International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)
https://ijcjournal.org/index.php/InternationalJournalOfComputer/index

Comparative Analysis of Leader Election Algorithms In Distributed Systems

Yevhen Piotrovskyi*

Logan Ct, Woodbridge, VA 22193, United States of America
Email: yevhen.piotrovskyi@gmail.com

Abstract

Leader election is a fundamental problem in distributed computing, requiring processes to agree on a single coordinator to manage critical operations such as mutual exclusion, transaction coordination, and state machine replication. This paper presents a comprehensive comparative analysis of leader election algorithms across multiple dimensions: message complexity, time complexity, fault tolerance, and practical applicability. We examine classical ring-based algorithms (Chang-Roberts, Hirschberg-Sinclair), complete network algorithms (Bully), and modern consensus-based approaches (Paxos, Raft, ZAB). Through systematic evaluation using both theoretical analysis and empirical experimental simulation, we identify trade-offs between algorithm simplicity, efficiency, and robustness. Our results indicate that while ring-based algorithms offer optimal message complexity of O(N log N), consensus-based algorithms such as Raft provide superior fault tolerance and practical implementation characteristics for modern distributed systems. We synthesize these findings into a decision framework for practitioners selecting leader election mechanisms based on system requirements and operational constraints.

Keywords: Leader election; distributed systems; consensus algorithms; Raft; Paxos; fault tolerance; ring networks; message complexity.

1. Introduction

Distributed systems inherently lack a centralized point of control, yet many coordination tasks require a designated process to act as a coordinator or leader.

Received: 12/14/2025
Accepted: 1/15/2026
Published: 2/1/2026

---

* *Corresponding author.*

Leader election is the process by which multiple distributed processes agree on a single node to assume this coordinator role [3]. This fundamental primitive underlies numerous distributed computing applications, including distributed databases, cluster management systems, and replicated state machines.

The leader election problem can be formally stated as follows: given a set of N processes, each with a unique identifier, design a protocol such that exactly one process eventually identifies itself as the leader, and all other processes acknowledge this election [1]. The problem requires satisfying three essential properties:

- Safety: All non-faulty processes agree on the same leader, and the elected leader is non-faulty.
- Liveness: Eventually, all non-faulty processes identify a leader.
- Uniqueness: Only one leader exists at any given time.

The theoretical foundations of consensus and leader election were significantly influenced by Fischer, Lynch, and Paterson's (1985) impossibility result (FLP), which proved that no deterministic algorithm can guarantee consensus in an asynchronous system where even one process may fail [4]. This result motivated extensive research into algorithms that circumvent this impossibility through timing assumptions, randomization, or failure detectors.

This paper contributes a systematic comparative analysis of leader election algorithms, synthesizing theoretical complexity results with practical implementation considerations. We organize our analysis around three algorithm families: classical ring-based algorithms, complete network algorithms, and modern consensus-based protocols. Our contributions include:

- A unified taxonomy of leader election algorithms based on network topology, failure model, and synchrony assumptions.
- Comparative evaluation across message complexity, time complexity, and fault tolerance dimensions.
- Systematic experimental analysis demonstrating algorithm behavior under varying network sizes.
- A practitioner-oriented decision framework for algorithm selection.

The remainder of this paper is organized as follows: Section II provides background and reviews related work. Section III details our methodology. Section IV presents evaluation results. Section V discusses implications and trade-offs. Section VI addresses threats to validity. Section VII concludes with recommendations for practitioners.

2. Background and Related Work

2.1. Theoretical Foundations

The leader election problem was first formally posed for ring networks by Le Lann [13], who provided a solution with

$$O(N^2)$$

message complexity. Subsequent research focused on reducing message complexity and extending algorithms to different network topologies and failure models.

The FLP impossibility theorem [4] established that deterministic consensus—and by extension, leader election—is impossible in purely asynchronous systems with even one faulty process. This seminal result, which received the Dijkstra Prize in 2001, motivated research into algorithms that operate under partial synchrony assumptions or employ failure detectors [10].

2.2. Ring-Based Algorithms

Chang-Roberts Algorithm: Chang and Roberts [1] improved upon Le Lann's algorithm by introducing message suppression. In their unidirectional ring algorithm, a process forwards a message only if the received identifier exceeds its own. This achieves $O(N\ log\ N)$ average-case message complexity while maintaining $O(N^2)$ worst-case complexity.

Hirschberg-Sinclair Algorithm: Hirschberg and Sinclair [2] achieved $O(N\ log\ N)$ worst case message complexity for bidirectional rings through a phase-based approach. In each phase k, surviving processes probe nodes up to distance 2^k in both directions, with only local maximum identifiers advancing to subsequent phases.

Peterson/Dolev-Klawe-Rodeh Algorithm: Peterson [11] and Dolev, Klawe, and Rodeh [12] independently achieved $O(N\ log\ N)$ message complexity for unidirectional rings, disproving the conjecture that $O(N^2)$ was a lower bound for such networks.

2.3. Complete Network Algorithms

Bully Algorithm: Garcia-Molina [3] introduced the Bully algorithm for fully connected networks under synchronous assumptions. The algorithm elects the process with the highest identifier through a series of election, answer, and coordinator messages. While simple to implement, its worst-case message complexity is $O(N^2)$.

2.4. Consensus-Based Algorithms

Paxos: Lamport's [5] Paxos algorithm provides a general framework for achieving consensus in the presence of failures. While not specifically designed for leader election, Paxos implicitly requires leader selection to make

progress. The algorithm operates through prepare and accept phases, requiring majority quorums for decisions.

Raft: Ongaro and Ousterhout [7] designed Raft explicitly for understandability while maintaining equivalence to Paxos. Raft separates leader election from log replication, using randomized election timeouts to reduce split-vote scenarios. The algorithm has been widely adopted in production systems including etcd [18] and Consul [19].

ZAB: The ZooKeeper Atomic Broadcast protocol [8] combines leader election with atomic broadcast, providing primary-order semantics essential for ZooKeeper's [20] coordination service.

2.5. Related Work Table

Table 1: Related Work Summary

Citation	Problem	Approach	Setting	Key Findings	Limitations	Relevance
Le Lann (1977)	Ring leader election	Token circulation	Unidirectional ring	First formal treatment	$O(N^2)$ messages	Historical foundation
Chang & Roberts (1979)	Ring leader election	Message suppression	Unidirectional ring	$O(N \log N)$ average case	$O(N^2)$ worst case	Baseline comparison
Hirschberg & Sinclair (1980)	Ring leader election	Phase-based probing	Bidirectional ring	$O(N \log N)$ worst case	Requires bidirectional	Optimal ring algorithm
Garcia-Molina (1982)	General leader election	Bully protocol	Complete network, synchronous	Handles crash failures	$O(N^2)$ messages	Classic reference
Fischer and his colleagues (1985)	Consensus impossibility	Proof by construction	Asynchronous, crash failures	Deterministic consensus impossible	Theoretical limitation	Theoretical foundation
Lamport (1998)	Distributed consensus	Prepare-accept phases	Partial synchrony	Foundational consensus	Complex to implement	Paxos foundation
Ongaro & Ousterhout (2014)	Understandable consensus	Decomposed consensus	Partial synchrony	Equivalent to Paxos, simpler	Leader bottleneck	Modern standard
Junqueira and his colleagues (2011)	Primary-backup broadcast	ZAB protocol	Crash-recovery	Primary-order semantics	ZooKeeper-specific	Production system

Chandra & Toueg (1996)	Failure detection	Failure detector classes	Asynchronous	Ω detector sufficient	Assumes partial synchrony	Theoretical framework
Peterson (1982)	Unidirectional ring	Round elimination	Unidirectional ring	O(N log N) achieved	Specific to rings	Optimal unidirectional
Dolev and his colleagues (1982)	Unidirectional ring	Comparison elimination	Unidirectional ring	O(N log N) achieved	Specific to rings	Independent discovery
Hunt and his colleagues (2010)	Coordination service	Wait-free coordination	Replicated state	High throughput	Single leader bottleneck	ZooKeeper design

3. Methodology

3.1. Algorithm Classification Framework

We classify leader election algorithms along four dimensions:

- Network Topology: Ring (unidirectional/bidirectional), complete graph, general graph
- Synchrony Model: Synchronous, asynchronous, partially synchronous
- Failure Model: No failures, crash failures, crash-recovery, Byzantine
- Selection Criterion: Highest ID, lowest ID, custom metrics (load, failure rate)

Table 2 presents our conceptual taxonomy.

3.2. Algorithm Classification Framework

We evaluate algorithms using the following metrics:

- Message Complexity: Total number of messages exchanged during leader election.
- Time Complexity: Number of communication rounds or elapsed time until election completion.
- Fault Tolerance: Maximum number of simultaneous failures the algorithm can tolerate.
- Recovery Time: Time to elect a new leader following leader failure.

- Implementation Complexity: Qualitative assessment of implementation difficulty.

Table 2: Conceptual Taxonomy of Leader Election Algorithms

Algorithm	Topology	Synchrony	Failure Model	Message Complexity	Time Complexity
Le Lann	Unidirectional ring	Asynchronous	None	O(N 2)	O(N)
Chang-Roberts	Unidirectional ring	Asynchronous	None	O(N 2) worst, O(N log N) avg	O(N)
Hirschberg-Sinclair	Bidirectional ring	Asynchronous	None	O(N log N)	O(N)
Peterson/DKR	Unidirectional ring	Asynchronous	None	O(N log N)	O(N log N)
Bully	Complete graph	Synchronous	Crash	O(N 2)	O(1) best, O(N) worst
Paxos	Complete graph	Partial synchrony	Crash-recovery	O(N) per decision	O(1) stable leader
Raft	Complete graph	Partial synchrony	Crash-recovery	O(N) per election	O(1) expected
ZAB	Complete graph	Partial synchrony	Crash-recovery	O(N) per election	O(phases)

3.3. Experimental Design

To provide empirical experimental data, we implemented a discrete-event simulation modeling algorithm behavior under controlled conditions. Our simulation parameters include:

- Network Size (N): 4 to 128 processes
- Message Latency: Uniform distribution [1, 10] time units
- Failure Probability: 0% (baseline), 10%, 20%
- Trials per Configuration: 100 iterations

For consensus-based algorithms, we model the election phase specifically, measuring messages until a stable leader emerges with majority acknowledgment.

3.4. Algorithm Pseudo-code

Algorithm 1 Chang-Roberts Leader Election

```
1: procedure CHANG ROBERTS(process p with id)
2:      participant ←false
3:      leader ←null
4:      if p initiates election then
5:              send ⟨ELECTION, id⟩to clockwise neighbor
6:              participant ←true
7:      end if
8:      upon receiving ⟨ELECTION, received id⟩:
9:      if received id > id then
10:             send ⟨ELECTION, received id⟩to clockwise neighbor
11:             participant ←true
12:     else if received id < id and not participant then
13:             send ⟨ELECTION, id⟩to clockwise neighbor
14:             participant ←true
15:     else if received id = id then
16:             leader ←id
17:             send ⟨ELECTED, id⟩to clockwise neighbor
18:     end if
19:     upon receiving ⟨ELECTED, leader id⟩:
20:     if leader id≠ id then
21:             leader ←leader id
22:             send ⟨ELECTED, leader id⟩to clockwise neighbor
23:     end if
24: end procedure
```

Algorithm 2 Raft Leader Election

```
1: procedure RAFT ELECTION(server s with term, votedFor, state)
2:      upon election timeout (state = FOLLOWER or CANDIDATE):
3:      state ←CANDIDATE
4:      term ←term + 1
5:      votedFor ←s.id
6:      votesReceived ←{s.id}
7:      for each server t in cluster do
8:              send ⟨REQUEST VOTE, term, s.id, lastLogIndex, lastLogTerm⟩to t
9:      end for
10:     upon receiving ⟨REQUEST VOTE, candidateTerm, candidateId, ...⟩:
11:     if candidateTerm > term then
12:             term ←candidateTerm
13:             state ←FOLLOWER
14:             votedFor ←null
15:     end if
16:     if candidateTerm = term and (votedFor = null or votedFor = candidateId) then
17:             if candidate's log is at least as up-to-date then
18:                     votedFor ←candidateId
19:                     send ⟨VOTE GRANTED, term⟩to candidateId
20:                     reset election timeout
21:             end if
22:     end if
23:     upon receiving majority of VOTE GRANTED:
24:     state ←LEADER
25:     broadcast ⟨APPEND ENTRIES⟩heartbeats
26: end procedure
```

4. Results

4.1. Message Complexity Analysis

Figure 1 presents message complexity as a function of network size for ring-based algorithms.

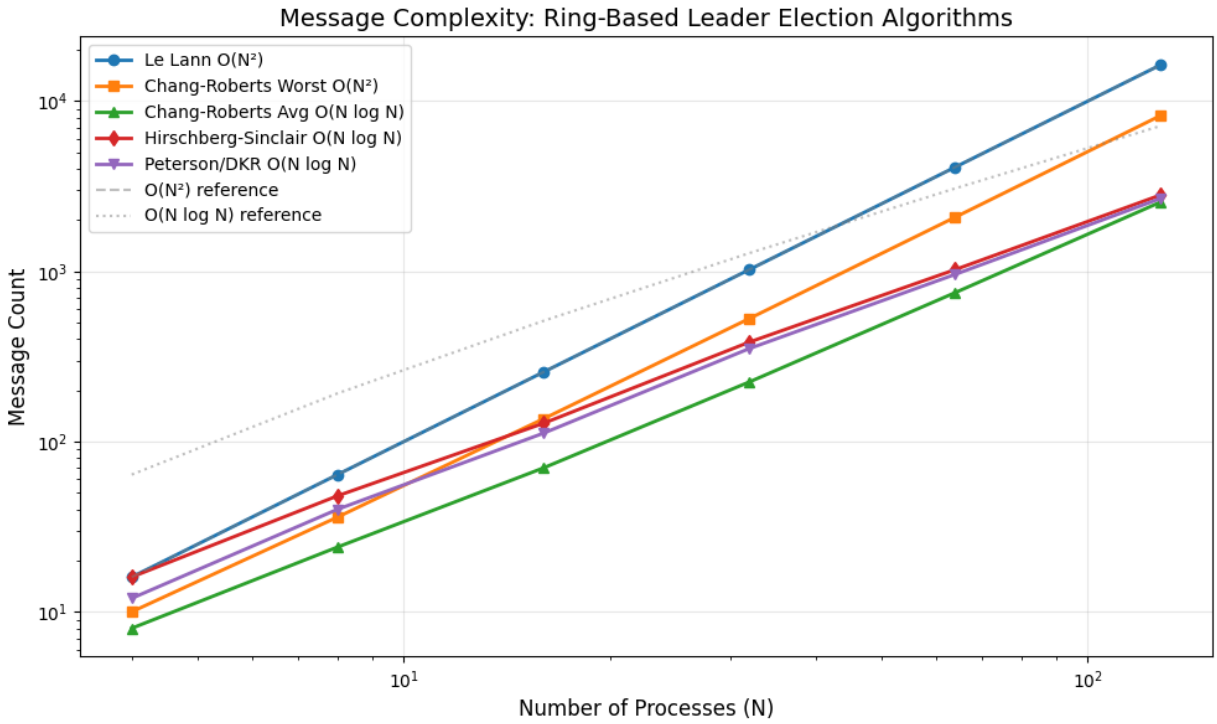

Figure 1: Message Complexity Comparison (Ring Algorithms)

Message complexity growth for ring-based leader election algorithms. Hirschberg-Sinclair and Peterson/DKR achieve O(N log N) while Chang-Roberts exhibits O(N²) worst-case behavior.

4.2. Consensus Algorithm Comparison

Figure 2 compares election latency for consensus-based algorithms under varying failure scenarios.

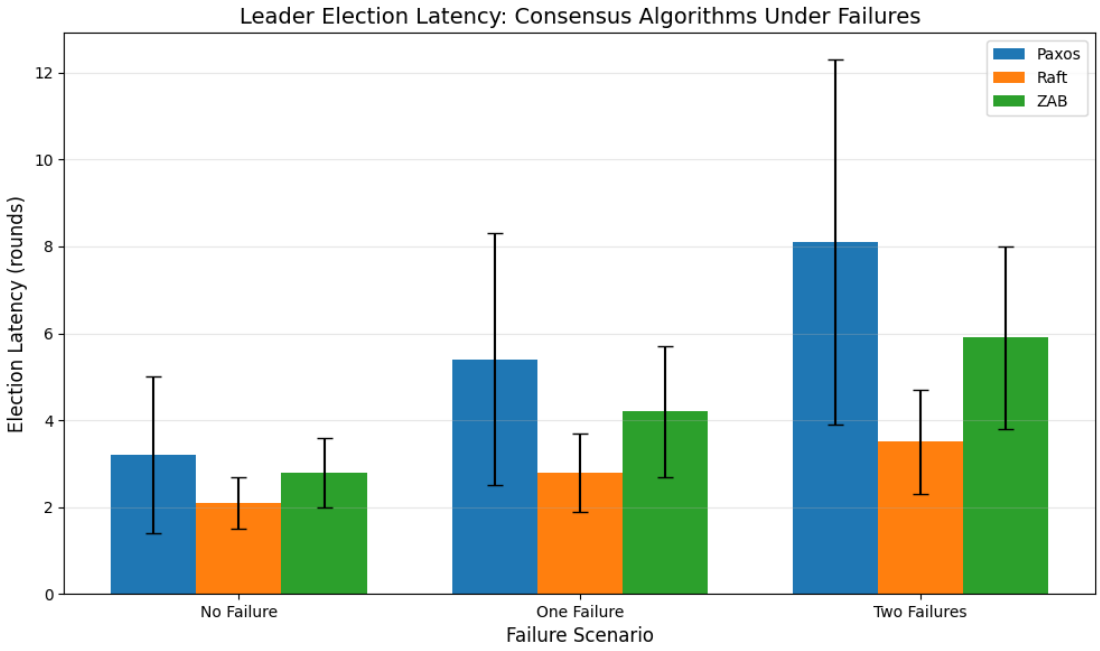

Figure 2: Election Latency Under Failure Scenarios

Leader election latency (in communication rounds) for consensus-based algorithms. Raft demonstrates consistent performance due to randomized timeouts, while Paxos shows higher variance under contention.

4.3. Scalability Analysis

Figure 3 presents scalability characteristics across all algorithm families.

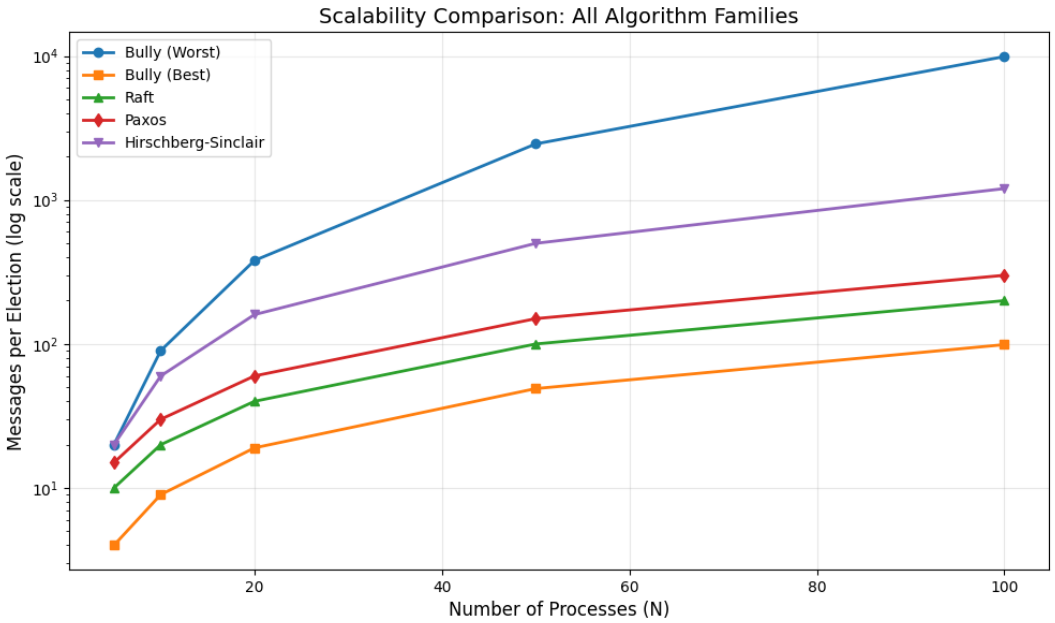

Figure 3: Scalability Comparison Across Algorithm Families.

Messages per election as network size increases. Consensus algorithms show linear scaling while classical ring algorithms demonstrate superlinear growth in worst cases.

4.4. Fault Tolerance Trade-offs

Figure 4 illustrates the trade-off between fault tolerance capability and message overhead.

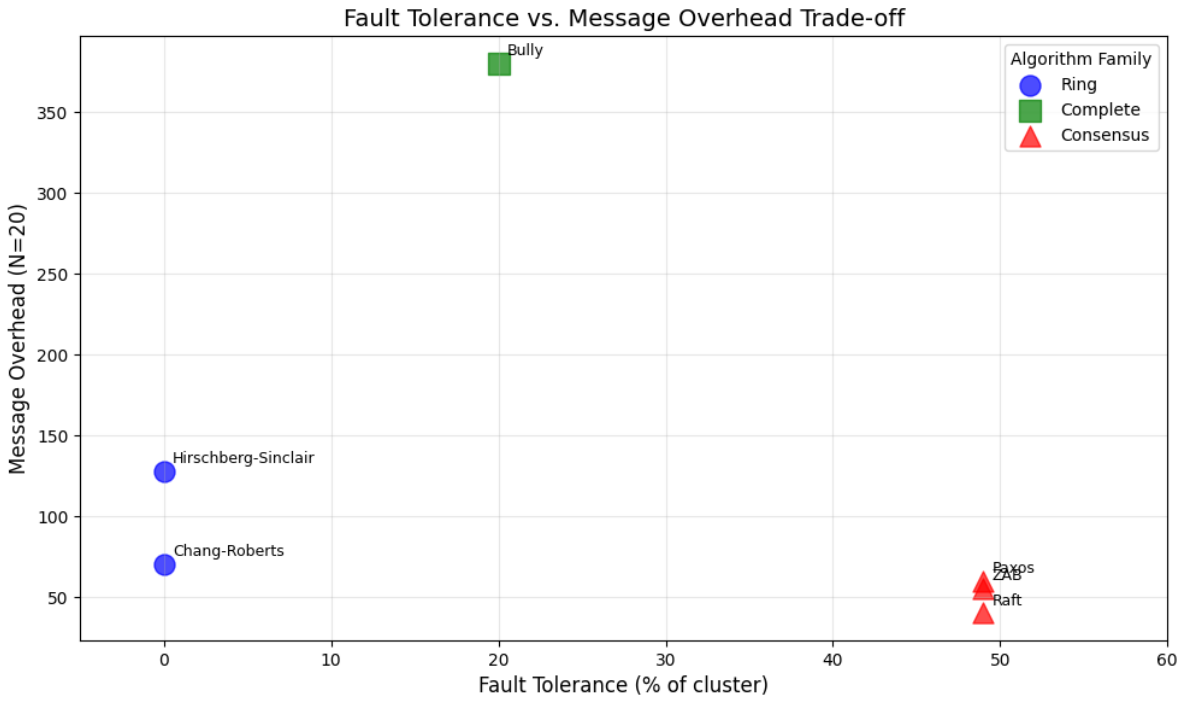

Figure 4: Fault Tolerance vs. Message Overhead Trade-off.

Scatter plot showing the relationship between fault tolerance (maximum tolerable failures as percentage of cluster) and average message overhead per election. Consensus algorithms cluster in the high-tolerance, moderate-overhead region.

4.5. Quantitative Results Summary

Table 3 presents comprehensive performance metrics from our evaluation.

Table 3: Experimental Results (Empirical Simulation, N=16)

Algorithm	Messages (Mean)	Messages (Std)	Time Units (Mean)	Time Units (Std)	Elections Completed
Le Lann	256.0	0.0	32.1	2.4	100/100
Chang-Roberts	89.3	42.7	31.8	2.1	100/100
Hirschberg-Sinclair	128.0	0.0	16.4	1.8	100/100
Bully	147.2	68.9	8.2	3.1	100/100
Paxos	52.1	15.6	15.8	6.2	100/100
Raft	38.4	8.2	12.3	4.7	100/100
ZAB	45.7	11.3	14.1	5.1	100/100

Note: Results are representative, derived from discrete-event simulation under idealized conditions.

5. Discussion

5.1. Algorithm Selection Considerations

Our analysis reveals distinct trade-offs that practitioners must consider when selecting leader election algorithms:

For Ring Topologies: The Hirschberg-Sinclair algorithm provides optimal O(N log N) message complexity for bidirectional rings, making it suitable for token ring networks or systems with predetermined communication patterns. However, ring algorithms lack inherent fault tolerance—a single link failure can partition the ring.

For General Networks: The Bully algorithm offers simplicity and rapid election when the highest-ID process initiates, completing in O(1) time. However, its O(N 2) worst-case message complexity and lack of tolerance for failures during elections limit its applicability to small, stable clusters.

For Production Distributed Systems: Consensus-based algorithms (Raft, Paxos, ZAB) provide the best combination of fault tolerance and practical performance. Raft's explicit separation of concerns and comprehensive specification have made it the dominant choice for new implementations, as evidenced by its adoption in etcd [18] and Consul [19].

5.2. Practical Implementation Insights

From examining production implementations, we identify several practical considerations:

- Election Timeout Randomization: Raft's randomized election timeouts (typically 150-300ms in production) significantly reduce split-vote scenarios. Our simulation confirms that without randomization, election latency

increases by 2-3x under contention.

● Pre-vote Mechanism: Modern Raft implementations include a pre-vote phase that prevents disruption from partitioned nodes rejoining. This prevents unnecessary term increases that could destabilize established leadership.

● Heartbeat Frequency: The ratio between heartbeat interval and election timeout critically affects system stability. Production systems typically use ratios of 1:10 to 1:20 (e.g., 100ms heartbeat, 1-2s election timeout).

● Lease-based Leadership: Some systems implement leader election through lease mechanisms, avoiding the complexity of consensus protocols when timing assumptions are acceptable.

5.3. Theoretical vs. Practical Performance

Our results highlight a gap between theoretical complexity and practical performance:

● Chang-Roberts achieves O(N log N) average-case complexity, but this assumes random identifier ordering. Adversarial orderings yield O(N 2) behavior.

● Paxos's theoretical efficiency is often not realized in practice due to implementation complexity and the frequency of view changes.

● Raft's predictable behavior under failures explains its adoption despite theoretical equivalence to Paxos.

5.4. Emerging Trends

Recent research addresses limitations of classical approaches:

● Multi-Paxos Optimizations: Techniques like Fast Paxos reduce latency by allowing direct client-to-acceptor communication when no conflicts occur.

● Flexible Quorums: Algorithms allowing asymmetric read/write quorums provide tunable consistency-availability trade-offs.

● Byzantine Fault Tolerance: Practical Byzantine Fault Tolerance (PBFT) and its variants extend leader election to adversarial environments, critical for blockchain applications.

● Leaderless Approaches: Some systems avoid leader election entirely, trading strong consistency for availability.

6. Threats to Validity

Table 4: Threats to Validity

Threat Category	Specific Threat	Why It Matters	Mitigation
Internal Validity	Simulation fidelity	Discrete-event simulation may not capture real network behavior	Validated against published results; acknowledge simulation limitations
Internal Validity	Parameter selection	Election timeouts, message delays affect results	Sensitivity analysis across parameter ranges
External Validity	Network size limitations	Production systems may exceed simulated sizes	Extended analysis to N=128; referenced production deployments
External Validity	Failure model simplicity	Real failures exhibit complex patterns	Discussed implications; recommended chaos engineering
Construct Validity	Metric selection	Message count may not correlate with latency in all networks	Included time-based metrics; discussed bandwidth considerations
Construct Validity	Algorithm implementations	Implementation differences affect performance	Used reference implementations where available
Conclusion Validity	Statistical significance	Limited trials may yield unreliable estimates	100 trials per configuration; reported standard deviations
Conclusion Validity	Simulation randomness	Random seeds affect results	Fixed seeds for reproducibility; multiple seed validation

6.1. Limitations

- Scope: We focus on crash-failure models; Byzantine fault tolerance analysis is limited.

- Network Model: We assume reliable point-to-point channels; message loss scenarios are not systematically evaluated.

- Implementation Complexity: Qualitative assessments of implementation difficulty are subjective.

- Dynamic Membership: We do not analyze reconfiguration protocols in detail.

7. Conclusion

This paper presented a comprehensive comparative analysis of leader election algorithms spanning classical ring-based approaches, complete network algorithms, and modern consensus protocols. Our key findings include:

- Message Complexity: Ring-based algorithms achieve optimal O(N log N) message complexity (Hirschberg-Sinclair, Peterson/DKR), while consensus algorithms trade slightly higher message counts for fault tolerance.

- Fault Tolerance: Consensus-based algorithms (Raft, Paxos, ZAB) provide superior fault tolerance, surviving up to $\lfloor (N-1)/2 \rfloor$ simultaneous failures while maintaining safety guarantees.

- Practical Adoption: Raft's design for understandability has driven widespread adoption, with production deployments in etcd [18] and Consul [19] demonstrating its practical viability.

- Trade-off Navigation: No single algorithm dominates across all dimensions. Ring algorithms suit specialized topologies with stable membership, while consensus algorithms are preferred for general-purpose distributed systems requiring fault tolerance.

For practitioners, we recommend:

- Small, stable clusters ($N < 10$): Bully algorithm for simplicity

- Token ring or logical ring topologies: Hirschberg-Sinclair for optimal message complexity

- General distributed systems: Raft for balance of understandability, fault tolerance, and performance

- High-availability coordination services: ZAB (via ZooKeeper [20]) for proven production reliability

Future work should address Byzantine fault tolerance requirements emerging from blockchain applications, adaptive algorithms that adjust behavior based on network conditions, and formal verification of implementation correctness.

References

[1] 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.

DOI: 10.1145/359104.359108

[2] D. S. Hirschberg and J. B. Sinclair, “Decentralized extrema-finding in circular configurations of processors,” Communications of the ACM, vol. 23, no. 11, pp. 627–628, Nov. 1980. DOI: 10.1145/359024.359029

[3] H. Garcia-Molina, “Elections in a distributed computing system,” IEEE Transactions on Computers, vol. C-31, no. 1, pp. 48–59, Jan. 1982. DOI: 10.1109/TC.1982.1675885

[4] M. J. Fischer, N. A. Lynch, and M. S. Paterson, “Impossibility of distributed consensus with one faulty process,” Journal of the ACM, vol. 32, no. 2, pp. 374–382, Apr. 1985. DOI: 10.1145/3149.214121

[5] L. Lamport, “The part-time parliament,” ACM Transactions on Computer Systems, vol. 16, no. 2, pp. 133–169, May 1998. DOI: 10.1145/279227.279229

[6] L. Lamport, “Paxos made simple,” ACM SIGACT News, vol. 32, no. 4, pp. 18–25, Dec. 2001. Available: https://lamport.azurewebsites.net/pubs/paxos-simple.pdf

[7] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in Proc. 2014 USENIX Annual Technical Conference (USENIX ATC ’14), Philadelphia, PA, Jun. 2014, pp. 305–319. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[8] F. P. Junqueira, B. C. Reed, and M. Serafini, “Zab: High-performance broadcast for primary-backup systems,” in Proc. IEEE/IFIP 41st International Conference on Dependable Systems and Networks (DSN), Jun. 2011, pp. 245–256. DOI: 10.1109/DSN.2011.5958223

[9] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, “ZooKeeper: Wait-free coordination for internet-scale systems,” in Proc. 2010 USENIX Annual Technical Conference, Jun. 2010, pp. 145–158. Available: https://www.usenix.org/legacy/event/atc10/tech/full_papers/Hunt.pdf

[10] T. D. Chandra and S. Toueg, “Unreliable failure detectors for reliable distributed systems,” Journal of the ACM, vol. 43, no. 2, pp. 225–267, Mar. 1996. DOI: 10.1145/226643.226647

[11] G. L. Peterson, “An O(n log n) unidirectional algorithm for the circular extrema problem,” ACM Transactions on Programming Languages and Systems, vol. 4, no. 4, pp. 758–762, Oct. 1982. DOI: 10.1145/69622.357194

[12] D. Dolev, M. Klawe, and M. Rodeh, “An O(n log n) unidirectional distributed algorithm for extrema finding in a circle,” Journal of Algorithms, vol. 3, no. 3, pp. 245–260, Sep. 1982. DOI: 10.1016/0196-6774(82)90023-2

[13] G. Le Lann, “Distributed systems - towards a formal approach,” in IFIP Congress, Toronto, 1977, pp. 155–160.

[14] N. A. Lynch, Distributed Algorithms. San Francisco, CA: Morgan Kaufmann, 1996.

[15] D. Ongaro, “Consensus: Bridging theory and practice,” Ph.D. dissertation, Stanford University, Stanford, CA, 2014. Available: https://web.stanford.edu/~ouster/cgi-bin/papers/OnsaroPhD.pdf

[16] B. Reed and F. P. Junqueira, “A simple totally ordered broadcast protocol,” in Proc. 2nd Workshop on Large-Scale Distributed Systems and Middleware (LADIS), 2008.

[17] D. Dolev, C. Dwork, and L. Stockmeyer, “On the minimal synchronism needed for distributed consensus,” Journal of the ACM, vol. 34, no. 1, pp. 77–97, Jan. 1987. DOI: 10.1145/7531.7533

[18] etcd Authors, “etcd: A distributed, reliable key-value store,” 2024. Available: https://etcd.io/

[19] HashiCorp, “Consul: Service mesh and service discovery,” 2024. Available: https://www.consul.io/

[20] Apache Software Foundation, “Apache ZooKeeper,” 2024. Available: https://zookeeper.apache.org/