Project 2 - Final Report

Project Team

- **Brooks Jackson**
- **Eric May**
- **Pierce Conway**

Table of Contents

System Design

Architecture Overview

```
┌───────────────────────────────────────────────────────────────┐
│                      GAME SERVER                              │
│  ┌─────────────┐    ┌─────────────┐      ┌─────────────────┐  │
│  │ Main Server │───►│ UDP Listener│      │  Question Bank  │  │
│  └─────────────┘    └─────────────┘      └─────────────────┘  │
│         ▲                                                     │
│         │                                                     │
│         ▼                                                     │
│  ┌─────────────┐                                              │
│  │ TCP Listener│                                              │
│  └─────────────┘                                              │
└───────────────────────────────────────────────────────────────┘
                   ▲                     ▲
                   │                     │
          TCP      │            UDP      │
        ┌──────────┘        ┌────────────┘
        │                   │
        ▼                   ▼
┌─────────────┐    ┌─────────────┐    ┌─────────────┐
│  Player 1   │    │  Player 2   │    │  Player 3   │
└─────────────┘    └─────────────┘    └─────────────┘
```

The system follows a client-server architecture with:

- **Centralized Server**: Manages game state, questions, and player interactions
- **Multiple Clients**: Each player connects via their own client instance
- **Dual Protocol Communication**:
  - TCP for reliable question/answer transmission
  - UDP for fast buzzer responses

Server Design

Components

1. **Main Server Thread**

- Accepts new client connections
- Manages game lifecycle (20 questions)
- Coordinates between all components

2. **ClientThread (Per Client)**

- Dedicated thread per connected client
- Handles all TCP communication:
    - Question delivery
    - Answer collection
    - Score updates
    - Game state notifications

3. **UDPThread (Single Instance)**

- Listens for buzzer presses from all clients
- Maintains synchronized buzz queue
- Handles out-of-order UDP packets

4. **QuestionBank**

- Manages pool of 20 questions
- Loads questions from text file
- Tracks current question

Key Data Structures:

- `ConcurrentHashMap<Integer, ClientThread>`: Thread-safe active client list
- `ConcurrentLinkedQueue<Integer>`: Ordered buzz queue
- `ConcurrentHashMap<Integer, Integer>`: Player scores

Client Design

Components:

1. GUI Layer (Swing)

- Question/options display
- Interactive controls (Poll/Submit buttons)
- Score/timer visualization

2. Network Layer

- TCP Connection: Persistent server connection
- UDP Socket: For buzzer presses
- Message handlers for server communication

3. Game State Manager

- Tracks current question
- Manages answer eligibility
- Handles timer countdowns

Network Protocol

TCP Messages (Reliable Delivery)

Message Type	Direction	Purpose	Payload
QUESTION	$S \rightarrow C$	Deliver new question	Question object
ACK	$S \rightarrow C$	Confirm buzzer success	None
NACK	$S \rightarrow C$	Reject buzzer attempt	None
CORRECT	$S \rightarrow C$	Right answer	None
WRONG	$S \rightarrow C$	Wrong answer	None
TIMEOUT	$S \rightarrow C$	Answer timeout	None
SCORE_UPDATE	$S \rightarrow C$	Broadcast scores	Map<ClientID, Score>

Message Type	Direction	Purpose	Payload
GAME_OVER	S → C	End game signal	None
PLAYER_ANSWER	C → S	Submit answer	PlayerAnswer object

UDP Messages (Fast Buzzer)

Message Type	Direction	Purpose	Payload
BUZZ	C → S	Buzzer attempt	None

Implementation Details

Concurrency Control

Server-Side:

- Thread pool for client connections
- Synchronized access to shared buzz queue
- Atomic score updates
- Volatile flags for game state

Client-Side:

- Swing event dispatch thread
- Separate thread for TCP message listening
- Thread-safe GUI updates via `SwingUtilities.invokeLater()`

TCP/UDP Communication

TCP Implementation:

- Persistent sockets per client
- Object streams for serialization
- Heartbeat detection for disconnected clients

UDP Implementation:

- Timestamp-based ordering
- Client IP verification

- Queue management with thread safety

GUI Implementation

Key Features:

- Responsive layout with leaderboard
- Visual timer countdown
- Disabled states during inappropriate times
- Immediate feedback for answers

Event Handling:

- Poll button → UDP "buzz"
- Submit button → TCP answer
- Radio buttons → Answer selection

Game Logic

Question Flow:

- Server broadcasts question (TCP)
- 15-second buzz period (UDP)
- First buzzer gets 10s to answer (TCP)
- If correct: +10 points, next question
- If wrong/timeout: -10/-20, next buzzer
- Repeat until correct answer or queue empty
- After 20 questions: end game

Scoring Rules:

- Correct answer: +10
- Wrong answer: -10
- Timeout: -20
- First correct answer wins points

Test Cases

The following is a list of the test cases we ran on our code. We wanted to go through an extensive testing process to make sure **all** of the project requirements were fulfilled.

Category	**Test Case**	**Test Method**	**Expected Result**	**Status**
Connection	Single client connects	Start server, connect 1 client	Client joins successfully	✅
	Multiple clients connect	Connect 5 clients simultaneously	All clients join with unique IDs	✅
	Client disconnects/ reconnects	Disconnect client during game	Server detects disconnect	✅
Game Flow	Full 20-question game	Play complete game	All questions delivered in order	✅
	No buzzes received	Let buzz timer expire	Server moves to next question	✅
	First buzz gets priority	3 clients buzz, verify order	First buzzer gets ACK	✅
Answer Handling	Correct answer submission	Client selects right option	+10 points, correct message	✅
	Wrong answer submission	Client selects wrong option	-10 points, wrong message	✅
	Answer timeout	Don't answer within 10s	-20 points, timeout message	✅
Scoring	Score synchronization	Compare client/ server scores	Scores match exactly	✅
	Negative scores	Force wrong answers	Handles negative values	✅
	Multiple correct rounds	Test consecutive right answers	Score accumulates properly	✅

Installation and Usage

Prerequesites

- Java Runtime Environment (JRE) 17 or later
  - You can verify you have this installed by running `java --version` in your terminal

Installation

How to play

Configuration

- **Set the correct IP address** in `config.txt` (change from 127.0.0.1 to server's real IP):
  - **Windows**:

    ```
    ipconfig | findstr "IPv4"
    ```

  - **macOS/Linux**:

    ```
    ifconfig | grep "inet " | grep -v 127.0.0.1
    ```

    or

    ```
    hostname -I
    ```

- **Firewall Commands** (run as admin/root) - optional but useful if you are having trouble connecting to the server:
  - **Windows** (allow ports):

    ```
    New-NetFirewallRule -DisplayName "TriviaGame" -Direction Inbound -Protocol TCP -Lo
    New-NetFirewallRule -DisplayName "TriviaGame" -Direction Inbound -Protocol UDP -Lo
    ```

  - **macOS**:

    ```
    sudo /usr/libexec/ApplicationFirewall/socketfilterfw --add /path/to/trivia-server.
    sudo /usr/libexec/ApplicationFirewall/socketfilterfw --unblockapp /path/to/trivia-
    ```

  - **Linux** (UFW):

```
sudo ufw allow 7000/tcp
sudo ufw allow 7001/udp
```

- **macOS Security Bypass** (first run):
  i. Right-click the `.jar` file → "Open"
  ii. When blocked message appears:
     - Go to System Settings → Privacy & Security
     - Click "Open Anyway" under Security
     - Confirm with password/Touch ID
     - Re-open the `.jar` file

Server admin

- Run `trivia-server.jar`
- Monitor connections in console

Players

- Launch `trivia-client.jar`
- Use interface:
  - Poll button: Buzz in to answer the question (UDP)
  - Radio buttons: Select the correct answer
  - Submit: Confirm the answer (TCP)

Game flow

- 20 Questions auto-progress
- Real-time leaderboard updates
- Winner announced after final question

Troubleshooting

Issue	Solution
"Port in use"	Change ports in `config.txt`
Connection timeout	Verify server IP/firewall
Missing questions	Check `questions.txt` format