Leader Election Algorithm in 2D Torus Networks with the Presence of One Link Failure

Mohammed Refai[1], Ahmad Sharieh[2], and Fahad Alshammari[3]
[1]Sciences and Information Technology Collage, Zarqa Private University, Jordan
[2]King Abdullah II School for Information Technology, University of Jordan, Jordan
[3]Information Technology and Computer Science College, University of Malaya, Malaysia

Abstract: *Leader election algorithms solve the instability problem in the network which is caused by leader failure .In this paper, we propose a new leader election algorithm in two dimensional torus networks. The algorithm aims to elect one node to be a new leader. The new leader is identified by some characteristics not in the other nodes in the network. When the process is terminated, the network is returned to a stable state with one node as leader where other nodes are aware of this leader. The new algorithm solves this problem despite the existence of one link failure. In a network of N nodes connected by two dimensional torus network, the new algorithm uses O(N) messages to elect a new leader in O($\sqrt{N}$) time steps. These results are valid for both cases: simple case (when the leader failure is detected by one node) and in the worst case (when the failure is discovered by up to N-1 nodes).*

Keywords: *Concurrency, leader election, link failure, message complexity, 2D torus networks.*

Received May 13, 2008; accepted November 25, 2008

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 into a new state where only one node is distinguished computationally (called leader) and all other nodes are aware of this leader [1 ,4, 8].

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 tasks. 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 automatically elected to keep the network working. The LEAs solves this problem by substituting the failed leader by a new deserved leader [5, 30, 31].

Election process is a program distributed over all nodes. It starts when one or more nodes discover that the leader has failed. It terminates when the remaining nodes know who the new leader is. The LEAs are widely used in centralized systems to solve single point failure problem [4]. For example, in client-server, the LEAs are used when the server fails and the system needs to transfer the leadership to another station. The LEAs are also used in token ring. When the node that has the token fails, the system should select a new node to have the token [1].

In distributed systems, there are many network topologies like hypercube, meshes, torus, ring, bus, …, *etc.,* [30]. These topologies may be either hardware processors or software processes embedded over other hardware topology [10, 14]. This study will focus on the 2D torus topology where one node works as a leader. This paper proposes a new election algorithm to solve leader failure in 2D torus network automatically. Also it guarantees to solve the leader failure problem despite of the existence of one link failure.

The election algorithms start when the leader failure is detected by one node in a simple case or subset of nodes reached to (N-1) at the worst case. It terminates when the new leader is elected and all other nodes become aware of the new leader.

Section 2 presents related work. Section 3 describes the 2D torus model structure and properties. Section 4 presents the new algorithm. Mathematical proof for the time steps and message complexity is presented in section 5. Section 6 will conclude the results and suggest future works.

2. Related Work

Leader election algorithm was studied by many researchers [1, 2, 4, 6, 7, 10, 11, 12, 13, 14, 17, 18, 20, 22, 23, 25, 26, 27, 28, 34]. In these studies, the researchers presented different methods to deal with the leader election algorithms. In distributed systems, a major problem is the leader failure and the relevant

leader election algorithm. The election algorithms were varied based on the following:

- The nature of the algorithms (dynamic vs. static) [7, 12, 22, 23].
- Node Identity (ID) (unique identity vs. anonymous ID) (distinguished vs. not distinguished) [34].
- Topology types such as: ring, tree, complete graph, meshes, torus, and Hypercube [1, 9, 22, 23].
- Communication mechanism used (synchronous vs. asynchronous) [22, 23].
- Transmission media (wired vs. wireless or radio) [13].
- Some of the previous work dealt with the link failure [1, 26].

The leader election solution was first thought of at the end of the seventies, it was started by the ring and complete networks [1, 17, 26, 18, 11]. In the nineties meshes, hypercube and tree were studied. To date, these topologies and wireless networks are still being studied [13, 17]. In [26], Singh proposed a protocol for leader election which is tolerant to intermittent link failure in the complete graph network. In [12], Gerard proposed an election algorithm for oriented hypercube, where each edge is assumed to be labeled with its dimension in the hypercube. In [9], the election problem in hypercube networks was studied, by using two models with sense of direction. In [3], the problem of one link failure besides the leader failure in the hypercube was solved. In [1], the problem of, fault tolerant and leader election in asynchronous complete (fully connected) distributed networks was considered.

Antonoiu and Srimani [4] proposed a self-stabilizing algorithm for leader election in a tree graph. In [19], Navneet and others presented two new leader election algorithms for mobile ad-hoc networks. In [29], they proposed two algorithms assume asynchronous distributed system in which the various rounds of election proceed in a lock-step fashion.

Most of the previous researchers employed theoretical proof to verify their algorithms. They used the big *O* notation to obtain the complexity [16] of the number of messages and time steps which represent the domain factors of the algorithm complexity [9, 11]. Other researchers used simulation to validate their algorithms [25].

3. Model Description

In 2D torus network, interconnection topology is a torus graph with $N = X * Y$ nodes (X is the number of nodes in the X dimension, and Y is the number of nodes in the Y dimension of the torus network). This section explains; the model description; properties, and design assumptions for this research [32, 33].

The 2D torus network is similar to 2D mesh, except in the connection between the first and the last nodes in each dimension. These connections make all nodes connected with four neighbors (left, right, up, down) in order to present more flexible topology [32, 33]. Figure 1 shows a two dimensional torus network with seven columns and four rows (7 X 4).

3.1. Model Properties

The target architectures for the proposed algorithm are distributed-memory, and two dimensional torus multi-computers. For research analysis, we use the model with the following properties: the multi-computers consist of N nodes, which can be labeled 0, 1, 2, … , N-1. The nodes, physically, form an $X * Y$, (rows) * (columns), two-dimensional torus.

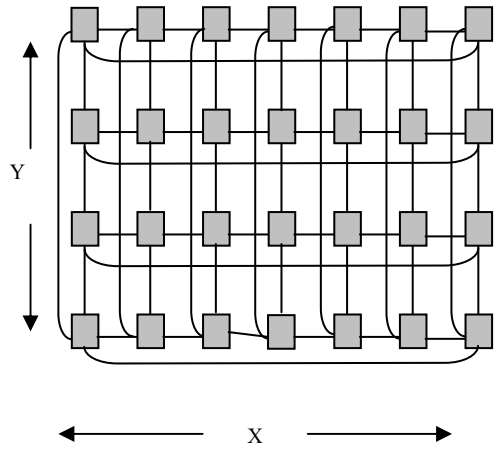

Figure 1. 2D (7x4) Torus network.

Communication is with only one node at a time. Multi cast is not implemented in hardware. A node can send or receive simultaneously to, and from, the same- or different nodes. The network uses *XY*-routing: a message is routed within a row to the column that contains the destination node and subsequently routed within the column. Leader failure can occur any time. This failure may be discovered by one node in a simple case, or concurrently by more than one node-reached in a worst case to N-1 nodes. The proposed algorithm solves leader failure even when there is a link failure. Each node has a distinguished ID used in the election algorithm. Each node is connected by four links as in Figure 2, which shows node links.

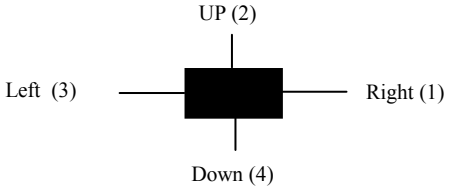

Figure 2. Node links and codes ks.

A torus network has advantages that make it one of the preferable topologies. Torus is an attractive

structure for parallel processing due to its symmetry and regularity [32]. Diameter of the torus is $X * Y$. A node is labeled as (X,Y)-and uses X-Y routing techniques. The number of links is $X*Y$. In fact, it has been shown to be a very versatile and robust architecture which is capable of executing several efficient parallel algorithms. This topology is a suitable architecture for designing tightly coupled systems in both parallel and distributed systems.

3.2. Assumptions

This research assumes the following:

- Routers should work all the time even with fault node-because the fault is in leader properties.
- All communication links are bidirectional.
- Leader node could fail due to different reasons which will lead to loss of the leadership property. Other nodes can detect this failure when the time out exceeds without acknowledgement. Nodes which detect this failure start the election algorithm.
- To solve leader failure problem, each node calculates a weight that defines its relative importance. Then, compares it with the weight of other nodes that it has received and propagate the maximum weight. This weight is represented by a Identification Distinguish (ID) for each node.
- Each node has a distinguished ID. The election algorithm depends on this ID.
- When the leader node crashes, its ID degrades to 0. So, it can not win the election.
- One intermittent link failure is recoverable.
- Leader failure may be detected by a subset of nodes (concurrent failure). This case becomes complicated when the failure is detected by N-1 nodes (worst case).

Each node has the following variables:

- ID: a unique value for the election process.
- Position: the label indicates its position.
- Leader ID, leader position.
- Phase and step.
- State: leader or normal or candidate.

4. Proposed Algorithm

Before describing the proposed algorithm, the definition of node state, phase and messages help to understand the algorithm.

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

- Normal: network is normal and no leader failure is detected by this node.
- Candidate: there is a failure and the election process is in progress inside this node.
- Leader: one node must have this state in a stable network.

The algorithm uses X and Y to represent the dimensions and x and y to represent node position.

Phases: the proposed algorithm is composed of four phases, as follows:

- *Phase One:* the node that detects leader failure informs the failure event to its row.
- *Phase Two:* the nodes in candidate states do election process within each column to obtain the result in the first row.
- *Phase Three:* nodes in the first row make the election within the first row to obtain the result in one node.
- *Phase Four:* the node that aware of the new leader in phase three, broadcast the new leader to all nodes.

Now, let us explain the events in each phase of the proposed algorithm.

- *Phase One*: the algorithm starts by node(s) that detects leader failure. This node sends failure messages through link 1 (right) and 3 (left) to inform its row about leader failure. A failure message informs the receivers about leader failure. To avoid the probability of link failure in this phase, the failure message is sent in two directions. Each node which receives this message performs the following: changes its state to candidate. Passes the failure message to the opposite direction through the opposite link (1 to 3, or 3 to 1), depending on the direction it receives the message. Starts phase two: selects its ID as greater ID, and sends election message through link 2 (Up). The election message is composed of (message type, Phase, Step, Greater ID, Position of the Greater ID, and position of the message initiator). Ignores the received message if the state is candidate.
- *Phase Two*: the candidate nodes send election messages through links 2. Any node which receives the election message compares its ID with the received ID in order to continue with the greater. This process ends when the initiator position receives the same message. After the column election, the result is sent to the first node of each column. This phase faces two problems: concurrent initialization and link failure. To deal with the first problem, any candidate node which receives the election message ignores the message. If there is no link failure, the result for the column is found in the node that completed the ring. This node sends the result to the node labeled (x, 0). To solve the second problem, the node that sends the election message waits for acknowledgment. If the node doesn't receive this message after time out, it detects that there is a link failure. The role of the node that detects the link failure is to send link-failure

message through link 3. The node which receives this message forwards it through link 2, and then left to pass the failure link. To complete the algorithm, the result is sent to the node labeled (x, 0). After all, one node is aware of the column leader so that the result for the leader is within the first row of the network.

- *Phase Three:* when the node which is labeled (0, 0) (the most left node in the first row) finishes phase 2, it starts phase 3 by sending election message through links 1 and waits for acknowledgment. Any node which receives a phase three election message from the left sends an acknowledgment message. Then it compares IDs and sends a phase three election message through link 1. If the node doesn't receive the acknowledgment message after time out, it detects there is link failure. To solve this problem in this phase, this node sends a link-failure message through link 2, then link 1 and down to pass the link failure. In this phase, any node which receives a phase three message before finishing phase two waits for the last message in phase two and then continues. Phase three is terminated when phase three election message is received by node (0, 0). This node starts phase four by broadcasting the result to all nodes.
- *Phase Four:* after phase three, one node aware of the new leader information. This node broadcasts the result as follows:

  a. Row broadcast: The nodes sends a leader message in two directions through links 1 and 3 in order to make all nodes in the first row aware of the new leader.
  b. Column broadcast: the receivers in a row broadcast, change their contents regarded the leader, and changes its state to normal. Then, they send the leader message through links 2 and 4. Any node which is aware of the new leader in phase four ignores any new message about election algorithm.

The initiators of the leader message, within the row in row broadcast and within the columns in column broadcast send the leader message in two directions. This is to recover the probability of one link failure.

4.1. Example

This example is applied on a 4X4 torus network. Assume that the link between nodes (0, 1) and (0, 2) is failed as shown in Figure 3(a). Node (1, 2) detects leader failure. So, it starts the algorithm by sending two leader failure messages to inform about the failure. Failure messages are sent through links 1 and 3 (thin arrows). Node (1, 2) also starts phase two by sending election message through link 2 (bold arrow) as shown in Figure 3(a). In the second step, the nodes that received the leader failure, passes this message to the reverse direction and starts phase two. The node that starts the algorithm in the first step waits for acknowledgement. Node (2, 2) continues phase two after comparing IDs and selecting the greater one. Then, it sends Ack message to node (1, 2), as shown in Figure 3(b). The nodes continue the algorithm as in Figure 3(c) and 3(d).

We can see the election messages as bold arrows and Ack messages as gray arrows. The election steps in phase two are continued until the messages reach to the election initiator in the column. Then, the column results are sent to the first line-as shown in dots arrows.

Phase three is started when node (0, 0) receives its column results by sending election message via link 1. Node (0, 1) passes this message and returns the Ack message. As shown in Figure 3(j), when node (0, 1) exceeds the waiting time, it detects the link failure. So, it uses the detour shown in Figure 3(j). The election process continues until the node (0, 0) receives the election message. So, it obtains the identification of the new leader. In Figure 3(k), node (0, 0) starts broadcasting the new leader information to the first row.

Each node receives the leader message; changes its state to normal, and broadcasts the leader information to its column as in Figure 3(l).

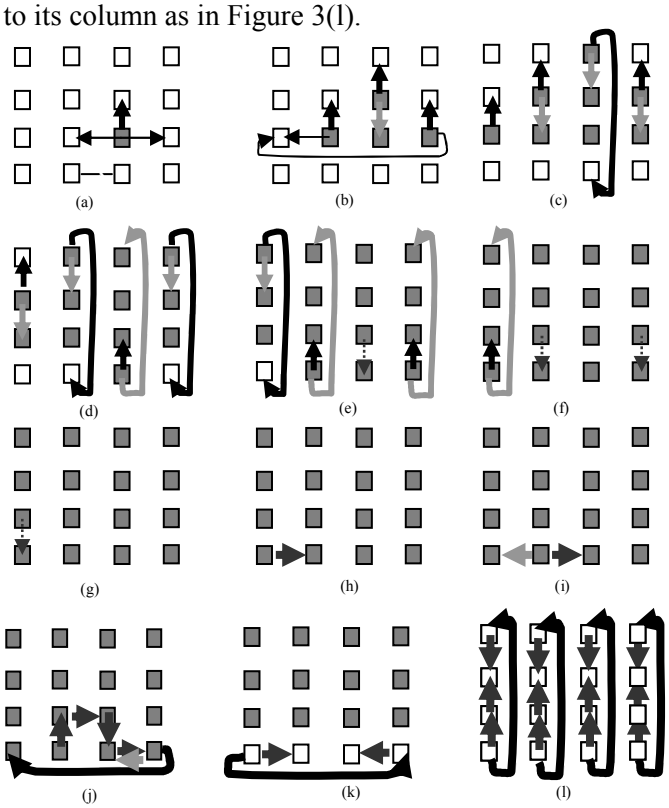

Figure 3. Steps for explaining the proposed algorithm when the link between nodes (0, 1) and (0, 2) fails and the leader failure is detected by node (1, 2).

4.2. Abstract Algorithm

This section presents the pseudo code for the algorithm. A number of assumptions and variables

have to be assigned. Each node has the following variables:

- Local ID: the node ID that participates in the election process.
- Local Pos: the node position.
- The algorithm uses five types of messages:
- Election: vomposed of: steps of phase one to four, ID (the winner ID); Pos (the winner position); and initiator.
- Leader: contains the new leader (ID and position).
- Link-failure: similar to the election message, except the type to pass the link failure.
- Column-result: is used in phase two in order to inform the column election result to the first row.
- Failure: is used to inform the row about the leader failure detection.

The nodes are in one of four states:

- Normal: when the node is unaware of any failure and the network is stable.
- 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 a stable network.
- Failure: when the leader lose the leader prosperities.

Figure 4 shows the pseudo code for the proposed algorithm.

5. Performance Evaluation

Performance evaluation is carried out by computing the number of messages and time steps. The analyses process is carried out for two cases. The first case is the simple case, when the failure is detected by one node. While the second case, is when the leader failure is detected by subset of nodes which can reach all nodes in the worst case.

5.1. Simple Case

5.1.1. Number of Messages

- *Phase One:* one node detects the leader failure. This node starts phase one by sending 2 leader-failure messages through links 1 and 3. Step two to step X/2 + 1: each step needs two messages (any node which receives the leader failure message sends this message through the inverse link). The last two nodes may use an extra two messages if the node sends a leader-failure message before receiving it from inverse links. Another way to find the number of messages needed for phase one is to think about receiving messages. Each node receives one message-except the last two nodes which receive two messages. So, the number of messages needed for phase one is (X +2) Messages.
- *Phase Two:* steps one to step Y, in each step, X election messages are needed through links labeled 2 and the same number is also needed for acknowledgment. So, the total number of messages is expressed as in equation 1. Each column needs one message in order to inform the first row of the column-result message. This needs X messages. The total number of messages for phase 2 is 2XY+X messages.

$$\sum_{i=0}^{y-1} 2x = 2(xy) \text{ Messages} \quad (1)$$

- *Phase Three:* when node (0,0) receives the column result message, it starts phase three: step one**,** node (0,0) sends election message through link 1 and waits for an acknowledgement. Step 2 to step X: Each node receives the election message and sends an acknowledgment message through link 3 and an election message through link 1. When node (0,0) receives the election message, it obtains the new leader information after sending one message for acknowledgement. In other words in phase three each node sends two messages (election and acknowledgement messages). So, the number of messages in phase three is as in equation 2.

$$\sum_{i=0}^{x-1} 2 \text{ 2x messages} \quad (2)$$

- *Phase Four:* node (0,0) starts a row broadcast by sending two messages through links 1 and 3 in step 1. In steps 2 to X/2, two messages are used in each step. In the last step, two extra messages are used if the last nodes send the message before receiving it. Row broadcast needs (X+ 2) messages.

In column broadcast, as in the row broadcast, Y+2 messages are needed for each column. Therefore, the total number of messages for X columns is X(Y+2). For phase four the number of messages needed is as in equation 3.

$$X+2 + XY+2X = XY+3X+2 \quad (3)$$

In order to cover the link failure in phase two or phase three the algorithm needs three messages. So, the total number of messages overall the algorithm is as in equation 4.

$$(X+2) + (2YX + X) + 2X + (XY+3X+ 2) +3 = 3XY+7X+7 \quad (4)$$

when X=Y, then XY =N. Thus, the total number of messages in terms of N is expressed in equation 5.

$$3N+7N^{1/2}+7 = O(N) \text{ Messages} \quad (5)$$

```
1.  Case state = Normal
  Upon detecting failure
      {
        State = Candidate
         Phase = 1
         Send inform message in links (1 and 3)
        Phase =2
         Send election message on link (2)
          Wait for Ack message
      }
  Upon receive inform message
      {
    Pass Inform message on Links (1 or 3 opposite direction.
         State = Candidate
          Phase = 2
         Step = 1
        Send election message on links (2)
       Wait for Ack message
       }
Upon receiving election message on link 2
 Send Inform message on Links (1 and 3).
        {
            State = Candidate
            Compare received ID with Local-ID and select
the Winner-ID
            Step = Step +1
           Send election message on link (2)
          Wait for Ack message
        }
2- Case state = Candidate
Upon Receiving Election message
 If (Phase = 2)
    {
        Compare received ID with Local-ID and select the
Winner-ID
        If Step = Y (Network width) then
         Send Column-Leader message to node (0, y) (node
same column and first-row)
        Else
            {
           Step = Step+1
            Send election message on link (2)
           Wait for Ack message
          Send ackt message on link (4)
           }
       If time out without receive Ack message then
        {
          Send link-failure message on link 3
        }
    }
Upon receiving link-failure message
        If the message was received from link 3, then
forward it on link 2
        If the message was received from link 2, then
forward it on link 1
        If the message was received from link 1, then
          {
            Compare received ID with Local-ID and select
the Winner-ID
                If Step = Y ( Network width) then
             Send Column-Leader message to node(0, y)
(node same column and first-row)
             Else {
                Step = Step+1
                Send election message on link (2)
                 }
Upon Receiving Column-Leader message
   {
If x=0  and  y =0
 Select greater ID
Phase = 3,
step =1
          Send election message on link 1

    Wait for Ack message
   }
If Phase = 3 then
 {
        Compare received ID with Local-ID and select the
Winner-ID
        If Step = X ( Network Length) then
         State = normal
          Send leader message on link 1 and 3
        Else
          {
            Step = Step+1
            Send election message on link (1)
           Wait for Ack message
          Send Ack message on link (3)
          }
       If time out without receive acknowledgment message
then
         {
           Send link-failure message on link 2
         }
  Upon Receiving link-failure message
     If the message was received from link 4 then forward  it
on link 1
      If the message was received from link 1 then forward it
on link 4
     If the message was received from link 2 then
          {
            Compare received ID with Local-ID and select
the            Winner-ID
           If Step = X ( Network length) then
            State =normal,  Send leader message on link 1
and 3
          Else
          {
            Step = Step+1
            Send election message on link (1)
          }
    }
Upon Receiving Leader message
   If the message from link 1 or 3  then
    {
      Pass the message to inverse  link (1 to 3 or 3 to 1)
     State = normal
    Send Leader message on links 2 and 4
    }
 If the message from link 2 or 4
{
  State = normal
 Pass the message to inverse  link (2 to 4 or 4 to 2)
}
```

Figure 4. Leader election algorithm in 2D torus with the presence of one link failure.

5.1.2. Time Steps

In any case, the number of Time Steps that needs to complete the leader election algorithm in 2D torus - with the presence of one link failure- is at most $O(\sqrt{N})$ steps.

In order to find the number of time steps, a complete computation is carried out for each phase, and then the total number of computation for the overall algorithm is calculated. We apply the computations at the simple case and then at the worst case as follows:

- *Phase One:* step 1, a node detects the failure and sends Leader-failure message to the right and left neighbors. steps 2 to X/2 + 1: nodes receive the leader-failure messages forward the message through the inverse link. At the same step, the node sends an acknowledgement message. The same procedure is repeated until step *X*/2+1. When the leader failure message is received by a candidate node, the time steps for phase one is (*X*/2+1) Steps.
- *Phase Two:* step 1, after phase 1, the candidate nodes start phase two by sending election messages through links labeled 2. Step 2 to step *Y*: nodes which receive the election messages make the comparison and pass election messages up with the greater ID. After *Y* -1 step the result of the column leader is found in one node. The node sends the column result to the first row. So the algorithm needs (*Y*+1) steps to complete phase 2 as in equation 6.

$$1+Y-1+1 = Y+1 \tag{6}$$

- *Phase Three:* node (0, 0) starts the election process in step one by sending the greater ID through link 1 (right). This process continues as follow, step 2 to step *X*: Any node which receives the election message, makes the comparison and sends the election message with greater ID to the right neighbor. Phase three is terminated when node (0, 0) receive the election message from link 3 (left). This process needs *X* steps. To tolerate the probability of the presence of one link failure in phases 2 and 3 the algorithm needs 3 steps as explained in the algorithm description.
- *Phase Four:* row broadcasting needs are (*X*/2+1) steps. Since node (0, 0) has the new leader information and it sends this information in two directions (left and right), the row broadcasting is terminated after *X*/2 steps and an extra step may occur if the last two nodes send the messages before receiving the message-or if *X* dimension is odd. So, the total is (*X*/2+1) steps.

The column broadcasting, using the same way column broadcasting needs (Y/2+1) steps. The total time steps overall the algorithm is as in equation 7.

$$X/2 + 1 + Y+1 + X + (X/2+1) + (Y/2 + 1) + 3$$
$$= 2X + 3/2Y + 7 \text{ time steps} \tag{7}$$

when X= Y = $\sqrt{N}$, the number of time steps can be expressed as in equation 8.

$$2\sqrt{N} + \frac{3}{2}\sqrt{N} + 7 = O(\sqrt{N}) \text{ steps} \tag{8}$$

5.2. Worst Case

5.2.1. Number of Messages

In the worst case, all nodes detect the leader failure simultaneously and start the algorithm.

- *Phase One:* all nodes detect the leader failure and start the algorithm. So each node sends two messages in links 1 and 3. Phase one is finished after one step because all of the nodes become a candidate. The number of messages is equal 2(*XY*) messages.
- *Phase Two:* after phase one, all nodes are in the candidate state, so all nodes start phase two.

Step One all nodes send election messages through link 2. All nodes also send acknowledgement messages. Therefore, step1 needs 2*XY* messages. Step 2 to *Y*: one node in each column will continue the election for the column. In each step, *X* nodes send election messages and *X* nodes send acknowledgement messages. The number of messages needed is as in equation 9.

$$\sum_{I=2}^{Y} 2X = 2X(Y-1) \tag{9}$$

After step *Y* in phase 2, one node in each column has the election result for this column. The results are sent to the first row. So, *X* messages are needed for this step. So, all over the number of messages is as in equation 10.

$$2XY + 2XY - 2X + X = 4XY - X \tag{10}$$

Phase three and Phase four are the same as in the simple case as follow:

- *Phase Three:* when node (0, 0) receives the column result message, it starts phase three. Step one: node (0, 0) sends election message through link 1 and waits for acknowledgement. Step 2 to step *X*: each node which receives the election message sends an acknowledgment message through link 3, and election message through link 1. When node (0, 0) receives the election message, it obtains the new leader information after sending one message for acknowledgement. In other words, in phase three, each node sends two messages (election and acknowledgement) messages. Thus, there will be 2*X* messages.
- *Phase Four:* node (0, 0) starts row broadcast by sending two messages through links 1 and 3 in step

1. In steps 2 to $X/2$, two messages is used in each step. In the last step, two extra messages are used if the last nodes send the message before receive. Row broadcast needs $X+2$ messages.

In column broadcast, $Y+2$ messages are needed for each column. Therefore the total for X columns is $X(Y+2)$. For phase four, the of messages is as in equation 11.

$$X+2 + XY+2X = XY+3X+2 \quad (11)$$

To cover the link failure in phase two-or phase three-the algorithm needs three messages so the total number of messages overall the algorithm is as in equation 12.

$$2XY+ 4XY – X + 2X + (XY+3X+ 2) +3 = 7XY + 4X + 5 \quad (12)$$

when $X=Y$, the $XY = N$. So, the total number of messages in terms of N is expressed by formula 13.

$$7N+4N^{1/2}+5 = O(N) \text{ messages} \quad (13)$$

6. Time Steps

When all nodes detect the leader failure simultaneously, the time steps will be as follows:

- *Phase one:* all nodes start the algorithm by sending leader-failure message. The different in this case is that, all nodes states become candidate after step one and start phase two. Therefore, phase one needs one time step to complete.
- *Phase Two:* in step one, all nodes start phase two. In step two, one node in each column continues the election, while all other nodes in the same column stop the process. So, the time steps in this phase are equal Y steps and need one step for column result message. Thus the total for this phase is $Y+1$ steps.
- *Phase Three and Phase Four:* are the same as in the simple case. The total time steps overall the algorithm in the worst case is as in equation 14.

$$1+ Y+1 +X +(X/2+1) + (Y/2 + 1) + 3 = 3/2X+3/2Y+7 \text{ Time steps} \quad (14)$$

when $X= Y = \sqrt{N}$, the number of time steps can be expressed as in equation 15.

$$\frac{3}{2}\sqrt{N}+\frac{3}{2}\sqrt{N}+7=O(\sqrt{N})\text{steps} \quad (15)$$

7. Conclusion and Future Work

In this work, a leader election algorithm in 2D torus network is proposed and analyzed. This algorithm consists of four phases. Each phase has many steps and messages. Phase one is initiated when one or more nodes detects leader failure, it initiates the election process. In phase one, the node(s) that detects the leader failure informs other nodes in the same row about leader failure. In phase two, nodes which aware of leader failure start the election process throughout the column. Phase two terminates when one node has the winner in the column. The results of all columns are transferred to the first row. In phase three, another election is applied on the first row to obtain the new leader information in the first node. Last phase, broadcasts one to all is applied to disseminate the new leader information to all nodes. All algorithm phases considered the probability of link failure. Algorithm performance was evaluated by calculating the number of messages and the steps throughout the whole algorithm. Several possible extensions to improve the work are proposed here. An algorithm can be designed to solve the leader failure in meshes networks with the presence of link failure. An algorithm can be designed to solve leader failure in hypercube when the ID is not distinguished. An election algorithm on topologies such as mesh- can be investigated in a wireless communication environment.

References

[1] Abu-Amara H. and Lokre J., “Election in Asynchronous Complete Networks with Intermittent Link Failures,” *Computer Journal of IEEE Transactions on Computers*, vol. 34, no. 7, pp. 778-788, 1994.

[2] Akbar B., Effatparvar M., and Effatparvar M., “Bully Election Algorithm Improvement with New Methods and Fault Tolerant Mechanism,” *in Proceedings of Computer Science and Engineering and Electrical and Electronics Engineering*, North Cyprus, pp. 501-506, 2006.

[3] Ajluni N. and Refai M., “Leader Election Algorithm in Hypercubes with the Presence of One Link Failure,” *in Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications*, USA, pp. 26-29, 2006.

[4] Antonoiu G. and Srimani K., “A Self-Stabilizing Leader Election Algorithm for Tree Graphs,” *Computer Journal of Parallel and Distributed Computing*, vol. 34, no. 59, pp. 227-232, 1996.

[5] Coulouris G., Dollimore J., and Kindberg T., *Distributed Systems Concept and Design*, Addison-Wesley, USA, 2005.

[6] Devillers M., Griffioen D., Romijn J., and Vaandrager F., “Verification of Leader Election Protocol Formal Method Applied to IEEE,” *Springer International Journal on Software Tools for Tecknology Transfer*, vol. 5, no. 4, pp. 123-125, 2004.

[7] Dolev S., Israeli A., and Moran S., “Uniform Dynamic Self-Stabilizing Leader Election,” *Computer Journal of IEEE Transaction on Parallel and Distributed Systems*, vol. 8, no. 4, pp. 424-440, 1997.

[8] Duato J., Yalamanchili S., and Ni L., *Interconnection Networks an Engineering Approach*, IEEE Computer Society, Addison Wesley, California, 1997.

[9] Flocchini P. and Mans B. "Optimal Elections in Labeled Hypercube," *Computer Journal of Parallel and Distributed Computing*, vol. 33, no. 26, pp. 76-83, 2005.

[10] Foster I., *Designing and Building Parallel Programs*, Addison-Wesley, USA, 1994.

[11] Fredrickson N. and Lynch N., "Election a Leader in Asynchronous Ring," *Computer Journal of the ACM*, vol. 34, no. 5, pp. 98-115, 1987.

[12] Gerard T., "Linear Election for Oriented Hypercube," *Technical Report TR-RUU-CS-93-39*, Utrecht University, 1993.

[13] Jean F., "Quasi Optimal Leader Election Algorithms in Radio Network with Log-Logarithmic Awake Time Slots," *in Proceedings of Institut National de Recherche en Informatique et Automatique*, France, pp. 97-100, 2005.

[14] Junguk L. and Geneva G., "A Distributed Election Protocol for Unreliable Networks," *Computer Journal of Parallel and Distributed Computing*, vol. 35, no. 22. pp. 35-42, 1996.

[15] Kumar V., Grama A., Gupta A., and Karypis G., *Introduction to Parallel Computing*, The Benjamin/Cumminy Publishing, California, 2003.

[16] Levitin A., *Introduction to the Design and Analysis of Algorithms*, Addison Wesley Company, USA, 2003.

[17] Miroslav K. and Wojciech R., cs.huji.ac.il /labs/.../adhoc /kuty lowski_2003 advers daryimmuneleader.pdf, 2007.

[18] Molina H., "Elections in A Distributed Computing Systems," *Computer Journal of IEEE Transactions on Computers*, vol. 31, no. 1, pp. 48-59, 1982.

[19] Navneet M., Jennifer L., and Welch V., "Leader Election Algorithms for Mobile Ad Hoc Networks," *Technical Document CCR-9972235*, 2001.

[20] Ostrovsky R., Rajagoplan S., and Vazirani U., "Simple and Efficient Leader Election in the Full Information Model," *in Proceedings of the Twenty-Sixth Annual ACM Syposium on Theory of Computing*, Canada, pp. 564-573, 1994.

[21] Power H., *Algorithms and Application in Parallel Computing, WIT Press/ Computational*, Mechanics Publications, USA, 1999.

[22] Refai M. and Ababneh E., "Leader Election Algorithm in 3D Torus Networks," *Master Theses*, Al-Albayet University, 2002.

[23] Refai M. and Ajlouni N., "A New Leader Election Algorithm in Hypercube Networks," *Symposium Proceedings Volume II Computer Science and Engineering and Electrical & Electronics Engineering*, North Cyprus, pp. 497-501, 2006.

[24] Richard E. and Kumarss N., *Foundations of Algorithms Using Java PseudoCode*, Jones and Bartlett Publishers, Canada, 2004.

[25] Russell A., Saks M., and Zuckerman D., "Lower Bounds for Leader Election and Collective Coin-Flipping in the Perfect Information Model," *in Proceedings of the Symposium on the Theory of Computing*, Atlantic City, pp. 25-29, 1999.

[26] Singh G., "Leader Election in the Presence of Link Failures," *Computer Journal of IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 3, pp. 33-38, 1996.

[27] Singh G., "Efficient Distributed Algorithms for Leader Election in Complete Networks," *in Proceedings of 11th IEEE International Conferences on Distributed Computing Systems,* New York, pp. 472-479, 1991.

[28] Singh G., "Efficient Leader Election Using Sense of Direction," *Technical Document KS66506*, Kansas State University, Manhatten, 1997.

[29] Sudarshan V., DeCleene B., Immerman N., Kurose J., and Towsley D., "Leader Election Algorithms for Wireless Ad Hoc Networks," *in Proceedings of IEEE DISCEX III*, California, pp. 291-311, 2003.

[30] Tanenbaum A., *Distributed Systems*, Prentice-Hall International, New Jersey, 2002.

[31] Tanenbaum A., *Distributed Operating Systems*, Prentice-Hall International, New Jersey, 1995.

[32] William K., Nellson D., and Ryan S., http://bkocay.cs.umanitoba.ca/g&g/articles/Torus .pdf, 2007.

[33] William K. and Winnipeg M., http:// bkocay.cs.umanitoba.ca/g&g/articles/Embeddings.pdf, 2007.

[34] Yamshita M. and Kammeda T., "Leader Election Problem on Networks in which Processor Identity Numbers are not Distinct," *Computer Journal of IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 9, pp. 5-10, 1999.

Mohammed Refai received his undergraduate studies in computer science from Mutah University, Jordan and MSc degree in computer science from Alalbayet University, Jordan. He received his PhD in computer science from Amman Arab University for Graduated Studies, Jordan. He is currently worked as assistant professor in the Faculty of Science and Information Technology at Zarqa Private University, Jordan. His main research interests include many aspects in parallel and distributed systems, simulation, and data mining.

Ahmad Sharieh received his BSc in computer science from the University of Tennessee, USA. His MSc in computer sciences from Western Kentucky University, USA. His PhD in computer and information sciences from Florida State University, USA.

Fahad Alshammari received his BSc in computer science, Department of Computer Science, King Saud University, Saudi Arabia. His MSc in computer science, from Department of Computer Science, University of Jordan, Jordan. He is currently a postgraduate studies student, Information Technology and Computer Science College, University of Malaya, Malaysia.