NETWORK PROGRAMMING IN JAVA USING SOCKETS

Prerna Malik, Poonam Rawat
Student, Dronacharya College of Engineering, Gurgaon, India

Abstract- **Network programming refers to writing programs that could be processed across various devices which are connected to each other via a network. Network programming is similar to socket programming or Client-Server programming. It basically uses the Client Server model. In Client-Server programming there are two different programs or process, one which initiates communication called Client process and other who is waiting for communication to start called Server process. Sockets provide the communication mechanism between two computers. A socket is an endpoint of a two-way communication link between two programs running on the network. A client program creates a socket on its end of the communication and attempts to connect that socket to a server. When the connection is made, the server creates a socket object on its end of the communication. The client and server can now communicate by writing to and reading from the socket. This paper represents information about Network programming using java. Network programming is an essential factor to perceive the implications of communication work across processes based on internet. In this we describe about the programming code involved in client-server communications and different types of sockets used in such communication model.**

I. INTRODUCTION

The term *network programming* refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network. Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server. When the connection is made, the server creates a socket object on its end of the communication. The client and server can now communicate by writing to and reading from the socket. The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand. The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them. The *java.net* package supports two common network protocols:

- ***TCP*****:** TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically over the Internet Protocol (IP), and is referred as TCP/IP.

- ***UDP***: UDP stands for User Datagram Protocol, a connectionless protocol that allows for packets of data to be transmitted between applications.

II. OVERVIEW OF SOCKET PROGRAMMING

Sockets were developed in 1981 at the University of California, Berkeley. The project was sponsored by ARPA (Advanced Research Projects Agency) in 1980. Initially, in 1983, the sockets were referred as Berkeley Sockets. The main objective was the transport of TCP/IP software to UNIX. In 1986, AT&T introduced the Transport Layer Interface (TLI) with socket like functionality, which was more network independent. UNIX includes both TLI and Sockets after SVR4.[3]

2.1 Socket definition:

A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent

the connection between a client program and a server program. The java.net package provides two classes--Socket and ServerSocket--that implement the client side of the connection and the server side of the connection, respectively.

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server. When the connection is made, the server creates a socket object on its end of the communication. The client and server can now communicate by writing to and reading from the socket.

2.1.1. TCP sockets

- TCP Is a byte-stream
- During data packet transmission, no packetizing and addressing is required by application.
- Formatting has to be provided by application.
- Two or more successive data sends on the pipe connected to socket may be combined together by TCP in a single packet

2.1.2. UDP sockets

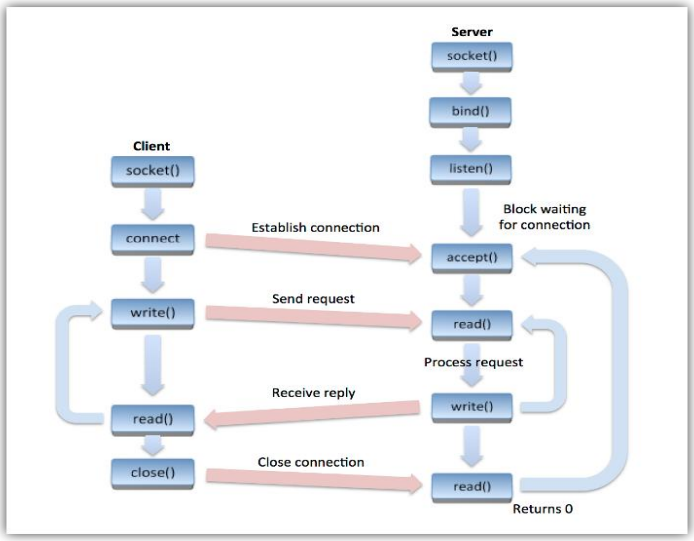
- UDP is packet-oriented
- Information sent in packet format as needed by application.
- Every packet requires address information.
- Lightweight, no connection required.
- Overhead of adding destination address with each packet

2.2 Steps for Establishing a TCP Connection between Two Computers Using Sockets:

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

- The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
- The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a Socket object, specifying the server name and port number to connect to.
- The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.
- 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.
- TCP is a two-way communication protocol, so data can be sent across both streams at the same time. There are following useful classes providing complete set of methods to implement sockets.

Workflow of a socket:



[10]

2.3. The classes, interfaces and Exceptions in socket programming

Table 1: The classes in socket communication

ContentHandler
DatagramSocketImpl
DatagramPacket
DatagramSocket
HttpURLConnection
InetAddress
MulticastSocket
ServerSocket
Socket
SocketImpl
URL
URLEncoder
URLStreamHandler
URLConnection

Table 2: The interfaces used in socket programming

ContentHandlerFactory
FileNameMap
SocketImplFactory
URLStreamHandlerFactory

Table 3: Exceptions in socket programming

BindException
ConnectException
MalformedURLException
NoRouteToHostException
ProtocolException
SocketException
UnknownHostException
UnknownServiceException

2.4 ServerSocket Class Methods:

Four constructors are contained in The ServerSocket class:

1. **public ServerSocket(int port) throws IOException:** Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.

2. **public ServerSocket(int port, int backlog) throws IOException:** Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.[8]

3. **public ServerSocket(int port, int backlog, InetAddress address) throws IOException:** Similar to the previous constructor, the InetAddress parameter specifies the local IP address to bind to. The InetAddress is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addresses to accept client requests on.[7]

4. **public ServerSocket() throws IOException:** Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket. If the ServerSocket constructor does not throw an exception, it means that your application has successfully bound to the specified port and is ready for client requests.[9]

2.5. Common methods of the ServerSocket class:

1. **public int getLocalPort():** Returns the port that the server socket is listening on. This method is useful if 0 is passed as the port number in a constructor.
2. **public Socket accept() throws IOException:** Waits for an incoming client. This method blocks until either a client connects to server on specified port or the socket times out assuming that the time-out value has been set by setSoTimeout() method. Otherwise this method blocks indefinitely.
3. **public void setSocketTimeot(int timeout):** Sets the time-out value for how long the server socket waits for a client during the accept().
4. **public void bind(SocketAddress host, int backlog):** Binds the socket to specified server and port in the SocketAddress object. It is used in case of no object constructor. When the ServerSocket invokes accept(), the method does not return until a client connects. After a client does connect, the ServerSocket creates a new Socket on an unspecified port and returns a reference to this new Socket. A TCP connection now exists between the client and server, and communication can begin.

2.6 Socket Class Methods:

The *java.net.Socket* class represents the socket that both the client and 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. The Socket class has five constructors that a client uses to connect to a server:

1. **public Socket(String host, int port) throws UnknownHostException, IOException:** This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.

2. **public Socket(InetAddress host, int port) throws IOException:** This method is identical to the previous constructor, except that the host is denoted by an InetAddress object.

3. **public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException:** Connects to the specified host and port, creating a socket on the local host at the specified address and port.

4. **public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException:** This method is identical to the previous constructor, except that the host is denoted by an InetAddress object instead of a String

5. **5 .public Socket():** Creates an unconnected socket. Use the connect() method to connect this socket to a server. When the Socket constructor returns, it does not simply instantiate a Socket object but it actually attempts to connect to the specified server and port.

Some methods in the Socket class are listed here which can be invoked by both the client and server:

1. **public void connect(SocketAddress host, int timeout) throws IOException:** This method connects the socket to the specified host. This method is needed only when you instantiated the Socket using the no-argument constructor.
2. **public InetAddress getInetAddress():** This method returns the address of the other computer that this socket is connected to.
3. **public int getPort():** Returns the port the socket is bound to on the remote machine.
4. **public int getLocalPort():** Returns the port the socket is bound to on the local machine.
5. **public SocketAddress getRemoteSocketAddress():** Returns the address of the remote socket.
6. **public InputStream getInputStream() throws IOException:**Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.
7. **public OutputStream getOutputStream() throws IOException:** Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.
8. **public void close() throws IOException:** Closes the socket, which makes this Socket

object no longer capable of connecting again to any server.

2.7 InetAddress Class Methods: This class represents an Internet Protocol (IP) address.
Some methods which are required while doing socket programming:

1. static InetAddress getByAddress(byte[] addr): Returns an InetAddress object given the raw IP address .
2. static InetAddress getByAddress(String host, byte[] addr): Create an InetAddress based on the provided host name and IP address.
3. static InetAddress getByName(String host): Determines the IP address of a host, given the host's name.
4. String getHostAddress(): Returns the IP address string in textual presentation.
5. String getHostName(): Gets the host name for this IP address.
6. static InetAddress InetAddress getLocalHost(): Returns the local host.
7. String toString(): Converts this IP address to a String.

III. TCP SOCKET PROGRAMMING

In order to do communication over the TCP protocol, a connection must first be established between the pair of sockets. While one of the sockets listens for a connection request (server), the other asks for a connection (client). Once two sockets have been connected, they can be used to transmit data in both (or either one of the) directions.

3.1.1 Server Programming in Java

A server program creates a specific type of socket that is used to listen for client requests (server socket), In the case of a connection request, the program creates a new socket through which it will exchange data with the client using input and output streams. The socket abstraction is very similar to the file concept that is, developers have to open a socket, perform I/O, and close it.

The steps for creating a simple server program are:

1. Open the Server Socket:

```
ServerSocket server = new ServerSocket( PORT );
```

2. Wait for the Client Request:

```
Socket client = server.accept(); Socket Programming 351
```

3. Create I/O streams for communicating to the client

```
DataInputStream is = new DataInputStream(client.getInputStream());
DataOutputStream os = new DataOutputStream(client.getOutputStream());
```

4. Perform communication with client
Receive from client:

```
String line = is.readLine();
```

Send to client:

```
os.writeBytes("Hello\n");
```

5. Close socket:

```
client.close();
```

Program 1:

```
// SimpleServer.java: A simple server program.
import java.net.*;
import java.io.*;
public class SimpleServer {
 public static void main(String args[]) throws IOException {
 // Register service on port 1254
 ServerSocket s = new ServerSocket(1254);
 Socket s1=s.accept(); // Wait and accept a connection
 // Get a communication stream associated with the socket
 OutputStream s1out = s1.getOutputStream();
 DataOutputStream dos = new DataOutputStream (s1out);
 // Send a string!
 dos.writeUTF("Hi there");
 // Close the connection, but not the server socket
 dos.close();
 s1out.close();
 s1.close();
 }
}
```

3.1.2 A simple Client Program in Java

The steps for creating a simple client program are:

1. Create a Socket Object:

```
Socket client = new Socket(server, port_id);
```

2. Create I/O streams for communicating with the server.

```
is = new
DataInputStream(client.getInputStream());
os = new
DataOutputStream(client.getOutputStream());
```

3. Perform I/O or communication with the server:

Receive data from the server:

```
String line = is.readLine();
```

Send data to the server:

```
os.writeBytes("Hello\n");
```

4. Close the socket when done:

```
client.close();
```

Program 2:

```
// SimpleClient.java: A simple client program.
import java.net.*;
import java.io.*;
public class SimpleClient {
 public static void main(String args[]) throws IOException {
 // Open your connection to a server, at port 1254
 Socket s1 = new Socket("localhost",1254);
 // Get an input file handle from the socket and read the input
 InputStream s1In = s1.getInputStream();
 DataInputStream dis = new DataInputStream(s1In);
 String st = new String (dis.readUTF());
 System.out.println(st);
 // When done, just close the connection and exit
 dis.close();
 s1In.close();
 s1.close();
 }
}
```

IV. UDP SOCKET PROGRAMMING

TCP guarantees the delivery of packets and preserves their order on destination. Sometimes these features are not required and since they do not come without performance costs, it would be better to use a lighter transport protocol. This kind of service is accomplished by the UDP protocol which conveys datagram packets.

Datagram packets are used to implement a connectionless packet delivery service supported by the UDP protocol. Each message is transferred from source machine to destination based on information contained within that packet. That means, each packet needs to have destination address and each packet might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.

4.1 Java supports datagram communication through the following classes:

- DatagramPacket
- DatagramSocket

4.1.1 The class DatagramPacket contains several constructors that can be used for creating packet object.

The key methods of DatagramPacket class are:

1. **byte[] getData():** Returns the data buffer.
2. **int getLength():** Returns the length of the data to be sent or the length of the data received.

3. **void setData(byte[] buf):** Sets the data buffer for this packet.
4. **void setLength(int length):** Sets the length for this packet.

4.1.2 The class DatagramSocket supports various methods that can be used for transmitting or receiving data over the network. The two key methods are:

1. **void send(DatagramPacket p):** Sends a datagram packet from this socket.
2. **void receive(DatagramPacket p):** Receives a datagram packet from this socket.

4.2 UDP server program

A simple UDP server program that waits for client's requests and then accepts the message (datagram) and sends back the same message:

Program 3

```
// UDPServer.java: A simple UDP server program.
import java.net.*;
import java.io.*;
public class UDPServer {354 Object-Oriented Programming with Java
 public static void main(String args[]){
 DatagramSocket aSocket = null;
 if (args.length < 1) {
 System.out.println("Usage: java UDPServer <Port Number>");
 System.exit(1);
 }
 try {
 int socket_no = Integer.valueOf(args[0]).intValue();
 aSocket = new DatagramSocket(socket_no);
 byte[] buffer = new byte[1000];
 while(true) {
 DatagramPacket request = new DatagramPacket(buffer,
 buffer.length);
 aSocket.receive(request);
 DatagramPacket reply = new DatagramPacket(request.getData(),
 request.getLength(),request.getAddress(),
 request.getPort());
 aSocket.send(reply);
 }
 }
 catch (SocketException e) {
 System.out.println("Socket: " + e.getMessage());
 }
```

```
catch (IOException e) {
System.out.println("IO: " + e.getMessage());
}
finally {
if (aSocket != null)
aSocket.close();
}
}
}
```

4.3 UDP client program:

A corresponding client program for creating a datagram and then sending it to the above server and then accepting a response:

Program 4

```
// UDPClient.java: A simple UDP client program.
import java.net.*;
import java.io.*;
public class UDPClient {
 public static void main(String args[]){
 // args give message contents and server hostname
 DatagramSocket aSocket = null;
 if (args.length < 3) {
 System.out.println(
 "Usage: java UDPClient <message> <Host name> <Port number>");
 System.exit(1);
}
 try {
 aSocket = new DatagramSocket();
 byte [] m = args[0].getBytes();
 InetAddress aHost = InetAddress.getByName(args[1]);
 int serverPort = Integer.valueOf(args[2]).intValue();
 DatagramPacket request =
 new DatagramPacket(m, args[0].length(), aHost, serverPort);
 aSocket.send(request);
 byte[] buffer = new byte[1000];
 DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
 aSocket.receive(reply);
 System.out.println("Reply: " + new String(reply.getData()));
 }
 catch (SocketException e) {
 System.out.println("Socket: " + e.getMessage());
```

```
}
catch (IOException e) {
System.out.println("IO: " + e.getMessage());
}
finally {
if (aSocket != null)
aSocket.close();
}
}
}
```

V. CONCLUSIONS

In this paper an attempt was made to investigate in to network programming using sockets. Socket programming is best suitable for communication between clients and server. There are several works done on socket programming ever since its advent. This is proved as Windows, Macintosh and UNIX provide interoperability with socket interface. Java is taking over socket programming however the portability can't be matched. Keeping all aspects of the paper we believe socket programming has yet to evolve in its types and ways it will perform in applications. All these will require the thought to have faster and reliable transactions between the server and clients.

RESOURCES

[1] J. F. Kurose and K. W Ross, "Computer Networking- A Top-Down Approach featuring the Internet" , 2nd Edition (Addison Wesley World Student Edition)

[2] Q. Charatan and A. Kans, "Java in two semesters" , 2nd edition McGraw Hills publication,2006

[3] Introduction to Sockets, web.njit.edu/~gblank/cis604/Lectures/604Sockets.ppt

[4] Oracle.comhttp:// docs.oracle.com/cd/E19683-01/816-5042/6mb7bck68/index.html

[5] H. Schildt, The Complete Reference, Seventh Edition, Chap. 27

[6] Tanenbaum, "Computer Networks", Second edition

[7] Author P. Burden," Socket Programming"

[8] Author G. McMillan," Socket Programming HOWTO"

[9] Socket Concepts: http://publib.boulder.ibm.com/infocenter/iseries

[10]https://docs.oracle.com/javase/tutorial/networking/sockets/

[11]http://inst.eecs.berkeley.edu/~ee122/sp06/LectureNotes/Socket%20Programming.pdf

[12]http://www.buyya.com/java/Chapter13.pdf

[13]http://i.ytimg.com/vi/aEDV0WlwXTs/maxresdefault.jpg