Chapter 4

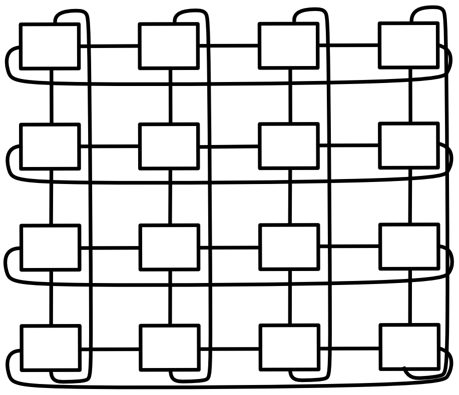
Lea-TN: A Leader Election Algorithm Considering Node and Link Failures in a Torus Network

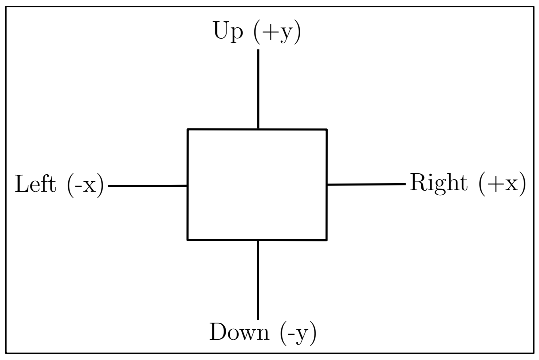
Torus network topology offers many advantages such as higher speed, lower latency, better fairness, and lower energy consumption. For these kinds of benefits, nowadays, it is used to construct many parallel and distributed systems like IBM Blue Gene, IBM Sequoia, Mira, and Sugon TC8600. In this chapter, first, we introduce a lower bound $\Omega(N \log_3 N)$ of message complexity on a comparison-based leader election for a $2D$ torus network (where N is the number of nodes in the network). Next, we propose a leader election algorithm (Lea-TN) for a $2D$ torus network. The Lea-TN algorithm is designed considering both the node and link failures of the torus networks. It is a deterministic algorithm that can elect a leader even there are some link or node failures in the system. We consider the number of non-faulty links and the subsisting nodes' failure rate to elect a reliable leader. The algorithm chooses a node with a higher number of non-faulty links and the least failure rate among the subsisting nodes as the leader. We introduce new patterns for sending messages that help reduce the number of exchanged messages and the execution time of the election process. The proposed algorithm (Lea-TN) enables a node to identify its link failures during the election also. Further, we simulate the Lea-TN algorithm and compare its performance with the well-known existing algorithms.

Outline: The rest of this chapter is organized as follows. Section 4.1 details the system model for which we design the leader election algorithm. Section 4.2 presents a lower bound message complexity of the leader election problem. Section 4.3 describes the proposed leader election method. The complexity analysis, proof of self-stabilization and fault tolerability of the proposed election method are also provided in this section. Section 4.4 present the empirical evaluation of our election method and section 4.5 summaries the chapter.

4.1 System Model

We consider a crash-recovery distributed system [24] consisting of N nodes. The underlying network of this system is a synchronized $2D$ torus with n rows and n columns (where N is a finite integer, $N \geq 9$, $n \geq 3$, and $N = n \times n$). Figure 4.1(a) shows a 4×4 $2D$ torus network. Every node has a unique node identifier (node Id) of $O(\log N)$ bits to identify the node uniquely. All the links in this system are bidirectional, and the nodes communicate with each other by sending messages [91] [23]. In this system, nodes take steps at perfectly synchronized rounds. All messages sent in a round are received at the end of that round. Every perfect node has exactly four edges that connect that node to its adjacent four nodes. Figure 4.1(b) shows a node and its four links. A node i has a send-buffer and a receive-buffer associated with each link of the node. Any system node or link may fail independently and recover from the failure state to the working state. We assume that initially, the system has no node or link failures. Formally, a graph $T_{n,n} = (\Pi, L)$ represents this system, where Π denotes the set of nodes ($|\Pi| = N$ and $\Pi = \{N\_Id_i\}$ where $i = 0, 1, 2, ..., N-1$). If there is no link failure, the degree of a node N_Id_i is 4. L denotes the set of network connectivity between the nodes and $|L| = 2N$.


(a) A 4 × 4 $2D$ torus network.



(b) A node is connected to its adjacent right, left, upper, and lower nodes through the +x, -x, +y, and -y links, respectively.

Figure 4.1: Details of a $2D$ torus network.

Node and Link Failures: We consider a crash-recovery distributed system abstraction [24] for designing a new leader election algorithm. In this abstraction, a node or a link can crash and may subsequently recover. Failure of a node or a link occurs when either stops functioning correctly, and recovery consists in restoring it from an erroneous state to an error-free state. Let's assume that a node or a link does not fail again within a reasonable amount of time after it recovers from the failure state. A link cannot create or alter messages. Whenever a node recovers from the failure state to the working state, it updates itself by collecting the current system information from its adjacent nodes. If a node is recovered during the election, we exclude it from participating in the election. At the end of the election, this node collects the elected leader information from its adjacent nodes.

4.2 Lower Bound

This section proposes a lower bound [42] [77] of the message complexity of a comparison-based leader election for a synchronous $2D$ torus network. The lower bound of the message complexity states that the leader cannot be elected by exchanging fewer messages than that bound. We calculate this lower bound for the worst-case scenario of the leader election problem. When all the nodes of a distributed system realize the leader failure and initiate election (a node starts election by creating an election initiating message) simultaneously, it is the worst-case scenario of the comparison-based leader election problem. Depending on the system's criteria, one node (either the node having the maximum value of the comparison parameter or the minimum value of the comparison parameter) gets elected as the leader in the comparison-based leader election. We assume that the node with the highest comparison parameter value will be elected. Hence, the election initiating message created by the node with the highest comparison parameter value must be sent to all the nodes by exchanging the minimum number of messages. On the other hand, the election initiating messages created by the other nodes have to be discarded as soon as possible. Here, a node with a higher value of the comparison parameter has a higher priority. That is why a node discards an election message created by a node with a lower value of the comparison parameter than itself.

A spanning tree with N nodes enables a message to be broadcast from one node to all other nodes with only $N-1$ edge traversals [81]. So, the spanning-tree protocol broadcasts information from one node to all other nodes using the least number of message exchanges. To find a lower bound message complexity, we first generate a spanning ternary tree from a $2D$ torus network (cf. Figure 4.2). A ternary tree is nothing but a *k-ary* tree [113] [65] (also known as *m-ary* tree) where $k = 3$. If the message created by the node with the highest value of the comparison parameter is broadcasted following the ternary tree, then the minimum number of messages will be exchanged for broadcasting. On the other hand, when the ternary tree is a complete ternary tree, and the value of every parent's comparison parameter is higher than that of its child nodes, then the election initiating messages created by other nodes will be discarded as quickly as possible. In this situation, an election initiating message created by a node traverses only its parent node and the descendant nodes.

Assume h is the height of a complete ternary tree of N nodes, and the total number of messages required to elect a leader is TM .

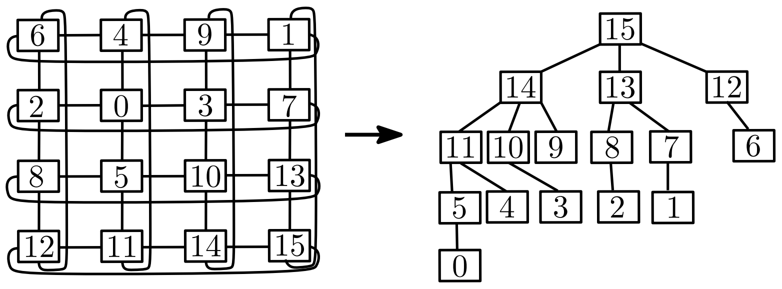


FIGURE 4.2: Building a ternary tree from a (4×4) $2D$ torus network

$$h = \lceil \log_3 N \rceil \tag{4.1}$$

$TM = \sum_{i=0}^{h}$ Sum of the number of parent and descendant nodes for each node of the tree's i^{th} level.

$$\begin{aligned}
&= \sum_{i=0}^{h} 3^i(\frac{3^{h+1-i}-1}{2}) \\
&\geq \sum_{i=0}^{h-1} 3^i(\frac{3^{h-i}-1}{2}) \\
&= \sum_{i=0}^{h-1} \frac{1}{2}(3^h - 3^i) \\
&= \sum_{i=0}^{h-1} \frac{1}{2}(3^h) - \sum_{i=0}^{h-1} \frac{1}{2}(3^i) \\
&= \frac{1}{2}h(3^h) - \frac{1}{4}(3^h - 1) \\
&= \frac{1}{2}h(3^h) - \frac{1}{4}3^h + \frac{1}{4} \\
&= \frac{1}{2}3^h(h - \frac{1}{2}) + \frac{1}{4} \\
&= \frac{1}{2}3^{\lceil \log_3 N \rceil}(\lceil \log_3 N \rceil - \frac{1}{2}) + \frac{1}{4} \\
&\geq \frac{1}{2}3^{\log_3 N}(\log_3 N - \frac{1}{2}) + \frac{1}{4} \\
&= \frac{1}{2}N(\log_3 N - \frac{1}{2}) + \frac{1}{4} \\
&= \frac{1}{2}N \log_3 N - \frac{1}{4}N + \frac{1}{4} \\
&= \Omega(N \log_3 N)
\end{aligned}$$

Thus we can state that the lower bound of the message complexity of a comparison-based leader election for a synchronous $2D$ torus network is $\Omega(N \log_3 N)$.

4.3 Proposed Methodology

Our goal is to devise a new leader election algorithm that can (1) tolerate multiple link and node failures (2) elect a comparatively more reliable leader from among the subsisting nodes (3) satisfy uniqueness, agreement, and termination properties (4) exchange fewer messages and take less time to elect the leader (5) help to identify the link failures during the election.

Before taking the algorithm any further, we first explain assumptions, definitions, types of message, and new patterns for sending the message.

4.3.1 Assumptions

We consider the following assumptions for this system:

- The nodes are homogeneous, but their failure rates may be different.
- Each node has its own timer, and a node can send and receive election messages simultaneously.
- It takes one unit of time to complete a round.

4.3.2 Definitions

The definitions are provided according to their usage in the following work.

Definition 1 ($\boldsymbol{Fr_i}$): It denotes the failure rate[84] [48] of a node i. The failure rate refers to how often something fails, such as a component, a node or an engineered system. We usually express it in failures per unit of time. In general, Weibull distribution [14] [95] is used for failure analysis because of its flexibility in describing failure rate as it can represent all three regions of the bathtub curve. According to two-parameter Weibull distribution, the unreliability $F_{X_i}(t)$ and the failure density function $f_{X_i}(t)$ of a node can be written as

$$F_{X_i}(t) = 1 - e^{-\left(\frac{t}{\eta_i}\right)^{\beta_i}} \tag{4.2}$$

where $\eta_i > 0$ is called the scale parameter, $\beta_i > 0$ is the shape parameter, and t denotes time.

$$f_{X_i}(t) = \frac{d}{dt}F_{X_i}(t) = \frac{\beta_i}{\eta_i}\left(\frac{t}{\eta_i}\right)^{(\beta_i-1)} e^{-\left(\frac{t}{\eta_i}\right)^{\beta_i}} \tag{4.3}$$

So, the failure rate of a node i between times t_0 and t_1 is

$$FR_i = \int_{t_0}^{t_1} f_{X_i}(t)dt \tag{4.4}$$

Here, $\beta_i < 1$ means the failure rate is a decreasing function of time; $\beta_i = 1$ means the failure rate is constant, and $\beta_i > 1$ means the failure rate is an increasing function of time.

Definition 2 ($\boldsymbol{Lfi}$): It refers to the link failure indicator where $Lfi \geqslant 0$. If an election message creator node has no link failure, then Lfi is initialized by 0. Otherwise, it is initialized by 1. After that, when the election message goes through a node which has some link failure, then Lfi is incremented by 1. If the message goes through a node which has no link failure and $Lfi \neq 0$, Lfi is decremented by 1.

Definition 3 ($\boldsymbol{Nnf_i}$): It represents the number of non-faulty links of a node i. In a network, a node is considered as an important node if it has a higher number of non-faulty links, i.e., it is directly connected to many system nodes. In the case of a $2D$ torus network $Nnf_i \in \mathbb{N}$, and $Nnf_i \leq 4$.

Definition 4 ($\boldsymbol{Lfact_i}$): It refers to the leader factor of a node i. We use this factor to elect a reliable and efficient leader for the system. We define a function by combining the number of non-faulty links (Nnf_i) and the failure rate (Fr_i) of a node i to calculate the leader factor of the node. A node with a higher number of non-faulty links has a higher number of communication paths to communicate with the other nodes in the network. Hence, the node elected as the system leader should have a higher number of non-faulty links. As such, once it is the leader, it can easily communicate with the other nodes by balancing the network traffic through different communication paths. Thus, the leader factor of a node should be proportional to the number of non-faulty links of the node i.e.,

$$Lfact_i \propto Nnf_i \tag{4.5}$$

Reliability is the ability of a system or component to perform its required functions under specified conditions for a period of time. A lower failure rate of a node indicates the higher reliability of this node. Hence, the elected system leader should have a lower failure rate, ensuring that the leader will have a long life and that the system will not need to run the election algorithm frequently. Thus, the leader factor of a node should be inversely proportional to the failure rate of the node i.e.,

$$Lfact_i \propto \frac{1}{Fr_i} \tag{4.6}$$

$$Lfact_i = x(Nnf_i) + y(\frac{1}{Fr_i}) \tag{4.7}$$

From the relations (4.5) and (4.6), we build the equation (4.7) to calculate the leader factor of a node i, where $x + y = 1$ and $x, y \geq 0$. According to the requirement, in this equation, we can choose different values between 0 and 1 for x and y to prioritize the number of non-faulty links (Nnf_i) and the failure rate (Fr_i) differently. After combining the equations (4.4) and (4.7), we get

$$Lfact_i = x(Nnf_i) + y(\frac{1}{\int_{t_0}^{t_1} f_{X_i}(t)dt}) \tag{4.8}$$

Definition 5 (M_Lfact_i): It is used to store the maximum leader factor of a node by a node i. Initially, node i hoards its own leader factor in M_Lfact_i. Then, node i updates M_Lfact_i according to the received election messages. Node i replaces the lower M_Lfact_i by the higher one if the leader factor of the received message generator node is greater than M_Lfact_i.

Definition 6 ($N\_Id\_Mlf_i$): During the election, a node i uses the $N\_Id\_Mlf_i$ to store the Id of a node that has the maximum leader factor. A node i updates $N\_Id\_Mlf_i$ according to the received election messages. Node i replaces the Id $N\_Id\_Mlf_i$ by the Id of the received message generator node if the leader factor of the received message generator node is greater than that of the node stored in $N\_Id\_Mlf_i$.

Definition 7 ($Flist_i$): A node i uses the list $Flist_i$ to store node and link failures of its adjacent nodes and links. This list contains the node status of a node and its adjacent nodes as well as the link status of all the links associated with these nodes. For example in Table 4.1, we show the $Flist_i$ of a node i for a particular instant of time. Here, 1 denotes the aliveness of a node or link, and 0 denotes the failure of a node or link. Whenever a node gets failure or recovery information of the adjacent nodes and links, it updates its $Flist_i$ accordingly. A node i sets the value of its Nfl_i according to the link status information of its $Flist_i$.

TABLE 4.1: List for store the information of adjacent node and link failures of a node i

		Link Status			
Node	Node Status	+y	+x	-y	-x
Self (node i)	1	1	0	1	1
Upper node	1	1	1	1	1
Right node	0	0	0	0	0
Lower node	1	1	1	1	1
Left node	1	1	1	1	1

4.3.3 Types of Message and Message Sending Pattern

In our proposed algorithm, we use three types of messages i.e., election message, acknowledgement message, and failure information message. The election message is used to start the election. It has five fields, namely the Election message creator Id (Emc_Id), the Failed leader Id (Fl_Id), the Leader Factor ($Lfact_c$) of the message creator node, the Link failure indicator (Lfi), and the Message sending pattern (Msp). This message is represented by $Msg[Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp]$. An acknowledgement message is used to know the proper delivery of the election message. When a node sends an election message to its adjacent node, it expects an acknowledgement from that adjacent node. The acknowledgement message has two fields. The first field is the Id of the node that creates the acknowledgement message (Amc_Id), and the second field is the received election message creator Id ($Remc\_Id$). The acknowledgement message is represented by $Ack[Amc\_Id, Remc\_Id]$. A node uses the failure information message to inform its adjacent node about its link failure. After sending an election message through a link, a node waits for an acknowledgement. If the node does not get the acknowledgement through that link during a time out period, it detects that the link is failed. The node then generates a failure information message and sends it to its adjacent nodes. This message has three fields i.e., the failure information message generator Id ($Fimc\_Id$), the link status bits (Lsb), and the Id of the election message creator that helped to identify the link failure (Emc_Lf_Id). We use four bits as the link status bits (Lsb) to represent the link status of a node. First, second, third, and fourth bits (from left to right) refer to the status of +y link, +x link, -y link, and -x link, respectively. 0 indicates the link failure, and 1 indicates the link aliveness. For example 1011 indicates that +x is failed and the rest of the links are alive. The failure information message is represented by $Fim[Fimc\_Id, Emc\_Lf\_Id, Lsb]$.

Now, we introduce three specific message sending patterns (pattern 1, pattern 2, and pattern 3) and one particular message sending pattern sequence (pattern 2 – pattern 3 – pattern 1) to transmit the election message. We use these patterns and sequence of patterns to reduce the number of message exchanges during the election.

Pattern 1: When a node receives an election message, and the message sending pattern is 1, then the node sends the message to the front node regarding the incoming link and direction of the message. In Figure 4.3 (c), node 1 gets an election message from node 5 through the $-x$ link, and the message sending pattern is 1. Node 1 sends the message to node 3, because as per the $-x$ link and the direction of the incoming message, node 3 is the front node of node 1.

Pattern 2: If a node receives an election message, and the message sending pattern is 2, then the node transmits the message to the front and right nodes as per the incoming link and direction of the message. For example, in Figure 4.3 (a), node 1 receives an election message from node 5 through the $-x$ link and the message sending pattern is 2. The node then sends the message to nodes 3 and 4, because as per the $-x$ link and the direction of the incoming message, node 3 is the front node and node 4 is the right node of node 1.

Pattern 3: If a node receives an election message, and the message sending pattern is 3, the node transmits the message to all its adjacent nodes except the node that sent the election message. For example, in Figure 4.3 (b), node 1 receives an election message from node 5 through the $-x$ link, and the message sending pattern of the message is 3, then node 1 sends the message to nodes 2, 3 and 4 but not to node 5 because node 1 received the message from node 5.

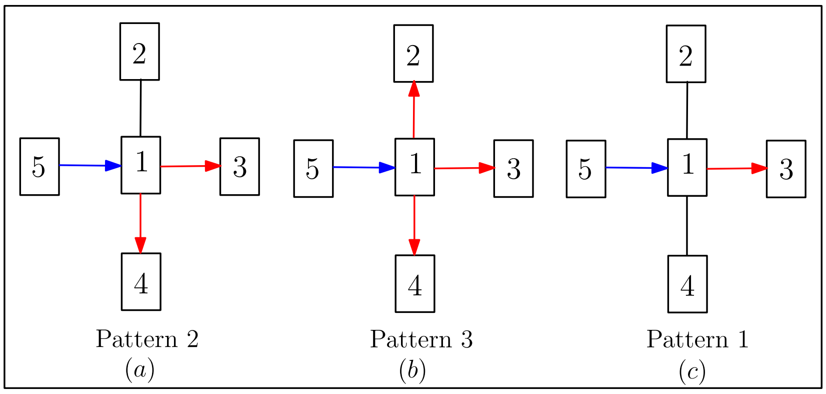


FIGURE 4.3: Message sending patterns

Pattern sequence: Using the above-discussed message sending patterns (pattern 1, pattern 2, and pattern 3), we introduce a particular sequence i.e., pattern 2 – pattern 3 – pattern 1 to send the election message. Whenever a node detects the leader's failure, it creates an election message and transmits this message to all its adjacent nodes to initiate the election. After that, if there is no link failure, the message is broadcast following this message sending pattern sequence (pattern 2 – pattern 3 – pattern 1) repeatedly.

After getting an election message, if a node finds that the message field Lfi is 0 and the node itself has no link failure, it follows the message sending pattern sequence and transmits the election message according to the value of the Msp of the received message. If the Lfi is not equal to 0 or the node itself has any link failure, then the node transmits the election message to all its adjacent nodes except the node that sent the election message.

4.3.4 Description of Lea-TN

Throughout the Lea-TN algorithm, we use three procedures: $Create\_Msg()$, $Check\_Link\_Failure()$ and $Send\_Msg()$. Procedure $Create\_Msg()$ creates an election message. Procedure $Check\_Link\_Failure()$ checks the $Flist_i$, finds the number of link failures and sets the election message sending pattern (value of Msp). Whereas, procedure $Send\_Msg()$ sends the election message according to the message sending pattern. We describe $Create\_Msg()$, $Check\_Link\_Failure()$ and $Send\_Msg()$ below.

$\boldsymbol{Create\_Msg()}$

1. $Emc\_Id \leftarrow N\_Id_i$, $Lfact_c \leftarrow Lfact_i$, $N\_Id\_Mlf_i \leftarrow N\_Id_i$ and $M\_Lfact_i \leftarrow Lfact_i$
2. If $Nfl_i == 0$, then $Msp \leftarrow 2$ and $Lfi \leftarrow 0$
3. If $Nfl_i \neq 0$, then $Msp \leftarrow 3$ and $Lfi \leftarrow 1$
4. Create an election message $Msg[Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp]$ and send it to all the adjacent nodes.

$\boldsymbol{Check\_Link\_Failure(Nfl_i, Lfi)}$

1. If $Nfl_i \neq 0$, then $Msp \leftarrow 3$ and $Lfi \leftarrow Lfi + 1$.
2. If $Nfl_i == 0$ and $Lfi \neq 0$, then $Msp \leftarrow 3$ and $Lfi \leftarrow Lfi - 1$.

$\boldsymbol{Send\_Msg(Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp)}$

1. If $Msp == 3$ and $Lfi > 0$, then $Msp \leftarrow 3$ and send the election message $Msg[Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp]$ according to Pattern 3.
2. If $Msp == 3$ and $Lfi == 0$, then $Msp \leftarrow 1$ and send the election message $Msg[Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp]$ according to Pattern 3.
3. If $Msp == 2$, then $Msp \leftarrow 3$ and send the election message $Msg[Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp]$ according to Pattern 2.
4. If $Msp == 1$, then $Msp \leftarrow 2$ and send the election message $Msg[Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp]$ according to Pattern 1.

The leader election algorithm is composed of three phases:

- Election commencement
- Received message processing
- Elected leader identification

Algorithm 5: Election commencement

```
// How a node i starts the election.
if (A node i realizes that the leader is crashed) then
    $Create\_Msg()$
    Start timer for $2(T_d + 3)$ rounds.
end
```

Election commencement: When a node perceives that the current leader is crashed, it starts the election by making an election message, transmits this message to all the adjacent nodes, and expects an acknowledgement from each adjacent node in the next round. Let us assume that a node i detects that the leader is crashed.

Algorithm 6: Message Processing

```
// Execution of a round by a node $i$. In a round, a node $i$ may get a maximum of three types
   message i.e., $Msg[Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp]$, $Ack[Amc\_Id, Remc\_Id]$, and
   $Fim[Fimc\_Id, Emc\_Lf\_Id, Lsb]$
1  if (Node i gets atleast one election message) then
2      Create an acknowledgement message ($Ack[Amc\_Id, Remc\_Id]$) regarding every received election message
         and send it to the respective nodes.
3      if (The $Lfact_c$ of all the received messages is the same) then
4          Choose the message ($Msg[Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp]$) that has the maximum $Emc\_Id$ among
             all the received election messages.
5      else
6          Choose the message ($Msg[Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp]$) that has the maximum $Lfact_c$ among
             all the received election messages.
7      end
8      Discard the other received election message(s).
9      if ($N\_Id\_Mlf_i == Fl\_Id$) then
10         if ($Lfact_i < Lfact_c$) then
11             $N\_Id\_Mlf_i \leftarrow Emc\_Id$, $M\_Lfact_i \leftarrow Lfact_c$
12             $Check\_Link\_Failure(Nfl_i, Lfi)$
13             $Send\_Msg(Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp)$
14         else
15             if ($Lfact_i == Lfact_c$) then
16                 if ($N\_Id_i > Emc\_Id$) then
17                     Discard the received election message.
18                     $Create\_Msg()$
19                 else
20                     $N\_Id\_Mlf_i \leftarrow Emc\_Id$
21                     $Check\_Link\_Failure(Nfl_i, Lfi)$
22                     $Send\_Msg(Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp)$
23                 end
24             else
25                 Discard the received election message.
26                 $Create\_Msg()$
27             end
28         end
29         Start timer for $2(T_d + 3)$ rounds.
30     else
31         if ($M\_Lfact_i < Lfact_c$) then
32             $N\_Id\_Mlf_i \leftarrow Emc\_Id$, $M\_Lfact_i \leftarrow Lfact_c$
33             $Check\_Link\_Failure(Nfl_i, Lfi)$
34             $Send\_Msg(Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp)$
35         else
36             if ($M\_Lfact_i == Lfact_c$) then
37                 if ($N\_Id\_Mlf_i \geq Emc\_Id$) then
38                     Discard the received election message.
39                 else
40                     $N\_Id\_Mlf_i \leftarrow Emc\_Id$
41                     $Check\_Link\_Failure(Nfl_i, Lfi)$
42                     $Send\_Msg(Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp)$
43                 end
44             else
45                 Discard the received election message.
46             end
47         end
48     end
49 end
50 if (Node i does not get all the expected acknowledgement message(s) ) then
51     Modify the $Flist_i$ and $Nfl_i$, and set each bit value of $Lsb$ according to the acknowledgement message(s)
         that it was supposed to get but did not.
52     Create the link failure information message and send it to the adjacent nodes.
53     Set $Msp \leftarrow 3$ and $Lfi \leftarrow 3$ of the election message generated by the node $N\_Id\_Mlf_i$ and send the
         message to the adjacent node(s), which did not get this election message before.
54 end
55 if (Node i gets at least one link failure information message) then
56     Modify the $Flist_i$ and $Nfl_i$ according to the received link failure information messages' $Lsb$.
57     Compare the $Emc\_Lf\_Id$ of each link failure information message with the $N\_Id\_Mlf_i$
58     if ($N\_Id\_Mlf_i == Emc\_Lf\_Id$) then
59         Set $Msp \leftarrow 3$ and $Lfi \leftarrow 3$ of the election message generated by the node $N\_Id\_Mlf_i$ and send the
             message to the adjacent node(s), which did not get this election message before.
60     end
61 end
62 if (Node i gets a request from an adjacent node that recovered from the failure state during the election) then
63     Send the current system information to that node and prohibit it from taking part in the running election.
64 end
```

Algorithm 7: Elected leader identification

```
// How a node $i$ identifies the elected leader.
1  if (The allocated time of the timer of a node i is out.) then
2      $L\_Id_i \leftarrow N\_Id\_Mlf_i$
3      if ($N\_Id_i == L\_Id_i$) then
4          $Status_i \leftarrow Leader$
5      else
6          $Status_i \leftarrow Non\text{-}leader$
7      end
8      The node Id stored in $L\_Id_i$ is elected as the new leader.
9      Send the newly elected leader's information to the adjacent node(s) that recovered
         from the failure state during the election.
10 end
```

First, node i checks the list that contains the adjacent node and link failures ($Flist_i$) to identify the number of its failed link(s) (Nfl_i) and sets the values of Msp and Lfi according to the value of Nfl_i. If node i has no link failure, i.e., $Nfl_i == 0$, the values of Msp and Lfi are set to 2 and 0, respectively. Otherwise, the values of Msp and Lfi are set to 3 and 1, respectively. Next, node i creates an election message $Msg[Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp]$, (where $EMCID \leftarrow N\_Id_i$, $Lfact_c \leftarrow Lfact_i$) and transmits it to all the adjacent nodes. Node i copies its own Id into its $N\_Id\_Mlf_i$, takes part in the election as a contender and starts its timer for $2(T_d + 3)$ time units.

Received message processing: We now explain how, in a round, a node i processes different types of received messages and progresses the election. In a round, a node can get three types of message (election message, acknowledgement message, and failure information message). First, node i checks whether it has got an election message(s). If it has, it creates an acknowledgement message ($Ack[Amc\_Id, Remc\_Id]$) regarding every received election message and send them to the respective nodes that sent the election message(s). Node i then chooses the election message that has maximum $Lfact_c$ among all the received election messages and discards the other. In a round, if a node receives all the messages with the same $Lfact_c$, the message with the highest Emc_Id among them will get preference. Assume node i chooses the message $Msg[Emc\_Id, Fl\_Id, Lfact_c, Lfi, Msp]$

whose $Lfact_c$ is maximum among all the received messages. It then compares its $N\_Id\_Mlf_i$ with the message's Fl_Id. If $N\_Id\_Mlf_i == Fl\_Id$, that means node i is not an election initiator and it has received the election message for the first time. In this case, the node compares its $Lfact_i$ with the leader factor of the message-creator node ($Lfact_c$). If $Lfact_c > Lfact_i$ or '$Lfact_c == Lfact_i$ and $Emc\_Id > N\_Id_i$', then the node i copies the values of Emc_Id and $Lfact_c$ into its $N\_Id\_Mlf_i$ and M_Lfact_i, respectively. It then sets the values of Msp and Lfi based on the number of failed links of its own. After that, node i transmits this processed message according to the message sending pattern of that message and starts the timer for $2(T_d+3)$ time units. If $Lfact_c < Lfact_i$ or '$Lfact_c == Lfact_i$ and $Emc\_Id < N\_Id_i$', then node i constructs a new election message, and transmits it to all the adjacent nodes and starts the timer for $2(T_d + 3)$ time units. If node i is an election initiator or it has already processed at least one message, then it compares its M_Lfact_i with the received message's $Lfact_c$. If $Lfact_c > M\_Lfact_i$ or '$Lfact_c == M\_Lfact_i$ and $Emc\_Id > N\_Id_i$', the node i copies the values of Emc_Id and $Lfact_c$ into it's own $N\_Id\_Mlf_i$ and M_Lfact_i, respectively. It then sets the values of Msp and Lfi based on the number of its own failed links. It then transmits this processed message according to the message sending pattern of that message. Otherwise, the node i discards the message.

In a round, if a node i sends an election message to its adjacent node(s), it expects acknowledgement message(s) from that (those) adjacent node(s) in the next round. If a node does not get all the expected acknowledgement message(s) of a round, then it realizes that some of its links are failed. The node then modifies the $Flist_i$ and Nfl_i, and sets each bit value of Lsb based on the expected acknowledgement message(s) that it did not get from the adjacent node(s). After that, it creates a link failure information message ($Fim[Fimc\_Id, Emc\_Lf\_Id, Lsb]$) and sends this message to all the adjacent nodes. At the same time, node i sets the value of Msp and Lfi of the election message generated by the node whose Id is stored in $N\_Id\_Mlf_i$ and sends the message to the adjacent node(s) that did not get this election message before.

In a round, if a node i gets link failure information message(s), it modifies the $Flist_i$ and Nfl_i according to the link status bits Lsb of the received link failure information messages(s). After that, the node compares the failure information message generator Id ($Fimc\_Id$) of each message with its $N\_Id\_Mlf_i$. If $Fimc\_Id == N\_Id\_Mlf_i$, node i sets the value of Msp and Lfi of the election message generated by the node whose Id is stored in $N\_Id\_Mlf_i$ and sends the message to the adjacent node(s) that it did not get this election message before.

Elected leader identification: Finally, when a timer's (timer of node i) preset time is out ("Session Time Out"), then node i gets to know about the newly elected leader. It recognizes the node whose Id is stored in its $N\_Id\_Mlf_i$ as the elected leader. It then copies the node Id stored in its $N\_Id\_Mlf_i$ into L_Id_i. If $N\_Id_i == L\_Id_i$, the node sets its status as a leader ($Status_i \leftarrow Leader$). Otherwise, the node sets its status as a non-leader ($Status_i \leftarrow Non\text{-}leader$).

Lemma 4.1. *If there is no link failure in $T_{n,n}$, the proposed message sending method (message sending patterns and message sending pattern sequence) takes a maximum of $\sqrt{N}$ rounds to send an election message to all the nodes in the system.*

Proof: According to our system model, in a round, a node can send and receive messages simultaneously. When a node initiates the election, it creates an election message and transmits it to its four adjacent nodes. After that, the message is broadcast following the message sending pattern sequence (pattern 2 – pattern 3 – pattern 1) repeatedly. We observe that a maximum 4 adjacent nodes of the message generator node get the election message at the end of the first round. At the end of the second round, 8 more nodes get the election message. At the end of the third round, 12 more nodes get the election message. In this way, the election message continues to be broadcast up to the $\lfloor\sqrt{N}/2\rfloor$ rounds. After the $\sqrt{N}/2^{th}$ round, in each round, the number of nodes that get the election message is reduced by 4 than the number of nodes of the previous round. We thus observe that a total 5 nodes (including the election message generator node) get the message at the end of the first round. At the end of the second round, a total of 13 nodes get the election message.

At the end of the third round, a total of 25 nodes get the election message and so on. That means the election message is broadcast over the network following the centered square number sequence. We now obtain the following sequence (sequence (4.9)), where N is the total number of nodes of a $T_{n,n}$ network, and an odd integer. In this sequence, the i^{th} term represents the total number of nodes that get the election message after the i^{th} round. So, the total number of terms of this sequence represents the total number of rounds required to broadcast the election message all over the network.

$$\{5, 13, 25, 41, ..., (N+1)/2, (N+4\sqrt{N}-3)/2, ..., (N-12), (N-8), (N-4), N\} \tag{4.9}$$

Sequence (4.9) consists of two sequences i.e., sequence (4.10) and sequence (4.11)

$$\{5, 13, 25, 41, ..., (N+1)/2\} \tag{4.10}$$

and

$$\{N, (N-4), (N-8), ..., (N+4\sqrt{N}-3)/2\} \tag{4.11}$$

Assume sequences (4.10) and (4.11) have y and z number of terms, respectively. The general term of the sequence (4.10) is $2x^2+2x+1$. Hence, the y^{th} term is $2y^2+2y+1$. As the last term is $(N+1)/2$ we get the total number of terms (y) in the sequence (4.10) from the following equation:

$$2y^2+2y+1 = (N+1)/2 \tag{4.12}$$

From equation (4.12), we get

$$4y^2+4y-(N-1) = 0 \tag{4.13}$$

If we solve the quadratic equation (4.13), we get $y = (\sqrt{N}-1)/2$ or $y = -(\sqrt{N}+1)/2$. However $y = -(\sqrt{N}+1)/2$ is not acceptable, because the total number of terms of a sequence cannot be a negative number. So, $y = (\sqrt{N}-1)/2$. On the other hand, the general term of the sequence (4.11) is $N - (2x^2 - 2x)$. Hence, the z^{th} term is $N - (2z^2 - 2z)$. As the last term is $(N + 4\sqrt{N} - 3)/2$ we get the total number of terms (z) in the sequence (4.11) from the following equation:

$$N - (2z^2 - 2z) = (N + 4\sqrt{N} - 3)/2 \tag{4.14}$$

If we solve the quadratic equation (4.14), we get $z = (\sqrt{N} - 1)/2$. So, the total number of terms in sequences (4.10) and (4.11) is $y + z = (\sqrt{N} - 1)$. That means, if N is odd, an election message takes $(\sqrt{N} - 1)$ rounds to reach all the nodes in the system. Likewise, if N is even, we obtain the following sequence (sequence (4.15)).

$$\{5, 13, 25, 41, ..., (N/2-\sqrt{N}+1), (N/2+\sqrt{N}-1), ..., (N-13), (N-5), (N-1), N\} \tag{4.15}$$

In the same way as we find out the number of terms in sequence (4.9), we can prove that sequence (4.15) has $\sqrt{N}$ number of terms. That means, if N is even, an election message takes $\sqrt{N}$ rounds to reach all the nodes in the system. So, we can conclude that if there is no link failure in $T_{n,n}$, the proposed message sending method (message sending patterns and message sending pattern sequence) takes a maximum of $\sqrt{N}$ rounds to send an election message to all the nodes in the system. ■

Lemma 4.2. *Let us consider a $T_{n,n}$, where n is even, $n > 2$ and $N - 2\sqrt{N}$ links are removed (link failures) in the following manner:* (1) *we divide the $T_{n,n}$ network into four identical parts i.e., Parts 1, 2, 3, and 4. Each part contains $N/4$ nodes, and the diameter of each part is $\sqrt{N} - 2$.* (2) *From the $T_{n,n}$, all the wrap-around links ($2\sqrt{N} - 4$ links) that connect the nodes of first and last rows and first and last columns, except the wrap-around links connecting the four corner nodes are removed.*

(3) *From each part, we remove all the internal horizontal links* ($N/4 - 3\sqrt{N}/2 + 2$ *links*). (4) *The* ($2\sqrt{N} - 4$) *links connecting these four parts (Parts 1, 2, 3, and 4) are removed, except the link connecting the four central nodes of network* $T_{n,n}$. *Now, we prove that this* $N - 2\sqrt{N}$ *number of link removals (link failures) from the* $T_{n,n}$ *does not change the diameter of the remaining* $T_{n,n}$.

Proof: The remaining $T_{n,n}$ is a symmetric network and has four identical parts as in Figure 4.4 (In this figure, we consider a $T_{8,8}$ network and delete 48 links from the $T_{8,8}$ by following the manner as described above and get the remaining $T_{8,8}$ network). The diameter of each identical part is $\sqrt{N} - 2$ and every part contains $N/4$ nodes. Two extreme diagonal nodes (red colored nodes in Figure 4.4) of each identical part are connected to the other identical parts of the remaining $T_{n,n}$ network. A maximum of two hops is needed to go from these extreme diagonal nodes of one part of the network to any other part. A node uses the path that goes through the nearest extreme diagonal node to send a message to any other node of the different parts of the network. In other words, half the nodes of an identical part use one extreme diagonal node and the rest of the nodes use another extreme diagonal node. So, the maximum graph distance between any two nodes will be $(\sqrt{N}-2)/2+(\sqrt{N}-2)/2+2 = \sqrt{N}$. Hence, the diameter of the remaining $T_{n,n}$ network is $\sqrt{N}$. Even the removal of all the internal vertical links instead of all internal horizontal links from each identical part of the $T_{n,n}$ network, the diameter of the remaining $T_{n,n}$ network will remain the same. ■

Lemma 4.3. *Let us consider a* $T_{n,n}$, *where* n *is odd,* $n > 2$ *and* $N - 2\sqrt{N} - 3$ *links are removed (link failures) as follows* (1) *we divide the* $T_{n,n}$ *network into four identical parts i.e., Parts 1, 2, 3, and 4. Each part contains* $(N+1)^2/4$ *nodes, and the diameter of each part is* $2\lfloor\sqrt{N}/2\rfloor$. *These four parts have at least one and at most* $\lceil\sqrt{N}/2\rceil$ *common nodes.* (2) *From the* $T_{n,n}$, *we remove all the wrap-around links* ($2\sqrt{N} - 6$ *links*) *that connect the nodes of the first and last rows and the first and last columns except the wrap-around links that connect the four corner nodes and the middle nodes of the first and last rows and the first and last columns.* (3) *From each part, we remove all the internal horizontal links* ($(N-4\sqrt{N}+3)/4$ *links*).

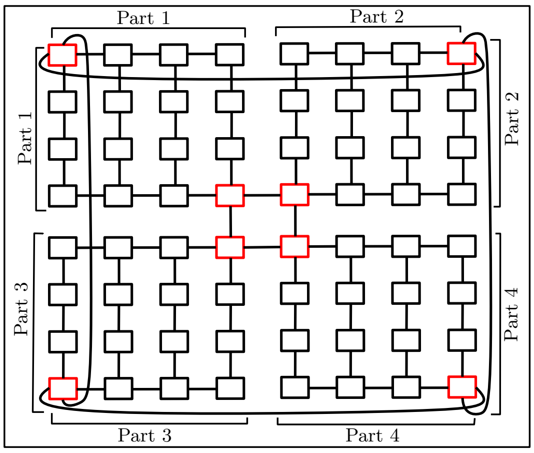


FIGURE 4.4: Example of the remaining $T_{8,8}$ network, after deleting 48 links from the $T_{8,8}$ network in the manner described above.

We prove that this $N - 2\sqrt{N} - 3$ number of link removals (link failures) from the $T_{n,n}$ does not change the diameter of the remaining $T_{n,n}$.

Proof: The remaining $T_{n,n}$ is a symmetric network. It has four identical parts as in Figure 4.5 (In this figure, we consider a $T_{9,9}$ network and delete 60 links from the $T_{9,9}$ by following the manner as described above and get the remaining $T_{9,9}$ network), and the diameter of each part is $2\lfloor\sqrt{N}/2\rfloor$. There is at least one and at most $\lceil\sqrt{N}/2\rceil$ common nodes between any two parts of the remaining $T_{n,n}$ network. In Figure 4.5, blue colored nodes are common nodes. No extra hop is needed to go from one part to another part of the network through these common nodes. Each part of the remaining $T_{n,n}$ network has four extreme diagonal nodes. Among these four nodes, three are directly connected to the other parts of the network, while the fourth node (red colored node in Figure 4.5) is directly connected to two parts and indirectly connected to another part of the network. So, a maximum of two hops is needed to go from these nodes to any other part of the network. Any two parts of the remaining $T_{n,n}$ network have some common nodes. The nodes of any two parts that are a maximum of $(\sqrt{N} - 1)/2$ graph distance away from a particular common node can communicate using the path that goes through that particular common node. Now consider any two parts that have only one common node (Parts 1 and 4 or Parts 2 and 3). The nodes that are a maximum of $(\sqrt{N} - 1)/2$ graph distance away from this common node can communicate using the path that goes through this common node. So, the maximum graph distance between any two among these nodes is $(\sqrt{N}-1)/2 + (\sqrt{N}-1)/2 = (\sqrt{N}-1)$. The nodes that are beyond the $(\sqrt{N}-1)/2$ graph distance away from the common node are a maximum of $(\sqrt{N}-1)/2-1$ graph distance away from any one of the three other extreme diagonal nodes. So, these nodes can communicate using the path that goes through the nearest extreme diagonal node. Hence, the maximum graph distance between any two among these nodes is $(\sqrt{N}-1)/2-1+(\sqrt{N}-1)/2-1+2 = (\sqrt{N}-1)$. Likewise, we can prove that the maximum graph distance between any two nodes of two different parts that have $\lceil\sqrt{N}/2\rceil$ common nodes is also $(\sqrt{N}-1)$. If N is a square number, then $(\sqrt{N}-1) = 2\lfloor\sqrt{N}/2\rfloor$. Thus, the diameter of the remaining $T_{n,n}$ network is $2\lfloor\sqrt{N}/2\rfloor$. From each identical part of the $T_{n,n}$ network, instead of removing all the internal horizontal links, if all the internal vertical links are removed, then the diameter of the remaining $T_{n,n}$ network is not changed. ■

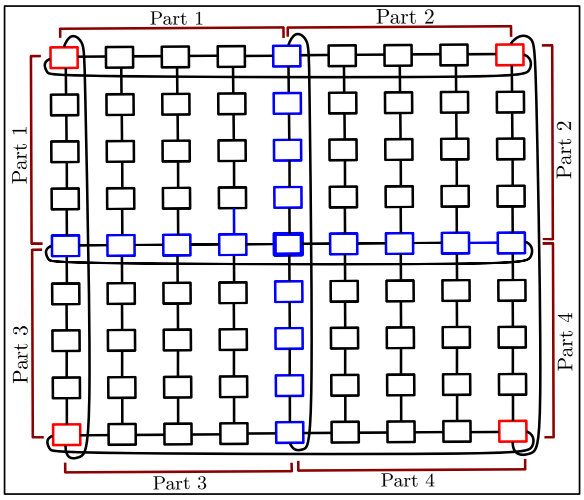


FIGURE 4.5: Example of the remaining $T_{9,9}$ network, after deleting 60 links from the $T_{9,9}$ network in the manner described above.

Lemma 4.4. *If the link failures or/and node failures in the $T_{n,n}$ do not change the diameter of the $T_{n,n}$, the proposed message sending method (message sending patterns and message sending pattern sequence) takes a maximum of $(\sqrt{N}+3)$ rounds to send an election message to all the nodes in the system.*

Proof: In lemma 4.1, we proved that if there is no link failure in $T_{n,n}$, the proposed message sending method takes a maximum of $\sqrt{N}$ rounds to send an election message to all the nodes in the system. In the case of link failure, if a node receives its link failure information in advance, instead of following the message sending pattern sequence, the node transmits the election message to all the adjacent nodes except the node that sent the election message. As the link or/and node failures do not change the $T_{n,n}$ network's diameter, a maximum of $\sqrt{N}$ rounds is required to broadcast a message all over the system. On the other hand, if a node does not receive its link failure information in advance, it can detect link failure during the election too. In a round, after sending an election message through a particular link, if a node does not get an acknowledgement through that specific link in the next round, then the node identifies that the connection is failed. Whenever a node determines link failure(s), it creates a failure information message and sends it to the adjacent nodes connected through the active links. At the same time, the node sends the election message to the adjacent node(s) that did not get the election message before, and that are connected through an active link. A node thus needs three extra rounds to handle the link failure situation. In a round, the nodes that participate in the election's progress can receive, process, and send the message simultaneously. Hence, a total of $(\sqrt{N} + 3)$ rounds is required to send a message to all the nodes in the system. ■

Theorem 4.5. *The proposed algorithm can tolerate* $N - 2\sqrt{N}$ *or* $N - 2\sqrt{N} - 3$ *link failures under the following condition.*

$$LFT_{pa} = \begin{cases} N - 2\sqrt{N}, & \textit{if } N \textit{ is even and the link failures occur as described in Lemma 4.2.} \\ N - 2\sqrt{N} - 3, & \textit{if } N \textit{ is odd and the link failures occur as described in Lemma 4.3.} \end{cases}$$

Proof: To prove this lemma, we have to show that the proposed algorithm can elect a node as the system leader, even if there are $N - 2\sqrt{N}$ (if N is even) or $N - 2\sqrt{N} - 3$ (if N is odd) link failures in the system. In Lemma 4.1, we have proved that if there is no link failure in the system, then $\sqrt{N}$ rounds are required to broadcast a message all over the system. On the other hand, in Lemma 4.4, if the link or/and node failures in the $T_{n,n}$ do not change the diameter of $T_{n,n}$, then a maximum of $(\sqrt{N}+3)$ rounds is required to broadcast a message all over the system. In Lemmas 4.2 and 4.3, we have proved that $N-2\sqrt{N}$ (if N is even) or $N-2\sqrt{N}-3$ (if N is odd) link failures do not change the diameter of the remaining $T_{n,n}$ network. So, a maximum of $(\sqrt{N}+3)$ rounds is required to broadcast an election message all over the system. A situation where only one node can initiate the election, and the node with the maximum leader factor is $\sqrt{N}$ hops away from the election initiator. In this case, a maximum of $2(\sqrt{N}+3)$ rounds is required to elect the node with the maximum leader factor as the system leader. According to the proposed algorithm, a node starts its timer for $2(\sqrt{N}+3)$ rounds. So, the Lea-TN algorithm can elect a leader even if there are $N-2\sqrt{N}$ (if N is even) or $N-2\sqrt{N}-3$ (if N is odd) link failures in the system. ■

4.3.5 Proof of Self-stabilization

Uniqueness, *agreement*, and *termination* are three essential conditions of a self-stabilizing leader election algorithm that help build a consistent system. Here we prove that the Lea-TN algorithm satisfies these three conditions. We prove the *uniqueness* and *agreement* properties by using the contradiction method.

Lemma 4.6. *Every proper execution of the Lea-TN algorithm elects only one node as the system leader until the failure of nodes and/or links increases the network's diameter.*

Proof: After the leader crashes, the Lea-TN algorithm elects the node that has the highest leader factor in the system as the new leader. Here, two cases may appear: (1) every node has a distinct leader factor, (2) multiple nodes have the same leader factor. Assume that two nodes i and j get elected as leaders simultaneously. So, node i contains the highest leader factor $(Lfact_i)$, while node j also contains the highest leader factor $(Lfact_j)$. That means $Lfact_i > Lfact_j$ as well as $Lfact_i < Lfact_j$. The first scenario cannot occur because every node has a distinct leader factor. Hence, in this case our assumption is wrong. In the second case, multiple nodes may have the highest leader factor. Here, nodes i and j are elected simultaneously so $Lfact_i = Lfact_j$, where $Lfact_i$ and $Lfact_j$ are the leader factors of nodes i and j, respectively. In this situation, the Lea-TN algorithm elects the node with the maximum node Id among these nodes that has the highest leader factor. As node i gets elected as the leader so $N\_Id_i > N\_Id_j$. On the other hand, node j also gets elected as the leader so $N\_Id_i < N\_Id_j$. Our system model states that every node has a unique node Id. Hence, $N\_Id_i > N\_Id_j$ and $N\_Id_i < N\_Id_j$ are not possible simultaneously. So, in the second case also our assumption is wrong. Therefore, nodes i and j cannot be elected as the system leaders simultaneously, and the uniqueness condition is met. ■

Lemma 4.7. *In every proper execution of the Lea-TN algorithm, all the system nodes agree with the elected leader until the failure of nodes and/or links increases the network's diameter.*

Proof: As stated in the Lea-TN algorithm, when a system leader crashes, the node with the highest leader factor gets elected as the new leader. When a node initiates the election or receives an election message for the first time, it starts its timer for $2(T_d + 3)$ rounds. After the time out, a node gets to know about the newly elected leader. During the election, the election message created by the node with the highest leader factor gets spread over the whole network, and the rest message(s) gets discarded. An election message takes $(T_d + 3)$ rounds to spread over the whole network. In the worst case scenario, every node gets the election message created by the node with the highest leader factor within $2(T_d + 3)$ rounds. Hence, after the time out of the timer, every node gets the node's information with the highest leader factor (elected leader Id) and agrees with the elected leader by storing the elected leader Id into its L_Id. ■

Lemma 4.8. *The Lea-TN algorithm is terminated in a finite time.*

Proof: After the leader crashes, when a node commences the election or receives an election message for the first time, it starts its timer for $2(T_d + 3)$ rounds. After the time out, a node is informed of the newly elected leader. As our system is synchronous, there is a known upper bound of message processing delay and message transmission delay (from one node to another node). It thus takes finite time units to complete a round. Here, T_d represents the diameter of the $T_(n, n)$ network. So $T_d = 2\lfloor\sqrt{N}/2\rfloor$ and $2(T_d + 3) = 2(2\lfloor\sqrt{N}/2\rfloor + 3)$. According to our system model, N is a finite integer. So $2(2\lfloor\sqrt{N}/2\rfloor+3)$ is also a finite number. If the upper bound of completion of a round is c time units (where c is a constant), then a leader gets elected within $2c(2\lfloor\sqrt{N}/2\rfloor + 3)$ time units that is finite. So, the Lea-TN algorithm is terminated in a finite time. ■

Theorem 4.9. *The Lea-TN algorithm satisfied the self-stabilizing conditions of a leader election algorithm until the failure of nodes and/or links increases the network's diameter.*

Proof: Lemma 4.6, Lemma 4.7 and Lemma 4.8 prove that the Lea-TN algorithm satisfies the uniqueness, agreement and termination conditions respectively. That means the Lea-TN algorithm satisfies all the three conditions of a self-stabilizing leader election algorithm until the failure of nodes and/or links increases the network's diameter. Hence, the Lea-TN algorithm acts as a self-stabilizing leader election algorithm until the failure of nodes and/or links increases the network's diameter. ■

4.3.6 Performance Analysis

When a distributed algorithm's performance and efficiency are concerned, parameters like message complexity, time complexity, and space complexity have to be measured.

4.3.6.1 Message Complexity

According to our system model, the nodes communicate with one other by exchanging messages, so message complexity depends on the total number of exchanged messages during the election. Here, we calculate the message complexity considering two different scenarios, i.e., without link failures and with link failures.

Best Case: If the only node with the highest leader factor among all the nodes realizes the leader failure and starts the election, then this is the best case of the propounded algorithm. If there are no link failures in the system, then the election message created by the highest leader factor's node takes $\sqrt{N}$ rounds to be broadcasted over the network. In the first round, the election initiating node creates an election message and transmits it to all the adjacent nodes. Next, the election message is spread following the message sending pattern sequence (i.e., pattern 2 – pattern 3 – pattern 1) until the message reaches every node in the system. According to the message sending pattern 2, pattern 3, and pattern 1, a node sends $2, 3$, and 1 messages, respectively. So on average, in each pattern, two messages are sent. Whenever a node gets an election message, it creates an acknowledgement and sends it to the node which sent the election message. If N is odd, the total number of exchanged messages during the election is $2[4 + 2 \times 2\{4 + 8 + 12 + ... + (\sqrt{N} - 1)/2)^{th}\ term\}] = 4N + 4$. Likewise, if N is even, the total number of exchanged messages is $2[4 + 2 \times 2\{4 + 8 + 12 + ... + ((\sqrt{N}/2) - 1)^{th}\ term\} + \{4 + 2\sqrt{N} - 6\}]$ $= 4N - 4\sqrt{N} + 4$. Hence, in the best case, message complexity is $O(N)$.

According to the Lea-TN algorithm, if a node identifies a link failure during the election, it sends a failure information message and election message to the adjacent nodes to handle the failure situation. Here, a node exchanges a maximum of 3 failure information messages and 3 election messages. Hence, after identification of a link failure, a maximum of 6 messages needs to be exchanged. The Lea-TN algorithm can tolerate a maximum of $N - 2\sqrt{N}$ link failures. A maximum of $6N - 12\sqrt{N}$ extra messages needs to be exchanged to handle the link failure situation. That means a maximum of $10N - 8\sqrt{N} + 4$ messages are exchanged to elect a leader. In this case, message complexity is also $O(N)$.

Worst Case: First, we consider that there is no link failure in the system. In this situation, after the leader crashes, if all the nodes initiate the election simultaneously, it becomes the worst-case scenario of the proposed algorithm. Here, the algorithm takes $\sqrt{N}$ rounds to complete the election. In the first round, every node creates an election message and transmits it to all the adjacent nodes. So, $4N$ messages are exchanged in the first round. From the second round, the election messages are spread following the message sending pattern sequence. In the second round, $N-1$ nodes participate in exchanging the election messages. After that, in each round the number of nodes that participate in exchanging the election messages is decreased by $4(s-2)$ (where s the round number) until it reaches round $\lceil\sqrt{N}/2\rceil$. After $\lceil\sqrt{N}/2\rceil$ rounds, in each round the number of nodes that participate in exchanging the election messages is decreased by $4(\sqrt{N}-s)$. Whenever a node gets an election message, it creates an acknowledgement and sends it to the node which sent the election message. As we discussed earlier, on average, in each pattern, two messages are sent. If N is odd, the total number of messages exchanged is $8N+4\{(N-1)+(N-5)+(N-13)+.......+((\sqrt{N}+1)/2)^{th}\ term\}+4\{4+12+24+.......+((\sqrt{N}-3)/2)^{th}\ term\}$=$(6N\sqrt{N}+19N-34\sqrt{N}-39)/3$. If N is even, the total number of messages exchanged is $8N+4\{(N-1)+(N-5)+(N-13)+.......+(\sqrt{N}/2)^{th}\ term\}+4\{5+13+25+.....+((\sqrt{N}/2)-1)^{th}\ term\}$=$(12N\sqrt{N}+39N-27\sqrt{N}-48)/6$. Thus, in the worst case, message complexity of the Lea-TN algorithm is $O(N\sqrt{N})$.

As we mentioned earlier, the Lea-TN algorithm can tolerate a maximum of $N-2\sqrt{N}$ link failures. A maximum of $6N-12\sqrt{N}$ extra messages needs to be exchanged to handle the link failure situation. So, in the worst-case scenario with link failures, message complexity is also $O(N\sqrt{N})$.

4.3.6.2 Time Complexity

Time complexity denotes the time required for completion of the election process.

Best & Worst Cases: Whenever a node starts the election or receives an election message for the first time, it sets and starts its timer for $2(T_d+3)$ rounds. After that, when this preset time ends (Session Time Out), the node can detect the newly elected leader. Here, T_d represents the diameter of the $T_(n,n)$ network. So $T_d = 2\lfloor\sqrt{N}/2\rfloor$ and $2(T_d+3) = 2(2\lfloor\sqrt{N}/2\rfloor+3)$. As our system is synchronous, there is a known upper bound of message processing delay and message transmission delay. Thus, there is an upper bound of time to complete a round. If the upper bound of completion of a round is c time units (where c is a constant), then a leader is elected within $2c(2\lfloor\sqrt{N}/2\rfloor+3)$ time units. So, $2c(2\lfloor\sqrt{N}/2\rfloor+3)$ time units are required to elect a new system leader. Hence, in the best-case and worst-case, time complexity is $O(\sqrt{N})$.

4.3.6.3 Space Complexity

The space complexity of an algorithm quantifies the amount of space or memory taken by the algorithm to run as a function of the input length. Here, we quantify the required amount of space to run the Lea-TN algorithm in a particular node. To execute this algorithm, every node stores the information of its four adjacent nodes. If there are N nodes in the system, then $O(\log N)$ bits are required to represent node Id to identify each node uniquely. So, each node needs $O(\log N)$ bits to store the information of its four adjacent nodes. On the other hand, three types of message are used in the proposed election process and $O(\log N)$ bits are required for each type of message. Hence, space complexity of the Lea-TN algorithm is $O(\log N)$.

4.3.7 Illustrative Example

To better understand the Lea-TN algorithm, we explain the election process with an illustrative example. Here, we take a 5×5 $2D$ torus network that has some link failures and node failures. A node does not know the failures of its links or adjacent nodes in advance. Figure 4.6 (a) shows the network where each rectangle represents a node, and the number inside the rectangle is the node Id. Here, the dotted orange line represents a failed link. We assume that node 21 and the former system leader node 18 have failed, and node 17 has detected the leader failure first. To make it simpler, we also assume that node 12 has the highest leader factor and that node 17 has the second highest in the same. In the first round of the election, node 17 constructs an election message and transmits it to all its adjacent nodes i.e., $12, 16, 22$ (cf. Figure 4.6 (b)). Node 17 stores its own node Id into $N\_Id\_Mlf_{17}$ and sets and starts its timer for $2(4+3) = 14$ time units (Here, we assume that a message needs one time unit to be processed and transmitted from one node to an adjacent node). When a node commences the election or gets the election message for the first time, it starts its timer for 14 time units.

In the second round, each of the nodes 12, 16, and 22 creates an acknowledgement message regarding the received election message created by node 17 and sends this acknowledgement message to node 17 (cf. Figure 4.6 (c)). Node 12 discards the received message created by node 17, because node 12 has a higher leader factor than that of node 17. Then, node 12 creates an election message and sends it to its adjacent nodes i.e., $7, 11, 13$, and 17 (cf. Figure 4.6 (c)). Node 12 stores its own node Id into $N\_Id\_Mlf_{12}$. On the other hand, after getting the election message created by node 17, nodes 16 and 22 copy the node Id 17 into $N\_Id\_Mlf_{16}$ and $N\_Id\_Mlf_{22}$, respectively. After that, node 16 and node 22 transmit the received election message (created by node 17) to nodes 11, 15, 21 and nodes 2, 21, 23, respectively. Before transmitting a received message, every node updates two message fields (i.e., Lfi and Msp) according to its link failure status. In the third round, each of the nodes 7, 11, 13, and 17 creates an acknowledgement message regarding the received election message created by node 12 and sends it to node 12 (cf. Figure 4.6 (d)). Then, node 17 updates its $N\_Id\_Mlf_{17}$ by node Id 12, and nodes $7, 11$, and 13 copy the node Id 12 into their respective $NIDMLF$ and all four nodes then transmit this message as per the message sending pattern. Likewise, each of the nodes 15, 2, and 23 creates the acknowledgement regarding the received election message created by node 17 and sends it to node 16 and node 22, respectively. At the end of this round, node 16 is supposed to get the acknowledgement message from nodes 11, 15, and 21, but it only receives this message from node 15. Node 16 can thus understand its +y and -y link failures. Likewise node 22 can understand its -x link failure. That is why in the fourth round, nodes 16 and 22 create the failure information message and send this message to the adjacent nodes that are connected through live links. When a node is informed of its link failure(s) or receives the failure information message(s), it transmits the election message (created by the node whose Id is stored in its $NIDMLF$) to the adjacent node(s) to which the message was not sent before.

When a node receives the message created by node 17 after getting the message created by node 12, it discards the latter message. However, when a node receives the message created by node 12 after getting the message created by node 17 it updates it $NIDMLF$ by node Id 12 and transmits the message (created by node 12) to the adjacent nodes as per the message sending pattern. Thus, at the end of the sixth round, the message created by node 12 finally prevails in the network and node 12 is elected as the new leader (cf. Figure 4.6 (e), (f), (g), and (h)). When a timer's preset time (here 14 time units) runs out, then the node that started the timer is informed of the newly elected leader (node 12). In Figure 4.6, the blue colored arrow indicates the transmission of the message created by node 17, and the red colored arrow indicates the transmission of the message created by node 12.

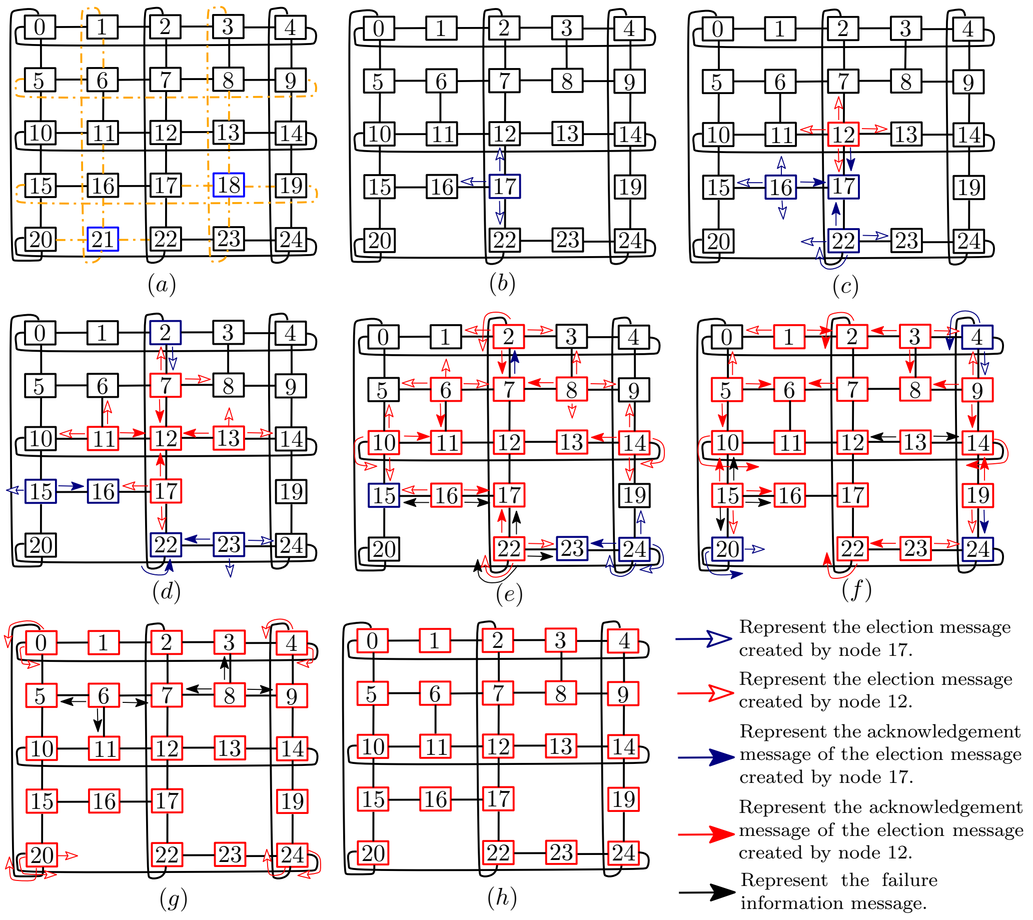

(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
Represent the election message created by node 17.
Represent the election message created by node 12.
Represent the acknowledgement message of the election message created by node 17.
Represent the acknowledgement message of the election message created by node 12.
Represent the failure information message.

FIGURE 4.6: Different steps of the election process of the Lea-TN algorithm explained on a 5×5 $2D$ torus network with some link and node failures.

4.4 Empirical Evaluation

In this section, we simulate the Lea-TN algorithm and evaluate its performance. This section also describes the experimental setup and the implementation according to our system model.

4.4.1 Experiment Setup

A single machine was used for all the experiments. The machine was equipped with an Intel (R) Core(TM) i7-3770 processor (3.40GHz, 8MB cache), 26GB DDR3 RAM, 1TB 5400rmp HDD, NVIDIA GeForce graphics, running Ubuntu Linux Release 16.04 (xenial kernel 4.4). GCC version 5.4.0 was used for the C programming environment, and MPICH version 3.2 was used for the message passing interface (MPI). Here, we consider a process as a node, and such N nodes form a cluster by connecting themselves through a $2D$ torus network. Now, for task scheduling, load balancing, and clock synchronization, a node needs to be chosen as a cluster-head (the leader). Here, we have implemented the "Lea-TN algorithm", the "LEA with One Link Failure" [94] and the "Dynamic LEA with Multi Links Failure" [8] to compare their performances for choosing the cluster-head. While simulating, we have considered different numbers of nodes such as $9, 16, 25, \ldots.., 576$ to get the number of exchanged messages and time steps required to elect the cluster-head for the cluster of different sizes. First, we simulated these algorithms without considering the link failures, and the results are shown in Figure 4.8. We then considered the link failures: the results are shown in Figure 4.9.

4.4.2 Performance Comparison and Discussion

In this section, we first compare the performance of the Lea-TN algorithm with the well-known algorithms that work on the $2D$ torus network. This comparison is made based on time consumption, message consumption and number of link failures tolerability of these algorithms. We then compare the performance of the Lea-TN algorithm with the other existing algorithms that work on different networks in a tabular form.

Only two existing leader election algorithms i.e., "LEA with One Link Failure" [94] and "Dynamic LEA with Multi Links Failure" [8] work on a $2D$ torus network. In [94] and [8], the authors claim that in the worst case, the message complexity of these algorithms is $O(N)$ and $O(N+F)$, respectively (where N is the total number of nodes and F is the number of failed links). However, we got the worst-case message complexity of these algorithms [94] [8] as $O(N\sqrt{N})$ in our calculation. In section 4.2, we proved that the lower bound message complexity (in the worst-case) of a comparison-based leader election for a $2D$ torus network is $\Omega(N \log_3 N)$. So in the worst case, the message complexity of these two algorithms [94] [8] cannot be $O(N)$ or $O(N+F)$. We shall now explain how we calculated the worst-case message complexity of these algorithms [94] [8].

Each of these two algorithms [94] [8] has four phases. In the first phase, the node that detects the leader's failure informs the other nodes in the same row of the failure by sending messages through its left and right links. In the worst case (when all the nodes detect the failure of the leader simultaneously), $2N$ messages are exchanged in this phase. In the second phase, column-wise election is started, and the node Id of the best node is stored in the node in the first row of that column. If the nodes of every column of the $2D$ torus network are arranged in decreasing order according to their Identification distinguish (Id) (cf. Figure 4.7), then at least $N\sqrt{N} + N$ messages have to be exchanged to complete the second phase of this algorithm. This is because in the second phase, $(N + \sqrt{N})/2$ election messages and $(N + \sqrt{N})/2$ acknowledgement messages need to be exchanged in order to complete the election process within each column. In a square $2D$ torus network, there are $\sqrt{N}$ columns. So, at least $\sqrt{N}(N+\sqrt{N}) = N\sqrt{N}+N$ messages have to be exchanged to complete the second phase of the algorithm. In the third phase, an election is carried out among the best nodes of each column. As the best node Ids are stored in the first row of the network, the leader can be elected by an election among the nodes of the first row by sending messages. In this phase, at least $\sqrt{N}$ election messages and $\sqrt{N}$

acknowledgement messages are required to elect the final leader of the system. In the fourth phase, N messages are exchanged to declare the elected leader. Hence, in the worst case, the algorithm has to exchange at least $N\sqrt{N} + 4N + 2\sqrt{N}$ messages to elect a new leader. Even the simulation result of these two algorithms also supports that their worst-case message complexity is $O(N\sqrt{N})$. In [94] and [8], the third phase of the election is started by the node $(0, 0)$. However, there is no clear explanation as to, if the node $(0, 0)$ crashes, which node will start the third phase. On the contrary, the Lea-TN algorithm can elect a new system leader even after the node $(0, 0)$ crashes. We also observe that the Lea-TN algorithm can tolerate multiple node failures until the node failures increase the $2D$ torus network's diameter. After analyzing the graphs in Figures 4.8 (a) and (b), we can conclude that the Lea-TN algorithm exchanges more messages than the "LEA with One Link Failure" [94] and "Dynamic LEA with Multi Links Failure" algorithms to elect a new system leader both in the best and worst case scenarios. On the other hand, the graph in Figure 4.8 (c) shows that the Lea-TN algorithm takes fewer time steps (about twice as few time steps) than these two algorithms to elect a new leader both in the best and worst case scenarios. One more significant point is that the "LEA with One Link Failure" can tolerate only a single link failure, whereas the Lea-TN algorithm can tolerate $N - 2\sqrt{N} - 3$ (if N is odd) or $N - 2\sqrt{N}$ (if N is even) link failures. In [8], the authors only mention that their algorithm can tolerate F link failures but they did not mention any relation between F and N. So it is not possible to find out how many maximum link failures the "Dynamic LEA with Multi Links Failure" [8] can tolerate. That is why it is impossible to compare the performance of the "Dynamic LEA with Multi Links Failure" and the Lea-TN algorithm considering the maximum link failure tolerability. Hence, we only simulate the Lea-TN algorithm by considering $N - 2\sqrt{N} - 3$ (if N is odd) or $N - 2\sqrt{N}$ (if N is even) link failures: the simulated results are shown in Figure 4.9. Here we consider two cases, case 1: only the node with the highest leader factor initiates the election; case 2: all the nodes initiate the election simultaneously. The graphs in Figure 4.9 (a) show the number of exchanged messages by the Lea-TN algorithm to elect a leader in case 1 and case 2. The graph in Figure 4.9 (b) shows the time required by the Lea-TN algorithm to elect a leader both in case 1 and case 2. Furthermore, we simulate the Lea-TN algorithm considering the different number of link failures: the simulated results are shown in Figure 4.10. Here, we simulate the algorithm taking six different N i.e., $N = 225$, $N = 256$, $N = 289$, $N = 324$, $N = 361$, and $N = 400$. Figure 4.10 (a) represents the simulation result where the node with the highest leader factor initiates the election, and Figure 4.10 (b) represents the simulation result where all the nodes initiate the election simultaneously.

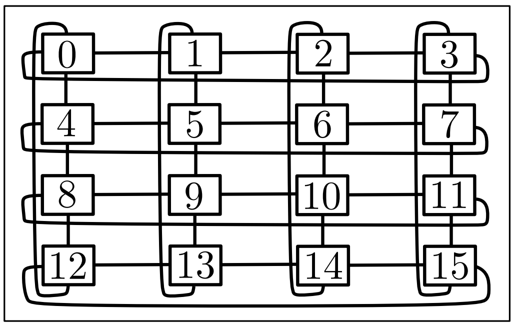


FIGURE 4.7: A $T_{4,4}$ network, every column of which is arranged in decreasing order according to the node ID.

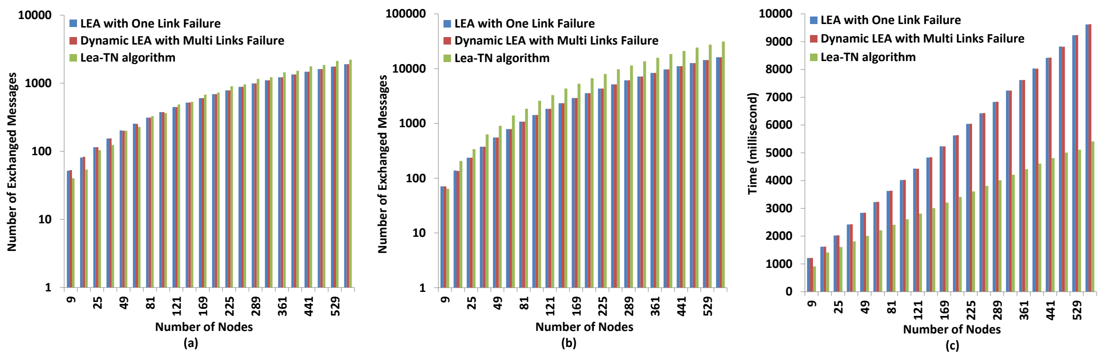


FIGURE 4.8: These figures represent the simulation results of the "LEA with One Link Failure [94]", the "Dynamic LEA with Multi Links Failure [8]" and the "Lea-TN algorithm" without considering the link failures. Figures (a) and (b) represent the graph of the number of messages exchanged in the best case and worst case, respectively, whereas (c) depicts the time taken to conduct the election.

Along with these advantages, the Lea-TN algorithm always elects a leader with a higher number of non-faulty links and a lower failure rate. The lower the failure rate of the leader, the more it is reliable, meaning that the Lea-TN algorithm provides a more reliable leader. The leader node controls and coordinates various system activities, so all other nodes need to communicate with the leader frequently. The

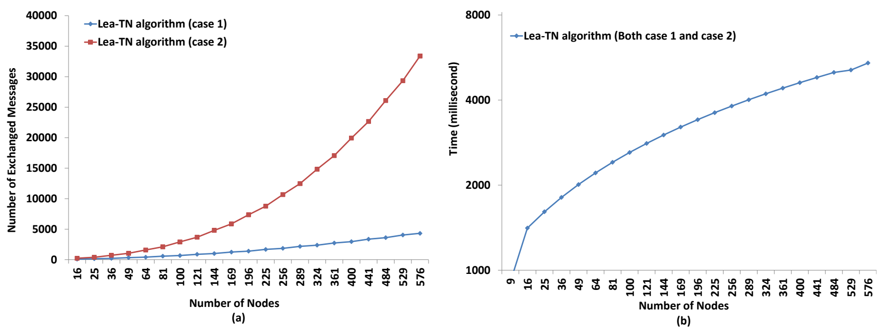


FIGURE 4.9: These figures represent the simulation results of the Lea-TN algorithm considering $N-2\sqrt{N}-3$ (if N is odd) or $N-2\sqrt{N}$ (if N is even) link failures. Here, case 1 represents the scenario where one node with the highest leader factor initiates the election, and case 2 represents the scenario where all nodes initiate the election simultaneously. Graphs in (a) represent the number of exchanged messages by the Lea-TN algorithm to elect a leader in case 1 and case 2. The graph in (b) depicts the time taken to conduct the election both in case 1 and case 2.

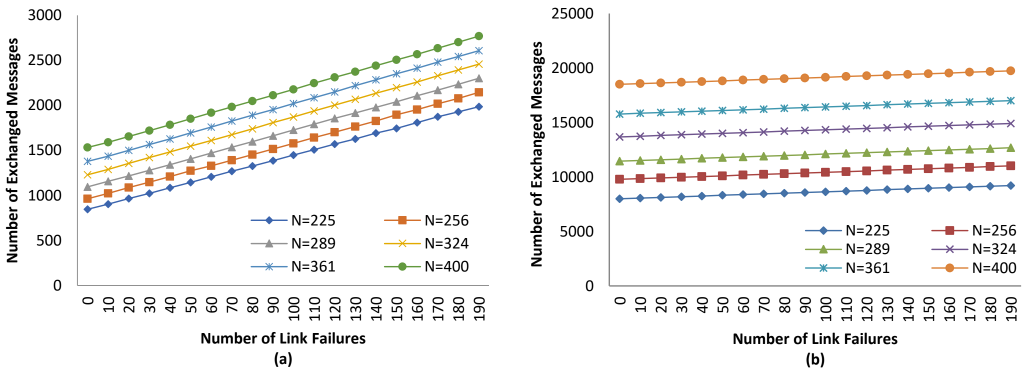


FIGURE 4.10: These figures represent the simulation results of the Lea-TN algorithm considering the different number of link failures. Here, the x-axis represents the number of link failures, and the y-axis represents the number of exchanged messages to elect the leader. In these figures, N refers to the total number of nodes in a system. Graph (a) represents the simulation result where the node with the highest leader factor initiates the election. Graph (b) represents the simulation result where all the nodes initiate the election simultaneously.

higher number of non-faulty links of the leader provides a higher number of different communication paths between the leader and the other nodes that helps balance the system's network congestion. If a system has more than one leader at any one time, it may move to an inconsistent state. So, *uniqueness* and *agreement* properties must be fulfilled by a leader election algorithm. This means that in every proper execution of the algorithm, it should elect only one node as the system leader. Since it has already been proved that the Lea-TN algorithm is a self-stabilizing algorithm, our algorithm always produces a single leader for the system that helps build a consistent distributed system.

In Table 4.2, we present a comparative study between the Lea-TN algorithm and other well-known algorithms that work on different networks. Ring [75], LCR [28], and Timer-based [19] leader election algorithms work on a unidirectional ring network. In the worst case, their message complexity is greater than that of the Lea-TN algorithm ($O(N^2) > O(N\sqrt{N})$). In the best and worst cases, the time complexity of these three algorithms is also greater than that of the Lea-TN algorithm ($O(N) > O(\sqrt{N})$). So, the proposed algorithm exchanges fewer messages and takes fewer time steps than those of these three algorithms. When we compare the Lea-TN algorithm with the HS algorithm [59], we find that in the worst case the HS algorithm exchanges fewer messages than the Lea-TN algorithm because $O(N \log N) < O(N\sqrt{N})$, but the HS algorithm takes more time steps than the Lea-TN algorithm because $O(N) > O(\sqrt{N})$. Bully, H. Abu-Amara's [3], and G. Singh's [103] algorithms work on a complete network and exchange more messages than the Lea-TN algorithm because $O(N^2) > O(N\sqrt{N})$, but Bully and G. Singh's [103] algorithms take fewer time steps than the Lea-TN because $O(1) < O(\sqrt{N})$ and $O(log^*N + 2) < O(\sqrt{N})$. In the best case, H. Abu-Amara's algorithm [3] takes fewer time steps than the Lea-TN algorithm, but in the worst case H. Abu-Amara's algorithm takes more time steps than the Lea-TN algorithm to elect a new leader because $O(N) > O(\sqrt{N})$. Among all these algorithms, the authors of the Ring, LCR, Timer-based, HS, and Bully algorithms did not provide any details of link and node failures. On the other hand, H. Abu-Amara's [3] algorithm can tolerate fewer link failures than the Lea-TN algorithm because $\lfloor (N/2) - 3 \rfloor < (N - 2\sqrt{N})$, but G. Singh's [103] algorithm can tolerate more link failures than the Lea-TN because of $(N^2/4 - N/2) > (N - 2\sqrt{N})$.

TABLE 4.2: This table represents a comparative study between the Lea-TN algorithm and the other existing algorithms that work on different networks. Comparison parameters are message complexity, time complexity, network topology, total number of links, and maximum link failure tolerability. Here, N is the total number of nodes in the system, and complexities are represented using Big O-notation.

Name of algorithm	Message complexity		Time complexity		Network topology	Total number of links	Maximum number of link failure tolerability
	Best case	Worst case	Best case	Worst case			
Ring [75]	$O(N)$	$O(N^2)$	$O(N)$	$O(N)$	Unidirectional Ring	N	-
LCR [28]	$O(N)$	$O(N^2)$	$O(N)$	$O(N)$	Unidirectional Ring	N	-
Timer Based [19]	$O(N)$	$O(N^2)$	$O(N)$	$O(N)$	Unidirectional Ring	N	-
HS [59]	$O(N)$	$O(N \log N)$	$O(N)$	$O(N)$	Bidirectional Ring	N	-
Bully [50]	$O(N)$	$O(N^2)$	$O(1)$	$O(1)$	Complete Network	$(N^2 - N)/2$	-
Modified Bully [64]	$O(N)$	$O(N^2)$	$O(1)$	$O(1)$	Complete Network	$(N^2 - N)/2$	-
H. Abu-Amara's [3]	$O(N)$	$O(N^2)$	$O(1)$	$O(N)$	Complete Network	$(N^2 - N)/2$	$\lfloor (N/2) - 3 \rfloor$
G. Singh's [103]	$O(N)$	$O(N^2)$	$O(log^* N + 2)$	$O(log^* N + 2)$	Complete Network	$(N^2 - N)/2$	$N^2/4 - N/2$
LEA with One Link Failure [94]	$O(N)$	$O(N\sqrt{N})$	$O(\sqrt{N})$	$O(\sqrt{N})$	$2D$ Torus	$2N$	One Link
Dynamic LEA with Multi Links Failure [94]	$O(N)$	$O(N\sqrt{N})$	$O(\sqrt{N})$	$O(\sqrt{N})$	$2D$ Torus	$2N$	Multiple links (does not mention the number of link failures in terms of N)
Lea-TN	$O(N)$	$O(N\sqrt{N})$	$O(\sqrt{N})$	$O(\sqrt{N})$	$2D$ Torus	$2N$	$(N - 2\sqrt{N} - 3)$ (if N is odd) or $(N - 2\sqrt{N})$ (if N is even)

4.5 Summary

Throughout this chapter, we proposed a new fault-tolerant leader election algorithm (Lea-TN) for a synchronous distributed system whose underlying network is a $2D$ torus. Here, we introduced a lower bound of message complexity, i.e., $\Omega(N \log_3 N)$ of a comparison-based leader election for a synchronous torus network. We introduced new message sending patterns that help reduce the number of exchanged messages and necessary time steps of the election process. This Lea-TN algorithm always chooses a leader with a higher number of non-faulty links and a lower failure rate. Such an elected leader is more reliable and easily accessible from the other nodes of the system. The Lea-TN algorithm fulfills three self-stabilizing conditions, i.e., uniqueness, agreement, and termination, which help in constructing a consistent system. The algorithm can also tolerate multiple link and node failures until the link and node failures increase the network's diameter. Additionally, at the end of every proper execution of this algorithm, a node gets to know about its link failures and its adjacent nodes' link failures.