International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 06, June 2026)

A Quantum Leap: A Novel Algorithm for Leader Election in Distributed Environment

Abhishek Dutta[1], Moumita Sarkar[2]
[1]*Assistant Teacher, Assembly of GOD Church School*
[2]*Research Scholar, India*

***Abstract*—Leader election plays a perilous role for guarantee- ing coordination among processes in Distributed System. Widely used traditional algorithms are often suffer from high message complexity and delayed leader election which yield more compli- cations in handling letdown of existing leader in heterogeneous networks. These inadequacies lead to network congestion and augmented latencies. This paper proposes a novel leader election algorithm that reduces message overhead, improves response time which in turn ensures consistent synchronization in dynamic distributed system.**

***Index Terms*—Leader Election, Election Algorithm, Message Complexity, Time Complexity, Distributed System**

I. Introduction

In large scale distributed system, the most observed challenges in traditional leader election strategies are message flooding, delay in election and congestion. We will propose an efficient leader election algorithm which is capable enough to reduce the unnecessary message passing and prolonged election time for smooth coordination by allowing only one process to proceed finally with the election when a failure is detected by supressing others. This streamlined process significantly reduces message traffic and minimizes delays. Additionally, a hierarchical acknowledgment system quickly confirms the newly elected leader, further enhancing efficiency. By minimizing redundant messages and ensuring a fixed, rapid interaction pattern, our algorithm proves highly scalable and efficient, outperforming traditional methods in both speed and reliability.

II. State Of The Art

Some leader election algorithm considers fully connected network, while others consider ring topology or spanning tree structure to work upon [1]–[3]. In [1], Bully algorithm was proposed where election is started by notifying processes having higher id than the process who detected the failure. In Ring Algorithm [2], processes are organized in a logical ring. Every process sends an election message along with its ID to the subsequent process, and after the message circulates the entire ring, the process having the highest ID becomes the leader.

In [4], a variation of [2] is proposed where every process sends its id to the next process and the highest id subsist would become the new leader. The algorithms proposed in [1], [2], and [4] all have $O(n)$ time complexity and $O(n^2)$ message passing complexity. [5], accounts three very popular leader election algorithms: HS (Hirschberg-Sinclair) uses bidirectional forwarding strategy where processes send election messages both the way in logical network, with each process forwarding a leader election message if it finds a higher ID in both directions, ultimately electing the highest ID as the leader. Here message passing complexity is $O(n \log n)$ and time complexity is $O(\log n)$; TimeSlice Algorithm, chosen for systems where processes run at varying speeds, assigning fixed time slices for processes to check if they are still active with a leader determined based on the allotted time slots; and Variable Speeds Algorithm, where processes run at different, perhaps unidentified speeds, and leader election regulates animatedly by comparing process speeds or altering timeouts based on perceived speed differences, electing the fastest or most proficient process. Both TimeSlice and Variable Speeds algorithms encompasses the complexity in order of $O(n)$ in case of message passing as well as time. A sorting-based message exchange with $O(n \log n)$ complexity was proposed in [6] who had introduced a queued linked list based approach with the additional overhead of maintaining the linked list. In [7], a timer-based leader election algorithm using a ring structure was proposed, with a best and average case message passing complexity of $O(n)$, a worst-case message passing complexity of $O(n^2)$, and a time complexity of $O(n)$ in all cases where the author treated each node equally capable to be the leader. To reduce the message and time complexity,[8] proposed an approach to select the highest potential leader among preselected potential leaders.

III. Problem Statement

Leader election in distributed systems is key for coordination, but traditional algorithms face issues like high message complexity, network congestion, and delays, especially in large networks.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 06, June 2026)

In case of node failures and huge amount of message passing before electing a new leader, leading to inefficiency. A better algorithm should reduce message overhead, improve response times, handle failures effectively.

IV. Problem Formulation

In a distributed system modeled as a directed graph G = (V, E), with n processes and directed communication links E, the goal is to elect a unique leader for coordination. The election must guarantee legitimacy (leader should be the will- ing process), convergence (one leader only) and termination. The algorithm must circulate the message along the edges of the graph and be efficient enough to handle failure of current leader without any ambiguity.

V. Proposed Leader Election Algorithm

A. Proposed Algorithm

In a distributed system modeled as a directed graph G = (V, E), with n processes P_i (where $i \in [0, n-1]$) and unidirectional edges to P_{i+1}, the goal is to elect a coordinator for resource management. When a process detects a leader failure, it sends an "ELECTION" message to the next process, including the failed ID. Processes decide to join the candidate list or forward the message. The algorithm minimizes message complexity to O(m) and ensures timely completion in O(t), maintaining resilience against process failures.

B. Assumptions

1) Each process in the distributed system is uniquely identified by an integer process ID, ID_i, where $i \in \{0, 1, 2, \ldots, n-1\}$ and n represents the total number of processes in the structure.
2) The structure allows for directed message passing; if a message is sent from process P_i, it is received by process P_{i+1}.
3) Faulty processes, defined as those that do not respond within a specified timeout period $T_{timeout}$, that if process P_j is recognized as faulty, the message will be delivered to P_{j+1} instead.
4) Not always the process with highest process id is declared as leader ; thus, it is possible that $ID_{coordinator} < ID_j$ for some j.
5) Upon receiving an election message, a process P_j may add itself to the candidate list or forward the re- ceived message to the next process without announcing itself as a coordinator.
6) The communication channels between all processes are fully reliable, ensuring that any message sent is guaranteed to be received accurately, allowing us to express this as $P_i \xrightarrow{\text{Message}} P_j$, where message delivery is confirmed.

C. Notation and Description

- n: total processes in the distributed system.
- P_i: ith process in the distributed system, where i in range 0 to n −1.
- ID_i: inimitable integer ID allotted to process P_i.
- $T_{timeout}$: stipulated duration of process response; would be considered faulty if it fails.
- m: total number of messages exchanged throughout the leader election method.
- $P_{(i+1)}$: next process to P_i.
- $P_{(i-1)}$: previous process to P_i.
- "ELECTION" message: sent by a process to initiate the leader election process.
- "ACKNOWLEDGEMENT" message: sent by processes to confirm acceptance of election message.
- "LEADER" message: sent by the elected leader to an- nounce its status to other processes.
- "SUPPRESSION" message: sent to inform a lower ID process that its election message has been suppressed and will not be forwarded.

The proposed algorithm is designed to elect a new leader when the current leader fails in a distributed system. Each process maintains a participation flag, a candidate list, the failed process identifier, and a timeout value during initialization. When a process detects a leader failure, it initiates an election by sending an ELECTION message to the next process. Upon receiving the ELECTION message, eligible processes add their identifiers to the candidate list and forward the message to continue the election process. A suppression mechanism is used to eliminate redundant election requests by sending a SUPPRESSION message when duplicate election messages are received from lower-identifier processes. After the election phase, the process with the highest identifier that receives no further election messages is selected as the new leader and sends ACKNOWLEDGEMENT messages. Finally, the elected leader broadcasts a LEADER message to all active processes to inform them about the new coordinator.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 06, June 2026)

Algorithm 1 Leader Election Algorithm

1: **Input:** Process IDs, failed process ID
2: **Output:** Elected leader
3: **Initialization:**
4: **for** each process P_i **do**
5: flag ← 0 ▷ Flag for election participation 6: candidates ← [] ▷ List of candidate process IDs
7: failedID ← None ▷ ID of failed process
8: timeout ← T ▷ Timeout for acknowledgements
9: **end for**
10: **Detection of Leader Failure:**
11: **if** process P_i detects failure of the current leader **then**
12: Initiate an election
13: **end if**
14: **Election Initiation:**
15: Send "ELECTION" message to $P_{(i+1)}$
16: **Processing Election Requests:**
17: **for** each received "ELECTION" message at P_j **do**
18: **if** P_j wants to be the leader, **then**
19: Add P_j to the candidates list
20: Forward the "ELECTION" message to $P_{(j+1)}$
21: **else**
22: Forward the "ELECTION" message to $P_{(j+1)}$
23: **end if**
24: **if** receives second "ELECTION" from lower ID **then**
25: Send "SUPPRESSION" message to the sender
26: **end if**
27: **end for**
28: **Leader Declaration:**
29: **if** process with the highest ID receives no further messages
30: **then**
30: Start sending "ACKNOWLEDGEMENT" message
31: **end if**
32: **Leader Announcement:**
33: **if** process receives a "LEADER" message **then**
34: Forward to all processes
35: **else**
36: Declare itself as leader if all others failed
37: **end if**

D. Description using example

Let's investigate the performance of our proposed algorithm in worst case, average case and best case scenario referring to fig I, II and III respectively to show that it exhibits substantial improvement in leader election by reducing the time gap between leader failure and election of new leader as well as reducing the massage passing overhead.

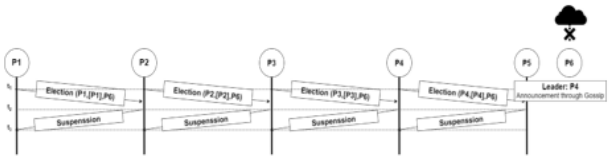

Fig. I. Worst Case

In the worst-case scenario, all processes detect P_6's failure and initiate elections. Without suppression, each process would communicate with all others, causing message overload. How- ever, the algorithm's suppression mechanism quickly stops redundant elections. When P_5, the highest process ID, detects P_6's failure, it declares itself the leader. Within just three time instances-failure detection, suppression, and leader declaration-the election is completed efficiently, making the algorithm much faster than others with minimal message overhead.

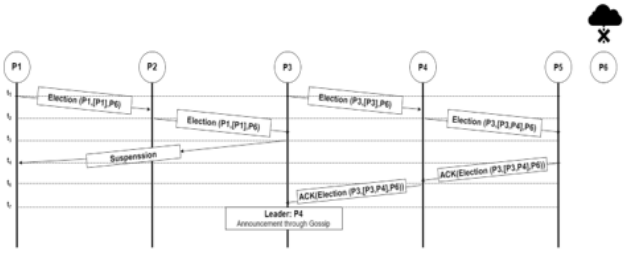

Fig. II. Average Case

In the average-case scenario, when both P_1 and P_3 detect P_6's failure and initiate elections, the proposed algorithm efficiently suppresses redundant messages, allowing only one election to proceed. P_5, being the highest ID, starts sending an acknowledgement message back to the previous process, and the leader selection is completed as soon as this acknowledgement reaches the highest ID process that initiated the election. This minimizes message exchanges and reduces election time, making the algorithm faster and more efficient than others.

Fig. III. Best Case

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 06, June 2026)

In the best-case scenario, P_5, being the highest ID, detects the failure of P_6 and immediately declares itself the leader in the first time instant. Since no other processes initiate an election, the leader is declared instantly without any additional communication or acknowledgment messages. This ensures the fastest possible resolution.

E. Time complexity analysis

The proposed election algorithm consistently maintains a time complexity of $O(1)$ in all scenarios—best, average, and worst—because it operates with a fixed number of interactions, independent of the number of processes n. When a process P_i detects a leader failure, it sends a single "ELECTION" message to P_{i+1}, with the total time represented as:

$$T(n) = k \cdot c$$

where k is the constant number of operations and c is the constant time for each. Unlike traditional algorithms like the Bully algorithm, Ring and LCR, and HS, the proposed solution guarantees that the time complexity remains independent of n. With reduced massage passing overhead and upholding constant interaction the proposed algorithm proved itself to be superlative for large scale distributed system.

F. Message passing complexity

In the best-case scenario, when only one process detects the leader's failure, the process with the highest ID can immediately declare itself the leader. This occurs without triggering additional elections or message exchanges from other processes, resulting in a message complexity of $O(1)$. In contrast with traditional strategies involving repetitive message passing, the proposed algorithm shows the completion of leader election process almost immediately. In the average case scenario though, multiple process may simultaneously detect the leader failure and initiate election, the novelty of the proposed algorithm lies in the generation of suppression message that ensure only one election message to be alive. The message complexity in this case is $O(k + \log k)$, where k is much smaller than n. The reduced message complexity makes the proposed solution sustainable for highly scalable distributed network which rely on linear communication The worst case scenario occurs if all the existing processes detects the leader failure and initiate the election simultaneously. Here, the existence of suppression message discard the unnecessary message passing though complexity will be $O(n)$ theoretically, which is even less as compared to Bully algorithm with complexity $O(n^2)$ and HS algorithm with complexity $O(n \log n)$, but actually the proposed algorithm ensure less number of messages and much more faster election.

VI. Simulation

A. Experimental setup and Dataset description

We did not use any conventional dataset for this study instead simulated Best, Average and Worst case scenario for Bully algorithm, Proposed algorithm and other Circular struc- ture based algorithms in Python using Google Colab. This code captures the communication across the involving processes. Result is reported across different number of processes in the range from 5 to 2500. Performance is measured in terms of number of messages exchanged and required time for the algorithms in each of the three cases.

B. Result and Discussion

Fig IV depicts the significant difference in efficiency that is found when the quantity of messages sent in various algorithms is evaluated. For example, the Bully algorithm with a process size of 10 exchanges 9 messages in the best case, 20 messages in the average scenario, and 88 messages in the worst situation. This figure makes it clear that the quantity of messages exchanged by Ring, LCR, TimeSlice, and VariableSpeed is roughly equal. Similar to Ring, the amount of messages in LCR go sequentially through the whole ring of processes. Although subgroups are processed in stages in HS, which improves message complexity, the messages must still pass through the ring's various sections in order. The message complexity is proportional in TimeSclice as well. Due to efficient design the proposed algorithm exhibits a constant time complexity across all the scenarios as shown in Fig V. The entire time gap between detection of leader failure to new leader announcement does not necessarily dependent on the number of process where in all other algorithms time complexity is measured in terms of n, as their performance scales directly with the number of processes involved, leading to significant delays in larger systems.

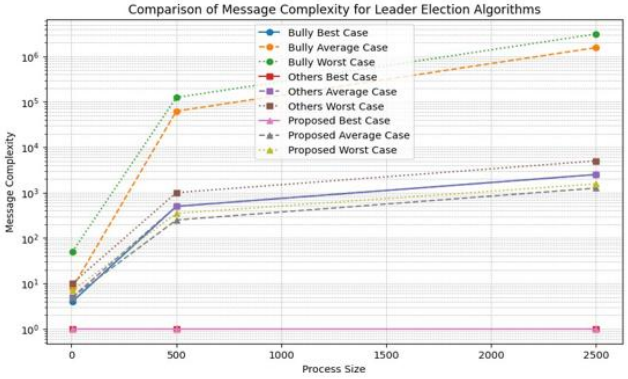

Fig. IV. Message complexity comparison

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 06, June 2026)

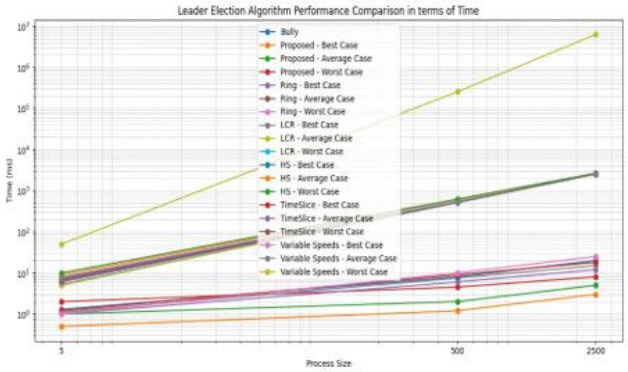

Fig. V. Time complexity comparison

The main contributions of the proposed algorithm are as follows:

Suspension Message: Without suspension, every process P_i sends messages to all higher ID processes, resulting in $n - i - 1$ messages for every process. The number of messages without suppression can be approximated by adding the messages sent by all processes:

$$M_{\text{no-suppression}} = \sum_{i=1}^{n}(n-i) = (n-1)+(n-2)+\cdots+1+0$$

$$= \frac{(n-1)n}{2}$$

With the suspension scheme, each process can send one suppression message if it receives a lower-ID election request:

$$M_{\text{suppression}} = k + (k-1) = 2k - 1$$

where k is the number of active participants.

Selecting Process ID (Highest and willing) as Leader: Logically every process is equally capable to act as a leader, but we kept our choice straightforward to circumvent ambiguity, added reckoning, and postponements. The process having highest process ID would be the best choice to be the upcoming leader but to be fair to the processes' decision of whether to participate or not in leadership, the highest process ID among the willing processes will be the next leader.

Gossip over Broadcasting: In networking broadcasting is an expensive issue, that's why we have used the Gossip protocol for leader announcement than broadcasting. As a result, message overhead has been reduced significantly and energy efficiency is increased. Moreover, it facilitates fast convergence.

Time Complexity: The time complexity of the proposed algorithm shows that it is invariant of number of processes involved. In the best case scenario, the highest process id detects the failure and choose itself as leader without any ambiguity and extra message exchange. In the average case scenario, the suppression message are highly effective for avoiding congestion due to message flooding. Even in the worst case scenario, the election is completed within few instant of time maintaining rapid convergence in constant time.

VII. Conclusion

In this paper, we propose an algorithm that presents a suppression mechanism and hierarchical acknowledgment, mitigating message flooding, delayed election, and congestion problems in traditional leader election strategies. The message-passing complexity of our algorithm outperforms traditional algorithms, and a critical contribution is achieving O(1) time complexity across all scenarios. These findings offer a novel perspective on maintaining seamless system performance, making it ideal for distributed infrastructures, where reducing coordination overhead is crucial for system efficiency.

REFERENCES

[1] H. Gracia-Molina, "Elections in a distributed computing system," IEEE Transactions on Computers, vol. C-31, no. 1, pp. 48–59, Jan 1982.

[2] N. Fredrickson and N. Lynch, "Electing a leader in a synchronous ring," Journal of the ACM, vol. 34, no. 1, pp. 98–115, Jan 1987.

[3] S. Tanenbaum and M. V. Steen, Distributed Systems: Principles and Paradigms, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2007.

[4] E. Chang and R. Roberts, "An improved algorithm for decentralized extrema-finding in circular configurations of processes," Communications of the ACM, vol. 22, no. 5, pp. 281–283, May 1979.

[5] N. A. Lynch, Distributed Algorithms. San Mateo, CA: Morgan Kaufmann, 1997.

[6] M. Numan, F. Subhan, and W. Z. Khan, "Well-organized bully leader elec- tion algorithm for distributed system," in 2018 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET), 2018, pp. 1–6.

[7] A. Biswas and A. Dutta, "A timer based leader election algorithm," in 2016 International IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress, 2016, pp. 1–7.

[8] A. Biswas and A. K. Tripathi, "Preselection based leader election in distributed systems," in Intelligent Distributed Computing XIV, Studies in Computational Intelligence, vol. 1026, 2022, pp. 265–274.