Socket Programming

1. What is socket? UDP and TCP differences

A socket is a mechanism for allowing communication between processes, be it programs running on the same machine or different computers connected on a network. More specifically, Internet sockets provide a programming interface to the network protocol stack that is managed by the operating system. Using this API, a programmer can quickly initialise a socket and send messages without having to worry about issues such as packet framing or transmission control. There are a number of different types of sockets available, but we are only really interested in two specific Internet sockets. These are: • Stream sockets • Datagram sockets

What differentiates these two socket types is the transport protocol used for data transmission. A stream socket uses the Transmission Control Protocol (TCP) for sending messages. TCP provides an ordered and reliable connection between two hosts. This means that for every message sent, TCP guarantees that the message will arrive at the host in the correct order. This is achieved at the transport layer so the programmer does not have to worry about this, it is all done for you. A datagram socket uses the User Datagram Protocol (UDP) for sending messages. UDP is a much simpler protocol as it does not provide any of the delivery guarantees that TCP does. Messages, called datagrams, can be sent to another host without requiring any prior communication or a connection having been established. As such, using UDP can lead to lost messages or messages being received out of order. It is assumed that the application can tolerate an occasional lost message or that the application will handle the issue of retransmission. There are advantages and disadvantages to using either protocol and it will be highly dependent on the application context. For example, when transferring a file you want to ensure that, upon receipt, the file has not become corrupted. TCP will handle all the error checking and guarantee that it will arrive as you sent it. On the other hand, imagine you are sending 1000 messages detailing player position data every second in a computer game. The application will be able to tolerate missing messages here so UDP would be more suitable.

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

The communication that occurs between the client and the server must be reliable. That is, no data can be dropped and it must arrive on the client side in the same order in which the server sent it.

TCP provides a reliable, point-to-point communication channel that client-server applications on the Internet use to communicate with each other. To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. To communicate, the client and the server each reads from and writes to the socket bound to the connection.

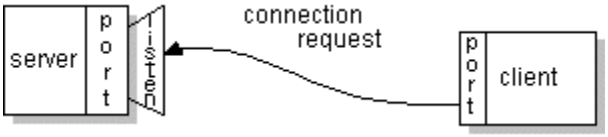
Addressing

Using Internet Sockets, we can identify a host using an IP address and a port number. The IP address uniquely identifies a machine while the port number identifies the application we want to contact at that machine. There are a range of well known port numbers, such as port 80 for HTTP, in the range 0 - 1023. When choosing port numbers, anything above the well know port number range should be fine, but you cannot guarantee that another application will not be already using it.

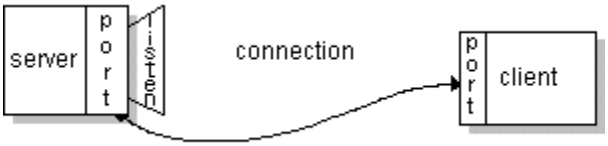
5. socket features - how it works

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.

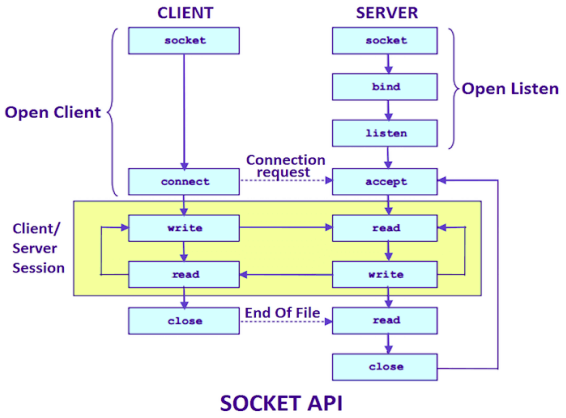


If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection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



6. **Java Socket Programming**

Java provides a collection of classes and interfaces that take care of low-level communication details between the client and the server.

These are mostly contained in the java.net package, so we need to make the following import:

import java.net.*;

We also need the java.io package which gives us input and output streams to write to and read from while communicating:

import java.io.*;

Java.net package

Socket Class methods are found in Java. A socket is bound to be a port number so that the TCP recognizes the port number in which the data is to be sent. Java provides a set of classes one of which is java.net. This is used for the fast development of network applications. Key classes, interfaces, and exceptions that are present in java.net package simplify the complexity involved in creating client and server programs

The **java.net.Socket** class represents the socket that both the client and the server use to communicate with each other. The client obtains a Socket object by instantiating one, whereas the server obtains a Socket object from the return value of the accept() method.

More Info:
https://courses.cs.washington.edu/courses/cse341/98au/java/jdk1.2beta4/docs/api/java/net/Socket.html

The **java.net.ServerSocket** class is used by server applications to obtain a port and listen for client requests

More Info:https://courses.cs.washington.edu/courses/cse341/98au/java/jdk1.2beta4/docs/api/java/net/ServerSocket.html

The following steps occur when establishing a TCP connection between two computers using sockets –

1. The server instantiates a **ServerSocket** object, denoting which port number communication is to occur on.
2. The server invokes the **accept**() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
3. After the server is waiting, a client **instantiates a Socket object**, specifying the server name and the port number to connect to.
4. The constructor of the Socket class attempts to **connect the client** to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.
5. On the server side, **the accept() method returns a reference** to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream

Simple Server side socket program:

```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
```

```
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Simple program for the client

```
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Task

one-way Client and Server setup where a Client connects, sends messages to server and the server shows them using socket connection.

Server Side

```
//A Java program for a Server
import java.net.*;
import java.io.*;

public class Server
{
	//initialize socket and input stream
	private Socket		socket = null;
	private ServerSocket server = null;
	private DataInputStream in  = null;

	// constructor with port
	public Server(int port)
	{
		// starts server and waits for a connection
		try
		{
			server = new ServerSocket(port);
			System.out.println("Server started");

			System.out.println("Waiting for a client ...");

			socket = server.accept();
			System.out.println("Client accepted");

			// takes input from the client socket
			in = new DataInputStream(
				new BufferedInputStream(socket.getInputStream()));

			String line = "";

			// reads message from client until "Over" is sent
			while (!line.equals("Over"))
			{
				try
				{
					line = in.readUTF();
					System.out.println(line);

				}
				catch(IOException i)
				{
					System.out.println(i);
				}
			}
			System.out.println("Closing connection");

			// close connection
			socket.close();
			in.close();
		}
		catch(IOException i)
```

```
                {
                        System.out.println(i);
                }
        }

        public static void main(String args[])
        {
                Server server = new Server(5000);
        }
}
```

Client Side

```
// A Java program for a Client
import java.net.*;
import java.io.*;

public class Client
{
        // initialize socket and input output streams
        private Socket socket           = null;
        private DataInputStream input = null;
        private DataOutputStream out    = null;

        // constructor to put ip address and port
        public Client(String address, int port)
        {
                // establish a connection
                try
                {
                        socket = new Socket(address, port);
                        System.out.println("Connected");

                        // takes input from terminal
                        input = new DataInputStream(System.in);

                        // sends output to the socket
                        out = new DataOutputStream(socket.getOutputStream());
                }
                catch(UnknownHostException u)
                {
                        System.out.println(u);
                }
                catch(IOException i)
                {
                        System.out.println(i);
                }
```

```
                // string to read message from input
                String line = "";

                // keep reading until "Over" is input
                while (!line.equals("Over"))
                {
                        try
                        {
                                line = input.readLine();
                                out.writeUTF(line);
                        }
                        catch(IOException i)
                        {
                                System.out.println(i);
                        }
                }

                // close the connection
                try
                {
                        input.close();
                        out.close();
                        socket.close();
                }
                catch(IOException i)
                {
                        System.out.println(i);
                }
        }

        public static void main(String args[])
        {
                Client client = new Client("127.0.0.1", 5000);
        }
}
```

References

https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

https://www.tutorialspoint.com/java/java_networking.htm

https://www.javatpoint.com/socket-programming

https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

https://www.edureka.co/blog/socket-programming-in-java/

https://www.baeldung.com/a-guide-to-java-sockets

https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_71/rzab6/howdosockets.htm

https://medium.com/platform-engineer/web-api-design-35df8167460

https://www.infoworld.com/article/2853780/socket-programming-for-scalable-systems.html