Project Report
On

"Multiple Client and Server ChatApplication using Multicast in Java WindowBuilder"

B.Sc. in Information and Communications Engineering
Department of ECE
East West University,Dhaka
Bangladesh

Submitted By

Hasanuzzaman
ID: 2013-2-50-008

Al MesbaSadab
ID: 2013-3-50-014

Under the supervision of
Dr. Mohammad Arifuzzaman
Assistant Professor
Department of Electronics and Communications Engineering
East West University

ACKNOWLEDGEMENT

At first we wish to convey our cordial thanks and gratitude to almighty Allah for everything. We would like to thank our parents and everyone else who has supported us all the way through to complete the project program successfully and also to those who rendered their cooperation in making this report.

We would like to express our gratitude to our supervisor Dr. Mohammad Arifuzzaman, Assistant Professor, Dept. of Electronics and Communications Engineering, East West University for his guidance and support throughout this project work. His constant source of inspiration to us throughout the period of this work.

We are grateful to Chairman, all faculty members of the Department of Electronics and Communications Engineering as well as the concerned officials for their cooperation.

………………………..

Hasanuzzaman

ID: 2013-2-50-008

………………………..

Al Mesba Sadab

ID: 2013-3-50-014

DECLARATION

We hereby declare that we carried out the work reported in this project in the Department of Electronics and Communications Engineering, East West University, under the supervision of Dr. Mohammad Arifuzzaman. We solemnly declare that to the best of our knowledge, no part of this report has been submitted elsewhere for award of any degree. All sources of knowledge used in this report have been duly acknowledged.

………………………..

Hasanuzzaman

ID: 2013-2-50-08

..……………………..

Al Mesba Sadab

ID: 2013-3-50-014

CERTIFICATE

This is to certify that the project entitled **“Multiple Client and Server Chat Application using Multicast in Java WindowBuilder””,** being submitted by Hasanuzzaman and Al Mesba Sadab Department of Electronics and Communications Engineering, East West University, Dhaka in partial fulfillment for the award of the degree of Bachelor of Science in Information and Communications Engineering(**ICE**), is a record of major project carried out by them. They have worked under my supervision and guidance and have fulfilled the requirements which, to my knowledge, have reached the requisite standard for submission of this project.

....

Dr. Mohammad Arifuzzaman

Assistant Professor

Dept. of Electronics and Communications Engineering

APPROVAL

This is to certify that the project entitled **"Multiple Client and Server Chat Application using Multicast in Java WindowBuilder"** submitted to the respected member of the faculty of Engineering for partial fulfillment of requirement for the degree of Bachelor of Information and Communications Engineering(**ICE**) under complete supervision of the undersigned.

Submitted by:

Hasanuzzaman
ID :2013-2-50-008

Al Mesba Sadab
ID :2013-3-50-014

....

Dr. Mohammad Arifuzzaman
Assistant Professor
Dept. of Electronics and Communications Engineering
East West University

....

Dr. M. Mofazzal Hossain
Chairperson & Associated Professor
Dept. of Electronics and Communication Engineering
East West University

Abstract

This report presents a details overview in developing a client-server based chat application using socket programming. The application is developed using Java programing. The primary objective of this report is to present the principles behind socket programming and the libraries available for socket programming applications in Java.

Table of Contents

	Topic	Page Number
Chapter 1	INTRODUCTION	01
Chapter 2	SYSTEM ANALYSIS	02
	2.1 System objective	02
	2.2 System consideration	02
	2.3 Operation concept and scenario	03
Chapter 3	SYSTEM SPECIFICATION	04
	3.1 Hardware requirement	04
	3.2 Software requirements	04
Chapter 4	DEVELOPMENT OF SOFTWARE	05
	4.1 Socket overview	05
	4.2 Project scope	06
	4.2.1 Server	07
	4.2.2 Development of server	07
	4.2.3 Client	08
	4.2.4 Development of client	09
	4.3 The client server block diagram	10
Chapter 5	TCP AND UDP SOCKET PROGRAMING	11
	5.1 A simple client program in Java	12
	5.2 UDP socket programing	12
	5.3 Multicast	15
Chapter 6	SYSTEM DESIGN DETAILS	17
	6.1 GUI Module Name and Description	17
	6.1.1 Design Alternatives	17
	6.1.2 Design Details	17
	6.2 Resolving Names Module Name and Description	17
	6.3 Graphical User Interface	18
	6.4 Testing	18
Chapter 7	MAIN OVERVIEW OF THE WORK	19
	7.1 The novelty of our project	22
	7.2 Future work	22
Chapter 8	Conclusion	23
	Reference	31

CHAPTER 1

INTRODUCTION

Chatting is a method of using technology to bring people and ideas "together" despite of the geographical barriers. The technology has been available for years but the acceptance it was quit recent. Our project is an example of a multiple client chat server. It is made up of 2 applications the client application, which runs on the user's Pc and server application, which runs on any Pc on the network. To start chatting client should get connected to server. We will focus on TCP and UDP socket connections which are a fundamental part of socket programming.

Keywords*:* sockets, client-server, Java network programming-socket functions, Multicasting etc.

CHAPTER 2

SYSTEM ANALYSIS

2.1 System Objectives

Communication over a network is one field where this tool finds wide ranging application. Chat application establishes a connection between 2 or more systems connected over an intra-net or ad-hoc. This tool can be used for large scale communication and conferencing in an organization or campus of vast size, thus increasing the standard of co-operation. In addition, it converts the complex concept of sockets to a user friendly environment. This software can have further potentials, such as file, image transfer, identify of user locations and voice chatting options that can be worked upon later. We are really working hard and soul to develop that application.

2.2 System Considerations

Approach:

The application has been designed using Java programing language in WindowBuilder (Windows Application environment).

Methodology:

The user interacts with the tool using a GUI.

- The GUI operates in two forms, the List form & the chat form.
- The List form contains the names of all the systems connected to a network.
- The chat form makes the actual communication possible in the form of text.

2.3 Operational Concepts and Scenarios

Operation of the application based on the inputs given by the user:

- When the run button is clicked then the chat form is initialized with a connection between the host and the client machine.
- Note: server must be started at first before a client start.
- Contains a rich textbox which send messages from one user to another
- Contains a textbox for messages to be written that is sent across the network.
- Contains a Send button.
- When the sent button is clicked, in the background, the text in the textbox is encoded and sent as a packet over the network to the client machine. Here this message is decoded and is shown in the rich textbox.

CHAPTER 3

SYSTEM SPECIFICATION

3.1 Hardware requirements

In hardware requirement we require all those components which will provide us the platform for the development of the project. The minimum hardware required for the development of this project is as follows—

Ram minimum 128 MB

Hard disk—minimum 5 GB

Processor- Pentium 3

These all are the minimum hardware requirement required for our project. We want to make our project to be used in any. Type of computer therefore we have taken minimum configuration to a large extent.128 MB ram is used so that we can execute our project in a least possible RAM. Hard disk is used because project takes less space to be executed or stored. Therefore, minimum hard disk is used. Others enhancements are according to the needs.

3.2 Software requirements

Software's can be defined as programs which run on our computer .it act as petrol in the vehicle. It provides the relationship between the human and a computer. It is very important to run software to function the computer. Various software's are needed in this project for its development.

Operating system—Windows 7 and Java eclipse

CHAPTER 4

DEVELOPMENT OF SOFTWARE

4.1. Socket Overview

A socket is an object that represents a low level access point to the IP stack. This socket can be opened or closed or one of a set number of intermediate states. A socket can send and receive data down disconnection. Data is generally sent in blocks of few kilobytes at a time for efficiency; each of these block are called *a packet*.

All packets that travel on the internet must use the Internet Protocol. This means that the source IP address, destination address must be included in the packet. Most packets also contain a port number. A port is simply a number between 1 and 65,535 that is used to differentiate higher protocols. Ports are important when it comes to programming your own network applications because no two applications can use the same port.

Packets that contain port numbers come in two flavors: UDP and TCP/IP. UDP has lower latency than TCP/IP, especially on startup. Where data integrity is not of the utmost concerned, UDP can prove easier to use than TCP, but it should never be used where data integrity is more important than performance; however, data sent by UDP can sometimes arrive in the wrong order and be effectively useless to the receiver. TCP/IP is more complex than UDP and has generally longer latencies, but it does guarantee that data does not become corrupted when travelling over the internet. TCP is ideal for file transfer, where a corrupt file is more unacceptable than a slow download; however, it is unsuited to internet radio, where the odd sound out of place is more acceptable than long gaps of silence.

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket. Socket Programming is used for communication between machines using a Transfer Control Protocol (TCP). It can be connectionless or connection-oriented. Server Socket

and Socket classes are used for connection-oriented socket programming. After creating a connection, the server develops a socket object on its end of the connection. The server and client now start communicating by writing to and reading from the socket. The client needs to know two basic information, which are: Port number& IP address of server.

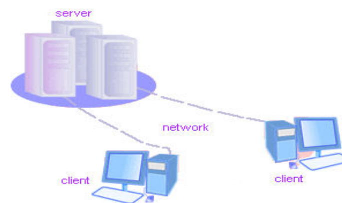

Fig1: Server-Client general model

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

4.2 Project Scope

This project can be mainly divided into two modules:

1. **Server**
2. **Client**

This project is mainly depended on client/server model. The client requests the server and server responses by granting the clients request. The proposed system should provide both of the above features along with the followed ones:

4.2.1 server

A server is a computer program that provides services to other computer programs (and their users) in the same or other computers. The computer that a server program runs in is also frequently referred to as a server. That machine may be a dedicated server or used for other purposes as well. Example Server, Database, Dedicated, Fileserver, Proxy Server, Web Server. The server is always waiting for client's requests. The clients come and go down but the server remains the same.

A server application normally listens to a specific port waiting for connection requests from a client. When a connection request arrives, the client and the server establish a dedicated connection over which they can communicate. During the connection process, the client is assigned a local port number, and binds a *socket* to it. The client talks to the server by writing to the socket and gets information from the server by reading from it. Similarly, the server gets a new local port number (it needs a new port number so that it can continue to listen for connection requests on the original port). The server also binds a socket to its local port and communicates with the client by reading from and writing to it. The client and the server must agree on a protocol that is, they must agree on the language of the information transferred back and forth through the 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.

4.2.2 Development of Server

The steps involved in establishing a socket on the server side are as follows −

- Create a socket with the **socket()** system call.
- Bind the socket to an address using the **bind()** system call. For a server socket on the Internet, an address consists of a port number on the host machine.
- Listen for connections with the **listen()** system call.

- Accept a connection with the **accept()** system call. This call typically blocks the connection until a client connects with the server.
- Send and receive data using the **read()** and **write()** system calls.

4.2.3 Client

On the client site 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.

Fig2:Client reguesting for connection to server

Fig3:Client requesting for connection to server

The model used for this project is the single server – multiple client models. The following specifications must be implemented:

1. The server and client are two separate programs.

2. Multiple clients must be able to connect to a single server.

3. All input and output is via I/O Interface (GUI is Required)

4.2.4 Development of Client

The system calls for establishing a connection are somewhat different for the client and the server, but both involve the basic construct of a socket. Both the processes establish their own sockets. The steps involved in establishing a socket on the client side are as follows:

- Create a socket with the **socket()** system call.
- Connect the socket to the address of the server using the **connect()** system call.
- Send and receive data. There are a number of ways to do this, but the simplest way is to use the **read()** and **write()** system calls.
- 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.

  input = new **DataInputStream(client.getInputStream());**

  output = new **DataOutputStream(client.getOutputStream());**

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

  Receive data from the server: **String line = input.readLine();**

  Send data to the server: **output.writeBytes("Hello Hasanuzzaman\n");**

  4. Close the socket when done:
- **client.close();**

4.3 The Client and Server Block Diagram

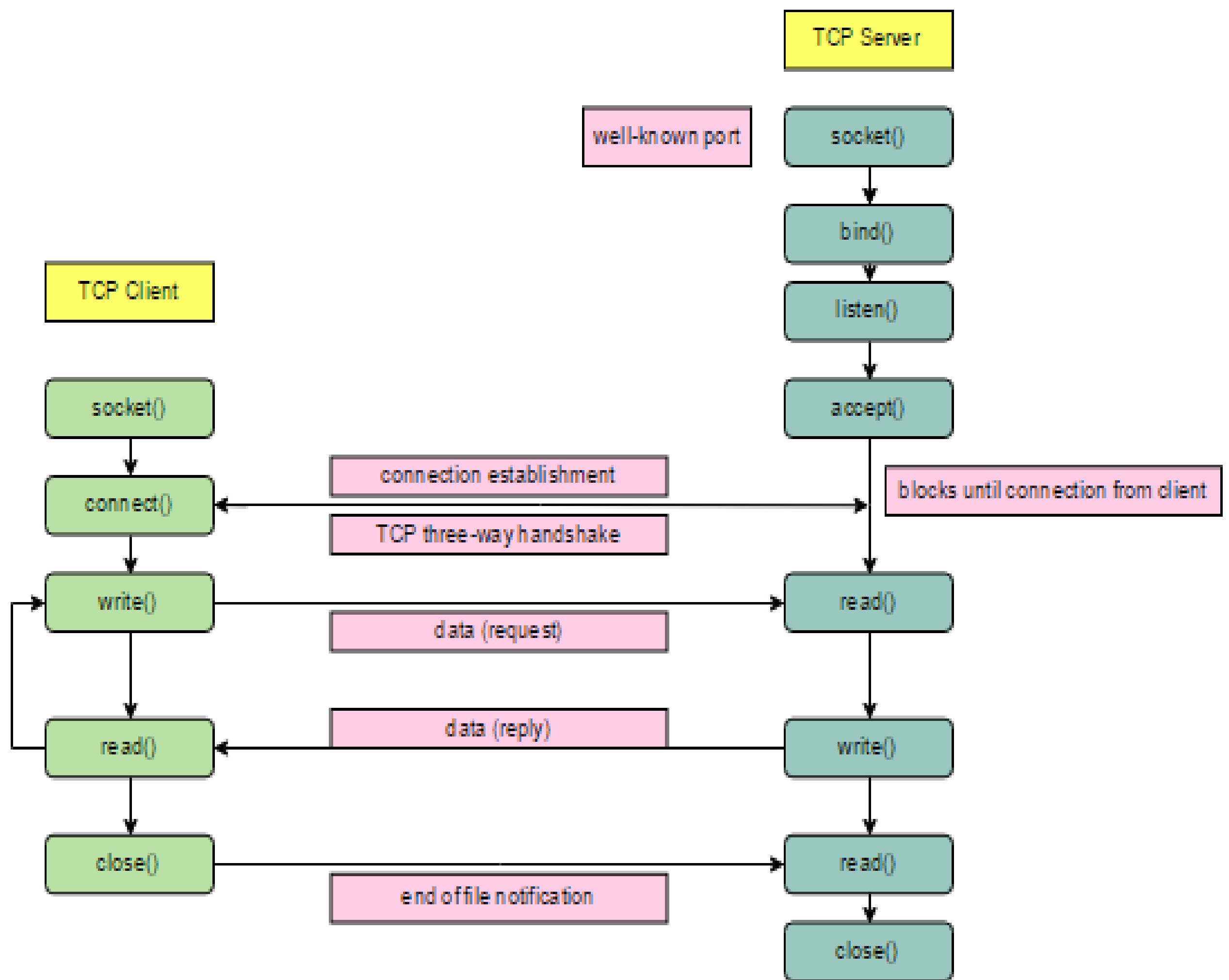

Fig4: Block diagram of client-server

CHAPTER 5

TCP AND UDP SOCKET PROGRAMMING

The two key classes from the java.net package used in creation of server and client programs are:

- **ServerSocket**
- **Socket**

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.

1. Open the Server Socket:

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

2. Wait for the Client Request:

Socket client = **server.accept();**

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

DataInputStream input = new **DataInputStream(client.getInputStream());**

DataOutputStream output = new **DataOutputStream(client.getOutputStream());**

4.Perform communication with client

Receive from client: **String** line = **input.readLine();**

Send to client: **output.writeBytes("Hello\n");**

5. Close socket:

client.close();

An example program illustrating creation of a server socket, waiting for client request, and then Responding to a client that requested for connection by greeting it is given at **appendix [A]**

5.1 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.

input = new **DataInputStream(client.getInputStream());**

output = 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!Hasanuzzaman\n");**

4. Close the socket when done:

client.close();

An example program illustrating establishment of connection to a server and then reading a message sent by the server and displaying it on the console is given at Appendix[A]

5.2 UDP SOCKET PROGRAMMING

The User Datagram Protocol is an unreliable, connectionless oriented protocol that uses an IP address for the destination host and a port number to identify the destination application. The UDP port number is distinct from any physical port on a computer such as a COM port or an I/O port address. The UDP port is a 16-bit address that exists only for the purpose of passing certain types of datagram information to the correct location above the transport layer of the protocol stack.

A UDP datagram header consists of four (4) fields of two bytes each:

1. **source port number**
2. **destination port number**
3. **datagram size**
4. **checksum**

In order to use a UDP socket for network programming one has to follow the following steps as shown in figure given below:

End point is a combination of IP address and port number. Endpoint objects allow us to easily establish and communicate over TCP/IP network connections between client and server processes, possibly residing on different hosts. Once a network connection is established between a client and a server, the two can "talk" to each other by reading from and writing to the connection.

The previous two example programs used the TCP sockets. As already said, TCP guarantees the delivery of packets and preserves their order on destination. 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 isn't guaranteed. The format of datagram packet is:

Figure 5: Format of datagram packet

Java supports datagram communication through the following classes:

- **DatagramPacket**
- **DatagramSocket**

The class **DatagramPacket** contains several constructors that can be used for creating packet object. One of them is:

DatagramPacket(byte[] buf, int length, InetAddress address, int port);

This constructor is used for creating a datagram packet for sending packets of length. Length to the specified port number on the specified host. The message to be transmitted is indicated in the first argument. The key methods of **DatagramPacket** class are: Returns the length of the data to be sent or the length of the data received.

voidsetData(byte[] buf)

- Sets the data buffer for this packet.

voidsetLength(int length)

- Sets the length for this packet.

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

- Sends a datagram packet from this socket.

  void send(DatagramPacket p)

- Receives a datagram packet from this socket.

  void receive(DatagramPacket p)

A simple UDP server program that waits for client's requests and then accepts the message (**datagram**) and sends back the same message is given below. Of course, an extended server program can manipulate client's messages/request and send a new message as a response.

A corresponding client program for creating a datagram and then sending it to the above server and then accepting a response is listed below. An example of TCP and UDP server client program is given at **Appendix[A].**

5.3 Multicast

Multicast is the term used to describe communication where a piece of information is sent from one or more points to a set of other points. Multicasting is the networking technique of delivering the same packet simultaneously to a group of clients. In this case there is may be one or more senders, and the information is distributed to a set of receivers. Multicasting is broader than unicast, point-to-point communication but narrower and more targeted than broadcast communication. Multicasting sends data from one host to many different hosts, but not to everyone; the data only goes to clients that have expressed an interest by joining a particular multicast group multicast socket that sends a copy of the data to a location (or a group of locations) close to the parties that have declared an interest in the data. Several identical copies of the data traverse the Internet; but, by carefully choosing the points at which the streams are duplicated, the load on the network is minimized. One example of an application which may use multicast is a video server sending out networked TV channels. Simultaneous delivery of high quality video to each of a large number of delivery platforms will exhaust the capability of even a high bandwidth network with a powerful video clip server. This poses a major salability issue for applications which required sustained high bandwidth. One way to significantly ease scaling to larger groups of clients is to employ multicasting networking IP multicast provides dynamic many-to-many connectivity between a set of senders (at least 1) and a group of receivers. The format of IP multicast packets is identical to that of unicast packets and is distinguished only by the use of a special class of destination address (class D IPv4 address) which denotes a specific multicast group. Since TCP supports only the unicast mode, multicast applications must use the UDP transport protocol. Unlike broadcast transmission (which is used on some local area networks), multicast clients receive a stream of packets only if they have previously elect to do so (by joining the specific multicast group address). Membership of a group is dynamic and controlled by the receivers (in turn informed by the local client applications). The routers in a multicast network learn which sub-networks have active clients for each multicast group and attempt to minimize the transmission of packets across parts of the network for which there are no active clients. An example program of multicast is given at **Appendix[A]**

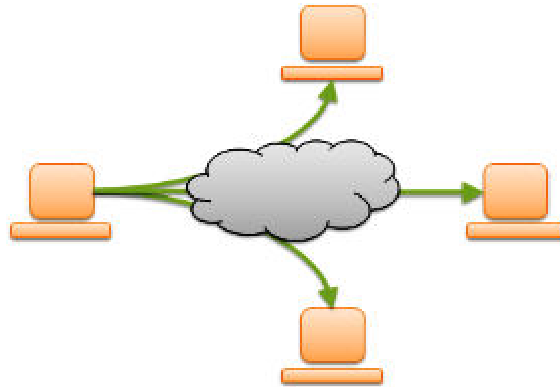

Fig6: A general view of multicasting

Multicast is a special feature of UDP protocol that enable programmer to send message to a group of receivers on a specific multicast IP address and port. Multicast has advantage in this scenario. Let us say we want to send "Hello" message to 100 computers on my network. Perhaps, my first solution is to send the "Hello" message to each of them via UDP or TCP. Consume a lot of processing power on sender as it needs to send to every receiver, bandwidth flooding and the arrival time is not the same for every receiver.

Seeing this problem, we propose second solution by employing Multicast. Multicast runs over UDP protocol [4]

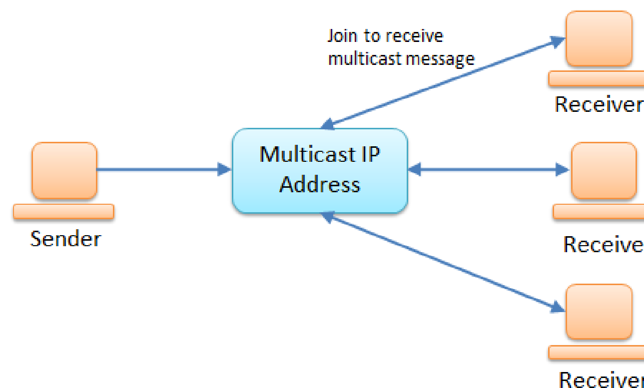

Fig7: Multicasting using UDP protocol adopted from [4]

If the client wishes to receive multicast message, it must join the group with multicast IP (224.2.2.3) and port 8888.

Chapter 6

SYSTEM DESIGN DETAILS

6.1 GUI Module Name and Description

This module deals with the application's interface with the end user.

6.1.1 Design Alternatives

- Java programming approach is used as the tool has been developed in window Builder environment

6.1.2 Design Details

At a minimum, the following should be described -

a) Processing within module

We develop the user interface for the application through which the user interacts with the tool. It consists of a main window and boxes which are displayed as per the menu selections made by the user.

b) Error checking

Errors occurring because of connection problems. Errors occurring due to incorrect input by the user.

6.2 Resolving Names Module Name and Description

In this module the application resolves the names of the systems connected to a network. These names are displayed in the form of a list. In this module the user communicates with the desired user in the form of text. A connection is formed between the server and client with the help of sockets which itself uses ports for packet data transfer. We show a windows form application

that makes communication graphic oriented and user friendly. Three GUI interfaces are captured under this:

- **Send Button**

  When this button is pressed, the message in the textbox is displayed in the TextArea.

- **Instant Buttons:** Normally we need to type text to send to the end user, but this features could help to send text immediately without typing.

6.3 Graphical User Interface

The user interface that the software provides to the user is interactive. It provides two different forms, one for list of systems and the other for the actual text chatting.

Component	Description
JLabel	An area where uneditable text or icons can be displayed.
JTextField	An area in which the user inputs data from the keyboard. The area can also display information.
JButton	An area that triggers an event when clicked.
JCheckBox	A GUI component that is either selected or not selected.
JComboBox	A drop-down list of items from which the user can make a selection by clicking an item in the list or possibly by typing into the box.
JList	An area where a list of items is displayed from which the user can make a selection by clicking once on any element in the list. Double-clicking an element in the list generates an action event. Multiple elements can be selected.
JPanel	A container in which components can be placed.

Fig8: Some basic GUI components.

6.4 Testing

In this menu items were tested to ensure no functions has been missed out. This is done for the smooth working of the project. This is done after the completion of system; all the queries were carried out again to ensure that no errors have been introduced.

CHAPTER 7

MAIN OVERVIEW OF OUR WORK

In general, all of the machines on the Internet can be categorized as two types: servers and clients

Single server should be able to chat simultaneously with multiple clients. In order to do this, we will have to implement the server program using threads. Once a client program contacts a server, the server process spawns a thread that will handle the client. The communication between a client and its server thread will be like a single client-single server chatting Application. A multicast chatting tool that will be used to communicate among a group of processes. Each process should be able to send and receive any number of messages. The chat tool should have the following functionalities:

Get the message from the user and send it to all the other processes belonging to the group. A process can receive a copy of the message. Read the messages sent by any other process and display the message to the user. Sever send message to the multiple client (at least 1) as well as client can send message individually to the desired client. For doing this at first we need to start the server and then client again if we press run then another client will be appeared which 2nd client is. Thus way we can start multiple client window. Server can send the same message to all of running client windows, similarly client can individually can sent message to the server.in this work we have been developed some feature using java window Builder that typically doesn't exist. The features are: -

- Developed an ID system for multiple clients. Each client will be assigned individual ID. When client send any message to server then a window will ask to type his/her name. This name will be printed at the server textArea with its message.

- Server and client will print the current time and date respect.
- When we send message by clicking **Send** button, beep tune to be hear that indicate message has been send.

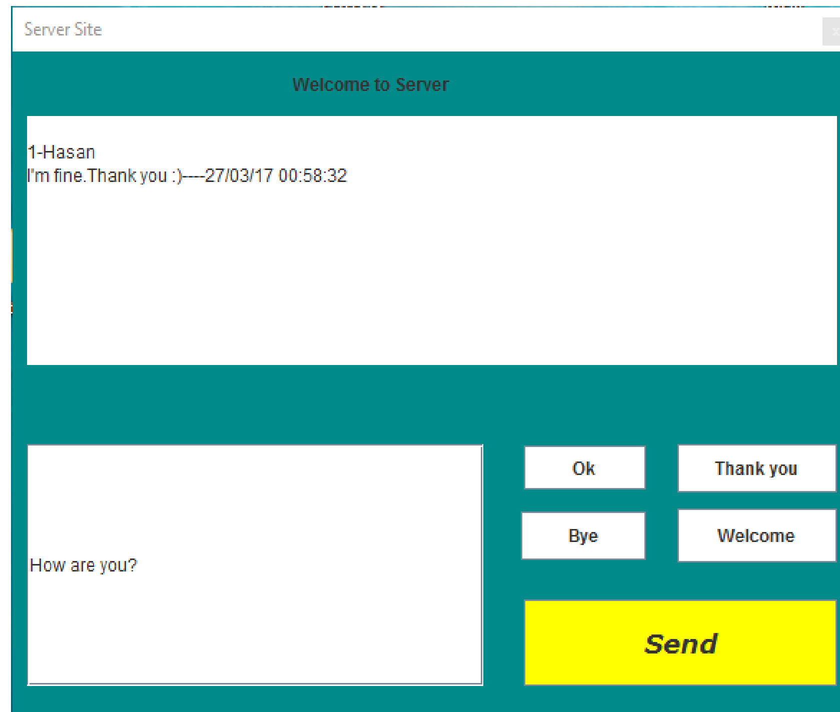

Figure 8:Interface of Server window

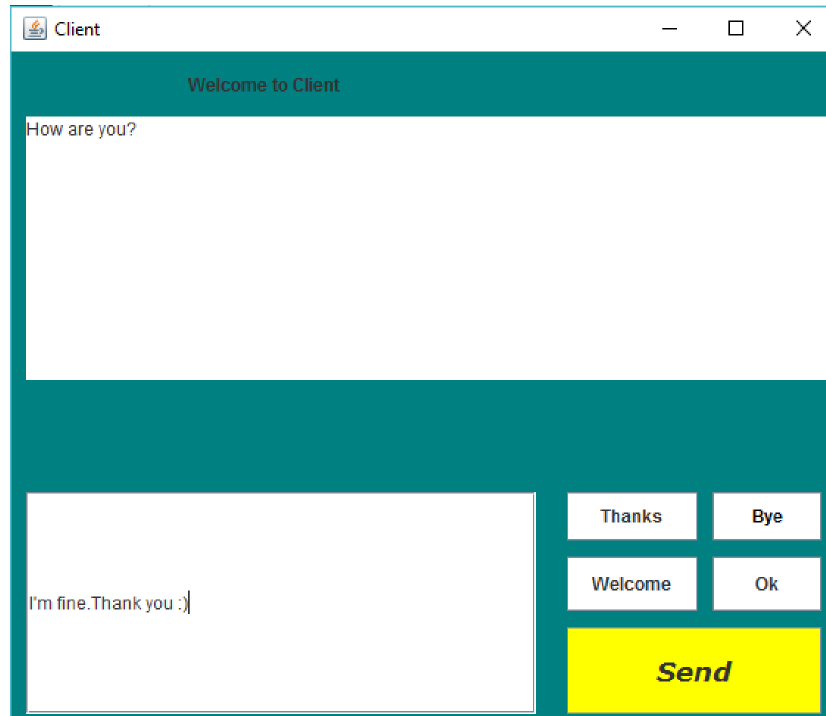

Figure10: Interface of Client window

- Developed some instant button that would help to send message more quickly without typing that. The button here is OK, thank you, Bye, welcome etc. both of the client and server side.it would help the user to send instant message.
- Developed message filtering system for some restricted adult words using array and Boolean. If anyone can try to send any kind of restricted work, then instantly a warning window will appear with a message of Sorry! Some restricted keyword got matched, you can't send this message.

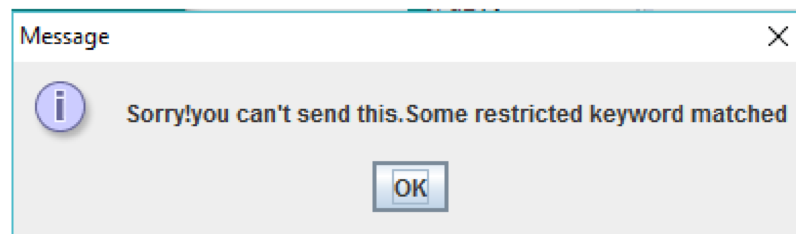

Fig8: The warning message for restricted keywords

7.1 The Novelty our proposed work

The following features that does not exist yet that makes our apps different than any other Services: -

- Instant buttons
- The individual ID will prove identity whether he/she uses fake name or not
- Chat filtering
- User friendly and customization

7.2 Future work

There is always a room for improvements in any software package, however good and efficient it may be done. But the most important thing should be flexible to accept further modification. Right now we are just dealing with text communication. In future this software may be extended to include features such as:

• **Files transfer:** this will enable the user to send files of different formats to others via the chat application.

• **Voice chat:** this will enhance the application to a higher level where communication will be possible via voice calling as in telephone.

• **Video chat:** this will further enhance the feature of calling into video communication.

CHAPTER 8

Conclusion

We Developed network applications in Java by using sockets, threads, and Web services. These software is portable, efficient, and easily maintainable for large number of clients. Our developed web-based chatting software is unique in its features and more importantly easily customizable. The java.net package provides a powerful and flexible set of classes for implementing network applications. Typically, programs running on client machines make requests to programs on a server Machine. These involve networking services provided by the transport layer. The most widely used transport protocols on the Internet are TCP (Transmission control Protocol) and UDP (User Datagram Protocol). TCP is a connection-oriented protocol providing a reliable flow of data between two computers. On the other hand, UDP is a simpler message-based connectionless protocol which sends packets of data known as datagrams from one computer to another with no guarantees of arrival.

Appendix[A]

```
// 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();
}
}
```

```
// 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();
}
}

// UDPServer.java: A simple UDP server program.
import java.net.*;
import java.io.*;
public class UDPServer {

public static void main(String args[]){
DatagramSocketaSocket = null;
if (args.length< 1) {
System.out.println("Usage: java UDPServer<Port Number>");
System.exit(1);
}
try {
intsocket_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();
}
}
}
```

```
// 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
DatagramSocketaSocket = 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();
InetAddressaHost = InetAddress.getByName(args[1]);
intserverPort = Integer.valueOf(args[2]).intValue();
DatagramPacket request =
newDatagramPacket(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();
}
}
}
```

```
importjava.io.IOException;
importjava.net.DatagramPacket;
importjava.net.DatagramSocket;
importjava.net.InetAddress;
importjava.net.UnknownHostException;

public class MulticastSocketServer {

final static String INET_ADDR = "224.0.0.3";
final static int PORT = 8888;

public static void main(String[] args) throws
UnknownHostException, InterruptedException {
        // Get the address that we are going to connect to.
InetAddressaddr = InetAddress.getByName(INET_ADDR);

        // Open a new DatagramSocket, which will be used to send
the data.
try (DatagramSocketserverSocket = new DatagramSocket()) {
for (inti = 0; i< 5; i++) {
                String msg = "Sent message no " + i;
```

```
                // Create a packet that will contain the data
                // (in the form of bytes) and send it.
DatagramPacketmsgPacket = new DatagramPacket(msg.getBytes(),
msg.getBytes().length, addr, PORT);
serverSocket.send(msgPacket);

System.out.println("Server sent packet with msg: " + msg);
Thread.sleep(500);
            }
        } catch (IOException ex) {
ex.printStackTrace();
        }
    }
}


importjava.io.IOException;
importjava.net.DatagramPacket;
importjava.net.InetAddress;
importjava.net.MulticastSocket;
importjava.net.UnknownHostException;

public class MulticastSocketClient {

final static String INET_ADDR = "224.0.0.3";
final static int PORT = 8888;

public static void main(String[] args) throws
UnknownHostException {
        // Get the address that we are going to connect to.
InetAddress address = InetAddress.getByName(INET_ADDR);

        // Create a buffer of bytes, which will be used to store
        // the incoming bytes containing the information from the
server.
```

```
        // Since the message is small here, 256 bytes should be
enough.
byte[] buf = new byte[256];
 // Create a new Multicast socket (that will allow other
sockets/programs
        // to join it as well.
try (MulticastSocketclientSocket = new MulticastSocket(PORT)){
            //Joint the Multicast group.
clientSocket.joinGroup(address);

while (true) {
                // Receive the information and print it.
DatagramPacketmsgPacket = new DatagramPacket(buf, buf.length);
clientSocket.receive(msgPacket);

                String msg = new String(buf, 0, buf.length);
System.out.println("Socket 1 received msg: " + msg);
            }
        } catch (IOException ex) {
ex.printStackTrace();
        }
    }
}
```

Reference:

[1] https://github.com/hasanuzzaman2013/java_windowbuilder

[2] https://www.tutorialspoint.com/java/java_networking.htm

[2] http://cs.lmu.edu/~ray/notes/javanetexamples/

[3] http://www.javaworld.com/article/2077322/core-java/core-java-sockets-programming-in-java-a-tutorial.html

[3]http://www.nakov.com/inetjava/lectures/part-1-sockets/InetJava-1.5-UDP-and-Multicast-Sockets.html

[5] https://www.tutorialspoint.com/unix_sockets/client_server_model.html

[4] https://examples.javacodegeeks.com/core-java/net/multicastsocket-net/java-net-multicastsocket-example/

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

[6] http://java-source.net/open-source/chat-servers

[7] https://fivedots.coe.psu.ac.th/~ad/jg/ch19/ch19.pdf

[8] https://gist.github.com/arbo77/3318971

[9] https://venturebeat.com/2016/06/15/sapho-apps-in-chat-are-the-future-of-work/

[10] https://www.lifewire.com/socket-programming-for-computer-networking-4056385

[11] http://www.javatpoint.com/socket-programming

[12] www.wikipedia.com