Technical Specification - Lin-Kei Tseng

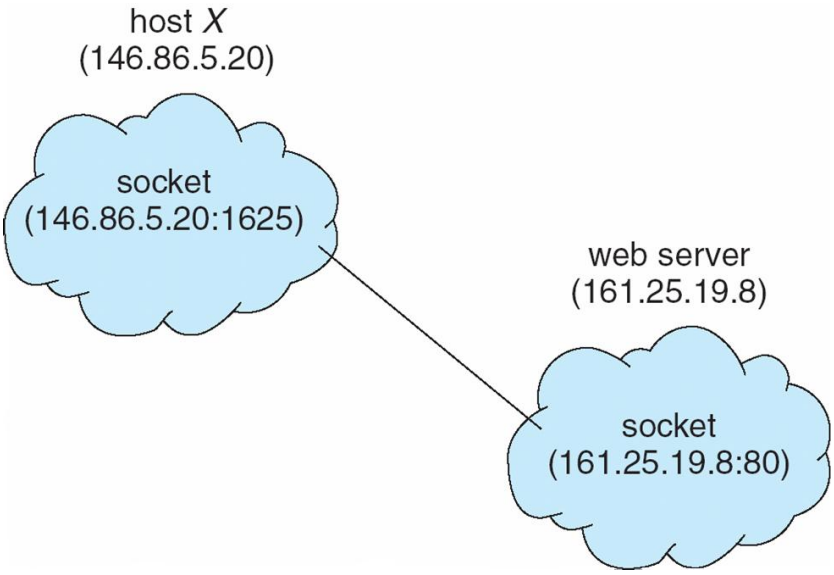
3.6 Communication in Client–Server Systems (http://codex.cs.yale.edu/avi/os-book/OS9/)

There are four kinds of techniques for communication in Client-Server(*Sockets, Remote Procedure Calls, Pipes, Remote Method Invocation (Java)***), and I will focus on Sockets to build a One-on-One chat room and a multi-chat room .**

On textbook, A socket is defined as an endpoint for communication. Communication consists between **a pair of sockets**. That is to say, we need to create a least a port (listening) for a pair of sockets.

Concatenation of IP address and port – a number included at start of message packet to differentiate network services on a host. For example in C, The socket 161.25.19.8:**1625** refers to **port 1625** on **host** 161.25.19.8. Special **IP address 127.0.0.1** (loopback) to refer to system on which process is running. All ports below 1024 are well known, used for standard services. This mechanism allows a client and server on the same host to communicate using the TCP/IP protocol.

The structure of socket in C:



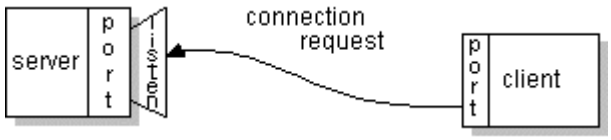
Sockets in Java:

There are three types of sockets: Connection-oriented (**TCP**), Connectionless (UDP),and Multicast Socket class– data can be sent to multiple recipients. I use TCP Socket structure in my program example.

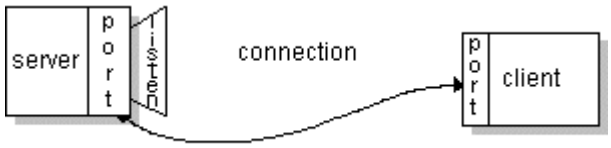
Definition:
A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the **TCP layer** can identify the application that data is destined to be sent to.

What Is a Socket?

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. **The server** just waits, listening to the socket for a client to make a connection request. **On the client-side:** The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.



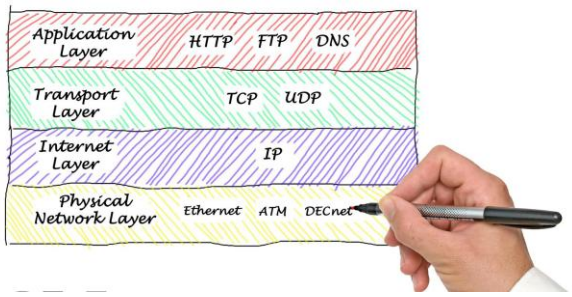
If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets. **An endpoint is a combination of an IP address and a port number**. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple **connections** between your host and the server.

What is a TCP ?

TCP (Transmission Control Protocol)is a connection-based protocol that provides a reliable flow of data between two computers. Example application: HTTP, FTP, Telnet, SMTP.



The Concept of Multicast Sockets Communication

"LOL" Multi-User Chat Room

I use **node.js and socket.io** to build a online version multi-chat room. Because **socket.io** has a library which server and client can use at same time, socket.io can create a chat room with sockets in a short time without occupy lots resource and it is easier than Java version.

The example of code concept is following:

app.js

```
io.on('connection', function(socket){
    //新user
     socket.on('add user',function(msg){
        socket.username = msg;
        console.log("new user:"+msg+" logged.");
        io.emit('add user',{
            username: socket.username
        });
    });
});
```

If there is a new client join, the program will execute "connection's" callback function, and a socket will be sent into system. The system can use this socket to communicate with client. "socket.on" is an event for listening client-side, and it is like the function in jQuery.

For example: `$('#btn').on('click',function(...))`

After we add an event " add user ", the event will wait for be actived by client-side. The new user's "username" will be transferred into this event, and the username is added in "socket" we created before. This socket can identify the different users.

socket.on('add user',function(msg)

The **"io.emit"** is used to send data to all clients who connect with server. "add user" event has a variable with data which will be sent to all clients.

io.emit('add user',{username: socket.username});

The server's code is following:

index.html

```
var socket = io();
var name = prompt("請輸入暱稱","guest");

socket.emit("add user",name);

socket.on('add user',function(data){
  appendMessage(data.username+"已加入");
});
```

After clients enter their username, the system will use "socket.emit" sent message to server. The server's **socket.on ('add user',....)** will be actived, and then the system can record every action happed on the event "add user". Therefore, server will sent message said "New user: xxx logged " on the screen.

```
socket.emit("add user", name);

socket.on('add user', function(data) {

    appendMessage(data.username + " logged...");
```

On server-side we need to avoid sent the same message to myself to see the same message twice. Using `socket.broadcast.emit(...)` can sent data to all users exclude myself. The `io.emit` sends to all socket data directly.

Summary function:

Sent event: `socket.emit` , `io.emit(server-side)`

Get event: `socket.on`