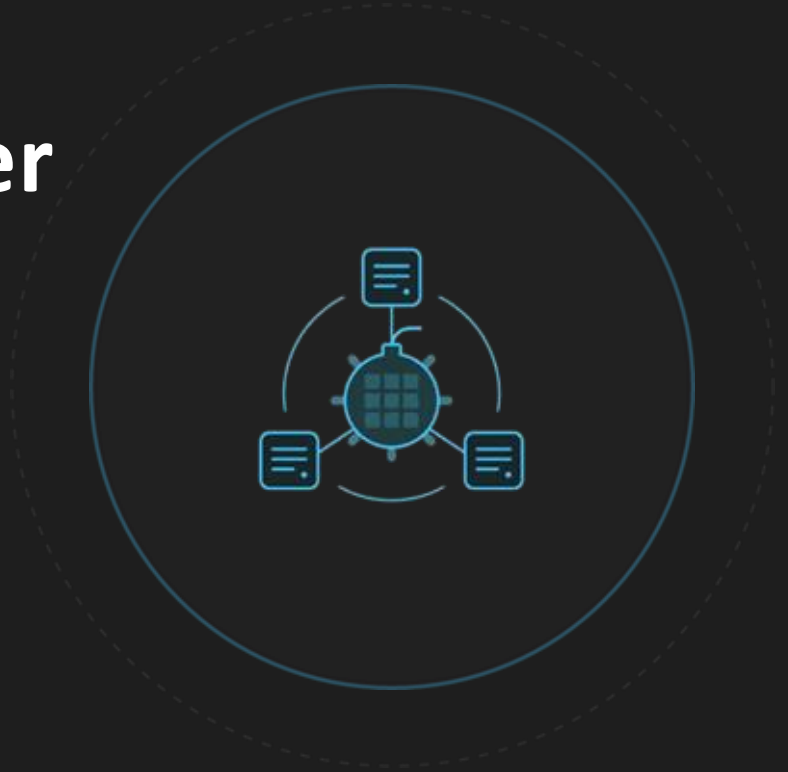
Distributed Systems Course

Distributed Minesweeper

A Fault-Tolerant, SMR-Based Multiplayer Architecture



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Nicolás Maire Bravo · **Iago López Lamas**

Concept & Motivation

Real-Time Cooperative Play

- Cooperative: all players share one live board
- Every click → immediate state change for everyone

Minimalist Architecture

- 100% volatile in-memory — no external database
- Pure Python TCP sockets — zero dependencies

Guarantees & Controls

- CP over AP: inconsistency = game is broken
- Mouse-driven UI: left-click reveals, right-click flags

CAP Theorem Decision

Consistency
All nodes, same board

Availability
Always return a response

Partition Tolerance
Survive network splits

We chose C + P — consistency first, always

Architecture Overview

Chain Topology

3-Node Fixed Chain

Leader Follower 1 Follower 2

Ports: :5000 (L), :6000 (F1), :7000 (F2)

Client Connections

Exclusive Access

Clients connect only to the active Leader node.

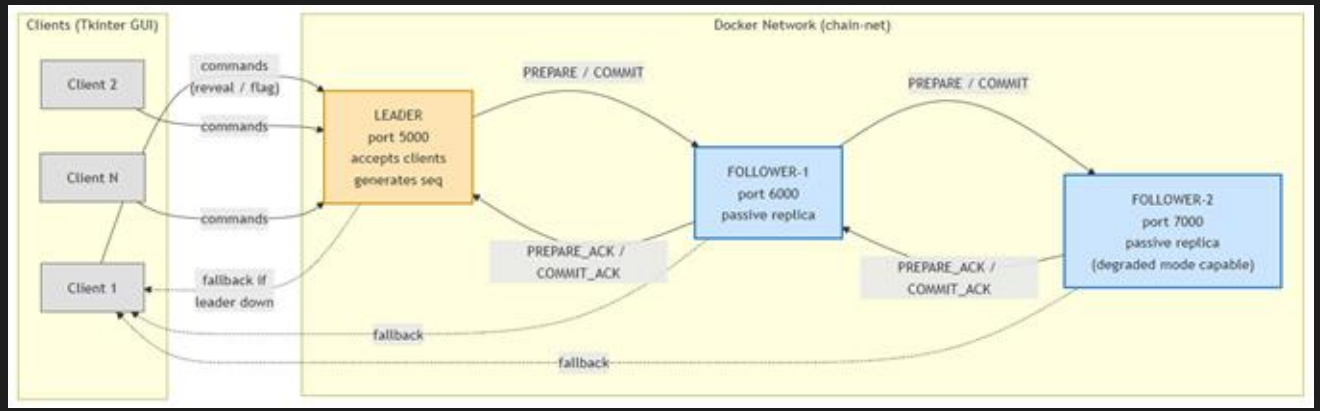
Fallback: Tries L F1 F2 in order

Infrastructure

Passive Replicas

Followers act as passive replicas with no client sockets.

Docker bridge + internal DNS discovery



Network Protocol & TCP Framing

THE TCP CHALLENGE

- TCP is a byte stream — zero message boundaries
- One send() may arrive split across multiple recv()
- Rapid clicks can coalesce into a single chunk

THE FRAMING SOLUTION

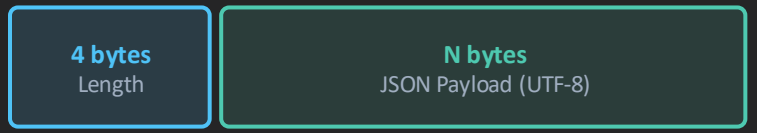
Use a 4-byte big-endian length prefix header

Guarantees exact message slicing regardless of network buffering or packet fragmentation.

IMPLEMENTATION

- `recvall(n)` loops until exactly n bytes accumulated
- Payload: **UTF-8 JSON** — native Python, trivial debug

Wire Frame Anatomy



`struct.pack('!I', len)` *prefix → big-endian, platform-independent*

```python
# Transmission logic (Python)
def send_msg(sock, data):
hdr = struct.pack("!I", len(data))
sock.sendall(hdr + data)

def recv_msg(sock):
raw_hdr = recvall(sock, 4)
n = struct.unpack("!I", raw_hdr)[0]
return recvall(sock, n)
```

Replicated State Machine (RSM) Logic

- **Seed Generation:** The leader generates the seed → all replicas build the same initial board.
- **Network Efficiency:** Only commands travel over the network, avoiding sending the full board state.
- **Client Flow:** Clients send their intentions directly to the leader.
- **Supported Actions:** Two-phase protocol → PREPARE / COMMIT

Command vs. Full State

COMMAND

```
{"action":"reveal",
"r":2, "c":3,
"seq":7}
```

≈ 50 bytes ✓

vs.

FULL STATE

16×16 Board

Matrix with mines, flags, revealed cells, and game states...

>> kilobytes ×

Same seed + same commands in the same order → all replicas converge

Two-Phase PREPARE / COMMIT Protocol

Phase 1 — PREPARE

1. Client sends reveal(r, c) to Leader
2. Leader assigns monotonic seq number
3. Leader sends PREPARE_COMMAND → F1
4. F1 stores as pending (not applied) → forwards → F2
5. F2 stores as pending → sends PREPARE_ACK → F1
6. F1 sends PREPARE_ACK → Leader

Phase 2 — COMMIT

1. Leader sends COMMIT_COMMAND → F1
2. F1 forwards COMMIT_COMMAND → F2
3. F2 applies command → sends COMMIT_ACK → F1
4. F1 applies command → sends COMMIT_ACK → Leader
5. Leader applies command (applies LAST — intentional)
6. Leader broadcasts updated board to all clients

Failover Mechanisms

Follower Crash

F1 or F2 ↓ **(Leader alive)**

- PREPARE ACK times out (5s)
- Leader rejects the operation
- No promotion — cluster shrinks
- Consistency preserved — op fails
- Client must retry manually

Leader Crash

Leader ↓ **(F1, F2 alive)**

- F1 detects closed TCP socket
- F1 discards pending ops
- F1 creates PromotedLeaderRuntime
- Client fallback list connects to F1
- Board returned at last committed seq

Leader + F1 Crash

Leader + F1 ↓ **(F2 only)**

- F2 detects no upstream socket
- F2 self-promotes to Leader
- Enters Degraded Mode (CP → AP)
- Accepts local writes without replica
- Any further crash kills session

Client Reconnection

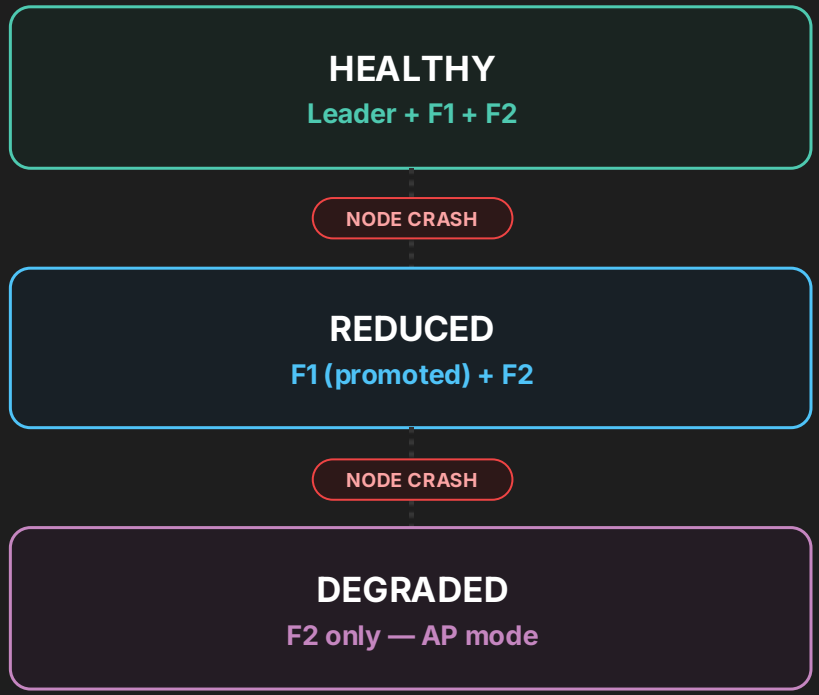
Automatic, transparent
~1s freeze, session resumes

Degraded Mode: CP → AP When Only F2 Survives

- Triggered when Leader + F1 both crash
- F2 detects no upstream socket: self-promotes
- Drops PREPARE/COMMIT: no chain left to walk
- Accepts local writes without replication
- Deliberate trade: availability over consistency
- Any further crash permanently kills the session

Healthy + Reduced = CP | Degraded = AP (scoped exception)

Degraded Mode is documented — not an oversight.



Implementation Details

Threading & Locks

- One thread per connected client socket
- Separate ACK listener in ChainReplicationManager
- FollowerRuntime main loop: dedicated thread
- threading.Lock on GameStateMachine + RoomManager
- Lock on seq counter prevents race conditions

Client — MVC Pattern

- Tkinter main loop must own the primary thread
- Daemon thread handles blocking socket recv()
- gui.after() safely pushes updates to main thread
- Controller decouples socket events from UI layer
- ConnectionState: tracks active socket + fallback index

Docker + X11 Forwarding

- 3 backend nodes: Docker bridge network
- GUI client also containerised — Linux only
- network_mode: host + DISPLAY environment var
- Mounts /tmp/.X11-unix → host X11 server
- macOS / Windows: client runs natively (X11 not portable)

Server-Side Seed Anti-Cheat

- Server generates board seed — never the client
- Client cannot know mine positions in advance
- Seed sent inside create_room PREPARE/COMMIT cmd
- Server is sole source of truth for board layout
- Cluster runs on closed Docker network (no TLS needed)

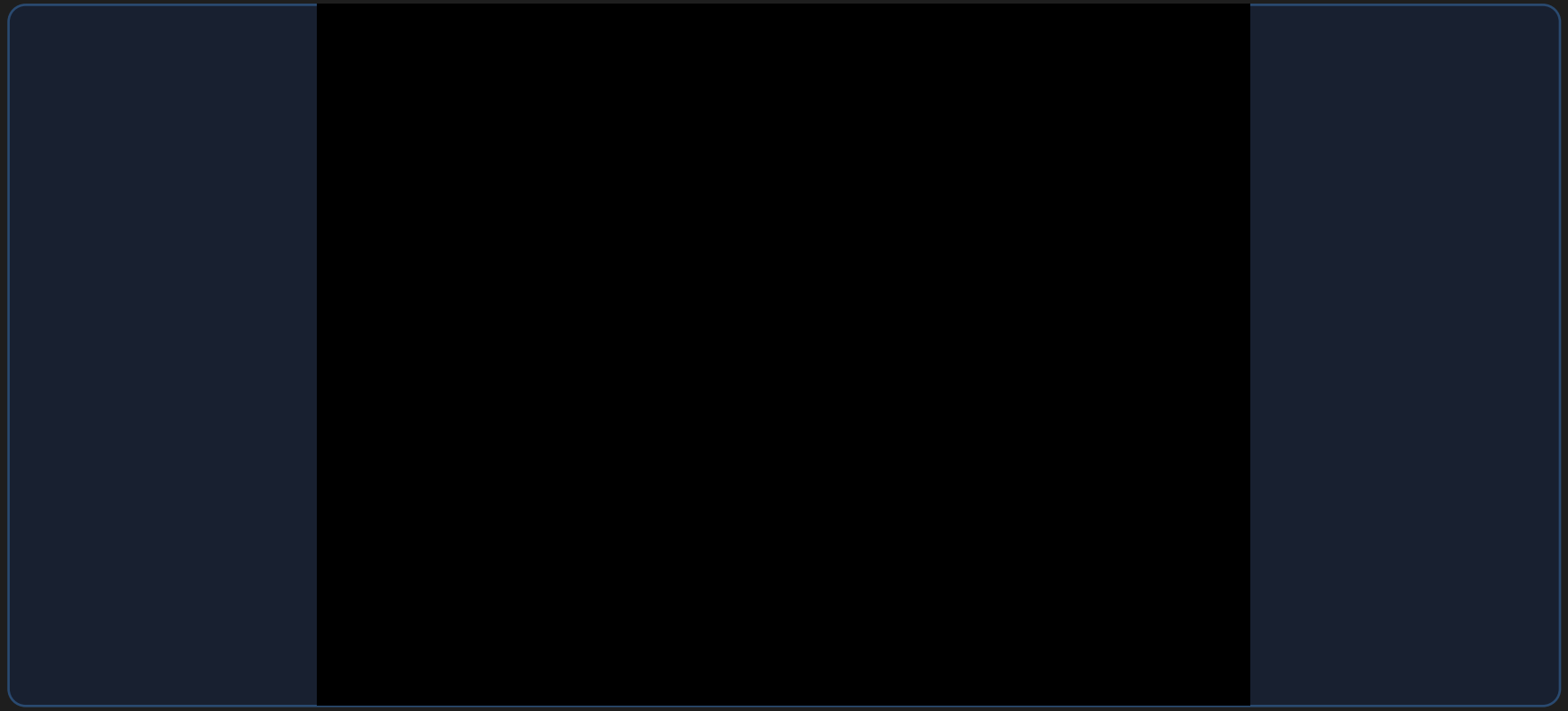
Validation & Testing

9	5	0
Total Tests	Test Modules	Mocks — Real Cluster

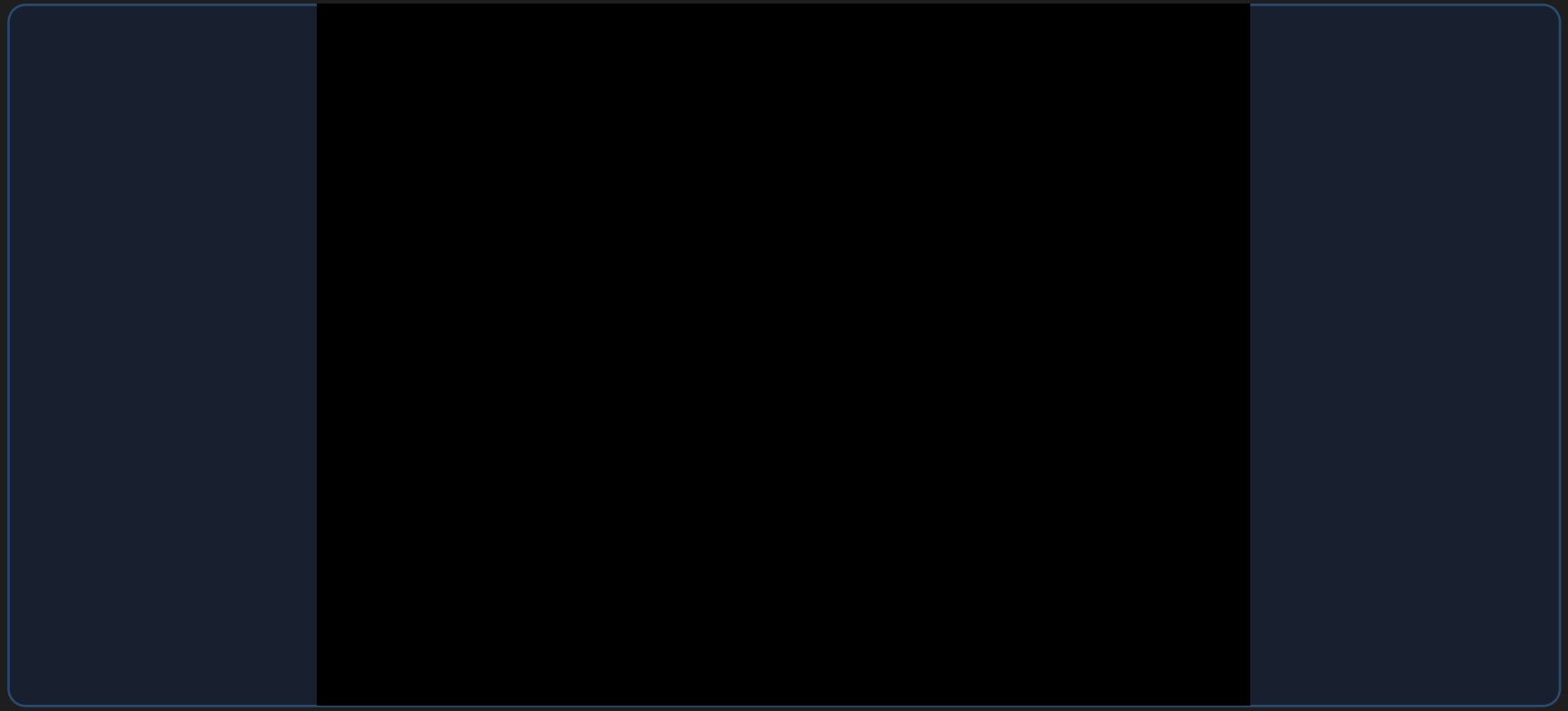
`test_model.py`	FR3, FR5	Game rules: first click safe, adjacent counters
`test_replication.py`	FR4, NFR4	All replicas converge — same board + seq
`test_failover.py`	FR7, NFR2	Leader crash → F1 promotes → cascade
`test_multiplayer_rooms.py`	FR1, FR3, NFR3	Room isolation, two clients same room
`test_consistency_failures.py`	NFR1	F2 unreachable → leader rejects op

pytest boots a real 3-node cluster via subprocess.Popen — no mocking, no stubs

DEMO: usual execution *(happy path)*



DEMO: failover execution



Future Works

Short-Term

- **Node Rejoin:** REJOIN msg + leader snapshot
- **Disk Persistence:** Save games dict after every COMMIT
- **Chain Length:** Configurable (e.g., 4+ nodes)
- **Failover:** Follower timeout mid-transaction & chain retry
- **Tolerating follower failures mid-transaction**

Long-Term Redesign

- **Multi-Room:** One leader + replication group per room
- **Coordinator Node:** Handles room creation & routing
- **Parallel PREPARE:** Quorum ACKs instead of chain
- **Consensus:** Raft / Paxos for proper leader election
- **Partitions:** Correct behaviour under network splits

Questions?

Thank you for your attention.

Nicolás Maire Bravo
nicolas.mairebravo@studio.unibo.it

Iago López Lamas
iago.lopezlamas@studio.unibo.it