Chapter 5

Preselection Based Leader Election in Distributed Systems

A distributed real-time system (DRTS) consists of autonomous computing nodes connected by a real-time network. Nodes in such a system cooperate to achieve a common goal within specified deadlines. The correctness of such (DRTS) system behavior depends not only on the logical results of the computations but also on the time when the results are produced. Missing the deadline may have disastrous consequences. Many industries use real-time systems that are distributed locally and globally. Airlines use flight control systems, Uber and Lyft use dispatch systems, manufacturing plants use automation control systems. This chapter presents a leader election method for distributed real-time systems where nodes are connected through an arbitrary network topology. The distributed real-time systems need to meet the specified deadline. So after the leader crashes, the system needs to select another leader instantly. In this chapter, we introduce the concept of the primary leader and the provisional leader that helps to an instant selection of a leader. The existing algorithms elect only one node as a leader. In contrast, the proposed algorithm identifies r comparatively higher potential leader capable nodes in the system and designates the highest potential node among them as the primary leader. The other $r-1$ higher potential leader capable nodes are kept in reserve so that while the primary leader fails, another leader capable node from these nodes can be selected instantly. To reduce the time complexity and the message complexity of the election process, based on the eccentricity of the nodes, we divide a distributed system into two layers (i.e., inner-layer and outer-layer). Only the inner-layer nodes take part to identify the list of potential nodes. We introduce the concept of the quality-coefficient to identify the potential nodes of the system. The quality-coefficient of a node is calculated by combining the eccentricity, processing capacity (Pc), Memory capacity (Mc), Degree (Deg) and Eccentricity (Etc) of the node. Our algorithm always tries to elect a node with the highest quality-coefficient as the leader so that the system gets a high quality leader. The algorithm proposed herein satisfies the *uniqueness*, *agreement*, and *termination* conditions that make it a self-stabilizing one. We also simulate the proposed algorithm on several arbitrary network topologies and compare the results with the well-known existing algorithms to evaluate the algorithm's performance.

Outline: In section 5.1, we describe our considered distributed real-time system model. Section 5.2 presents the proposed leader election algorithm. Complexity analysis and an illustrative example of the algorithm are also given in this section. Further we show that the proposed algorithm is a self-stabilizing leader election algorithm. All the experimental results and performance comparison of the proposed algorithm are given in section 5.3. Section 5.4 summarizes the work of this chapter.

5.1 System Model

We consider a distributed system comprised of N nodes connected through an arbitrary network topology for this work, where N is a finite integer and $N \geq 2$. Every node has a unique Id that helps to identify a node uniquely. The links that connect these nodes are bidirectional. The nodes communicate through message exchanging. Every node of the system has a local clock, and the local clocks of the nodes need not be synchronized. We assume that a node knows all the adjacent nodes' information.

A graph $\chi = (\Pi, L)$ represents this system; where Π is the set of nodes ($|\Pi| = N$) and L is the set of all network connectivity (links) between the nodes.

5.1.1 Assumptions

The following assumptions are considered for the work described in this chapter.

- Every node has a unique Id. For the sack of simplicity, we assume that the Id of a node belongs to 0 to $N - 1$.
- Every node knows its eccentricity.
- Every node knows the highest and lowest values of every property used to calculate the quality-coefficient.

5.1.2 Definitions

The following definitions are provided for clarity regarding their usage in the rest of this chapter.

Definition 1 ($\boldsymbol{W_{in}}$): Making use of the eccentricity [66] of the nodes, we divide the system into two layers i.e., the inner-layer and the outer-layer. W_{in} refers to the width of the inner layer, and we represent it as a mathematical function of the system's diameter. In this work, we consider $W_{in} = \lfloor \sqrt{D} \rfloor$.

Definition 2 ($\boldsymbol{ILN}$): It is the set of inner-layer nodes. Ect_i is the eccentricity of a node i. If $Ect_i < R + W_{in}$, the node i belongs to the inner-layer of the system.

Definition 3 ($\boldsymbol{OLN}$): It is the set of outer-layer nodes. If $Ect_i \geq R + W_{in}$, the node i belongs to the outer-layer of the system.

Definition 4 ($\boldsymbol{Qc_i}$): It refers to the quality-coefficient of a node i. Depending on this quality-coefficient, we identify the potential nodes in the system. In this

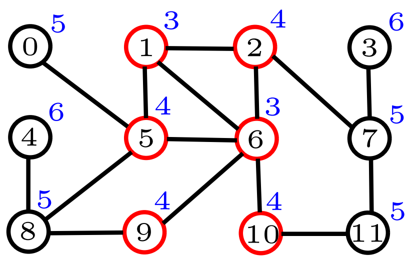

Figure 5.1: An arbitrary network consisting of 12 nodes and 15 links.

work, we consider four properties (Processing capacity (Pc), Memory capacity (Mc), Eccentricity (Ect) and Degree (Deg)) of a node to calculate its Qc_i. A node with higher processing capacity, higher memory, higher degree, and lower eccentricity is considered a higher potential node. Measurement units of all these properties are different. That is why we normalize them using linear max-min data normalization method [112] and calculate the quality-coefficient using equation 5.1. Here, we assume that every node knows the highest and lowest values of every property. If X is a property, then $max(X)$ and $min(X)$ refers to the highest and lowest values of X, respectively and w_X is the weight of X. The weight is used to prioritize the properties. For example, in equation 5.1, $max(Pc)$ and $min(Pc)$ refers to the highest and lowest values of the processing capacity and w_{Pc} is the weight of the processing capacity. In this equation, $w_{Pc}, w_{Mc}, w_{Deg}, w_{Ect} \geq 0$ and $w_{Pc}+w_{Mc}+w_{Deg}+w_{Ect} = 1$.

$$Qc_i = w_{Pc}\frac{Pc_i - min(Pc)}{max(Pc) - min(Pc)} + w_{Mc}\frac{Mc_i - min(Mc)}{max(Mc) - min(Mc)} + w_{Deg}\frac{Deg_i - min(Deg)}{max(Deg) - min(Deg)} + w_{Ect}\frac{max(Ect) - Ect_i}{max(Ect) - min(Ect)} \quad (5.1)$$

5.1.3 Types of Message

In this work, we use three types of messages to elect a new leader. They are leader crashed information message ($Leader\_crash[Fl\_Id]$), election message ($Emsg[Emc\_Id, Fl\_Id, Qc_c]$), and new-leader declaration message ($New\_leader[El\_Id, Np\_List]$). When an outer layer node realizes the leader failure, the node creates a $Leader\_crash[Fl\_Id]$ message and sends it to a nearest inner layer node to inform the inner layer node that the leader has crashed. This message has only one field (i.e., Fl_Id). On the other hand, the election message ($Emsg[Emc\_Id, Fl\_Id, Qc_c]$) is used to initiate the election and make the list of potential nodes through the election. It has three fields (i.e., Emc_Id, Fl_Id, and Qc_c). New-leader declaration message ($New\_leader[El\_Id, Tl, Np\_List]$) is used for declaration of a new leader for the system. It has three fields (i.e., El_Id, Tl and Np_List).

5.2 Proposed Leader Election Algorithm

Unlike the existing algorithm, instead of electing one node, the proposed algorithm identifies r potential leader capable nodes and designates the highest potential node among them as the leader. Later on, when the system leader crashes, rather than running the whole election algorithm, instantly, the system can choose the next leading alive node from the list of potential leader capable nodes as the provisional leader. So the tasks that need coordination do not get halted for a long time for the lack of a coordinator. As a result, the overall system performance gets increased. Afterwards, the system elects the primary leader invoking the entire election algorithm as per its convenience. In the proposed election method, based on the eccentricity of the nodes, we divide the network into two layers, i.e., inner-layer and outer-layer. Only the inner-layer nodes participate in identifying the r potential nodes among them. The proposed algorithm identifies r nodes as the list of potential nodes with the higher quality-coefficient from all the inner layer nodes. In this algorithm we a function i.e., $Create\_Emsg()$. The definition of this function is as follows.

***Create_Emsg*()**

1. $Emc\_Id \leftarrow NId_i$
2. $Qc_c \leftarrow Qc_i$
3. Create an election message $Emsg[Emc\_Id, Fl\_Id, Qc_c]$, and send it to all the adjacent inner layer nodes.

Election initiation: This phase illustrates how an election is initiated in the proposed election method. Algorithm 8 describes the election initiation steps. The leader failure detection can be done by the outer-layer node or by the inner-layer node. When a node i perceives the leader's failure, it removes the crashed leader's information from its P_List_i. Then it checks its P_List_i and performs the following action.

Algorithm 8: Election Initiation

```
// When a node $i$ perceives the leader's failure and initiates the election.
if (the node $i$ realizes the leader's failure) then
    Remove $Fl\_Id$ from the $P\_List_i$ list.
    if ($P\_List_i == Empty$) then
        if (node $i \in OLN$) then
            Create a $Leader\_crash[Fl\_Id]$ message, send it to a nearest inner layer node and start timer for
              $2T_f$ unit time.
        else
            $Create\_Emsg()$.
            Append $NId_i$ and $Qc_i$ into the $P\_List_i$, start the timer for $2T_{in}$ unit time
        end
    else
        Choose the highest potential alive node from the $P\_List_i$ list as the provisional system leader,
          $El\_Id \leftarrow$ the new system leader Id, $L\_Id_i \leftarrow El\_Id$
        $Tl \leftarrow 1$, $Pl_i \leftarrow 1$, $Np\_List \leftarrow P\_List_i$, create a $New\_leader[El\_Id, Tl, Np\_List]$ message and
          broadcast it over the network.
    end
end
```

1. If the P_List_i is empty, and the node i is an outer-layer node, then the node i creates a $Leader\_crash[Fl\_Id]$ message and sends it to the nearest inner-layer node to inform about the leader's failure and starts its timer for $2T_f$ unit time.

2. If the P_List_i is empty, and the node i is an inner layer node or it gets a $Leader\_crash[Fl\_Id]$ message, then the node i creates an $Emsg[Emc\_Id, Fl\_Id, Qc_c]$ message and initiates an election by sending it to all the adjacent inner layer nodes. After that It appends self information (NId_i and Qc_i) into the P_List_i and starts timer for $2T_{in}$ unit time.

3. If the P_List_i is not empty, the node i selects the highest potential alive node from the P_List_i list and declares it as the provisional system leader by creating and broadcasting the $New\_leader[El\_Id, Tl, Np\_List]$ message over the network.

Finding potential nodes: This phase explains how the proposed election method identifies the r potential nodes and lists them. Algorithm 9 describes the steps of finding the higher quality-coefficient nodes. Suppose a node j gets a $Emsg[Emc\_Id, Fl\_Id, Qc_c]$ message, then it performs the following actions.

1. If a node j gets the election messages for the first time, the node j makes its P_List_j empty. Then it puts the node Id and quality-coefficient received through the message in its P_List_j and sends the received election message to all the adjacent inner layer nodes except the election message sender node. If node j is not an election initiating

Algorithm 9: Potential node finding

```
// When a node $j$ receives an election messages
   ($Emsg[Emc\_Id, Fl\_Id, Qc_c]$).
1  if (Node j gets an election messages for the first time) then
2      Make the $P\_List_j$ empty.
3      Put the node Id and quality-coefficient received through the message in the
         $P\_List_j$.
4      Send the received message to all the adjacent inner layer nodes except its sender.
5      if (Node j is not an election initiating node) then
6          Place the self Id and self quality-coefficient in the $P\_List_j$ in such a way so
             that the $P\_List_j$ gets arranged in descending order according to the
             quality-coefficient.
7          $Create\_Emsg()$, start the timer for $2T_{in}$ unit time.
8      end
9  else
10     if (Node j already got the same election message.) then
11         Discard the received election message ($Emsg[Emc\_Id, Fl\_Id, Qc_c]$)
12     else
13         if ($P\_List_j == Full$) then
14             if ($Qc_c$ is less than the lowest quality-coefficient in $P\_List_j$) then
15                 Discard the received election message ($Emsg[Emc\_Id, Fl\_Id, Qc_c]$)
16             else
17                 Remove the entry with the lowest quality-coefficient from the $P\_List_j$.
18                 Place the node Id and quality-coefficient in the correct position in the
                     $P\_List_j$ so that the order gets maintained.
19                 Send the received election message to all the adjacent inner layer nodes
                     except its sender.
20             end
21         else
22             Place the node Id and quality-coefficient received through the message in
                 the correct position in the $P\_List_j$ so that the order gets maintained.
23         end
24     end
25 end
```

node, it places self Id and self quality-coefficient in the P_List_j in such a way so that the entries in the P_List_j get arranged in descending order according to the quality-coefficient. It creates an election message with its self information, sends it to all the adjacent inner-layer nodes and starts its timer for $2T_{in}$ unit time.

2. When node j gets further election messages, it checks whether it has already got the same election message. If it already got the same election message, it discards the message. Otherwise it checks whether its P_List_j is full. If P_List_j is full,

node j compares the Qc_c (received by the election message) with the lowest quality-coefficient in the P_List_j. If Qc_c is less than the lowest quality-coefficient in the P_List_j, the node j discards the received election message. Otherwise, it removes the entry with the lowest quality-coefficient from the P_List_j, places the node Id and quality-coefficient received through the election message in the correct position in the P_List_j so that the order gets maintained. Then it sends the received election message to all the adjacent inner layer nodes except the sender of the election message. On the other hand, if P_List_j is not full, node j places the node Id and quality-coefficient received through the election message in the correct position in the P_List_j. In the case of any tie or symmetry, node Id is considered to break the tie or symmetry, and the higher node Id gets the preference.

Leader declaration: In this phase, the inner layer nodes declare the newly elected leader by creating a new-leader declaration message ($New\_leader[El\_Id, Tl, Np\_List]$) and sending it to all other nodes of the system. Algorithm 10 describes the newly elected leader declaration steps.

1. After the timer's timeout, an inner layer node i selects the node with the highest quality-coefficient from its P_List_i as the new system leader. Next, the node i creates a $New\_leader[El\_Id, Tl, Np\_List]$ message (where, El_Id contains the newly elected leader Id, $Tl = 0$ and Np_List contains the r potential node Ids), and sends it to all the adjacent outer layer nodes.

2. When a node j gets a $New\_leader[El\_Id, Tl, Np\_List]$ message, it considers El_Id as the new leader, copies Np_List into its P_List_j and sends the received message to all the adjacent outer layer nodes. If node j further gets the $New\_leader[El\_Id, Tl, Np\_List]$ message it discards the message.

5.2.1 Illustrative Example

This section explains the proposed algorithm with the help of an example. Suppose, a distributed system consists of 12 nodes connected through an arbitrary network topology

Algorithm 10: Elected leader declaration

```
// After the time out of a node $i$.
1  if (timer of node $i$ == timeout) then
2      if (node $i \in ILN$) then
3          Choose the node with the highest quality-coefficient from the $P\_List_i$ as the
           new system leader, $El\_Id \leftarrow$ the new system leader Id.
4          $L\_Id_i \leftarrow El\_Id$, $Tl \leftarrow 0$, $Np\_List \leftarrow P\_List_i$, create a
           $New\_leader[El\_Id, Tl, Np\_List]$ message and send it to all the adjacent outer
           layer nodes.
5      else
6          if (node $i$ did not get any new-leader declaration message) then
7              Re-initiate the election.
8          end
9      end
10 end
// Suppose, a node $j$ gets an $New\_leader[El\_Id, Tl, Np\_List]$.
11 if (node $j$ gets a $New\_leader[El\_Id, Tl, Np\_List]$ message for the first time) then
12     $L\_Id_i \leftarrow El\_Id$, sends the $New\_leader[El\_Id, Tl, Np\_List]$ message to all the
       adjacent outer layer nodes except its sender
13 else
14     Discard the message
15 end
```

(cf. Figure 5.2 (a)). In Figure 5.2, a circle represents a node, and the integer inside the circle refers to the node Id. The blue-colored integer outside the circle is the eccentricity of a node. The diameter and radius of the network are 6 and 3, respectively and the width of the inner layer is 2. All the red-colored nodes belong to the inner layer, and the remaining nodes belong to the outer layer. Other details of the nodes are given in Table 5.1. In this example, the minimum and maximum processing capacity, memory capacity, eccentricity and degree are 1 and 6, 4 and 24, 1 and 11, and 1 and 11 respectively. We want to identify 3 potential nodes through the proposed election method. That means, here $r = 3$ and the P_List_i can contain information of at most three nodes. In this example, we assume that a message takes one unit of time to get transmitted from one node to its adjacent node. Suppose, node 12 was the system leader. When node 12 crashes, node 10 realizes it and initiates the election creating the $Emsg[10, 12, 0.48]$ message (as node 12 is crashed, we did not show it in Figure 5.2). Then it puts self Id (10) and self quality-coefficient (0.48) into the P_List_{10}, sends the $Emsg[10, 12, 0.48]$ to node 6 and starts timer for 4 time units. Node 6 gets the $Emsg[10, 12, 0.48]$ message and it is its first received election message, so it places self Id and Id 10 and their corresponding quality-coefficients in the first and second position of the P_List_6 respectively. Then node 6 sends the $Emsg[10, 12, 0.48]$ to nodes $1, 2, 5$ and 9. Afterwards, node 6 creates the $Emsg[6, 12, 0.61]$ message, sends it to nodes $1, 2, 5, 9$ and 10 and starts the timer for 4 time unit. When the nodes $1, 2, 5$ and 9 get the $Emsg[10, 12, 0.48]$ and $Emsg[6, 12, 0.61]$ message, they store Ids 6, 10 and self Id and their corresponding quality-coefficients into their respective $PNList$ according to the descending order of the quality-coefficient. Each of nodes $1, 2, 5$ and 9 will send the $Emsg[6, 12, 0.61]$ message to its all adjacent inner layer nodes except node 6. Afterwards, each of them creates the election message using self information and sends the same to each one's all adjacent inner layer nodes and start the timer for 4 time unit. On the other hand, when node 10 gets the $Emsg[6, 12, 0.61]$ message, it places node Id 6 and the quality-coefficient of node 6 (0.61) in the correct position (first position) in its P_List_{10}. Next, when node 6 gets election messages created by nodes $1, 2, 5$ and 9, it sends the election messages created by nodes 1 to nodes $2, 5, 9$ and 10, and the election messages created by 5 to nodes $1, 2, 9$ and 10. In this way, all the inner layer nodes gets the election messages created by nodes $6, 1$ and 5 and identify nodes $6, 1$ and 5 as the three most potential nodes of the system and select node 6 as the new system leader. After the timer's time-out of an inner layer node, the node creates a new-leader declaration message and sends it to its adjacent outer layer nodes to inform the outer layer nodes about the new system leader and the identified potential leader capable nodes.

TABLE 5.1: Details of the arbitrary network shown in Figure 5.2 (a)

Node Id	Processing capacity	Memory capacity	Eccentricity	Degree	Layer of the node	Quality-coefficient
0	2.8	20	5	1	Outer	0.44
1	4.1	16	3	3	Inner	0.55
2	3.6	12	4	3	Inner	0.45
3	4.2	16	6	1	Outer	0.43
4	3.2	12	6	1	Outer	0.33
5	3.9	12	4	4	Inner	0.49
6	4.3	16	3	5	Inner	0.61
7	4.3	12	5	3	Outer	0.46
8	2.9	8	5	3	Outer	0.34
9	4.0	8	4	2	Inner	0.40
10	3.6	16	4	2	Inner	0.48
11	4.5	12	5	2	Outer	0.45

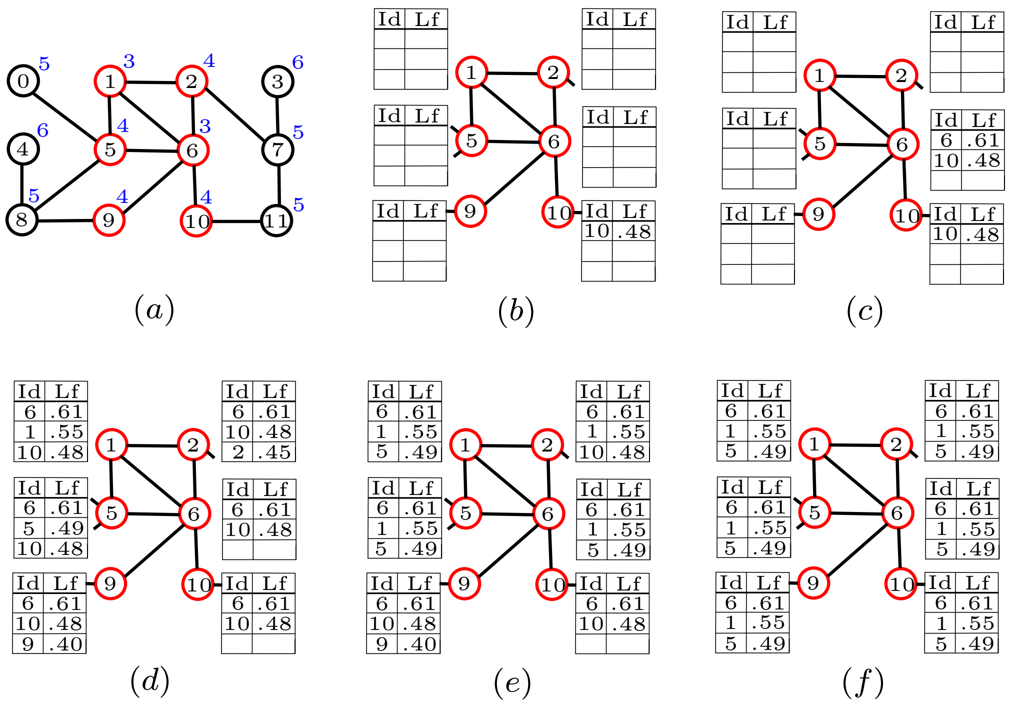

FIGURE 5.2: (a) is an arbitrary network consisting of 12 nodes and 15 links. (b), (c), (d), (e) and (f) are the several steps of the proposed algorithm to identify the higher potential nodes of the inner layer.

5.2.2 Complexity Analysis

In this section, we calculate the time and message complexity of the proposed algorithm.

5.2.2.1 Message complexity

According to the system model, the nodes communicate through message exchanging, so the message complexity of the algorithm depends on the total number of exchanged messages during the election.

Best case: When the primary leader crashes and a node is selected as the provisional leader from the list of potential nodes. It is the best-case scenario of the proposed algorithm. In the best case, only one node realizes the leader's failure and selects the highest potential alive node from the list as the provisional leader, and declares the newly selected leader by creating and broadcasting a new-leader declaration message. Hence, in the best case, the message complexity of the proposed algorithm is $O(|L|)$, where L is the set of all network connectivity.

Worst case: When the list of r potential nodes needs to be identified through the election, it becomes the worst case of our algorithm. In this case, at a time, all nodes realize the leader's failure and initiate the election. So the outer layer nodes create $O(|OLN|)$ leader crashed messages. Furthermore, the inner layer nodes exchange $O(r \cdot |L_{in}|)$ (where L_{in} is the set of all network connectivity in the inner layer.) election messages to identify the new r higher potential nodes of the system. After that, the highest prominent node from the list is selected and declared as the new leader. The algorithm exchanges $O(|L_{out}|)$ (where L_{out} is the set of all network connectivity of the outer-layer.) messages for the declaration of the new leader. Hence, in the worst case, the message complexity is $O(|OLN|+r\cdot|L_{in}|+|L_{out}|)$.

5.2.2.2 Time complexity

In the case of leader election, time complexity refers to the time steps required to elect a new system leader. Here, we assume that a message takes c unit time to travel from one node to its adjacent node, where c is a constant.

Best Case: In the best case, the algorithm takes a maximum of $D \cdot c$ unit time to select and declare the provisional leader. So, in the best case, the time complexity is $O(D)$.

Worst case: In the worst case, the algorithm takes a maximum of $2D \cdot c$ unit time to identify the r higher potential nodes and declare the highest potential node as the system leader. Hence, in the worst case, the time complexity is also $O(D)$.

5.2.3 Proof of Self-stabilization

Uniqueness, agreement and termination are three necessary conditions of the self-stabilizing leader election algorithm. Now, we will show that our algorithm satisfies these three conditions.

Lemma 5.1. *Every proper execution of the proposed algorithm elects only one node as the system leader.*

Proof: When an election is conducted to identify the r potential leader capable nodes from the inner layer nodes, the election messages created by the r nodes with the higher quality-coefficients get broadcast in the inner layer of the network. Thus all the inner layer nodes get to know about the r nodes with comparatively higher quality-coefficients and elect the highest quality-coefficient node as the leader. If every inner layer node has a distinct quality-coefficient, then the node with the highest quality-coefficient among them is elected. On the other hand, if multiple nodes have the highest quality-coefficient, then the node Id is used to break the symmetry, and the node with the highest Id among them gets the preference to become the system leader. Every node has a unique Id, so the algorithm always selects only one node as the system leader.

Lemma 5.2. *In every proper execution of the proposed algorithm, all the nodes of the system get to know about the elected leader and agree with it.*

Proof: During the election, the election messages created by the r nodes with the comparatively higher quality-coefficients get broadcast in the inner layer of the network. Thus all the inner layer nodes get to know about the r nodes that have the higher quality-coefficients and the elected leader. The last phase of our algorithm is the leader declaration phase. In this phase, a leader declaration message informs all the outer layer nodes about the newly elected leader. So at the end of the election, all the system nodes must know about the newly elected system leader.

Lemma 5.3. *The proposed algorithm is terminated in a finite time.*

Proof: The propounded algorithm takes a maximum of $2D$ hops time to select the r most prominent nodes and elects a leader. According to the system model, a message gets exchanged between two nodes within a bounded time. Suppose a message takes c unit of time to get exchanged from one node to its adjacent node, where c is constant. After the time out of the timer, a node gets to know about the selected list of r number of potential nodes and the newly elected leader. So a node terminates the execution of the election process by $2 \cdot D \cdot c$ unit of time. The system consists of a finite number of nodes (N), so the diameter D of the network is also finite. Hence the proposed algorithm gets terminated in a finite time and elects a leader for the system.

Theorem 5.4. *The proposed algorithm is a self-stabilizing leader election algorithm.*

Proof: Lemma 5.1, Lemma 5.2 and Lemma 5.3 prove that the proposed algorithm satisfies the uniqueness, agreement and termination conditions respectively. That means the algorithm satisfies all the three conditions of a self-stabilizing leader election algorithm. Hence, the proposed algorithm is a self-stabilizing leader election algorithm. ■

5.3 Experimental Analysis and Discussion

The network layering concept of the proposed election method helps to reduce the message complexity and time complexity of the election process. Instead of all the nodes, only the inner layer's nodes directly participate in the election in identifying the list of potential nodes. That is why during the election, the required number of exchanged messages and time steps get reduced. The existing algorithms elect one node as the leader, whereas the proposed algorithm identifies the list of r most potential leader capable nodes and designates the highest potential node as the primary leader. When the system leader crashes, for the lack of a coordinator, the coordination-based tasks of the system get halted until a new leader is elected. According to the proposed election method, whenever a primary leader fails, instantly, the system can choose a provisional leader from the list so that due to the lack of coordination, the system's work does not get halted for a long time. This provisional leader controls and coordinates the system until the primary leader is being elected by the next election. The provisional leader makes the system more flexible because the system can wait for a suitable time to conduct a new election without hampering system performance. Thus The identification of r potential nodes helps to improve the system performance. Identification of the potential nodes and selecting one of them as the leader is made based on the quality-coefficient. The quality-coefficient of a node is calculated considering the processing capacity, memory capacity, eccentricity and degree of the node. A node gets elected as the leader with comparatively higher processing capacity, higher memory capacity, higher degree, and lower eccentricity in the proposed election method. A higher processing and memory capacity leader can perform its leadership tasks faster, which improves the overall system performance. On the other hand, the leader's higher degree and lower eccentricity reduce network traffic and help in faster communication between the leader and other nodes. In the proposed method, the width of the inner layer is changeable. It is favorable to keep the size of the inner layer small, but if required, we can change the width of the inner layer (W_{in}) by taking a different mathematical function of D so that the election algorithm can always select a good quality leader for the system.

Table 5.2: Details of the different arbitrary networks for the simulation.

Sl. no.	Total no. of nodes	No. of inner layer nodes	No. of outer layer nodes	Total no. of edges	Diameter
Network 1	12	6	6	15	6
Network 2	20	8	12	29	8
Network 3	40	14	26	56	12
Network 4	60	21	39	81	17
Network 5	80	28	52	106	21
Network 6	100	32	68	127	25

We simulate the proposed leader election algorithm on six different arbitrary network topologies. We get to know the number of exchanged messages and time steps to elect the leader in the best and worst-case scenario through this simulation. These arbitrary networks were constructed by the different number of nodes and links. In our experiment, we consider $r = 5$. The networks details are given in Table 5.2 and the simulation results are shown in Figures 5.3 and 5.4. All experiments were performed on a single machine. The machine was decorated with Intel core(TM)i5-2410M processor (2.3GHz, 4MB cache, 2.9GHz Turbo Boost), 8GB RAM, 1TB HDD, NVIDIA GeForce graphics, running Ubuntu Linux Release 16.04 (xenial kernel 4.4). We used *python 3.6* as the programming language and *mpi4py* as the message passing interface (MPI).

We compare the proposed algorithm with three well-known and highly cited algorithms known as Mega-Merger [49] [100], Yo-yo [100], and Vasudevan's algorithm [114]. This comparison is made in terms of the requirement of the total exchanged messages and total time to complete the election process, and shown in Figure 5.3 and Figure 5.4. Figure 5.3 (a) and (b) show the number of exchanged messages in the best case and worst case scenario, respectively. These two graphs show that the proposed algorithm exchanged fewer messages than Mega-Merger, Yo-yo, and Vasudevan's algorithm to elect the leader. On the other hand, Figure 5.4 (a) and (b) show the time to elect the leader in the best case and worst case scenario, respectively. Here it is clear that the proposed algorithm takes

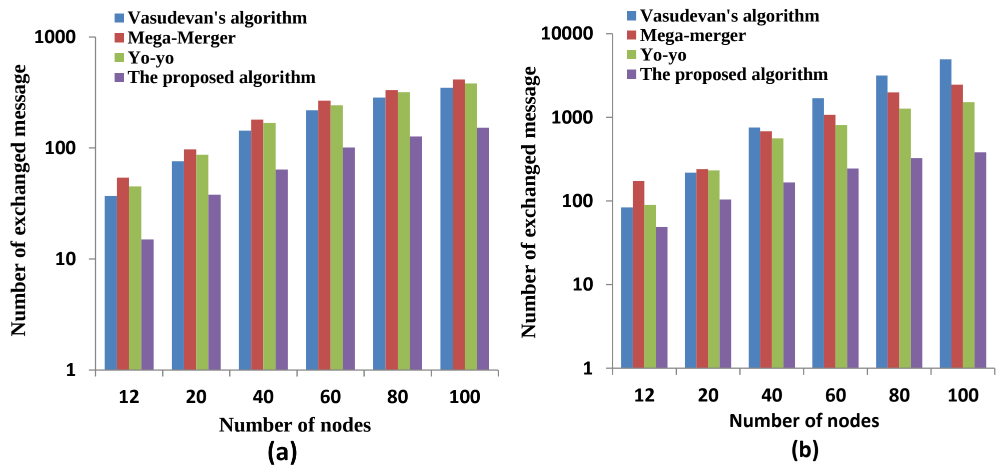

FIGURE 5.3: The total number of exchanged messages required by different algorithms to complete the election process. (a) and (b) represent the number of exchanged messages in the best case and worst case, respectively. (c) and (d) represent the required time in the best case and worst case, respectively

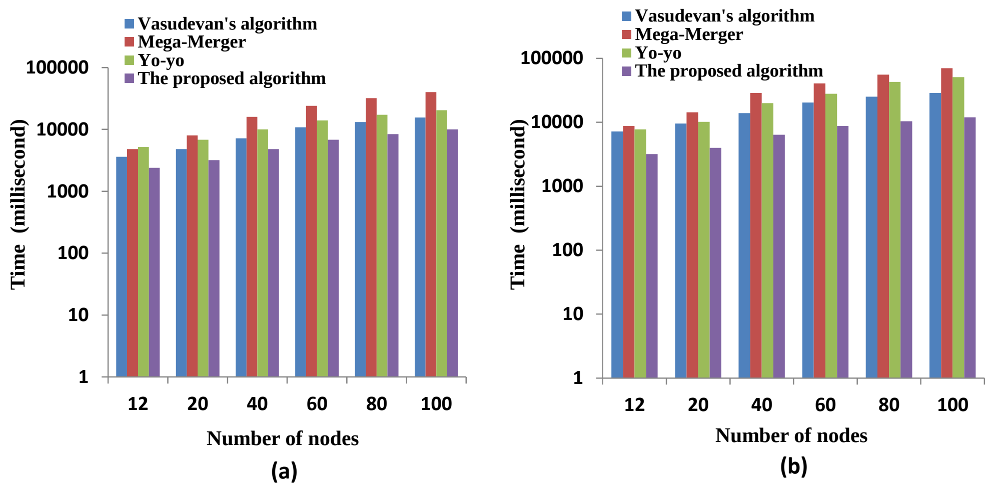

FIGURE 5.4: The total time required by different algorithms to complete the election process. (a) and (b) represent the required time in the best case and worst case, respectively

less time than Mega-Merger, Yo-yo, and Vasudevan's algorithm to elect the leader. After the simulation, we get to know a limitation of the proposed algorithm. In some cases, it may happen that after the primary leader crashes, the next chosen provisional leader from the list might be slightly inferior because of the loss associated with the primary leader crash in the form of links failure.

5.4 Summary

This chapter presented a new approach of leader election for a distributed real-time system consisting of N nodes connected through an arbitrary network topology. We introduce the concept of a provisional leader that helps to reduce system performance degradation during the election. The algorithm can identify a list of some most potential leader nodes and designate the best one from the list as the primary leader initially. After that, when the primary system leader crashes instantly, the next prominent node of the list is selected as the provisional leader. This provisional leader controls and coordinates the system until the next election is electing the primary leader. That means the provisional leader makes the system more flexible because the system can wait for a suitable time to conduct a new election without hampering system performance. On the other hand, the concept of layering the system helps to reduce the required number of exchanged messages and time steps to elect a system leader. Calculating the quality-coefficient of a node considering its processing capacity, memory capacity, eccentricity, and degree of a node and electing the leader based on the quality-coefficient always helps select one of the best nodes as the leader.