Neighborhood Surveillance Group Membership Protocol

For Distributed Drone Swarm Coordination

Project Documentation

December 9, 2025

Abstract

This report describes the design and implementation of a distributed group membership protocol for drone swarms, based on Cristian's neighborhood surveillance model (1991). The approach reduces monitoring overhead by having each node observe only its nearest neighbors, rather than all peers. This yields scalable fault detection while maintaining correctness under partial synchrony assumptions. In addition to core membership management, the implementation includes a visualization system, a configurable network simulator, and support for formation control. The resulting system provides a practical and extensible framework for experimentation in robust swarm coordination.

Contents

1 Introduction **3**

2 Background **3**
- 2.1 The Group Membership Problem . . . 3
- 2.2 Cristian's Insight . . . 3

3 Protocol Overview **3**
- 3.1 State and Neighbor Selection . . . 3
- 3.2 Heartbeats . . . 4
- 3.3 Three-Phase Reconfiguration . . . 4
  - 3.3.1 Phase 1: INIT . . . 4
  - 3.3.2 Phase 2: ACK . . . 4
  - 3.3.3 Phase 3: COMMIT . . . 4
- 3.4 Tie-Breaking Rules . . . 5

4 Network Simulation **5**
- 4.1 Latency Model . . . 5
- 4.2 Packet Loss . . . 5
- 4.3 Additional Impairments . . . 5

5 Visualization System **5**
- 5.1 Membership Visualization . . . 5
- 5.2 Formation Visualization . . . 6

6 Testing and Failure Scenarios **6**
6.1 Cascading Failures 6
6.2 Test Suite 6

7 Implementation Details **6**
7.1 Code Structure 6
7.2 Logging 6

8 Protocol Properties **6**
8.1 Safety 6
8.2 Liveness 7
8.3 Timing 7

9 Configuration **7**
9.1 Protocol Timing 7
9.2 Network Parameters 7

10 Usage Instructions **7**
10.1 Building 7

11 Future Work **8**

12 Conclusion **8**

1 Introduction

Coordinating a swarm of autonomous drones requires a reliable understanding of which nodes are currently active. Because drones may fail due to battery depletion, communication loss, or environmental hazards, establishing a consistent view of group membership is foundational to any higher-level behavior.

A naive solution—having each drone monitor all others—results in $O(n^2)$ heartbeat traffic and becomes impractical for larger swarms. Cristian's 1991 work demonstrated that a ring-based neighborhood-surveillance structure enables every failure to be detected while reducing per-node monitoring overhead to $O(1)$.

This project implements a proximity-based variant of Cristian's approach. Key extensions and differences include:

- Neighbors are selected using Euclidean distance rather than ring indices.
- Drones exchange positions during reconfiguration to ensure consistent neighbor assignment.
- A visualization system facilitates debugging and analysis.
- Formation control features are integrated for evaluation and demonstration purposes.

2 Background

2.1 The Group Membership Problem

Maintaining a correct set of active participants in a distributed system is challenging due to message delays, loss, and node failures. Distinguishing slow communication from genuine failure is inherently ambiguous under asynchrony.

Practical systems assume *partial synchrony*: messages arrive within an unknown but bounded delay. This assumption is suitable for drone networks, where communication characteristics are reasonably constrained.

2.2 Cristian's Insight

Cristian's neighborhood-surveillance model arranges nodes in a logical ring, assigning each node two monitors. A single failure is guaranteed to be detected by at least one neighbor. This reduces monitoring complexity while ensuring complete coverage. The tradeoff is a potentially longer propagation delay before the entire group converges on a consistent view.

3 Protocol Overview

3.1 State and Neighbor Selection

Each drone maintains:

- Its two geographically closest neighbors.
- Timestamps of the most recent heartbeat from each neighbor.
- Its current group membership identifier.
- Whether a reconfiguration process is active.

Proximity-based neighborhood selection improves communication reliability at the cost of dynamic neighbor changes as drones move.

3.2 Heartbeats

Every `heartbeatInterval` seconds, each drone sends:

```
HEARTBEAT {
    senderId,
    position,
    timestamp
}
```

If a heartbeat is not received within `heartbeatInterval + timeoutDelta`, the neighbor is suspected failed, and reconfiguration is initiated.

3.3 Three-Phase Reconfiguration

Reconfiguration proceeds through three phases. This ensures that all nodes converge not only a consistent view of membership but also a consistent neighbor set to fit the drone problem.

3.3.1 Phase 1: INIT

The detecting drone broadcasts:

```
RECONFIG_INIT {
    reconfigId,
    initiatorId
}
```

This message announces the start of a reconfiguration without presupposing the final member set.

3.3.2 Phase 2: ACK

Nodes willing to join the new configuration respond with:

```
RECONFIG_ACK {
    reconfigId,
    senderId,
    position
}
```

Including position data ensures consistent neighbor determination after commit.

3.3.3 Phase 3: COMMIT

After waiting for acknowledgments, the initiator broadcasts:

```
RECONFIG_COMMIT {
    reconfigId,
    initiatorId,
    numMembers,
    members: [{id, position}, ...]
}
```

Upon receiving the COMMIT message, nodes:

1. Install the new member list.
2. Update their group identifier.
3. Recompute neighbors using the supplied positions.
4. Resume normal monitoring.

3.4 Tie-Breaking Rules

Concurrent reconfiguration attempts are resolved using the following logic:

```
if (inReconfig) {
    if (newReconfigId < pendingReconfigId)
        return;

    if (newReconfigId == pendingReconfigId &&
        newInitiatorId >= currentInitiatorId)
        return;

    // Otherwise, switch to the newer reconfiguration
}
```

More recent `reconfigId` values dominate; ties are broken by initiator ID. This deterministic ordering ensures convergence.

4 Network Simulation

A configurable network simulator models realistic communication conditions.

4.1 Latency Model

$$\text{delay} = \text{baseLatency} + \alpha \cdot \text{load} + \beta \cdot \text{distance}$$

4.2 Packet Loss

$$P(\text{drop}) = \text{baseLossProb} + \gamma \cdot \text{load}$$

A burst-loss mode simulates temporary interference by introducing correlated packet drops.

4.3 Additional Impairments

- **Duplication**: Models non-idempotent network paths.
- **Jitter**: Adds timing variability.
- **Omissions**: Represents local processing delays or missed transmissions.

5 Visualization System

5.1 Membership Visualization

This mode displays:

- Alive and failed nodes.

- Neighbor relationships.
- Heartbeat and broadcast traffic.

System state indicators visualize consistency and failure detections.

5.2 Formation Visualization

A secondary mode renders formation transitions with smooth interpolation and color gradients. Although not required for the protocol, this mode aids demonstration and provides a clear illustration of swarm responsiveness to membership changes.

6 Testing and Failure Scenarios

6.1 Cascading Failures

A worst-case scenario involves sequential failure of neighbors faster than the detection timeout. While theoretically challenging, the protocol still guarantees that surviving nodes converge on a correct membership.

6.2 Test Suite

The test harness covers normal operation, isolated failures, adjacent failures, cascading failures, simultaneous failures, and operation under realistic network conditions. Tests use both deterministic and probabilistic network models.

7 Implementation Details

7.1 Code Structure

```
src/
  core/                 Main simulation loop and drone logic
  membership/           Membership protocol implementation
  network/              Network simulator
  control/              Formation controller
  visualization/        Rendering subsystem
  util/                 Logging utilities
```

The membership implementation is intentionally isolated to promote clarity and testability.

7.2 Logging

Logging is grouped by category (Heartbeats, Reconfig, Failures, Messages) and can be enabled or disabled at runtime.

8 Protocol Properties

8.1 Safety

1. **Agreement**: Nodes sharing the same group ID share the same member list.
2. **Validity**: Only nodes that send ACKs appear in a COMMIT message.

3. **Non-inclusion of failed nodes**: Failed nodes cannot contribute ACKs and therefore cannot appear in new membership lists.

8.2 Liveness

1. **Failure detection**: Missing heartbeats eventually trigger detection.
2. **Termination**: Every reconfiguration completes within a bounded number of rounds.

8.3 Timing

$$T \approx (h + \delta) + \Delta_1 + 2\Delta_2 + \Delta_1 \approx 4.5 \text{ seconds (default parameters)}$$

9 Configuration

All parameters are defined in `SimulationConfig.h` and adjustable via GUI.

9.1 Protocol Timing

Parameter	Default	Description
heartbeatInterval	2.0s	Heartbeat period
timeoutDelta	0.6s	Failure detection margin
deltaSmall	0.5s	Upper bound on message delay
deltaLarge	0.5s	ACK collection window
reconfigMinInterval	1.0s	Minimum spacing between reconfigs

9.2 Network Parameters

Parameter	Default	Description
baseLatency	0.05s	Minimum latency
baseLossProb	0.0	Baseline drop probability
enableBursts	false	Correlated loss model
omissionProb	0.0	Chance of skipped sends

10 Usage Instructions

10.1 Building

```
mkdir build && cd build
cmake ..
cmake --build .
./drone-show
```

Dependencies are automatically fetched by CMake during the first build.

11 Future Work

Potential extensions include:

1. Introducing a SUSPECT state to mitigate transient-network false positives.
2. Supporting k-miss failure tolerance.
3. Implementing a load balancing multiple concurrent group membership protocol.
4. Creating hierarchical group structures for long distance coordination.
5. Deploying the protocol on physical drone hardware.

12 Conclusion

This project implements a proximity-based neighborhood-surveillance group membership protocol for drone swarms. The system incorporates robust reconfiguration, a configurable network simulator, a visualization framework, and a comprehensive test suite. The results demonstrate that neighborhood-based monitoring is an effective and scalable approach for swarm coordination under realistic communication conditions.