Design and Implementation of Codenames

Darcely Pena, Michael Freda

Dickinson College

penad@dickinson.edu

fredam@dickinson.edu

Abstract— With the rise of the internet throughout the past couple decades, people are becoming progressively fonder of engaging in games online with their friends rather than in person. This has resulted in party games slowly transitioning into online play. In this paper, we will address how this conversion is being achieved through the efforts of Java libraries such as java.io, java.net, and java.utils, along with the protocols that allow this to be possible.

I. Introduction

Codenames is a multiplayer game where two teams compete against each other to identify their assigned words based on clues given by their team's respective spymaster. The game is played with a list of twenty-five randomly selected words, of which the spymaster is responsible for providing one-word clues and a number indicating how many word cards relate to the hint to their teammates, the agents. The agents proceed to guess which words belong to their team from the bank of words while avoiding the other team's cards and the black card, which serves as an assassin by making the team who guesses it automatically lose. Whichever team is first to identify all their words will win the game, assuming they do not pick the black card in any of their turns.

II. System Architecture

The architecture uses a client-server paradigm with a dedicated server that provides communication between four clients during the game. The application sends and receives messages through the server socket designated for the server, of which it uses the connection socket and player request object to process player request messages sent from the clients throughout the game. The server sends numerous messages to each of the clients throughout the game such as status updates in terms of when the game will start, along with team and role assignments, card specifics for each role, turn notifications, and countless other functions. The server exchanges messages between the clients in terms of hints, number of cards related to the hints, and attempted guesses from the respective client at turn. In

terms of the protocols used by the application system, we can use the protocol stack diagram below in Fig 1 to illustrate the layer in which our application resides and the roles of each layer in enabling communication between the various components of the system.

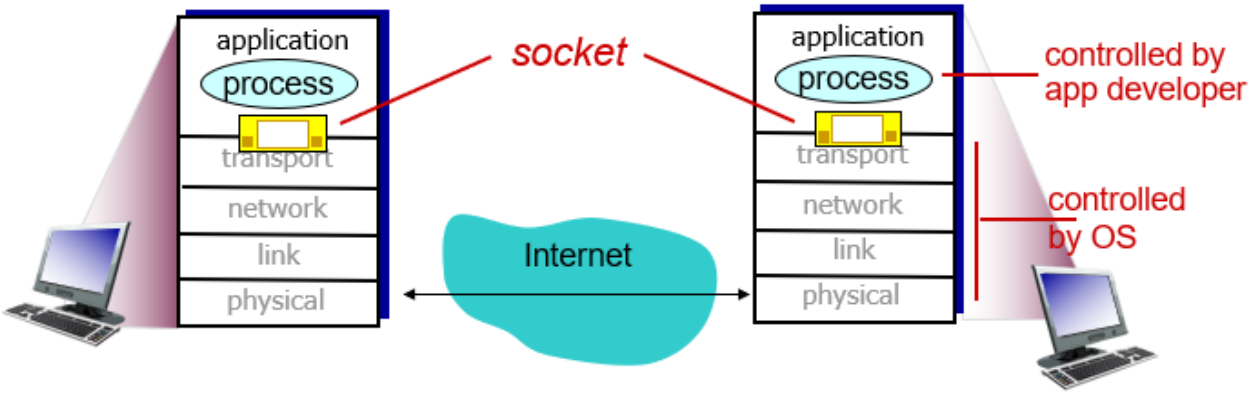

Fig. 1: Protocol Stack Diagram

The system application resides within the Application layer, rather than in between the Application layer and the Transport layer as it is just using the socket, while not operating outside the user space within the Application layer. In terms of each layer's roles, the Application, Transport, and Network layers are being highlighted within our application. The Application layer uses http as its protocol to ensure the datagrams are converted into packets for our application to function properly. The Transport layer would use TCP as its protocol to allow for reliable transport of segments to be converted into datagrams. Finally, the Network layer would use the Internet protocol, IPv4, to convert frames into segments. The messages between entities are exchanged by sending the messages from the server socket and the client sockets receiving these messages. In terms of timing diagrams, the server socket serves as the sender and the client sockets serve as the receiver below. This represents the relationship between the sender socket and the client socket in practice. We have multiple of these timing diagrams occurring as we have four client sockets receiving messages from the server socket in the application.

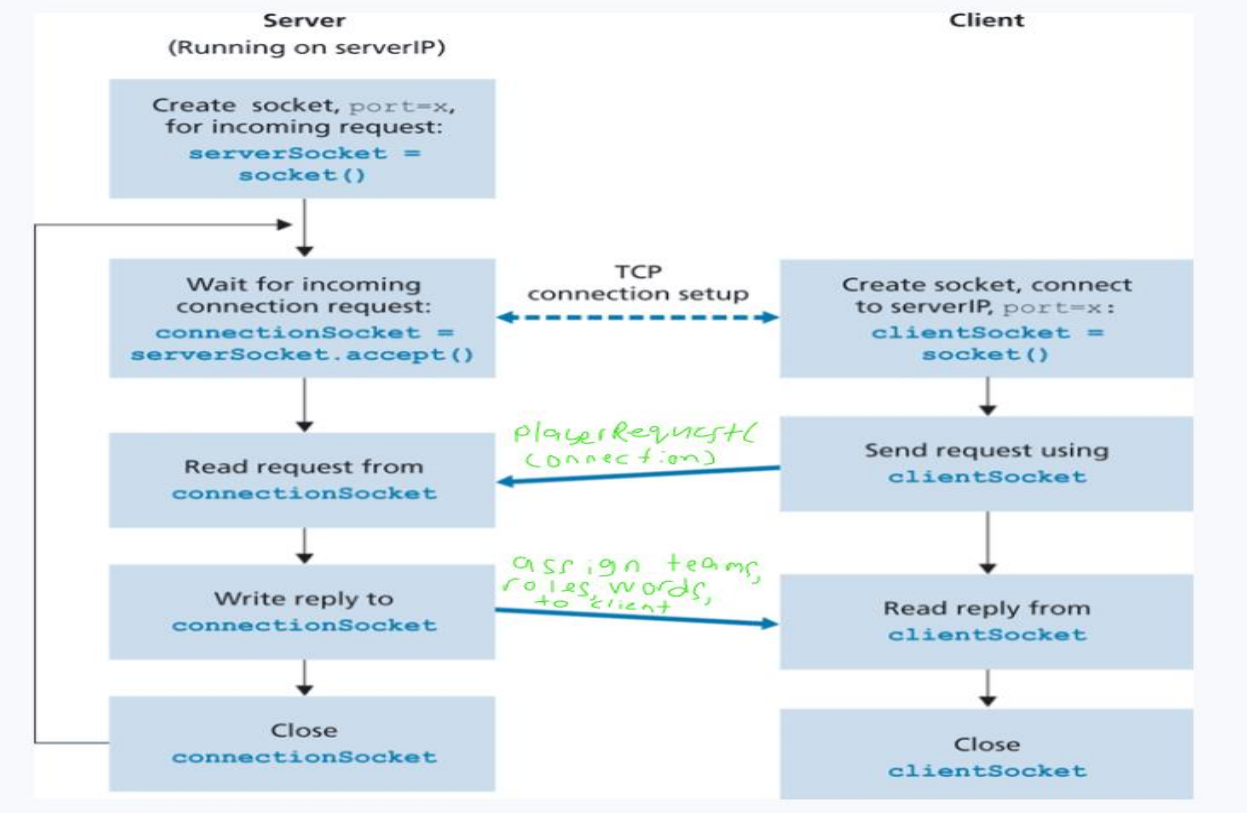

Fig. 2: Timing Diagram

The above diagram highlights the initial handshake between the server and client, then the exchange of messages between the two sockets. It demonstrates one connection between the server and a client, but for our application this would be expanded into four connections between one server socket and four client sockets.

III. System Implementation

The programming language selected to write the code was Java as it contained multiple libraries that allowed for an efficient and effective establishment of our server-client paradigm required for the application to work successfully. The project specifically utilized the java.io, java.net and java.util libraries.

A. java.io Library

The java.io library provides classes for performing input and output operations, essentially allowing the application to read and write data between the server and client. This is important as the application uses the server to communicate with the clients and to deliver data, messages in our case of Codenames, between the clients during the game (hints, guesses, etc.). Multiple classes from the library

are used in the code, such as BufferedReader, DataOutputStream, and InputStream. The BufferedReader class is used to read text from a character-input stream, speeding up the IO process as we are reading a larger block of characters at a time, rather than one. The DataOutputStream allows our application to write primitive Java data types, of which are output to the clients in the game using the writeBytes method. Finally, the InputStream class is simply used for reading input streams of bytes.

B. java.net Library

The java.net library provides classes for implementing network applications such as the Codename application described in this paper. This library handles the sockets that are pivotal to handling messages between the server and the clients present in the game. The application utilizes the ServerSocket and Socket classes, which serve as the server socket and client sockets respectively throughout the code.

C. java.util Library

The java.util library handles the miscellaneous utility classes, which in the case of the Codenames application, includes the Random class. The Random class is used in the code to randomly select the list of twenty-five words from the provided word bank, and to also designate a certain number of these words to separate categories such as the red cards, blue cards, and the black card.

D. Application Execution

The application can be executed installing Java on your machine, then compiling the CodenamesServer.java and CodenamesClient.java files in a terminal. After the files are compiled, you can proceed to starting the server class typing java CodenamesServer in terminal, after which you will type java CodenamesClient in four separate terminal windows. The server and clients all must use the same port number for the application to work. After four clients connect the game will start with teams, roles, and words being assigned, after which the hints and guessing will begin.

```
C:\Users\micha\eclipse-workspace\comp352\src>java comp352.CodenamesServer
Waiting for four players to join...
Player 1 has joined the room.
Player 2 has joined the room.
Player 3 has joined the room.
Player 4 has joined the room.
All four players have joined the room. The game will start now!
```

```
C:\Users\micha\eclipse-workspace\comp352\src>java comp352.CodenamesClient
Waiting for other players to join...
Your team is:
Red
Your role is:
Spymaster
Game Cards:Comic
```

IV. Conclusion

Technology is rapidly engulfing countless aspects of our lives, including social engagement that was previously restricted to an in-person experience. The Codenames game is brought to life digitally by using a TCP connection for a server-client paradigm to bring together people without the limitations imposed by physical location. The Java coding language and its numerous libraries open this realm of endless possibilities for development. Some potential future enhancements consist of player scalability to allow for more than 4 players, so one spymaster and 2 or more agents on each team, requiring an even overall number of players in the game. A clue validator that works to prevent cheating by blocking hints that are one of the words provided, resulting in a punishment of forfeiting their turn to the other team. A sound implementation for time-left countdowns, along with successful and failed guesses. The room for innovation is limitless, showing the beauty of coding and the ability to connect over a network online.

References

Kurose, J., & Ross, K. (2020). *Computer Networking, 8th edition*. Pearson.

Package java.io. (n.d.). Oracle Docs. https://docs.oracle.com/javase/8/docs/api/java/io/package-summary.html

Package java.net. (n.d.). Oracle Docs. https://docs.oracle.com/javase/8/docs/api/java/net/package-summary.html

Package java.utils. (n.d.). Oracle Docs. https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html