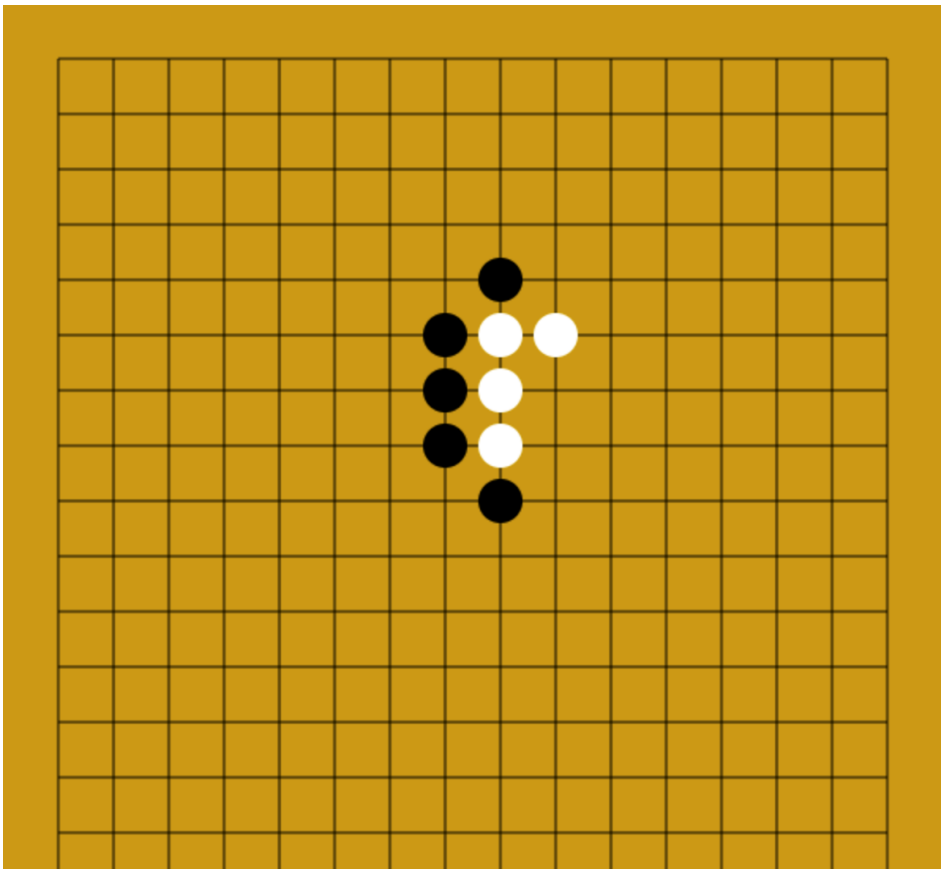
WebSocket Based Gomoku

Course: Data Communication Computer Networks (CSCI 53600)

Xiaoyuan Yu, Kaiming Cui
xyu1@iu.edu, cuik@iu.edu
2000458297, 2000407298

December 9, 2019



Abstract

This project focus on data communication through network by using WebSocket protocol. The project is based on server-client architecture. A server deployed on Node.js always wait for connections from clients. Once connected, server and client send messages to each other through socket.io frame. The multiplayer Five in a Row game includes a game lobby and multiple battle rooms. Both the lobby and the rooms have a simple chat system which allow players chat with each other. We captured all packets sent between server and clients by using Wireshark and will show how the server and clients work together to implement a this multiplayer Five in a Row game.

Keywords: WebSocket, JavaScript, Node.JS, Socket.io.

1 Introduction

Gomoku, also called Five in a Row, is an abstract strategy board game. It is traditionally played with Go pieces (black and white stones) on a Go board. It can be played using the 1515 board or the 1919 board[1]. The game is known in several countries under different names.

The game follows the typical pattern that players take turns to place a stone of their color on an empty intersection. The winner is the first player to form an unbroken chain of five stones horizontally, vertically, or diagonally. This pattern suits the idea of WebSocket perfectly.

WebSocket is a computer communications protocol, providing full-duplex communication channels over a single TCP connection[2]. WebSocket not only enables communications between a web browser (or other client application) and a web server but also helps with lower down the overhead compared to half-duplex protocols, thus facilitating real-time data transfer from and to the server.

The operations of WebSockets are event-based, server and clients listen on the reserved ports and receive messages from each other. The server is responsible for data synchronization and updating for all connected clients. All messages send between serve and clients will contain two parts. The first part is event name, the second part contains parameters that is optional. According to what the received event is, receiver enters different functions to handle the event. In our case, each player will trigger the socket when it's his turn and he has made a move. Thus the server can receive the move and if it's legit it can send out the message to the opponent through another Websocket. In the following sections we will discuss the design of our system and analyze it.

2 Structure

2.1 system structure

The whole system is based on server-client architecture. On the server side, a game lobby server keeps listening on the port 3000, waiting for any connections. Once it receive a connection request, it will create a socket object for that connection that means each client has a socket object associated with it on the server. Once a connection is established, server and client could use socket.emit

to emit package to each other. Besides, server could broadcast the package to all connected clients by using socket.broadcast or io.emit.

On the client side, each client actually is a user's browser that opens index.html page. Once the browser get the index.html page from server and open it, the index.html page will create a connection to the game Lobby server by calling socket.io.connect function. The index.html page shows the game lobby to player which include all the rooms and a lobby chat system. All data references game lobby is synchronized by server for all clients that means the lobby is the same for every player, and the updates are synchronized. The chat system shows system message and client's message to all clients in the lobby. System message include a new player comes in the lobby or a player left the lobby (close the browser). The room page is presented by fight.html which is hidden if you are in lobby. When player click on an available seat, the fight.html will be initialized and shown to player. The fight.html (battle room) includes a chess board, a exclusive chat system and function buttons that provide regret function, new game function and go back to lobby function. Notice, fight.html is embedded in the index.html by using iframe. It is shown and hidden through CSS style. When player click on the goLobby button, fight.html will be cleared and then hidden from player.

2.2 data structure

2.2.1 server side data structure

The server is mainly responsible for maintaining all the online users and rooms available for the incoming players.

All of the online players are stored in a list of tuples which is formatted as $[name, room_id, current_status]$. The 'name' field is unique for each player online and acts also as the identification of the player. The $'room_id'$ and $'current_status'$ identifies whether the player is playing.

The rooms available on the server is also maintained via a list of tuples: $[room_id, player1, player2]$. The $'room_id'$ identify the room and the other fields maintains the name of players in this room.

2.2.2 client side data structure

On the client side, the most important data structure is the current state of the board. The board is represented by a 2D matrix of integers where 0 means empty, 1 means it's occupied by a black piece and 2 means there is a white one. This matrix is used for canvas to draw the board in the browser, it's synchronized between two players by the server.

In addition, the client also records the previous moves done by both players which facilities the operation to reset the board if someone wants to regret his last step. The client also stores some indicators like whose turns it is and the ID of the game.

3 Package Analysis

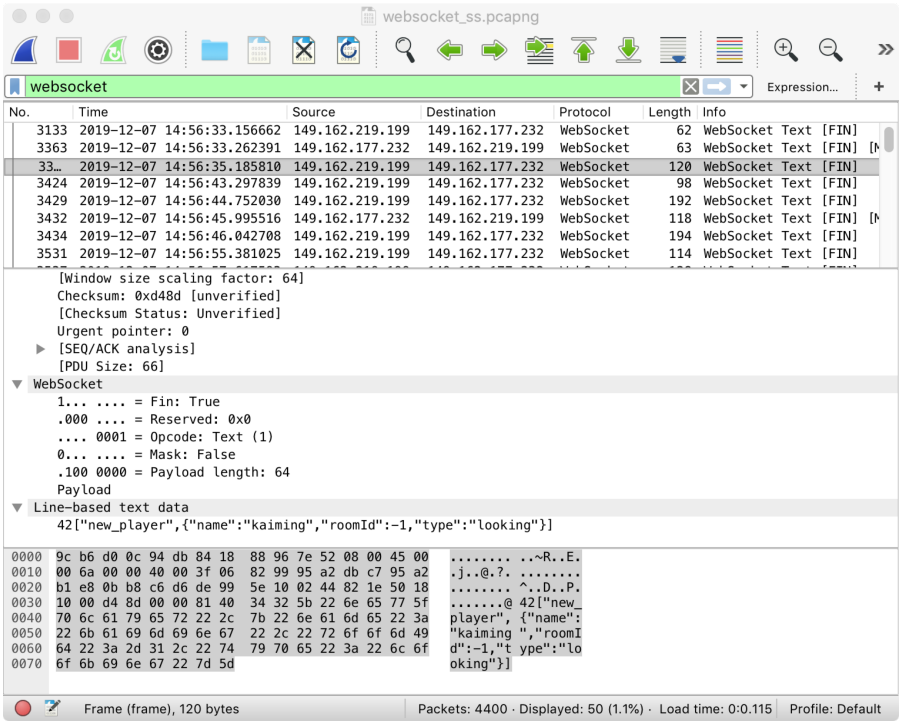


Figure 1: Message send to clients when a new player comes in the lobby. The chat system in the lobby will show XXX comes in.

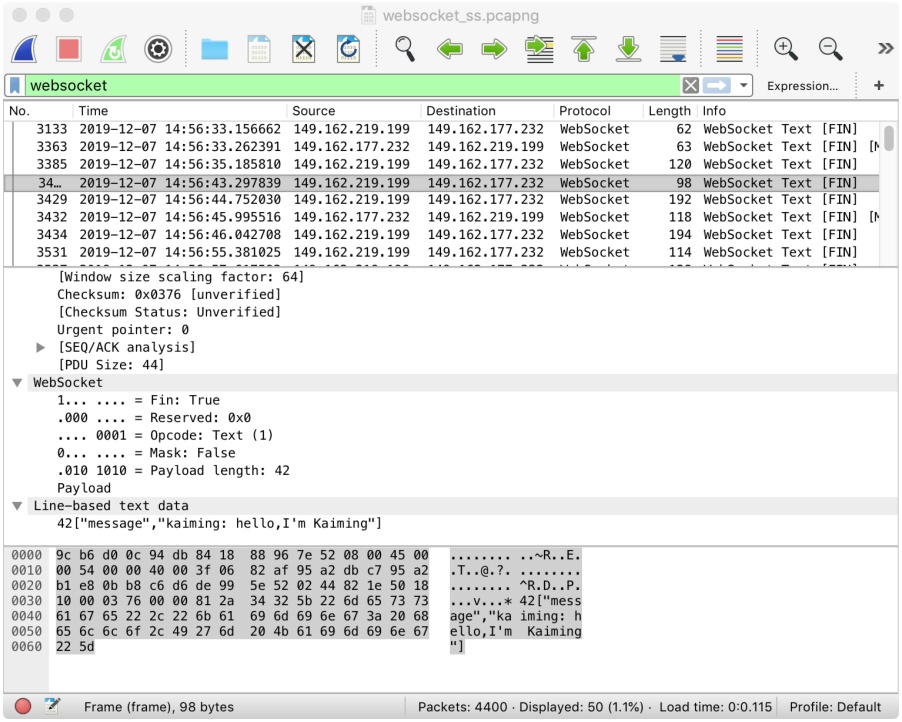


Figure 2: package send to clients when a player send a message. The chat system in the lobby will show that message and any clients will see it.

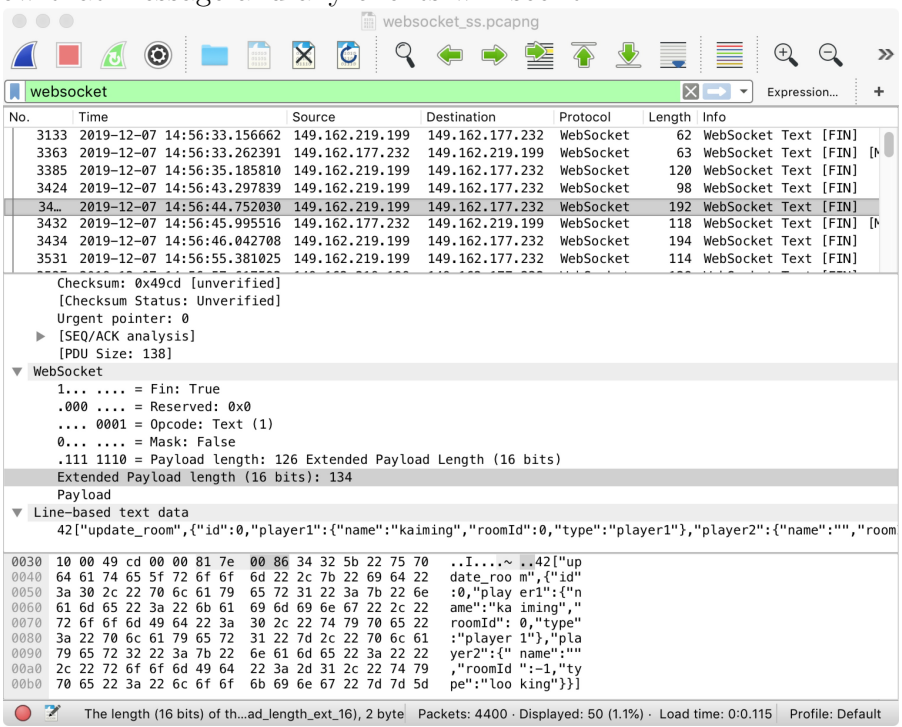


Figure 3: package send to clients when a player site down a seat that means enters a room. All clients will receive this message and update its own room list. Therefore all clients' room list is same. Everyone can see who is in which room and if the client is already in a room, it will see its opponent.

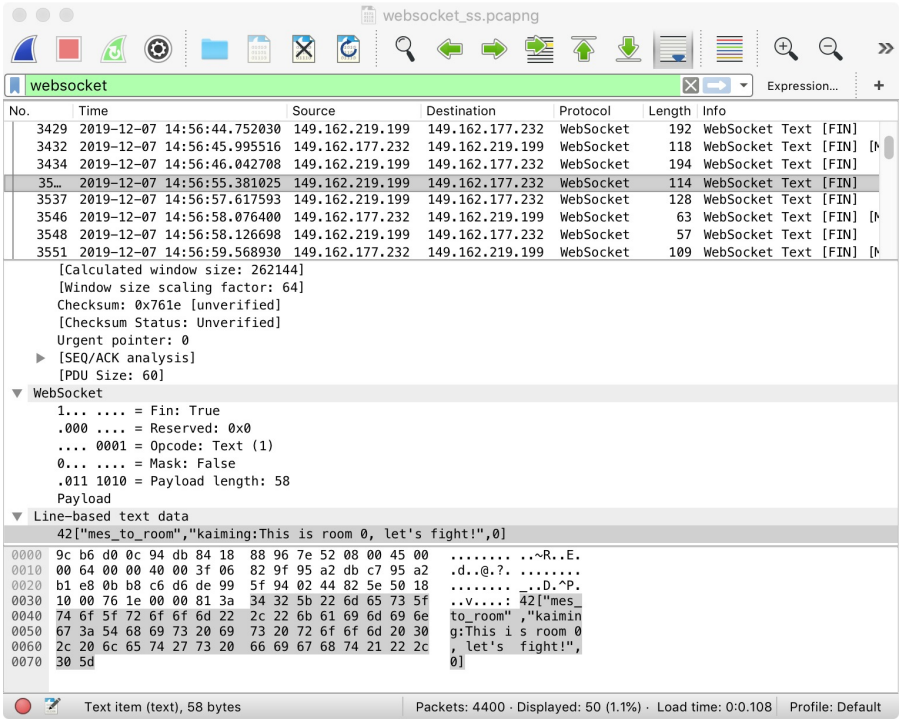


Figure 4: package send to clients that in a specific room, only the clients in that room will see the message on their room's chat board.

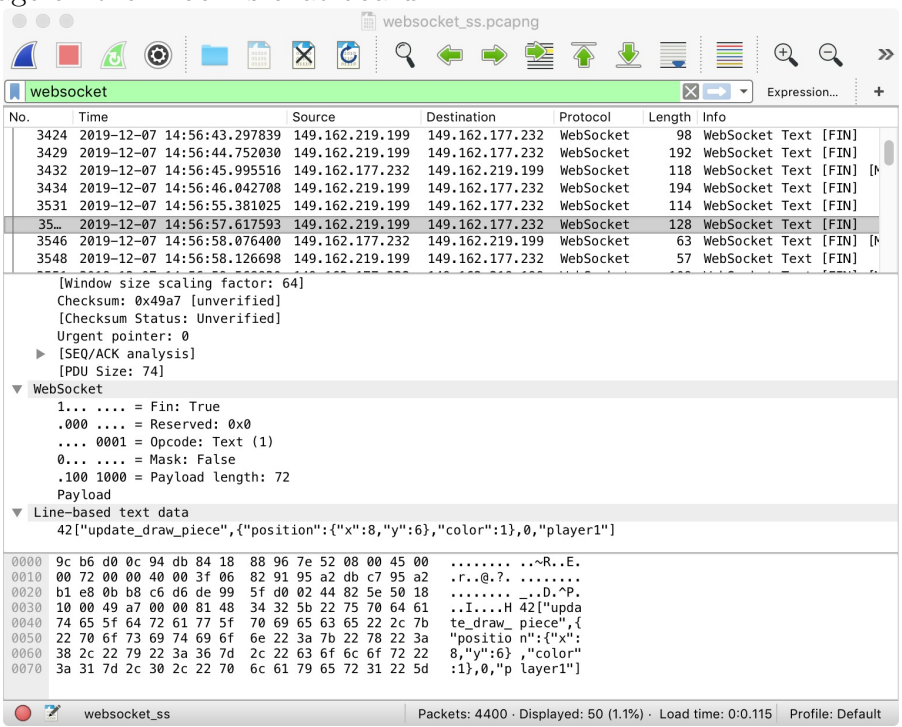


Figure 5: package send to clients that in a specific room, only the clients in that room will update their pieces list. They will see the new piece in their chess board. In addition, each client will check if there is a victor and decide if the game is done.

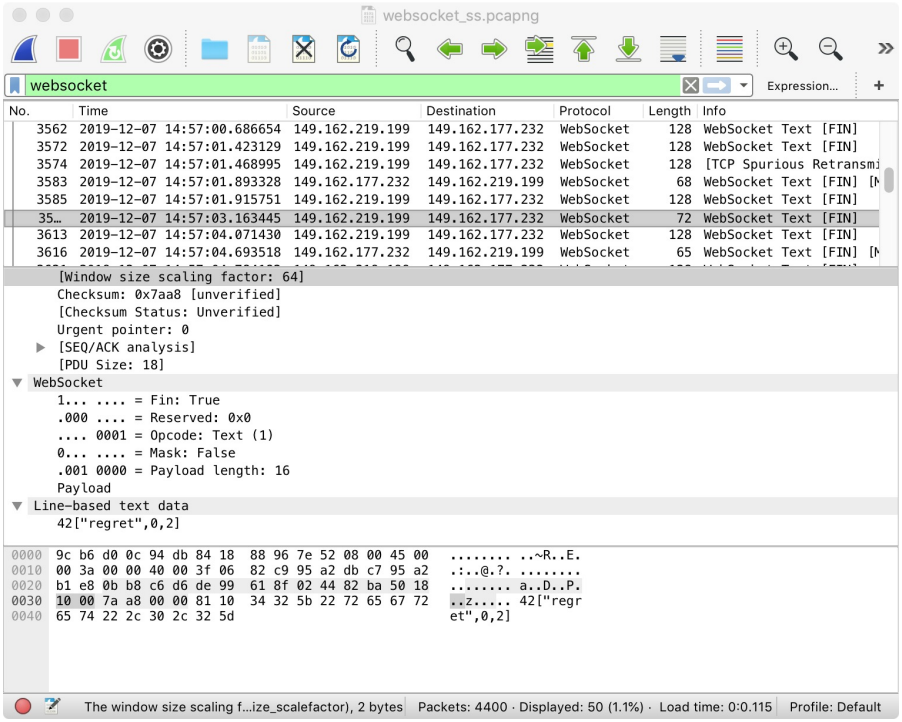


Figure 6: package send to clients that in a specific room, only the clients in that room will update their pieces list. The two or one pieces that are recently added will be removed from the chess board.

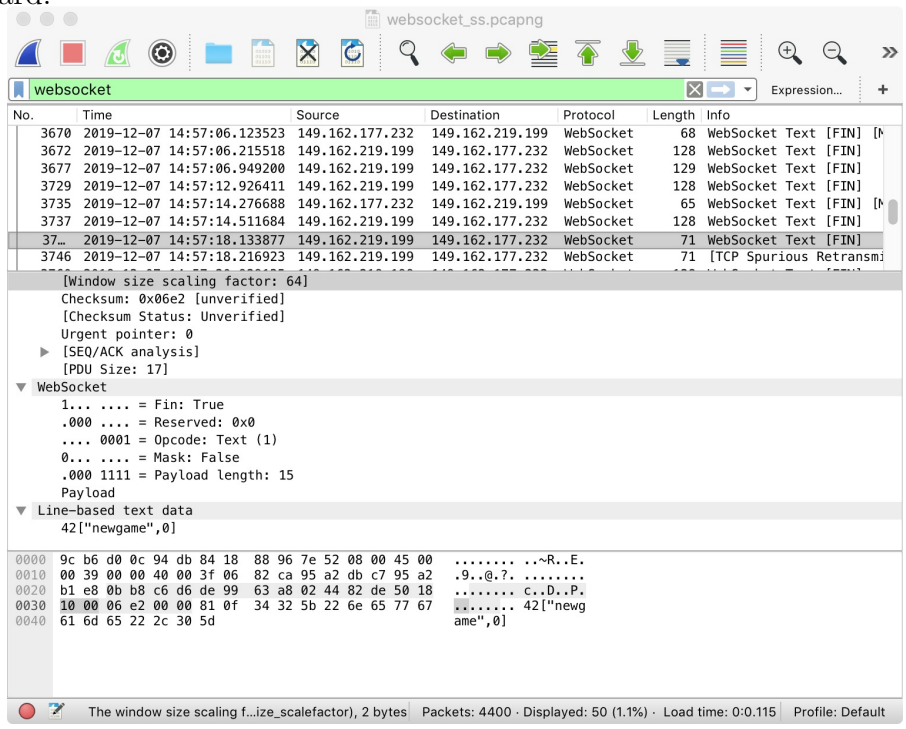


Figure 7: package send to clients that in a specific room, only the clients in that room will clear their chess board and initial their pieces list to start a new game.

References

[1] *https://en.wikipedia.org/wiki/Gomoku.*

[2] *https://en.wikipedia.org/wiki/WebSocket.*