Shreyansh Mohnot

December 4, 2017

CSCI 53600: Data Communication and Computer Networks, Fall 2017

Project - Five in a Row Game

1. Problem:

The problem for the project is to develop and implement an online "Five in a Row" game. It is an abstract strategy board game and sometimes also known as Gomoku, Gobang. It is traditionally played with go pieces (black and white stones) on a go board (19x19 intersections).

Black plays the first turn, and then white. Both the players alternate in placing a stone of their color on an empty intersection. The winner is the first player to get an unbroken row of five stones horizontally, vertically, or diagonally.

2. General Idea:

For a multiplayer online game, first of two things are necessary. A server and A client. On researching about WebSockets and their implementation, I could find that WebSockets work on HTTP protocol. Its design allows for faster game experience. WebSockets upgrade the HTTP protocol connection and then communicate.

I made a rough sketch on how a game would work in practical client and server scenario. I would need client program and server program. Basically, Gomoku is a simple strategy game unless and until you are playing with a human. So, for the server, I implement creation of WebSocket inside it and checking the wining conditions for each move by the client. This makes handling simpler as incase if a client wins then we can send message to all the connected clients that this player wins. If no wins then just draw the game.

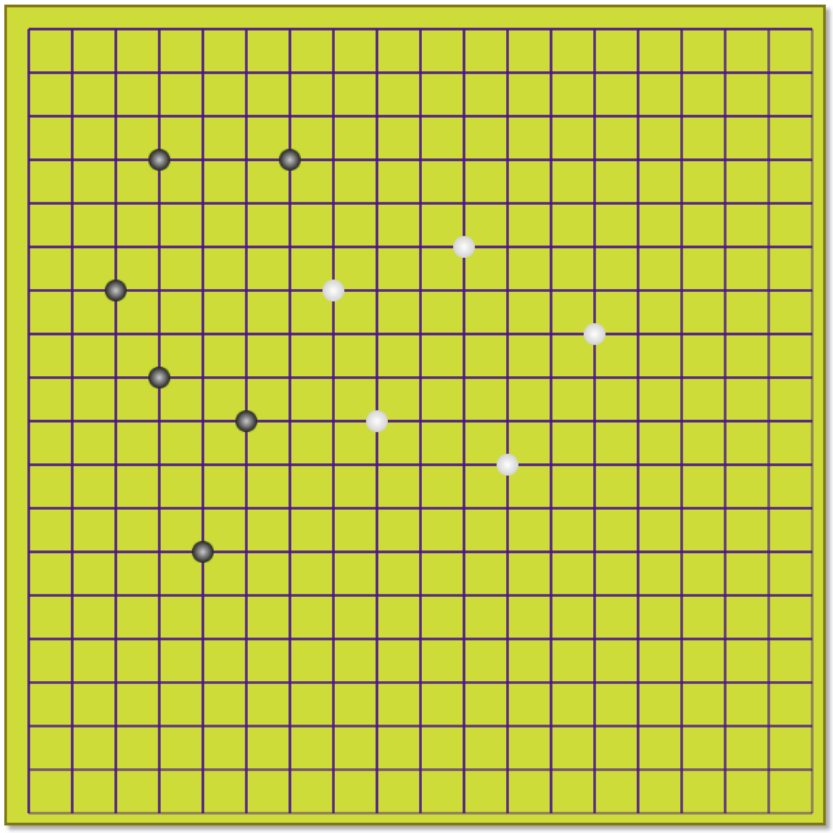
3. Implementation:

This is an online game developed in HTML, JavaScript on Client side while server is in NodeJS. Client use WebSocket (RFC6455) protocol to connect to the server and, request and return game values to it. For value passing JSON objects are used. Server is created in JavaScript's server implementation known as NodeJS. However, the WebSocket server first uses and HTTP request and then upgrades that request to WebSocket for the support in the game.

I preferred JavaScript and NodeJS for deployment of application as WebSocket are browser compatible and every client has access to one or the other web browsers. So, it would become easier to connect to WebSockets from the browser and play online. Also, JSON objects are lightweight and can handle high amount of data in one object so, parsing is done through it.

4. Client Program:
    - Clients just run the game file.
    - For each file run, users are assigned black and white stone in a serial order.
    - First client to connect always will get black stone.
    - Then the second client gets white stone.
    - Clicks on gameboard generates index values which are sent to server.
    - Send them to server as JSON objects.
    - Waiting for values to response from server.
    - Analyze those values from type and decapsulate them from JSON format.
    - Clients can differentiate between the update values for gameboard and winning situations.

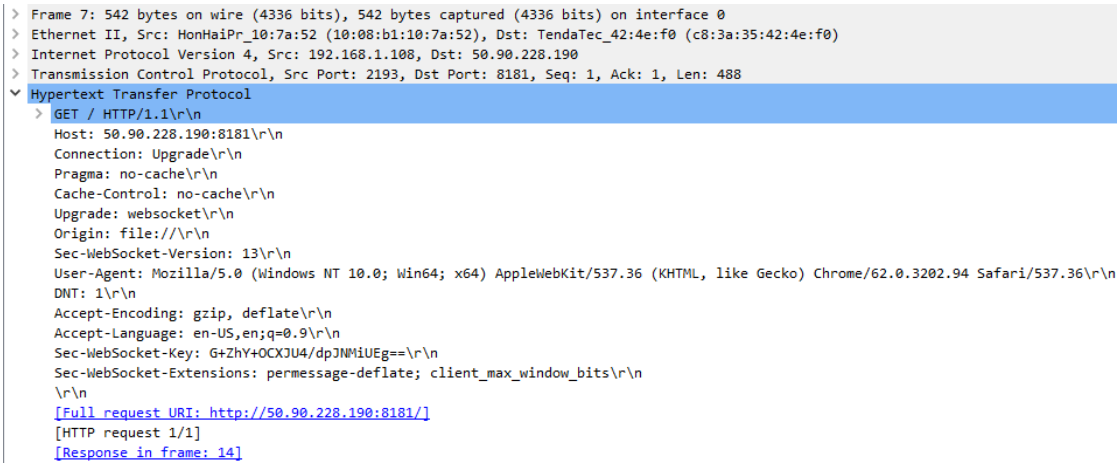
- If a game is win then it disables screen click and shows which player won.
- Can have clients watch game if one cannot join and wait for them to finish the game.
- The click is disabled on the screen incase if there is over watch.
- Snapshot of Client Side:



5. Server Program:
   - Connects to the all the clients requesting access to WebSockets.
   - Make sure when the first two clients and each one is assigned a black and a white stone respectively.
   - If more then two clients, then just freeze the display and allow them to over watch the game.
   - History stack to get the game instance at the connection time from new clients and update their gameboards.
   - Receive update requests from the clients playing.
   - Update gameboard with user values and push them into history stack.
   - Checks wining criteria whenever a click is done by clients on the gameboard.
   - If we have a winner, then declare that winner all the clients connected and freeze the screens until they turn it off.
   - When a client leaves just remove him from the history and close that connection.
   - Support over watching by clients who join after two players have joined. They can watch the game but cannot contribute in the game.

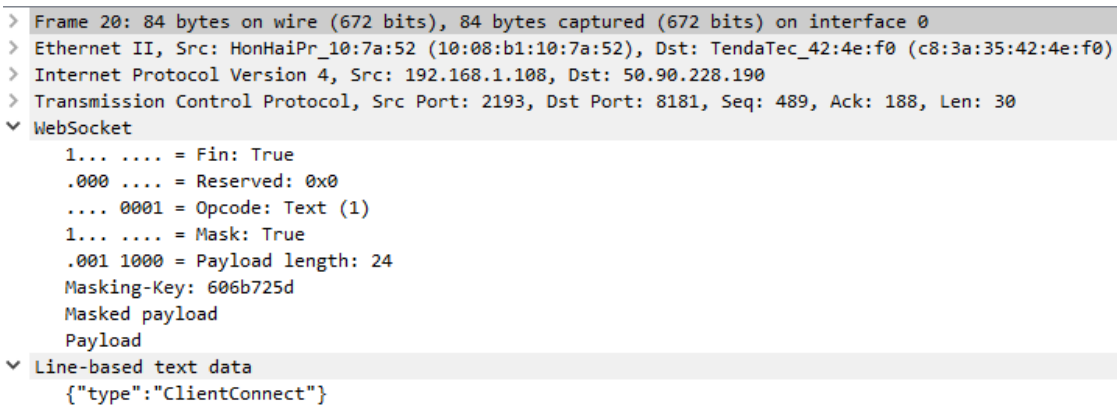
6. Wireshark Analysis:
   I used wireshark to analyze how the protocol works on the OSI Layer. I have included some frames from analysis.

   I. Initial Client Connect Handshake Packet:



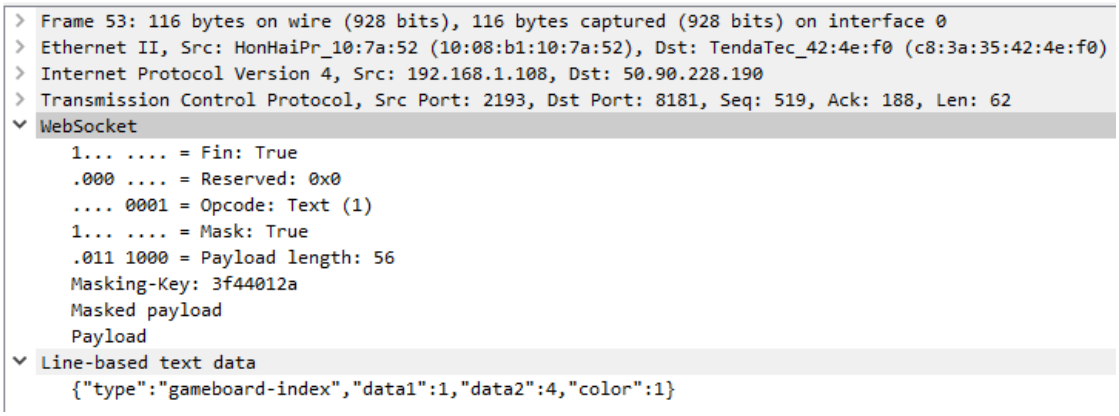
Whenever client load the window, the http upgrade request is send from client to server. Handshaking is done as it happens on transport layer. Response from server is to switch protocol from HTTP to WebSocket.

   II. Receive packet from sever:



This packet contains the information that client is now connected. It is followed by history of the clients connected. Server passes it to every new client which gets connected. History packets get deleted when client disconnects.

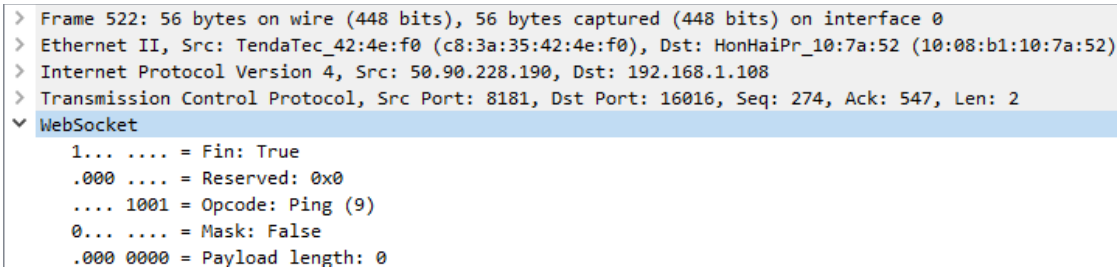
   III. Gameboard update packets:



This packet is send from the client to server requesting gameboard update. Whenever client send a new packet to server, it is always masked with a new key. WebSocket frames just add a line based text as the value send. Frame has opcode: Text. This is how a frame looks like in my implementation of the game.

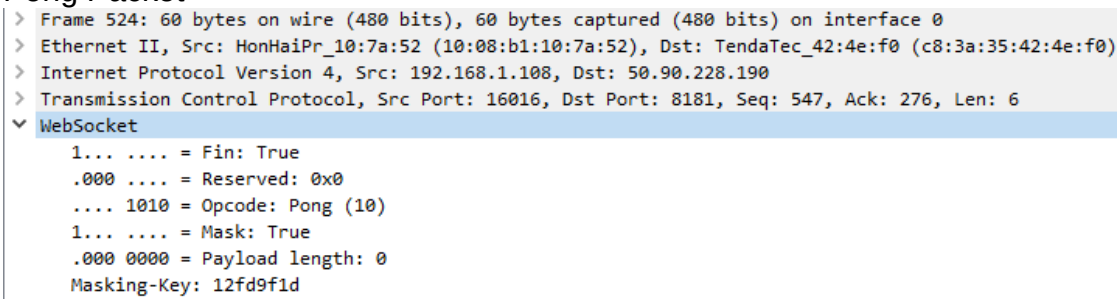
IV. Ping/Pong packets:

Ping Packet –



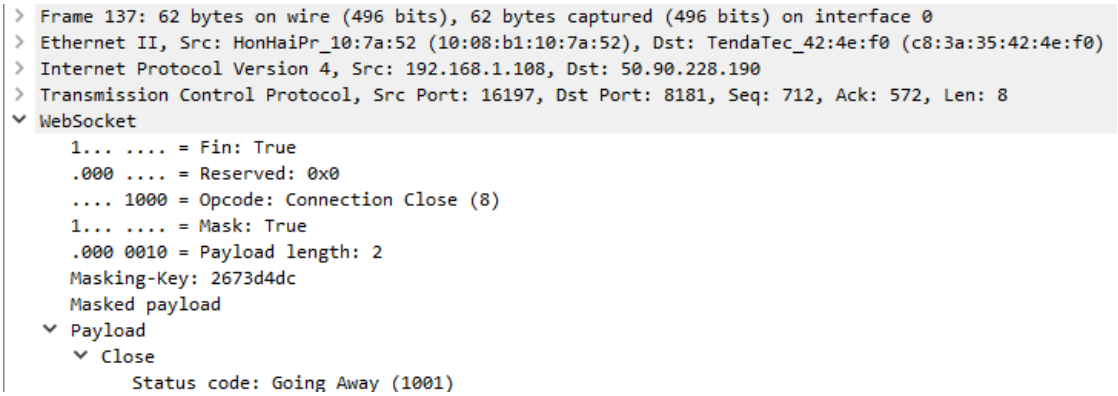
This packet is used by the server to ping client to maintain an active state. This helps in keeping the connection alive all the time until client explicitly closes the connection. It contains opcode: Ping. This is same as doing a persistent connection and every time just check the client state.

Pong Packet –

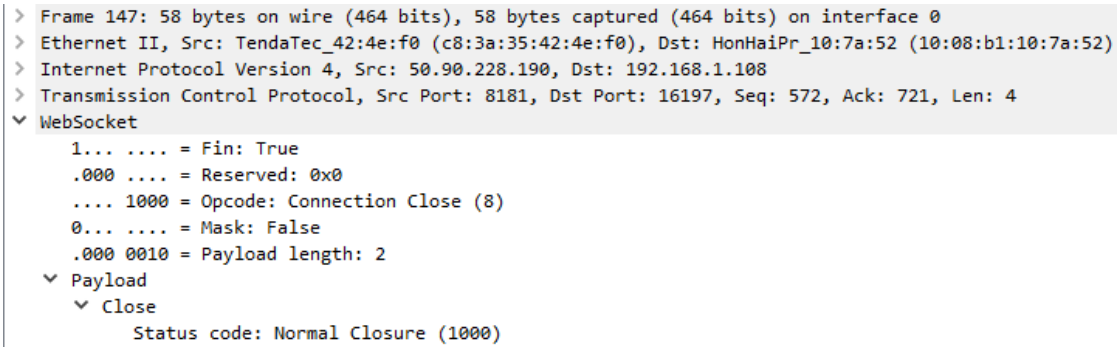


For each ping, client does a pong request. It contains opcode: Pong. This request is done to tell the server about the client status, its active or closed respectively. However, close has different packet status.

V. Connection Close packet:



This packet explains how WebSocket connection close request is send from client when the browser is closed. This WebSocket frame contains opcode: Connection Close and a status code: Going Away.



This packet tells to end the closing connection with normal closure of the protocol.

7. <u>Code Analysis:</u>

   This project is a multiplayer game. Game are much interesting to build as you need to think ahead of time and know what user wants after each step and how to deliver that into the game. Coming to the program which implemented in this game is light weighted as much of the processing is done on server side (because they are intended for that purpose).

   In my code, I tried to incorporate each possible failure which I thought of could be encountered in the process of playing the game. Some problems face and how I resolved them:

   - First client to connect gets black color. Its not a race condition. Hence, I just implemented using type checking each connection.
   - Point of failures were present when checking winning criteria's. Optimized my algorithm to check the conditions but still it's not perfect.
   - Gameboard update requests were not being reflected over to other clients. Server was unable to parse. I modified some variables and done.
   - Use of history to store each client move such that whenever a new client joins for over watch he could see the most recent buildup of the game.
   - If one client leaves in the middle of the game, there are error checks such that the prevailing client would be dropped from the server and his data is truncated.

   Algorithms for checking winning positions:

To win is much interesting in a way to implement the conditions for wining. There are four possible way any player could win the game.

- All five stones vertically of the same player.
- All five stone horizontally of the same player.
- All five stones diagonally – forward slope of the same player.
- All five stones diagonally – backwards slope of the same player.

8. Conclusions:

This project is an interesting to work on. It taught me how an unused and unheard-of feature of most widely used web protocol becomes so useful now. The WebSocket uses upgrade headers which were inbuilt by its creators but was not used. It was much surprise to know about hidden HTTP features which are used in WebSocket protocol. How the simple HTTP request was upgraded to a much simpler yet complex protocol and then it switches between HTTP and WebSocket finally. Few key points which I liked to share -

- The HTTP upgrades. How it just upgrades itself to more secure protocol by adding more functionality to it such as security key headers.
- Then HTTP knows about when to switch since WebSocket request came.
- WebSocket requests generated contain each a mask value which adds to security here in this protocol.
- Each message contains information encapsulated inside a frame.
- This protocol knows how to keep connection persistent which the clients. It uses ping/pong strategy which I liked the most in this protocol. For the connection to keep alive and can just assume that either the server or the browser will make sure that they exchange ping/pongs.
- A secure protocol. It supports HTTPS also.
- Connection closing feature when a WebSocket connection is closed, it reports how the connection terminated and its status value, which is extra from the HTTP where only an error code is sent.

9. References:

- https://www.html5rocks.com/en/tutorials/websockets/basics/
- https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications
- https://books.google.com/books?id=TU0nCgAAQBAJ&lpg=PA69&ots=3wfNDF0Uyf&dq=createRadialGradient%20javascript%20game&pg=PA69#v=onepage&q&f=false