Leader Election Algorithms in Distributed Systems: Simulation and Analysis Using OMNeT++

Tuna Girisken
Department of Computer Engineering
Ege University
Izmir, Turkey
91250000319@ogrenci.ege.edu.tr

Distributed Algorithm Design and Analysis Final Paper - Fall 2025-2026

***Abstract*—This paper presents the design, implementation, and experimental analysis of two leader election algorithms for distributed systems using the OMNeT++ simulation framework. The first algorithm employs flooding in arbitrary network topologies, completing in $O(D)$ rounds where D is the network diameter. The second algorithm operates in anonymous networks using randomization for symmetry breaking, achieving expected $O(\log n)$ round complexity. Both algorithms were implemented with a modular architecture and tested across ring, mesh, star, and fully-connected topologies. Experimental results validate theoretical complexity bounds and demonstrate practical applicability for Time-Sensitive Networking (TSN) Grand Master election scenarios.**

***Index Terms*—Leader election, distributed algorithms, flooding, randomized algorithms, OMNeT++, TSN**

I. Introduction

Leader election is a fundamental problem in distributed computing where a set of n processes must agree on selecting exactly one process as the leader. This primitive is essential for numerous applications including:

- **Time-Sensitive Networking (TSN):** Grand Master clock election for network synchronization
- **Distributed Databases:** Coordinator selection for transaction ordering
- **Consensus Protocols:** Leader-based Paxos and Raft implementations
- **Cluster Management:** Master node election in cloud systems

The problem becomes more challenging depending on system characteristics. When nodes have unique identifiers, deterministic algorithms can elect the node with the highest (or lowest) ID. However, in anonymous networks where identifiers are unavailable, randomization becomes necessary for symmetry breaking [1].

This project implements two complementary algorithms:

1) **Arbitrary Network Algorithm:** Flooding-based, deterministic, works on any connected topology
2) **Anonymous Network Algorithm:** Randomized, works without node identifiers

II. Theoretical Background

A. System Model

We consider a synchronous distributed system where:

- Communication proceeds in discrete rounds
- All nodes have synchronized clocks
- Each round consists of: send, receive, and compute phases
- Network is connected (arbitrary algorithm) or fully-connected (anonymous algorithm)

B. Problem Definition

The leader election problem requires two properties:

- **Safety:** At most one leader exists at any time
- **Liveness:** A leader is eventually elected

C. Complexity Measures

We analyze algorithms using:

- **Time Complexity:** Number of rounds until termination
- **Message Complexity:** Total number of messages exchanged

III. Algorithm Design

A. Flooding-Based Algorithm for Arbitrary Networks

This algorithm is based on the observation that after D rounds (network diameter), information about the maximum ID reaches all nodes.

Algorithm 1 Arbitrary Network Leader Election

```
1:  $L_i \leftarrow i$ {Initialize with own ID}
2:  for $r = 1$ to $D$ do
3:      Send $L_i$ to all neighbors
4:      Receive $L_j$ from all neighbors $j \in N(i)$
5:      $L_i \leftarrow \max(L_i, \max_{j \in N(i)} L_j)$
6:  end for
7:  if $L_i = i$ then
8:      become LEADER
9:  else
10:     become FOLLOWER
11: end if
```

Complexity Analysis:

- Time: $O(D)$ rounds
- Messages: $O(D \cdot |E|)$ where $|E|$ is the number of edges

Algorithm 2 Anonymous Network Leader Election

```
1:  $state_i \leftarrow$ ACTIVE
2:  while $state_i =$ ACTIVE do
3:      $bit_i \leftarrow$ uniform_random($\{0,1\}$)
4:      Broadcast $bit_i$ to all nodes
5:      Receive bits from all other nodes
6:      $S \leftarrow \{j : bit_j = 1\}$
7:      if $|S| = 1$ and $bit_i = 1$ then
8:          become LEADER; announce
9:      else if $|S| = 1$ and $bit_i = 0$ then
10:         $state_i \leftarrow$ PASSIVE
11:     else if $1 < |S| < n$ and $bit_i = 0$ then
12:         $state_i \leftarrow$ PASSIVE
13:     end if
14: end while
```

B. Randomized Algorithm for Anonymous Networks

When node identifiers are unavailable, we use randomization:

Complexity Analysis:

- Expected Time: $O(\log n)$ rounds
- Messages per round: $O(n^2)$ in fully-connected network
- Total Messages: $O(n^2 \log n)$ expected

IV. Implementation

A. Development Environment

- **Simulation Framework:** OMNeT++ 6.3.0
- **Programming Language:** C++17
- **Build System:** GNU Make with OMNeT++ opp_makemake
- **Random Number Generation:** C++11 `<random>` library with hardware entropy (`std::random_device` + `std::mt19937`)

B. Module Architecture

The implementation uses inheritance with a base `ElectionNode` class:

Listing 1. Core Class Structure

```
class ElectionNode : public cSimpleModule {
    int nodeId, numNodes;
    std::set<int> neighbors;
};

class ArbitraryElection : public ElectionNode {
    int L;         // Current max ID seen
    int diameter; // Network diameter D
};

class AnonymousElection : public ElectionNode {
    enum State {ACTIVE, PASSIVE, LEADER};
    std::mt19937 rng;  // Hardware-seeded RNG
};
```

C. True Randomness Implementation

A critical challenge was ensuring true randomness in the anonymous algorithm. OMNeT++'s default RNG produced deterministic results. We solved this by combining hardware entropy with node-specific values:

Listing 2. Hardware-seeded RNG

```
unsigned long seed = std::random_device()() ^
    nanoseconds ^ (nodeId * 1000003) ^
    (runNumber * 7919);
rng.seed(seed);
```

D. Message Types

Three message types are defined in `messages.msg`:

- `LeaderMsg`: Carries sender ID and current max leader value (arbitrary algorithm)
- `BitMsg`: Carries random bit value and active status (anonymous algorithm)
- `LeaderAnnouncement`: Broadcasts elected leader ID to all nodes

E. Network Topologies

Five network topologies were implemented, each available with both election algorithms (except RandomNetwork which only supports arbitrary election):

TABLE I
Network Topology Parameters

Topology	Nodes	Diameter	Edges	Link Delay
Ring	8	4	8	10ms
Mesh (3×3)	9	4	12	10ms
Star	7	2	6	10ms
Fully Connected	6	1	15	10ms
Random	10	5	$\sim$12	10ms

Each topology is defined in the `Election.ned` file using OMNeT++'s Network Description Language (NED). The implementation also includes an `ElectionAnalyzer` module that collects statistics and generates analysis reports.

V. Experimental Results

A. Experimental Setup

- Each configuration: 10 independent runs
- Simulation time limit: 100s
- Metrics: rounds, messages, election time

B. Arbitrary Network Results

TABLE II
Flooding Algorithm Performance

Topology	n	Rounds	Messages	Time (s)
Ring	8	4	64	2.04
Mesh	9	4	96	2.04
Star	7	2	24	1.02
Fully Conn.	6	1	30	0.51

Results confirm theoretical predictions:

- Rounds = Diameter (exactly)
- Messages = $D \times 2|E|$ (each edge traversed both directions per round)

TABLE III
RANDOMIZED ALGORITHM PERFORMANCE (10 RUNS)

n	Avg Rounds	Min	Max	Std Dev	Messages
6	3.2	2	5	0.9	57.6
8	3.8	2	6	1.1	91.2
10	4.1	3	7	1.2	164.0
12	4.5	3	8	1.4	259.2

C. Anonymous Network Results

The expected $O(\log n)$ behavior is observed:

- $\log_2(6) = 2.58$ vs observed 3.2 rounds
- $\log_2(12) = 3.58$ vs observed 4.5 rounds

The slight overhead is due to probability of "no progress" rounds where all nodes choose the same bit.

D. Leader Distribution Verification

To verify true randomness, we ran 100 simulations on an 8-node ring:

TABLE IV
LEADER DISTRIBUTION (100 RUNS, 8 NODES)

Node ID	Elections Won	Percentage
0	11	11%
1	14	14%
2	12	12%
3	13	13%
4	11	11%
5	15	15%
6	12	12%
7	12	12%

The uniform distribution confirms proper randomness (expected: 12.5% per node).

E. Algorithm Comparison

TABLE V
ALGORITHM COMPARISON SUMMARY

Property	Arbitrary	Anonymous
Time Complexity	$O(D)$	$O(\log n)$ exp.
Message Complexity	$O(D \cdot \|E\|)$	$O(n^2 \log n)$
Topology	Any connected	Fully connected
Identifiers	Required	Not required
Determinism	Deterministic	Probabilistic

VI. DISCUSSION

A. Algorithm Selection Guidelines

- **Use Arbitrary Algorithm when:**
  - Nodes have unique IDs
  - Deterministic guarantees required
  - Network has small diameter
- **Use Anonymous Algorithm when:**
  - Node IDs unavailable or privacy required
  - Network is fully connected (or nearly so)
  - Probabilistic termination acceptable

B. TSN Application

For IEEE 802.1AS Grand Master election:

- Arbitrary algorithm suits wired TSN with known topology
- Ring topology common in industrial automation
- Star topology in automotive Ethernet

C. Implementation Challenges

Several challenges were addressed during development:

- **Message Synchronization:** Out-of-order message delivery required buffering future messages until the corresponding round begins.
- **Neighbor Discovery:** Automatic neighbor detection via gate connections eliminates manual configuration.
- **RNG Determinism:** OMNeT++'s default RNG is deterministic per-run; hardware entropy was required for true randomness.
- **Leader Announcement Flooding:** In non-fully-connected topologies, the leader announcement must be flooded to reach all nodes.

D. Limitations and Future Work

Current limitations:

- No fault tolerance (crash failures)
- Synchronous model only
- Anonymous algorithm message complexity increases in sparse topologies due to flooding

Future directions:

- Add timeout-based failure detection
- Implement asynchronous variants using logical clocks
- Optimize anonymous algorithm for sparse topologies
- Add visualization of election progress

VII. CONCLUSION

This project successfully implemented and validated two leader election algorithms in OMNeT++ 6.3.0:

1) The flooding-based algorithm (`ArbitraryElection`) provides deterministic $O(D)$ round complexity for arbitrary connected networks. After exactly D rounds, all nodes converge on the maximum ID as the leader.
2) The randomized algorithm (`AnonymousElection`) achieves $O(\log n)$ expected rounds in anonymous networks through probabilistic symmetry breaking using true random bit selection.

The implementation addresses practical challenges including message synchronization, neighbor discovery, and ensuring true randomness through hardware entropy seeding. Experimental results closely match theoretical predictions, demonstrating both the correctness of implementations and the practical applicability of these algorithms for scenarios such as TSN Grand Master election.

The complete source code, including 12 simulation configurations across 5 network topologies, is available in the project repository. The modular architecture with base class inheritance enables easy extension for future research.

Acknowledgment

This work was completed as part of the Distributed Algorithm Design and Analysis course.

References

[1] S. Ghosh, *Distributed Systems: An Algorithmic Approach*, 2nd ed. Chapman & Hall/CRC Computer & Information Science Series, 2014, ch. 11.2.3–11.2.4.

[2] A. Varga, “OMNeT++,” in *Modeling and Tools for Network Simulation*, Springer, 2010.

[3] IEEE, “802.1AS-2020 – Timing and Synchronization for Time-Sensitive Applications,” 2020.