

Distributed Bomberman

Sistemi Distribuiti — A.Y. 2024/25

Implementazione distribuita e fault-tolerant del classico gioco Bomberman

Gioele Santi · Giovanni Rinchuso

Python 3.10+

socket / threading

Pygame

TCP + JSON

Il gioco: concept e regole

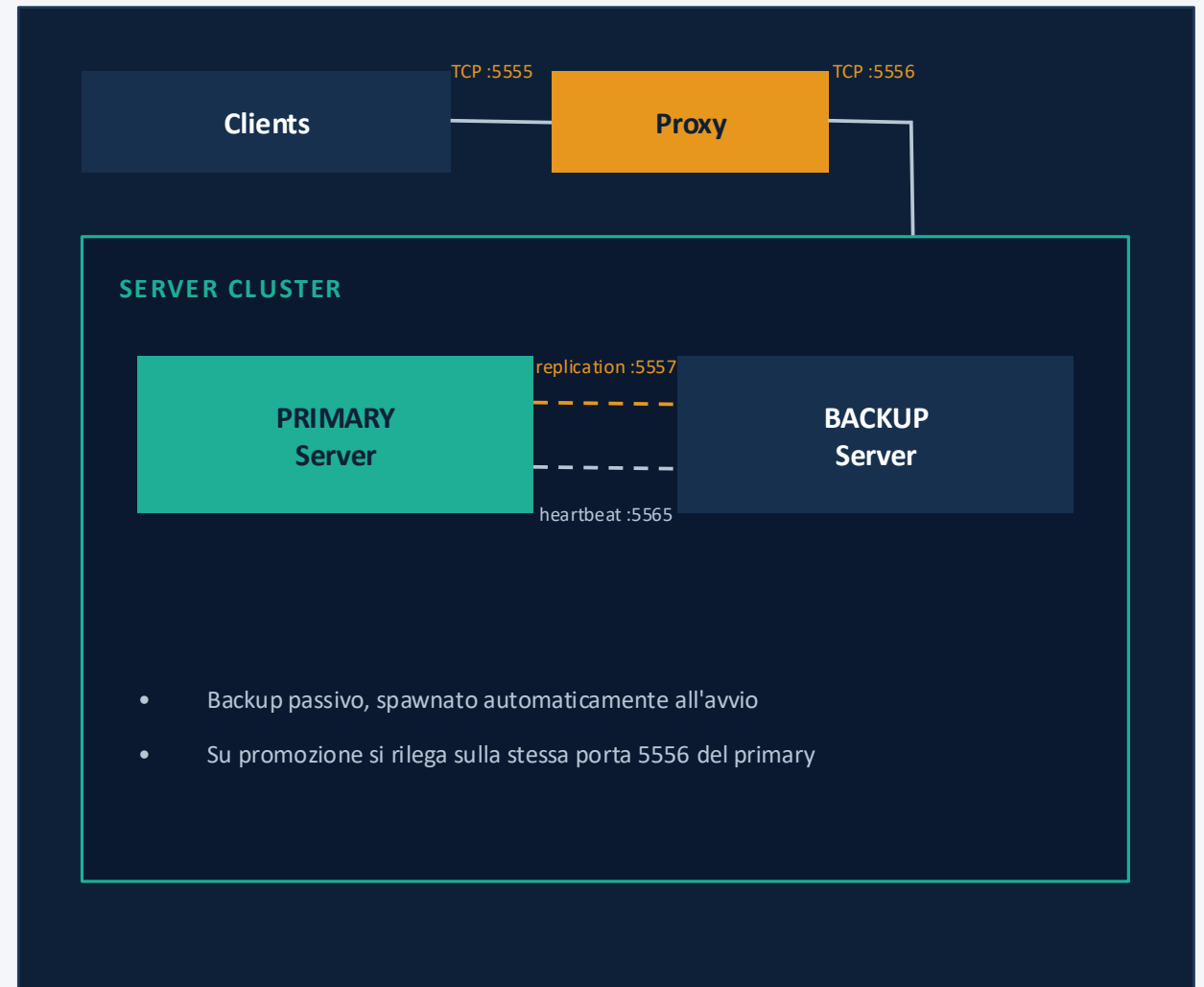
Bomberman in tempo reale, multiplayer, fault-tolerant — fino a 4 giocatori, spettatori illimitati.

1 Join	2 Lobby	3 Playing	4 Victory
Il client si connette al proxy su :5555. Riceve un nome casuale e un <code>session_id</code> univoco (UUID4).	Il primo a entrare diventa host. Con almeno 2 giocatori connessi, l'host preme Invio per avviare il match.	Movimento e bombe si applicano subito, in modo sincrono. Il tick a 10Hz gestisce timer, esplosioni, rigenerazione blocchi.	L'ultimo sopravvissuto vince (pareggio se muoiono tutti). Countdown di 5s, poi si torna in lobby in automatico.

Architettura: client-server con proxy trasparente

Modello Client-Server centralizzato

- Il server è l'unica Single Source of Truth: i client sono puri renderer
- Un solo componente (GameService) muta lo stato, protetto da un RLock
- Broadcast dello stato via TCP ogni 100ms a tutti i client connessi
- Il proxy nasconde topologia primary/backup: i client conoscono solo lui
- Niente logica di discovery lato client — un solo indirizzo statico



Protocollo di comunicazione

Tipi di messaggio		
UP / DOWN / LEFT / RIGHT	Client → Server	Movimento, applicato subito
BOMB	Client → Server	Piazza una bomba sulla cella corrente
START_GAME	Client → Server	Avvio match (solo host, ≥2 player)
JOIN_GAME	Spectator → Server	Richiesta slot player, se libero
CHAT:{msg}	Client ↔ Server	Messaggio broadcast a tutti
state snapshot	Server → Client	Proiezione JSON, ogni 100ms
RECONNECT:{id}	Client / Proxy → Server	Ripristino sessione post-drop
HEARTBEAT / ALIVE	Backup ↔ Primary	Probe di vita ogni 500ms

FRAMING (NEWLINE-DELIMITED)

```
buffer: bytes = b""
while b"\n" in buffer:
    line, buffer = buffer.split(b"\n", 1)
    handle(line.decode("utf-8"))
```

Scelte progettuali

- JSON: leggibile, facile da debuggare
- Uno snapshot versionato ("version": 1) per schema incompatibili

Trasporto

- TCP ovunque: consegna ordinata e affidabile
- Buffer di frame limitato a 64KB per peer

GameService & Finite State Machine

LOBBY

Attende ≥ 2 player; host preme Invio per iniziare

PLAYING

Movimenti e bombe applicati subito; tick gestisce timer

VICTORY

Vincitore registrato; countdown di 5s

countdown scade → torna a LOBBY

GameService: concorrenza

- Un thread per connessione client + un game-loop thread dedicato
- Ogni metodo pubblico è protetto da `@_synchronized` (`threading.RLock`)
- Il lock è rientrante: alcuni metodi ne richiamano altri internamente

Perché un solo lock coarse-grained

- A 10Hz la contesa sul lock è trascurabile
- Ogni mutazione pubblica diventa banalmente atomica
- Nessun rischio di deadlock da ordini di lock incoerenti

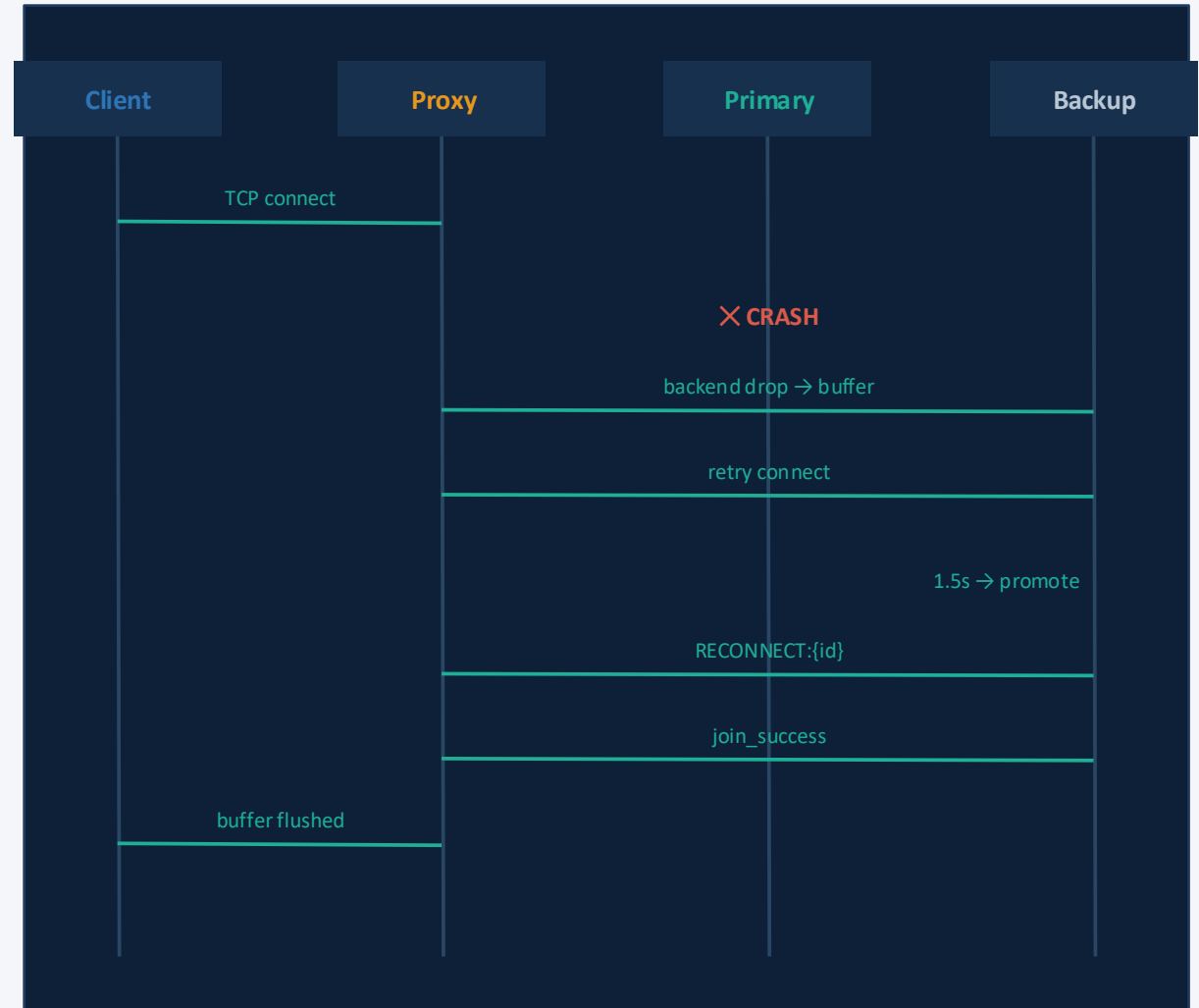
Il livello Proxy: relay trasparente

Come funziona

- Un thread per client, select() multipla client ↔ backend senza bloccare
- Intercetta il join_success per catturare il session_id
- Durante un failover bufferizza tutto ciò che il client invia
- Alla riconnessione invia RECONNECT:{id} e scarica il buffer

Benefici

- Lo switchover del server è invisibile al client
- Nessuna logica di discovery lato client
- La sessione sopravvive al cambio di processo fisico



Pipeline: il game tick (10Hz)



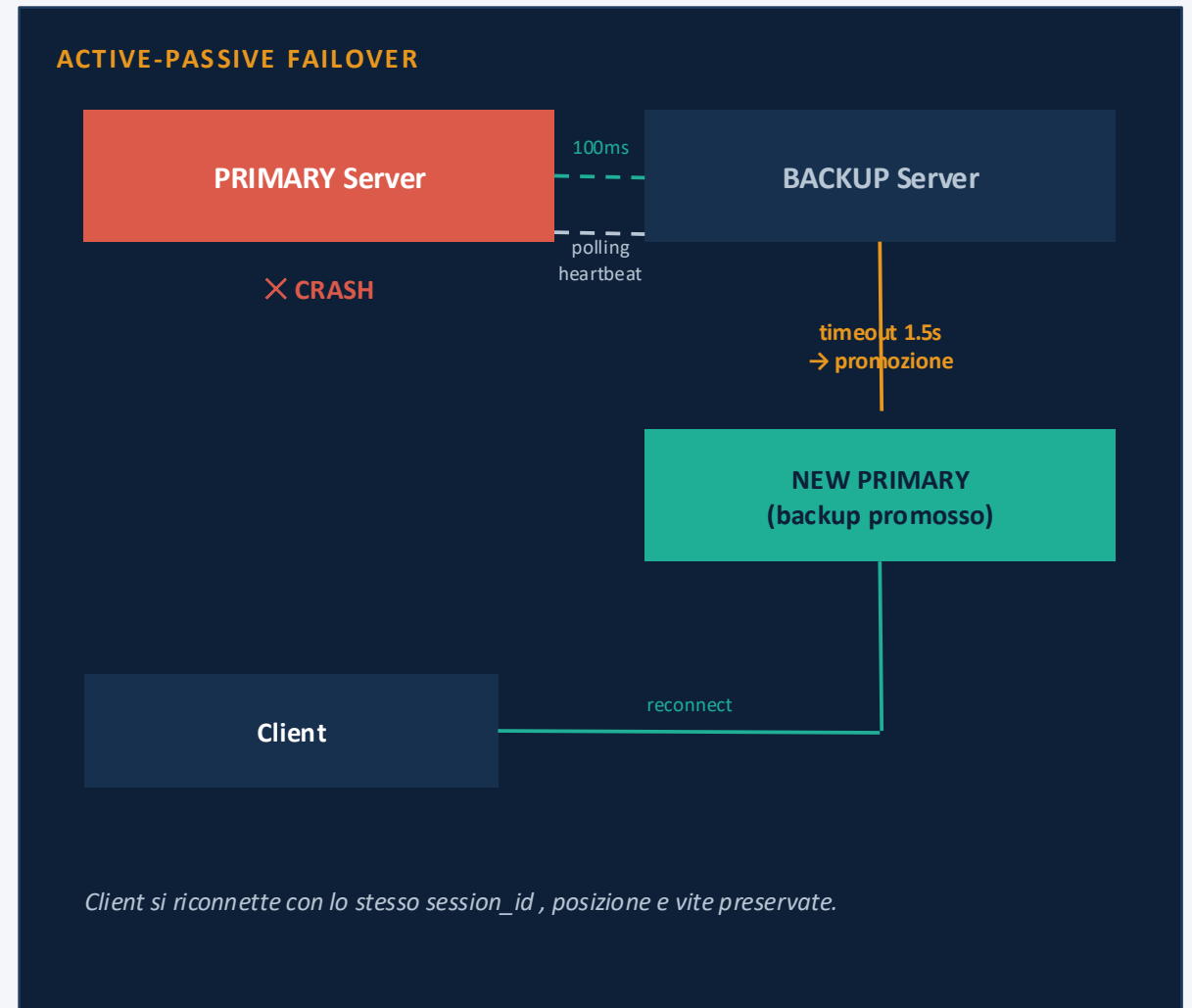
Meccanica	
Bomba	20 tick = 2.0s
Raggio esplosione	2, a croce, si ferma ai muri
Blocco colpito	Distrutto, consuma l'arm
Colpo da esplosione	-1 vita (3 vite totali)
Ultimo vivo	Vittoria, countdown 5s
Zero vivi	Pareggio

Target: un solo giocatore rimasto in vita → vittoria automatica

Fault Tolerance: replicazione primary-backup

Meccanismo heartbeat

- Il backup effettua il probe del primary ogni 500ms (TCP breve)
- Dopo 1.5s senza risposta, dichiara il primary morto
- Si promuove unilateralmente, nessun quorum richiesto
- Ripristina l'ultimo snapshot replicato e si rilega su :5556



Client resilience: il retry loop del proxy

Retry loop lato client

- Si attiva solo se cade il proxy stesso (un failover del primary resta invisibile)
- Ritenta l'indirizzo del proxy ogni 2 secondi
- Overlay "Connection Lost" mostrato finché non si riconnette
- Una volta riconnesso, rispedisce RECONNECT:{session_id}

NetworkManager

- Callback on_disconnected quando il socket si chiude a sorpresa
- BombermanClient.reconnect() sostituisce solo il NetworkManager
- Finestra e rendering restano intatti

Disconnessione rilevata

on_disconnected()

Overlay attivo

"Connection Lost", rendering in pausa

Retry loop

tenta il proxy ogni 2s, evento QUIT resta gestito

Riconnesso

nuovo NetworkManager, RECONNECT:{id} inviato

Successo

join_success(reconnected=True) → stato ripristinato

Interfaccia grafica: Pygame

Lobby Screen

- Elenco player e spettatori in tempo reale
- Host evidenziato, prompt "Press ENTER" solo per lui
- Tasto J per gli spettatori: richiesta slot player

Game Screen

- Griglia di gioco, bombe ed esplosioni animate
- Sidebar con vite di ogni player e chat integrata
- Stato aggiornato ad ogni broadcast (100ms)

Victory Screen

- Vincitore (o pareggio) e countdown di ritorno in lobby
- Tasto Invio per richiedere subito la rivincita

Controlli

- Frece per muoversi, Spazio per la bomba
- T per aprire la chat, Invio per inviare, Esc per annullare

Validazione: test automatici & acceptance

Unit testing (unittest / pytest)

- test_core.py (18) — logica pura: mappe, bombe, esplosioni, vittoria
- test_game.py (31) — modello e controller client, rete mockata
- test_controller.py (28) — comandi server e NetworkManager
- test_game_service.py (19) — ciclo di vita player/spettatore, tick
- test_state_codec.py (4) — round-trip della replicazione

Acceptance (manuale)

- Kill del processo in ascolto sulla porta 5556 a partita in corso
- Verifica: il backup si promuove, i client restano connessi

✓ Failover singolo

Nuovo PID su :5556, sessione e posizione preservate

✓ Doppio failover

Due promozioni consecutive verificate di fila

Conclusioni & Sviluppi Futuri

Room multiple

1

Più partite simultanee sullo stesso server

Power-up

2

Raggio bomba, bomba extra, velocità per arricchire il gameplay

Autenticazione

3

Login per punteggi persistenti tra sessioni

Test di failover automatici

4

Test di failover senza passi manuali

Grazie per l'attenzione