steps for the exchange and one for reduction step. Phase two needs (3(Log(N) -3))steps , and three steps for the two dimensional hypercube. Time steps in phase two as shown in Equation (21). $3(\mathrm{Log(N)} - 3) + 3 = 3\mathrm{Log(N)} - 6\,\mathrm{Steps}$

(21)

Phase Three: Broadcasting (One-To-All), the leader message in hypercube needs Log(N) time steps. In the case of link failure the algorithm needs another two time steps to inform the unreachable node as described above. Phase three needs (Log(N) + 2) steps.

The time steps over all the algorithm is the result of Equation (22).

$$3\,\mathrm{Log(N)}3\; + 3\mathrm{Log(N)} - 6\; +\; \mathrm{Log(N)} + 2\; = 7\mathrm{LogN} - 7\; = \mathrm{O}\,(\mathrm{Log(N)})\ \mathrm{Steps} \qquad (22)$$

5.2.2.2.Worst Case:

Worst case is when the algorithm is started by (N-1) nodes that detect leader failure.

Theorem 8: The number of messages needed to complete the algorithm, in the worst case, is at most O(N Log(N)) messages.

Proof:

To find the number of messages, we compute them for each phase, and then calculate the total over the entire algorithm.

Phase One: When the failure is detected by all nodes, Each node sends one message during each election step in the first phase. The algorithm needs (Log(N)) election step so the number of messages during the election steps is (N Log(N)) messages. The number of authentication and acknowledgement messages adaptive depends on the number of nodes that detect the error. But in any case it does not exceed (N) messages. The total number of messages does not exceed (3N Log (N)) messages.

Phase Two: As explained in the description of phase 2 the nodes start the reduction process to conclude the result in one node. Half of the participant nodes in phase 2 send two messages for each. One message to avoid the probability of link failure and the second to go on the reduction process. This process is valid until the result becomes inside 4 nodes. Then the algorithm needs 6 messages to obtain the leader ID in one node. This shown in Equation (23).

$$\frac{N}{4}+\frac{N}{4}+\frac{N}{8}+\frac{N}{8}+\cdots+4+4+6=2\sum_{i=2}^{Log(N)-2} 2^i +6$$
$$=2(\frac{N}{2}-5)+6= N-4 \text{ Messages} \qquad (23)$$

Phase Three: Broadcast needs N-1 messages, since each node receives one leader message except the initiator. This is shown in Equation (24).

$$1+2+4+8+\cdots+\frac{N}{2}=\sum_{i=0}^{Log(N)-1} 2^i = N-1 \;\; Messages$$ (24 To cover the probability of link failure, we need (N-2) Messages as in Equation(25).

$$2+4+8+\ldots+\frac{N}{2}=\sum_{i=1}^{LOg(n)-1} 2^i = N-2 \;\; Messages \qquad (25)$$

If there is a link failure, we need another message; so the number of messages for phase Three is as in Equation (26).

$$(N-1)+(N-2)+1=2N-1 \;\; \text{Messages} \qquad (26)$$

Total : From the time steps in all phases the total number of messages over all the algorithm is in Equation (27).

$$3N(Log(N))+(N-4)+(2N-1) = O(N\,Log(N)) \;\; \text{messages} \qquad (27)$$

Theorem 9 *In any case the election algorithm in the hypercube become stable with a new leader by O(Log(N)) steps.*

Proof:

To find the total time steps, we compute the time steps for each phase and then add the results to find it overall the algorithm.

Phase One: Reducing the nodes containing the leader ID to the half nodes of the model needs 3Log (N) time steps.

Step 1: node detect the failure sends the election message to the node differ in the first right bit needs one time step, no need for the authentication and the acknowledgement.

Step 2 to Log(N) : each step needs 3 time steps, first for election message, second for the authentication message and third for the acknowledgement.

The last step doesn't need the acknowledgment. The total as shown in Equation (28).

$$1 + 3(\text{Log}(N)) - 1 = 3\,\text{Log}(N) \quad \text{Steps} \qquad (28)$$

Phase Two: this phase continues the election process with (d-1) dimensional hypercube with all its nodes aware of the election, and has the result from the first phase. The first stage of this phase needs three time steps: two steps for the exchange and one for reduction step. Stage two needs (3(Log(N) -3))steps , and three steps for the two dimensional hypercube. Time steps in phase two as shown in Equation (29).

$$3(\text{Log}(N) - 3) + 3 = 3\text{Log}(N) - 6\,\text{Steps} \qquad (29)$$

Phase Three: Broadcasting (One-To-All), the leader message in hypercube needs Log(N) time steps. In the case of link failure the algorithm needs another two time steps to inform the unreachable node as described above. Phase three needs (Log(N) + 2) steps.

The time steps over all the algorithm is the result of Equation (30).

$$3\,\mathrm{Log(N)} + 3\mathrm{Log(N)} - 6 + \mathrm{Log(N)} + 2 = 7\mathrm{LogN} - 4 = O\,(\mathrm{Log(N)})\ \mathrm{Steps} \qquad (30)$$

It is obvious that the time step in the worst case is the same as in simple case.

Analysis for Message Size:

Algorithm use two types of messages : Election Message and Leader Message.

* Election message composed of:

Message type needs	1 bit
Phase	2 bits
Step	Log(Log(N)) bits
Winner ID	Log N bits
Winner Position	Log N bits

The election message length is:

Total : 2Log N + Log(Log (N)) + 3 = O Log N Bits

* Leader Message Composed of:

Message type needs	1 bit
Phase	2 bits
Step	Log(Log(N)) bits

Leader ID	Log N bits
Leader Position	Log N bits

Total : 2Log N + Log(Log (N)) + 3 = O Log N Bits

5.3. Simulation

This section explains the simulation for the first algorithm. Before explaining the simulation the visual basic programming language and its Object Oriented Programming (OOP) properties will be described**.**

5.3.1 Programming Language _ Visual Basic 6 (VB6).

VB6 is a language that provides powerful features such as Object Oriented Programming, Event Handling, Structured Programming, Graphical User Interface and much more. The VB6 is an interpreted language using event programming. It has many objects and tools that make the programming less difficult. Models and Classes in VB6 can be added to make the program short and powerful.

5.3.2 Leader Election in Hypercube Simulation

The node is the main object in our simulation. Each node in the hypercube is represented as instant of designed class. The class contains buffer, can send and receive the messages. The routing from source to destination depends on the number schema that uses the labels of nodes. This numbering schema has the useful property of the minimum distance between two nodes is given by the number of bits that are different in two labels. For example nodes 0101 and 1110 are three links apart, since they differ at three bits positions. In Section 5.3.2.1, the input variables and the results are described. Some cases will be taken as examples. Analyses for the results of these examples will be introduced. The section will be started with introduction into how to use the simulation.

5.3.2.1 Introduction to Simulator.

When the simulation starts, an input box appears. The user needs to enter the number of process or nodes in the hypercube to be simulated as shown in Figure-15. It is a must to enter number N such that $2^x = N$ when x is an integer number. This number can be changed any time from the menu using set configuration.

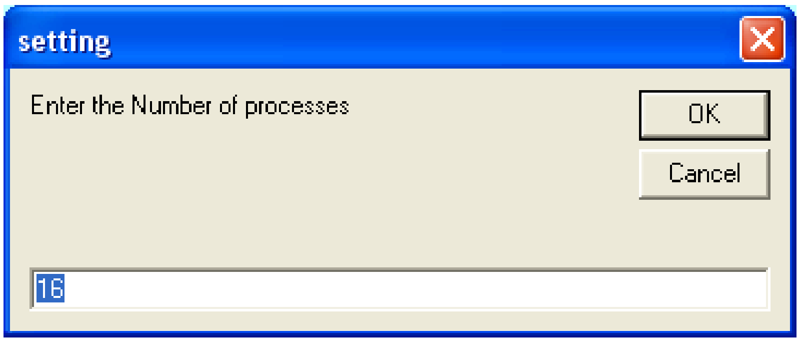


Figure-19The first screen in the simulation.

The main screen is shown in figure-16. it shows the simulation environment which can be described as follows:

1. Menus Bar: which contains the following menus:

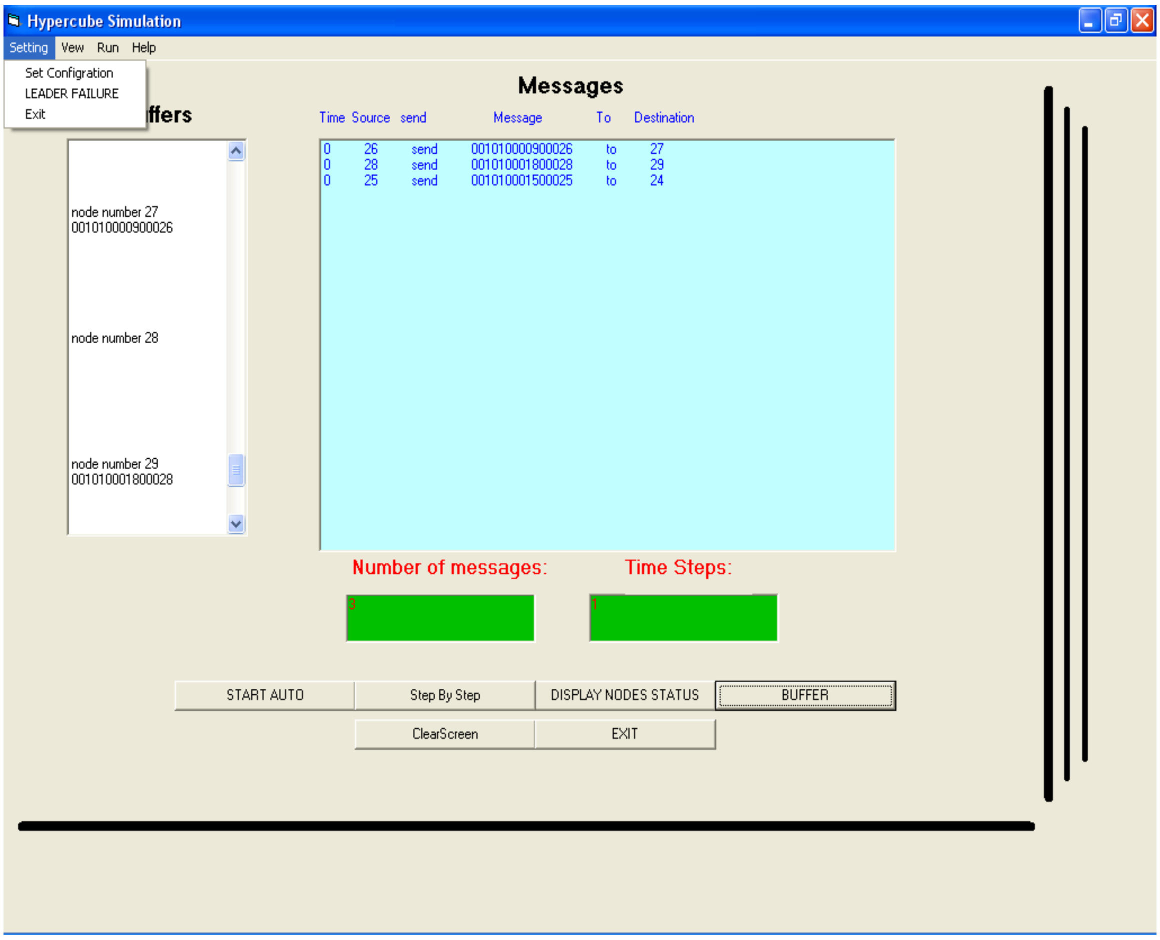
- Setting menu: use to establish the configuration for the simulation. Three choices in this menu:

  a- Set configuration: reset or establish the nodes configuration. This is done automatically in the first run. If the user needs another run, this option must be used.

b- Leader failure: This option is used to select the number of nodes to detect the leader failure to start the election process. This number must be less than the number of nodes in the hypercube.

c- Exit: used to end the simulation.

- View Menu: used to clear screen and to change the back color of the main screen.
- Run Menu: used to start and stop the simulation.
- Help Menu: used to give information about the programmer and guidance to use the simulation.



```
node number 27
001010000900026

node number 28

node number 29
001010001800028
```

```
Time  Source  send     Message            To    Destination
0     26      send     001010000900026    to    27
0     28      send     001010001800028    to    29
0     25      send     001010001500025    to    24
```

Number of messages: 3

Time Steps: 1

Figure-20 Simulation main screen

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

This is the blue colored box and it displays any new message during the simulation run.

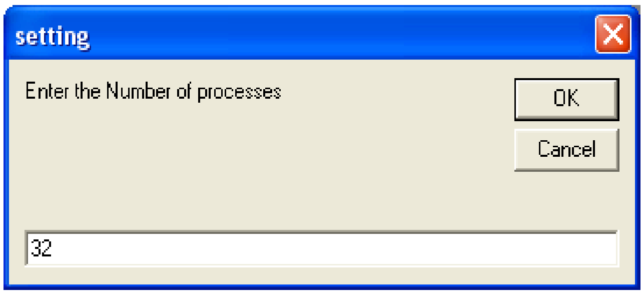
3. Buffers list: shows the contents of all buffers in hypercube nodes. This list is invisible by default, user can show it when press on buffer command.
4. Number of messages: shows the number of messages during the simulation and total number of messages as the execution is finished.
5. Time Steps: counter compute the current time step the algorithm reaches and the total time steps when the execution is finished.
6. Commands Buttons. The following commands are found on the main screen:
   - a- Start Auto: this Command completes the election algorithm by executing all steps and gives the final results.
   - b- Step by Step: this command runs the simulation step by step. Each step show what happened in one time step.
   - c- Buffers: when pressed on this command the contents of the buffers in hypercube are shown in the buffers list.
   - d- Clear Screen: used to clear the main screen .
   - e- Exit: to end the simulation and go out from the program.

5.3.2.2. Examples.

In this sub section three examples are introduced. Example one will show the execution of the election algorithm in five dimensional hypercube. To explain the steps and messages, log file is displayed in Table 2. Example two explains the contents of the log file and nodes buffers during the execution of the algorithm. Example three executes the algorithm using a large hypercube with 2048 nodes.

Example one:

When the program starts, user assigns the number of nodes in hypercube. In this example, assume this number equal (32) as shown in Figure-17

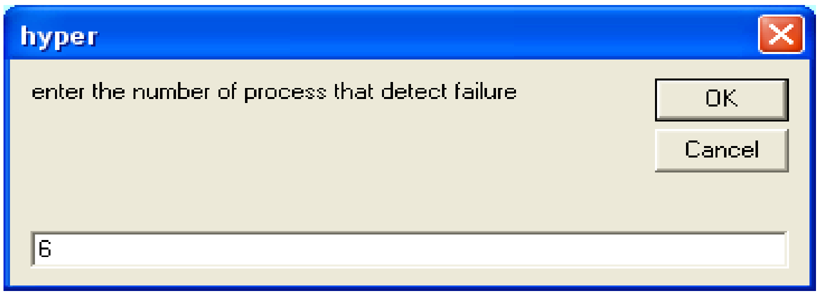


Enter the Number of processes

32

Figure-21 Input the number of Processes Dialog box

When OK is pressed, the main screen is opened. Before the program is executed, the number of nodes that detect leader failure must be entered by user. This is achieved by using setting menu, then leader failure submenu. The number must be less than the number of hypercube nodes. In this example, assume that this number equals to (6). The addresses of these nodes are distributed randomly among the hypercube nodes.


hyper

enter the number of process that detect failure

6

Figure-22 Input the number of processes detects the failure Dialog box.

The algorithm is ready to start now. User can select step by step command and the messages that appear on the screen will be as in table-2. In the table, rows represent messages. Each row represents one message the columns contain time steps, source nodes, messages and destinations.

Message composed of 15 digits as follows:

- Digit in position 1 represent message type: 0 for election messages, 1 for broadcasting messages.
- Digits in positions 2-3 represent the phase in the election process.
- Digits in positions 4-5 represent the step inside the phase.
- Digits in positions 6-10 represent candidate ID. In broadcasting messages, positions 6-10 represent leader ID.
- Digits in positions 11-15 represent candidate position. In broadcasting messages, positions 6-10 represent leader Position.

Example one explain the messages and time steps needed for the election process in hypercube with 32 nodes, as shown in Table 2.

Table-2 Messages in details for example one

Step	Source	Message	Destination
0	4	"001010000500004"	5
0	7	"001010002900007"	6
0	10	"001010001200010"	11
0	0	"001010001600000"	1
0	12	"001010001800012"	13
0	22	"001010002600022"	23
1	0	"001020001600000"	2
1	1	"001020002100001"	3
1	4	"001020000500004"	6
1	5	"001020001900005"	7
1	6	"001020002900007"	4
1	7	"001020002900007"	5
1	10	"001020001200010"	8
1	11	"001020002800011"	9
1	12	"001020001800012"	14
1	13	"001020002700013"	15
1	22	"001020002600022"	20
1	23	"001020002600022"	21
2	0	"001030001600000"	4
2	1	"001030002100001"	5
2	2	"001030003000002"	6
2	3	"001030002100001"	7
2	4	"001030002900007"	0
2	5	"001030002900007"	1
2	6	"001030002900007"	2
2	7	"001030002900007"	3
2	8	"001030001200010"	12
2	9	"001030002800011"	13
2	10	"001030001200010"	14
2	11	"001030002800011"	15
2	12	"001030001800012"	8
2	13	"001030002700013"	9
2	14	"001030001800012"	10
2	15	"001030002700013"	11
2	20	"001030002600022"	16
2	21	"001030002600022"	17
2	22	"001030002600022"	18
2	23	"001030002600022"	19
3	0	"001040002900007"	8
3	1	"001040002900007"	9
3	2	"001040003000002"	10
3	3	"001040002900007"	11
3	4	"001040002900007"	12
3	5	"001040002900007"	13
3	6	"001040003000002"	14
3	7	"001040002900007"	15

In step 0 nodes detect leader failure start the algorithm. In this example, There are 6 nodes started the algorithm. The message in the first line is as follows:
Step =0.
Source node = 4
"001010000500004" means:
0: message type is election.
01: Phase One.
01: Step One.
00005: Candidate ID.
00004: Candidate Position.
Destination = 5.

In step 1, nodes that aware of the election send the election message through link 2.

In step 2, nodes that are aware of the election process send the election message through link 3.

The process is the same until step 5

3	8	"001040001800012"	0
3	9	"001040002800011"	1
3	10	"001040001800012"	2
3	11	"001040002800011"	3
3	12	"001040001800012"	4
3	13	"001040002800011"	5
3	14	"001040001800012"	6
3	15	"001040002800011"	7
3	16	"001040002600022"	24
3	17	"001040002600022"	25
3	18	"001040002600022"	26
3	19	"001040002600022"	27
3	20	"001040002600022"	28
3	21	"001040002600022"	29
3	22	"001040002600022"	30
3	23	"001040002600022"	31
4	0	"001050002900007"	16
4	1	"001050002900007"	17
4	2	"001050003000002"	18
4	3	"001050002900007"	19
4	4	"001050002900007"	20
4	5	"001050002900007"	21
4	6	"001050003000002"	22
4	7	"001050002900007"	23
4	8	"001050002900007"	24
4	9	"001050002900007"	25
4	10	"001050003000002"	26
4	11	"001050002900007"	27
4	12	"001050002900007"	28
4	13	"001050002900007"	29
4	14	"001050003000002"	30
4	15	"001050002900007"	31
4	16	"001050002600022"	0
4	17	"001050002600022"	1
4	18	"001050002600022"	2
4	19	"001050002600022"	3
4	20	"001050002600022"	4
4	21	"001050002600022"	5
4	22	"001050002600022"	6
4	23	"001050002600022"	7
4	24	"001050002600022"	8
4	25	"001050002600022"	9
4	26	"001050002600022"	10
4	27	"001050002600022"	11
4	28	"001050002600022"	12
4	29	"001050002600022"	13
4	30	"001050002600022"	14
4	31	"001050003100031"	15
5	1	"002010002900007"	0

5	3	"002010002900007"	2
5	5	"002010002900007"	4
5	7	"002010002900007"	6
5	9	"002010002900007"	8
5	11	"002010002900007"	10
5	13	"002010002900007"	12
5	15	"002010003100031"	14
6	2	"002020003000002"	0
6	6	"002020003000002"	4
6	10	"002020003000002"	8
6	14	"002020003100031"	12
7	4	"002030003000002"	0
7	12	"002030003100031"	8
8	8	"002040003100031"	0
Start Broadcasting			
9	0	"103010003100031"	1
10	0	"103020003100031"	2
10	1	"103020003100031"	3
11	0	"103030003100031"	4
11	1	"103030003100031"	5
11	2	"103030003100031"	6
11	3	"103030003100031"	7
12	0	"103040003100031"	8
12	1	"103040003100031"	9
12	2	"103040003100031"	10
12	3	"103040003100031"	11
12	4	"103040003100031"	12
12	5	"103040003100031"	13
12	6	"103040003100031"	14
12	7	"103040003100031"	15
13	0	"103050003100031"	16
13	1	"103050003100031"	17
13	2	"103050003100031"	18
13	3	"103050003100031"	19
13	4	"103050003100031"	20
13	5	"103050003100031"	21
13	6	"103050003100031"	22
13	7	"103050003100031"	23
13	8	"103050003100031"	24
13	9	"103050003100031"	25
13	10	"103050003100031"	26
13	11	"103050003100031"	27
13	12	"103050003100031"	28
13	13	"103050003100031"	29
13	14	"103050003100031"	30
13	15	"103050003100031"	31

Phase Two is started in time step 5.

Phase Three is started in time step 9.
Message type is 1 means broadcast message.
Step 9: node 0 send leader message to node 1.
Step 10: nodes 0,1 send leader message to nodes 2,3 and so until this message is reached to all nodes in the hypercube.

Example 2 :

This example shows the contents of the buffers during algorithm steps in addition to the messages. The size of hypercube is 16 nodes. The number of nodes detecting a failure is eight nodes. The messages in each step are listed first followed by the contents of the buffers. Tables from 3 to 13 shows the messages used and below each table the contents of nodes buffers after each step.

Table-3 Step 0 in example 2

Step	Source	Message	Destination
0	1	"001010000300001"	0
0	11	"001010001300011"	10
0	6	"001010000400006"	7
0	7	"001010001400007"	6
0	13	"001010001200013"	12
0	9	"001010000600009"	8
0	2	"001010000200002"	3
0	5	"001010000900005"	4

8 nodes detect the leader failure at the same time. These nodes send election messages to neighbors through link 1. These messages will reach to destinations buffers.

```
"node number 0"
"001010000300001"
"node number 1"
"node number 2"
"node number 3"
"001010000200002"
"node number 4"
"001010000900005"
"node number 5"
"node number 6"
"001010001400007"
"node number 7"
"001010000400006"
"node number 8"
"001010000600009"
"node number 9"
"node number 10"
"001010001300011"
"node number 11"
"node number 12"
"001010001200013"
"node number 13"
"node number 14"
"node number 15"
```

Nodes 0, 10,7,6,12,8,3 and 4 were received the election messages

Table-4 Step 1 in example 2

Step	Source	Message	Destination
1	0	"001020000700000"	2
1	1	"001020000300001"	3
1	2	"001020000200002"	0
1	3	"001020001100003"	1
1	4	"001020001000004"	6
1	5	"001020000900005"	7
1	6	"001020001400007"	4
1	7	"001020001400007"	5
1	8	"001020000800008"	10
1	9	"001020000600009"	11
1	10	"001020001500010"	8
1	11	"001020001300011"	9
1	12	"001020001200013"	14
1	13	"001020001200013"	15

16 nodes that aware of the leader failure send election messages to neighbors through link 2. These messages will reach to destinations buffers.

```
"node number 0"
"001020000200002"
"node number 1"
"001020001100003"
"node number 2"
"001020000700000"
"node number 3"
"001020000300001"
"node number 4"
"001020001400007"
"node number 5"
"001020001400007"
"node number 6"
"001020001000004"
"node number 7"
"001020000900005"
"node number 8"
"001020001500010"
"node number 9"
"001020001300011"
"node number 10"
"001020000800008"
"node number 11"
"001020000600009"
"node number 12"
"node number 13"
"node number 14"
"001020001200013"
"node number 15"
"001020001200013"
```

Destination buffers that received the election messages in step 1.

Table-5 Step 2 in example 2

Step	Source	Message	Destination
2	0	"001030000700000"	4
2	1	"001030001100003"	5
2	2	"001030000700000"	6
2	3	"001030001100003"	7
2	4	"001030001400007"	0
2	5	"001030001400007"	1
2	6	"001030001400007"	2
2	7	"001030001400007"	3
2	8	"001030001500010"	12
2	9	"001030001300011"	13
2	10	"001030001500010"	14
2	11	"001030001300011"	15
2	12	"001030001200013"	8
2	13	"001030001200013"	9
2	14	"001030001200013"	10
2	15	"001030001200013"	11

The election process continues as above through link 3

```
"node number 0"
"001030001400007"
"node number 1"
"001030001400007"
"node number 2"
"001030001400007"
"node number 3"
"001030001400007"
"node number 4"
"001030000700000"
"node number 5"
"001030001100003"
"node number 6"
"001030000700000"
"node number 7"
"001030001100003"
"node number 8"
"001030001200013"
"node number 9"
"001030001200013"
"node number 10"
"001030001200013"
"node number 11"
"001030001200013"
"node number 12"
"001030001500010"
"node number 13"
"001030001300011"
"node number 14"
"001030001500010"
```

Buffers contents after step 2 in phase 1

"node number 15"
"001030001300011"

Table-6 Step 3 in example 2

Step	Source	Message	Destination
3	0	"001040001400007"	8
3	1	"001040001400007"	9
3	2	"001040001400007"	10
3	3	"001040001400007"	11
3	4	"001040001400007"	12
3	5	"001040001400007"	13
3	6	"001040001400007"	14
3	7	"001040001400007"	15
3	8	"001040001500010"	0
3	9	"001040001300011"	1
3	10	"001040001500010"	2
3	11	"001040001300011"	3
3	12	"001040001500010"	4
3	13	"001040001300011"	5
3	14	"001040001500010"	6
3	15	"001040001300011"	7

Messages in step 3 phase 1

"node number 0"
"001040001500010"
"node number 1"
"001040001300011"
"node number 2"
"001040001500010"
"node number 3"
"001040001300011"
"node number 4"
"001040001500010"
"node number 5"
"001040001300011"
"node number 6"
"001040001500010"
"node number 7"
"001040001300011"
"node number 8"
"001040001400007"
"node number 9"
"001040001400007"
"node number 10"
"001040001400007"
"node number 11"
"001040001400007"
"node number 12"
"001040001400007"
"node number 13"
"001040001400007"

Buffers contents after step 4 in phase 1

```
"node number 14"
"001040001400007"
"node number 15"
"001040001400007"
```

Table-7 Step 4 in example 2

Step	Source	Message	Destination
4	1	"002010001400007"	0
4	3	"002010001400007"	2
4	5	"002010001400007"	4
4	7	"002010001400007"	6

Messages in step 1 phase 2

```
"node number 0"
"002010001400007"
"node number 1"
"node number 2"
"002010001400007"
"node number 3"
"node number 4"
"002010001400007"
"node number 5"
"node number 6"
"002010001400007"
"node number 7"
"node number 8"
"001040001400007"
"node number 9"
"001040001400007"
"node number 10"
"001040001400007"
"node number 11"
"001040001400007"
"node number 12"
"001040001400007"
"node number 13"
"001040001400007"
"node number 14"
"001040001400007"
"node number 15"
"001040001400007"
```

Buffers contents after step 1in phase 2

Table-8 Step 5 in example 2

Step	Source	Message	Destination
5	2	"002020001500010"	0
5	6	"002020001500010"	4

Messages in step 2 phase 2

```
"node number 0"
"002020001500010"
"node number 1"
"node number 2"
```

```
"node number 3"
"node number 4"
"002020001500010"
"node number 5"
"node number 6"
"node number 7"
"node number 8"
"001040001400007"
"node number 9"
"001040001400007"
"node number 10"
"001040001400007"
"node number 11"
"001040001400007"
"node number 12"
"001040001400007"
"node number 13"
"001040001400007"
"node number 14"
"001040001400007"
"node number 15"
"001040001400007"
```

Buffers contents after step 2 in phase 2

Table-9 Step 6 in example 2

Step	Source	Message	Destination
6	4	"002030001500010"	0

Messages in step 3 phase 2

```
"node number 0"
"002030001500010"
"node number 1"
"node number 2"
"node number 3"
"node number 4"
"node number 5"
"node number 6"
"node number 7"
"node number 8"
"001040001400007"
"node number 9"
"001040001400007"
"node number 10"
"001040001400007"
"node number 11"
"001040001400007"
"node number 12"
"001040001400007"
"node number 13"
"001040001400007"
"node number 14"
```

Buffers contents after step 3 in phase 2

```
"001040001400007"
"node number 15"
"001040001400007"
```

Table-10 Step 7 in example 2

Step	Source	Message	Destination
7	0	"103010001500010"	1

Messages in step 1 phase 3

```
"node number 0"
"node number 1"
"103010001500010"
"node number 2"
"node number 3"
"node number 4"
"node number 5"
"node number 6"
"node number 7"
"node number 8"
"001040001400007"
"node number 9"
"001040001400007"
"node number 10"
"001040001400007"
"node number 11"
"001040001400007"
"node number 12"
"001040001400007"
"node number 13"
"001040001400007"
"node number 14"
"001040001400007"
"node number 15"
"001040001400007"
```

Buffers contents after step 1in phase 3

Table-11 Step 8 in example 2

Step	Source	Message	Destination
8	0	"103020001500010"	2
8	1	"103020001500010"	3

Messages in step 2 phase 3

```
"node number 0"
"node number 1"
"node number 2"
"103020001500010"
"node number 3"
"103020001500010"
"node number 4"
"node number 5"
"node number 6"
"node number 7"
"node number 8"
"001040001400007"
```

Buffers contents after step 2 in phase 3

```
"node number 9"
"001040001400007"
"node number 10"
"001040001400007"
"node number 11"
"001040001400007"
"node number 12"
"001040001400007"
"node number 13"
"001040001400007"
"node number 14"
"001040001400007"
"node number 15"
"001040001400007"
```

Table-12 Step 9 in example 2

Step	Source	Message	Destination
9	0	"103030001500010"	4
9	1	"103030001500010"	5
9	2	"103030001500010"	6
9	3	"103030001500010"	7

Messages in step 3 phase 3

```
"node number 0"
"node number 1"
"node number 2"
"node number 3"
"node number 4"
"103030001500010"
"node number 5"
"103030001500010"
"node number 6"
"103030001500010"
"node number 7"
"103030001500010"
"node number 8"
"001040001400007"
"node number 9"
"001040001400007"
"node number 10"
"001040001400007"
"node number 11"
"001040001400007"
"node number 12"
"001040001400007"
"node number 13"
"001040001400007"
"node number 14"
"001040001400007"
"node number 15"
"001040001400007"
```

Buffers contents after step 3 in phase 3

Table-13 Step 10 in example 2

Step	Source	Message	Destination
10	0	"103040001500010"	8
10	1	"103040001500010"	9
10	2	"103040001500010"	10
10	3	"103040001500010"	11
10	4	"103040001500010"	12
10	5	"103040001500010"	13
10	6	"103040001500010"	14
10	7	"103040001500010"	15

Messages in step 4 phase 3

```
"node number 0"
"node number 1"
"node number 2"
"node number 3"
"node number 4"
"node number 5"
"node number 6"
"node number 7"
"node number 8"
"001040001400007"
"103040001500010"
"node number 9"
"001040001400007"
"103040001500010"
"node number 10"
"001040001400007"
"103040001500010"
"node number 11"
"001040001400007"
"103040001500010"
"node number 12"
"001040001400007"
"103040001500010"
"node number 13"
"001040001400007"
"103040001500010"
"node number 14"
"001040001400007"
"103040001500010"
"node number 15"
"001040001400007"
"103040001500010"
```

Buffers contents after step 4 in phase 3. the Some buffers contains two messages because the first was not use in phase two.

The number of messages in example 2 are 76 message and the number of time steps are 11.

Example 3:

This example shows a leader election in large hypercube with N = 2048, with diameter equal to 11 and the number of nodes that detects a leader failure are 100. The messages for the algorithm exceed 375 pages so Appendix 2 will show just step 10 in the algorithm execution.

The results from example 3 are :

Number of messages = 17338 Time steps = 31

These results ensure that the number of messages does not exceed (O (N)) in the simple case and (O(NLog(N))) in the worst case. It is also ensures that the time steps is always the same (O(Log(N))). The results in the simulation are better than the previous works in time steps and number of messages.

5.3.2.3 Simulation Survey:

The following tables show different states (inputs for the simulation) using two hypercubes. First table for hypercube with size equal 16 nodes, and the second with size equal 1024 nodes:

- **State 1:** Table-14 summaries the inputs and the results when the simulation was executed using four dimensional hypercube. The first column lists trial number. Column two contains number of nodes, column three contains nodes that detect leader failure, and column 4 and 5 contain simulation results (number of messages and time steps). Ten trails were recorded as follows:

Table-14 Simulation inputs and results for different states when N = 16

trial	Number of Nodes detect failure		messages	time steps
	Number	nodes		
1	1	4	37	11
2	15	8,7,11,13,0,15,2,10,4,1,5,12,6,14,3	85	11
3	5	14,11,12,6,7	63	11
4	7	1,5,13,11,8,9,4	71	11
5	3	4,12,10	59	11
6	4	11,12,7,5	62	11
7	2	13,12	38	11
8	7	2,1,5,10,7,6,11	67	11
9	8	5,4,3,13,12,6,9,11	74	11
10	6	14,12,6,1,11,9	72	11

- **State 2:** Table-15 summaries the inputs and the results when the simulation was executed using ten dimensional hypercube. As in state 1. The first column lists trial number. Column two contains number of nodes, column three contains nodes that detect leader failure, and column 4 and 5 contain simulation results (number of messages and time steps). Ten trails were recorded as follows:

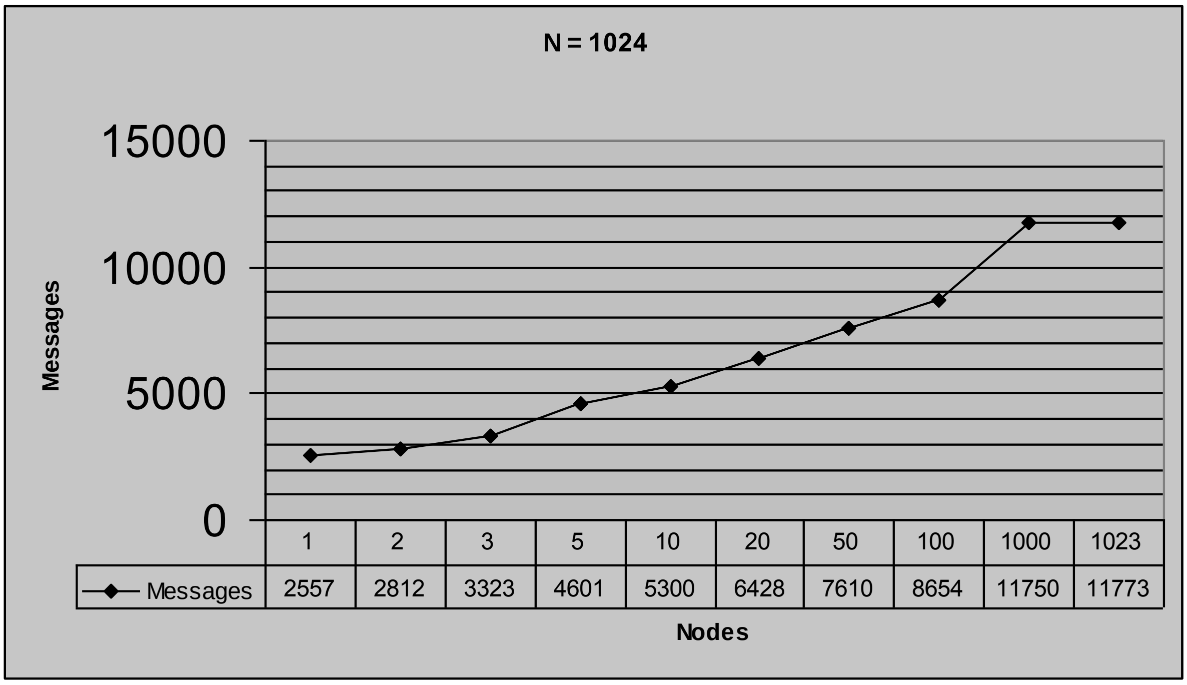
Table-15 Simulation inputs and results for different states when N = 1024

trial	Number of Nodes detect failure		messages	time steps
	Number	nodes		
1	1	154	2557	29
2	100	large number	8654	29
3	5	582,977,228,852,716	4601	29
4	10	589,699,171,486,675	5300	29
5	1023	large number	11773	29

6	1000	large number	11750	29
7	2	492,318	2812	29
8	20	large number	6428	29
9	50	large number	7610	29
10	3	300,200,112	3323	29

It can be seen from the previous tables that:

- Time steps are fixed for the leader election process in hypercube, regardless the number of nodes detecting the failure.
- On the other hand the number of messages is affected by the number of nodes detect the failure. Figure-19 shows that it increases as the number of nodes increases.



Nodes	1	2	3	5	10	20	50	100	1000	1023
Messages	2557	2812	3323	4601	5300	6428	7610	8654	11750	11773

Figure-23 Relation between number of nodes detect failure and number of messages

From Figure-19 it can be seen that the number of messages are low when the leader failure is detected by one node only. It is also obvious that the number of messages increase drastically as the number of node detecting a leader failure is more than one node.

5.4 Conclusion.

Chapter Five introduces a performance evaluation, and simulation program to validate the proposed algorithms. This chapter computed the number of messages and time steps needed by the algorithms to elect a new leader. The summations of the number of messages and time steps were represented by big (O) notation. When the hypercube size is (N) nodes, it uses (O(N)) messages within (O(Log(N))) time steps to solve the problem when the failure is detected by one node in the simple case. In the most complicated case, when failure is detected by (N-1) nodes, the problem is solved with (O (N Log (N))) messages within (O (Log (N))) time steps.

This chapter also presented a simulation program to validate the first algorithm. It proves that time steps are fixed for the leader election process in hypercube, regardless the number of nodes detecting the failure. On the other hand the number of messages is affected by the number of nodes that detects the failure. The number of messages increases relationally as the number of nodes increases. Details for the dissertation Results will be presented in the next chapter.

CHAPTER SIX
CONCLUSIONS AND FUTURE WORKS

6.1 Introduction.

This work presents two distributed solutions to the leader failure problem in Hypercube networks. The first solution is a distributed algorithm to elect a new leader in hypercube with minimum number of messages and time steps. The second solution is a leader election algorithm with the presences of one link failure. This algorithm solves the leader failure even when there is one link failure. This chapter concludes the dissertation and presents comparisons with previous work. It provides analyses for the simulation results for the first algorithm. Future work will be explained in the last section.

6.2 Results.

In this work, complexity analysis was used to evaluate performance for the algorithms. 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 each algorithm. These totals were translated to big O notation to express the complexity.

This section summarizes the steps of the proposed algorithms and gives the results.

6.2.1 Results of the First Algorithm.

The first algorithm consists of three phases. Each phase has a number of time steps and messages. Phase one is initiated when one or more nodes detects the leader failure, this initiates an election process. This phase reduces the numbers of participated nodes in the election process to N/2 nodes who are aware of the election. The second phase uses the reduction all-to-one communication operation (with the comparison process in each step) to have the result in one node. Finally, in the third phase, this node broadcasts the leader message to all nodes in the hypercube using broadcast one-to-all communication operation.

Algorithm performance was evaluated by calculating the number of messages and time steps for the three phases. When the hypercube size is N nodes, it uses (O(N))messages within (O(Log(N))) time steps to solve the problem when the failure is detected by one node in the simple case. In the most complicated case, when failure is detected by (N-1) nodes, the problem is solved with (O(N Log(N))) messages within (O(Log(N))) time steps.

6.2.2 Results of the Second Algorithm.

This algorithm consists of three phases. Each phase has many steps and messages. Phase one is initiated when one or more nodes detect a leader failure, It initiate the election process. This phase reduces the count of participated nodes in the election process to N/2 nodes aware of the election. In phase one, this algorithm considered the probability of the presence of one link failure. The second phase uses the reduction all-to-one communication operation "with additional steps to tolerate link failure" to have the result in one node. Finally, in the third phase, this node broadcasts leader message to all nodes in the hypercube using broadcast one-to-all communication operation, “with additional steps to tolerate link failure”.

Algorithm performance was evaluated by calculating the number of messages and time steps for the three phases. It uses O(N) messages within O(Log(N)) time steps to solve the problem when the failure is detected by one node in the simple case. In the most complicated case when failure is detected by N-1 nodes the problem is solved with O(N Log(N)) messages within O(Log(N)) time steps.

6.4 Simulation Results.

This work presented a simulation program that showed the application of the first algorithm. Many trials and examples were made using this simulation. Trials and its results were analyzed and explained. The following notes summarize the simulation results:

- Time steps are fixed for the leader election process in hypercube, regardless of the number of nodes detecting the failure.
- On the other hand the number of messages is affected by the number of nodes that detected leader failure. The number of messages increases as the number of nodes that detect leader failure increases.

6.5 Comparison between the Proposed Algorithm and Previous Works.

Four election algorithms in hypercube were reviewed. Our algorithm is better in time steps using (O(Log(N))) steps over the whole algorithm**.** The **Gerard** algorithm uses **(O(Log^2N)) steps. Flocchini and Mans** algorithm uses **(Θ(Log^3N)) steps**. **Castorino and Ciccarella** in 1999 use Log(N) Steps, when only one node detect the failure and they didn't take the worst case of the algorithm. **Dobrev and Ruzecka** didn't concentrate or calculate the time steps**.** The **Dobrev and Ruziicka** algorithm focused on a number of messages and ignored the time steps to get (NLogLog(N)) message complexity. The message size is better in Gerard algorithm where all the others are the same. Table-16 explains the differences between proposed algorithm and the previous works:

Table-16 Comparison between proposed and previous algorithms

Algorithm	Number of Messages	Time Steps	Message Size /bits	Year
Proposed algorithm	**O(N)**	**O(Log(N))**	**2Log N + Log(Log (N)) + 3**	**2006**
Flocchini and Mans	**Ω(N)**	**Θ(Log^3N)**	**3LogN +Log(Log(N))**	**1996**
Gerard	**O(N)**	**O(Log^2N)**	**O(Log(Log(N))**	**1993**
Dobrev and Ruzecka	**O(N(Log(Log(N))))**	**Not computed**	**Not Computed**	**1997**
Castorino and Ciccarella	**O(N)**	**O(Log(N))**	**3LogN**	**1999**

6.6 Future Works.

For **Future Work,** the work presented can be improved by carrying out the following:

- Design algorithm to solve the leader failure in meshes networks with the presence of link failure.
- Design an algorithm to solve leader failure in hypercube when the ID is not distinguished
- Design algorithm to solve the leader failure in trees networks with the presence of link failure.
- Write a survey of all election algorithms for each topology.
- Change the environment for meshes and other topologies to work under wireless communications.

Publications:

Ajluni N., Refai, M.,(2006), **Leader Election Algorithm in Hypercubes with the Presence of One Link Failure**, The 2006 International Conference on Parallel and Distributed Processing Techniques and Applications(PDPTA'06: Jun 26-29, Las Vegas, USA.

Refai, M., Ajluni N., (2006), **New Leader Election Algorithm in Hypercubes with Minimum time steps and number of messages**, European University of Lefke, 4th FAE International Symposium, Gemikonag, TRNC , 30 November- 1 December.

References

Abu-Amara, H. and Lokre, J.(1994) **Election in Asynchronous Complete Networks with Intermittent Link Failures,** IEEE Transactions on Computers, Vol. 34 No. 7, July , pp. 778-788.

Abu-Amara, H., (1988), **Fault- Tolerant Distributed Algorithm For Election in Complete Networks**, IEEE Transaction on Computers, 37 v4: PP 449-453.

Afek Y. and Gafni E. ,(1991), **Time and Message Bounds for Electing in Synchronous and Asynchronous Complete Networks,** SAIM Journal on Computing, 20 v 2 : PP 376-394.

Antonoiu, G. and Srimani, K.**(1996)A Self-Stabilizing Leader Election Algorithm for Tree Graphs,** Journal of Parallel and Distributed Computing, 34, Article No. 0059, pp. 227-232.

Castorino A. and Ciccarella G.,(1999), **Optimal-Election Algorithms for Hypercube**, **Seventh Euromicro Workshop on Parallel and Distributed Processing** p. 221, 1999.

Christof F.,Flaviv C., (1999), **A Highly Available Local Leader Election Service,** IEEE Transaction of Software Engineering.

Ciccarella G. and Patricelli, (1994), **A Distributed System Architecture for Embedded Control Systems**, Proc. Of Euromicro Workshop on Parallel and Distributed Processing, Malaga: PP 392-399, IEEE Computer Society Press.

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

Deitel and Deitel ,Nieto,T.R.,(1999). **Visual Basic 6,**Prentice Hall, New Jersey 07458, USA.

Devillers M., Griffioen D., Romijn J. and Vaandrager F., (2004) , **Verification of Leader Election Protocol, Formal Method Applied to IEEE 1394**, Springer International journal on Software Tools for Tecknology Transfer(STTT), December 2004.

Dobrev S. and Ruzicka P.., (1997) **Linear Broadcasting and N Log Log(N) Election in Unoriented Hypercube**. Institute of Informatics, Faculty of Mathematics and Physics, Comenius university, Slovak Republic.

Dolev S., Israeli A. and Moran S., (1997), **Uniform Dynamic Self-Stabilizing Leader Election**, IEEE Transaction on Parallel and Distributed Systems, VOL 8,NO.4, April .PP 424-440.

Duato, J. Yalamanchili, S. and Ni, L. , (1997) **Interconnection Networks an Engineering Approach,** IEEE Computer Society, The Institute of Electronic Engineers, Inc, Los Alamitos, California.

Fetzer, C., Raynal, M., and Tronel, F.(2000) . **A Failure Detection Protocol Based on A Lazy Approach**, Research Report 1367, IRISA, November.

Flocchini, P. and Mans, B. (1996).**Optimal Elections in Labeled Hypercube,** Journal of Parallel and Distributed Computing 33, Article No. 0026, pp. 76-83.

Fokkink .W and Jun Pang, (2004) **Simplifying Itia-Rodeh Leader Election for Anonymous Ring**, Vrije University, Amsterdam.

Foster I.(1994).**Designing and Building Parallel Programs**, Addison-Wesley Publishing Company, USA.

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

Gerard ,T.,(1993). **Linear Election for Oriented Hypercube**, Technical Report TR-RUU-CS-93-39, Department of computer Science, Utrecht University, The Netherlands.

Garcia- Molina, H ,(1982) **, Electing in a Distributed Computing Systems**, IEEE Trans. Comp. ,Vol 31, No.1,PP.48-59,jan 1982.

Jean-Franqois Marckert (2005), **Quasi-Optimal Leader Election Algorithms in Radio Network with Log-Logarithmic Awake Time Slots**, F.chyzak(ed.),INRIA,pp.97-00.

Junguk L. and Geneva G., (1996), **A Distributed Election Protocol for Unreliable Networks,** Journal of Parallel and Distributed Computing, 35, PP 35-42**.**

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

Larrea M., Ar´evalo S., and Fern´andez A.(1999), **Efficient Algorithms to Implement Unreliable Failure Detectors in Partially Synchronous Systems**. In Proceedings of the 13th International Symposium on Distributed Algorithms(DISC99), pages 34–48,

Bratislava, September.

Larrea M., Fernandez A., and Arevalo S.,(2000), **Optimal Implementation of the Weakest Failure Detector for Solving Consensus**. In Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems, SRDS 2000, pages 52–59, Nurenberg, Germany, October.

Larrea, M., Fernandez A., and Arevalo S.(2000) **Eventually Consistent Failure Detectors**. In Brief Announcement the 14th International Symposium on Distributed Algorithms(DISC00), Toledo, October.

Leeuwen, J. and Tan, R.,(1987), **An Improved Upper bound for Distributed Election in Bidirectional Rings of Processors,** Distributed Computing, 2 , PP 149-160.

Levitin A**., (**2003), **Introduction to The Design and Analysis of Algorithms,** Addison Wesley Company, USA.

Mattern, F. ,(1989), **Message Complexity of Simple Ring-Based Election Algorithms – An Empirical Analysis**, in IEEE 9th Int. Conf. Distributed Computing Syst. PP 94-100.

Michel R**., (**1985), **Distributed Algorithms And Protocols,** John Wiley & sons Publishing Company, New York,

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

Mostefaoui A. and Raynal M.,(2000), **Leader-Based Consensus**. Research Report 1372, IRISA, December.

Navneet M., Jennifer L., Welch, Nitin V., (2001), **Leader Election Algorithms for Mobile Ad Hoc Networks,** Supported in Part by NSF grant CCR-9972235.

Ostrovsky, R., Rajagoplan, S., and Vazirani, U.,(1994), **Simple and Efficient Leader Election in the Full Information Model**. In Proceedings of the Twenty-Sixth Annual

ACM Syposium on Theory of Computing.

Power H.., (1999), **Algorithms and Application in Parallel Computing,** WIT Press/Computational Mechanics Publications, USA.

Quinn, J.,(1994) **Parallel Computing Theory and Practice**, 2nd Edition, McGrow-Hill, Inc.

Raynal, M.,(1984), **Distributed Algorithms and Protocols**, John Wesley and Sons, New York USA.

Refai M. and Ababneh E., (2002) **Leader Election Algorithm in 3D Torus Networks**, Master Theses, Al-Albayet University – Jordan.

Richard E. and Kumarss N.,(2004), **Foundations of Algorithms Using Java PseudoCode**, Jones and Bartlett Publishers,Canada.

Russell, A., Saks, M., and Zuckerman, D.,(1999) **Lower Bounds For Leader Election And Collective Coin-Flipping In The Perfect Information Model**. In Proceedings of the Symposium on the Theory of Computing (STOC).

Russel, A. and Zuckerman D., 1998, **Perfect Information Leader Election In Log*N + O(1) Rounds**, In Proceedings of 39th Annual Symposium on Foundations of Computer Science (FOCS).

Schneider, M. (1993), **Self-Stabilization,** ACM Comput. Surveys 25(1),45-67, mar.

Shrira, L. and Goldreich, O. , (1987) , **Electing a Leader in a Ring with Link Failure**, ACTA Information, 24, PP 79-91.

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

Singh, G., (1991), **Efficient Distributed Algorithms for Leader Election in Complete Networks**, 11th IEEE Int. Conf. on Distributed Computing Systems, PP 472-479.

Singh G., (1997), **Efficient Leader Election Using Sense of Direction**, Department of Computing and Information Sciences, Kansas State University, Manhatten, KS66506.

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

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

Valerie, K. Jared, S. Vishal, S. and Erik, V. ,(2001), **Scalable Leader Election,** SODA, January 22-26, Miami, FL

Vos T.E.J. UNITY in Diversity,(2000) **A Stratified Approach to Verification of Distributed Algorithms.** PhD thesis, Utrecht University.

Yamshita M. and Kammeda T.,(1999), **Leader Election Problem on Networks in which Processor Identity Numbers are not Distinct**, IEEE Transactions on Parallel and Distributed Systems, VOL 10,No 9,September.

APPENDICES

Appendix 1
Simulation code:

1- Main Program Code

```
Option Explicit
Dim node() As New vertices



Private Sub about_Click(Index As Integer)
 form2.Show
End Sub


Private Sub Blue_Click(Index As Integer)
 Me.backcolor = vbBlue
End Sub

Private Sub CLS_Click(Index As Integer)
 CLS
End Sub
Private Sub cmdcls_Click()
 CLS
End Sub

Private Sub CMDExit_Click()
End
End Sub

Private Sub CMDstep_Click()
   Timer1.Enabled = True
   step = True
End Sub

Private Sub def_Click(Index As Integer)
 Me.backcolor = &H8000000F
End Sub

Private Sub Nodes_status_Click()
Dim i%
Print
 Print Tab(5); " normal"; Tab(13); "leader ID"; Tab(23); "leader_pos"; Tab(35); "l_phase"
_
       ; Tab(45); "l_step"; Tab(55); "my pos"; Tab(65); "my id"; Tab(75); "C_ID"; Tab(85);
"C_pos"; "          phase 2 finished"

 Print Space(10); String(75, "_")
 For i = 0 To N - 1

   Print Tab(5); node(i).normal;
    Print Tab(13); node(i).leader_id;
   Print Tab(23); node(i).leader_pos;
    Print Tab(35); node(i).local_phase;
   Print Tab(45); node(i).local_step;
    Print Tab(55); node(i).my_pos;
    Print Tab(65); node(i).my_id;
    Print Tab(75); node(i).candidate_id;
    Print Tab(85); node(i).candidate_pos;
    Print Tab(105); node(i).phase2done;
    Print Tab(115); node(i).end_after_phase
   Next
End Sub

Private Sub CStart_Click()
  Timer1.Enabled = Not Timer1.Enabled
  step = False
End Sub
Private Sub Cbuffer_Click()
Dim i%, j%, message$, msg$

Open "D:\Documents and Settings\mohammed\Desktop\hh" For Output As #1
List2.Clear

 List2.Visible = True

    For i = 0 To N - 1

   msg = "node number " & node(i).my_pos
    List2.AddItem (msg)

     node(i).show_buffer
     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()
 configration
 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
30  ff = InputBox("enter the number of process that detect failure")
   If ff > N Then MsgBox ("rong number tryagain"): GoTo 30
  leader_failure (ff)
End Sub

Private Sub Orange_Click(Index As Integer)
Me.backcolor = &H80C0FF
End Sub

Private Sub Red_Click(Index As Integer)
 Me.backcolor = vbRed
End Sub

Private Sub setconfigration_Click(Index As Integer)
configration

End Sub

Private Sub configration()
Randomize
 Open "D:\Documents and Settings\mohammed\Desktop\tt" For Output As #1
 Close #1
  step = False
  time_step = 0
  Lnumberofmessage.Caption = ""
  ltimesteps.Caption = ""

  Dim x#, i%, j%, bb%, r%, flag As Boolean, B As Boolean


   Do
     x = InputBox("Enter the Number of processes", "setting", "16")
      B = Fix(Log(x) / Log(2)) < (Log(x) / Log(2))
      If B Then
           MsgBox ("not correct")
       Else
           N = x
           d = Log(N) / Log(2)
           ReDim node(N)

      End If
   Loop While B

  ReDim node(N)
    For i = 0 To N - 1
       j = -1
        r = Int(((N) * Rnd()) + 1)
        Do Until j = i
          If r = node(j + 1).my_id Then
            r = Int(((N) * Rnd()) + 1)
           j = -1
           Else
           j = j + 1
```

```
         End If

         Loop
         node(i).my_id = r
         If r = N Then leader_pos = i
         Next i
        For i = 0 To N - 1

           bb = i
           ppp = False
              node(i).normal = True
              node(i).leader_id = N
              node(i).leader_pos = leader_pos
              node(i).local_phase = 0
             node(i).local_step = 0
             node(i).my_pos = i
             node(i).candidate_id = 0
             node(i).candidate_pos = 0
           For j = 1 To d

              node(i).bin = node(i).bin & (bb Mod 2)
              bb = bb \ 2
              Next j
      node(i).bin = Format(StrReverse(node(i).bin), String(d, "0"))


    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%, arc%, message$, dist%, j%, phase3step%, msg As String, mm As String
 List2.Visible = False
 ltimesteps.Caption = time_step
 Open "D:\Documents and Settings\mohammed\Desktop\tt" For Append As #1
  For i = 0 To N - 1
        If Not node(i).isempty Then
           If ppp And i >= N / 2 Then GoTo 20
            Do Until node(i).isempty Or node(i).read_btime = time_step
                message = node(i).read_buffer
                node(i).leader_id = -1
                node(i).leader_pos = -1
                    node(i).local_phase = Val(Mid(message, 2, 2))
                    If node(i).local_step <= Val(Mid(message, 4, 2)) Then
                      node(i).local_step = Val(Mid(message, 4, 2))
```

```
                        If node(i).candidate_id < node(i).my_id Then
                          node(i).candidate_id = node(i).my_id
                          node(i).candidate_pos = node(i).my_pos
                        End If
                        If node(i).candidate_id < Val(Mid(message, 6, 5)) Then
                          node(i).candidate_id = Val(Mid(message, 6, 5))
                          node(i).candidate_pos = Val(Mid(message, 11, 5))
                        End If
                        If node(i).local_step = d And node(i).local_phase = 1 Then
                           node(i).local_step = 1
                           node(i).local_phase = 2
                            If i < N / 2 Then ppp = True
                             If Mid(node(i).bin, (d - node(i).local_step) + 1) = "1" And Not
node(i).phase2done Then
                                dist = finddis(i, node(i).local_step)
                                message = Format(0, "0") & Format(node(i).local_phase, "00") &
Format(node(i).local_step, "00") & Format(node(i).candidate_id, "00000") &
Format(node(i).candidate_pos, "00000")
                                Call node(dist).add(message, time_step)
                               ' Print node(i).my_pos; "send  "; message; "to"; dist
                                Write #1, time_step; "     "; node(i).my_pos; "send  "; message; "to";
dist
                                 mm = node(i).my_pos
                                msg = time_step & Space(10) & node(i).my_pos & Space(11 -
Len(mm)) & "send" & Space(10) & message & Space(10) & "to" & Space(10) & dist
                      List1.AddItem (msg)
                                node(i).phase2done = True

                              End If
                              GoTo 20
                        End If

                    End If
                    If node(i).local_phase = 3 Then
                       node(i).leader_id = node(i).candidate_id
                       node(i).leader_pos = node(i).candidate_pos
                     End If
               Loop

        End If
        If node(i).local_phase = 1 Then
             If node(i).local_step = d Then


                     If node(i).candidate_id < node(i).my_id Then
                          node(i).candidate_id = node(i).my_id
                          node(i).candidate_pos = node(i).my_pos
                        End If
                        If node(i).candidate_id < Val(Mid(message, 6, 5)) Then
                          node(i).candidate_id = Val(Mid(message, 6, 5))
                          node(i).candidate_pos = Val(Mid(message, 11, 5))
                        End If
```

```
                  Else

                node(i).local_step = node(i).local_step + 1
                If node(i).local_step = d And i >= N / 2 Then
                  node(i).end_after_phase = True
                End If
                message = Format(0, "0") & Format(node(i).local_phase, "00") &
Format(node(i).local_step, "00") & Format(node(i).candidate_id, "00000") &
Format(node(i).candidate_pos, "00000")
                arc = CInt(node(i).local_step)
                    dist = finddis(i, arc)
                      Call node(dist).add(message, time_step)
                     ' Print node(i).my_pos; "send  "; message; "to"; dist
                    Write #1, time_step; "     "; node(i).my_pos; "send  "; message; "to"; dist
                     mm = node(i).my_pos
                    msg = time_step & Space(10) & node(i).my_pos & Space(11 - Len(mm))
& "send" & Space(10) & message & Space(10) & "to" & Space(10) & dist
                    List1.AddItem (msg)
              End If
        End If

    If node(i).local_phase = 2 And Not node(i).phase2done Then
         node(i).local_step = node(i).local_step + 1
          If node(i).local_step = d Then
            node(i).local_phase = 3
            node(i).local_step = 1
            node(i).leader_id = node(i).candidate_id
            node(i).leader_pos = node(i).leader_pos
            dist = finddis(i, node(i).local_step)

            message = Format(1, "0") & Format(node(i).local_phase, "00") &
Format(node(i).local_step, "00") & Format(node(i).candidate_id, "00000") &
Format(node(i).candidate_pos, "00000")
                      Call node(dist).add(message, time_step)
                     ' Print node(i).my_pos; "send  "; message; "to"; dist
                    Write #1, time_step; "     "; node(i).my_pos; "send  "; message; "to"; dist
                     mm = node(i).my_pos
                    msg = time_step & Space(10) & node(i).my_pos & Space(10) & "send" &
Space(10) & message & Space(10) & "to" & Space(10) & dist
                    List1.AddItem (msg)
              GoTo 20

          End If

         If Mid(node(i).bin, (d - node(i).local_step) + 1, 1) = "1" And Not
node(i).phase2done Then

         dist = finddis(i, node(i).local_step)

            message = Format(0, "0") & Format(node(i).local_phase, "00") &
Format(node(i).local_step, "00") & Format(node(i).candidate_id, "00000") &
Format(node(i).candidate_pos, "00000")
                      Call node(dist).add(message, time_step)
                    ' Print node(i).my_pos; "send  "; message; "to"; dist
                      mm = node(i).my_pos
                   Write #1, time_step; "     "; node(i).my_pos; "send  "; message; "to"; dist
                    msg = time_step & Space(10) & node(i).my_pos & Space(11 - Len(mm))
& "send" & Space(10) & message & Space(10) & "to" & Space(10) & dist
                   List1.AddItem (msg)
          node(i).phase2done = True

       End If
    End If
    If node(i).local_phase = 3 Then
        node(i).leader_id = node(i).candidate_id
          node(i).leader_pos = node(i).candidate_pos
           node(i).local_step = node(i).local_step + 1
           If node(i).local_step = d + 1 Then

               GoTo 20
           End If
          If node(i).local_step = d + 2 Then

               Timer1.Enabled = False
                For j = 0 To N - 1
                node(j).local_phase = 0
                   node(j).local_step = 0
                  node(j).normal = True
                   node(j).phase2done = False
                   node(j).candidate_id = 0
                   node(j).candidate_pos = 0
                 Next j
                 MsgBox ("END THE SIMULATION")
               Exit For
          End If
          dist = finddis(i, node(i).local_step)

          message = Format(1, "0") & Format(node(i).local_phase, "00") &
Format(node(i).local_step, "00") & Format(node(i).candidate_id, "00000") &
Format(node(i).candidate_pos, "00000")
                     Call node(dist).add(message, time_step)
                    ' Print node(i).my_pos; "send  "; message; "to"; dist
                   Write #1, time_step; "     "; node(i).my_pos; "send  "; message; "to"; dist
                    mm = node(i).my_pos
                   msg = time_step & Space(10) & node(i).my_pos & Space(11 - Len(mm))
& "send" & Space(10) & message & Space(10) & "to" & Space(10) & dist
                   List1.AddItem (msg)

   End If


20: Next i
```

```
  Close #1
    time_step = time_step + 110:
   If step = True Then Timer1.Enabled = False
End Sub

Private Sub leader_failure(x As Integer)
 Randomize
   Dim i As Integer, r%, message$, dist%, msg As String
   time_step = 1
    node(leader_pos).my_id = 0
    Open "D:\Documents and Settings\mohammed\Desktop\tt" For Append As #1
    For i = 1 To x
     r = N * Rnd()
     If node(r).normal = True Then
     'Print " process number:  "; r; "  detect the failure"
     node(r).normal = False
      node(r).local_phase = 1
       node(r).local_step = 1
       node(r).leader_id = -1
       node(r).leader_pos = -1
       node(r).candidate_id = node(r).my_id
       node(r).candidate_pos = node(r).my_pos
       dist = finddis(r, 1)
       message = Format(0, "0") & Format(node(r).local_phase, "00") &
Format(node(r).local_step, "00") & Format(node(r).candidate_id, "00000") &
Format(node(r).candidate_pos, "00000")
        'Print node(r).my_pos; "send  "; message; "to"; dist
         Write #1, 0; "     "; node(r).my_pos; "send  "; message; "to"; dist
         msg = "0" & Space(10) & node(r).my_pos & Space(10) & "send" & Space(10) &
message & Space(10) & "to" & Space(10) & dist
                    List1.AddItem (msg)
       Call node(dist).add(message, 0)
     Else
     i = i - 1
     End If
    Next i
    Close #1
    ltimesteps.Caption = time_step
End Sub

Private Sub Yello_Click(Index As Integer)
  Me.backcolor = vbYellow
End Sub
```

2- Vertices Class Code

```
Option Explicit

 Public my_id As Integer
 Public my_pos As Integer
 Public leader_pos As Integer
 Public leader_id As Integer
 Public candidate_pos As Integer
 Public candidate_id As Integer
  Public normal As Boolean
 Public local_phase As Integer
 Public local_step As Integer
 Public bin As String
 Public phase2done As Boolean
 Public end_after_phase As Boolean
 Private Type rec
   message As String * 15
   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)
   number_of_message = number_of_message + 1
    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)
  bh = (bh + 1) Mod buffer_size
Loop


  End Sub
```

3- Module Code

```
Public N As Integer
Public d  As Integer
 Public step As Boolean
Public leader_pos%, time_step As Integer, number_of_message, ppp As Boolean

Public Function finddis(ByVal node As Integer, ByVal arc As Integer) As Integer
   Dim x() As Integer, i%, ss%, diameter%
    diameter = (Log(N) / Log(2))
   ReDim x(1 To diameter)
   For i = 1 To diameter

    x(i) = node Mod 2
    node = node \ 2
    Next i
    x(arc) = x(arc) Xor 1


    For i = 1 To diameter
      ss = ss + 2 ^ (i - 1) * x(i)
      Next
      finddis = ss
End Function
```

Appendix 2

The messages for election algorithm in hypercube with N = 2048 when 100 nodes detect leader failure. Because of large number of messages step 10 messages list down in this appendix.

Step	Source	Message	Destination
10	1504	"001110204601360"	480
10	1505	"001110203401889"	481
10	1506	"001110201701270"	482
10	1507	"001110204001803"	483
10	1508	"001110203001604"	484
10	1509	"001110204301621"	485
10	1510	"001110201701270"	486
10	1511	"001110202701731"	487
10	1512	"001110204601360"	488
10	1513	"001110203401889"	489
10	1514	"001110201701270"	490
10	1515	"001110204001803"	491
10	1516	"001110201801820"	492
10	1517	"001110204301621"	493
10	1518	"001110201701270"	494
10	1519	"001110202701731"	495
10	1520	"001110204601360"	496
10	1521	"001110203401889"	497
10	1522	"001110201701270"	498
10	1523	"001110204001803"	499
10	1524	"001110201801820"	500
10	1525	"001110204301621"	501
10	1526	"001110201701270"	502
10	1527	"001110202701731"	503
10	1528	"001110204601360"	504
10	1529	"001110203401889"	505
10	1530	"001110201701270"	506
10	1531	"001110204001803"	507
10	1532	"001110201801820"	508
10	1533	"001110204301621"	509
10	1534	"001110201701270"	510
10	1535	"001110202701731"	511
10	1536	"001110204601360"	512
10	1537	"001110203401889"	513
10	1538	"001110201701270"	514
10	1539	"001110204001803"	515
10	1540	"001110203001604"	516
10	1541	"001110204301621"	517
10	1542	"001110201701270"	518
10	1543	"001110202701731"	519
10	1544	"001110204601360"	520
10	1545	"001110203401889"	521
10	1546	"001110201701270"	522
10	1547	"001110204001803"	523
10	1548	"001110201801820"	524

10	1661	"001110204301621"	627
10	1662	"001110201701270"	628
10	1663	"001110202701731"	629
10	1664	"001110204601360"	630
10	1665	"001110203401889"	631
10	1666	"001110201701270"	632
10	1667	"001110204001803"	633
10	1668	"001110203602604"	634
10	1669	"001110204301621"	635
10	1670	"001110201701270"	636
10	1671	"001110202701731"	637
10	1672	"001110204601360"	638
10	1673	"001110203401889"	639
10	1674	"001110201701270"	640
10	1675	"001110204001803"	641
10	1676	"001110201801820"	642
10	1677	"001110204301621"	643
10	1678	"001110201701270"	644
10	1679	"001110202701731"	645
10	1680	"001110204601360"	646
10	1681	"001110203401889"	647
10	1682	"001110201701270"	648
10	1683	"001110204001803"	649
10	1684	"001110203602604"	650
10	1685	"001110204301621"	651
10	1686	"001110201701270"	652
10	1687	"001110202701731"	653
10	1688	"001110204601360"	654
10	1689	"001110203401889"	655
10	1690	"001110201701270"	656
10	1691	"001110204001803"	657
10	1692	"001110201801820"	658
10	1693	"001110204301621"	659
10	1694	"001110201701270"	660
10	1695	"001110202701731"	661
10	1696	"001110204601360"	662
10	1697	"001110203401889"	663
10	1698	"001110201701270"	664
10	1699	"001110204001803"	665
10	1700	"001110203801604"	666
10	1701	"001110204301621"	667
10	1702	"001110201701270"	668
10	1703	"001110202701731"	669
10	1704	"001110204601360"	670
10	1705	"001110203401889"	671
10	1706	"001110201701270"	672
10	1707	"001110204001803"	673
10	1708	"001110201801820"	674
10	1709	"001110204301621"	675
10	1710	"001110201701270"	676
10	1711	"001110202701731"	677
10	1712	"001110204601360"	678
10	1713	"001110203401889"	679
10	1714	"001110201701270"	680
10	1715	"001110204001803"	681
10	1716	"001110203801604"	682
10	1717	"001110204301621"	683

10	1718	"001110201701270"	694
10	1719	"001110202701731"	695
10	1720	"001110204601360"	696
10	1721	"001110203401889"	697
10	1722	"001110201701270"	698
10	1723	"001110204001803"	699
10	1724	"001110201801820"	700
10	1725	"001110204301621"	701
10	1726	"001110201701270"	702
10	1727	"001110202701731"	703
10	1728	"001110204601360"	704
10	1729	"001110203401889"	705
10	1730	"001110201701270"	706
10	1731	"001110204001803"	707
10	1732	"001110203602604"	708
10	1733	"001110204301621"	709
10	1734	"001110201701270"	710
10	1735	"001110202701731"	711
10	1736	"001110204601360"	712
10	1737	"001110203401889"	713
10	1738	"001110201701270"	714
10	1739	"001110204001803"	715
10	1740	"001110201801820"	716
10	1741	"001110204301621"	717
10	1742	"001110201701270"	718
10	1743	"001110202701731"	719
10	1744	"001110204601360"	720
10	1745	"001110203401889"	721
10	1746	"001110201701270"	722
10	1747	"001110204001803"	723
10	1748	"001110203602604"	724
10	1749	"001110204301621"	725
10	1750	"001110201701270"	726
10	1751	"001110202701731"	727
10	1752	"001110204601360"	728
10	1753	"001110203401889"	729
10	1754	"001110201701270"	730
10	1755	"001110204001803"	731
10	1756	"001110201801820"	732
10	1757	"001110204301621"	733
10	1758	"001110201701270"	734
10	1759	"001110202701731"	735
10	1760	"001110204601360"	736
10	1761	"001110203401889"	737
10	1762	"001110201701270"	738
10	1763	"001110204001803"	739
10	1764	"001110203602604"	740
10	1765	"001110204301621"	741
10	1766	"001110201701270"	742
10	1767	"001110202701731"	743
10	1768	"001110204601360"	744
10	1769	"001110203401889"	745
10	1770	"001110201701270"	746
10	1771	"001110204001803"	747
10	1772	"001110201801820"	748
10	1773	"001110204301621"	749
10	1774	"001110201701270"	750

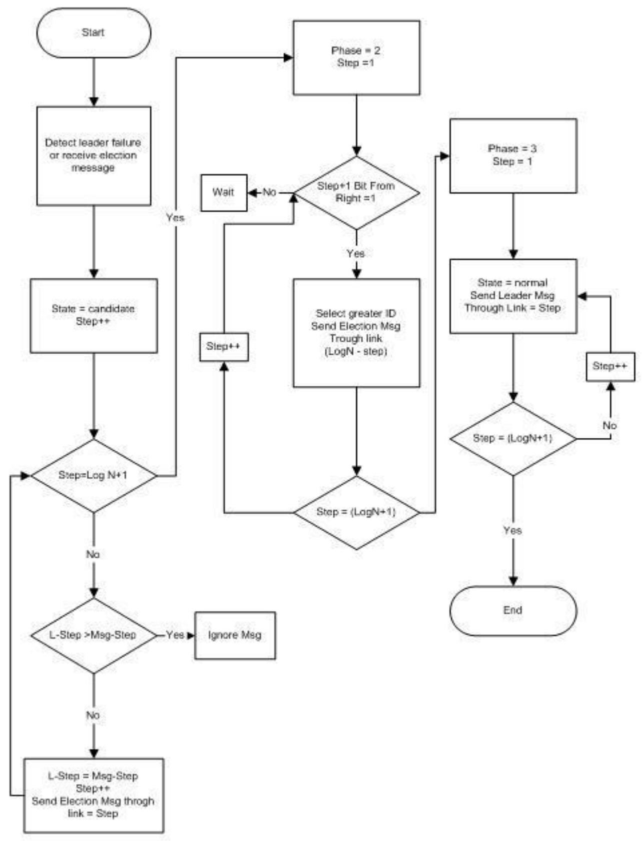
10	1775	"001110202701731"	751
10	1776	"001110204601360"	752
10	1777	"001110203401889"	753
10	1778	"001110201701270"	754
10	1779	"001110204001803"	755
10	1780	"001110201801820"	756
10	1781	"001110204301621"	757
10	1782	"001110201701270"	758
10	1783	"001110202701731"	759
10	1784	"001110204601360"	760
10	1785	"001110203401889"	761
10	1786	"001110201701270"	762
10	1787	"001110204001803"	763
10	1788	"001110201801820"	764
10	1789	"001110204301621"	765
10	1790	"001110201701270"	766
10	1791	"001110202701731"	767
10	1792	"001110204601360"	768
10	1793	"001110203401889"	769
10	1794	"001110201701270"	770
10	1795	"001110204001803"	771
10	1796	"001110203001604"	772
10	1797	"001110204301621"	773
10	1798	"001110201701270"	774
10	1799	"001110202701731"	775
10	1800	"001110204601360"	776
10	1801	"001110203401889"	777
10	1802	"001110201701270"	778
10	1803	"001110204001803"	779
10	1804	"001110201801820"	780
10	1805	"001110204301621"	781
10	1806	"001110201701270"	782
10	1807	"001110202701731"	783
10	1808	"001110204601360"	784
10	1809	"001110203401889"	785
10	1810	"001110201701270"	786
10	1811	"001110204001803"	787
10	1812	"001110203502004"	788
10	1813	"001110204301621"	789
10	1814	"001110201701270"	790
10	1815	"001110202701731"	791
10	1816	"001110204601360"	792
10	1817	"001110203401889"	793
10	1818	"001110201701270"	794
10	1819	"001110204001803"	795
10	1820	"001110201801820"	796
10	1821	"001110204301621"	797
10	1822	"001110201701270"	798
10	1823	"001110202701731"	799
10	1824	"001110204601360"	800
10	1825	"001110203401889"	801
10	1826	"001110201701270"	802
10	1827	"001110204001803"	803
10	1828	"001110203001604"	804
10	1829	"001110204301621"	805
10	1830	"001110201701270"	806
10	1831	"001110202701731"	807

Appendix 3:
Flowcharts for the proposed algorithms

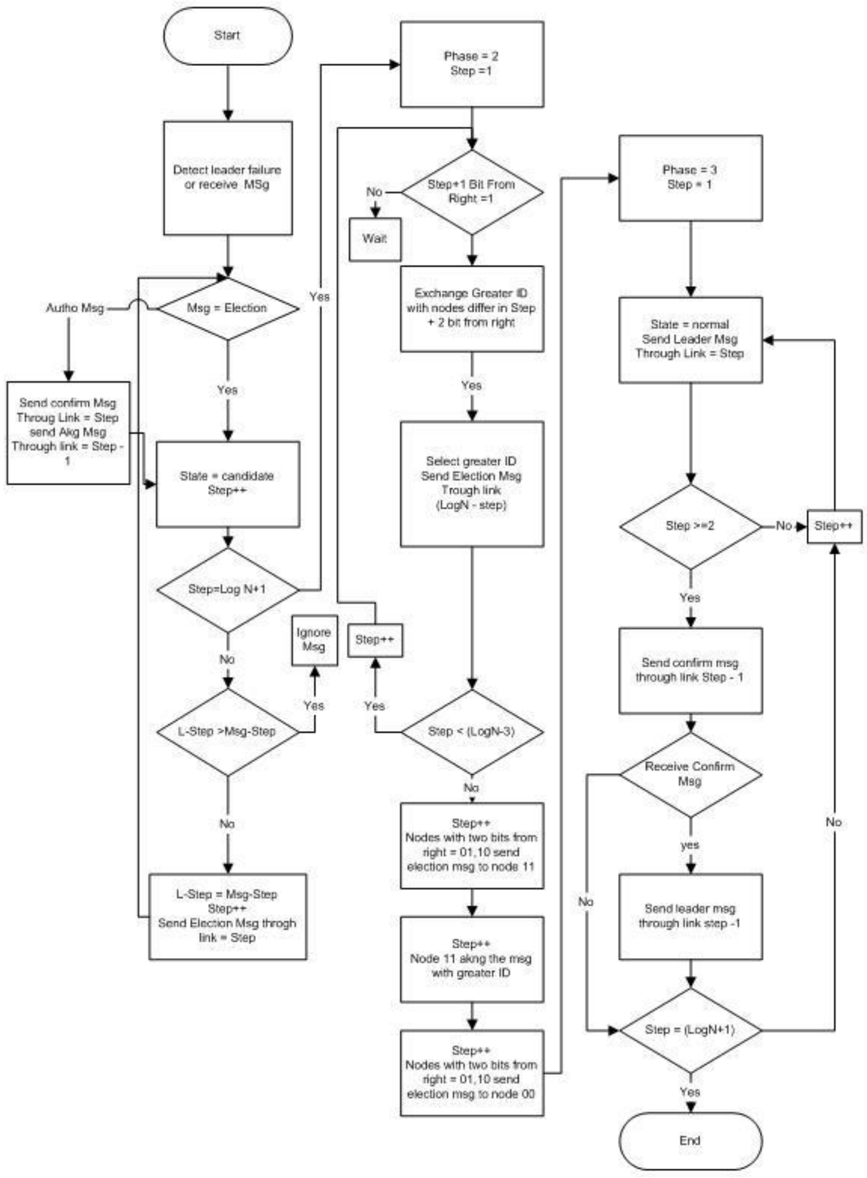
10	1832	"001110204601360"	808
10	1833	"001110203401889"	809
10	1834	"001110201701270"	810
10	1835	"001110204001803"	811
10	1836	"001110201801820"	812
10	1837	"001110204301621"	813
10	1838	"001110201701270"	814
10	1839	"001110202701731"	815
10	1840	"001110204601360"	816
10	1841	"001110203401889"	817
10	1842	"001110201701270"	818
10	1843	"001110204001803"	819
10	1844	"001110201801820"	820
10	1845	"001110204301621"	821
10	1846	"001110201701270"	822
10	1847	"001110202701731"	823
10	1848	"001110204601360"	824
10	1849	"001110203401889"	825
10	1850	"001110201701270"	826
10	1851	"001110204001803"	827
10	1852	"001110201801820"	828
10	1853	"001110204301621"	829
10	1854	"001110201701270"	830
10	1855	"001110202701731"	831
10	1856	"001110204601360"	832
10	1857	"001110203401889"	833
10	1858	"001110201701270"	834
10	1859	"001110204001803"	835
10	1860	"001110203001604"	836
10	1861	"001110204301621"	837
10	1862	"001110201701270"	838
10	1863	"001110202701731"	839
10	1864	"001110204601360"	840
10	1865	"001110203401889"	841
10	1866	"001110201701270"	842
10	1867	"001110204001803"	843
10	1868	"001110201801820"	844
10	1869	"001110204301621"	845
10	1870	"001110201701270"	846
10	1871	"001110202701731"	847
10	1872	"001110204601360"	848
10	1873	"001110203401889"	849
10	1874	"001110201701270"	850
10	1875	"001110204001803"	851
10	1876	"001110203502004"	852
10	1877	"001110204301621"	853
10	1878	"001110201701270"	854
10	1879	"001110202701731"	855
10	1880	"001110204601360"	856
10	1881	"001110203401889"	857
10	1882	"001110201701270"	858
10	1883	"001110204001803"	859
10	1884	"001110201801820"	860
10	1885	"001110204301621"	861
10	1886	"001110201701270"	862
10	1887	"001110202701731"	863
10	1888	"001110204601360"	864

10	1889	"001110203401889"	865
10	1890	"001110201701270"	866
10	1891	"001110204001803"	867
10	1892	"001110203001604"	868
10	1893	"001110204301621"	869
10	1894	"001110201701270"	870
10	1895	"001110202701731"	871
10	1896	"001110204601360"	872
10	1897	"001110203401889"	873
10	1898	"001110201701270"	874
10	1899	"001110204001803"	875
10	1900	"001110201801820"	876
10	1901	"001110204301621"	877
10	1902	"001110201701270"	878
10	1903	"001110202701731"	879
10	1904	"001110204601360"	880
10	1905	"001110203401889"	881
10	1906	"001110201701270"	882
10	1907	"001110204001803"	883
10	1908	"001110201801820"	884
10	1909	"001110204301621"	885
10	1910	"001110201701270"	886
10	1911	"001110202701731"	887
10	1912	"001110204601360"	888
10	1913	"001110203401889"	889
10	1914	"001110201701270"	890
10	1915	"001110204001803"	891
10	1916	"001110201801820"	892
10	1917	"001110204301621"	893
10	1918	"001110201701270"	894
10	1919	"001110202701731"	895
10	1920	"001110204601360"	896
10	1921	"001110203401889"	897
10	1922	"001110201701270"	898
10	1923	"001110204001803"	899
10	1924	"001110203001604"	900
10	1925	"001110204301621"	901
10	1926	"001110201701270"	902
10	1927	"001110202701731"	903
10	1928	"001110204601360"	904
10	1929	"001110203401889"	905
10	1930	"001110201701270"	906
10	1931	"001110204001803"	907
10	1932	"001110201801820"	908
10	1933	"001110204301621"	909
10	1934	"001110201701270"	910
10	1935	"001110202701731"	911
10	1936	"001110204601360"	912
10	1937	"001110203401889"	913
10	1938	"001110201701270"	914
10	1939	"001110204001803"	915
10	1940	"001110203502004"	916
10	1941	"001110204301621"	917
10	1942	"001110201701270"	918
10	1943	"001110202701731"	919
10	1944	"001110204601360"	920
10	1945	"001110203401889"	921

10	1946	"001110201701270"	922
10	1947	"001110204001803"	923
10	1948	"001110201801820"	924
10	1949	"001110204301621"	925
10	1950	"001110201701270"	926
10	1951	"001110202701731"	927
10	1952	"001110204601360"	928
10	1953	"001110203401889"	929
10	1954	"001110201701270"	930
10	1955	"001110204001803"	931
10	1956	"001110203001604"	932
10	1957	"001110204301621"	933
10	1958	"001110201701270"	934
10	1959	"001110202701731"	935
10	1960	"001110204601360"	936
10	1961	"001110203401889"	937
10	1962	"001110201701270"	938
10	1963	"001110204001803"	939
10	1964	"001110201801820"	940
10	1965	"001110204301621"	941
10	1966	"001110201701270"	942
10	1967	"001110202701731"	943
10	1968	"001110204601360"	944
10	1969	"001110203401889"	945
10	1970	"001110201701270"	946
10	1971	"001110204001803"	947
10	1972	"001110201801820"	948
10	1973	"001110204301621"	949
10	1974	"001110201701270"	950
10	1975	"001110202701731"	951
10	1976	"001110204601360"	952
10	1977	"001110203401889"	953
10	1978	"001110201701270"	954
10	1979	"001110204001803"	955
10	1980	"001110201801820"	956
10	1981	"001110204301621"	957
10	1982	"001110201701270"	958
10	1983	"001110202701731"	959
10	1984	"001110204601360"	960
10	1985	"001110203401889"	961
10	1986	"001110201701270"	962
10	1987	"001110204001803"	963
10	1988	"001110203001604"	964
10	1989	"001110204301621"	965
10	1990	"001110201701270"	966
10	1991	"001110202701731"	967
10	1992	"001110204601360"	968
10	1993	"001110203401889"	969
10	1994	"001110201701270"	970
10	1995	"001110204001803"	971
10	1996	"001110201801820"	972


Start
Detect leader failure or receive election message
State = candidate Step++
Step=Log N+1
Yes
No
L-Step >Msg-Step
Yes
Ignore Msg
No
L-Step = Msg-Step Step++ Send Election Msg throgh link = Step
Phase = 2 Step =1
Step+1 Bit From Right =1
No
Wait
Yes
Select greater ID Send Election Msg Trough link (LogN - step)
Step++
Step = (LogN+1)
Phase = 3 Step = 1
State = normal Send Leader Msg Through Link = Step
Step = (LogN+1)
No
Step++
Yes
End

First algorithm flowchart


Start
Detect leader failure or receive MSg
Autho Msg
Msg = Election
Send confirm Msg Throug Link = Step send Akg Msg Through link = Step - 1
Yes
State = candidate Step++
Step=Log N+1
Yes
No
L-Step >Msg-Step
Yes
Ignore Msg
No
L-Step = Msg-Step Step++ Send Election Msg throgh link = Step
Phase = 2 Step =1
Step+1 Bit From Right =1
No
Wait
Exchange Greater ID with nodes differ in Step + 2 bit from right
Yes
Select greater ID Send Election Msg Trough link (LogN - step)
Step++
Yes
Step < (LogN-3)
No
Step++ Nodes with two bits from right = 01,10 send election msg to node 11
Step++ Node 11 akng the msg with greater ID
Step++ Nodes with two bits from right = 01,10 send election msg to node 00
Phase = 3 Step = 1
State = normal Send Leader Msg Through Link = Step
Step >=2
No
Step++
Yes
Send confirm msg through link Step - 1
Receive Confirm Msg
yes
No
Send leader msg through link step -1
No
Step = (LogN+1)
Yes
End

Second algorithm flowchart