TCP Socket Programming

Last Week: UDP Socket programming

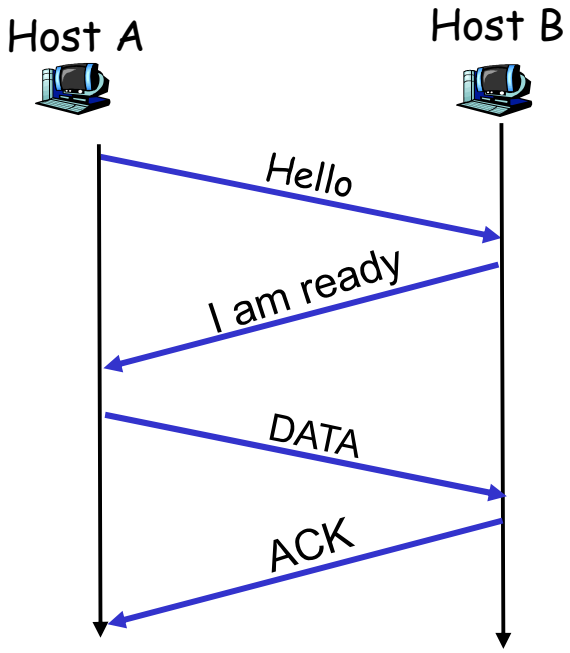
- If we use UDP for transmission, the data can be lost and disordered.
- DatagramSocket is used for both client and server: data is converted to group of bytes before sending.
- Client/Server program has 3 steps: To send packet to Client, Server has to know address and port of Client
- We need to make sure the received data is the one we want.

Contents

- TCP introduction
- TCP Socket client - Socket
- TCP Socket server – ServerSocket
- Data Streams

TCP

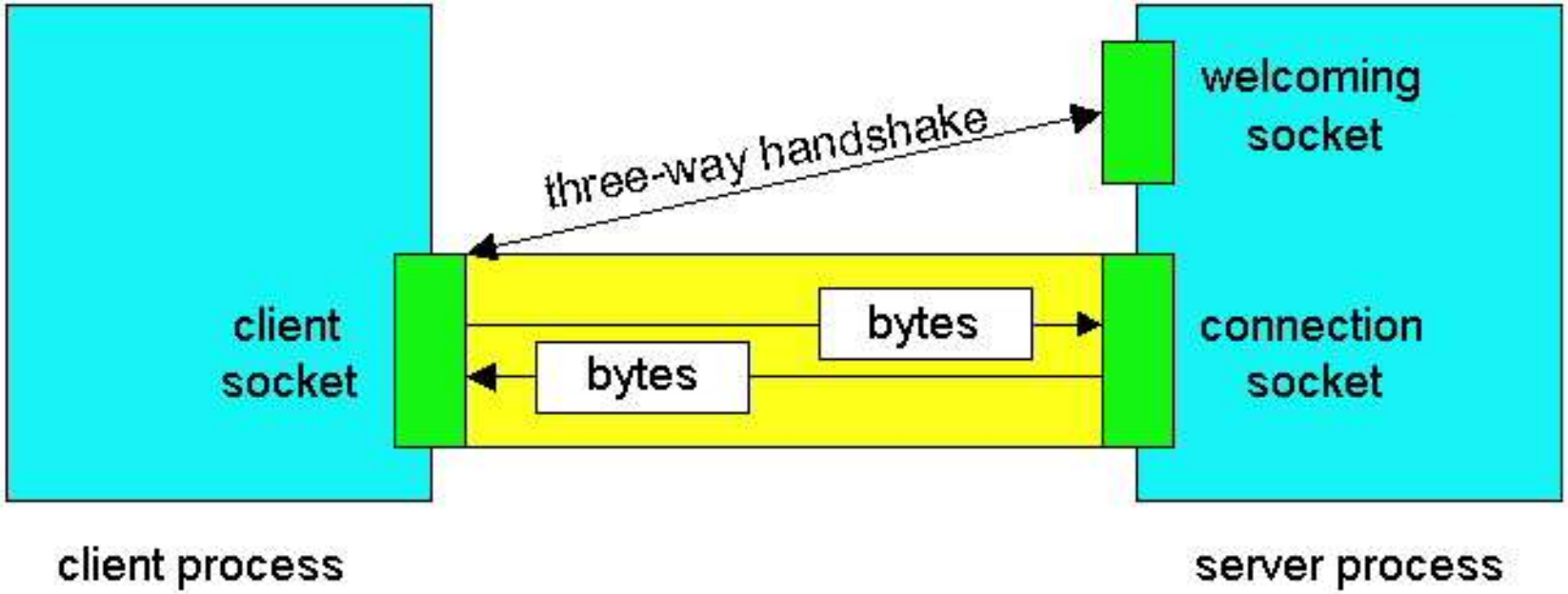
Source port			Destination port
Sequence Number			
Acknowledge Number			
Offset	Reserved	Flags	Window
Checksum			Urgent pointer
Options			Padding
Start of Data			



-Connection is set-up between Client and Server

-No lost and in order

Socket


three-way handshake
welcoming socket
client socket
bytes
bytes
connection socket
client process
server process

TCP Socket

- java.net.Socket
  - Used to create Socket at Client. Constructor:
    - Socket(String host, int port)
  - Methods: From this object we can create some useful objects
    - InputStream getInputStream()
    - OutputStream getOutputStream()
    - close()
- java.net.ServerSocket
  - Used to create a ServerSocket which will accept a connection from Client and create a Socket at Server. Constructor:
    - ServerSocket(int port)
  - Methods
    - Socket Accept()

Socket: Constructors and Methods

- Constructors
  - `Socket(InetAddress server, int port);`
  - `Socket(InetAddress server, int port, InetAddress local, int localport);`
  - `Socket(String hostname, int port);`
- Methods
  - `void close();`
  - `InetAddress getInetAddress();`
  - `InetAddress getLocalAddress();`
  - `InputStream getInputStream();`
  - `OutputStream getOutputStream();`

TCP Socket client

- Step 1: Create object Socket which connects to Server
  - **Socket** s = new **Socket** ("java.sun.com", 8189)
- Step 2: Create appropriate stream for reading/sending data from/to Server
  - BufferedReader in = new **BufferedReader**(new **InputStreamReader**(s.**getInputStream** ()));
  - BufferedWriter out = new **BufferedWriter**(new **OutputStreamReader**(s.**getOutputStream** ()));
  - Java has many different build-in streams. Depending on kind of data transmission, we need to choose appropriate streams.
- Step 3: Reading/Sending data from/to Server
  - String str = in.**readLine**();
  - out.**println** ("Echo:" + str + "\r");
- Step 4: Close Socket: s.**close**();

TCP Socket Server

- Step 1: Create object ServerSocket at a specific port

  ServerSocket ss = new **ServerSocket**(8189)

- Step 2: Waiting for request message from Client and create a Socket which connects to Client

  Socket con = ss.accept();

- Step 3: Create reading/sending stream from/to Client from this Socket as previous slide
  - BufferedReader in = new **BufferedReader**(new **InputStreamReader**(con.**getInputStream** ()));
  - BufferedWriter out = new **BufferedWriter**(new **OutputStreamReader**(con.**getOutputStream** ()));

- Step 4: Close Socket and ServerSocket:
  - con.**close**(); ss.close();

Example 1

- Write client program connecting to server program by **TCP**. Client sends a string, which is inputted from keyboard, to server. Server will convert all letters of this string to upper-case letters and send back to Client. Client will print the data received from Server to Console.

TCPClient.java

```
try{
        Socket s = new Socket("localhost",1234);

        BufferedReader Network_in = new BufferedReader(new
InputStreamReader(s.getInputStream()));
        BufferedWriter Network_out = new BufferedWriter(new
OutputStreamWriter(s.getOutputStream()));

        Scanner keyboard = new Scanner(System.in);
        System.out.println("Please input a string:");
        String data = keyboard.nextLine();
        Network_out.write(data+"\r\n");
        Network_out.flush();
        String result = Network_in.readLine();
        System.out.println("Data from Server:"+result);
        s.close();
}catch(Exception e){}
```

Step1: Connect to Server

Step2: Create data Streams

Step3: Implement the function of Client

Step4: Close the Socket

TCPServer.java

Step1: Create the Socket and waiting for connecting from Client

```
try {
        ServerSocket ss = new ServerSocket(1234);
        Socket con = s.accept();
        BufferedReader in= new BufferedReader(new InputStreamReader
(con.getInputStream()));
        BufferedWriter out = new BufferedWriter(new OutputStreamWriter
(con.getOutputStream()));

        String rdata = in.readLine();
        System.out.println(rdata);
        out.write(rdata.toUpperCase()+"\r\n");
        out.flush();

        con.close();
        ss.close();
        }
catch (Exception e) {}
```

Step3: Implement the function of Client

Step4: Close Socket and ServerSocket

Example 2

- Write client program connecting to server program by **TCP**. Client sends two integers, which are inputted from keyboard, to server. Server will calculate the result = number1 – number2 and send the result back to Client. Client will print the result received from Server to Console.

TCPClient.java

```
try{
  Socket s = new Socket("localhost",6789);

  DataInputStream Network_in = new DataInputStream(s.getInputStream());
  DataOutputStream Network_out = new DataOutputStream(s.getOutputStream());

  Scanner keyboard = new Scanner(System.in);
  System.out.println("Please input two integers:");
  int data1 = keyboard.nextInt();
  int data2 = keyboard.nextInt();
  Network_out.writeInt(data1);
  Network_out.writeInt(data2);
 int result = Network_in.readInt();
  System.out.println("Data from Server:"+ result);

  s.close();
}catch(Exception e){}
```

TCPServer.java

```
try {
    ServerSocket ss = new ServerSocket(6789);
    Socket con = ss.accept();

    DataInputStream in = new DataInputStream(con.getInputStream());
    DataOutputStream out = new DataOutputStream(con.getOutputStream());

    int number1 = in.readInt();
    int number2 = in.readInt();
    int result= number1 - number2;
    out.writeInt(result);

    con.close();
    ss.close();
  }catch (Exception e) {}
```

Input and Output Streams

- Java supports many input and output streams, depending on the kind of data.
- Client and Server should use the same streams.
- BufferedReader and BufferedWriter: character-oriented streams
  - Textual content
- DataInputStream and DataOutputStream: byte-oriented streams
  - Support primitive java data types: int, float, char: readInt()/writeInt()

SSL Socket

- javax.net.ssl.*
- Making a connection to **SSL server** for authentication of some application protocols
  - POP3, SMTP,…
- Example

```
SSLSocketFactory sslsocketfactory = (SSLSocketFactory)
SSLSocketFactory.getDefault();
SSLSocket pop3Socket = (SSLSocket)
sslsocketfactory.createSocket(host, port);
pop3Socket.setSoTimeout(15000);
BufferedReader in = new BufferedReader(new
        InputStreamReader(pop3Socket.getInputStream()));
BufferedWriter out = new BufferedWriter(new
        OutputStreamWriter(pop3Socket.getOutputStream()));
```

Supporting many clients simultaneously

- Can you rewrite your program to allow a TCP server support many TCP clients simultaneously?
  - Why?
  - How?

Summary

- If we use TCP for transmission, the data is delivered in order and no lost happens.
- Java hides all complex steps of setting up connection and data transmission between Client and Server.
- Client/Server program has 4 steps

Homework

- Please finish all assignments
- Next week is **Lab class for my group**.
- Next theory is multithread and advanced techniques supporting multithread.