Portland State University

Internet Relay Chat Class Project

Abstract

This memo describes the communication protocol for an IRC-style client/server system for the Internetworking Protocols class at Portland State University.

Table of Contents

1. Introduction

This specification describes a simple Internet Relay Chat (IRC) protocol by which clients can communicate with each other. This system employs a central server which ''relays'' messages that are sent to it to other connected users.
Users can join rooms, which are groups of users that are subscribed to the same message stream. Any message sent to that room is forwarded to all users currently joined to that room.
Users can also send private messages directly to other users.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119]. In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.

3. Basic Information

All communication described in this protocol takes place over TCP/IP, with the server listening for connections on port 3000. Clients connect to this port and maintain this persistent connection to the server. The client can send messages and requests to the server over this open channel, and the server can reply via the same. This messaging protocol is inherently asynchronous - the client is free to send messages to the server at any time, and the server may asynchronously send messages back to the client.

The server MAY choose to allow only a finite number of users and rooms, depending on the implementation and resources of the host system.

4. Message Infrastructure

4.1. Generic Message Display

```
  if (receiverName == 'all')
     $('#messages').append($('<li>').text(FormatDate(date) + '  ' +
userName + ': ' + msg));
  else {
     $('#messages').append($('<li>').text(FormatDate(date) + '  ' + '
Private chat: [ ' + userName + '  =>  ' +  receiverName + ' ] : ' +
msg).css({
              'color': '#000066',
              'font-weight': ‘bolder’}
```

4.2. Field Definitions

messages - the id of the div element for easier select and changes the inter elements using Jquery.

.append() - Jquery api used to insert content, specified by the parameter, to the end of each element in the set of matched elements.

.text() - Jquery api used to get the combined text contents of each element in the set of matched elements, including their descendants, or set the text contents of the matched elements.

userName - the passed in arguments send by server contain the value of the current user’s name according to the socket id.

receiverName - the passed in arguments send by server contains the value of the target user’s name.

msg - the passed in arguments send by the server contains the message send to the room

5. Connection

5.1. Connection

```
io.on('connection', function (socket){…}
```

5.1.1. Usage

Check chat message flag to ensure the server receive socket with right connecting flag.

6 Client Message

6.1 Create users

```
socket.on('new user', (userName){
…
…
io.sockets.in(currentSocket.room).emit('msg user join', userName,
currentSocket.room);
…
}
```

6.2 Add in room1 atomically

```
           currentUser.push(userName);
           currentSocket.push(socket);
           currentUser_room.push('room1');
           currentSocket.room = 'room1';
           socket.join(‘room1');
```

6.3 Update userList

```
    if ($('#userList').text().indexOf(name) == -1) {
     $('#userList').append($('<li>').text(name + ' in ' + room).css({
                        'font-weight': 'bolder' }));
  }} else if (userName != name && $('#userList').text().indexOf(name)
== -1) {
                $('#userList').append($('<li>').text(name + ' in ' +
room).css({
                    'font-weight': 'bolder'
                }));
} else if (userName != name && $('#userSelect').val().indexOf(name)
== -1) {
  $('#userSelect').append($('<option></option>').attr('value',
name).text(name));}
```

6.4. Listing Rooms

```
  <li><a id="room1" onclick="switchRoom(this.id)"
href="javascript:void(0)">Room1</a></li>
  <li><a id="room2" onclick="switchRoom(this.id)"
href="javascript:void(0)">Room2</a></li>
  <li><a id="room3" onclick="switchRoom(this.id)"
href="javascript:void(0)">Room3</a></li>
```

6.4.1. Usage

Show a list of all of the rooms currently can be used.

6.5. Switching(Joining) Rooms

```
socket.join(newroom);
io.sockets.in(currentRoom).emit('msg user leave',
currentUser[currentSocket.indexOf(socket)], currentSocket.room);

currentSocket.room = newroom;
currentUser_room[currentSocket.indexOf(socket)] = newroom;

io.sockets.in(newroom).emit('msg user join',
currentUser[currentSocket.indexOf(socket)], currentSocket.room);
currentUser.room = newroom;
```

6.5.1. Usage

Sent by the client to switch a chat room. Client in only one room at same time.

6.6. Leaving a Room

```
io.sockets.in(currentRoom).emit('msg user leave',
currentUser[currentSocket.indexOf(socket)], currentSocket.room);
```

6.6.1. Usage

Sent by the client to leave a chat room.
Upon receiving this message the server MUST remove the client from the specified room. The server SHOULD ignore leave requests when the client is not currently a member of the specified room.

6.7 Sending General Messages

```
socket.on('chat message', (userName, receiverName, msg, date) => {
if (receiverName == 'all') {
           socket.broadcast.to(currentSocket.room).emit('chat
message', userName, receiverName, msg, date);}
```

6.7.1. Usage

Sent by a client to send a text message to either a room or another user.

6.8 Sending Private Message

```
if (currentUser_room[currentUser.indexOf(receiverName)] ==
currentSocket.room) {

currentSocket[currentUser.indexOf(receiverName)].emit('chat message',
userName, receiverName, msg, date);
            }
            else {
    socket.emit('error no user', receiverName, currentSocket.room);}
```

6.8.1 Error Private Messages

```
  socket.emit('error no user', receiverName, currentSocket.room);
```

6.8.2. Usage

The server send to the client. If one client send message to another client who is not found in the same room, it SHOULD NOT consider the connection as terminated.

6.9 Disconnect

```
io.emit('msg user leave’,
currentUser[currentSocket.indexOf(socket)]);
currentUser.splice(currentSocket.indexOf(socket), 1);
currentSocket.splice(currentSocket.indexOf(socket), 1);
currentUser_room.splice(currentSocket.indexOf(socket), 1);

socket.on('disconnect', () => {
            $(window).unbind('beforeunload');
            alert('Lost connection!!\nRefresh page.');
            window.location.reload();
```

7. Server Messages

7.1. Listing Response

```
console.log(date + ' - ' + userName + ' to ' + receiverName + ' : ' +
msg);
console.log(userName + " join " + currentSocket.room);
console.log('update ' + currentUser[i] + ' in ' +
currentUser_room[i]);
console.log(currentUser[currentSocket.indexOf(socket)] + ' leave the
room’);
console.log('listening on http://localhost:3000/');
```

7.1.1. Usage

Generic listing response message sent by the server to inform the client of a list. Used for both listing rooms and listing users in a room.

8. Error Handling

As using Javascript, only created error hander for private message sending which message only be send when special user in special room. Send error message when no user matched in same room.

9. "Extra" Features Supported

Note that private messaging is supported in addition to meeting the other remaining project criteria.

10. Conclusion & Future Work

This specification provides a generic message passing framework for multiple clients to communicate with each other via a central forwarding server.
Without any modifications to this specification, it is possible for clients to devise their own protocols that rely on the text-passing system described here. For example, transfer of arbitrary binary data can be achieved through transcoding to base64. Such infrastructure could be used to transfer arbitrarily large files, or to establish

secure connections using cryptographic transport protocols such as Transport Layer Security (TLS).

11. Security Considerations

Messages sent using this system have no protection against inspection, tampering or outright forgery. The server sees all messages that are sent through the use of this service. 'Private' messaging may be easily intercepted by a 3rd party that is able to capture network traffic. Users wishing to use this system for secure communication should use/implement their own user-to-user encryption protocol.

12. IANA Considerations

None

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

13. Acknowledgments

This document was prepared using index.html

Authors' Addresses
Jingtao Cheng & Jiacheng Zhao
Portland State University Computer Science 1825 SW Broadway,
Portland, OR 97201
Email: cheng@pdx.edu jiacheng@pdx.edu