University of West Bohemia

Faculty of Applied Sciences

KIV/UPS Introduction to Computer Networks

Semester Project

Rock, Paper, Scissors online game

Student: Malakhov Leonid
Student ID: A22B0387P
Date: January 8, 2026

Contents

1 Introduction

This document describes the network protocol and implementation of an online multiplayer Rock-Paper-Scissors (RPS) game. The system implements a client-server architecture where multiple players can connect, create or join game rooms, and play against each other in real-time.

The game follows the standard Rock-Paper-Scissors rules: Rock beats Scissors, Scissors beats Paper, and Paper beats Rock.

2 Protocol Specification

2.1 Message Format

The protocol uses a line-based text format over TCP connections. Each message is a single line terminated by a newline character (`\n`).

General Format:

```
COMMAND [PARAM1] [PARAM2] ... [PARAMN]\n
```

- Commands and parameters are separated by single spaces
- Multi-word parameters (e.g., room names, nicknames) must be separated with "_"
- Empty lines are ignored
- Maximum line length: 512 bytes

2.2 Data Structures and Types

2.2.1 Player Representation

- **Nickname**: String, 3-32 characters, alphanumeric and underscores only
- **Token**: 16-character hexadecimal authentication token
- **Status**: `CONNECTED`, `SOFT_TIMEOUT`, `HARD_TIMEOUT`
- **State**: `ST_CONNECTED`, `ST_AUTH`, `ST_IN_LOBBY`, `ST_READY`, `ST_PLAYING`
- **Last Seen**: time_t
- **Replaced Flag**: Boolean marks that client was replaced with the new client
- **Invalid Message Streak**: Integer, 0-3
- **Room Id**: Integer, -1 or positive if belongs to any room

2.2.2 Room Structure

- **Room ID**: Integer, incrementation without the limits
- **Room Name**: String, 1-32 characters
- **Current Players**: Integer, 0-2
- **State**: `RM_OPEN`, `RM_FULL`, `RM_PLAYING`, `RM_PAUSED`
- **Players scores**: Integers
- **Players moves**: Chars
- **Round Start Time**: time_t
- **Awaiting Moves**: Boolean

2.3 Commands Reference

2.3.1 Client Commands

HELLO

```
HELLO "nickname"
```

Initial connection request. Server responds with authentication token.

- **Parameters**: nickname (1-20 chars, alphanumeric + underscore)
- **Response**: `OK token` or `ERR code message`
- **Validation**: Nickname uniqueness, length, character set, state

RECONNECT

```
RECONNECT token
```

Restore connection after timeout using previously issued token.

- **Parameters**: 16-char hexadecimal token
- **Response**: `OK` or `ERR code message`
- **Validation**: Token must match existing soft-timeout client

PING / PONG

```
PONG
```

Heartbeat response to server's `PING`.

LIST_ROOMS

```
LIST
```

Request the list of available game rooms.

- **Response**: Multi-line format:

```
R_LIST total_count
ROOM id name current_players/2 state
...
```

CREATE_ROOM

```
CREATE room_name
```

Create new game room.

- **Parameters**: room_name (1-32 chars)
- **Response**: `R_CREATED room_id` or error
- **Validation**: Name length, room availability, state

JOIN

```
JOIN room_id
```

Join existing room.

- **Response**: `R_JOINED` $room_i dorerror.Otherplayer, if present, willbenotifiedwith P_JOINED < n$
- `Validation: Id presence, room availability, state`

LEAVE

```
LEAVE
```

Leave current room.

- **Response**: `OK left` $_r oom < id_r oom > orerror.Otherplayer, if present, willbenotifiedwith P_L EF$
- `Validation: Id presence, room availability, state`

MOVE

```
MOVE move_choice
```

Submit move for current round.

- **Parameters**: `R`, `P`, or `S`
- **Response**: `ROUND_RESULT win|lose result your_move opp_move score1 score2`
- **Timeout**: 10 seconds per round, automatic loss if no move submitted
- **Validation**: Validness of move, states

2.3.2 Server Responses and Notifications

Success Responses

```
OK [data]
```

Error Responses

```
ERR code MESSAGE
```

Error codes:

- **100**: `BAD_FORMAT` - Wrong format of line (argument is missing etc.)
- **101**: `INVALID_STATE` - Client's state does not allow this action
- **104**: `UNKNOWN_ROOM` - Room was not found
- **106**: `ROOM_WRONG_STATE` - Target room is not opened
- **110**: `cannot_reconnect_now` - Wrong token or client with token is present
- **404**: `NOT_FOUND` - Room or token not found
- **409**: `ALREADY_IN_ROOM` - Client already in a room
- **500**: `SERVER_ERROR` - Internal server error

2.4 Protocol Flow

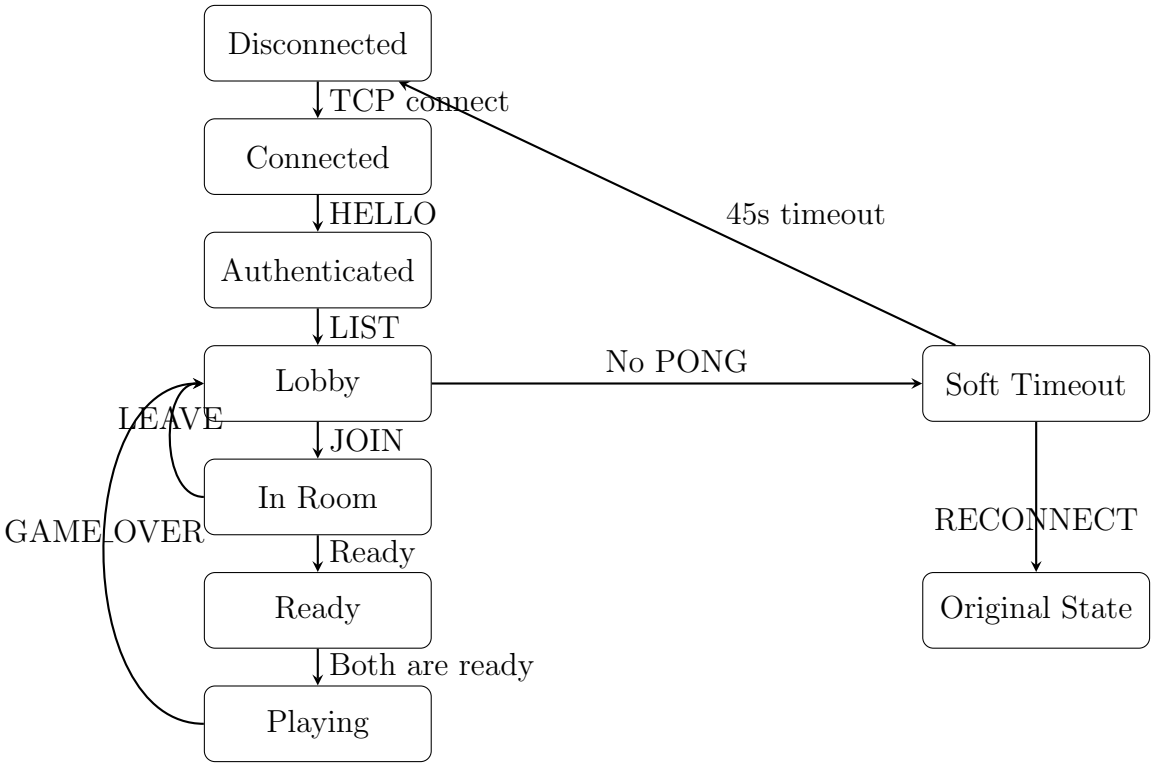

Figure 1: Client Connection State Diagram

2.4.1 Connection and Reconnection Flow

Initial Connection:

1. Client establishes TCP connection and gets ST_CONNECTED state.
2. Client sends `HELLO "nickname"`
3. Server validates nickname and responds with `WELCOME token`
4. Client stores token for potential reconnection
5. Client transitions to ST_AUTH state

Heartbeat Mechanism:

1. Server sends `PING` every 3 seconds
2. Client must respond with `PONG` within 6 seconds
3. If no response, server marks client as SOFT_TIMEOUT
4. Client connection is closed, but game and client's state preserved

Reconnection Procedure:

1. Client detects connection loss
2. Client re-establishes TCP connection
3. Client sends `RECONNECT token`
4. Server validates token and restores session:
    - Updates file descriptor
    - Resets timeout state to CONNECTED
    - Preserves room and game state
5. If token invalid or 45s hard timeout expired: `ERR 110`

Hard Disconnect Scenarios:

- 45 seconds elapsed since soft timeout without reconnection
- Client sends invalid token
- More than 3 consecutive malformed messages
- Client closes application.

When hard disconnect occurs:

- Client removed from all data structures
- If in game room: opponent notified via $\texttt{G}_E NDopp_l or \texttt{P}_L EFTnickname$
- `Room returned to waiting state if had` $\texttt{RM}_F ULLanddestroysif\texttt{RM}_P LAYING$

2.5 Input Validation and Constraints

Field	Constraint	Error Code
Nickname	3-32 chars, alphanumeric + _	100
Room Name	3-32 chars	100
Move Choice	R, P, or S	100
Token	16 hex chars	110 and Hard disconnect
Room ID	existing room	104
Max Clients	<64 total connections	200
Invalid Messages	<5 consecutive errors	Hard disconnect

3 Implementation Overview

3.1 Architecture

The system implements a classic client-server architecture:

- **Server**: C language, single-threaded accept loop with pthread worker threads
- **Client**: Java 8+ with Maven, event-driven GUI using JavaFX
- **Communication**: TCP sockets with line-based text protocol

3.2 Server Design

3.2.1 Module Decomposition

The server is organized into the following C modules:

- `server.c/h`: Main entry point, socket initialization, accept loop
- `client.c/h`: Client connection management, registration, timeout handling
- `room.c/h`: Room lifecycle, round management
- `commands.c/h`: Protocol command parsing and execution
- `send_line.c/h`: Thread-safe message transmission utility
- `game.c/.h`: Game management for the room

3.2.2 Threading Model

Main Thread:

- Listens on TCP port (default 2500)
- Accepts new connections
- Spawns worker thread per client
- Detaches threads for automatic cleanup

Client Worker Threads:

- Read commands from client socket (blocking I/O)
- Parse and execute commands via `handle_line()`
- Terminate on connection close or invalid message streak
- Clean up resources on exit

Timeout Monitor Thread:

- Dedicated thread checks timeouts every 200ms
- Enforces 6-second soft timeout (no PONG received)
- Enforces 45-second hard timeout (no reconnection)
- Enforces 10-second round timeout in games
- Sends PING messages every 3 seconds

3.2.3 Concurrency Control

All shared state is protected by a single global mutex `global_lock`:

```
pthread_mutex_t global_lock;
client_t *clients[MAX_CLIENTS];  // Protected
room_t rooms[MAX_ROOMS];         // Protected
```

3.3 Client Design

3.3.1 Key Components

NetworkManager:

- Manages TCP socket connection
- Separate executor threads for send/receive operations
- Buffered I/O with automatic flushing
- Connection state tracking

ProtocolHandler:

- Parses incoming server messages
- Emits events via EventBus
- Assembles multi-line responses (e.g., ROOMS_LIST)
- Validates message format

ReconnectionManager:

- Detects connection failures

- Implements exponential backoff retry strategy
- Stores authentication token for reconnection
- Automatic RECONNECT command transmission

EventBus:

- Decouples network layer from UI layer
- Publishes server events to subscribed listeners
- Supports type-safe event filtering
- Thread-safe event delivery

3.3.2 UI Architecture

The client uses a state-based UI with four screens:

1. **ConnectionUI**: Server connection and nickname entry
2. **LobbyUI**: Player list and navigation to rooms
3. **RoomsUI**: Room creation and browsing
4. **GameUI**: Active game with move submission and score display

UI updates are event-driven: network events trigger UI state transitions via EventBus subscriptions.

3.3.3 Concurrency Model

- **EDT (Event Dispatch Thread)**: All Swing UI updates
- **Network Send Thread**: Non-blocking command transmission
- **Network Receive Thread**: Blocking socket read, event publishing
- **Reconnection Thread**: Background retry logic

Thread synchronization via:

- `SwingUtilities.invokeLater()` for UI updates
- Synchronized collections in EventBus
- Atomic state flags in ReconnectionManager

4 Build and Deployment

4.1 Requirements

Server:

- GCC 4.1 or later
- Linux operating system
- Optional: Make

Client:

- Java JDK 8 or later
- Apache Maven 3.6+
- Any OS with Java support

4.2 Compilation

Server (using Make):

```
cd server
make
# Produces executable: ./server
```

4.3 Execution

Start Server:

```
./server [IP] [PORT]
# Example: ./server 0.0.0.0 2500
# Default if arguments are not present: 0.0.0.0:2500
```

Start Client:

```
cd client
mvn javafx::run
# Buildes and runs application.
```

Start Client (using script):

```
cd client
./client
```

4.4 Configuration

Server constants (defined in `server.h`):

- `MAX_CLIENTS = 64`
- `MAX_ROOMS = 32`

- `CLIENT_TIMEOUT_SOFT = 6s`
- `CLIENT_TIMEOUT_HARD = 45s`
- `ROUND_TIMEOUT = 10s`
- `PING_INTERVAL = 3s`

5 Conclusion

This implementation provides a robust online Rock-Paper-Scissors game with comprehensive connection management and fault tolerance. The protocol supports:

- Reliable client reconnection after network failures
- Concurrent game rooms with independent state
- Automatic timeout detection and recovery
- Scalable architecture supporting 64 simultaneous clients

Achieved Results:

- Full RPS game implementation with lobby and matchmaking
- Stable TCP protocol with heartbeat mechanism
- Clean separation of network, game logic, and presentation layers

Known Limitations:

- Single global mutex may become bottleneck under high load
- No persistent storage (games lost on server restart)
- Fixed maximum of 32 clients and 64 rooms
- No spectator mode or history functionality