Five in a Row

Project Report

Date: 11/30/2015

Author: Zhihao Cao

Email: caozh@umail.iu.edu

Student ID: 0002921224

Teammate: Rui Wang

Course: CSCI 53600

Instructor: Dr. Yao Liang

Index

Contribution

Design:

Socket understanding	Wang & Cao
Development platform	Wang & Cao
Communication structure	Wang & Cao
Game logic design	Wang & Cao
Application activity flows	Cao & Wang
Functions analysis	Cao & Wang

Implementation:

Functions-Server side	Wang
Functions-Client side	Cao
Testing and Debugging-Server side	Wang
Testing and Debugging-Client side	Cao

Analysis:

Packet activities analysis	Wang

Description

Five in a row is an abstract strategy game, also called Gomoku. It is traditionally played with black and white stones on a board with 15x15 or 19x19 intersections. The winner is the first player to get an unbroken row of five stones horizontally, vertically, or diagonally.

In this project, the “five in a row” game is developed as a client and server architecture web application (written in Javascript/HTML), so that one player can go to the game web-page and play against another player remotely.
The web application offers two positions for players. Game rules are designed to determine winner and loser. In addition, the game is able to determine if a player is disconnected and set another player in right activity if one of them suddenly leaves the game.

Using WebSocket (RFC 6455) protocol, we create the client-server-communication based on node.js environment. We use Socket.IO (a library offered by node.js) to create the sockets between server and client and let them have the real-time bi-directional communication.

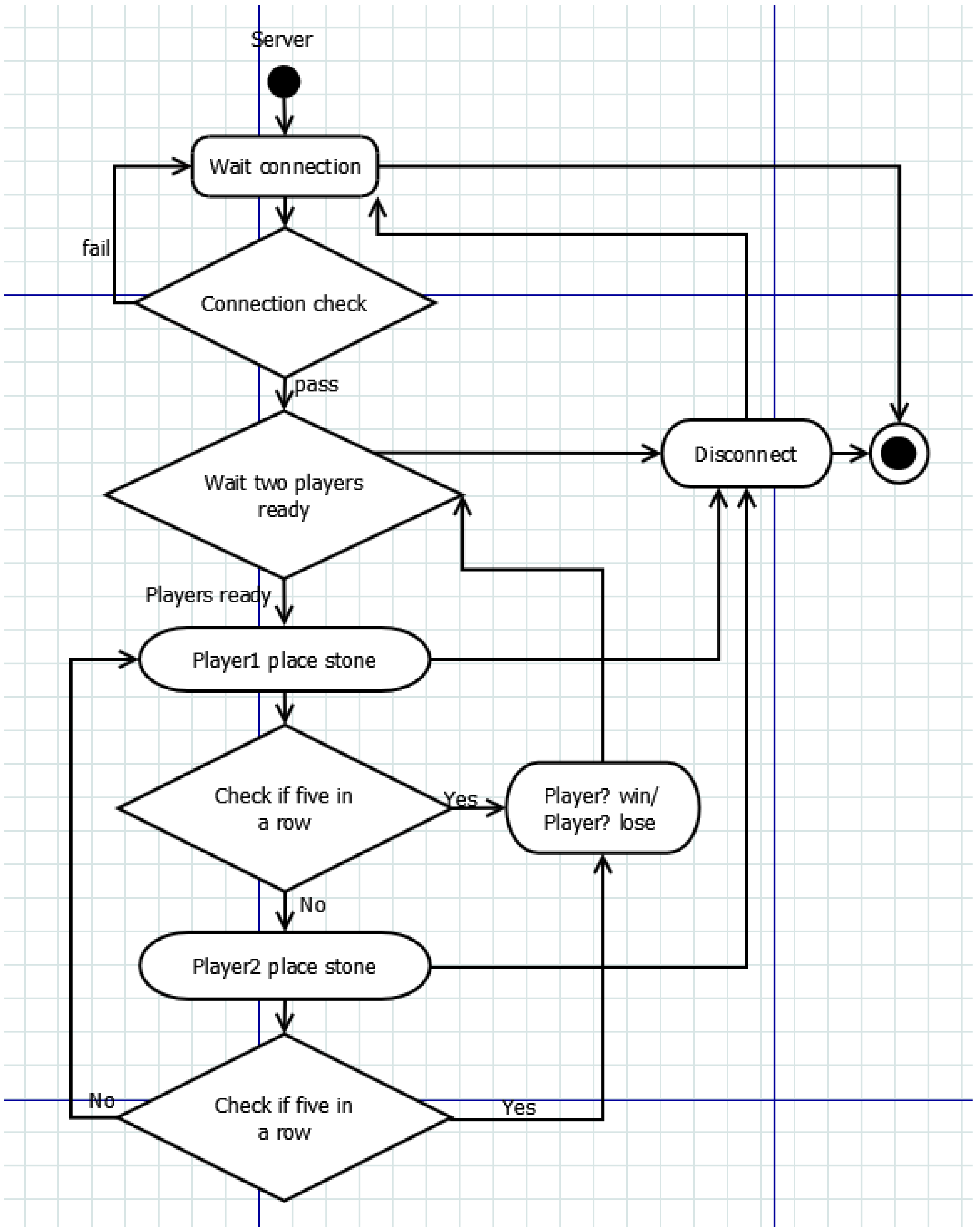
Node.js is an open-source, cross-platform runtime environment for developing server-side web applications. It provides an event-driven architecture and a non-blocking I/O API designed to optimize an application's throughput and scalability for real-time web applications.

Socket.IO is a JavaScript library for real-time web applications. It enables real-time, bi-directional communication between web clients and servers. It has two parts: a client-side library that runs in the browser, and a server-side library for node.js. Both components have a nearly identical API.

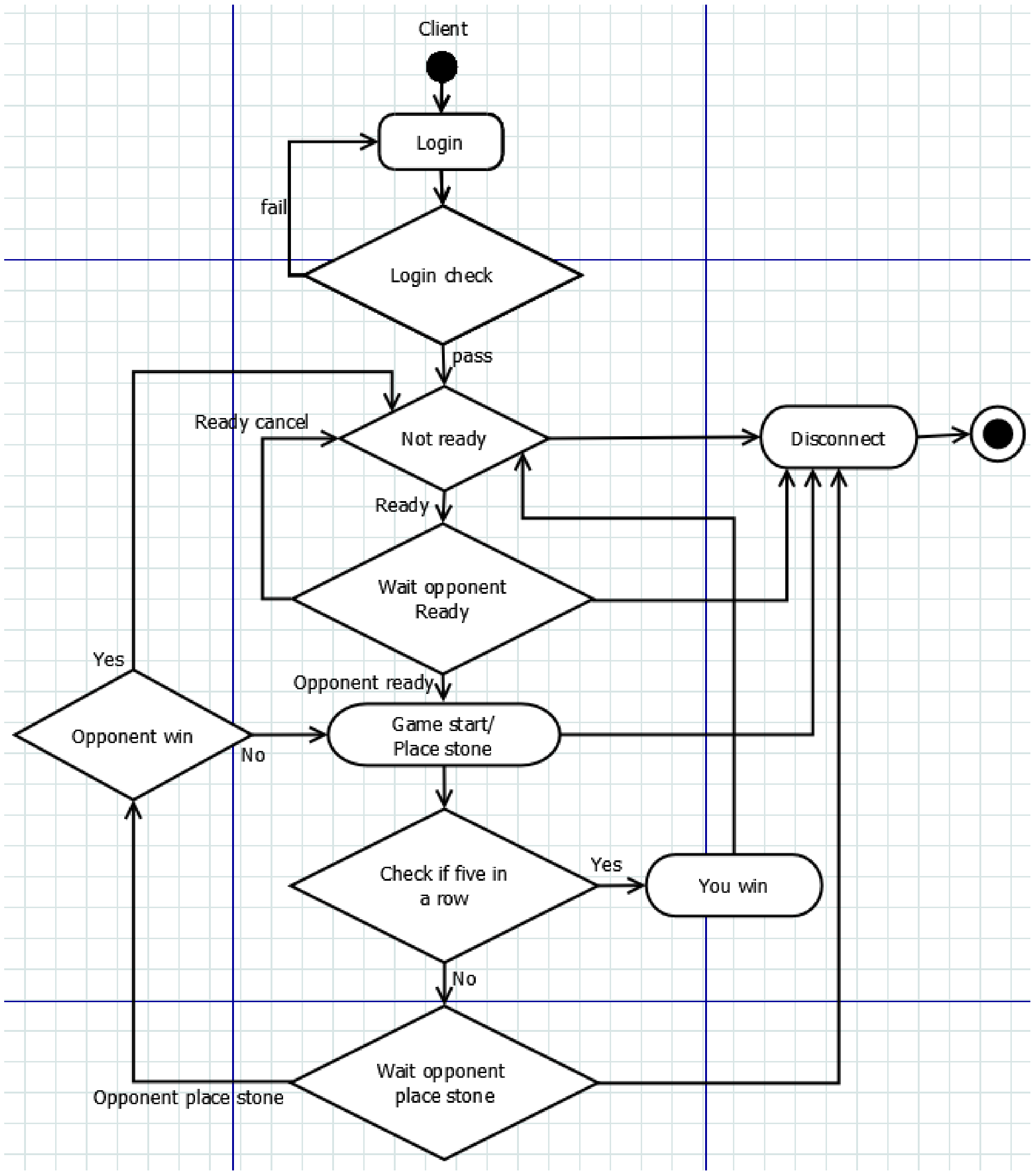
Socket.IO primarily uses the WebSocket (RFC 6455) protocol with polling as a fallback option, while providing the same interface. Although it can be used as simply a wrapper for WebSocket, it provides many more features, including broadcasting to multiple sockets, storing data associated with each client, and asynchronous I/O.

Design

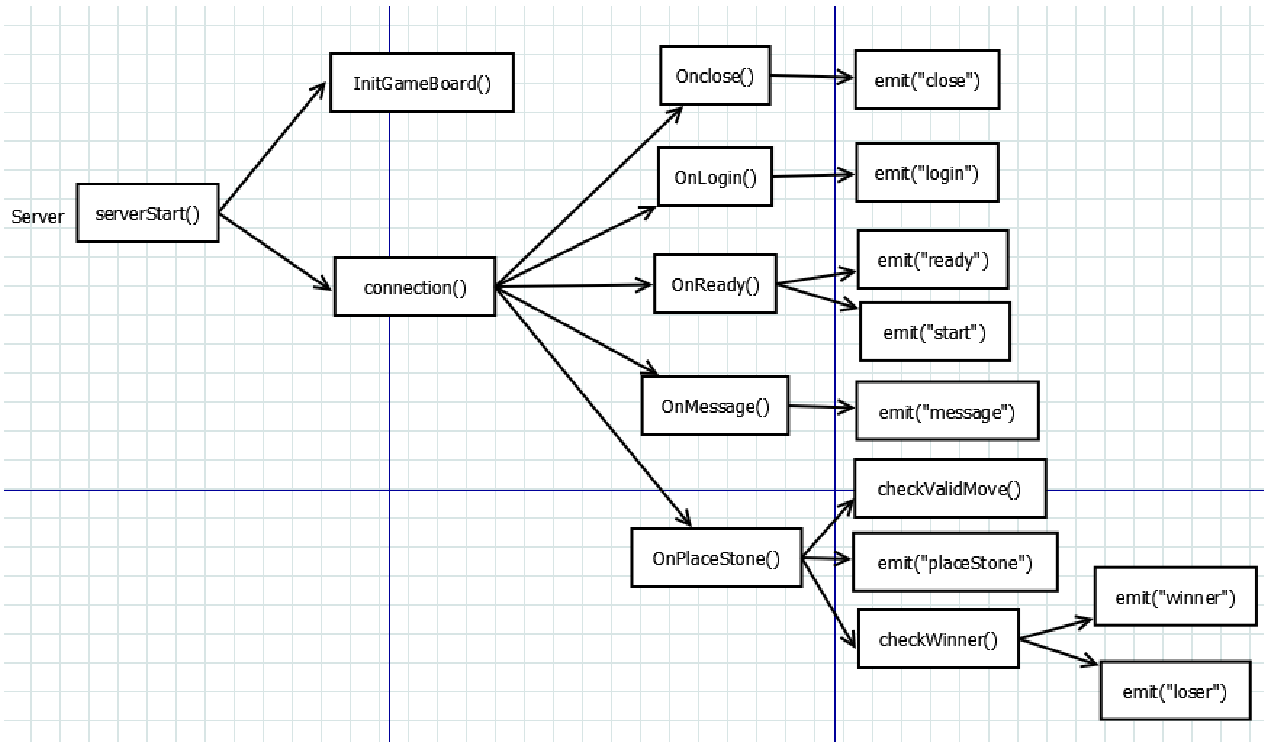
Activity diagram for server side

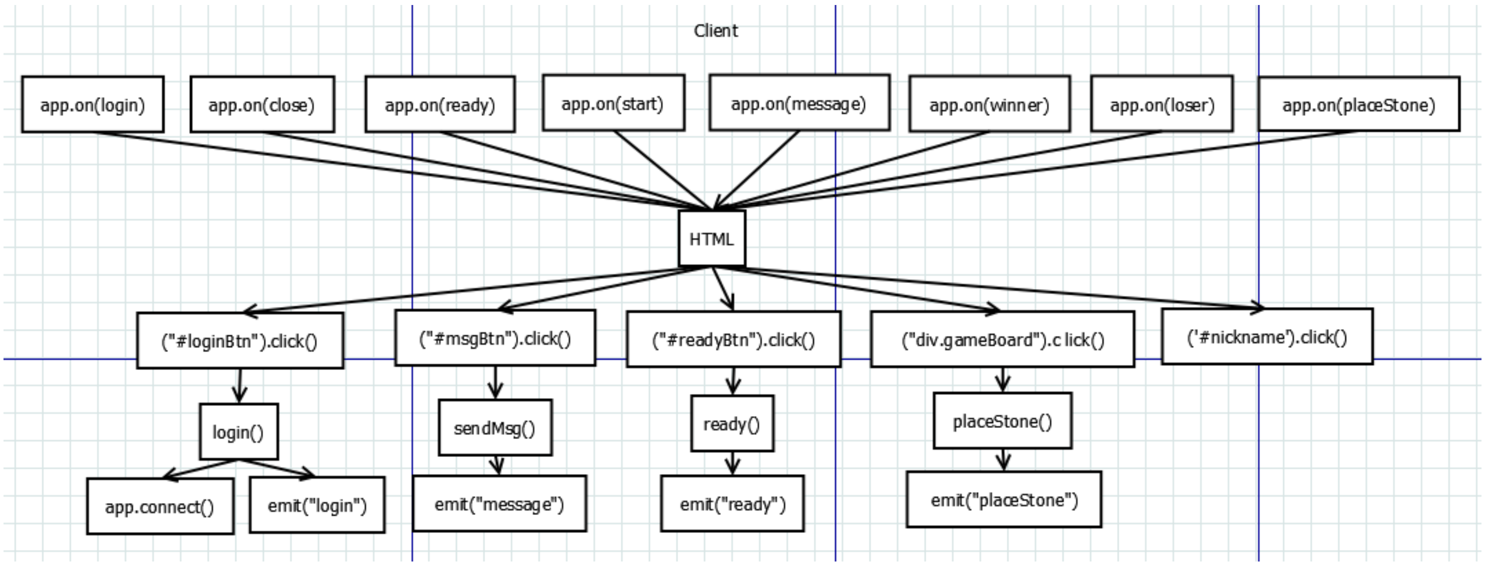


Activity diagram for client side



Dependency diagram





Difficulty

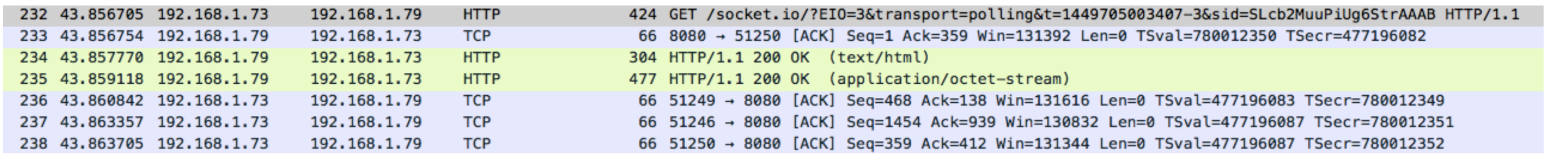
1. Setting up the client-server communication
   After we choose node.js to do this project, we wanted to let it run on the server as the project introduction states. The problem is that node.js cannot be install to our Pegasus environment because we have no the right to do that. Then, we tried to do it on local area network. We set up local server, and installed node.js but client could not get the html file from the server. The reason we could conclude so far is because of the firewall.

2. Synchronizing status/state between client and server
   In this application, many statuses should be kept both on client and server side. The client side initially only keeps socket id from the server. When the functions got more sophisticated, we realized only the socket id could not put the application in right status. Then, we used status key word to synchronize the status between client and server side, such as STAT_NOT_READY, STAT_READY, STAT_PLAYING, BLACK_STONE, and WHITE_STONE and so on.
   Besides, more situations the application has considered. Since players can come in and start playing and might leave in any moment, the application shall detect the client connection status and give appropriate response in case the application crash in some cases that it has no prebuild countermeasures.

3. Five in a row validating algorithm
   It is tricky to check if five in a row exist in the game board. Initially, we tried to find same color in five consecutive positions over the whole game board each time calls the algorithm function. But later we realized this way is inefficient and difficult to implement correctly. After doing some research, we learned a better way to do the algorithm. The point is to simulate the game board by a two-dimension array. Then we can get used of the coordinate for each intersection. The algorithm will check the game board in 4 ways. Horizontal, Vertical, skew-left and skew-right. By inputting a base stone coordinate, it finds the same color with the input stone's color over neighbor positions of these four directions. Once the counter reaches 5, the game has a winner.

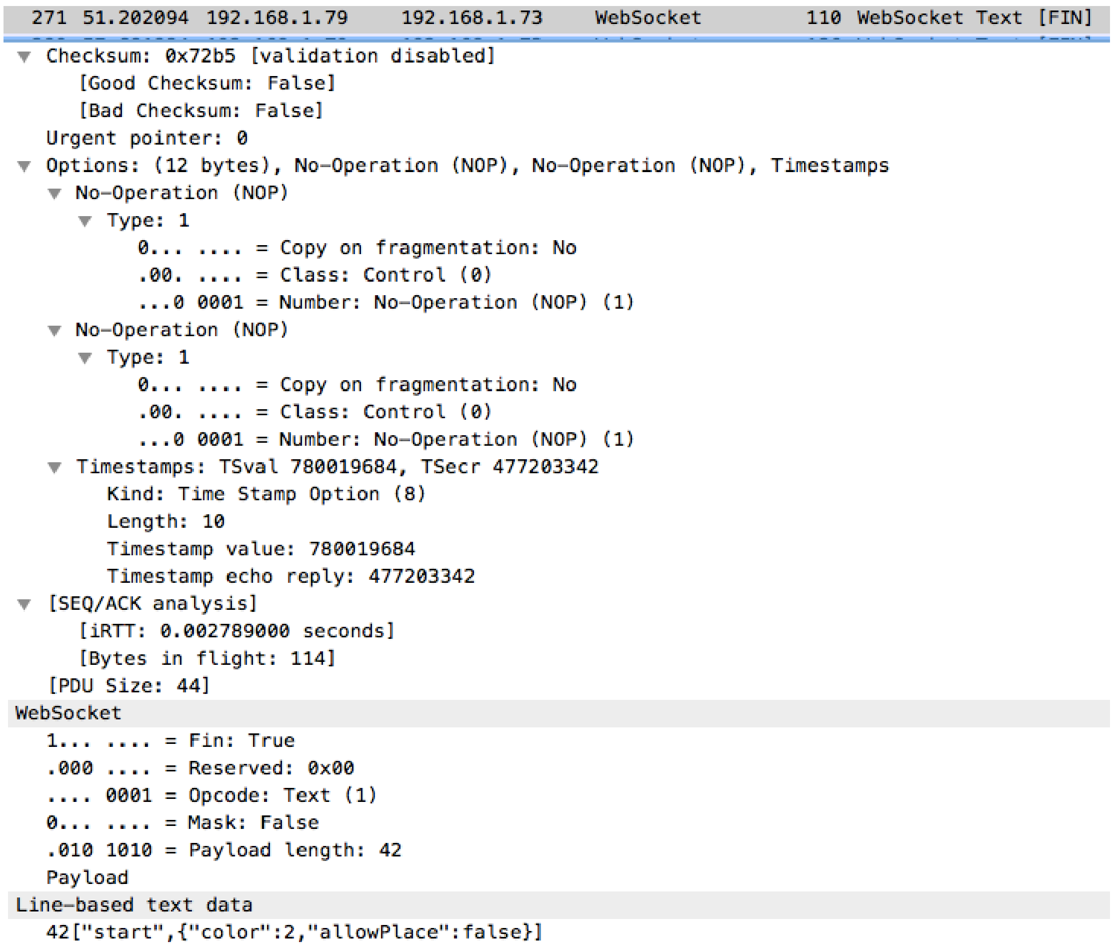
Analysis

Set up HTTP connection do the handshake to setup TCP connections. We are using socket.io library to create socket interface, and use the port number 8080. There are two HTTP connections because we are running two client webpages. This is when two clients connecting to the game and log in. When logging in, the client side will use WebSocket to communicate with server.

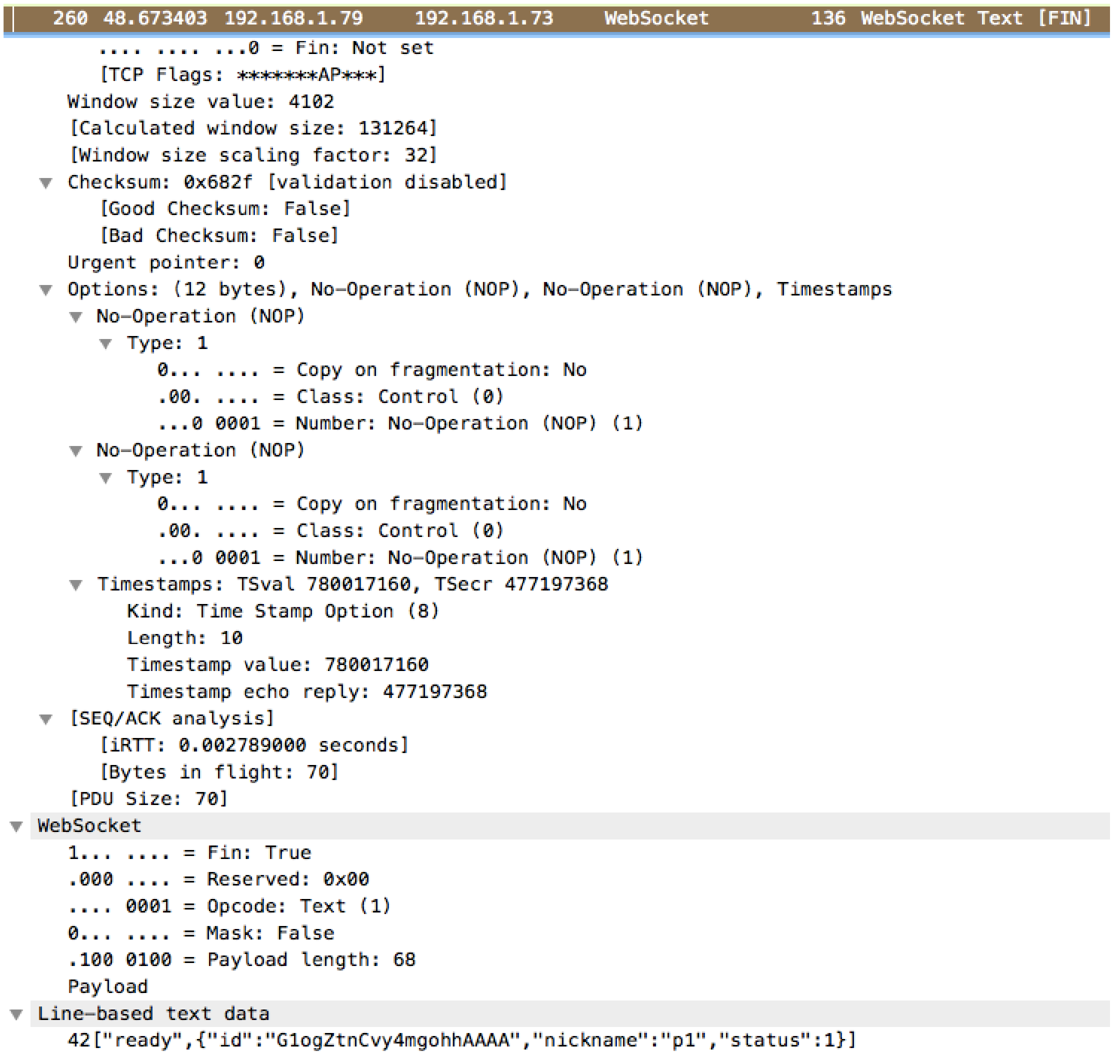


```
232 43.856705 192.168.1.73    192.168.1.79    HTTP    424 GET /socket.io/?EIO=3&transport=polling&t=1449705003407-3&sid=SLcb2MuuPiUg6StrAAAB HTTP/1.1
233 43.856754 192.168.1.79    192.168.1.73    TCP      66 8080 → 51250 [ACK] Seq=1 Ack=359 Win=131392 Len=0 TSval=780012350 TSecr=477196082
234 43.857770 192.168.1.79    192.168.1.73    HTTP    304 HTTP/1.1 200 OK  (text/html)
235 43.859118 192.168.1.79    192.168.1.73    HTTP    477 HTTP/1.1 200 OK  (application/octet-stream)
236 43.860842 192.168.1.73    192.168.1.79    TCP      66 51249 → 8080 [ACK] Seq=468 Ack=138 Win=131616 Len=0 TSval=477196083 TSecr=780012349
237 43.863357 192.168.1.73    192.168.1.79    TCP      66 51246 → 8080 [ACK] Seq=1454 Ack=939 Win=130832 Len=0 TSval=477196087 TSecr=780012351
238 43.863705 192.168.1.73    192.168.1.79    TCP      66 51250 → 8080 [ACK] Seq=359 Ack=412 Win=131344 Len=0 TSval=477196087 TSecr=780012352
```

Then the two clients (players) should be both ready to start playing the game. When one player clicks “ready”, the message will be sent by Web-Socket and tell the other player that “I’m ready to play”. When two clients are both ready, server will send a “start” message to both the clients

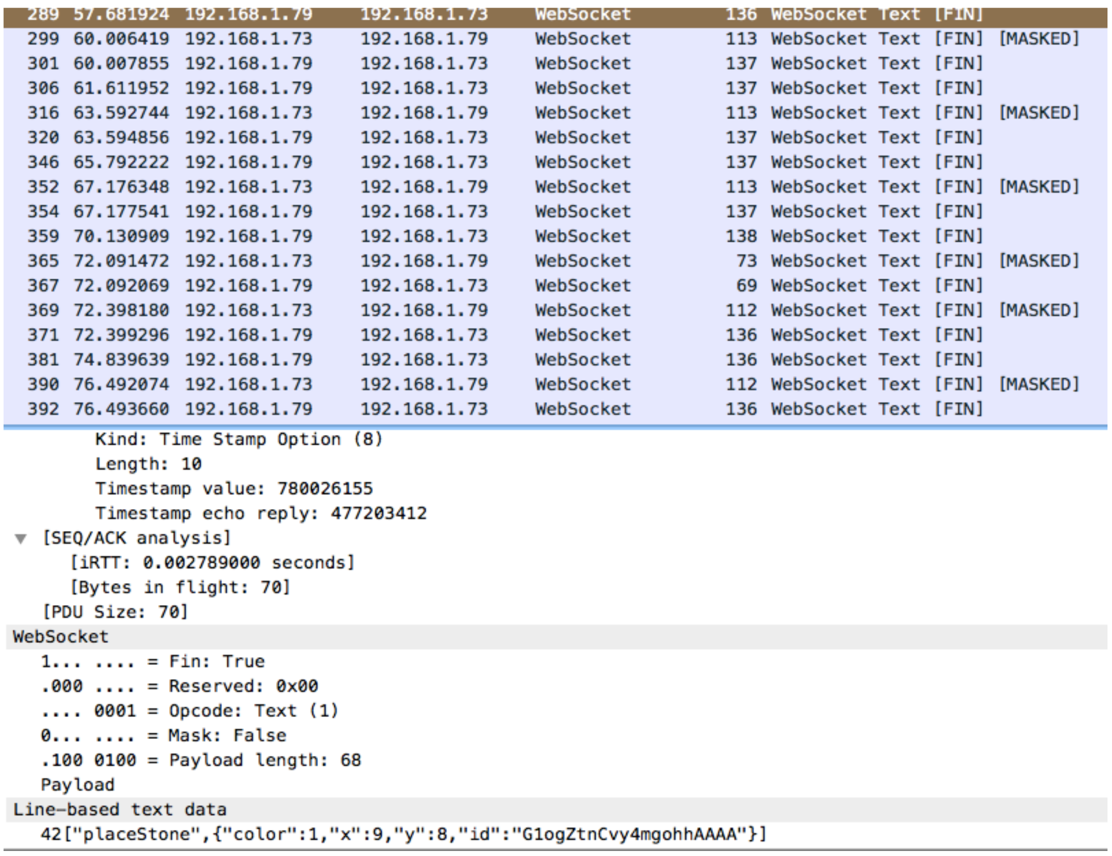


```
 271  51.202094  192.168.1.79     192.168.1.73     WebSocket        110  WebSocket Text [FIN]
▼ Checksum: 0x72b5 [validation disabled]
     [Good Checksum: False]
     [Bad Checksum: False]
  Urgent pointer: 0
▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▼ No-Operation (NOP)
    ▼ Type: 1
         0... .... = Copy on fragmentation: No
         .00. .... = Class: Control (0)
         ...0 0001 = Number: No-Operation (NOP) (1)
  ▼ No-Operation (NOP)
    ▼ Type: 1
         0... .... = Copy on fragmentation: No
         .00. .... = Class: Control (0)
         ...0 0001 = Number: No-Operation (NOP) (1)
  ▼ Timestamps: TSval 780019684, TSecr 477203342
       Kind: Time Stamp Option (8)
       Length: 10
       Timestamp value: 780019684
       Timestamp echo reply: 477203342
▼ [SEQ/ACK analysis]
     [iRTT: 0.002789000 seconds]
     [Bytes in flight: 114]
  [PDU Size: 44]
WebSocket
  1... .... = Fin: True
  .000 .... = Reserved: 0x00
  .... 0001 = Opcode: Text (1)
  0... .... = Mask: False
  .010 1010 = Payload length: 42
  Payload
Line-based text data
  42["start",{"color":2,"allowPlace":false}]
```



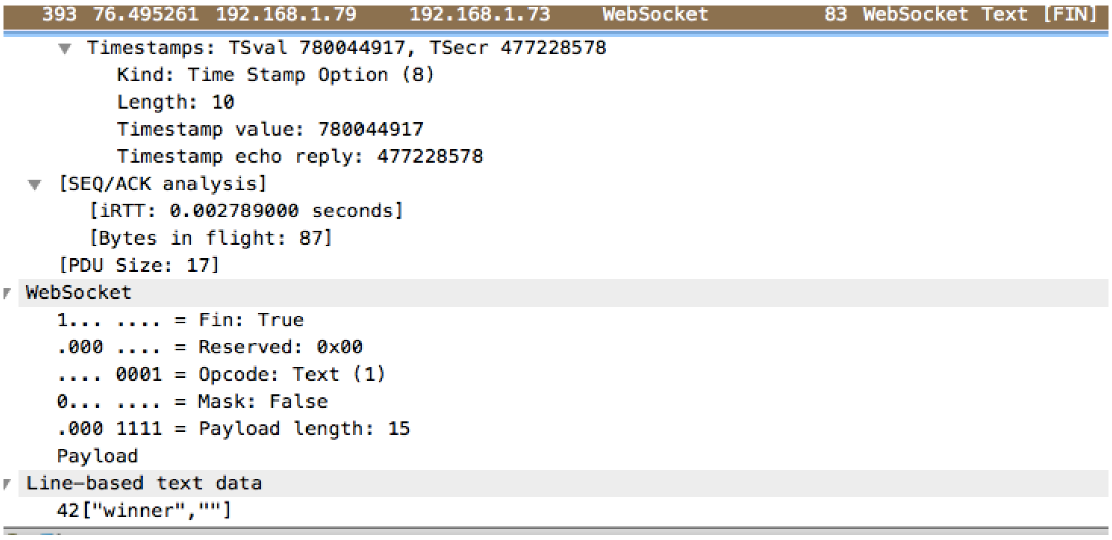
```
  260 48.673403 192.168.1.79      192.168.1.73      WebSocket          136 WebSocket Text [FIN]
        .... .... ...0 = Fin: Not set
        [TCP Flags: *******AP***]
     Window size value: 4102
     [Calculated window size: 131264]
     [Window size scaling factor: 32]
  ▼  Checksum: 0x682f [validation disabled]
        [Good Checksum: False]
        [Bad Checksum: False]
     Urgent pointer: 0
  ▼  Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    ▼  No-Operation (NOP)
      ▼  Type: 1
            0... .... = Copy on fragmentation: No
            .00. .... = Class: Control (0)
            ...0 0001 = Number: No-Operation (NOP) (1)
    ▼  No-Operation (NOP)
      ▼  Type: 1
            0... .... = Copy on fragmentation: No
            .00. .... = Class: Control (0)
            ...0 0001 = Number: No-Operation (NOP) (1)
    ▼  Timestamps: TSval 780017160, TSecr 477197368
          Kind: Time Stamp Option (8)
          Length: 10
          Timestamp value: 780017160
          Timestamp echo reply: 477197368
  ▼  [SEQ/ACK analysis]
        [iRTT: 0.002789000 seconds]
        [Bytes in flight: 70]
     [PDU Size: 70]
▼ WebSocket
     1... .... = Fin: True
     .000 .... = Reserved: 0x00
     .... 0001 = Opcode: Text (1)
     0... .... = Mask: False
     .100 0100 = Payload length: 68
     Payload
▼ Line-based text data
     42["ready",{"id":"G1ogZtnCvy4mgohhAAAA","nickname":"p1","status":1}]
```

All the following actions happened when players are playing the game. Once a player placed a stone, it sent the information to server by WebSocket protocol. The server checked if the winner exists.



```
 289 57.681924 192.168.1.79   192.168.1.73   WebSocket      136 WebSocket Text [FIN]
 299 60.006419 192.168.1.73   192.168.1.79   WebSocket      113 WebSocket Text [FIN] [MASKED]
 301 60.007855 192.168.1.79   192.168.1.73   WebSocket      137 WebSocket Text [FIN]
 306 61.611952 192.168.1.79   192.168.1.73   WebSocket      137 WebSocket Text [FIN]
 316 63.592744 192.168.1.73   192.168.1.79   WebSocket      113 WebSocket Text [FIN] [MASKED]
 320 63.594856 192.168.1.79   192.168.1.73   WebSocket      137 WebSocket Text [FIN]
 346 65.792222 192.168.1.79   192.168.1.73   WebSocket      137 WebSocket Text [FIN]
 352 67.176348 192.168.1.73   192.168.1.79   WebSocket      113 WebSocket Text [FIN] [MASKED]
 354 67.177541 192.168.1.79   192.168.1.73   WebSocket      137 WebSocket Text [FIN]
 359 70.130909 192.168.1.79   192.168.1.73   WebSocket      138 WebSocket Text [FIN]
 365 72.091472 192.168.1.73   192.168.1.79   WebSocket       73 WebSocket Text [FIN] [MASKED]
 367 72.092069 192.168.1.79   192.168.1.73   WebSocket       69 WebSocket Text [FIN]
 369 72.398180 192.168.1.73   192.168.1.79   WebSocket      112 WebSocket Text [FIN] [MASKED]
 371 72.399296 192.168.1.79   192.168.1.73   WebSocket      136 WebSocket Text [FIN]
 381 74.839639 192.168.1.79   192.168.1.73   WebSocket      136 WebSocket Text [FIN]
 390 76.492074 192.168.1.73   192.168.1.79   WebSocket      112 WebSocket Text [FIN] [MASKED]
 392 76.493660 192.168.1.79   192.168.1.73   WebSocket      136 WebSocket Text [FIN]
            Kind: Time Stamp Option (8)
            Length: 10
            Timestamp value: 780026155
            Timestamp echo reply: 477203412
  ▼ [SEQ/ACK analysis]
        [iRTT: 0.002789000 seconds]
        [Bytes in flight: 70]
     [PDU Size: 70]
 WebSocket
    1... .... = Fin: True
    .000 .... = Reserved: 0x00
    .... 0001 = Opcode: Text (1)
    0... .... = Mask: False
    .100 0100 = Payload length: 68
    Payload
 Line-based text data
    42["placeStone",{"color":1,"x":9,"y":8,"id":"G1ogZtnCvy4mgohhAAAA"}]
```

If there’s a player wins, server will send to both clients the winner message and the game finishes.



```
   393 76.495261 192.168.1.79     192.168.1.73     WebSocket          83 WebSocket Text [FIN]
      ▼ Timestamps: TSval 780044917, TSecr 477228578
            Kind: Time Stamp Option (8)
            Length: 10
            Timestamp value: 780044917
            Timestamp echo reply: 477228578
    ▼ [SEQ/ACK analysis]
          [iRTT: 0.002789000 seconds]
          [Bytes in flight: 87]
       [PDU Size: 17]
 ▼ WebSocket
       1... .... = Fin: True
       .000 .... = Reserved: 0x00
       .... 0001 = Opcode: Text (1)
       0... .... = Mask: False
       .000 1111 = Payload length: 15
       Payload
 ▼ Line-based text data
       42["winner",""]
```

Conclusion

This project has successfully deployed the Websocket (RFC 6455) protocol and implemented the web application over Local Area Network. During the development process, many challenges are encountered. So much time is cost to solve many problems. And that is why, a lot of network knowledge and programming skills are learned from this project.

Reference

1. http://www.hongkiat.com/blog/node-js-server-side-javascript/

2. https://nodejs.org/en/docs/

3. http://socket.io/get-started/chat/

4. https://www.npmjs.com/package/socket.io

5. http://danielnill.com/nodejs-tutorial-with-socketio/