

LEADER ELECTION ALGORITHM IN HYPERCUBES WITH THE PRESENCE OF ONE LINK FAILURE

Refai Mohd Naim M. Ajlouni
Amman Arab University for Graduate Studies
Amman – Jordan
naim@aau.edu.jo

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 know who the new leader is.

This study presents a distributed solution to this problem in Hypercube networks considering the link failure. A distributed leader election algorithm is proposed and its performance is evaluated. Contention and synchronization issues are also considered. In a network of N processors connected by a Hypercube network the proposed algorithm uses $O(N)$ messages to elect a new leader in $O(\log N)$ time steps.

Index Terms Hypercube Network, Leader Election, , Leader Failure, Link failure, message complexity, time complexity.

1. Introduction

One of the most fundamental problems in distributed systems is the leader failure. This problem can be solved by Leader election Algorithms (LEAs). These algorithms move the system from an initial state, where all the nodes are in the same computation state, to a new state in which one of these nodes is a leader or coordinator, and the rest are not.

Distributed systems are widely used to increase the speed of solving computing problems. These systems use many computers which cooperate with each other to execute the program. The control of these systems may be centralized "controlled" by one process called leader or distributed among the processors. In centralized control, Fault tolerant from leader crash is a very important issue in these systems. The (LEAs) solve this problem by substituting the failed leader by a new leader.

The election process is a program distributed over all nodes, it starts when one or more processors discover that the leader has failed, it terminates when the remaining processors know who the new leader is.

The leader election algorithms (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. The LEAs are also used in a token ring. When the node that has the token fails, the system should select a new node to have the token.

In distributed systems, there are many network topologies like hypercube, meshes, ring, bus,...etc. These topologies may be either the hardware processors, or the software processes embeded over the other hardware topology. This study will focus on the hypercube processes topology where one process works as a leader and the others as none. It will propose a new election algorithm to solve leader failure automatically in spite of the presence of one link failure. The proposed algorithm will be executed without user intervention.

Election algorithms start when the leader failure is detected by one process at simple case or by all processes at worst case. It terminates when all processes know the new elected leader.

Our paper is organized as follows. Section 2 presents literatures review. Section 3 describes the hypercube model structure and properties. Section 4 presents our algorithm. Section 5 gives a mathematical proof for the time steps and message complexity. Section 6

presents ebcd code for the election algorithm. Section 7 will conclude the results and suggest future works besides the list of important references.

2 Literature Review:

Singh G., **Leader Election in the Presence of Link Failures**, 1996: This literature 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. So up to $N^2/4 - N/2$ links overall the system may fail. Nodes represent the processors and edges representing bi-directional communication channels between the processors. In the problem of leader election, 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 and each iteration composed of phases. When the iterations reach $(\log N + 2)$ the node is the leader. The message complexity of the protocol is $O(N^2)$ (Singh, 1996).

Navneet M., Jennifer L., Welch, Nitin V., **Leader Election Algorithms for Mobile Ad Hoc Networks**, 2001: This paper 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. The algorithms require nodes to communicate with only their current neighbors, making it well suited to the ad hoc environment.

Sudarshan V., Declene 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 elections 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.

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

- 1- P sends an election message to every process with the higher number.
- 2- If no one responds, P wins the election and become 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.

Fredrickson and Lynch, **Election of a Leader in Asynchronous Ring**, 1987. This study assumes the processes are physically or logically ordered, so that each process knows who its successor is. The election message is built when any process notices that the leader is not functioning. The process sends messages containing its number to the successor. If the 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 process that started it all. That process recognizes this event when it receives an incoming message containing its own process number. At that point, the process sends a leader message to inform all the processes about the new leader (Fredrickson,87).

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 process detects the leader failure. In more complicated cases when the failure is detected by subset of the processes, the time complexity is linear, and the algorithm terminates in $O(N)$ time steps.

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 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 $F \leq (N-1)/2$. This algorithm uses at most $O(N^2 + NF^2)$ messages to elect a unique leader (Amara 94).

Flocchini and Mans, **Optimal Election in Labeled Hypercube, 1996**: In this literature 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 $\theta(\log^3 N)$ time steps using $\Omega(N)$ messages (Flocchini and Mans, 1996).

Antonoiu and Srimani, **A Self-Stabilizing Leader Election Algorithm for Tree Graphs, 1995**: This literature 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 of 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 literature shows that it may not be possible to design a self stabilizing protocol that can elect a leaf node to be the leader (Antonoiu and Srimani, 96). The literature didn't present analyses for the number of messages and time steps, it presented just the steps and description of the algorithm.

3 Model description:

A d -dimensional hypercube network is represented by N -nodes labeled by d binary bits from $(0$ to $2^d)$ (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 equals the Hamming distance between their canonical labels. The diameter and radius of the hypercube are 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 hypercubes and extend all labels in the first hypercube with a 0 and all labels in the second hypercube with 1. For each process in the first hypercube 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 and obtains its label from the nodes labels. The order of the different bit in the two nodes is the link label. This order from 1 to $\log N$. figure 1 shows the 3-dimension hypercube with the labels for its nodes.

This study assumed the following Assumptions:

- ⇒ One intermittent link failure is recoverable.
- ⇒ Routers working all the time even at the fault process.
- ⇒ Each process has the following variables:
ID: A unique value for the election process.
POSITION: The label indicates its position.
- ⇒ All the communication links are bidirectional.

4 Leader Election Algorithms in Hypercube with the Presence of Link Failure:

Our algorithm consists of three phases. Each phase has many steps and messages. Phase one is initiated when one or more nodes detect the leader failure, it starts the election process. This phase reduces the count of participated processes in the election process to $N/2$ processes knowing about the election. The second phase uses the reduction all-to-one communication operation "with our addition steps to tolerate link failure" to have the result in one process that has the address $(X_1 0_2 0_3 \dots 0_d)$ (X means 1 or 0). Finally in the third phase $(X_1 0_2 0_3 \dots 0_d)$ process broadcasts the leader message to all processes in

the hypercube using one-to-all broadcast communication operation, “with our addition steps to tolerate 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 detects the failure to change its state to candidate and initiate the election:

Step1: Send an election message to the node differs in the right most bit through link 1. Election message composed of (phase, step, winner ID, winner position). The election message changes 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 differ in the second right most bit through link 2.

Step r (r >2) : The senders from the previous step send an election message to the nodes which differ in the r-1 left bit through the link r-1. The receiver nodes send an authentication message (election message) through link r-2 to ensure that the election message reached the node or not due to the link failure. After sending of the authentication message the node waits for acknowledgement or time out to send election in step r.

The nodes that receive the authentication message may receive the election message in advance, so the authentication message is ignored after receiving the greater ID. In the second case the authentication message is received by a node in normal state. To tolerate the probability of link failure the node sends the election message through link (r-1). In both cases the nodes that receive the authentication message send an acknowledgement to the sender.

The nodes that received the authentication message send the election messages through r + 1 link to continue the algorithm. Surely any node sending any election or authentication message must use the greatest ID that it knows.

Step Log N: N/2 processes (the senders and receivers from the Log N –1 step) send an election messages to the nodes which differ in the Log N left bit through log N link. End the phase 1.

After this phase leader ID is included in 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 process receives the election message; it compares its own ID with the receiving ID then completes the next step with the greater ID. Phase one ended after Log N steps, reducing the participant nodes to N/2. The leader ID and its position for the whole hypercube becomes inside (Log N –1) dimensional hypercube. If the two sub hypercubes reach the step log N simultaneously ‘in the worst case’ the sub cube with the most left bit = 0 completes the algorithm. And the other sub terminates it.

Phase Two: The second phase uses the reduction all-to-one communication operation. We added some alterations to tolerate the link failure. The operation applied to N/2 node hypercube that resulted from phase 1, have the result in one process that have the address $X_10_2\dots0_d$. As follows:

Step1 To Step Log N –3 : nodes that reach phase 2 and with the (step + 1) and (step + 2) left most bit =1 ($X_11X_4\dots X_d$) exchange the greater id with the nodes with the same properties except the (step + 2) left most bit = 0 through the link (log N – step –1). The participants in the exchange need acknowledgement or wait until time out to progress. After the exchange these nodes send election message to nodes with the second left bit = 0 ($X_10X_3\dots X_d$) through link Log N – step. These steps conclude the result inside a two dimensional hypercube in spite of the probability of the link failure.

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

Step Log N-1: After comparison and get the greater ID node (11) send the winner to nodes (10,01).

Step Log N: In last step nodes (10,01) send election message to the node ($X_10_20_3\dots0_d$).

After the end of phase 2 the last node ($X_10_20_3\dots0_d$) is the only node knows about the leader ID.

Phase3: In this phase the node $(X_10_20_3\dots0_d)$ 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 $(X_10_20_3\dots0_d)$ send leader message through link (step) to node $(X_10_20_3\dots1_d)$.

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

Step 3 to Log N: Each node receives the leader message sent to it through the link step. To consider the link failure for this phase each node receives the leader message, then sends another message through link step - 2 to check if the message reached the other side. The node that receives the extra message may know about the broadcast so it ignores the extra message. If the node doesn't know about the message it will send the extra message through link step -1 .the last step isn't needed after the step Log N.

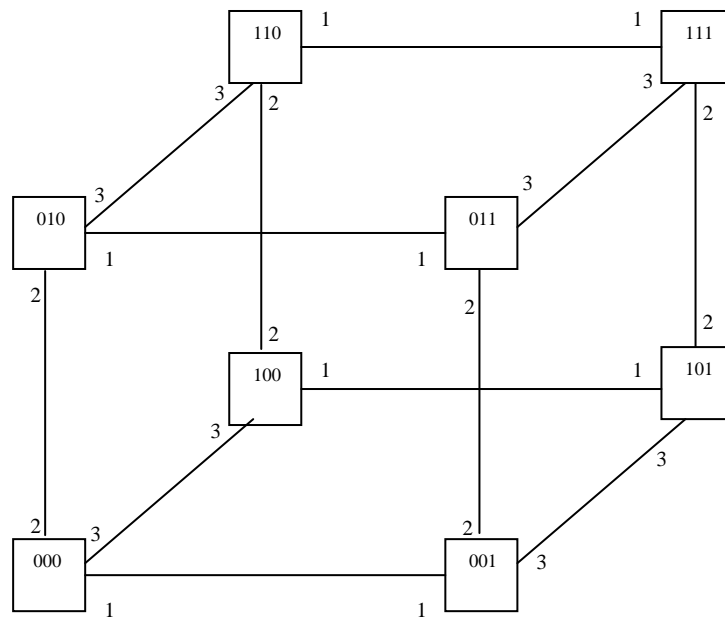


Fig-1 3 Dimension Hypercube

5 Analyses

We analyzed the algorithm by computing the number of messages and time steps. This analysis can be carried out in two ways. The first one is a simple case when the failure is detected by one process. The second way is when the leader failure is detected by a subset of nodes reaching in the worst case in all nodes at the same time.

5-1 Simple Case:

THEORM 5.1 The number of messages needed to complete the algorithm is at most $O(N)$ messages

Proof.

To find the number of messages, we record each phase and then calculate the overall total of the algorithm.

Phase One:

During this phase the algorithm use three types of messages: election messages where each node except the first node receives one message. The number of election messages is equal $(N-1)$.

$$1+2+4+\dots+N/2 = \sum_{i=0}^{\text{Log}N-1} 2^i = N-1 \quad (1)$$

The second type is the authentication message which starts from step 2, where each node send one authentication message until the end of this phase. The total number of this type is $N-2$ messages.

$$2 + 4 + \dots + N/2 = \sum_{i=1}^{\text{Log}N-1} 2^i = N-2 \quad (2)$$

The third type is the acknowledgement message. Each node which receives the authentication message sends an acknowledge

message except in the last step. The number of the acknowledgement messages is $N/2 - 2$.

$$2 + 4 + \dots + N/4 = \sum_{i=0}^{\log N - 2} 2^i = N/2 - 2 \quad (3)$$

If there is a link failure we need one message to inform the previous node about the election. So the total messages for phase 1 are equal to:

$$(N-1) + (N-2) + (N/2 - 2) + 1 = 5N/2 - 4 \quad (5)$$

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's nodes in phase 2 send two messages for each message received one message to avoid the probability of link failure and the second message to continue 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 as shown equation (6).

$$\begin{aligned} (N/4 + N/4 + N/8 + N/8 + \dots + 4 + 4) + 6 = \\ 2 \sum_{i=2}^{\log N - 2} 2^i + 6 = 2(N/2 - 5) + 6 = \\ N - 4 \text{ messages} \quad (6) \end{aligned}$$

Phase Three: Broadcast needs $N-1$ messages, since each node receives one leader message except the initiator.

$$1 + 2 + 4 + 8 + \dots + N/2 = \sum_{i=0}^{\log N - 1} 2^i = N - 1$$

to cover the probability of link failure we need

$$2 + 4 + 8 + \dots + N/2 = \sum_{i=1}^{\log N - 1} 2^i = N - 2$$

if there is a link failure we need another message so the number of messages for phase 3 is equal:

$$(N-1) + (N-2) + 1 = 2N-1 \text{ messages} \quad (7)$$

Total : From equations 5,6,7 the total number of messages over all the algorithm is :

$$\begin{aligned} (5N/2 - 4) + (N-4) + 2(N-1) = \\ (11N/2) - 10 = O(N) \text{ messages} \quad (8) \end{aligned}$$

THEOREM 5.2 The election algorithm in the hypercube return 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 the overall algorithm.

Phase One: Reducing the processes containing the leader ID to the half nodes of the model needs $\log N$ time steps.

Step1: node detecting a failure will sends an election message to the node that differs in the first most right bit and will needs one time step to do so; in this case there is no need for the authentication and the acknowledgement.

Step2: to $\log N$: each step needs 3 time steps, the first for election message, the second for the authentication message and the third for the acknowledgement.

The last step doesn't need the acknowledgment. The total for phase one is equal:

$$1 + 3(\log N - 1) - 1 = 3 \log N - 3 \text{ Steps} \quad (9)$$

Phase Two: this phase continues the election process with $d-1$ dimensional hypercube with all its nodes know about the election and receive the result from the first phase. The first stage of this phase needs 3 time steps: two steps for the exchange and one for reduction step. phase 2 needs $3(\log N - 3)$ steps, and 3 steps for the two dimensional hypercube

$$3(\log N - 3) + 3 = 3 \log N - 6 \text{ Steps} \quad (10)$$

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:

$$\log N + 2 \quad (11)$$

The total steps over all the algorithm from 9,10,11

$$\begin{aligned} \text{Total : } 3 \log N - 3 + 3 \log N - 6 + \log N + 2 \\ = 7 \log N - 7 \text{ Steps} \\ = O(\log N) \text{ Steps} \end{aligned}$$

5-2 Worst Case:

THEOREM 5.3 *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, we record the number of messages for each phase, and then calculated the total over the 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 and the time of detection. But in any case it doesn't increase N messages for each in each step. The total isn't increase $3N \log N$ messages.

Phase Two and Phase Three are the same as in the simple case.

Total : $3N(\log N) + (N - 4) + (2N - 1) = O(N \log N)$ messages

THEOREM 5.4 *In any case the election algorithm in the hypercube returns stable with a new leader by $O(\log N)$ steps*

Proof.

The steps are the same as in simple case $O(\log N)$ steps.

6 Abstract Algorithm

This section presents ebcd code for the election algorithm. Before writing the code, each node has the following variables:

- a. Local ID: the node ID is used by each node to participate in the election process.
 - b. Local Pos: The node Position.
 - c. Curr_Step: Last step in the election process.
- i. The algorithm uses two types of messages
 1. **Election:** contains Phase (1 OR 2), step, ID (the winner ID) ,Pos (The winner position).
 2. **Leader:** contains the new Leader (ID and Position).

3. **Authentication:** contains Phase (1 OR 2), step, ID (the winner ID) ,Pos (The winner position).
4. **Acknowledgement:** contains (ok) and the max ID known to it.
5. **Check:** node receives the authentication message and sends check message to avoid the effect of link failure.

ii. The nodes are in one of two states:

1. **Normal:** when the node doesn't know of any failure and the network is normal.
2. **Candidate:** when the node acknowledges the failure, and participates in the election process.

1. Case state = normal

Upon detect 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
  }
}
```

Send authentication

message on link step-1

//authentication message
contain same information 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 recover link failure
  }
  If step = Log N
```

```

    {
        Compare ID and choose the
        winner
        Phase = 2
        Step = 1
        State = candidate
    Send exchange message to the node
    differ in the third most left in label
        And wait until receive 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...IX))
        {
            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

2. 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 receive authentication message on link r

```

{

```

```

        Compare ID and send
        acknowledgment message on link r.
        ++Step
        Send election message on link r+
1
    }

```

```

Upon receive Acknowledgment message or
time out
    {
        Compare ID
        ++ Step
        Send election on link step
    }

```

```

Up on receive 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 receive Check message on link r
    Send leader message on link r+1

```

END OF THE ALGORITHM

7 Conclusions and Future Works

This paper presents a distributed solution to the leader failure problem in Hypercube networks with the presence of a link failure. A distributed leader election algorithm is proposed and its performance is evaluated. Contention and synchronization issues are also considered when designing the algorithm. In a network of N processors connected by a Hypercube network the proposed algorithm uses $O(N)$ messages to elect a new leader in $O(\log N)$ time steps. For **Future Work**, the work presented can be improved by carrying out the following:

- ≡ Design a simulation program to show proof of the algorithm.
- ≡ Design an algorithm to solve the leader failure in meshes with the presence of link failure.

8 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 1994, pp. 778-788.
- 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, 1996, pp. 227-232.
- Deitel and Deitel, Nieto, T.R., (1999). **Visual Basic 6**, Prentice Hall.
- Flocchini, P. and Mans, B. (1996). **Optimal Elections in Labeled Hypercube**, Journal of Parallel and Distributed Computing 33, Article No. 0026, pp. 76-83.
- 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.
- Kumar V., Grama A., Gupta A. and Karypis G. (2003). **Introduction to Parallel Computing**, The Benjamin/Cumminy Publishing Company, Inc, Redwood City, California.
- Molina G, H., (1982). **Elections in A Distributed Computing systems**, IEEE Transactions on Computers, Vol. 31 Jan 1982, pp. 48-59.
- Refai M. and Ababneh E., (2002) **leader election algorithm in 3D torus networks**, master theses, not published, al-albayet university – Jordan.
- Singh G., (1996). **Leader Election in the Presence of Link Failures**, IEEE Transactions on Parallel and Distributed Systems, VOL 7, No 3, March.
- Tanenbaum, A., (1995). **Distributed Operating Systems**, Prentice-Hall International, Inc, New Jersey.