Ourhood: Real-Time Streaming Quiz Game

- Quang Phung, Long Bui -

I) Introduction / Overview

Ourhood is a real-time multiplayer trivia game that allows users to participate in fast-paced quiz competitions, either with friends or random players. Designed using Java, Ourhood provides a synchronized, responsive, and interactive experience. The game leverages Java's networking capabilities, particularly the Socket and ServerSocket classes, to implement reliable communication via Transmission Control Protocol (TCP). The server orchestrates all gameplay logic, while clients connect to play and receive updates in real-time.

The game features multiple-choice questions, time-limited answers, and a dynamic point system. Hosts can personalize quiz sessions by choosing categories, question count, and time per question. Players compete for the highest score, with correct answers yielding points and incorrect or missed responses earning none.

II) System Architecture

a) Protocol & Communication Model

The game uses **Transmission Control Protocol (TCP)** to ensure reliable delivery of data between the server and connected clients. TCP guarantees that messages are delivered in order, without loss or duplication, which is essential for maintaining consistent game state and fair gameplay.

- **Application Layer (Main Layer):** The Application Layer is the topmost layer of the TCP/IP stack and provides the interface through which users and applications access network services [1].
- **Supporting Layers:** The Transport Layer, using TCP, ensures the reliable and ordered delivery of data between devices [2]. The Internet Layer is responsible for routing this data across networks, using IP addresses to identify both the source and destination devices [3]. At the lowest level, the Network Access Layer manages the transmission of data over physical media and handles the communication between devices on the same network [4]. Together, these layers work to ensure accurate and efficient data exchange across interconnected systems [5].

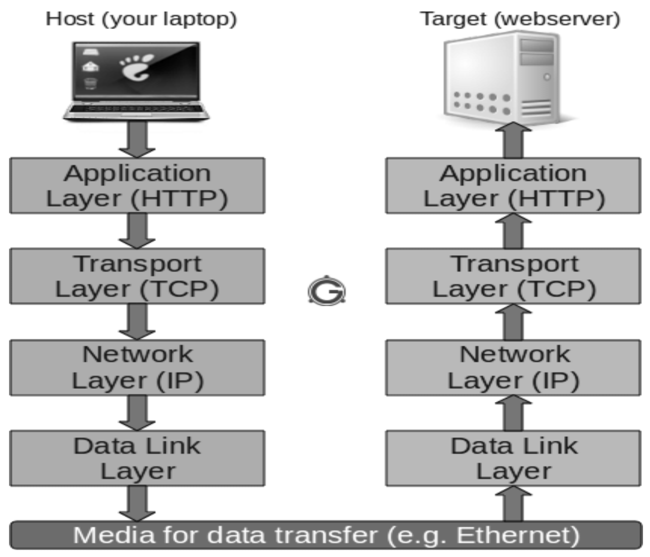


Fig. 1: Ourhood interacts directly with the application layer and data will be transferred along lower layers.

b) Client-Server Architecture

Ourhood uses a classic **Client-Server architecture** where a central server coordinates all gameplay activities, including question distribution, score tracking, and client synchronization.

- **Server**: Controls the game logic, distributes questions, validates answers, tracks player scores, and manages synchronization between clients.
- **Host**: A special type of client with elevated privileges to configure the quiz session before starting
- **Clients**: Regular players who connect to the server, receive questions, and send back answers

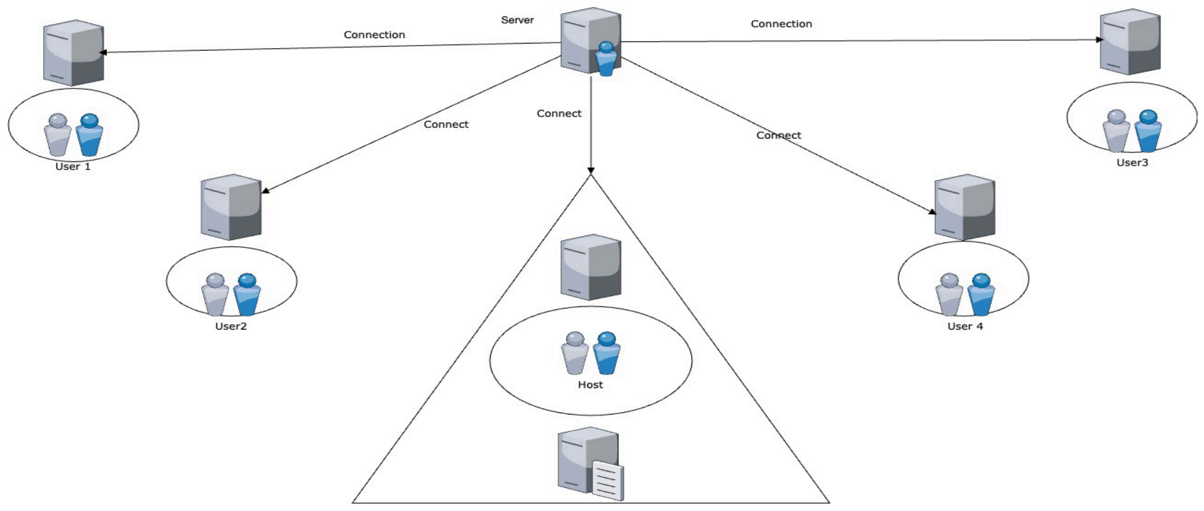
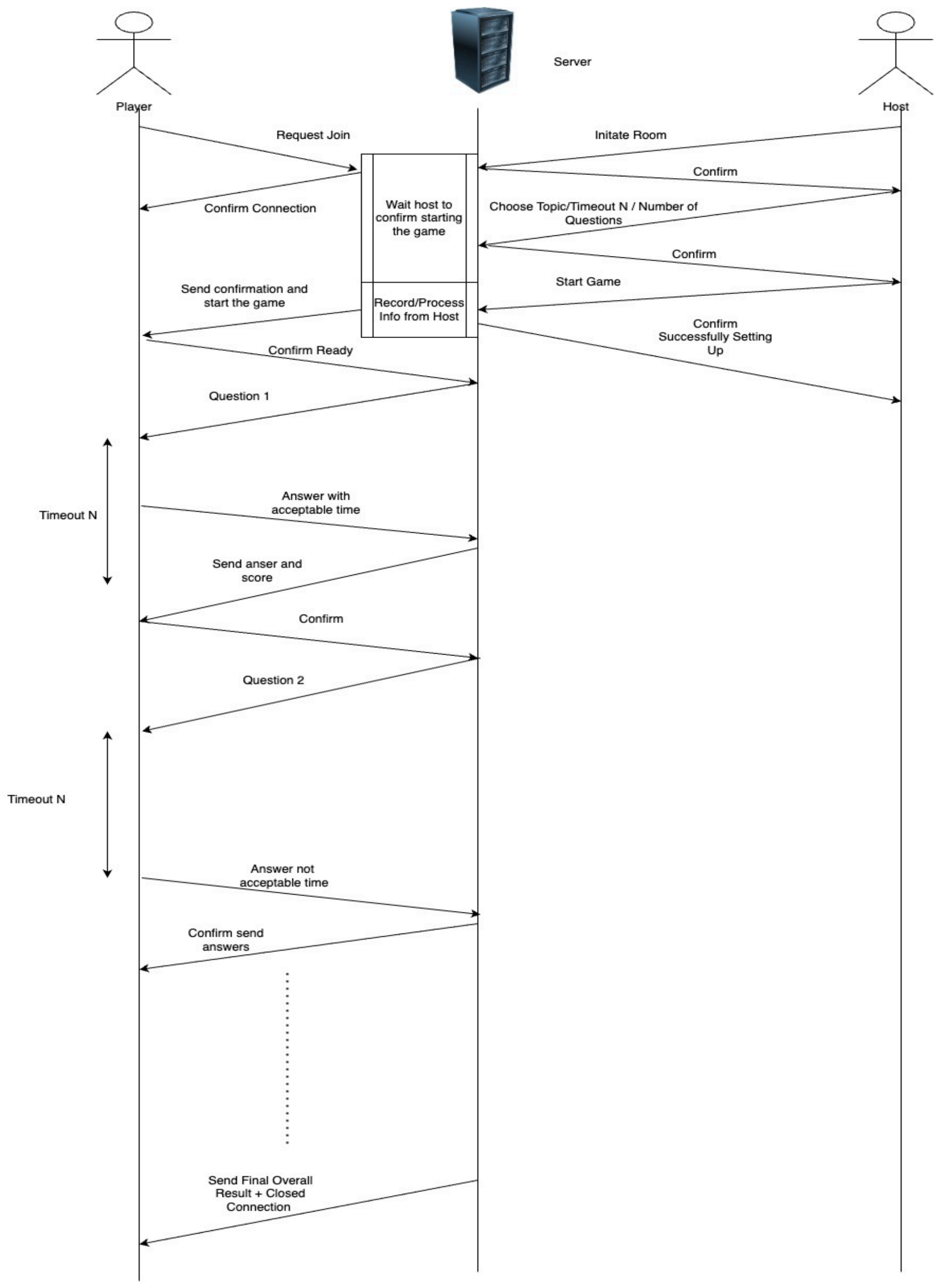


Fig. 2: Client-Server Model being implemented by Kahoot where both users and host connected to a server



Player
Server
Host
Request Join
Initate Room
Confirm
Confirm Connection
Wait host to confirm starting the game
Choose Topic/Timeout N / Number of Questions
Confirm
Start Game
Send confirmation and start the game
Record/Process Info from Host
Confirm Successfully Setting Up
Confirm Ready
Question 1
Timeout N
Answer with acceptable time
Send anser and score
Confirm
Question 2
Timeout N
Answer not acceptable time
Confirm send answers
Send Final Overall Result + Closed Connection

c) Message Exchange

In a multiplayer trivia game session, the client first **establishes a connection with the server** to join the game. Upon successful connection, the server **sends a welcome message to confirm the handshake**. Next, the host configures the game and **sends quiz metadata** to the server. Once the game begins, the server **broadcasts each question** to all connected clients, who then **submit their responses**. The server **validates the answers and updates player scores** accordingly. At the end of the game, the server **sends the final scores to all participants.**

Fig. 4: Messages Requests throughout a game

III) System Implementation

A) Game Model - Game Frame Setup

The code defines two classes: Question, which represents a multiple-choice trivia question with its options, correct answer, and topic; and Game, which manages a list of such questions, tracks the current question and answer, number of players, and question duration. Together, they provide the structure for a multiplayer trivia game by organizing question flow and game state. Important events included:

getQuestions: *Returns list of all game's questions.*

- getNumberOfQuestions – *Returns total count of game questions.*
- getNoPlayers – *Returns the number of players in the game.*
- updateNoPlayers – *Increments player count by one automatically.*
- gerDur – *Returns time limit per game question.*

B) Client Model - Time Out Management

The Client class establishes a TCP connection to a server, receives game configuration (like question duration and count), and handles interactive gameplay where the user answers timed multiple-choice questions. It reads each question from the server, collects the user's response within a time limit, sends the answer back, and displays round-by-round and final results. Each session will be displayed as:

```
while (System.currentTimeMillis() - startTime < questionDuration) { # set time limit
        if (inFromUser.ready()) {
                answer = inFromUser.readLine();      # read user answer
                System.out.println("You entered: " + answer);
                break;
}}
```

C) Server Model

The `Server` class manages the multiplayer quiz game backend by handling the host setup, accepting player connections, selecting random questions, and coordinating gameplay. It ensures proper question delivery, score tracking, and game flow, while also calculating and broadcasting the final results to all players once the game is over. Important Events Included:

- **initializeGame** – *Sets up game questions, timer, players list.*
- **acceptPlayerConnections** – *Waits for and accepts incoming player connections.*
- **startGame** – *Begins game loop, cycles through all questions.*
- **sendQuestionToAllPlayers** – *Sends current questions to every connected player.*
- **updateScore** – *Updates each player's score after answering.*
- **sortAndSendGameResult** – *Sorts scores and sends final leaderboard results.*

D) Request Class - Multithreading

The Request class is designed to handle client connections in a multithreaded manner by implementing the Runnable interface. Each Request object runs in its own thread, allowing the server to process multiple player interactions concurrently. This enables smooth handling of player requests, such as sending questions and receiving answers, without blocking other players. The run() method processes the player's requests in a separate thread, ensuring that each player's actions, like updating scores or sending game results, are managed independently. Method that initialized multithread:

```
public void run() {
        try { processRequest(); // Process client request
        } catch (Exception e) {
                System.out.println(e); // Print any exceptions that occur }
 }
```

E) Host Model - Special Client that Setting Up The Game

The Host class connects to the server and facilitates game setup by interacting with the server through a socket. It receives and displays messages from the server, allowing the host to select a topic, specify the number of questions, and set the duration for answering each question. After gathering all the inputs, the host sends the choices back to the server and completes the game setup.

IV) Conclusion / Summary

Ourhood successfully demonstrates a real-time, TCP-based trivia game using Java's networking and concurrency features. It offers a compelling mix of customizability, responsiveness, and scalability. The server can handle multiple clients concurrently, maintaining a consistent game state through synchronized logic and message broadcasting.

V) References

1. Comer, D. E. (2018). *Computer networks and internets* (6th ed.). Pearson
2. Tanenbaum, A. S., & Wetherall, D. J. (2013). *Computer networks* (5th ed.). Pearson.
3. Kurose, J. F., & Ross, K. W. (2017). *Computer networking: A top-down approach* Pearson.
4. Stallings, W. (2013). *Data and computer communications* (10th ed.). Pearson.
5. Peterson, L. L., & Davie, B. S. (2011). *Computer networks: A systems approach* (5th ed.). Elsevier.